ÜBER DIE AUSTAUSCHBARKEIT VON UNIVERSALITÄT UND EFFIZIENZ BEI INSTANZENNETZSIMULATOREN, INSBESONDERE FÜR DIGITALE HARDWARE

von

Flavio Rech Wagner

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Verleihung des akademischen Grades
DOKTOR - INGENIEUR (Dr.-Ing.)
genehmigte DISSERTATION

Dekan: Prof. Dr. rer. nat. E. von Puttkamer
1. Berichterstatter: Prof. Dr.-Ing. S. Wendt
2. Berichterstatter: Prof. Dr. rer. nat. G. Zimmermann

Datum der wissenchaftlichen Aussprache: 16.12.1983

D 386

Herrn Prof. Dr.-Ing. S. Wendt bin ich für die Anregung zu dieser Arbeit, hilfreiche Diskussionen und stetige Unterstützung während meines Aufenthaltes in Kaiserslautern zu großem Dank verpflichtet. Durch seine systemtechnischen Modelle hat er die Grundlagen für meine Untersuchungen geschaffen.

Herrn Prof. Dr. rer. nat. G. Zimmermann danke ich für die Übernahme des Korreferates und wertvolle Hinweise.

Herrn Prof. Dr.-Ing. R. Hartenstein danke ich für die Anregung zum Thema Hardware-Simulation.

Dem CNPq (Conselho Nacional de Desenvolvimento Cientifico e Tecnologico), Forschungsbeirat der brasilianischen Regierung, und der Universidade Federal do Rio Grande do Sul, Brasilien, bin ich für die finanzielle Unterstützung zu Dank verpflichtet.

Diese Arbeit widme ich meiner Familie und der Familie Wendt. Beide haben dafür gesorgt, daß ich zu einem erfolgreichen Abschluß kommen konnte.

Zusammenfassung

Ziel der vorliegenden Arbeit ist es, den Kompromiß zwischen Universalität und Effizienz bei Instanzennetzsimulatoren zu untersuchen, insbesondere für die Simulation von digitaler Hardware.

Ein Instanzennetzsimulator wird definiert mit Hinsicht auf maximale Universalität. Dieser Simulator muß für die Simulation beliebiger Instanzennetze anwendbar sein.

Hardware auf der Gatter- und auf der Register-Transferebene wird modelliert für Simulation mit dem definierten Instanzennetzsimulator. Ebenso werden spezifische Hardware-Simulatoren definiert, wobei repräsentative Modelle der Gatter- und der Register-Transferebene ausgewählt werden.

Da diese Hardware-Simulatoren nur für bestimmte Systemklassen geeignet sind, die Unterklassen von Instanzennetzen darstellen, bringen sie einen gewissen Effizienzgewinn gegenüber dem allgemeinen Instanzennetzsimulator. Die Messung dieses Gewinns und seine Zurückführung auf bestimmte Eigenschaften der Instanzennetze und der digitalen Systeme sind konkrete Ziele dieser Arbeit.

Um diese Messung zu ermöglichen, werden digitale Systeme durch Parametersätze dargestellt. Diese Parameter erlauben uns, exakte Ausdrücke für den Simulationszeitverbrauch aller definierten Simulatoren abzuleiten. Durch Variierung der Parameterwerte wird das ganze Spektrum der digitalen Systeme erfaßt.

Inhaltsverzeichnis

1. Einführung

Instanzennetze /WEN83/ dienen der Modellierung allgemeiner diskreter Systeme. Aus den universellen Eigenschaften allgemeiner Instanzennetze läßt sich ein zweckmäßiger Simulationsalgorithmus erstellen. Wenn wir vor der Aufgabe stehen, eine Klasse spezifischer diskreter Systeme simulieren zu müssen, liegen zwei Alternativen vor. Wir wählen zwischen der größeren Effizienz eines spezifischen Simulators, der nur für die Simulation dieser Systemklasse geeignet ist, aber den wir noch in einer gewöhnlichen Programmiersprache schreiben müssen, und der Universalität eines schon vorhandenen allgemeinen Simulators, der einen minimalen Satz universeller Eigenschaften voraussetzt, mit denen alle diskreten Systeme nachgebildet werden können. Den Effizienzgewinn des spezifischen Simulators gegenüber dem allgemeinen Simulator können wir als Folge einer Implementierung erklären, die wir durch bestimmte Einschränkungen in diesen universellen Eigenschaften erreichen.

Ziel dieser Arbeit ist es, diese Restriktionen zu qualifizieren, durch die wir einen für eine spezifische Systemklasse geeigneten Simulator als Sonderfall eines Instanzennetzsimulators erklären können, und den aus diesen Restriktionen resultierenden Effizienzgewinn des spezifischen Simulators gegenüber dem Instanzennetzsimulator zu messen.

Als Effizienzmaßstab verwenden wir nur den Zeitverbrauch der eigentlichen Simulationsphase, nunmehr als Simulationszeit bezeichnet. Der Speicherverbrauch der Simulationsalgorithmen wird nicht berücksichtigt, insbesondere weil er aufgrund der heutigen Technologie nicht mehr von größerer Bedeutung ist. Wir berücksichtigen auch keine Vorteile des universellen Instanzennetzsimulators, wie die Verfügbarkeit des Simulators und die Modellierung mit einem allgemeingültigen Konzept. Sie wären bedeutsam, wenn die Simulation Teil eines umfassenderen Entwurfsverfahrens wäre. Wir ziehen auch den Zeitverbrauch der Modellierungsphase und der Implementierung des Simulators nicht in Betracht.

Damit diese Untersuchung fair ist, müssen wir Systemklassen betrachten, die mit dem Instanzennetzmodell gut verträglich sind. Jeder Simulator begünstigt eine gewisse Klasse von Systemen, weil er eine Systemmodellierung erfordert, die für diese Klasse direkt und durchsichtig ist. Andere Systemklassen lassen sich dagegen für diesen Simulator schwieriger modellieren, was einen Effizienzverlust mit sich bringt. Instanzennetzsimulatoren sind hauptsächlich für die Simulation derjenigen Systeme geeignet, deren Komponenten ausschließlich über Operanden kommunizieren /WEN83/ (siehe Abschnitt 2.3). Auf der Register-Transfer- und der Gatterebene modellierte digitale Systeme sind typische Systeme dieser Art und wurden deswegen als Sonderklassen für diese Untersuchung ausgewählt. Innerhalb dieser Ebenen gibt es außerdem unterschiedliche Zeitmodelle, die Systeme mit

unterschiedlichen dynamischen Eigenschaften voraussetzen. Für diese Arbeit haben wir zwei Modelle auf der Register-Transferebene und zwei Modelle auf der Gatterebene ausgesucht. Digitale Systeme auf der Schaltungsebene gehören nicht mehr zur Klasse der diskreten Systeme und lassen sich nicht als Instanzennetze modellieren /WEN82/, während höhere Modellierungsebenen digitaler Systeme, wie Prozessor- und Software-Ebene, besser als Netze kommunizierender programmierter Instanzen dargestellt werden /WEN83/ (siehe Abschnitt 2.3).

Hardware-Simulatoren können außerdem in zwei Klassen aufgeteilt werden: Funktionelle Simulatoren, in denen wir nur die gesamte Systemfunktion berücksichtigen, und strukturelle Simulatoren, in denen wir die exakte Systemstruktur berücksichtigen. Um einen fairen Vergleich zu ermöglichen, sind alle hier behandelten Hardware-Simulatoren vom zweiten Typ, weil der Begriff "Instanzennetz" auch strukturell ist, obwohl jeder Instanz (Baustein) beliebige Funktion zugeordnet werden kann.

Für die Untersuchung ist es wichtig, daß die Simulationsalgorithmen vergleichbar sind. Vorhandene Simulatoren besitzen unterschiedliche zusätzliche Eigenschaften, wie z.B. eine gewisse Benutzerfreundlichkeit, so daß eine Analyse, die auf mit solchen Simulatoren erzielten Ergebnissen basiert, unbrauchbar ist. Vielmehr müssen wir Simulationsalgorithmen entwickeln, die auf die wesentlichen Eigenschaften der Instanzennetze und der zu untersuchenden Sonderklassen beschränkt sind. Für den Vergleich sind dann Voraussetzungen: a) Optimale spezifische Simulationsalgorithmen für die Sonderklassen; b) Ein optimaler Simulationsalgorithmus für Instanzennetze; und c) Die Modellierung der ausgewählten Sonderklassen als Instanzennetze, so daß diese Instanzennetzsimulation genau dieselben Ergebnisse liefert, die mit den spezifischen Simulatoren für die Sonderklassen erzielt werden.

Aus Sicht des Endbenutzers verdienen deswegen sowohl der spezifische Simulator für die Gatterebene als auch der aus der Instanzennetzmodellierung von Gatterebenen-Systemen resultierende Simulator die Bezeichnung "Gattersimulator". Um den Unterschied zwischen beiden zu verdeutlichen, werden wir jedoch nur den spezifischen Simulator als Gattersimulator bezeichnen. Diese Betrachtung gilt auch für den spezifischen und den aus der Instanzennetzmodellierung resultierenden Register-Transfer-Simulator.

Wir stellen uns zwei Hauptaufgaben vor. Erstens möchten wir aus dem möglichen Spektrum konkreter Systeme jeder Sonderklasse zwei Grenzfälle identifizieren, und zwar die Systeme, mit denen der spezifische Simulator den höchsten bzw. kleinsten Simulationszeitgewinn gegenüber dem Instanzennetzsimulator aufweist. Zweitens werden wir diesen Gewinn in Faktoren aufteilen und jeden Faktor als Folge einer im spezifischen Simulator gemachten Einschränkung in einer universellen Eigenschaft von Instanzennetzen erklären.

In Kapitel 2 geben wir die Definition von Instanzennetzen und leiten den entsprechenden optimalen Simulationsalgorithmus ab, wofür Modellierungs- und Implementierungsentscheidungen zu treffen sind. Obwohl ein Instanzennetzsimulator (BORIS /WEN79b/) vorhanden ist, ist die Implementierung dieses Simulators stark von seiner iteraktiven Anwendung beeinflußt worden und könnte deswegen in dieser Arbeit nicht berücksichtigt werden. Kapitel 3 beschreibt die eigentlich verwendete Vergleichsmethode, d.h., wie wir den Simulationszeitverbrauch der beiden zu vergleichenden Simulatoren errechnen und unter welchen Voraussetzungen diese Messungen erfolgen können. In Kapitel 4 stellen wir das Gatterebenen-Modell und die zwei zugehörigen Zeitmodelle vor und leiten die für diese Modelle optimalen spezifischen Simulationsalgorithmen ab. Wir bilden Gatterebenen-Systeme gemäß beiden Zeitmodellen durch Instanzennetze nach und führen für jedes Zeitmodell einen Vergleich zwischen der spezifischen und der Instanzennetzmodellierung durch. Als Ergebnis dieses Vergleichs bekommen wir die zwei vorher genannten Grenzsysteme. Dieselbe Vorgehensweise verwenden wir für Register-Transfer-Systeme in Kapitel 5. In Kapitel 6 interpretieren wir die in Kapiteln 4 und 5 erzielten Simulationszeitgewinne der spezifischen Simulatoren gegenüber den entsprechenden Instanzennetzmodellierungen. Diese Interpretation bezieht sich auf die Einschränkungen in den Instanzennetzeigenschaften, die den Effizienzgewinn jedes spezifischen Simulators erklären. Kapitel 7 schließlich faßt die wichtigsten Schlußfolgerungen zusammen.

2. Instanzennetzsimulation

2.1 Digitale Prozesse und ihre Beschreibung /WEN83/

Unter einem **System** verstehen wir ein Gebilde aus wechselwirkenden Komponenten, die einen beobachtbaren zeitvarianten Zustand erzeugen. Die Beobachtungssequenz bestimmter Systemvariablen zu bestimmten Zeitpunkten nennen wir einen **Prozess**. Wenn sowohl die Systemvariablen als auch die Zeitpunkte nur diskrete Werte annehmen können, haben wir einen **Digitalen Prozess**. Diese Definition setzt voraus, daß alle Systemzustandsänderungen augenblicklich erfolgen und zwischen zwei nacheinander folgenden Zustandsänderungen alle Systemvariablen unverändert bleiben. Die Wertänderung einer Systemvariable ist ein **Ereignis**, so daß wir einen digitalen Prozeß durch eine Ereignissequenz darstellen können, wobei einige davon gleichzeitig auftreten können. Diese Gleichzeitigkeit bezieht sich selbstverständlich auf den modellierten Zeitmaßstab. Mehrere Prozesse, d.h. Ereignissequenzen, können aus einer einzigen Prozeßstruktur entstehen. Wir interessieren uns deswegen nicht für eine bestimmte Ereignisprotokollierung, sondern für diese Prozeßstruktur, die alle möglichen resultierenden Prozesse erklärt. Diese Kenntnis erreichen wir dadurch, daß wir alle Beziehungen zwischen Ereignissen verstehen. Entweder sind zwei Ereignisse unabhängig voneinander, so daß sie in beliebiger Zeitreihenfolge auftreten können, oder gibt es eine kausale Abhängigkeit zwischen ihnen. Petri-Netze werden normalerweise als Beschreibungsmittel für Kausalstrukturen benutzt.

2.2 Definition von Instanzennetzen /WEN83/

Instanzennetze sind bipartite gerichtete Netze. Sie bestehen aus Bausteinen zweier Klassen, **Instanzen** und **Speicher**. Instanzen sind die aktiven und Speicher die passiven Komponenten. Der Ausgang einer Instanz darf nur mit einem Speichereingang verbunden sein, und das ist eine Pulsverbindung. Die Instanz liefert dem Speicher einen Auftrag "Setze den Speicherinhalt auf einen neuen Wert". Der Wertebereich eines Instanzenausgangs ist die Menge solcher Wertaufträge samt dem Wert "leer", der zwischen zwei Aufträgen angezeigt wird. Ein Auftrag ist das Ergebnis einer Instanzenaktion, die eine Funktion der Instanzeneingänge und des internen Instanzenzustandes ist. Speicher haben keine Verarbeitungsfähigkeit. Sie führen den Wertauftrag unmittelbar durch und zeigen den Inhalt immer an seinen Ausgängen, so daß eine Pegelverbindung mit den Instanzeneingängen vorhanden ist. Weil Speicher nur Modellgebilde für Systemvariablen sind /WEN83/, bezeichnen wir sie lieber als **Variablen**.

Abb. 2.1 zeigt, wie wir die Zugriffsarten der Instanzen auf die Variablen graphisch darstellen. Instanzen werden als Rechtecke und Variablen als Kreise repräsentiert. Instanzen

können lesenden, setzenden oder modifizierenden Zugriff auf die Speicher haben. **Zuständigkeitsbereich** /WEN82/ einer Instanz ist die Menge solcher Variablen, für deren Werte die Instanz zu einem bestimmten Zeitpunkt zuständig ist (notwendigerweise eine Untermenge aller Variablen, auf die die Instanz setzenden oder modifizierenden Zugriff hat). Instanzennetze können zeitvariante Zuständigkeitsbereiche haben. **Umgebung** nennen wir die Menge aller Variablen, zu denen die Instanz Zugang hat.

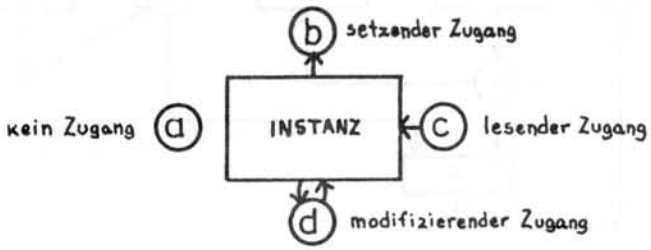


Abb. 2.1 - Zugangsarten der Instanzen zu den Variablen

Instanzennetze können Konflikte zweier Typen haben: **Zuständigkeits-** und **Erkennungskonflikte** /WEN82/. Beide lassen sich am besten durch Konflikte in dem Petri-Netz erklären, welches die Beziehungen zwischen den Instanzen zeigt.

Zuständigkeitskonflikte sind vorhanden, wenn die Zuständigkeitsbereiche zweier oder mehrerer Instanzen sich überlappen. Im Petri-Netz der Abb. 2.2 (aus /WEN82/) gibt es einen Zuständigkeitskonflikt zwischen den Instanzen C und F. Das Schalten einer Transition, die der Aktivierung einer dieser Instanzen entspricht, zerstört die Schaltbereitschaft der Transition für die andere Instanz. Dieser Konflikt entsteht aus der Systemspezifikation selbst und kann deswegen nur durch eine neue Definition gelöst werden. Für diese Arbeit spielen die Zuständigkeitskonflikte keine bedeutsame Rolle.

Erkennungskonflikte existieren, wenn für zwei Instanzen D und F (siehe Abb. 2.2) die Bedingungen für ihre Aktivierung erfüllt sind und die Aktivierung einer Instanz (F in diesem Fall) die Bedingungserfüllung für die andere zerstört, aber nicht umgekehrt. Zwei Aktivierungssequenzen sind möglich: Entweder werden D und danach F aktiviert, oder nur F. Erkennungskonflikte sind im Rahmen dieser Arbeit wichtig, weil sie die Simulationseffizienz beeinflussen können und ihre Auswirkungen durch die Implementierung des Simulators erlaubt bzw. verhindert werden können.

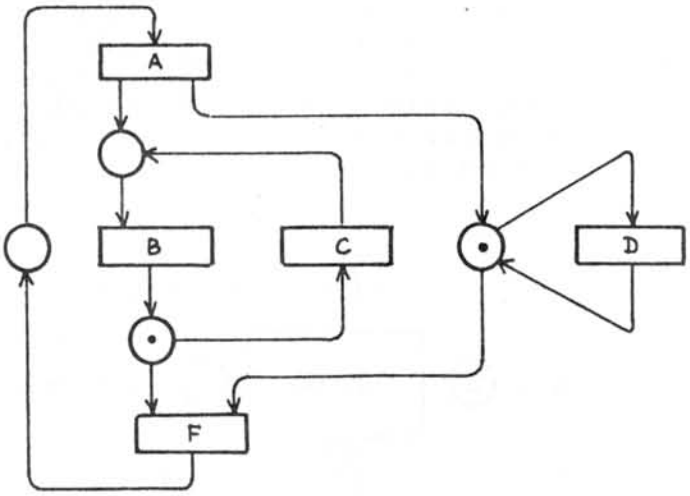


Abb. 2.2 - Erkennungs- und Zuständigkeitskonflikte (aus /WEN82/)

2.3 System als Netz kommunizierender programmierter Instanzen /WEN83/

Wir betrachten jetzt **Programmierte Systeme.** Weil Programme von nichtsequentieller Natur sein können, stellen wir sie als Petri-Netze dar /WEN79a/ /WEN83/, wobei Anweisungen den Transitionen zugeordnet sind und Konflikte durch Testen von Programmvariablen gelöst werden können. Um ein komplexes programmiertes System implementieren zu können, teilen wir das entsprechende Petri-Netz in kleinere Komponenten auf, basierend auf semantischen Aspekten, die uns hier nicht interessieren. Eine solche Dekomposition kann zwei Konsequenzen haben: 1) Marken müssen zwischen den Programmkomponenten fließen; und 2) Anweisungen in verschiedenen Komponenten brauchen Zugang zu gemeinsamen Variablen. Wir sprechen dann von **Marken-** bzw. **Operandenkommunikation** zwischen Programmkomponenten. Diese beiden Kommunikationsarten können gleichzeitig in demselben System notwendig sein. Die durch diese Dekomposition entstandenen Systemkomponenten nennen wir **Programmierte Instanzen.** Abb. 2.3 zeigt zwei kommunizierende programmierte Instanzen und ihre interne Struktur. Sie bestehen aus einem Speicher für die Markierung des Petri-Netzes, einem Speicher für ihr Programm und einem Abwickler. Markenkommunikation bedeutet, daß eine Instanz die Markierung einer anderen ändern kann. Auf einer tieferen Ebene wird diese Kommunikation immer als Operandenkommunikation implementiert (durch den Zugang zum Markierungsspeicher).

Hier liegt ein wesentliches Konzept vor. Instanzennetze erlauben nur Operandenkommunikation zwischen den Instanzen, Netze allgemeiner programmierter Instanzen dagegen auch Markenkommunikation. Die Modellierung eines Systems als ein Instanzennetz bedeutet aber keine wesentliche Einschränkung,

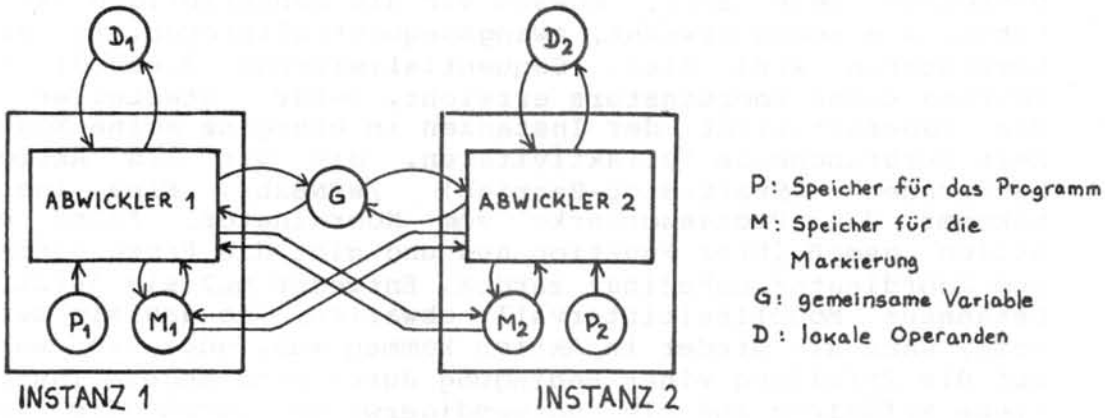


Abb. 2.3 - Kommunizierende programmierte Instanzen

weil wir Markenkommunikation immer als Operandenkommunikation modellieren können. Einen Verlust an Transparenz müssen wir gewiß in Kauf nehmen. Deswegen haben wir digitale Systeme auf der Gatter- und Register-Transfer-Ebene, deren Komponenten ausschließlich über Anschlußvariablen kommunizieren, als typische Systeme für die Arbeit genommen.

2.4 Nichtsequentielle Programme auf Monoprozessoren

Ein Netz kommunizierender programmierter Instanzen, sei es als Spezifikation eines zu realisierenden Systems oder als Modell für eine spätere Simulation gedacht, impliziert viele Abwickler, die nebenläufig agieren müssen. Wenn wir dieses System mit einem einzigen Monoprozessor realisieren bzw. das Modell für Simulationszwecke auf einen Monoprozessor bringen, müssen wir die nichtsequentiellen Programme zwangssequentialisieren. Das bedeutet, daß wir weitere Markenflüsse einführen müssen, um das neue Verhalten erklären zu können. Eine Unterscheidung zwischen verschiedenen Markentypen dient dem Verständnis /WEN83/. Die **Prozessormarke** gibt an, wo die Kontrolle über den einzigen Abwickler liegt, und sie kommt deshalb nur einmal im Modell vor. Eine **Sequenzermarke** ist für jede programmierte Instanz vorhanden und gibt an, wo sich die Instanz in ihrem Programm befindet, d.h. ab welchem Punkt das Instanzenprogramm weiter durchgeführt werden soll, wenn die Instanz die Prozessormarke zurückbekommt. In diesem Fall schmelzen beide Marken vorübergehend zusammen, bis die Instanz die Kontrolle an eine andere Instanz weitergibt.

2.5 Simulation

Als Ergebnis irgendeines Modellierungsverfahrens haben wir ein System als ein Netz programmierter Instanzen spezifi-

ziert. Wenn es ein Modell zur Simulation auf einem Monoprozessor sein soll, müssen wir die nebenläufigen Aktivitäten, wie schon erwähnt, zwangssequentialisieren. In allen Simulatoren wird diese Sequentialisierung durch die Einführung eines **Koordinators** erreicht. Dafür unterteilen wir die Daueraktivität der Instanzen in einzelne keine Modellzeit verbrauchende Teilaktivitäten, die wir als **Aktionen** bezeichnen (Start-Stop-Betrieb) /WEN82b/. Eine Instanz bekommt die Prozessormarke vom Koordinator, führt eine Aktion gemäß ihrer Funktion aus und gibt die Prozessormarke dem Koordinator unbedingt zurück. Entweder muß sie jetzt ein bekanntes Modellzeitintervall abwarten, so daß sie selbst weiß, wann sie wieder in Aktion kommen muß, oder sie wartet auf die Erfüllung einer Bedingung durch eine andere Instanz. Diese Erfüllung muß sie notwendigerweise durch die Wertänderung einer Anschlußvariable bemerken, wenn es sich um reine Instanzennetze handelt, d.h., wenn nur Operandenkommunikation möglich ist.

Der Koordinator hat folgende Hauptaufgaben: 1) Dispatchfunktion, d.h. nach irgendeinem Kriterium eine Ausführungsreihenfolge für die zum selben Zeitpunkt zu aktivierenden Instanzen festlegen; 2) Synchronisation zwischen Instanzen, d.h., Implementierung der Markenkommunikation; 3) Implementierung der Operandenkommunikation durch Aktivierung der Instanzen, die eine Änderung an ihren Eingangsvariablen haben (Variablen, auf die die Instanz lesenden oder modifizierenden Zugriff hat); und 4) Wenn keine Instanz mehr zum aktuellen Modellzeitpunkt aktivierbar ist, die Modellzeit bis zu dem Zeitpunkt fortschalten, zu dem eine Instanz gemäß ihrer Funktion wieder in Aktion gebracht werden muß.

Wenn der Modellierer die Spezifikation eines Systems als ein Petri-Netz hat und dieses in unterschiedliche Instanzen aufteilt, müssen die Beziehungen zwischen den daraus resultierenden Instanzen die Existenz eines bestimmten Koordinators voraussetzen /POT82/, der die obengenannten Aufgaben in irgendwelcher Form implementiert (vielleicht nur teilweise) und deswegen eine bestimmte Instanzendekomposition verlangt.

Ereignisse, die zu einem bestimmten Modellzeitpunkt eintreffen, sind im realen System gleichzeitig. Die Instanzenaktionen, die diesen Ereignissen entsprechen, müssen jedoch zum Zweck der Simulation, wie oben erwähnt, zwangssequentialisiert werden. Wir nennen **Systemüberführung** den sequentiellen Vorgang, der das System vom Modellzeitpunkt t1 zum nächsten Modellzeitpunkt t2 bringt. Im Falle der Instanzennetzsimulation entspricht der Systemüberführung eine Reihe von Instanzenaktionen.

2.6 Gängige Simulationskonzepte

In der Literatur werden Simulatoren in drei Klassen eingeteilt: ereignis-, prozeß- und aktivitäts-orientierte Simulatoren /KIV69/. Diese Klassifizierung bezieht sich auf

die Art, in der der Benutzer die Prozesstruktur berücksichtigt, und folgenderweise auf die Dekomposition des Systems in Instanzen. Wir können deswegen diese Konzepte aufgrund der Funktionen des Koordinators und der erlaubten Beziehungen zwischen den Instanzen erklären /POT82/.

Ereignis- und prozeß-orientierte Simulatoren erlauben Markenkommunikation zwischen den Instanzen, d.h. sie sind für Netze allgemeiner programmierter Instanzen geeignet. Ereignis-orientierte Simulatoren ordnen den Instanzen einzelne Transitionen des das System spezifizierenden Petri-Netzes zu, während prozeß-orientierte Simulatoren den Instanzen Gruppen von Transitionen zuordnen.

Aktivitäts-orientierte Simulatoren erlauben nur Operandenkommunikation zwischen Instanzen. Der Koordinator leistet aber keine Hilfe für diese Kommunikation. Er bringt alle Instanzen zu jedem Modellzeitpunkt in Aktion, und die Instanzen selbst müssen feststellen, ob ihre neuen Umgebungen eine Aktivität zu diesem Zeitpunkt erfordern oder nicht. Falls eine Instanzenaktion die Instanzenumgebung ändert, teilt sie das dem Koordinator mit, und er muß alle Instanzen wieder in Aktion bringen. Wir können diese Simulatoren zwar als Instanzennetzsimulatoren klassifizieren. Das ausgewählte Modell für Instanzennetzsimulation unterscheidet sich aber von diesem Konzept dadurch, daß es eine effizientere Operandenkommunikation implementiert.

Es muß gesagt werden, daß die meisten Simulatoren die Modellzeitfortschaltung realisieren, indem sie Systemzustandsänderungen, die zu künftigen Modellzeitpunkten stattfinden müssen, in einer "Ereignisliste" vermerken. Diese Art der Modellzeitfortschaltung, die wir auch für den Instanzennetzsimulator verwenden, hat jedoch mit der obengenannten Klassifizierung nichts zu tun. In diesem Sinne sind die in dieser Arbeit definierten Simulatoren keine ereignis-orientierten Simulatoren.

Wir können sagen, daß die Instanzennetzsimulation ein Oberbegriff für die aktivitäts-orientierte Simulation ist. Instanzennetzsimulation läßt sich nicht in die obengenannten Klassen einordnen.

2.7 Instanzennetzsimulation

Dieser Abschnitt stellt das ausgewählte Modell für Instanzennetzsimulation vor, nämlich die Aufgabenverteilung zwischen Koordinator und Instanzen. Damit der Abschnitt mit dem endgültigen Algorithmus für den Koordinator beendet werden kann, müssen wir jedoch vorher viele Implementierungsentscheidungen treffen. Einige davon beeinflussen auch die Systemmodellierung.

2.7.1 Instanzennetzsimulationsmodell

Das ausgewählte Modell erlaubt aufgrund der Definition von Instanzennetzen reine Operandenkommunikation zwischen den Instanzen. Der Koordinator unterstützt diese Kommunikation, indem er die Netzverschaltung kennt und damit die Instanzen identifiziert, die von einer Wertänderung einer Netzvariable betroffen werden. Dem Koordinator sind folgende Aufgaben zugeordnet:

a) Information aus den Instanzen bekommen, ob sie ihre Umgebung (U-Bedingung) geändert haben, und ob sie zu einem späteren Modellzeitpunkt (T-Bedingung) unbedingt in Aktion gebracht werden müssen;

b) Instanzen in Aktion bringen, für die eine Bedingung erfüllt ist, sei es eine U-Bedingung, falls die Umgebung geändert worden ist, oder eine T-Bedingung, falls die Modellzeit eingetreten ist, in der die Instanz eine neue Aktion verlangt; und

c) Modellzeit fortschalten, wenn keine Instanz mehr zu diesem Zeitpunkt aktivierbar ist, bis die nächste T-Bedingung erfüllt ist.

Abb. 2.4 zeigt die Kausalstruktur einer allgemeinen Instanz (aus /POT82/). Sie soll dem Koordinator
a) eine **Umgebungsänderungsmeldung** übergeben, falls sie eine Umgebungsvariable geändert hat, und
b) eine **Zeitmeldung** übergeben (mit Angabe einer künftiger Modellzeit), falls ihre Struktur eine Aktion zu einem späteren Zeitpunkt verlangt.

Die Instanz soll die Prozessormarke erst an den Koordinator abgeben, wenn keine weitere Aktivität zu diesem Zeitpunkt möglich ist. Wichtig in diesem Instanzennetzsimulationsmodell ist, daß der Koordinator eine Instanz aus verschiedenen Gründen aktivieren kann, und daß diese Gründe sich überlappen können. Deswegen bekommt die Instanz vom Koordinator keine Angabe über den Aktivierungsgrund. Sie selbst soll sich ihren Zustand merken und entscheiden, aufgrund des aktuellen Zustandes, der aktuellen Zeit und der Umgebungsvariablen, welche Aktion durchzuführen ist. Das bedeutet, daß die Instanz selbst die Position der Sequenzermarke verwalten und aufgrund dieser Position (einer Stelle SMi in Abb. 2.4) die Entscheidung treffen muß. Der Koordinator hat keinen Einfluß auf die Sequenzermarkenverwaltung, die Instanz bekommt die Prozessormarke immer an der gleichen Stelle (PM in Abb. 2.4).

Die Kommunikation zwischen Instanz und Koordinator wird durch den Raum KOM geschaffen. Über TC erfährt die Instanz die aktuelle Modellzeit (in der Implementierung ist TC eine globale Variable). Die Variablen TF und CHE werden vom Koordinator mit dem Wert "leer" vorbesetzt, bevor er die Instanz in Aktion bringt, und über sie teilt die Instanz dem Koordinator eine Zeitmeldung bzw. eine Umgebungsänderung

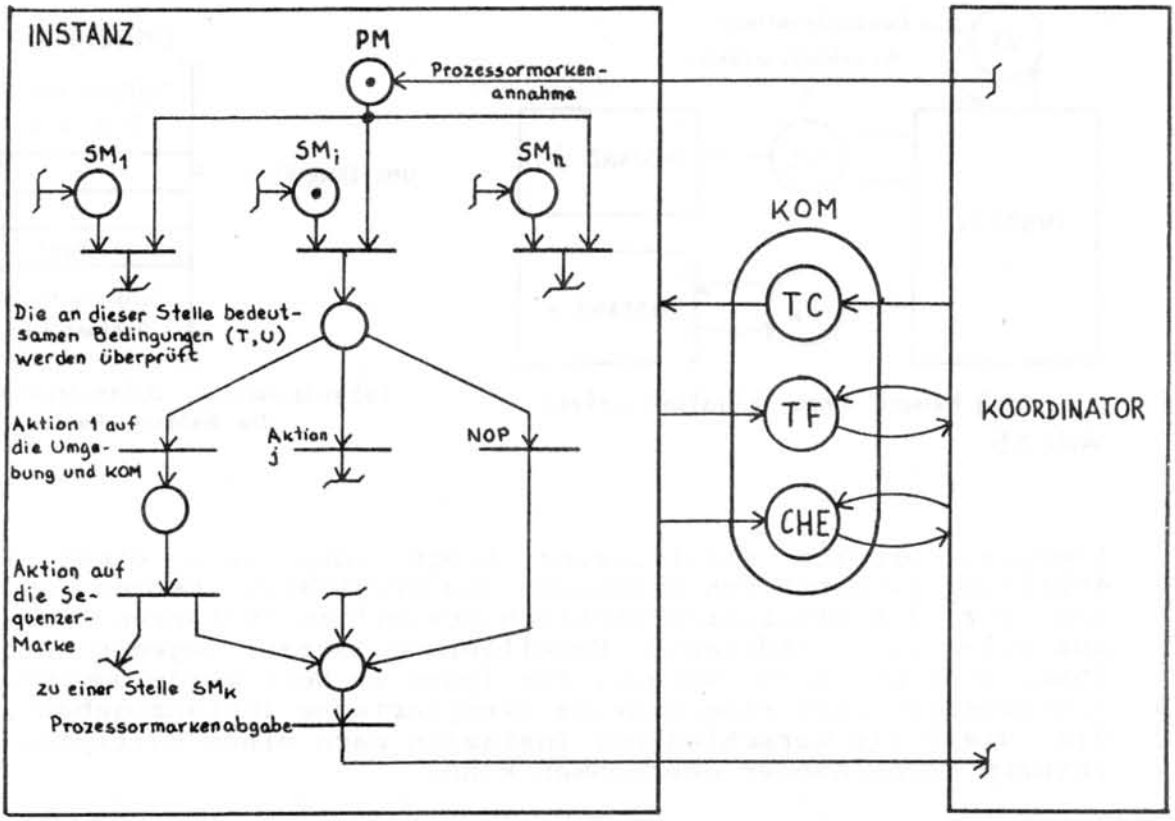


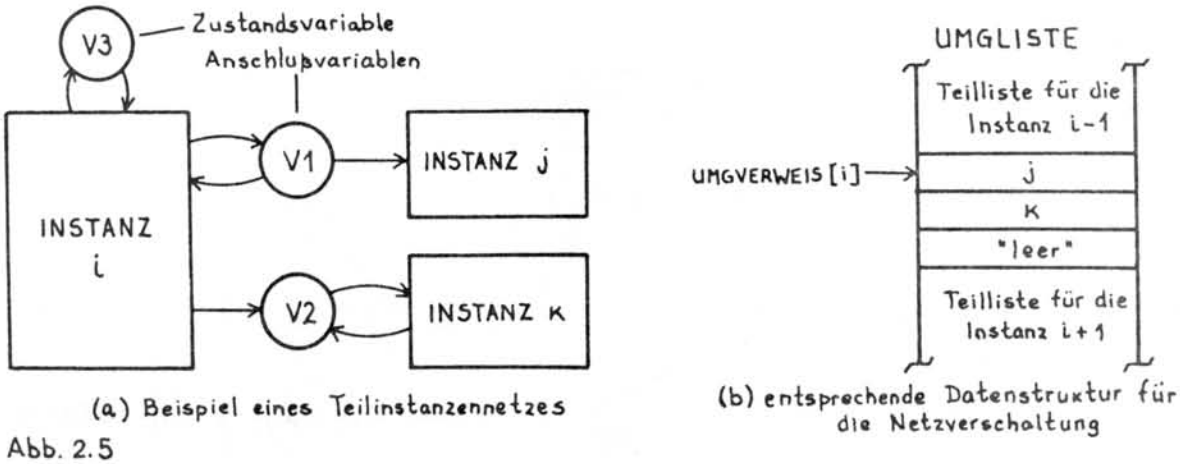
Abb. 2.4 - Allgemeine Kausalstruktur einer Instanz und Kommunikation mit dem Koordinator

mit. Die Instanz besetzt TF mit einer neuen Modellzeit und CHE mit dem Wert "belegt".

Aus der Forderung an den Koordinator, die Netzverschaltung zu kennen, um die Operandenkommunikation zu unterstützen, leiten wir die in Abb. 2.5 gezeigte Datenstruktur ab. Der Koordinator ordnet jeder Instanz eine Variable UMGVERWEIS zu, die auf eine UMGLISTE verweist. Die Liste ist aus Teillisten zusammengesetzt, eine für jede Instanz. In der Abb. 2.5 enthält die Teilliste i die Namen der Instanzen j und k, die lesenden oder modifizierenden Zugriff auf Variablen (V1 bzw. V2) haben, auf die die Instanz i setzenden oder modifizierenden Zugriff hat. Eine Teilliste endet immer mit einem mit dem Wert "leer" besetzten Wort.

2.7.2 Indirekte Modellierung

Wenn wir ein zu realisierendes System durch ein Instanzennetz spezifizieren, haben wir die Wahl zwischen indirekter und direkter Realisierung dieses Netzes /WEN83/. Wenn das Instanzennetzmodell jedoch Simulationszwecken dienen soll, sprechen wir lieber von direkter oder indirekter Model-



(a) Beispiel eines Teilinstanzennetzes

(b) entsprechende Datenstruktur für die Netzverschaltung

Abb. 2.5

lierung. Direkte Modellierung liegt vor, wenn eine 1:1 Abbildung zwischen den Instanzen des originalen Netzes und den für die Simulation wirklich erstellten Instanzen festzustellen ist. Indirekte Modellierung setzt dagegen ein Instanzenrepertoire voraus. Für jeden im Netz vorgekommenen Instanzentyp wird eine einzige exemplarische Instanz gebaut, die die Rolle verschiedener Instanzen nach einem Multiplex-Prinzip nacheinander übernehmen kann.

Der Koordinator muß dann auch die Multiplex-Funktion durchführen. Wenn er eine Instanz agieren läßt, bedeutet das, daß er eine allgemeine Instanzenprozedur aufruft, die die Namen sämtlicher Anschluß- und Zustandsvariablen bekommen soll, d.h. die die Rolle einer bestimmten Instanz des Netzes übernimmt.

Im weiteren verwenden wir bezüglich der Instanzen den Ausdruck "aufrufen" als gleichbedeutend mit "in Aktion bringen".

Wir haben uns im Rahmen dieser Arbeit für die indirekte Modellierung entschieden. Diese Annahme muß für alle analysierten Simulationsalgorithmen gelten, damit der Vergleich sinnvoll ist. Direkte Modellierung bringt zwar eine größere Effizienz, weil die Multiplex-Funktion und die allgemeine Datenstruktur (siehe nächsten Abschnitt) entfallen. Dafür verlieren wir aber die Übersicht über allgemeine Eigenschaften, weil die Instanzen immer für ein und nur ein betrachtetes System zugeschnitten sein müssen, und das war für die Entscheidung ausschlaggebend.

2.7.3 Datenstruktur für die Netzverschaltung

Da wir repertoirefähige Instanzen haben wollen, die ihre Verschaltung nicht kennen, muß der Koordinator Zugriff auf eine Datenstruktur haben, in der die Verschaltungsangaben abgespeichert sind. Für die Definition dieser Struktur sind

folgende allgemeine Netzeigenschaften entscheidend: a) Das Netz ist ungerichtet, d.h. Anschlußvariablen sind zu verschiedenen Zeitpunkten einmal Eingangs-, ein andermal Ausgangsvariablen; b) Jede Instanz darf eine beliebige Anzahl von Anschluß- und Zustandsvariablen haben; c) Die Anschluß- und Zustandsvariablen können von beliebigen Typen sein; und d) Eine Instanz kennt die mit ihr verbundenen Instanzen nicht.

Wir können davon ausgehen, daß es ein Array für jeden Datentyp im System gibt. Wir nennen sie die **SIGNAL-WERT-Arrays** (SW-Arrays). Da wir die Anschlußvariablen nicht eindeutig einer Instanz zuordnen können, weil das Netz ungerichtet sein kann, müssen wir annehmen, daß die typgleichen Anschlußvariablen einer Instanz über ein SW-Array zerstreut sind, so daß wir den Instanzennamen i nicht als Array-Verweis verwenden können. Die für eine Instanz notwendigen Verweise sind dann in einer Teilliste in nacheinander folgenden Positionen abgespeichert, und die Teillisten aller Instanzen bilden eine Liste. Weil jede Teilliste mehrere Verweise haben kann, dürfen wir den Instanzennamen auch nicht direkt als Index für den Teillistenanfang benutzen. Es ergibt sich eine Struktur mit dreifacher Indizierung. Obwohl die Zustandsvariablen eindeutig bestimmten Instanzen zugeordnet sind, müssen wir annehmen, daß eine Instanz n Zustandsvariablen von m Typen hat, so daß diese Variablen über viele Arrays zerstreut sind. Wir können auch hier den Instanzennamen nicht als direkten Verweis auf die Variablen verwenden. Es wird ebenfalls eine dreifache Indizierung benötigt.

Da wir für die Angaben über die Anschluß- und Zustandsvariablen eine ähnliche Datenstruktur brauchen, können wir eine einheitliche Struktur verwenden, wie Abb. 2.6 zeigt für die Instanz i der Abb. 2.5. Die PARAMLISTE enthält für jede Instanz Verweise auf ihre Anschlußvariablen (V1 und V2 für die Instanz i) und Zustandsvariablen (V3 für i). Auf den Anfang der Teil-PARAMLISTE einer Instanz wird von einem Array PARAMVERWEIS gezeigt, das durch den Instanzennamen indiziert wird.

Der Variablentyp für die Anschluß- und Zustandsvariablen braucht nicht in dieser Struktur abgespeichert zu sein. Der Koordinator hat einen systemabhängigen Programmteil, der diese Typen kennt (siehe Abschnitt 2.7.6). Auch systemabhängig sind die Anzahl, Typen und Größen der SW-Arrays. Während der Übersetzung der Netzangabe in die für den Simulator notwendige interne Struktur werden sie definiert und dimensioniert.

2.7.4 Ereignisliste

Die Zeitfortschaltungsfunktion realisiert der Koordinator, indem er für die von den Instanzen gelieferten Zeitmeldungen Einträge in eine **Ereignisliste** (EL) macht. Jeder Eintrag in dieser Liste wird als **Ereignisnotiz** bezeichnet. Wir

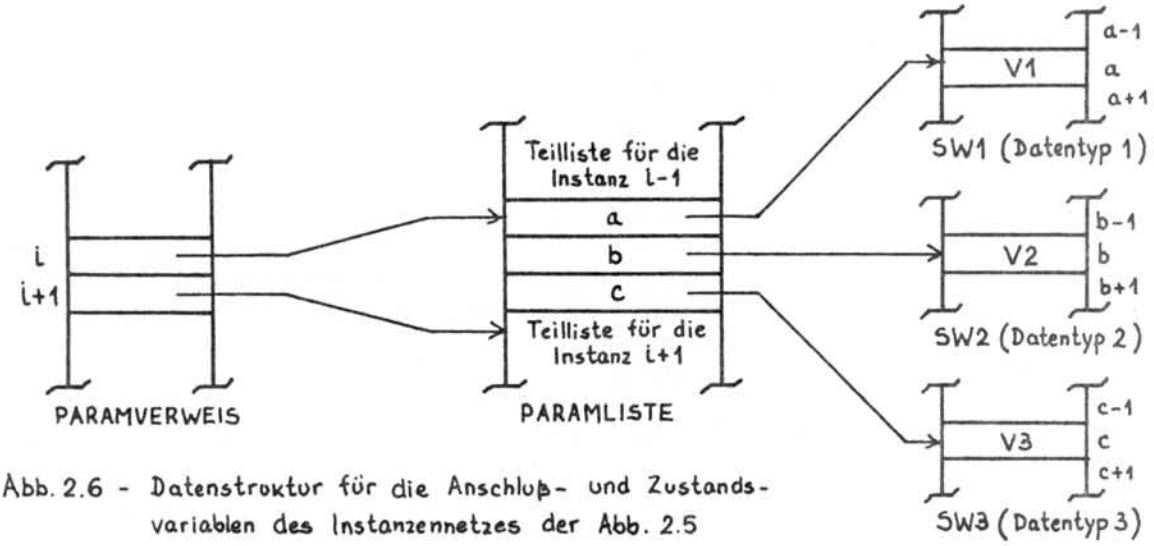


Abb. 2.6 - Datenstruktur für die Anschluß- und Zustandsvariablen des Instanzennetzes der Abb. 2.5

implementieren normalerweise eine Notiz als einen Record, dessen Felder Ereignisattributen entsprechen, z.B. dem Modellzeitpunkt, zu dem das Ereignis geschehen soll (Plan**zeit** genannt) und dem Namen der Instanz, die aktiviert werden muß.

Drei Grundoperationen müssen auf dieser Liste gemacht werden: a) Die Modellzeit bis zum Zeitpunkt des nächsten Ereignisses (d.h. Ereignisnotiz mit kleinster Planzeit) fortschalten; b) Alle Notizen identifizieren, die eine mit der laufenden Zeit identische Planzeit haben; und c) Neue Notizen in die Liste eintragen. Das modellierte System wird immer als ein geschlossenes System gesehen, d.h., wir müssen die Umgebung für das eigentlich gewünschte System auch durch Netzinstanzen modellieren, so daß exogene Ereignisse /GOR78/ nicht vorkommen. Alle Notizen in der Liste resultieren aus Aktionen modellierter Netzinstanzen.

Aus der Literatur ist eine Reihe von Organisationen für die Ereignisliste (Datenstruktur + Algorithmen für die Operationen auf der Liste) bekannt, die mehr und mehr versuchen, die Effizienz der Ereignisliste in Bezug auf die drei obengenannten Operationen zu optimieren, insbesondere die Eintragung neuer Notizen. Da alle Algorithmen die Notizen immer nach wachsender Planzeit in die Liste eintragen, sind Operationen a) und b) von Komplexität O(1). Die vorgeschlagenen Organisationen unterscheiden sich deswegen voneinander hauptsächlich durch die Datenstruktur für die Liste und den Algorithmus für die Eintragung neuer Notizen. Im weiteren verwenden wir den Ausdruck "Algorithmus" in Bezug auf die Ereignisliste als gleichbedeutend mit "Organisation", obwohl klar ist, daß zusätzlich zu dem Algorithmus eine Datenstruktur vorhanden sein muß. Comfort /COM79/ bietet eine Zusammenfassung und einen Vergleich zwischen den wichtigsten Algorithmen an.

Die Eintragungsdauer einer Notiz ist eine Funktion der Zahl S der bei der Eintragung durchsuchten Notizen und der Algorithmenkomplexität. Die Zahl S wiederum ist eine Funktion der Anzahl N der Notizen in der Liste und der Zeitverteilung der Ereignisse. Da der Instanzennetzsimulator für allgemeine Anwendungen gebaut werden muß, dürfen wir keine restriktive Annahme über N und die Zeitverteilung machen. Ein Kompromiß zwischen Algorithmeneffizienz und -Komplexität ist deswegen zu finden.

Der L-Algorithmus (EL als eine lineare Liste organisiert) ist am einfachsten, aber N/2 Notizen werden im Durchschnitt abgesucht. Diese O(N)-Komplexität verbietet seine Anwendung. Der **IL-Algorithmus** /WYM75/ /VAU75/ (EL als eine indizierte Liste organisiert) versucht S zu verringern, indem er die Liste in R gleich große Intervalle aufteilt und einen Verweis auf das Ende jedes Intervalls verwaltet. Wir bekommen eine O(N/R)-Komplexität. Dieser Algorithmus hat drei Probleme: a) Er braucht ein letztes Intervall, welches alle Notizen bekommt, deren Planzeiten außerhalb der Zeitspanne der R Intervalle liegen, und für "bösartige" Zeitverteilungen kann dieses Intervall die Effizienz stark benachteiligen; b) Die Zahl R stellt einen Kompromiß dar, weil eine zu große Zahl von Intervallen für Systeme mit kleinem Wert von N schlecht ist, wegen des relativ großen Aufwandes der Intervallverwaltung, während eine zu kleine Zahl dagegen für Systeme mit großem Wert von N schlecht ist, weil S immer noch groß bleibt; und c) In den Verteilungsgipfeln bekommen die entsprechenden Intervalle zu viele Ereignisse, in den Verteilungslücken dagegen zu wenige.

Komplexere Algorithmen versuchen diese Probleme zu lösen, indem sie die Anzahl und die Zeitspanne der Intervalle dynamisch veränderlich machen ("anpassungsfähige" Algorithmen). Der AL-Algorithmus /WYM75/ /COM79/ bietet $O(\sqrt{N})$-Komplexität und der TL-Algorithmus /FRA77/ und sein äquivalenter I/AL-Algorithmus /COM79/ sogar O(1)-Komplexität, aber dafür sind sie sehr komplex. Es wird gezeigt, daß schon ab $N \cong 80$ der I/AL-Algorithmus einen kleineren Wert für S als der IL-Algorithmus liefert, aber erst ab $N \cong 240$ die Eintragungsdauer kleiner ist, d.h., wir haben eine größere Eintragungsdauer je Notiz /COM79/.

Wir haben uns im Rahmen dieser Arbeit für den IL-Algorithmus entschieden, um unsere Analyse nicht unnötigerweise zu erschweren. Um den Nachteil des IL- gegenüber dem AL- und dem TL-Algorithmus auszugleichen, werden wir für jedes zu simulierende System die Zeitspanne eines Intervalls optimal dimensionieren. Dies würde in den anpassungsfähigen Algorithmen automatisch gemacht werden.

Abb. 2.7 veranschaulicht die von dem IL-Algorithmus für die Ereignisliste vorgesehene Datenstruktur. Die Notizen sind doppelt verkettet. Die Vorwärtsverkettung ist für die Zeitfortschaltung nötig, während die Rückwärtsverkettung eine FIFO-Strategie für gleichzeitige Ereignisse unterstützt, indem die Suche des Eintragungsplatzes innerhalb

eines Intervalls rückwärts läuft. Scheinereignisnotizen ("dummy notices"), die keiner Instanzenzeitmeldung entsprechen, begrenzen die Intervalle, und auf sie wird aus einem Array POINTER verwiesen. Eine Scheinnotiz ist für die Zeit "plus unendlich" (oder Null) vorhanden und mit der ersten und der letzten Notiz der Liste verkettet, so daß die Liste eine zyklische Organisation hat. POINTER hat eine Größe MAX+1 und ist für die Stellen 0..MAX-1 zyklisch organisiert. Der Inhalt der Stelle MAX des Arrays zeigt immer auf das letzte Intervall vor "unendlich". Eine Variable ICURRENT gibt die Arraystelle an, die auf das aktuelle Intervall verweist. Die Verweise zeigen immer auf die Intervallenden, damit die Absuche im Intervall rückwärts erfolgt. Jedes Intervall, ausgenommen das letzte, deckt eine Zeitspanne DT. Die Variable CURRENT verweist immer auf die Notiz mit nächster Planzeit, und LOWERBOUND gibt an, welche Anfangsmodellzeit das aktuelle Intervall hat.

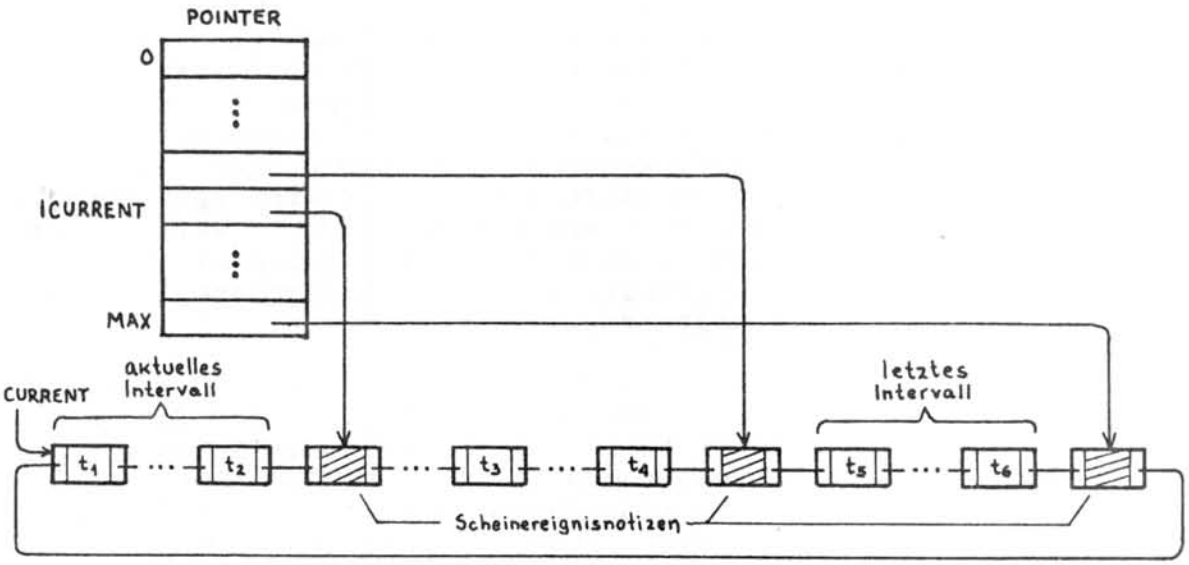
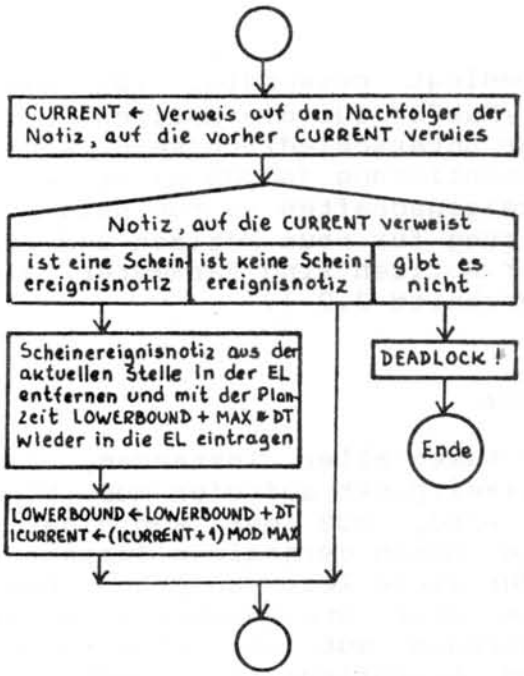


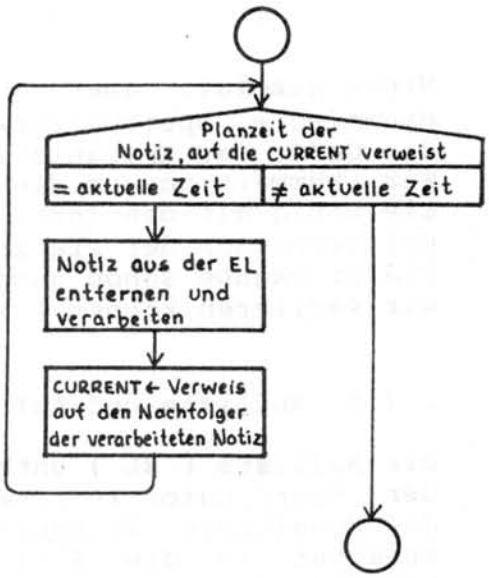
Abb. 2.7 - Datenstruktur für die Ereignisliste nach dem IL-Algorithmus

Abb. 2.8 zeigt die drei Grundoperationen auf der Ereignisliste, wenn wir die IL-Organisation verwenden. Jede Entfernung und Eintragung einer Notiz erfordet eine nicht gezeigte Verwaltung der Verweise, die für die Doppelverkettung der Notizen in der Liste verantwortlich sind.

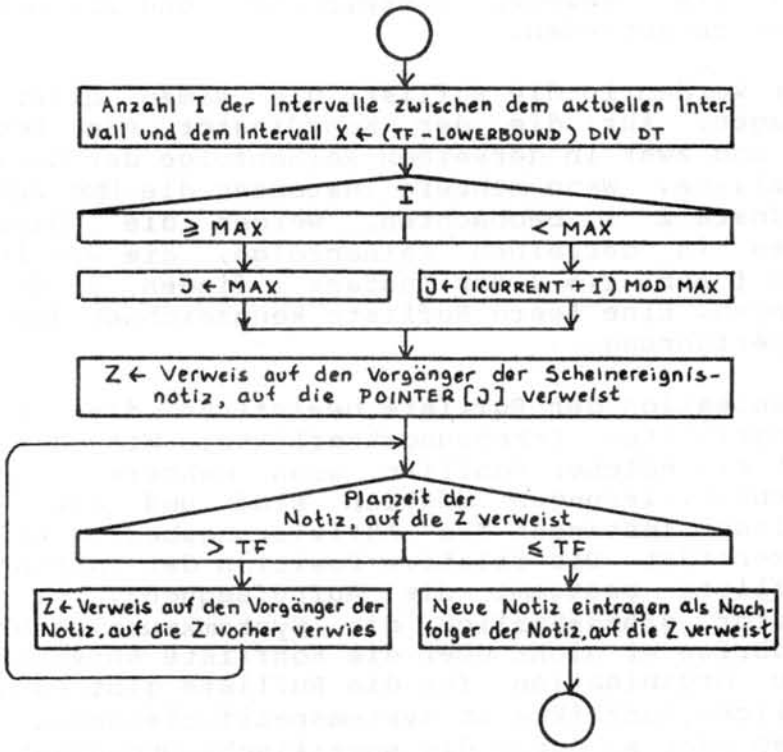
Die genaue Implementierung dieses Algorithmus ist im Abschnitt 2.7.6 zu finden. Eine Arraygröße MAX=30 wird in der Literatur als optimal bezeichnet /COM79/. Die Intervallgröße DT erwarten wir als einen Eingangsparameter. Wir werden den optimalen Wert dieser Größe für jedes zu simulierende System als eine Funktion der Zeitverteilung der Ereignisse feststellen.



(a) Zeitfortschaltung



(b) Identifizierung der Ereignisnotizen mit Planzeit = aktuelle Modellzeit



X = Intervall, in das die Notiz einzutragen ist
TF = Planzeit der einzutragenden Notiz

(c) Eintragung einer Ereignisnotiz

Abb. 2.8 - Grundoperationen auf der Ereignisliste nach dem IL-Algorithmus

Nicht gezeigt, aber auch unbedingt notwendig, ist eine dynamische Speicherverwaltung /KNU73/ für die Notizen, weil die Größe der Ereignisliste sehr unterschiedlich sein kann. Wir kümmern uns um diese Implementierung jedoch nicht, weil sie wenig mit den Instanzennetzeigenschaften zu tun hat. Der Zeitverbrauch der Platzreservierung für neue Notizen und der Platzrückgabe schon ausgeführter Notizen sind Parameter, die wir variieren können (siehe Abschnitt 3.2).

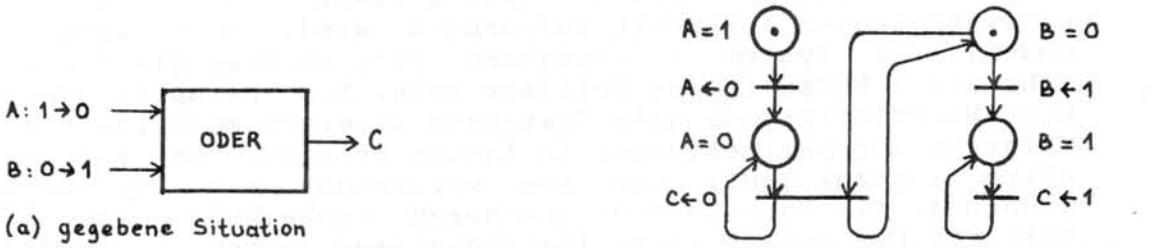
2.7.5 Rufliste und Aufrufsequenz

Die **Rufliste** (RL) enthält die Namen aller Instanzen, die der Koordinator zu einem Modellzeitpunkt aufrufen muß. Wenn die Modellzeit fortgeschaltet wird, muß der Koordinator zunächst in die Rufliste die Namen derjenigen Instanzen bringen, die eine Zeitmeldung für diese Zeit an ihn abgegeben haben, so daß für sie eine Ereignisnotiz in der Ereignisliste zu finden ist. Außerdem muß der Koordinator andere durch Umgebungsänderung betroffene Instanzen aktivieren. Nur zwei Operationen sind dann auf der Rufliste notwendig, nämlich die Eintragung und das Abholen eines Instanzennamens. Eine einfache Organisation ist für die Rufliste deswegen ausreichend, und wir haben uns für eine FIFO entschieden.

Zunächst werden in die Rufliste die Namen aller Instanzen eingetragen, für die der Koordinator eine Ereignisnotiz findet, und zwar in derselben Reihenfolge der Notizen in der Ereignisliste. Wenn mehrere Instanzen die Umgebungsänderung einer Instanz i beobachten, werden die Namen dieser Instanzen in derselben Reihenfolge, die wir in der Teil-UMGLISTE (Abb. 2.5) der Instanz i finden, in die Rufliste eingetragen. Eine leere Rufliste kennzeichnet das Ende einer Systemüberführung.

Die Organisation der Rufliste beeinflußt die im Abschnitt 2.2 eingeführten Erkennungskonflikte. Wie dort erwähnt, entsteht ein solcher Konflikt, wenn mehrere Sequenzen von Instanzenaktivierungen möglich sind und die Aktivierung einer dieser Instanzen die Aktivierungsbedingung für eine andere zerstört. Die relative Position der Instanzennamen in der Rufliste bestimmt die Aufrufsequenz. Da aber der Koordinator grundsätzlich die Systemkausalstruktur nicht kennt, dürfte er nicht über die Konflikte entscheiden. Eine neutrale Organisation für die Rufliste gibt es aber nicht, und mögliche Konflikte im systemspezifizierenden Petri-Netz werden so oder so durch die spezifische Organisation gelöst. Nur für konfliktfreie Netze würde der Simulator theoretisch immer dasselbe Ergebnis liefern, obwohl in der Praxis die deterministische Ruflisten-Organisation dies auch für konfliktbehaftete Netze tut. Der Modellierer muß aber damit rechnen, daß der Simulator die Erkennungskonflikte willkürlich löst. Für den Modellierer bleibt diese Organisation der Rufliste unbekannt, und er darf sich nicht auf eine gewisse Organisation verlassen, um ein Simulationsergebnis zu erzielen.

Abb. 2.9 zeigt ein Beispiel dazu. Wenn wir digitale Systeme modellieren, entspricht ein Erkennungskonflikt dem bekannten Hazard-Konzept /WEN74/. Zwei Übergänge A:1->0 und B:0->1 an den Eingängen eines ODER-Gatters sind für denselben Modellzeitpunkt geplant. In dem Petri-Netz erkennen wir einen Konflikt zwischen den Transitionen C<-0 und B<-1.



(a) gegebene Situation

(b) entsprechendes Petri-Netz (Teilmodellierung für die gegebene Situation)

Verzögerung im Verhältnis zum Zeitunterschied zwischen dem A- und dem B-Übergang	A←0 tritt vor B←1 auf	B←1 tritt vor A←0 auf
Verzögerung ist kleiner	A←0, C←0, B←1, C←1 (a)	B←1, C←1, A←0 (b)
Verzögerung ist größer	A←0, B←1, C←1 (c)	B←1, A←0, C←1 (d)

(c) mögliche Sequenzen von Signalübergängen

Abb. 2.9 - Erkennungskonflikte in Instanzennetzen und Hazards

Wir können durchaus vermuten, daß die Signale A und B in der Tat keine gleichzeitigen Übergänge haben, aber der Zeitunterschied zwischen den Übergängen kleiner ist als der kleinste modellierte Zeitschritt. Wir können auch vermuten, daß die Gatterverzögerung größer oder kleiner als dieser Zeitunterschied sein kann. Abhängig vom Verhältnis zwischen dem Zeitunterschied und der Gatterverzögerung können Hazardpulse am Gatterausgang auftreten. Aus dem Petri-Netz leiten wir vier mögliche Sequenzen von Transitionen ab, die verschiedenen Hardware-Realitäten entsprechen. In der Sequenz a ist ein Hazardpuls zu erkennen. Der Übergang A<-0 tritt vor B<-1 auf, und die Verzögerung ist genügend klein, so daß C auf 0 geht, noch bevor B<-1 auftritt.

In unserer Modellierung sind die Übergänge als Ereignisnotizen vermerkt. Durch die Organisation der Rufliste veranlaßt der Koordinator beide entsprechenden Instanzenaktionen, bevor sich irgendwelche Auswirkungen auf die ODER-Instanz bemerkbar machen. Die Sequenzen a und b kommen deswegen nie vor, und ein Hazardpuls, der auf die obengenannten Zeiteigenschaften zurückzuführen wäre, tritt nicht auf. Dem Benutzer soll diese Tatsache aber unbekannt sein, weil sie der Implementierung des Koordinators zuzuschreiben

ist.

Damit der Benutzer Hazards mit Sicherheit erkennen kann, braucht er selbstverständlich eine besondere Instanzennetz-modellierung des digitalen Systems. Auf diese Modellierung möchten wir jedoch nicht eingehen.

Es ist noch notwendig, die Rufliste zu dimensionieren. Es ist nicht vorauszusehen, wieviele Instanzen maximal zu einem einzigen realen Zeitpunkt aufzurufen sind. Auch wenn das modellierte System x Instanzen hat, könnten gleichzeitig mehr als x Namen in der Rufliste sein. Ein Beispiel dafür: Der Koordinator muß alle Instanzen zu einem Modellzeitpunkt aufgrund von Zeitmeldungen in Aktion bringen. Er ruft die erste Instanz auf, und sie verursacht eine Umgebungs-änderung, die in p anderen Instanzen beobachtbar ist. Die Rufliste hat danach x-1+p Instanzennamen, wobei einige davon doppelt vorkommen. Wegen der allgemeinen Anwendung des Simulators können wir keine Obergrenze für die Größe der Rufliste annehmen. Die Lösung heißt: a) Die Anzahl der Instanzen, deren Namen gleichzeitig in der Rufliste sein können, wird auf x begrenzt, indem wir jeder Instanz ein Flag zuordnen, das besagt, ob der Name der Instanz schon in der Rufliste ist. Das Flag wird zurückgesetzt, wenn der Koordinator die Instanz aufruft. b) Weil trotzdem im Laufe einer Systemüberführung der Name einer Instanz mehrmals in die Rufliste gebracht werden kann, weil Umgebungsänderungen zu jeder Zeit beobachtet werden können, versehen wir die Rufliste mit einer zyklischen Organisation, und zwar mit Größe x.

2.7.6 Endgültiger Algorithmus für den Koordinator

Abb. 2.10 zeigt den endgültigen Algorithmus für den Koordinator eines Instanzennetzsimulators, dessen Eigen-schaften den vorigen Abschnitten zu entnehmen sind.

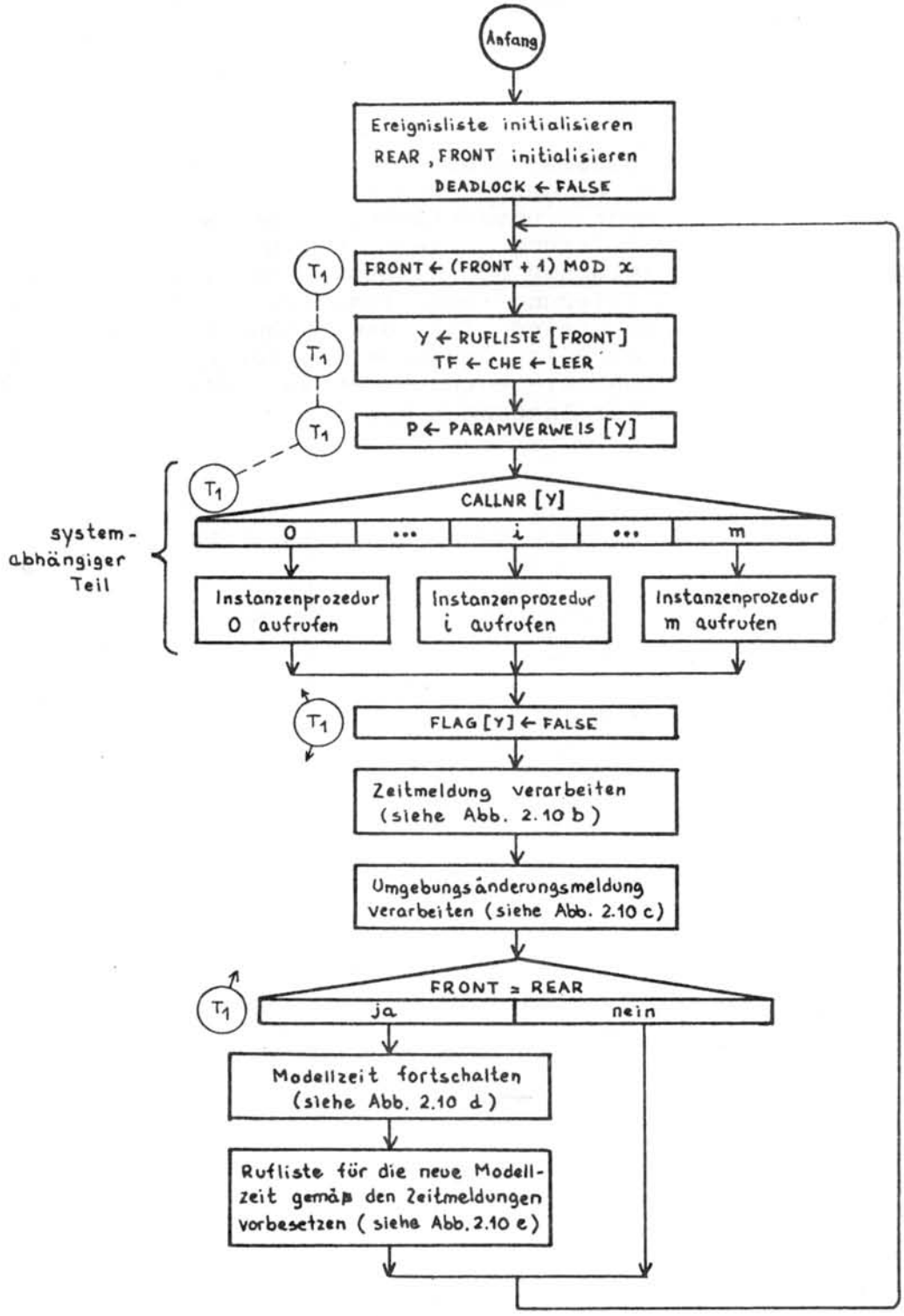
Dabei ist es zu beachten, daß dieser Algorithmus einen systemabhängigen Teil enthält, d.h., einen Teil, der nach dem zu simulierenden Netz compiliert wird. Ein Array CALLNR gibt für jede im Netz vorhandene Instanz den im Repertoire zugehörigen Instanzentyp an und bestimmt damit die aufzu-rufende Prozedur. Der systemabhängige Teil des Koordinators umfaßt die Identifizierung und den Aufruf dieser Prozedur. Diese Realisierung des Koordinators bringt eine größere Simulationseffizienz gegenüber der anderen möglichen Lösung, in der die Prozedurnamen, oder Verweise auf den Anfang der Prozeduren, in einer Tabelle abgespeichert wären.

Die Parameterübergabe sollte eigentlich auch vom Koordinator durchgeführt werden. Im systemspezifischen compilierten Teil hätte der Koordinator die Zahl, Typen und Adressen aller Instanzenparameter (Anschluß- und Zustandsvariablen) schon eingebaut. Dies entspricht der Multiplex-Funktion des Koor-dinators (siehe Abschnitt 2.7.2). Der Abschnitt 3.3 befaßt sich mit der verwendeten Lösung für die Parameterübergabe.

Eine Ereignisnotiz wird als ein Record implementiert, das 5 Felder enthält: NAME (den Namen der Instanz), EVTIME (die Planzeit des Ereignisses), DUMMY (ein boolesches Feld, das angibt, ob diese eine Scheinereignisnotiz ist), SUC und PRED (Verweise auf die folgende bzw. vorherige Notiz in der Ereignisliste).

2.7.7 Schlußfolgerung

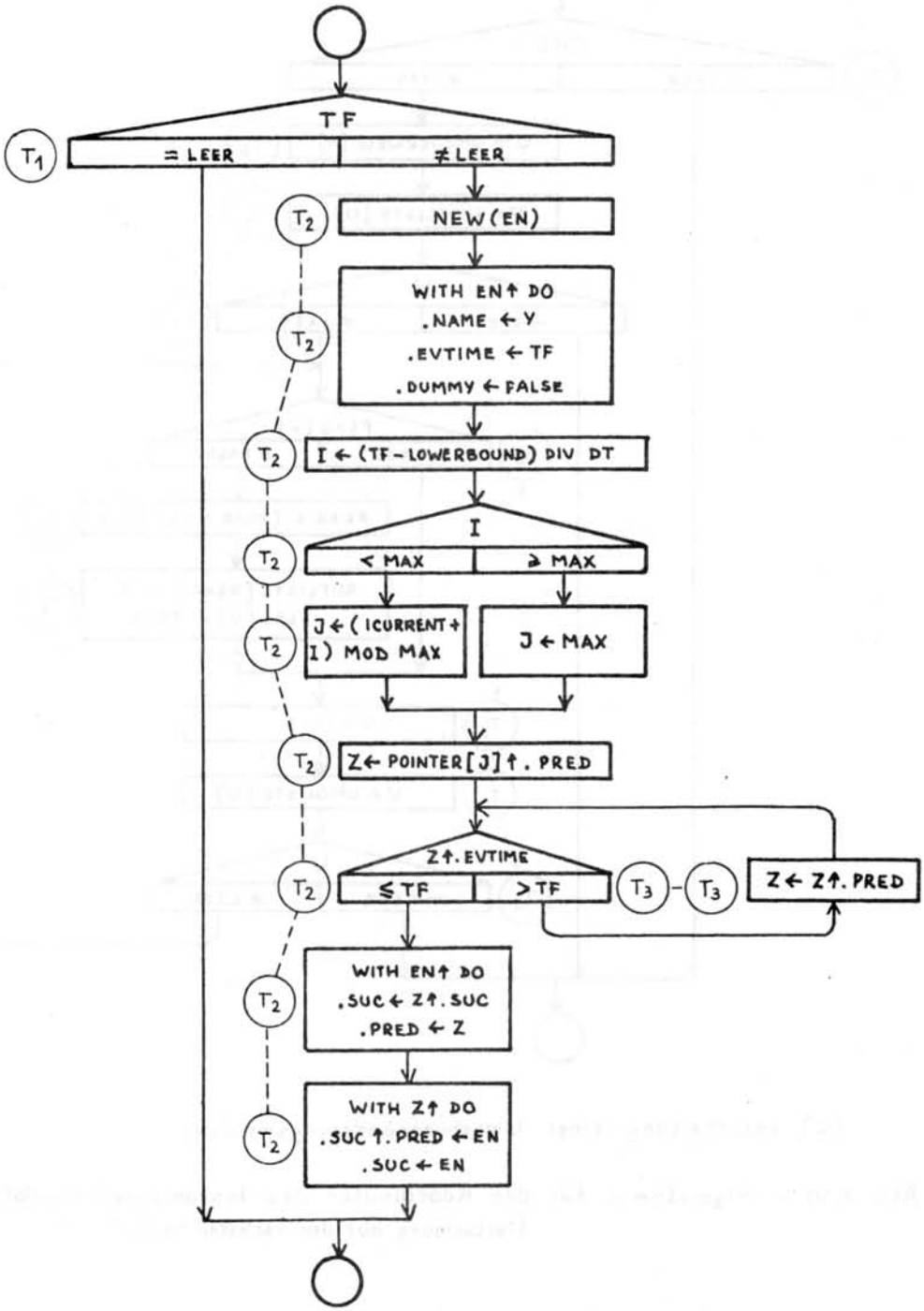
Als Schlußfolgerung dieses Kapitels können wir zusammenfassend sagen, daß für den Benutzer des Simulationssystems nur diejenige Implementierungsentscheidungen wichtig sind, die die Netzmodellierung beeinflussen, und zwar die Aufgabenverteilung zwischen Koordinator und Instanzen und die indirekte Modellierung von Instanzen. Alle anderen Implementierungsdetails sind für den Benutzer unwichtig. Insbesondere darf er die Organisation der Rufliste nicht berücksichtigen, obwohl sie entscheidenden Einfluß auf die Aktivierungssequenz der Instanzen hat.



(a) Hauptschleife

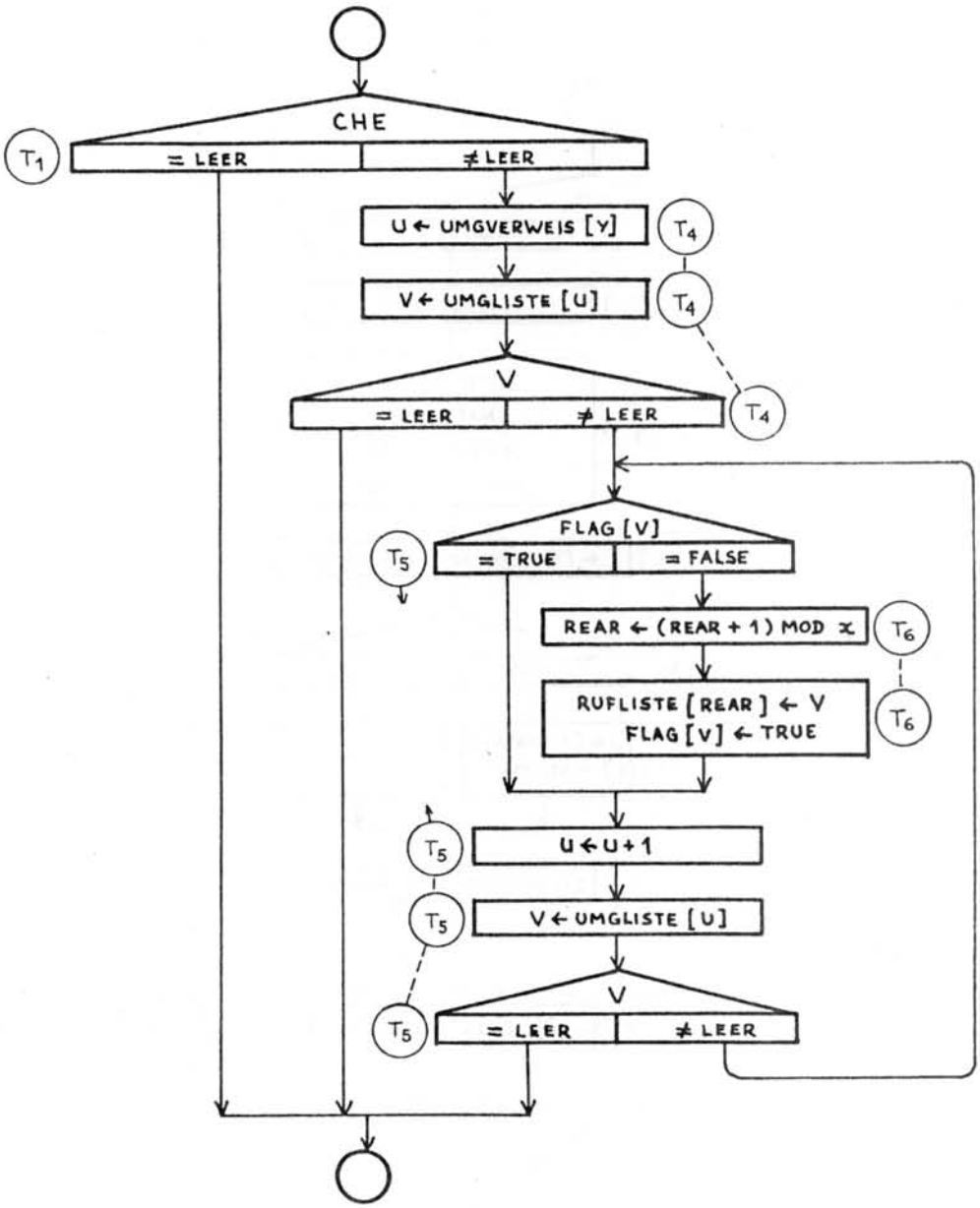
Abb. 2.10 - Algorithmus für den Koordinator des Instanzennetzsimulators

(Fortsetzung auf den nächsten Seiten)



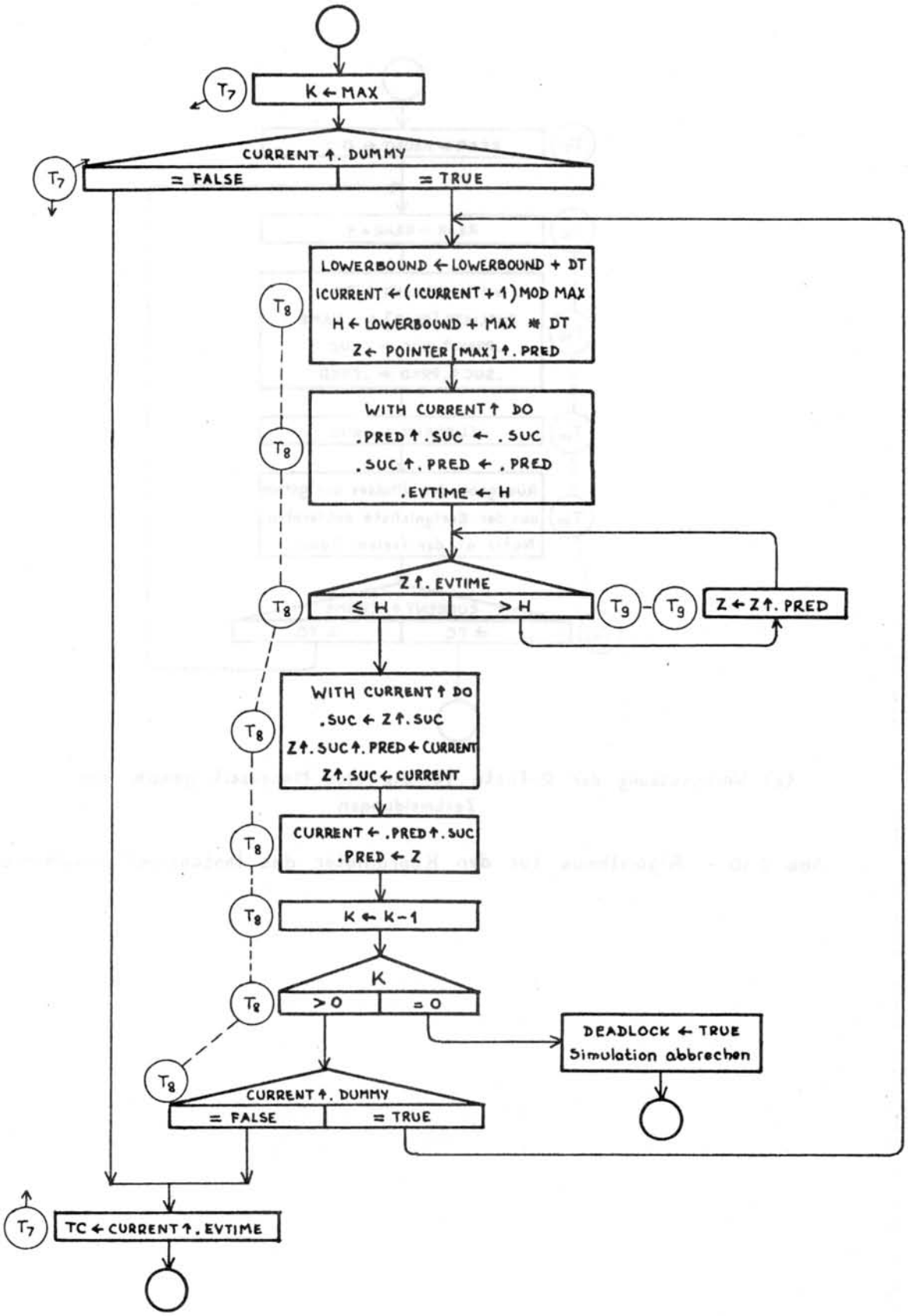
(b) Verarbeitung einer Zeitmeldung

Abb. 2.10 - Algorithmus für den Koordinator des Instanzennetzsimulators
(Fortsetzung auf den nächsten Seiten)



(c) Verarbeitung einer Umgebungsänderungsmeldung

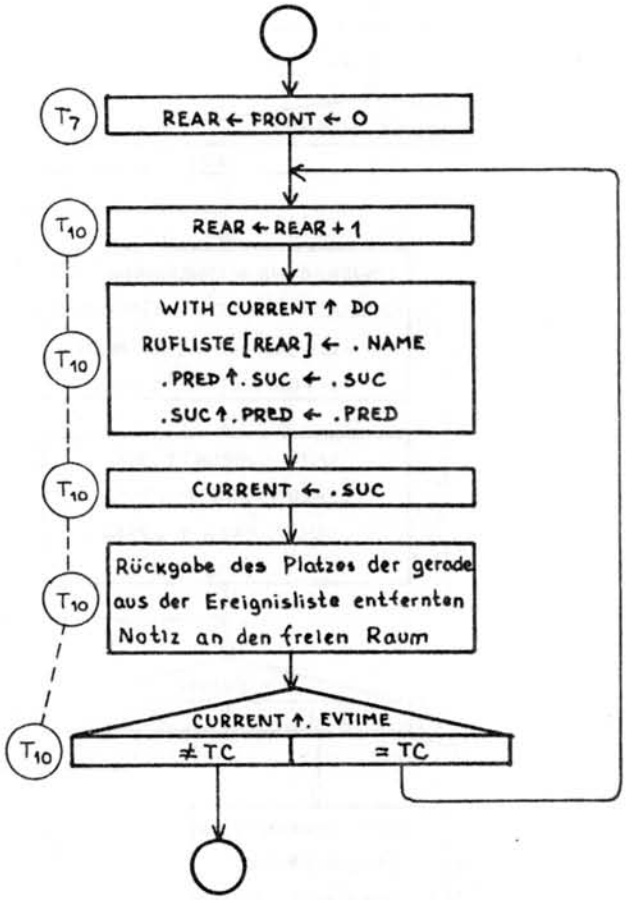
Abb. 2.10 - Algorithmus für den Koordinator des Instanzennetzsimulators
(Fortsetzung auf den nächsten Seiten)



(d) Fortschaltung der Modellzeit

Abb. 2.10 - Algorithmus für den Koordinator des Instanzennetzsimulators

(Fortsetzung auf der nächsten Seite)



(e) Vorbesetzung der Rufliste für die neue Modellzeit gemäß den Zeitmeldungen

Abb. 2.10 - Algorithmus für den Koordinator des Instanzennetzsimulators

3. Vergleichsmethode

3.1 Vergleichsansätze

Konkrete Aufgabe dieser Arbeit ist es, die unter Verwendung zweier unterschiedlichen Simulationsalgorithmen erzielten Simulationszeiten für verschiedene Systeme aus einer Klasse zu vergleichen.

Ein erster Vergleichsansatz wäre es, eine große Anzahl von Beispielsystemen auszusuchen, zu modellieren und unter Verwendung der beiden Algorithmen zu simulieren, und statistiche Angaben über die verbrauchte Simulationszeit zu sammeln. Diese Methode hat aber viele Probleme, die die erzielten Ergebnisse in Bezug auf ihre Gültigkeit in Frage stellen. Wieviele Systeme müssen wir simulieren? Wie können wir sicher sein, daß wir wirklich eine repräsentative Auswahl aus dem möglichen Klassenspektrum getroffen haben? Probleme sind auch mit der Genauigkeit der direkten Messung des Zeitverbrauchs verbunden /WUL81/. Außerdem verlangt diese Methode einen großen Modellierungsaufwand und setzt die Existenz der zu vergleichenden Algorithmen voraus, was auch einen großen Implementierungsaufwand bedeutet.

Ein anderer Ansatz wird üblicherweise für den Vergleich zwischen Algorithmen vorgeschlagen. Wir identifizieren eine typische Operation und stellen die Häufigkeit ihrer Ausführung für jeden Algorithmus fest /WUL81/, so daß eine Komplexitätsanalyse möglich wird. In numerischen und Sorting-Algorithmen z.B. sind typische Operationen leicht erkennbar: Arithmetische Operationen bzw. Vergleiche. Ein Simulationsalgorithmus hat solche typische Operationen allerdings nicht. Eintragung und Suche sind für die Ruf- und die Ereignisliste notwendig. Zugriffe auf die Systemvariablen in der Datenstruktur verbrauchen eine bedeutsame Zeit, wie wir später sehen werden. Für Hardware-Simulation spielen außerdem arithmetische und boolesche Operationen eine wichtige Rolle für die Verarbeitung der Zeitmodelle. Die Komplexitätsanalyse ist außerdem in dieser konkreten Arbeit schlecht anwendbar, weil, wie später bewiesen, alle betrachteten Simulationsalgorithmen Komplexität $O(p) + O(q) +\ldots$ haben, wo $p,q,\ldots$ Systemparameter sind, so daß der Unterschied zwischen den Algorithmen nur in den multiplikativen Konstanten liegt. Eine genaue Untersuchung dieser Konstanten wäre erforderlich, was wir bei der verwendeten Vergleichsmethode wirklich gemacht haben. Ein Beweis für diese Behauptung ist erst dann möglich, wenn wir in den Kapiteln 4 und 5 exakte analytische Zeitverbrauchsausdrücke für alle Algorithmen abgeleitet haben. In der Schlußfolgerung dieser Arbeit kommen wir nochmals zu diesem Thema und stellen diesen Beweis vor.

Wir kommen nun zu der eigentlich angewendeten Vergleichsmethode. Die zu simulierenden Systeme werden durch Parameter dargestellt. Diese Parameter beschreiben sowohl statische als auch dynamische Systemeigenschaften, die erst durch die

Simulation entstehen. Sie müssen so definiert sein, daß ein exakter analytischer Zeitverbrauchsausdruck für die Simulationsalgorithmen ableitbar ist und zwischen Parametern der zu vergleichenden Ausdrücke entweder eine Gleichheit oder erkennbare mathematische Beziehungen bestehen. Diese Systembeschreibung durch Parameter ermöglicht dann die Bearbeitung der in Kapitel 1 vorgestellten Hauptaufgaben dieser Arbeit. Durch die Variierung der Parameterwerte in vernünftigen Grenzen, die für jeden Parameter festzustellen sind, erfassen wir das ganze Klassenspektrum.

Aus einer Untersuchung der Sensibilität der Zeitverbrauchsausdrücke bezüglich der verschiedenen Parameter ergibt sich, mit welchem Satz von Parameterwerten wir den höchsten bzw. den kleinsten Simulationszeitgewinn des Hardware-Simulators gegenüber dem Instanzennetzsimulator erzielen. Es ist allerdings unbedingt erforderlich, die Beziehungen zwischen den Parametern festzustellen, die auf die Systemtopologie oder -Dynamik zurückzuführen sind. Erst unter Berücksichtigung dieser Beziehungen können wir die Parameterwerte variieren, ohne einen unerlaubten Freiheitsgrad anzunehmen. Die Aufteilung des Zeitgewinns in Faktoren, die den Einschränkungen in den universellen Instanzennetzeigenschaften zuzuschreiben sind, erfolgt dann auch durch eine analytische Aufteilung der Zeitverbrauchsausdrücke.

Als Voraussetzung für diese Vergleichsmethode müssen wir eine Annahme über die Ausführungszeiten sämtlicher Algorithmenschritte machen. Obwohl dies ein erkennbarer Nachteil der Methode ist, weil die Ergebnisse abhängig von dieser Annahme sind und diese auf konkrete oder hypothetische Sprachen, Compiler und Maschinen bezogen sein muß,versuchen wir dieses Problem zu vermindern, indem wir eine Untersuchung der Sensibilität der Zeitverbrauchsausdrücke bezüglich der Ausführungszeiten der Algorithmenschritte durchführen. Wir werden aber insbesondere sehen, daß unsere Schlußfolgerungen immer durch Algorithmenmeigenschaften erklärbar sind und daß die Annahme über die Ausführungszeiten keinen wesentlich Einfluß auf diese Schlußfolgerungen hat. Wir sollen die numerischen Ergebnisse nicht als absolute Werte sehen, sie sind vielmehr exemplarisch zu verstehen.

3.2 Annahme über die Ausführungszeiten der Algorithmenschritte

Abb. 3.1 veranschaulicht das Verfahren für die Ableitung der Ausführungszeiten der Algorithmenschritte. Die Schritte sind den Befehlen höherer Programmiersprachen ähnlichen Konstruktionen. (Beispiele sind der Tabelle 3.2 zu entnehmen. Wir haben eine PASCAL-ähnliche Notation für die Algorithmen benutzt). Eine hypothetische Prozessorarchitektur mit dem dazugehörigen Maschinenbefehlssatz wird zugrunde gelegt. Für unsere Arbeit ist eine Annahme über die Ausführungszeit dieser Maschinenbefehle notwendig. Alle vorgekommenen Algorithmenschritte müssen dann "compiliert" werden, so daß wir

aus der Sequenz der Maschinenbefehle und ihrer Ausführungszeiten die Ausführungszeiten der Algorithmenschritte ableiten können.

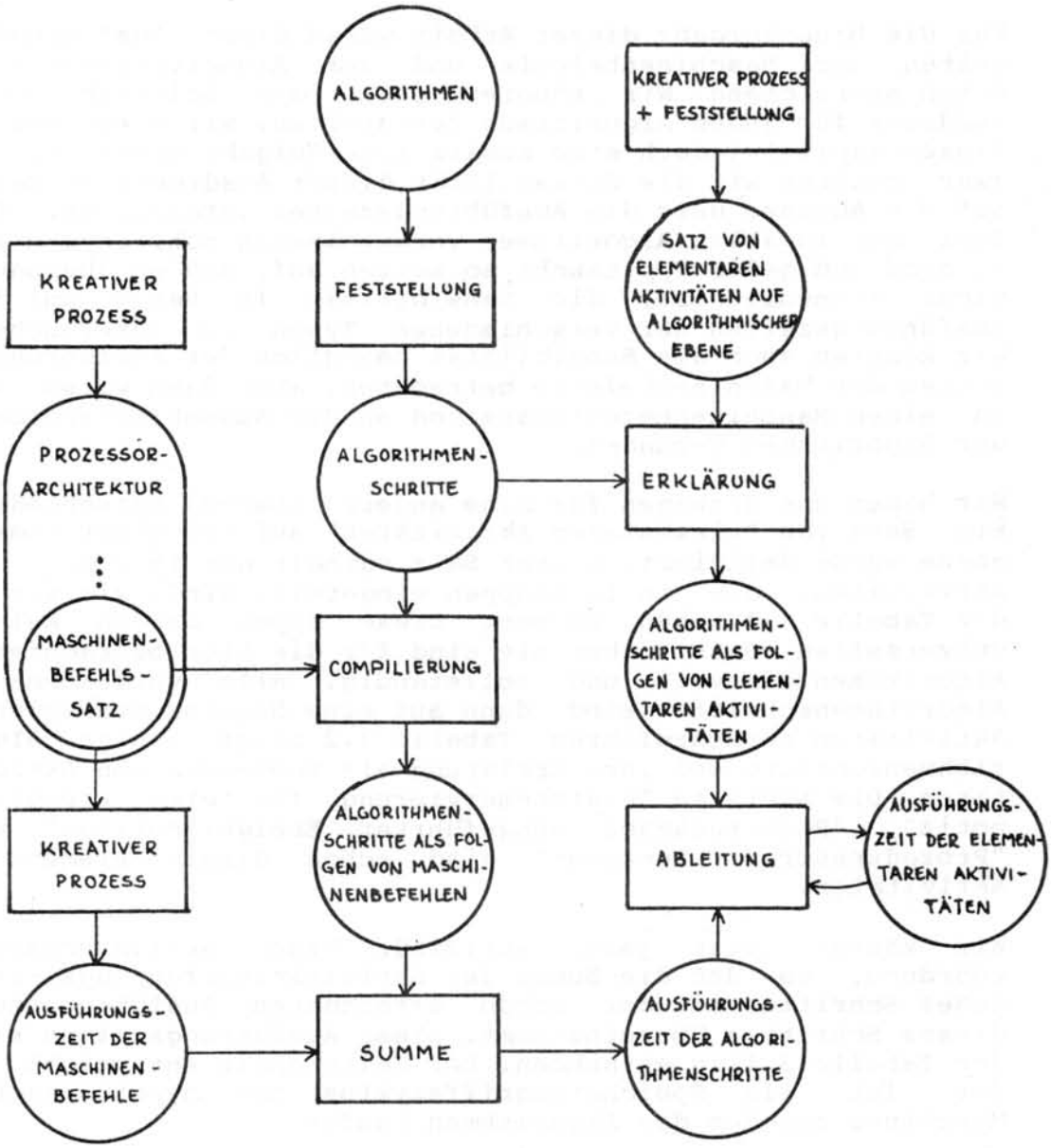


Abb. 3.1 - Ableitung der Ausführungszeit der Algorithmenschritte und der elementaren Aktivitäten der algorithmischen Ebene

Drei Schritte bilden eine Ausnahme zu dieser Vorgehensweise. Damit wir die Implementierung einer dynamischen Speicherverwaltung nicht betrachten mußten, haben wir den Schritten "Platzreservierung für eine Ereignisnotiz" und "Platzrückgabe ausgeführter Ereignisnotizen" direkt Ausführungszeiten

zugeordnet, ohne eine Übersetzung in Maschinenbefehle zu machen. Dasselbe haben wir für den Schritt "Prozeduraufruf + Rückkehr" gemacht, um zu vermeiden, eine Aufrufstruktur definieren zu müssen, die normalerweise Stack, Rettung von Registerinhalten usw. voraussetzt.

Für die Hauptaufgabe dieser Arbeit wären diese Ausführungszeiten der Maschinenbefehle und der Algorithmenschritte schon ausreichend. Wir könnten jetzt den Zeitverbrauchsausdruck für jeden Algorithmus feststellen. Wir haben uns in diesem Kapitel jedoch eine zusätzliche Aufgabe gestellt, und zwar möchten wir die Sensibilität dieser Ausdrücke in Bezug auf die Annahme über die Ausführungszeiten untersuchen. Die Zahl der in allen Algorithmen vorgekommenen Schrittypen ist so groß und jeder Typ taucht so selten auf, daß es überhaupt nicht sinnvoll ist, die Sensibilität in Bezug auf die Ausführungszeiten der verschiedenen Typen zu untersuchen. Wir könnten auch die Sensibilität bezüglich der Ausführungszeiten der Maschinenbefehle betrachten, aber dann wären wir an einen Maschinenbefehlssatz und an die Assemblerversionen der Algorithmen gebunden.

Wir haben uns deswegen für eine andere Lösung entschieden. Ein Satz von "elementaren Aktivitäten" auf der Algorithmenebene wurde definiert. Dieser Satz enthält nur 19 Typen von Aktivitäten, die in 10 Gruppen eingeteilt sind, wie wir in der Tabelle 3.1 sehen können. Diese Typen bilden keinen universellen Satz, aber sie sind für die hier betrachteten Algorithmen sinnvoll und vollständig. Alle vorgekommenen Algorithmenschritte sind dann auf eine Sequenz von solchen Aktivitäten zurückzuführen. Tabelle 3.2 zeigt einige Algorithmenschritte und ihre Erklärung als Sequenzen von Aktivitäten. Die Schritte "Platzreservierung für eine Ereignisnotiz", "Platzrückgabe ausgeführter Ereignisnotizen" und "Prozeduraufruf + Rückkehr" sind schon direkt elementare Aktivitäten.

Wir können jetzt jeder Aktivität eine Ausführungszeit zuordnen, so daß die Summe der Aktivitätsausführungszeiten jedes Schrittes mit der schon errechneten Ausführungszeit dieses Schrittes übereinstimmt. Diese Ausführungszeiten sind der Tabelle 3.1 zu entnehmen. Der Zeiteinheit entspricht in der Tat ein Speicherzugriffszyklus der hypothetischen Maschine, auf der die Algorithmen laufen.

Für die Sensibilitätsuntersuchung bringt diese Lösung Vorteile gegenüber den anderen aufgeführten Möglichkeiten. Wir haben einen relativ kleinen Satz von Aktivitätstypen, die sehr oft in den Algorithmen auftauchen, und brauchen in unserer Analyse die Maschinenebene und die Compilierung nicht mehr zu berücksichtigen.

Wir haben diese Sensibilitätsuntersuchung für zwei Systemklassen durchgeführt, nämlich die Gatterebene, Zeitmodell G1, und die Register-Transferebene, Modell RT2. Sie sind in den entsprechenden Kapiteln zu finden. Wir haben als Ergebnis dieser Untersuchung zwei Grenzfälle gesucht, und

Gruppe	Aktivität	Ausführungs-zeit
0	Zuweisungen	
	Zuweisung des Wertes einer Variable an eine andere Variable	S00 = 3
	Zuweisung einer Konstanten an eine Variable	S01 = 2
1	Arithmetische / Logische Operationen	
	einfache Operation	S10 = 2
	doppelte Operation	S11 = 4
	dreifache Operation	S12 = 6
	Inkrementierung bei einer Indizierung	S13 = 1
2	Inkrementierung einer Variable	S20 = 3
3	Vergleiche	
	Boolescher Test	S30 = 3
	Vergleich einer Variable mit einer Konstanten	S31 = 3
	Vergleich zwischen zwei Variablen	S32 = 4
	CASE	S33 = 5
4	Indizierungen	
	einfache Indizierung	S40 = 2
	doppelte Indizierung (Matrix)	S41 = 6
5	Prozeduraufruf + Rückkehr	S50 = 8
6	Zugriffe auf Felder von Records	
	isolierte Records	S60 = 2
	Records in Arrays organisiert	S61 = 4
7	Zugriff durch einen Pointer	S70 = 2
8	Platzreservierung für eine neue Ereignisnotiz	S80 = 15
9	Platzrückgabe schon ausgeführter Ereignisnotizen	S90 = 15

Tabelle 3.1 - Elementare Aktivitäten auf algorithmischer Ebene

zwar die Sätze von Ausführungszeiten, die am günstigsten für den spezifischen bzw. den Instanzennetzsimulator sind. Jede Ausführungszeit wurde dabei um 50% erhöht oder verringert, je nachdem welche Änderungsrichtung den angesehenen Grenzfall begünstigt. Wir haben diese Veränderungen ohne Rücksicht auf mögliche Beziehungen zwischen den Werten der Ausführungszeiten der elementaren Aktivitäten gemacht. Wir könnten diese Beziehungen erst dann berücksichtigen, wenn wir eine Abbildung von den Ausführungszeiten in Maschinenbefehle betrachteten. Diese Variierung der Ausführungszeiten, so als ob sie völlig unabhängig voneinander wären, bedeutet, daß die neuen erreichten Sätze von Ausführungszeiten tatsächlich Grenzfälle darstellen. Jeder Satz von zusammenhängenden Zeiten ergibt für den spezifischen

Simulator einen Zeitgewinn gegenüber dem Instanzennetz-simulator, der zwischen diesen Grenzfällen unbedingt liegen muß.

Algorithmenschritt	Ausführungszeit	
I1 <- I2 I <- K	S00 S01	Zuweisungen
I1 <- I2 + I3 I1 <- I2 DIV I3 I1 <- I2 + K	S00 + S10 S00 + S10 S01 + S10	Zuweisungen + einfache arithmetische Operationen
I1 <- (I2 + I3) MOD I4 I1 <- I2 + I3 + K I1 <- I2 + (I3 + I4) * I5	S00 + S11 S01 + S11 S01 + S12	Zuweisungen + doppelte und dreifache arithmetische Operationen
I1 <- I1 + 1	S20	Inkrementierung
B = TRUE ? I = K ? I1 = I2 ? I1 =< I2 ?	S30 S31 S32 S32	Vergleiche
I1 <- I2 [I3] I1 [I2] <- I2 [I3] B [I] = TRUE ? I1 >= I2 [I3]	S00 + S40 S00 + 2*S40 S30 + S40 S32 + S40	einfache Indizierung + Verschiedenes
I1 <- I2 [I3,I4]	S00 + S41	doppelte Indizierung
I1 <- I2 [I3 + 1]	S00 + S40 + S13	Indizierung mit Inkrementierung
R.I <- K I1 [I2] <- R.I	S01 + S60 S00 + S40 + S60	Zugriff auf Felder isolierter Records + Verschiedenes
I1 <- R [I2].I3	S00 + S40 + S61	Records in Arrays organisiert
P ↑.I <- K I1 <- P [I2] ↑.I3 I1 <- P ↑.I2 ↑.I3	S01 + S60 + S70 S00 + S40 + S60 + S70 S00 + 2*S60 + 2*S70	Zugriff durch einen Pointer + Verschiedenes

Bemerkungen: In = Integer-Variable, K = Konstante, B = boolesche Variable, R = Record aus Feldern von Integer, P = Pointer

Tabelle 3.2 - Erklärung einiger Algorithmenschritte als Folgen von elementaren Aktivitäten

Die hypothetische Prozessorarchitektur mit dazugehörigen Befehlssatz und die Compilierung der Algorithmenschritte in Sequenzen von Maschinenbefehlen sind im Rahmen dieser Arbeit unwichtig. Von Interesse sind hier nur die daraus resultie-

renden Ausführungszeiten der Algorithmenschritte und ihre Erklärung als eine Summe von Ausführungszeiten elementarer Aktivitäten.

3.3 Voraussetzungen für den Vergleich

Wir stellen in diesem Abschnitt alle solchen Voraussetzungen über die Algorithmen zusammen, die die Vergleichsergebnisse brauchbar und sinnvoll machen. Wir haben die Simulationszeit als Effizienz- und Vergleichsmaßstab gewählt, so daß wir alle Algorithmen mit Hinsicht auf einen minimalen Zeitverbrauch während der Simulation entwickeln sollen.

Unter **Simulation** verstehen wir hier nur eine solche Phase, in der eine schon vorhandene und ausgetestete für den Simulator geeignete Angabe, die dem zu untersuchenden System entspricht, simuliert wird. Wir schließen damit alle anderen Phasen aus, die dazu dienen, diese Systemangabe in der geeigneten Form zu erzeugen, auszutesten und zum Laufen zu bringen. Diese Phasen umfassen unter anderen Aktivitäten die Definition der Netzverschaltung und der Instanzenprozeduren, die Übersetzung dieser Systemangaben in eine für den Simulator geeignete interne Form, und die Systeminitialisierung (z.B. Vorbesetzungen von Anschluß- und Zustandsvariablen). Wir haben nicht nur auf die Betrachtung der Durchführung dieser Phasen verzichtet, sondern auch auf alle Konstruktionen, die die Algorithmen beeinflussen können und nur dazu dienen, diese Phasen zu unterstützen. Unter diesen Konstruktionen verstehen wir auch Strukturen höherer Programmiersprachen, die z.B. Lesbarkeit, Austesten und Zuverlässigkeit der Algorithmen bzw. der Instanzenprozeduren ermöglichen oder verbessern. Wie in Kapitel 1 schon erwähnt, müssen wir auch auf jegliche Benutzerfreundlichkeit bzw. Konstruktionen in den Algorithmen, die der Unterstützung einer solchen Benutzerfreundlichkeit dienen, verzichten. Erst unter allen diesen Umständen können wir uns darauf verlassen, daß die Simulationsalgorithmen sich auf die wesentlichen und notwendigen Eigenschaften der Instanzennetze und der untersuchten Systemklassen beschränkt haben und daß die Vergleichsergebnisse sinnvoll sind.

Der Vergleich sollte auch möglichst unabhängig von der Algorithmenimplementierung sein, d.h. von Implementierungssprache, Host-Computer und Compiler. Wie im letzten Abschnitt gesehen, ist diese Voraussetzung nicht ganz zu erfüllen, weil wir Annahmen über die Ausführungszeiten der Algorithmenschritte machen müssen. Außer diesen Annahmen bleiben aber alle anderen Implementierungsdetails unberücksichtigt, wie z.B. die Verträglichkeit zwischen Wortlängen im Simulator und im Host-Computer.

In Simulationssystemen, die dem Benutzer ein Bausteinrepertoire zur Verfügung stellen, sind drei Programmierungsebenen zu erkennen. Auf der ersten Ebene ist der Programmierer des eigentlichen Simulationskerns, d.h. wer den Koordinator schreibt. Der Repertoire-Implementierer sieht den Koordi-

nator und schreibt die Instanzenprozeduren gemäß der Aufgabenverteilung zwischen Koordinator und Instanzen. Dem Endbenutzer steht das Instanzenrepertoire zur Verfügung, er muß dem Simulationssystem nur eine Netzverschaltung angeben. Er weiß nicht, wie das Simulationssystem die Wechselwirkungen zwischen den Instanzen implementiert, und selbst den Koordinator sieht er nicht. In einem Simulationssystem, das nur der Simulation einer bestimmten Systemklasse dient, ist die Trennung zwischen den zwei unteren Ebenen nicht mehr vorhanden. Normalerweise schreibt der gleiche Programmierer den Simulationskern und die repertoirefähigen Instanzenprozeduren, so daß er mit Kenntnis der Datenstruktur des Simulationssystems diese Prozeduren in Hinblick auf maximale Effizienz schreiben kann. Dies ist für allgemeine Simulationssysteme nicht der Fall, weil für sie der Repertoire-Implementierer mit Sicherheit die Instanzenprozeduren ohne diese Kenntnis schreibt. Er weiß nicht, wo die Werte der Anschluß- und Zustandsvariablen abgespeichert sind. Die Instanz greift auf formale Parameter zu, die der Koordinator bei dem Aufruf durch die realen Variablen ersetzt. Gegenüber klassenspezifischen Simulatoren bringt diese Vorgehensweise zwei deutliche Nachteile in Bezug auf die Effizienz: a) Der Koordinator weiß nicht, welche Parameter die Instanz in einer Aktion eigentlich braucht, so daß er eine komplette Parameterübergabe durchführen muß, auch wenn die Instanz anhand ihrer Struktur nur auf einen Anteil der Parameter zugreifen muß; und b) der Parameterübergabemechanismus selbst verbraucht Zeit.

Dieser Effizienzverlust ist nicht dem Wesen der Simulationsalgorithmen zuzuschreiben, sondern den Anwendungszielen der Simulationssysteme. Ein Simulationssystem wäre nicht benutzerfreundlich, wenn der Benutzer seine interne Datenstruktur kennen müßte. Wir erkennen hier einen Kompromiß zwischen Effizienz und Benutzerkomfort. Da wir aber nur die wesentlichen Eigenschaften der Algorithmen untersuchen wollen, müssen wir diesen Effizienzverlust ausgleichen, indem wir dem Programmierer der Instanzenprozeduren ermöglichen, die Prozeduren mit Kenntnis der Datenstruktur der Abb. 2.5 zu schreiben. Die Instanzen greifen auf die Anschluß- und Zustandsvariablen direkt durch

SW [PARAMLISTE [PARAMVERWEIS[i] + offset]]

zu (siehe Abschnitt 2.7.3). Eine Parameterübergabe durch den Koordinator ist nicht notwendig.

Der Algorithmenschritt "Prozeduraufruf + Rückkehr" braucht dann die Parameterübergabe nicht zu berücksichtigen. Der Prozeduraufruf ist nur in der Instanzennetzsimulation notwendig, weil die spezifischen Simulatoren immer ein festes Repertoire zugrunde legen, so daß wir die Durchführung der Instanzenaktionen in den Rumpf des Simulationsalgorithmus verlagern können.

3.4 Zeitverbrauch des Koordinators eines Instanzennetzsimulators

Da wir den Algorithmus für den Koordinator unseres Instanzennetzsimulators und die Ausführungszeiten der einzelnen Algorithmenschritte schon haben, können wir dieses Kapitel mit der Berechnung des Zeitverbrauchs für den ganzen Koordinator abschließen. Der Koordinator ist in 10 Sektionen aufgeteilt, wie wir in der Abb. 2.10 erkennen. Jede Sektion ist entweder eine Sequenz von Schritten ohne Verzweigungen oder eine Schleife, so daß Sektionen immer vollständig ausgeführt werden. Jeder Sektion i ist eine Ausführungszeit Ti zugeordnet, die aus den elementaren Aktivitäten, die die Sektion bilden, leicht auszuwerten ist. Tabelle 3.3 stellt diese Zeiten Ti zusammen. Um den Zeitverbrauch des Koordinators während der Simulation eines bestimmten Systems zu berechnen, müssen wir wissen, wie oft jede Sektion durchlaufen wird. Der Zeitverbrauchsausdruck für den Koordinator hat dann die Form

F1*T1 + F2*T2 + ... + F10*T10,

wobei F1, F2,... Funktionen der Systemparameter sind.

Sektion-ausführungszeit	Beschreibung der Sektion
T1 = 57	Hauptschleife (= Verarbeitung der Rufliste)
T2 = 95 (1)	Eintragung einer Ereignisnotiz in die Ereignisliste
T3 = 15	Lineare Suche des Eintragungsplatzes einer Notiz im Zielintervall
T4 = 13	Zugriff auf den Anfang der Teil-UMGLISTE einer Instanz
T5 = 16	Holen des Namens einer Instanz aus der UMGLISTE
T6 = 15	Eintragung des Namens einer Instanz in die Rufliste aufgrund einer Umgebungsänderung
T7 = 20	Zeitfortschaltung
T8 = 116 (2)	Entfernung und neue Eintragung einer Scheinereignisnotiz
T9 = 15	Lineare Suche des Eintragungsplatzes einer Scheinnotiz im Zielintervall
T10 = 62	Holen des Namens einer Instanz aus der Ereignisliste und Eintragung in die Rufliste

Bemerkungen:

(1) Unter der Annahme, daß I < MAX immer (Durchschnittsfehler: 4 Zeiteinheiten jeder MAX Eintragungen)

(2) Unter der Annahme, daß K > 0 immer (sonst haben wir ein Deadlock)

Tabelle 3.3 - Ausführungszeiten für den Koordinator des Instanzennetzsimulators

Um die Parameterzugriffszeit von dem sonstigen Zeitverbrauch zu isolieren, was für die spätere Analyse des Zeitgewinns eines spezifischen Simulators gegenüber dem Instanzennetzsimulator wichtig ist, führen wir hier eine Grundzugriffszeit Ta ein, die so definiert ist:

Ta = S13 + 2*S40 = 5.

In einem Algorithmenschritt, der einen Zugriff auf die Datenstruktur enthält, können wir dann die Zugriffszeit erkennen und abkapseln. Beispiele dafür sind im folgenden aufgeführt, wobei

P = PARAMVERWEIS[Instanzenname].

Algorithmenschritt	Ausführungszeit
I <- SW [PARAMLISTE [P + 1]]	S00 + 2*S40 + S13 = S00 + Ta
I <- SW [PARAMLISTE [P + 1]] AND SW [PARAMLISTE [P + 3]]	S00 + S10 + 4*S40 + 2*S13 = = S00 + S10 + 2*Ta

4. Gatterebenen-Simulation

Die Anzahl der in der Literatur beschriebenen Simulatoren für die Gatterebene ist sehr groß (siehe z.B. /DAC/). Jeder Simulator hat seine eigene Modellvorstellung darüber, was die Gatterebene sei. Außerdem dienen die Simulatoren unterschiedlichen Zwecken, von einer reinen Überprüfung der Logik bis zum Test komplizierter Zeitbedingungen. Deswegen ist es sehr schwierig, von einem typischen Simulator zu sprechen. Einige allgemeine Eigenschaften und Strukturen sind trotzdem erkennbar /SZY75/ /BRE76/. Wir stellen uns dann vor, einen Simulationskern zu definieren, der möglichst allgemeine und wenige einfache Eigenschaften besitzt. Diesen Kern bauen wir durch zwei verschiedene Zeitmodelle aus, die keine große Komplexität für den Simulationsalgorithmus bringen aber trotzdem stellvertretend für alle Modelle stehen. Das Kernmodell nennen wir G und die Zeitmodelle G1 und G2.

Wie in Kapitel 3 erwähnt, wollen wir uns mit der Verträglichkeit zwischen der Implementierung des Simulators und dem Host-Computer nicht beschäftigen. Da die Gattersimulatoren nur ein Bit breite Signale verarbeiten (oder 2 Bit breite, wenn dreiwertige Logik verwendet wird), werden normalerweise viele Signalwerte gleichzeitig in einem Wort des Host-Computers abgespeichert. Ein typisches Beispiel dafür ist die parallele Fehlersimulation /THO75/. Wir werden uns hier verhalten, so als gäbe es ein Wort für jedes Signal. Der damit implizierte größere Speicherverbrauch ist für uns im Rahmen dieser Arbeit nicht relevant.

4.1 Beschreibung des Kernmodells eines Gatterebenen-Systems

Ein Gatterebenen-System besteht auschließlich aus Gattern, mit denen reine Verknüpfungslogik verbunden ist. Auf dieser untersten Ebene der diskreten Hardware-Modellierung stellen wir Flip-Flops immer durch ihre äquivalenten Gatterstrukturen dar. Gatter sind Bausteine, die beliebig viele Eingänge aber einen einzigen Ausgang haben.

Jedem Gatter ordnen wir eine mit einem Wert größer als Null besetzte Verzögerung zu. Wir berücksichtigen damit die Klasse der "Zero-Delay"-Simulatoren in dieser Arbeit nicht. Verzögerungen sind immer ganze Zahlen, die in einem relativ kleinen Zeitintervall fallen. Dies entspricht der Realität, wenn das zu simulierende System in einer einzigen Technologie aufgebaut wird. (Wir nehmen den g.g.T. aller im System vorhandenen Verzögerungen als Simulationszeitschritt. Im folgenden gehen wir so vor, als ob diesem Schritt eine Zeiteinheit zugeordnet wäre und alle Verzögerungen in Bezug auf ihn normalisiert worden wären.)

Signale haben einen diskreten Wertebereich, entweder (0,1) für zweiwertige oder (0,U,1) für dreiwertige Logik.

Wir betrachten Signaländerungen als augenblickliche Übergänge. Übergangszeiten können wir durch dreiwertige Logik modellieren (siehe Zeitmodell G2). Die Gleichzeitigkeit der Signaländerungen bezieht sich auf den gewählten kleinsten Zeitschritt. Hazardpulse, die im realen System wegen Zeitunterschieden, die kleiner als dieser Zeitschritt sind, auftauchen können, werden hier nicht modelliert. Wir können diese Fälle in der Simulation erst dann behandeln, wenn wir die Zeitunterschiede, die die Hazardpulse verursachen, bewußt als einen Zeitschritt oder ein Vielfaches eines Zeitschrittes modellieren.

4.2 Der Simulationskern für den Gattersimulator

Aus dem vorgestellten Modell eines Gatterebenen-Systems können wir einen Algorithmuskern für den Gattersimulator ableiten, den wir später durch die Einführung eines der Zeitmodelle vervollständigen. Ausschlaggebend für diese Kerndefinition ist die Zeitbedingung, daß der Simulator Signaländerungen, die für einen bestimmten Modellzeitpunkt geplant sind, wirklich als gleichzeitige Übergänge behandeln muß, d.h., daß alle Änderungen schon stattgefunden haben müssen, bevor irgendwelche Auswirkungen dieser Änderungen sich bemerkbar machen. Diese Bedingung erfüllen wir einfach dadurch, daß wir jede Systemüberführung in zwei Phasen unterteilen, die der Änderung der Signale und dem Durchlauf der Änderungen durch das System entsprechen.

Signaländerungen vermerken wir als Ereignisnotizen in einer Ereignisliste. Grundsätzlich enthält eine Notiz den Namen eines Gatters und den logischen Wert, den der Ausgang dieses Gatters nach dem Ereigniseintreffen übernehmen muß. Eine Notiz kann deswegen als eine Anweisung verstanden werden, die der Gattersimulator unmittelbar ausführt, wenn der entsprechende Modellzeitpunkt eingetreten ist.

Im Abschnitt 2.7.4 haben wir die Ereignisliste für den Instanzennetzsimulator im Hinblick auf die Optimierung dreier Operationen realisiert. Dieselben Operationen sind auch hier notwendig, nämlich die Zeitfortschaltung, die Suche aller Ereignisnotizen für die aktuelle Modellzeit und die Eintragung neuer Notizen. Es gibt jedoch jetzt einen wesentlichen Unterschied gegenüber der Instanzennetzsimulation, der die notwendige Organisation der Ereignisliste entscheidend beeinflußt. Wir kennen nämlich jetzt wichtige Eigenschaften der Zeitverteilung der Ereignisse. Die Verzögerungen sind ganze Zahlen und fallen in einer kleinen und begrenzten Zeitspanne ZSP. Es wird außerdem in der Literatur behauptet, daß für die meisten zu simulierenden Systeme der durchschnittliche Prozentsatz der zu jedem Zeitpunkt aktiven Elemente (Elemente, die Ausgangsänderungen haben) um 1 % /ULR69/ liege. Da typische zu simulierende Systeme weit über 100 Elemente haben, ist die durchschnittliche Anzahl der Ereignisse je Zeitschritt wesentlich größer als 1. Eine besondere Organisation für die Ereignisliste, die "Time-Mapping"-Organisation, ist wegen

der obengenannten Eigenschaften möglich /ULR69/ /SZY75/.

Wir teilen Zeitfortschaltungsmechanismen in "**Next-Event**"- und "**Fixed-Step**"-Mechanismen ein /NAN71/. Ein "Next-Event"-Mechanismus besteht, wenn wir die Modellzeit immer bis zum Zeitpunkt des nächsten Ereignisses vorrücken. "Fixed-Step"-Mechanismen dagegen erhöhen die Modellzeit immer um einen festen Schritt. Der erste Typ ist für Zeitverteilungen geeignet, die entweder ungleichmäßig sind, oder deren Zeitwerte reale Zahlen sind, während der zweite Typ sich für Zeitverteilungen eignet, deren Zeitwerte immer ganzzahlig sind und die Ereignisse immer in allen Schritten haben. Der IL-Algorithmus, den wir für den Instanzennetzsimulator verwendet haben, hat einen "Next-Event"-Mechanismus, während der Time-Mapping (TM)-Algorithmus mit einem "Fixed-Step"-Mechanismus versehen ist.

Die Ereignisliste für den TM-Algorithmus besteht aus Teillisten, die nur vorwärts durch das Feld NEXT der Ereignisnotizen verkettet sind. Für jeden Zeitschritt ("Zeitschlitz" genannt) des Zeitintervalls ZSP gibt es eine Teilliste, wie die Abb. 4.1 zeigt. Auf jede Teilliste wird aus einem Array TIMEWHEEL verwiesen, und zwar durch zwei Felder FIRST, für die erste Ereignisnotiz in der Liste, und LAST, für die letzte. Die aktuelle Modellzeit CURRENT verwenden wir als Index für die TIMEWHEEL, die zyklisch organisiert ist. Die drei Operationen auf der Ereignisliste lassen sich dann folgendermaßen realisieren:

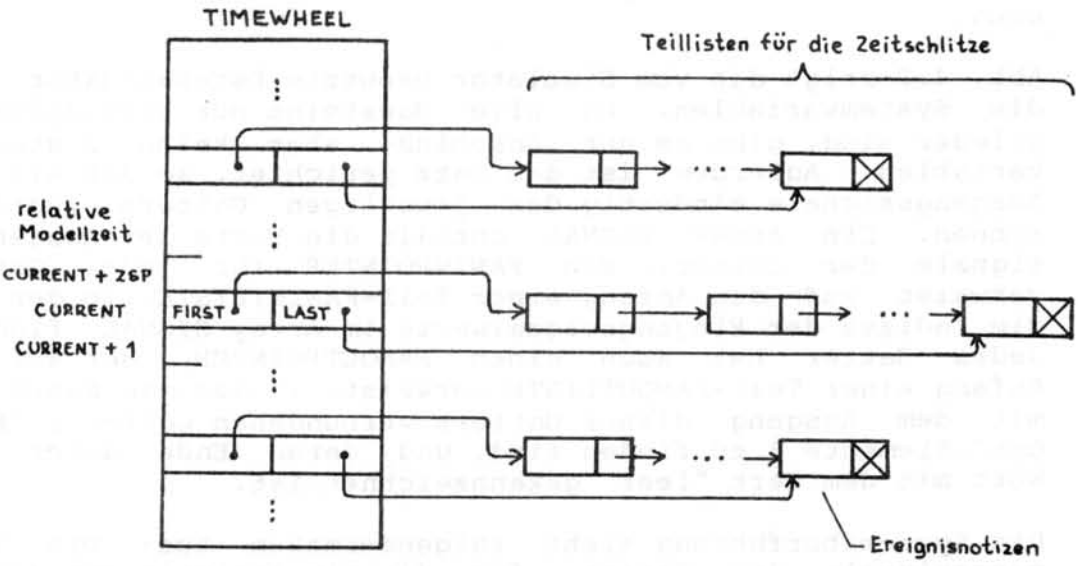


Abb. 4.1 - Datenstruktur für die Ereignisliste nach dem TM-Algorithmus

-Zeitfortschaltung
CURRENT wird inkrementiert.

-Suche der Notizen für die aktuelle Modellzeit
TIMEWHEEL (CURRENT) . FIRST (=EP) verweist auf die erste Notiz, zu den folgenden Notizen haben wir Zugang durch EP↑.NEXT, bis NEXT auf keine Notiz mehr zeigt.

-Eintragung neuer Notizen
Sei DELAY die Gatterverzögerung. Die absolute Planzeit APZ einer Notiz für dieses Gatter ist CURRENT + DELAY. Die relative Planzeit APZ MOD ZSP in der Ereignisliste ist gleichzeitig der Index für den gewünschten Zeitschlitz in der TIMEWHEEL, d.h., es gibt eine direkte Abbildung von Planzeit auf Eintragungsplatz, deswegen der Name "Time-Mapping". Die neue Notiz tragen wir am Ende der Teilliste dieses Zeitschlitzes mit Hilfe des Verweises LAST ein. Im Gegensatz zum IL-Algorithmus ist keine lineare Absuche für die Eintragung erforderlich.

Da für beide betrachtete Zeitmodelle notwendig ist, daß die Planung von Ereignissen rückgängig gemacht werden kann (die Gründe dafür besprechen wir in den jeweiligen Abschnitten), ist jede Ereignisnotiz mit einem booleschen Feld CANCEL versehen. Die Notiz wird nicht physikalisch aus der Liste entfernt, sie wird vielmehr als ungültig erklärt. Unter **Löschen** einer Notiz verstehen wir in Bezug auf den Gattersimulator das Setzen dieser Variable auf den Wert "TRUE".

Ein Mechanismus für dynamische Speicherverwaltung ist hier wie für den Instanzennetzsimulator nötig, da die Anzahl der Ereignisse systemabhängig ist und sehr unterschiedlich sein kann.

Abb. 4.2 zeigt die vom Simulator benutzte Datenstruktur für die Systemvariablen. Da alle Bausteine nur Verknüpfungsglieder sind, gibt es nur Anschluß- aber keine Zustandsvariablen. Außerdem ist das Netz gerichtet, so daß wir die Ausgangssignale eindeutig den jeweiligen Gattern zuordnen können. Ein Array SIGNAL enthält die Werte der Ausgangssignale der Gatter. Ein FANINPOINTER für jedes Gatter verweist auf den Anfang einer Teil-FANINILISTE, in der wir die Indizes der Eingangssignalwerte im Array SIGNAL finden. Jedes Gatter hat auch einen FANOUTPOINTER, der auf den Anfang einer Teil-FANOUTLISTE verweist, in der die Namen der mit dem Ausgang dieses Gatters verbundenen Gatter ("**Fan-Out**"-Elemente) zu finden sind, und deren Ende durch ein Wort mit dem Wert "leer" gekennzeichnet ist.

Die Systemüberführung sieht folgendermaßen aus: Die Ausgangssignale der Gatter, für die ein Ereignis für diesen Zeitpunkt geplant ist, werden aktualisiert. Die Namen aller Gatter, die mit diesen Ausgängen verbunden sind, bringen wir in eine Rufliste. Diese hat eine FIFO-Organisation, aber jede andere Organisation wäre auch möglich. Wir werten die logische Funktion aller Gatter aus, deren Namen in der Rufliste sind, und aufgrund des neu berechneten Ausgangswertes, des aktuellen Wertes, der Gatterverzögerung und des Zeitmodells stellen wir die Auswirkung dieser Auswertung

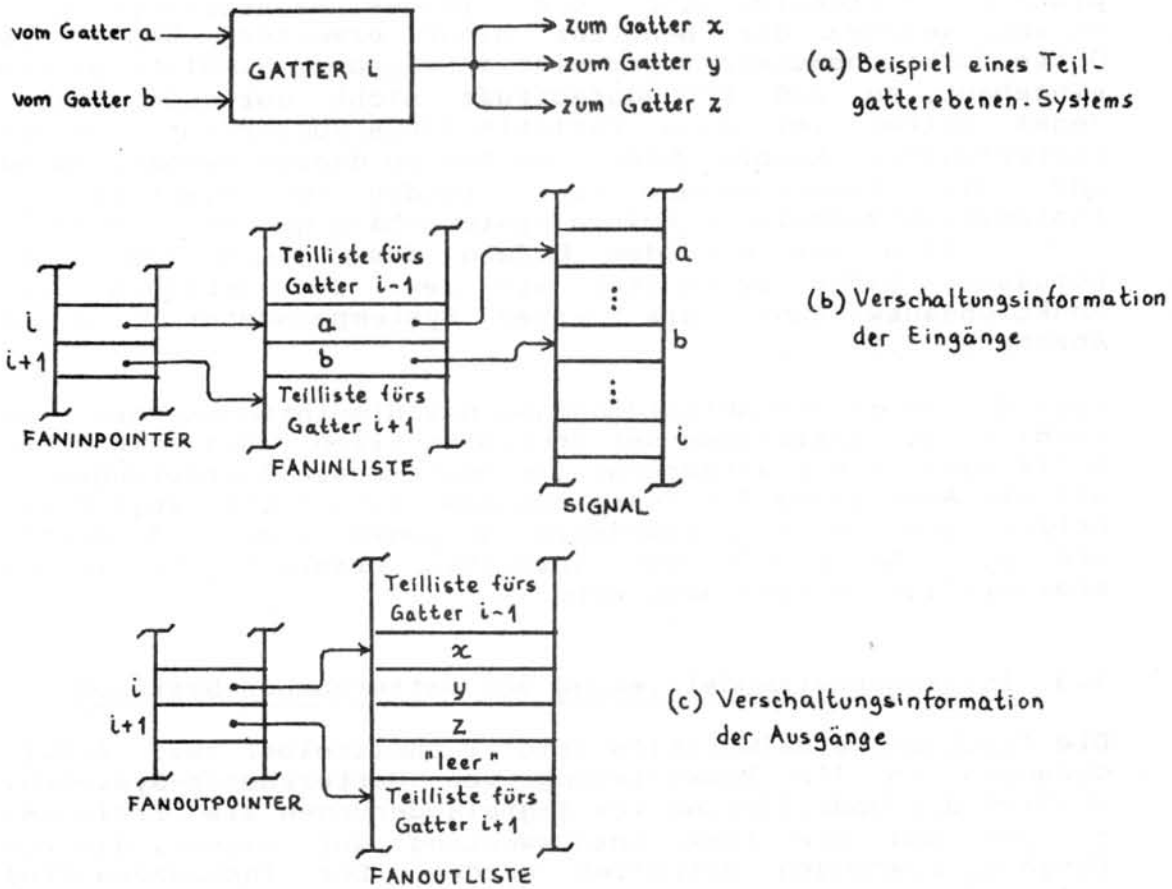


Abb. 4.2 - Datenstruktur für die Systemverschaltung im Gattersimulator

fest. Entweder ist keine Aktivität erforderlich, oder wir müssen eine neue Ereignisnotiz eintragen oder eine vorher eingetragene Notiz löschen.

Um die Funktion eines Gatters nur einmal zu jedem Modellzeitpunkt auszuwerten und nur eine Notiz für eine Ausgangsänderung einzutragen, wenn mehrere gleichzeitige Änderungen an den Eingängen dieses Gatters beobachtbar sind, ordnen wir jedem Gatter eine boolesche Variable FLAG zu, die Namensverdoppelungen in der Rufliste zu vermeiden hilft, wie die gleichnamige Variable im Instanzennetzsimulator. Wir begrenzen damit die Größe der Rufliste automatisch auf die Anzahl der Gatter im System, weil, wegen der Gatterverzögerungen, der neu berechnete logische Wert nur zu einem zukünftigen Zeitpunkt dem Gatterausgang zugewiesen wird und nie an einem anderen Gattereingang zum selben Zeitpunkt der Funktionsauswertung beobachtbar ist. Wir werden deswegen nie als Folge einer solchen Auswertung die Namen anderer Gatter in die Rufliste bringen müssen.

Wie im Abschnitt 3.3 schon diskutiert wurde, setzt ein typischer Gattersimulator ein festes Bausteinrepertoire voraus, welches der Benutzer nicht erweitern kann. Die Gatterfunktionsauswertungen sind deswegen im Simulationskern eingebaut, so daß Prozeduraufrufe nicht notwendig sind. Jedem Gatter ist eine Variable TYPNR zugeordnet, die die Gatterfunktion kennzeichnet. Das Testen dieser Kennzeichnung und die Funktionsauswertung bilden im Gegensatz zum Instanzennetzsimulator keinen systemabhängigen Programmteil. Wir werden uns mit dem Umfang des Repertoires nicht befassen. Dafür betrachten wir den Zeitverbrauch der Funktionsauswertung als einen Systemparameter (siehe Abschnitt 4.6.1).

Abb. 4.3 zeigt das Ablaufdiagramm des Simulationskernes. Er enthält die **Ausführung** der Ereignisnotizen (d.h. die einer Notiz unmittelbar entsprechenden Systemzustandsänderungen) und die Auswirkung der Funktionsauswertung nicht, weil diese Aktivitäten von den Zeitmodellen abhängen. Abb. 4.4 zeigt, wie eine Notiz mit der absoluten Planzeit APZ in die Ereignisliste eingetragen wird.

4.3 Instanzennetzmodellierung von Gatterebenen-Systemen

Die Instanzennetzsimulation erfüllt unmittelbar zwei Anforderungen an die Modellierung von Gatterebenen-Systemen, nämlich die Modellierung von Signaländerungen als Ereignisnotizen und die Funktionsauswertung der Gatter, die von Eingangsänderungen betroffen werden. Der Instanzenaufruf aufgrund von Zeitmeldungen ermöglicht dann, daß eine Gatterinstanz ihre Anschluß- und Zustandsvariablen aufgrund des Eintreffens eines Ereignisses ändert. Der Aufruf aufgrund von Umgebungsänderungen entspricht dem Signaländerungsdurchlauf.

Zwei weiteren Anforderungen, nämlich die Gleichzeitigkeit der Signaländerungen und die Erklärung von Ereignissen als ungültig, erfüllt jedoch der Koordinator des Instanzennetzsimulators nicht. Der Repertoire-Implementierer muß sie durch angemessene Modellierung in die Instanzenfunktionen verlagern. Die Tatsache, daß wir die Signaländerungen nicht automatisch als gleichzeitig modellieren können, ist auf die Instanzenaufrufreihenfolge zurückzuführen. Der Koordinator könnte eine Instanz aufgrund einer Umgebungsänderung schon in Aktion bringen, noch bevor er alle Instanzen aufgrund von Zeitmeldungen aufgerufen hat. Wir sind nur sicher, daß die erste Instanz zu einem neuen Modellzeitpunkt aufgrund einer Zeitmeldung aufgerufen wird (siehe Abschnitt 2.7.5).

Die Aufteilung der Systemüberführung in zwei Phasen, eine für die Signaländerungen, die andere für den Änderungsdurchlauf, wie im Gattersimulator, läßt sich in der Instanzennetzmodellierung nur realisieren, indem wir eine Systemmodellzeit (bezogen auf das Modell des Gatterebenen-Systems) durch zwei getrennte Simulationsmodellzeiten (bezogen auf die Zeitfortschaltung im Instanzennetzsimulator)

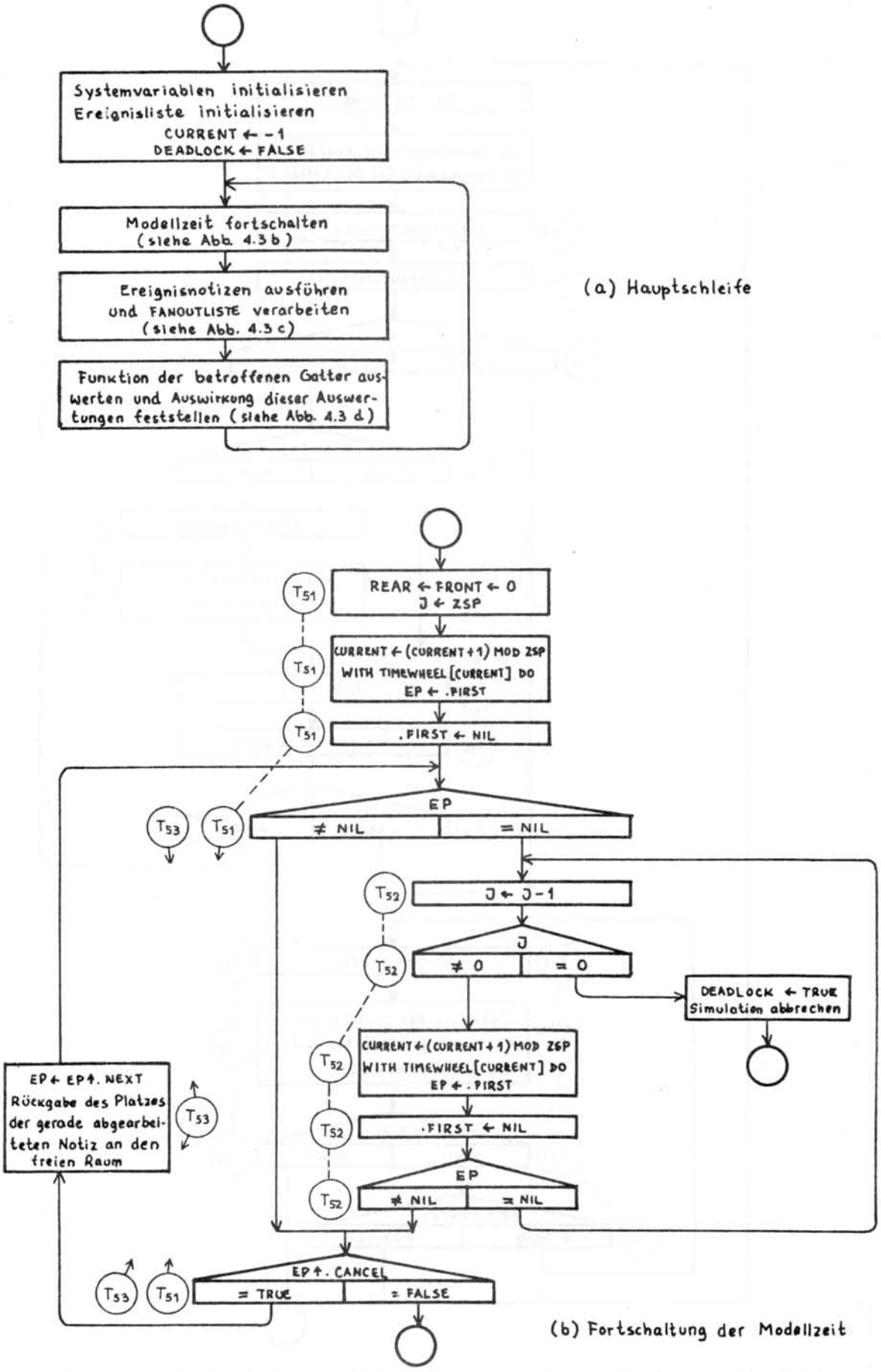
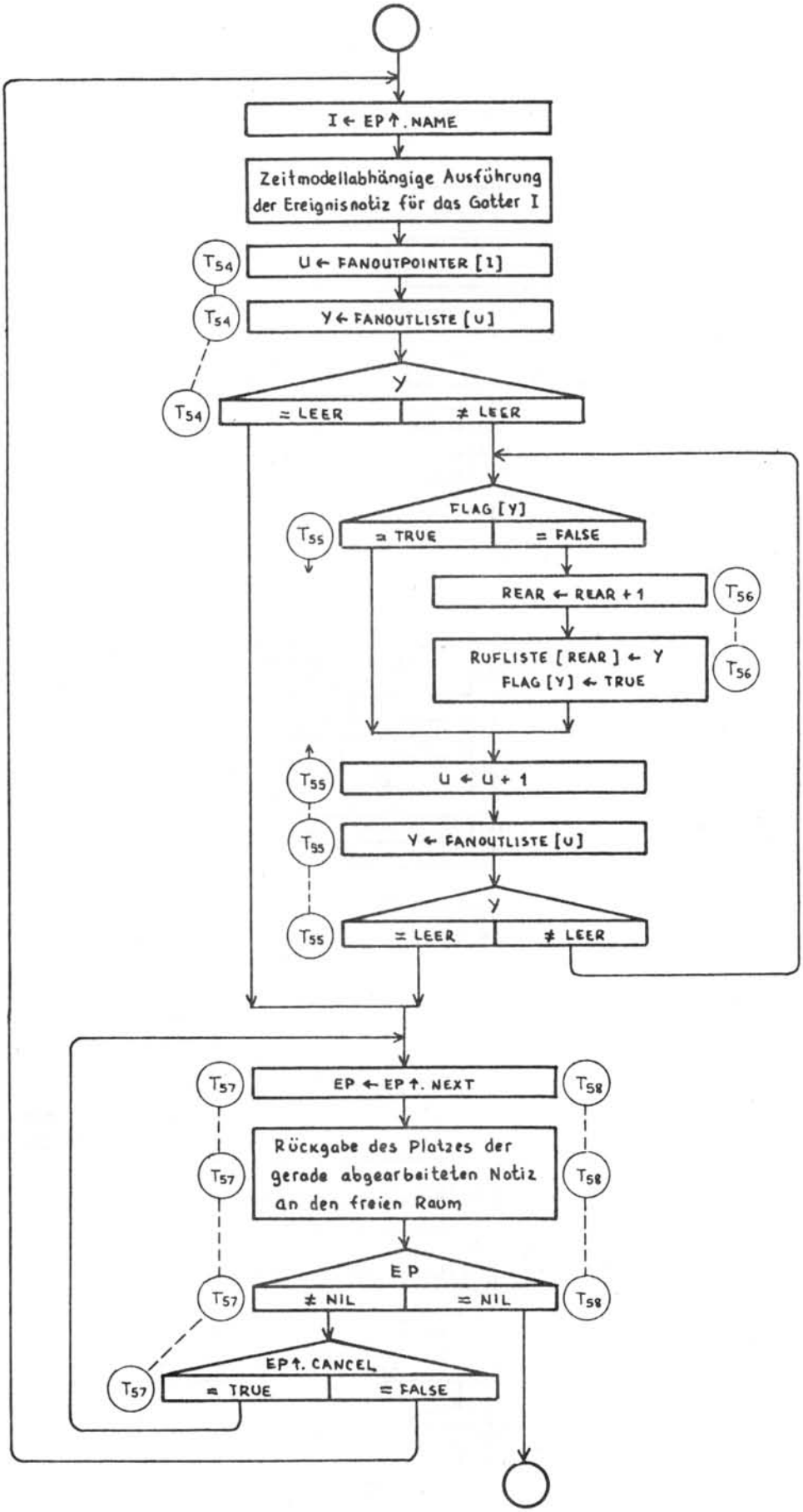
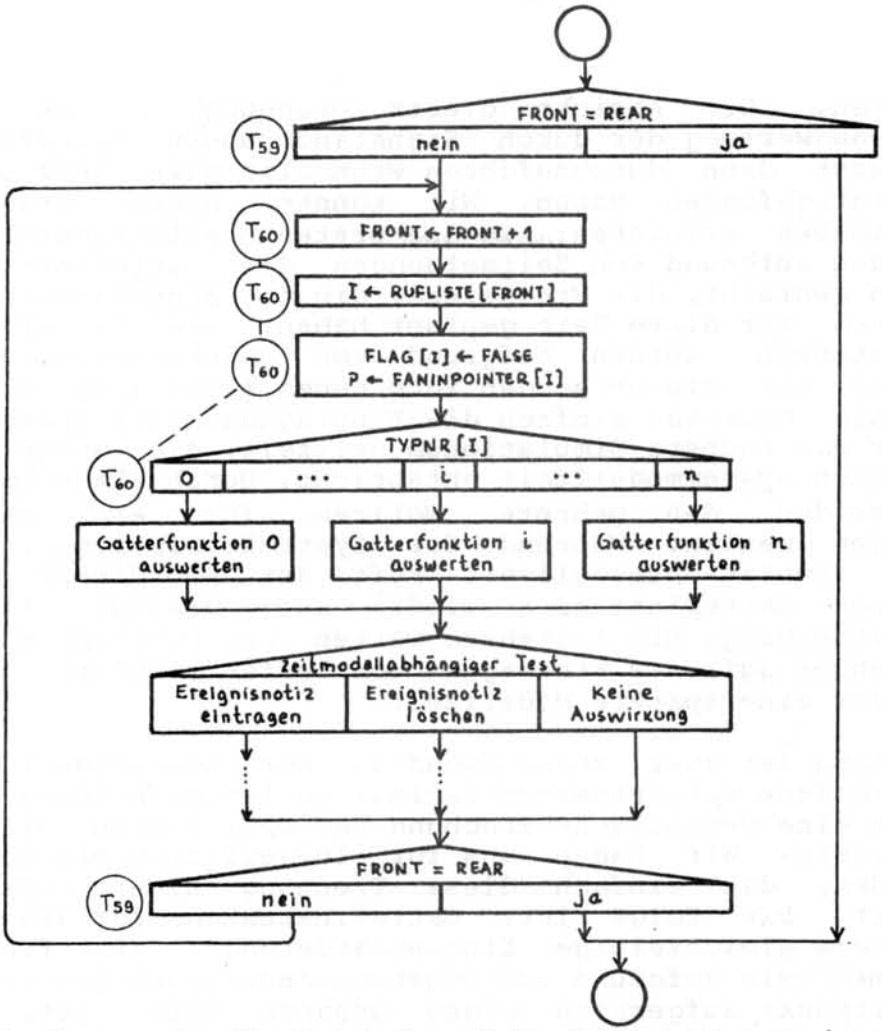


Abb. 4.3 - Kern eines Gattersimulators (Fortsetzung auf den nächsten Seiten)



(c) Ausführung der Ereignisnotizen eines Zeitschlitzes und Verarbeitung der FANOUTLISTE

Abb. 4.3 - Kern eines Gattersimulators (Fortsetzung auf der nächsten Seite)



(d) Funktionsauswertung für die betroffenen Gatter und Feststellung der Auswirkung dieser Auswertungen

Abb. 4.3 - Kern eines Gattersimulators

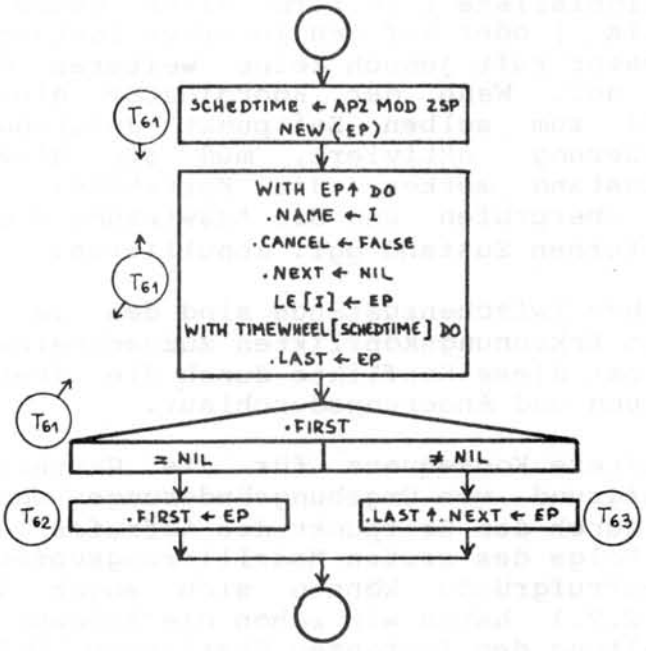


Abb. 4.4 - Eintragung einer Ereignisnotiz mit absoluter Planzeit APZ für das Gatter I in die Ereignisliste des Gattersimulators

modellieren. Die Absicht dieser Trennung ist es, die Funktionsauswertung der durch Signaländerungen betroffenen Gatter erst dann durchzuführen wenn alle diese Änderungen schon stattgefunden haben. Wir könnten diese Trennung folgendermaßen erreichen: In der ersten Simulationsmodellzeit werden aufgrund von Zeitmeldungen die Gatterinstanzen in Aktion gebracht, die Ereignisse, die den Signaländerungen entsprechen, für diese Zeit geplant haben. Die betroffenen Gatterinstanzen werden aufgrund von Umgebungsänderungen aufgerufen, aber sie berechnen ihre neuen Ausgangswerte noch nicht. Sie bewirken einfach die Eintragung einer Ereignisnotiz für die nächste Simulationsmodellzeit, die in der Tat keiner neuen Systemmodellzeit entspricht. Durch Flags können wir vermeiden, daß mehrere Notizen für ein Gatter eingetragen werden. Während der Systemüberführung in der nächsten Simulationsmodellzeit ruft der Koordinator die betroffenen Gatterinstanzen wieder auf, diesmal aufgrund einer Zeitmeldung. Die Instanzen werten ihre Funktionen aus und verlangen ggf. die Eintragung von "effektiven" Ereignisnotizen für eine spätere Modellzeit.

Diese Lösung ist aber zeitaufwendig. Der Koordinator muß jede betroffene Gatterinstanz zweimal in Aktion bringen, und wir haben eine wesentliche Erhöhung der Operationen auf der Ereignisliste. Wir haben uns für eine effizientere Lösung entschieden, die einfach diese Trennung nicht explizit modelliert. Die Folge ist: Gatterinstanzen, für die zwei oder mehrere gleichzeitige Eingangsänderungen stattfinden, können mehrmals aufgrund von Umgebungsänderungen zum selben Modellzeitpunkt aufgerufen werden. Ursache dafür ist, daß der Koordinator Instanzen aufgrund von Umgebungsänderungen aufrufen kann, noch bevor er alle Instanzen aufgrund von Zeitmeldungen in Aktion gebracht hat. Da alle Gatter mit einer Verzögerung versehen sind, wirkt sich eine daraus resultierende falsche Voraussage für den Gatterausgang nur auf die Ereignisliste (in Form einer neuen eingetragenen Ereignisnotiz) oder auf den internen Instanzenzustand aus. Der Koordinator ruft jedoch keine weiteren Gatterinstanzen irrtümlich auf. Wenn der Koordinator eine Instanz zum zweiten Mal zum selben Zeitpunkt aufgrund einer neuen Umgebungsänderung aktiviert, muß sie dies durch ihren internen Zustand merken, die Korrektheit einer vorigen Voraussage überprüfen und die Auswirkung dieser Voraussage in ihrem internen Zustand ggf. annullieren.

Diese falschen Zwischenzustände sind den im Abschnitt 2.2 eingeführten Erkennungskonflikten zuzuschreiben. Der Gattersimulator löst diese Konflikte durch die Trennung zwischen Signaländerung und Änderungsdurchlauf.

Als eine weitere Konsequenz für die Gatterinstanzen sind Aufrufe aufgrund von Umgebungsänderungen und Zeitmeldungen nicht mehr durch den Zeitpunkt des Aufrufes unterscheidbar, was eine Folge des ersten Modellierungsvorschlags war, und die zwei Aufrufgründe können sich sogar überlappen. Im Abschnitt 2.7.1 haben wir schon die Aufgabe der Sequenzermarkenverwaltung den Instanzen überlassen. Dafür müssen hier

Zustandsvariablen für jede Instanz vorhanden sein. Eine boolesche Variable EVENT gibt an, ob für die Instanz ein Ereignis geplant ist, und EVENTTIME liefert ggf. die entsprechende Modellzeit. Damit läßt sich auch die Planung von Ereignissen leicht rückgängig machen, indem wir den Wert FALSE der Variable EVENT zuweisen. Der Koordinator bringt zwar die Instanz noch in Aktion, aber sie findet in ihrem internen Zustand keinen Hinweis auf ein Ereignis für diese Zeit. Jedem Gattereingang Xi muß darüber hinaus eine Variable ZXi zugeordnet sein, die den letzten Wert von Xi besitzt. Erst durch einen Vergleich zwischen ZXi und dem aktuellen Wert von Xi können wir eine Umgebungsänderung erkennen.

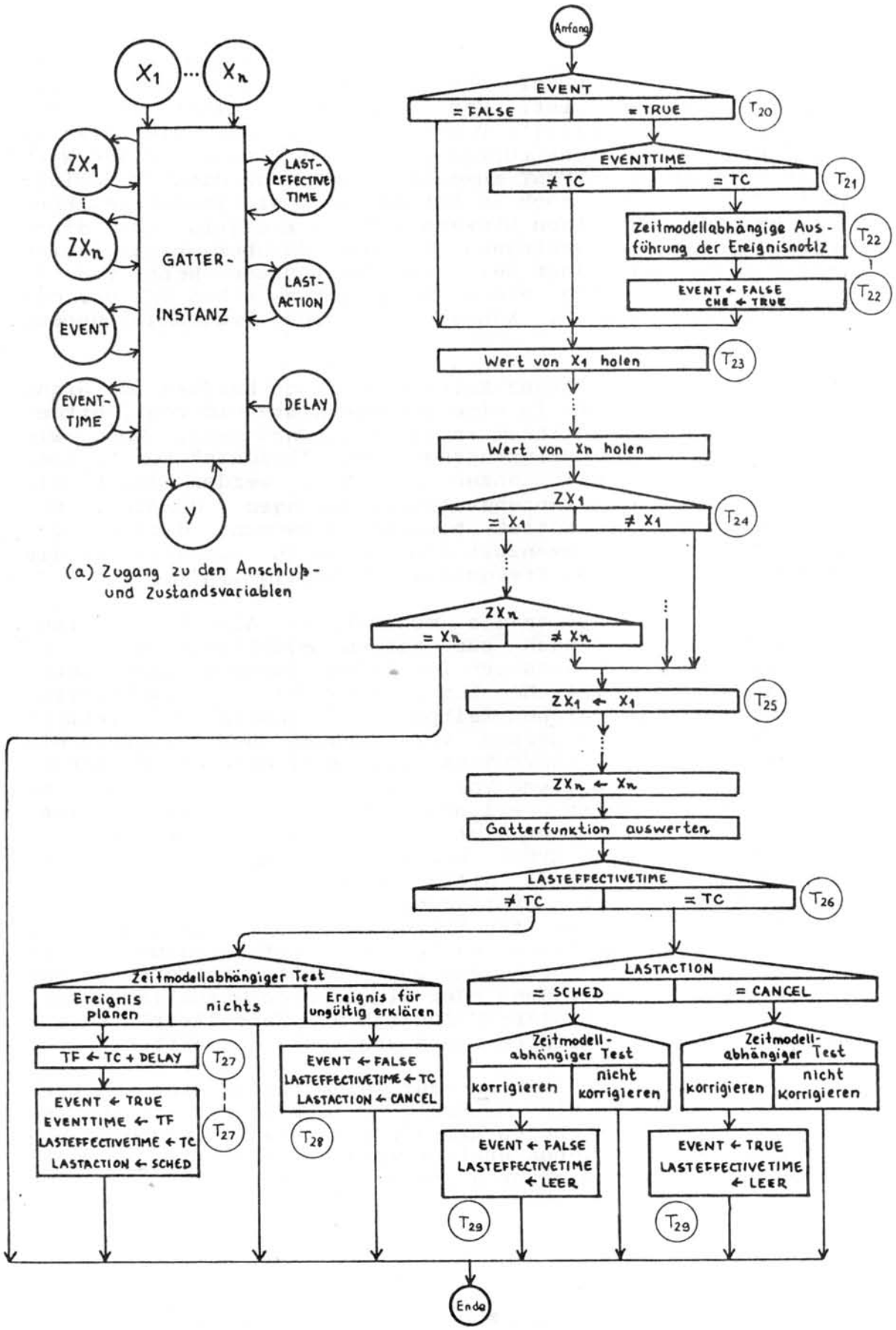
Obwohl eine Gatterinstanz keine wirklichen Notizen im Sinne des Gattersimulators in eine interne Liste einträgt, bildet sie diese Notizen in ihrem internen Zustand nach. Wenn wir im folgenden von "Eintragung" und "Löschen" von Notizen bezüglich der Gatterinstanzen sprechen, werden damit die entsprechenden Instanzenzustandsänderungen gemeint. Die "Ausführung einer Notiz" besteht deswegen darin, die unmittelbaren Instanzenzustandsänderungen auszuführen, die nach Eintreffen eines Ereignisses notwendig sind.

Damit eine Gatterinstanz die mehrmaligen Aufrufe aufgrund von Umgebungsänderungen zur selben Modellzeit merkt und mögliche falsche Voraussagen für ihren Ausgang annulliert, gibt es zwei zusätzlichen Zustandsvariablen. LASTEFFECTIVE-TIME gibt an, zum welchen Zeitpunkt die Instanz zum letzten Mal eine Ausgangsänderung vorausgesagt hat, während die boolesche Variable LASTACTION mit Wertebereich (SCHED, CANCEL) angibt, ob diese Voraussage die Eintragung oder das Löschen einer Notiz verlangt hat. Die Korrektur einer falschen Voraussage erfolgt dann einfach durch die schon eingeführte Variable EVENT. Die in dieser Arbeit behandelten Zeitmodelle erleichtern diese Korrektur, weil sie nur die Annullierung dieser Eintragung bzw. dieses Löschens erfordern, ohne eine zusätzliche gleichzeitige Eintragung oder Löschen zu benötigen. Falls ein Aufruf aufgrund einer dritten Umgebungsänderung zum selben Zeitpunkt vorkommt und der zweite eine Eintragung oder ein Löschen annulliert hat, hat die Variable LASTEFFECTIVETIME den Wert "leer", so als ob die vorigen Aktivitäten nicht stattgefunden hätten.

Abb. 4.5 zeigt den Rahmen für eine allgemeine Gatter-instanzenprozedur. Wir müssen sie durch zusätzliche den Zeitmodellen entsprechende Aktivitäten vervollständigen. Wie im Abschnitt 2.7.1 schon erklärt wurde, erfolgt die Kommuni-kation mit dem Koordinator durch die Variablen TF, TC und CHE.

4.4 Zeitmodell G1

Das Zeitmodell G1 benutzt zweiwertige Logik für die Auswer-tung der Gatterfunktionen und ordnet jedem Gatter eine einzige Übergangszeit für beide Übergangsrichtungen zu. Das



(a) Zugang zu den Anschluß- und Zustandsvariablen

(b) Ablaufdiagramm

Abb. 4.5 - Allgemeiner Rahmen für eine Gatterinstanz

Tiefpassverhalten ("Inertial Delay") /BRE76/ berücksichtigen wir auf eine vereinfachte Weise. Wenn eine Ausgangsänderung für eine Zeit t2 geplant ist und eine neue Eingangsänderung zum Zeitpunkt t1 < t2 derart geschieht, daß der neu berechnete und der aktuelle Ausgangswert gleich sind (und folglich sich von dem geplanten Wert für t2 unterscheiden), müssen wir diese Planung rückgängig machen. Damit gibt es keine Integration von nacheinander folgenden Eingangspulsen. Das Modell G1 ist deswegen nur für Systeme mit langsamen Signalvariationen geeignet.

Als Folge dieses Zeitmodells gibt es zu jedem realen Zeitpunkt maximal ein geplantes Ereignis für jedes Gatter. Wenn eine Eingangsänderung vorkommt, sind die Entscheidungsregeln der Tabelle 4.1 zu befolgen.

neu berechneter Ausgangswert	Gibt es ein geplantes Ereignis?	Folge
= aktueller Wert	nein ja	keine Planung rückgängig machen
≠ aktueller Wert	nein ja	Ereignis planen keine

Tabelle 4.1 - Entscheidungsregeln für die Ereignisplanung nach dem Zeitmodell G1

Um das Modell G1 im Simulationskern des Gattersimulators unterzubringen, brauchen wir einen Verweis EVENTP, der entweder auf eine für das Gatter vorhandene Ereignisnotiz zeigt oder den Wert "NIL" besitzt (falls keine Notiz vorhanden ist bzw. eine Notiz gelöscht wurde). Für die Vervollständigung des Gattersimulators (Abb. 4.3) müssen wir noch bestimmen, wie eine Ereignisnotiz ausgeführt wird und welche Auswirkung eine Gatterfunktionsauswertung hat. Da wir zweiwertige Logik verwenden, braucht eine Notiz keinen neuen Ausgangswert zu enthalten. Er ist immer das Komplement des aktuellen Wertes. Die Ausführung einer Notiz besteht einfach darin, die Variablen SIGNAL[i] und EVENTP[i] zu aktualisieren:

SIGNAL[i]<- not SIGNAL[i], und
EVENTP[i]<- NIL.

Für die Auswirkung der Funktionsauswertung (Abb. 4.6) müssen wir die vorher genannten Entscheidungsregeln befolgen. Dafür sei NEWVALUE die Variable, die den neu berechneten Ausgangswert zugewiesen bekommen hat. Die Eintragung einer Notiz ist der Abb. 4.4 zu entnehmen.

Für die Unterbringung des Modells G1 in der Instanzennetzmodellierung des Gatterebenen-Systems sind zu den im letzten Abschnitt schon eingeführten Variablen keine zusätzlichen

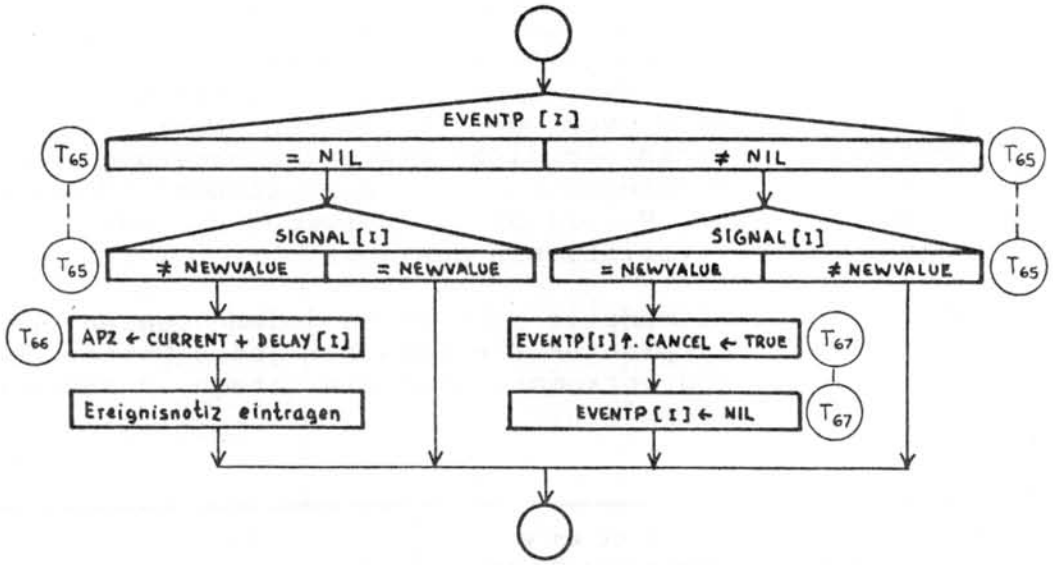


Abb. 4.6 - Auswirkung der Auswertung der Gatterfunktion im Gattersimulator gemäß dem Zeitmodell G1

Variablen notwendig. Der Ausführung einer Notiz entspricht nur die Anweisung Y <- not Y.

Die Entscheidungsregeln für die Annullierung von zu dem selben Modellzeitpunkt durchgeführten Eintragungen und Löschen von Notizen sind sehr leicht aus dem Zeitmodell ableitbar. Die sind in der Tabelle 4.2 aufgeführt.

zum selben Zeitpunkt schon durchgeführt	neu berechneter Ausgangswert	Folge
Eintragung einer Notiz	= aktueller Wert ≠ aktueller Wert	Eintragung annullieren keine
Löschen einer Notiz	= aktueller Wert ≠ aktueller Wert	keine Löschen annullieren

Tabelle 4.2 - Entscheidungsregeln für die Annullierung einer Eintragung bzw. eines Löschens einer Notiz nach dem Zeitmodell G1

Für die Auswirkung der Gatterfunktionsauswertung, die die Abb. 4.7 erläutert (der Klarheit wegen ist eine gewisse Überlappung mit der Abb. 4.5 vorhanden), werden die Regeln aus der Tabelle 4.2 berücksichtigt.

Die Eintragung und das Löschen von Ereignissen und die Korrektur falscher Voraussagen erfolgen wie in dem allgemeinen Rahmen für Gatterinstanzen (Abb. 4.5).

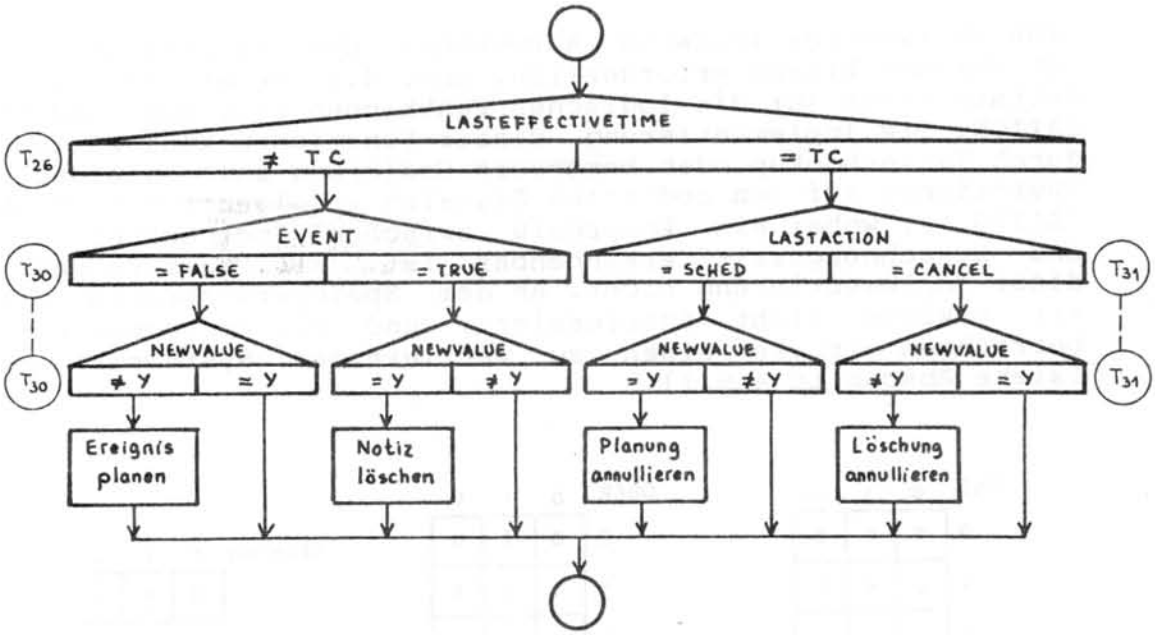


Abb. 4.7 - Auswirkung der Auswertung der Gatterfunktion in der Instanzennetzmodellierung der Gatterebene gemäß dem Zeitmodell G1

4.5 Zeitmodell G2

4.5.1 Modelleigenschaften

Dieses Modell wird durch eine Zweideutigkeitsregion ("Ambiguity Region") /SZY70/ /CHA71/ gekennzeichnet. Diese Region stellt die Übergangsphase zwischen den zwei definierten logischen Ausgangszuständen 0 und 1 dar. Während des Überganges nimmt ein Signal den Wert "undefiniert" (U) an. Ziel dieses Modells ist es, den "Worst-Case" zu identifizieren, d.h., wenn ein oder mehrere Gattereingänge den Wert U haben und dadurch die logische Funktion den Gatterausgangswert U ergibt, soll die längst mögliche Übergangsphase für den Ausgang vorausgesagt werden. Dies erreichen wir durch zwei Modelleigenschaften.

a) Jedem Gatter sind zwei Verzögerungswerte MINDEL und MAXDEL zugeordnet, die kleinste bzw. die höchste Verzögerung, die das Gatter haben kann.

b) Der Wert U ist der "dominante" Wert, während 0 und 1 "nichtdominant" sind. Folgende Regel müssen wir beachten, um den Verzögerungswert zu bestimmen, der für einen Übergang anzuwenden ist /BRE76/:

Übergang	Verzögerung
nichtdominanter -> dominanter Wert	= MINDEL
dominanter -> nichtdominanter Wert	= MAXDEL

Wenn wir diese Regeln befolgen, nimmt ein Signal den Wert U möglichst früh an und geht auf den definierten Wert 0 oder 1 möglichst spät.

Eine dreiwertige logische Auswertung der Gatterfunktionen ist darüber hinaus erforderlich. Abb. 4.8 zeigt die Wahrheitstabellen für die logischen Funktionen UND, ODER und NEGATION. Die Implementierung dieser Funktionen kann entweder durch Table-Lookup oder besondere Codierung samt geeigneten Operationen auf den codierten Signalen erfolgen (z.B. siehe /ALI78/), wobei ein Kompromiß zwischen Speicherverbrauch und Berechnungszeit erkennenbar ist. Wir kümmern uns um diese Implementierung nicht. An dem Speicherverbrauch sind wir sowieso nicht interessiert, und die Berechnungszeit betrachten wir als einen zu variierenden Systemparameter (siehe Abschnitt 4.6.1).

UND	0	1	U
0	0	0	0
1	0	1	U
U	0	U	U

ODER	0	1	U
0	0	1	U
1	1	1	1
U	U	1	U

NEGATION	0	1	U
	1	0	U

Abb. 4.8 - Dreiwertige logische Funktionen

Damit das Modell G2 noch deutlicher vom ersten Modell unterscheidbar ist, zeigen die Gatter jetzt kein Tiefpassverhalten. Damit werden die Gatter reine Verzögerungsleitungen (samt logischen Funktionen, selbstverständlich), ausgenommen die Verzerrung durch die unsymmetrischen Übergangszeiten. Hauptfolge dieses Verhaltens ist, daß zu einem bestimmten realen Zeitpunkt mehrere Ereignisse für ein Gatter geplant sein können.

Wenn wir diese zwei Merkmale, die Zweideutigkeitsregion und die Abwesenheit von Tiefpassverhalten, zusammensetzen und gewährleisten, daß alle Signalquellen Wertereihenfolgen liefern, die zwischen zwei definierten Zuständen (0 und 1) immer während eines gewissen Modellzeitintervalls den Wert U haben, entstehen viele für die Implementierung der Simulationsalgorithmen wichtige Eigenschaften.

a) Alle Gatterausgänge haben Wertesequenzen
nichtdominant -> dominant -> nichtdominant
(Lemmata 1 und 2 von Eichelberger /EIC65/).

b) Nur das Ereignis mit größter Planzeit zwischen allen für ein Gatter vorhandenen Ereignissen kann für ungültig erklärt werden müssen.

c) Wenn die Planung eines Ereignisses annulliert wird, ist aufgrund desselben Eingangsübergangs eine gleichzeitige Planung eines neuen Ereignisses nie notwendig (d.h. der neu berechnete Ausgangswert und der dem vorletzten Ereignis entsprechende Wert sind identisch).

d) Nachdem eine Ereignisplanung rückgängig gemacht worden ist, wird die nächste Aktivität für das Gatter zu einem späteren Modellzeitpunkt sicherlich die Planung eines neuen Ereignisses sein.

e) Die Entscheidungsregeln für die Auswirkung einer Gatterfunktionsauswertung hängen vom Verhältnis zwischen den Zeiten t1 und t2 ab, wobei t1 die Planzeit eines möglichen neuen Ereignisses und t2 die größte Planzeit zwischen allen Ereignissen für dieses Gatter ist. Sie sind in der Tabelle 4.3 gezeigt.

t1	neu berechneter Ausgangswert	Folge
=< t2	interessiert nicht	Planung rückgängig machen
> t2	= Ausgangswert nach dem Ereignis für t2	keine
	≠ Ausgangswert nach dem Ereignis für t2	neues Ereignis planen

Tabelle 4.3 - Entscheidungsregeln für die Ereignisplanung nach dem Zeitmodell G2

f) Falls kein geplantes Ereignis vorhanden ist, hängt die Planung natürlich vom Verhältnis zwischen dem neu berechneten Wert und dem aktuellen Ausgangswert ab.

Auf den Beweis dieser Eigenschaften wird hier verzichtet, weil er sehr lang und für die Arbeitsziele unwichtig ist.

4.5.2 Gattersimulator unter Verwendung des Zeitmodells G2

Zunächst erweitern wir eine Ereignisnotiz um zwei Felder TIME und VALUE wegen der Entscheidungsregeln und der dreiwertigen Logik. Außerdem ist ein Pointer LASTEVENT auf die für jedes Gatter mit der größten Planzeit versehenen Ereignisnotiz notwendig, damit das Löschen direkt auf der Ereignisliste durchgeführt werden kann.

Die Variable LASTVALUE enthält den logischen Wert, der mit dem neu berechneten Ausgangswert hinsichtlich einer Planung zu vergleichen ist. Für die Besetzung dieser Variable müssen wir folgende Regeln beachten:

a) Wenn wir eine neue Notiz in die Ereignisliste eintragen, bekommt LASTVALUE den Ausgangswert, der in der Notiz vermerkt ist (aus der Eigenschaft e);
b) Wenn wir eine Notiz löschen, bekommt LASTVALUE den neu berechneten Ausgangswert (aus der Eigenschaft c); und
c) Wenn wir eine Notiz ausführen und sie die einzige Notiz für dieses Gatter war, bekommt LASTVALUE den neuen aktuellen Ausgangswert (aus der Eigenschaft f).

Nach einem Löschen verweist LASTEVENT immer noch auf die gelöschte Notiz. Die Eigenschaft d) vom letzten Abschnitt und die Existenz der Variable LASTVALUE erlauben diesen falschen Zwischenzustand, der nach der nächsten Eintragung einer Notiz für dieses Gatter korrigiert wird.

Die absolute Planzeit einer neuen Notiz ergibt sich eindeutig aus dem neu berechneten Ausgangswert. Wenn er U ist, haben wir einen Übergang von einem nichtdominanten auf einen dominanten Wert und müssen MINDEL benutzen. Wenn er 0 oder 1 ist, muß der Wert in der mit größter Planzeit versehenen Notiz bzw. der bisher aktuelle Ausgangswert U sein, und es handelt sich dabei um einen Übergang von einem dominanten auf einen nichtdominanten Wert, der MAXDEL verlangt.

Abbildungen 4.9 und 4.10 zeigen den Ablauf der Ausführung einer Notiz bzw. der Auswirkung einer Funktionsauswertung, mit denen wir den Gattersimulator vervollständigen.

4.5.3 Instanzennetzmodellierung unter Verwendung des Zeitmodells G2

In unserem allgemeinen Rahmen für die Instanzennetzmodellierung von Gatterebenen-Systemen haben wir die Variablen EVENT und EVENTTIME unter der Annahme vorgesehen, daß jede Instanz zu jedem realen Zeitpunkt höchstens ein geplantes Ereignis hätte. Da dies für das Zeitmodell G2 nicht der Fall ist, müssen wir die Variable EVENTTIME in ein Array umwandeln, welches die Planzeiten aller der Instanz entsprechenden Ereignisse enthält (siehe Abb. 4.11). Dieses Array ist zyklisch mit einer zu dimensionierenden Größe M organisiert. Die Verweise NEXT und NEW zeigen auf die Arraystellen, wo die Planzeit des nächsten stattzufindenden Ereignisses zu suchen bzw. wo die Planzeit eines neuen Ereignisses einzutragen ist. Die boolesche Variable EVENT kann jetzt entfallen, da die Abwesenheit von Ereignissen durch die Gleichheit zwischen NEXT und NEW gekennzeichnet ist. Damit muß M die höchste Anzahl von gleichzeitig vorzukommenen Ereignissen für ein Gatter um 1 überschreiten.

Da dreiwertige Logik jetzt vorhanden ist, ist der neue nach dem Eintreffen eines Ereignisses vom Gatterausgang zu übernehmende logische Wert nicht mehr eindeutig aus dem aktuellen Wert ableitbar. Dieser neue Wert muß ebenfalls in einem Array (namens VALUE) eingetragen sein, welches dieselbe Organisation wie EVENTTIME hat und auf das auch von

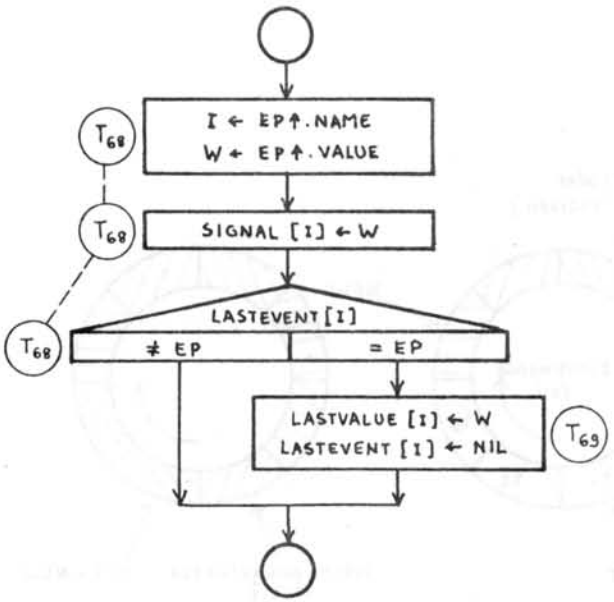


Abb. 4.9 - Ausführung einer Ereignisnotiz im Gattersimulator gemäß dem Zeitmodell G2

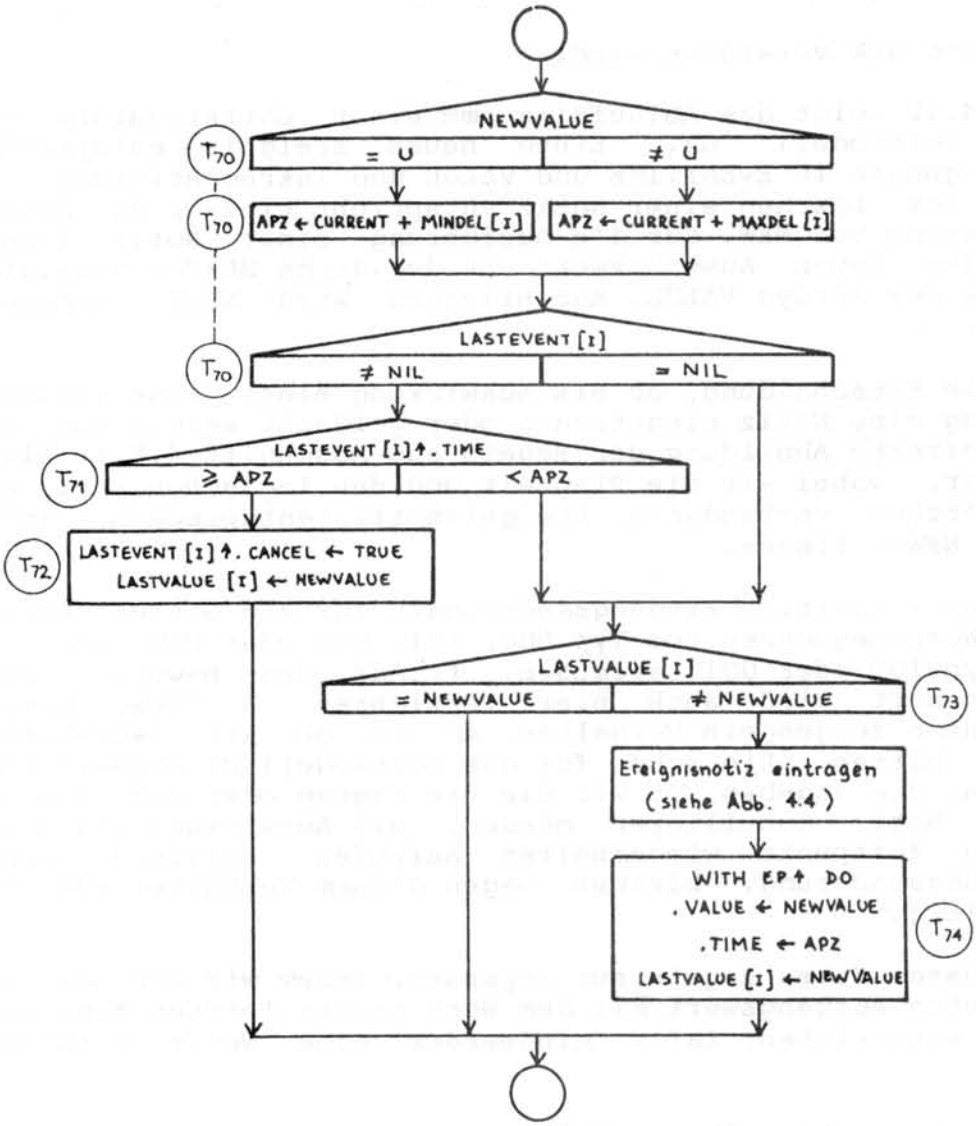


Abb. 4.10 - Auswirkung der Auswertung der Gatterfunktion im Gattersimulator gemäß dem Zeitmodell G2

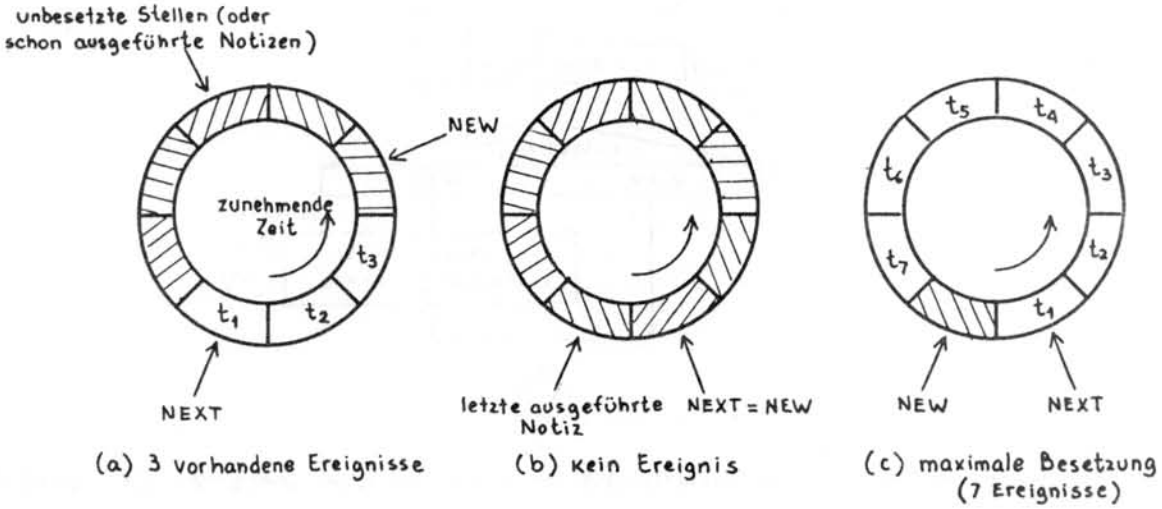


Abb. 4.11 - Organisation des Arrays EVENTTIME für die Instanzennetzmodellierung von Gatterebenen-Systemen nach dem Zeitmodell G2 (Beispiele für M = 8)

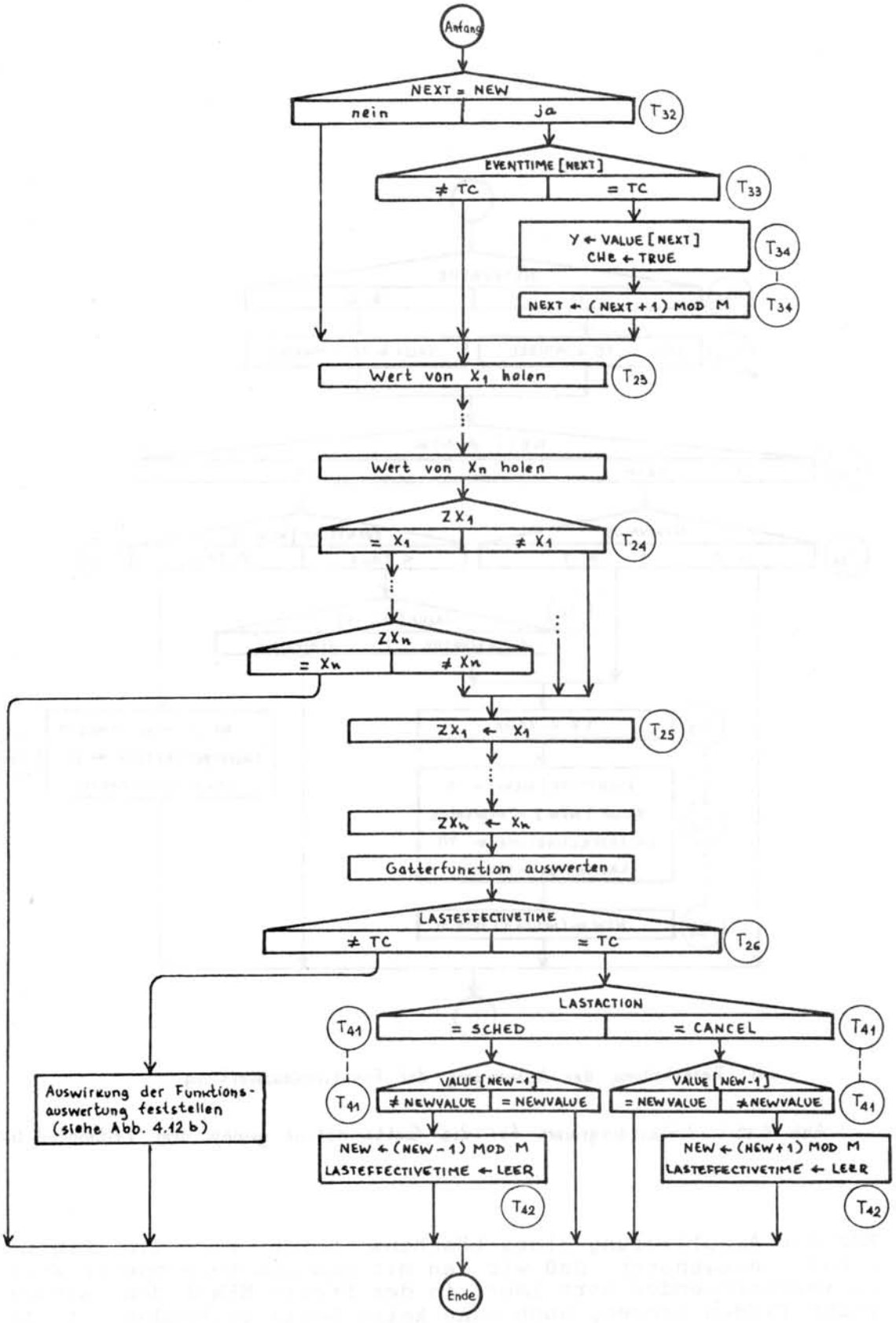
NEXT und NEW verwiesen wird.

Abb. 4.12 zeigt das Ablaufdiagramm einer Gatterinstanz für das Zeitmodell G2. Einem neuen Ereignis entsprechen Eintragungen in EVENTTIME und VALUE und Inkrementierung von NEW. Dem Löschen einer Notiz entspricht einfach die Dekrementierung von NEW. Für die Ausführung einer Notiz finden wir den neuen Ausgangswert in der durch NEXT angezeigten Stelle des Arrays VALUE. Anschließend wird NEXT inkrementiert.

Für die Entscheidung, ob als Auswirkung einer Funktionsauswertung eine Notiz eingetragen oder gelöscht werden muß, ist eine direkte Abbildung der Regeln aus Abschnitt 4.5.1 leicht machbar, wobei wir die Planzeit und den logischen Wert, die der letzten vorhandenen Ereignisnotiz entsprechen, immer durch NEW-1 finden.

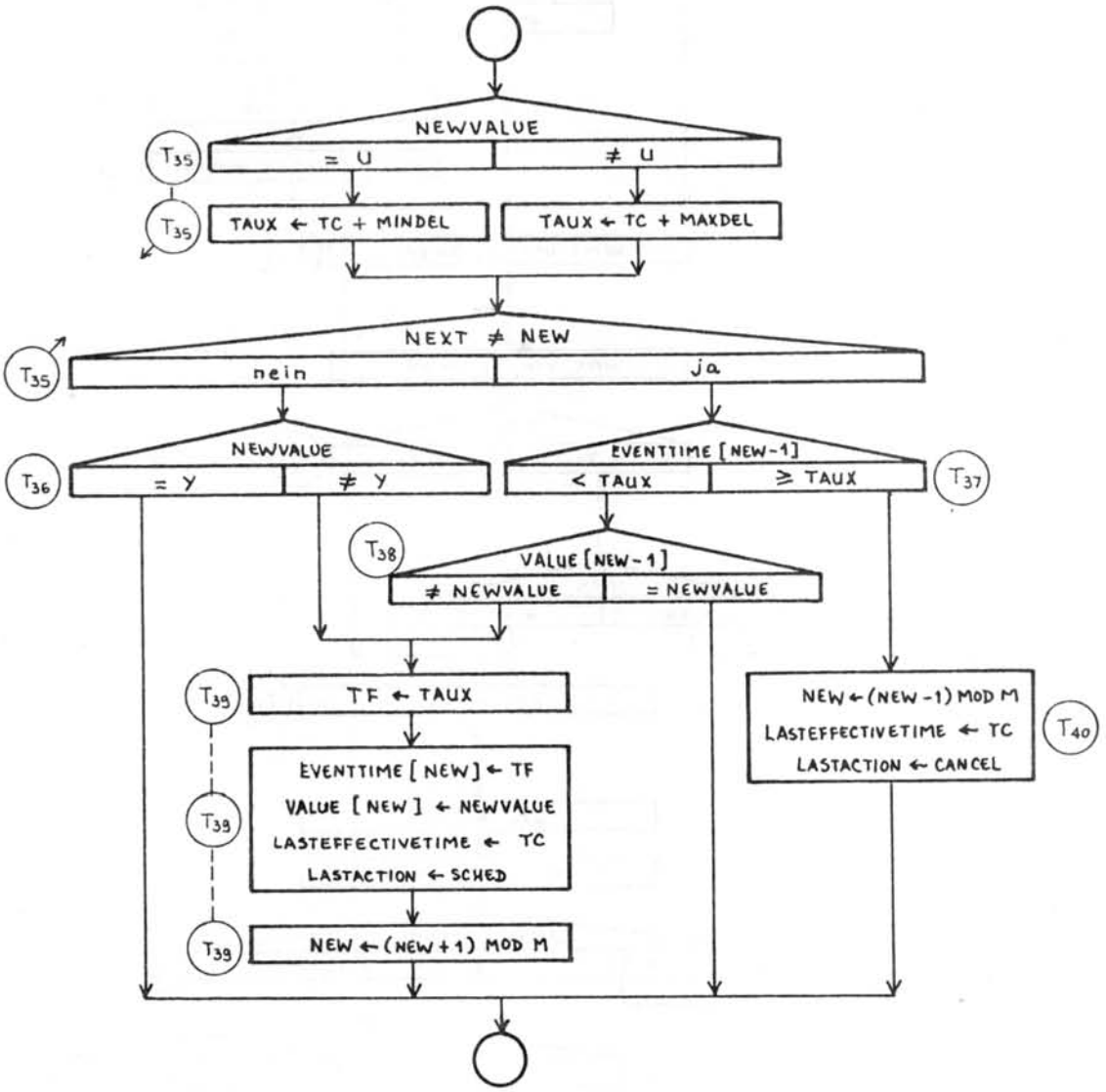
Zwei gleichzeitige Eingangsänderungen für ein Gatter können nur Wertesequenzen vom Typ 0U0, 1U1, U0U oder U1U, aber nie vom Typ 1U0 oder 0U1 erzeugen. (Auf den Beweis dieser Eigenschaft wird auch hier verzichtet.) Diese Wertesequenzen zeigen ein Verhalten, so als ob wir zweiwertige Logik hätten. Die schon für das Zeitmodell G1 abgeleiteten Regeln, die angeben, ob wir die Eintragung oder das Löschen einer Notiz annullieren müssen, als Auswirkung eines zum selben Zeitpunkt wiederholten Aufrufes aufgrund einer Umgebungsänderung, bleiben wegen diesem Verhalten auch für G2 gültig.

Wir müssen diese Regeln nur anpassen, indem wir den neu berechneten Ausgangswert mit dem Wert in der letzten Ereignisnotiz vergleichen, falls mindestens eine Notiz vorhanden ist.



(a) Hauptrumpf

Abb. 4.12 - Ablaufdiagramm für die Gatterinstanz gemäß dem Zeitmodell G2
(Fortsetzung auf der nächsten Seite)



(b) Feststellung der Auswirkung der Funktionsauswertung

Abb. 4.12 - Ablaufdiagramm für die Gatterinstanz gemäß dem Zeitmodell G2

Für die Annullierung eines Löschens haben wir die Eigenschaft ausgenutzt, daß wir den mit dem neu berechneten Wert zu vergleichenden Wert immer in der Stelle NEW-1 des Arrays VALUE finden können, auch wenn keine Notiz vorhanden ist. In diesem Fall haben wir NEW = NEXT (siehe Abb. 4.11b) und da die höchste effektive Arraybesetzung M-1 Stellen benötigt, hat die Stelle NEXT-1 (= NEW-1) den aktuellen Ausgangswert.

Für die Annullierung einer Eintragung ist zu beachten, daß wir gegenüber dem Zeitmodell G1 den Test getauscht haben, weil der Vergleich jetzt mit dem (möglicherweise) falschen geplanten Ausgangswert erfolgt, und nicht mit dem Wert, der vor der Eintragung letzter Wert war.

Die Regeln für die Entscheidung, ob eine Eintragung oder ein Löschen annulliert werden muß, sind dann in der Tabelle 4.4 aufgeführt, wobei LW der Ausgangswert in der letzten Notiz und AW der aktuelle Ausgangswert ist.

zum selben Zeitpunkt schon durchgeführt	neu berechneter Ausgangswert,		Folge
	falls keine Notiz vorhanden ist	falls Notizen vorhanden sind	
Eintragung	- - - - - -	= LW ≠ LW	keine Eintragung annullieren
Löschen	= AW ≠ AW	= LW ≠ LW	keine Löschen annullieren

LW = Ausgangswert in der Notiz mit größter Planzeit
AW = aktueller Ausgangswert

Tabelle 4.4 - Entscheidungsregeln für die Annullierung einer Eintragung bzw. eines Löschens einer Notiz nach dem Zeitmodell G2

4.6 G-Systemparameter

In diesem Abschnitt definieren wir alle Systemparameter, die für eine genaue Ableitung der Zeitverbrauchsausdrücke für die Simulation der Gatterebene mit dem spezifischen und dem Instanzennetzsimulator notwendig sind. Zusätzlich zu den Grundparametern definieren wir einige abgeleitete Parameter, die einem besseren Verständnis der Ausdrücke dienen. Bei der Definition der Grundparameter legen wir gleichzeitig vernünftige Wertebereiche fest. Beziehungen zwischen Parametern werden intuitiv und falls möglich mathematisch festgestellt, damit wir die Parameter nicht irrtümlicherweise unabhängig voneinander variieren.

4.6.1 Statische Parameter

Wir stellen unter der Bezeichnung "statisch" alle Parameter zusammen, die entweder topologische Netz- und Gattereigenschaften widerspiegeln oder nicht durch die spezifischen Simulationsalgorithmen entstanden sind. Wir definieren dann

N als die Anzahl der Gatter im System (sein Wertebereich wird zusammen mit dem dynamischen Parameter p festgelegt),

Y = 1,7 .. 2,7 als die durchschnittliche Anzahl der Gattereingänge,
U = 2 .. 3 als die durchschnittliche Anzahl der mit einem Gatterausgang verbundenen Gatter (Fan-Out-Zahl),
a = 0 .. 1 als die durchschnittliche Wahrscheinlichkeit, daß ein Gatterausgang infolge einer oder mehrerer gleichzeitiger Eingangsänderungen sich ändert, und
Z als den durchschnittlichen Zeitverbrauch der Gatterfunktionsauswertung.

Den Parameter a brauchen wir nicht für die Zeitverbrauchsausdrücke, aber er hilft uns während der Ableitung einiger Parameterbeziehungen. Wenn wir eine reine Wahrscheinlichkeitsanalyse machten, um einen Wert a' für ein gewisses Gatter zu bestimmen, würden wir feststellen, daß a' eine reine Funktion der Anzahl Y' der Gattereingänge ist, und zwar

$$a' = \frac{1}{2^{Y'-1}} .$$

Diese Beziehung läßt sich nicht für die Mittelwerte a und Y anwenden. Falls wir die Prozentsätze der im System mit i Eingängen vorhandenen Gatter hätten, könnten wir eine Beziehung zwischen a und Y feststellen. Wir möchten uns aber mit dem Repertoireumfang nicht befassen. Diese Beziehung würde die Tatsache zum Ausdruck bringen, daß ein größerer Prozentsatz von Gattern mit vielen Eingängen den Wert von Y nach oben und den von a nach unten schiebt.

Wie schon erwähnt betrachten wir den Zeitverbrauch der Auswertung der Gatterfunktionen als einen Parameter Z, um uns mit der Implementierung dieser Berechnungen und mit dem Repertoireumfang nicht beschäftigen zu müssen. Außerdem vermeiden wir damit, die Prozentsätze aller im System vorhandenen Repertoiretypen als Parameter betrachten zu müssen. Da das Modell G2 dreiwertige Logik verwendet, ist die Komplexität der Funktionsauswertung für G2 wesentlich größer als für das Modell G1, das zweiwertige Logik benutzt. Wir legen deshalb verschiedene Wertebereiche
Z1 = 4 .. 10 für das Modell G1 und
Z2 = 10 .. 30 für das Modell G2 fest.
Diese Zeit umfaßt den Zugriff auf die Werte der Gattereingänge in der Datenstruktur nicht. Sie deckt nur die reine logische Funktionsauswertung und ist damit für den Gatter- und den Instanzennetzsimulator gleichwertig.

Weil die Parameter N, Y, U, a und Z unabhängig vom verwendeten Simulator sind, haben sie für ein bestimmtes System in beiden Simulatoren dieselben Werte.

4.6.2 Dynamische Parameter

Die dynamischen Parameter spiegeln die Anzahl der Gatter wider, deren Namen während der Simulation in die Ereignis- und in die Rufliste eingetragen werden. Wir definieren

p als die durchschnittliche Rate der Gatter, die zu einem bestimmten Modellzeitpunkt eine effektive (nicht gelöschte) Ereignisnotiz auszuführen haben. Damit ist p*N die durchschnittliche Anzahl der effektiven Notizen für eine Modellzeit. Da wir uns für einen "Fixed-Step"-Zeitfortschaltungsmechanismus entschieden haben, ist unser Gattersimulator nur für Systeme mit p*N > 1 geeignet, was sowieso die Mehrheit der Gatterebenen-Systeme trifft (siehe Abschnitt 4.2). Wir legen dann einen Wertebereich

p*N = 1 .. 15

fest. Wir werden sehen, daß die Parameter p und N in den Zeitverbrauchsausdrücken nie getrennt auftauchen, so daß ihre einzelnen Wertebereiche unwichtig sind.

Eine wichtige Eigenschaft unserer Messung des Zeitverbrauchs ist die Erhaltung der Ereigniszahl im Laufe der Zeit. Wir nehmen an, daß während eines genügend langen Intervalls das System im Dauerzustand simuliert wird, d.h. daß die Anzahl der Ereignisnotizen in der Ereignisliste weder wächst noch abnimmt, sondern um einen Mittelwert schwankt. Nur unter diesen Umständen ist es sinnvoll, über durchschnittliche Zahlen zu sprechen. Damit die Aktivität konservierend ist, muß die durchschnittliche Anzahl der zu einem Modellzeitpunkt ausgeführten Notizen gleich der Anzahl der neuen effektiven eingetragenen Notizen sein. Das bedeutet, daß p*N sowohl die ausgeführten als auch die effektiv eingetragenen Notizen zu einem Modellzeitpunkt bezeichnet.

Wir definieren weiter

c als das durchschnittliche Verhältnis zwischen der Anzahl der gelöschten und der Anzahl der effektiven Ereignisnotizen. Damit ist c*p*N die Durchschnittsanzahl der gelöschten Notizen und (c+1)*p*N die Durchschnittsanzahl aller vorhandenen (effektiven + gelöschten) Notizen zu einem Modellzeitpunkt. Über den Wertebereich von c werden wir später diskutieren.

Wir haben Gatterebenen-Systeme als Instanzennetze mit der bewußten Absicht modelliert, mit dem Instanzennetz- und dem Gattersimulator die gleichen Ergebnisse zu erzielen. Dies wird nur dadurch erreicht, daß am Ende einer Systemüberführung dieselben Voraussagen über den künftigen Simulationsverlauf gemacht sind, obwohl die Simulation während der Systemüberführung wegen der Erkennungskonflikte in der Instanzennetzmodellierung über unterschiedliche Zwischenzustände laufen kann. Aufgrund dieser Eigenschaft müssen wir am Ende einer Systemüberführung dieselben Notizen eingetragen und gelöscht haben, und das bedeutet, daß p*N und c dieselben Werte für beide Simulatoren haben müssen.

Es ist hier sinnvoll, zwei abgeleitete Parameter zu definieren, die die Ausdrücke besser interpretierbar machen. Seien

E die durchschnittliche Anzahl der effektiven Ereignisnotizen zu den Modellzeitpunkten, für die wir mindestens ein effektives Ereignis geplant haben, und

D die durchschnittliche Anzahl der vorhandenen (effek-

tiven + gelöschten) Ereignisnotizen zu den Modellzeitpunkten, für die wir mindestens ein Ereignis geplant haben. Wir haben dann

```
E = p*N , falls p*N >= 1,
E = 1   , falls p*N <  1,

D = (c+1)*p*N , falls (c+1)*p*N >= 1 und
D = 1         , falls (c+1)*p*N <  1.
```

Wir definieren noch

W als die durchschnittliche Anzahl der mit einem Gatterausgang verbundenen Gatter, deren Namen wir wegen einer Wertänderung dieses Ausgangs in die Rufliste bringen, falls der Gattersimulator verwendet wird. Falls p*N Gatter ihre Ausgänge zu einem Modellzeitpunkt ändern, werden damit W*p*N Gatter betroffen, und nicht U*p*N, weil einige dieser W*p*N Gatter mit zwei oder mehreren der p*N Gatter verbunden sein können. Die Variable FLAG, die in beiden Simulatoren vorhanden ist, verhindert, daß der Name eines Gatters mehrmals in die Rufliste eingetragen wird. Daraus folgt W =< U mit dem Wertebereich

W = 1,5 .. 3.

Wir werden später sehen, warum die Anzahl der in die Rufliste einzutragenden Gatternamen für die Instanzennetzmodellierung nicht W ist.

4.6.3 Zeitfortschaltung in der Instanzennetzmodellierung

Für das Verständnis dieses Abschnittes müssen wir die IL-Organisation der Ereignisliste für den Instanzennetzsimulator im Blick haben. Wir definieren

q als die Wahrscheinlichkeit, daß die nächste Ereignisnotiz eine Scheinnotiz ist, wenn die Modellzeit fortgeschaltet wird,

Me als die durchschnittliche Anzahl der für die Eintragung einer neuen Notiz im Zielintervall linear durchlaufenen Notizen, und

Ms wie Me, aber für die Eintragung einer Scheinereignisnotiz.

Diese Definitionen gelten für alle möglichen Anwendungen des Instanzennetzsimulators. Im Abschnitt 2.7.4 haben wir jedoch schon gesagt, daß wir die Intervallgröße DT für jedes zu simulierendes System optimal dimensionieren müssen. Aus dieser Intervallgröße können wir die Werte für q, Me und Ms ableiten. Die folgende Analyse gilt für die Instanzennetzmodellierung von Gatterebenen-Systemen, falls p*N >= 1 ist.

Wenn die Ereignisse eine gleichmäßige Verteilung zeigen, so daß für jede Modellzeit mindestens eine Notiz vorhanden ist, haben wir q = 1/DT. DT ist die Zeitspanne eines Intervalls zwischen zwei aufeinander folgenden Scheinereignisnotizen (siehe Abschnitt 2.7.4). Aus einer ausführlicher Wahrscheinlichkeitsanalyse läßt sich auch

$$Me = (DT - 1) * D / 4 \qquad (4.1)$$

ableiten. Da wir die Scheinnotizen immer in das letzte Intervall wieder eintragen, gilt dieser Ausdruck nur dann für Ms, wenn die Ereignisse über alle Intervalle gleich verteilt sind. Wir werden eine solche gleichmäßige Verteilung für diese Analyse voraussetzen. Eine Änderung von Ms aus dem durch (4.1) festgelegten Wert kann problemlos erfolgen, weil dieser Parameter einen geringen Einfluß auf den gesamten Zeitverbrauch hat.

Aus den Sektiondefinitionen für den Koordinator im Abschnitt 3.4 leiten wir folgenden Ausdruck für den Zeitverbrauch der Simulation eines Modellzeitschrittes ab, was die Operationen auf der Ereignisliste betrifft:

$$ZV = T7 + q * (T8 + T9 * Ms) + D * (K + T3 * Me),$$

wobei K für die restlichen Sektionen steht. Unter unserer Ausführungszeitannahme für die Algorithmenschritte haben wir

$$ZV = 20 + q * (116 + 15 * Ms) + D * (K + 15 * Me). \qquad (4.2)$$

Für die Minimierung von ZV besteht ein Kompromiß zwischen q und Me (und Ms), weil eine Verminderung der Anzahl der durchlaufenen Notizen (kleiner Wert von Me und Ms) nur durch eine Erhöhung des Wertes von q (kleinere Intervalle) erreichbar ist. Da Me und Ms Funktionen von D und q sind, können wir (4.2) in Bezug auf q minimieren, indem wir dZV/dq gleich Null machen (dD/dq ist selbstverständlich gleich Null). Daraus ergibt sich

$$q^2 = \frac{3,75 * D^2}{116 - 3,75 * D} . \qquad (4.3)$$

Wir können allerdings nicht beliebige Werte für q festlegen. Da der minimale Zeitabstand zwischen Ereignissen, die für unterschiedliche Zeiten geplant sind, den Wert 1 hat, hat ein Wert DT < 1 keinen Sinn. Ein solcher Wert von DT würde Intervalle ohne Ereignisse verursachen. Für DT >= 1 gilt immer q=1/DT. Aus der Definition hat q einen maximalen Wert von 1. Außerdem werden wir uns auf ganzzahlige Werte von DT beschränken. Unter diesen Voraussetzungen bekommen wir gemäß (4.3) die optimalen Werte für q, die in der Tabelle 4.5 zusammengefaßt sind.

Für Systeme mit p*N < 1 nehmen wir immer c = 0 an, weil unser Interesse an solchen Systemen sowieso begrenzt ist. Für solche Systeme definieren wir DT(ef) als die Anzahl der im Intervall vorhandenen Modellzeitpunkte, die mindestens eine Notiz haben. Aus dieser Definition leiten wir

$$DT(ef) = DT * p * N \text{ und}$$
$$q = 1 / DT(ef)$$

ab. Der Ausdruck (4.1) für Me bleibt gültig, aber wir müssen darauf achten, daß für p*N < 1 der Parameter D immer den Wert 1 hat (siehe Definition im Abschnitt 4.6.2). Deswegen

D	optimaler Wert für q	optimaler Wert für DT=1/q	beste Annäherung für DT	beste Annäherung für q
1	0,18	5,6	5	1/5
2	0,37	2,7	3	1/3
3	0,57	1,8	2	1/2
4	0,77	1,3	1	1
>=5	1,00	1,0	1	1

Tabelle 4.5 - Dimensionierung der Intervalle der Ereignisliste für die Instanzennetzmodellierung von Gatterebenen-Systemen

können wir einheitliche optimale Werte für q und Me für alle Systeme mit p*N < 1 und c = 0 festlegen:
DT(ef) = 5, q = 1/5 und Me = (5-1)*1/4 = 1.

4.6.4 Zeitfortschaltung im Gattersimulator

Für diesen Abschnitt ist die Kenntnis über die TM-Organisation der Ereignisliste für den Gattersimulator wichtig. Wir definieren

X als die durchschnittliche Anzahl der während einer Zeitfortschaltung leer abgesuchten Zeitschlitze, die keine Ereignisnotizen haben. Für Systeme mit p*N >= 1 und gleichmäßiger Verteilung der Ereignisse, für die jeder Zeitschlitz mindestens eine Notiz hat, hat X den Wert Null. Darin liegt nämlich der Vorteil eines "Fixed-Step"-Zeitfortschaltungsmechanismus. Für ungleichmäßige Verteilungen könnte X ungleich Null sein, auch wenn p*N >= 1 wäre. Wir berücksichtigen diese Fälle aber nicht, weil eine zusammenhängende analytische Festlegung des Einflußes dieser Ungleichmäßigkeit auf q und Me einerseits und X andererseits zu komplex wäre. Außerdem würde X aufgrund einer solchen Ungleichmäßigkeit einen Wert wenig größer als Null haben, und sein Einfluß auf den gesamten Zeitverbrauch wäre bedeutungslos. Gleichmäßige Verteilungen liegen auch den Ausdrücken für E und D im Abschnitt 4.6.2 zugrunde.

Für Systeme mit p*N < 1, c = 0 und gleichmäßigen Verteilungen, für die kein Zeitschlitz mehr als eine Notiz hat, gilt der Ausdruck

$$(X + 1) * p * N = 1 . \qquad (4.4)$$

Wenn beispielweise p*N = 0,2 gilt, hat das System dann eine Ereignisnotiz je 5 Zeitschlitze. Das bedeutet, daß während einer Zeitfortschaltung 4 leere Zeitschlitze (X = 4) durchlaufen werden, bevor wir eine Notiz finden.

Aus den möglichen Abläufen für den Algorithmus des Gattersimulators (Abb. 4.3) erkennen wir, daß zwei zusätzliche Parameter notwendig wären, und zwar

Xc , die durchschnittliche Anzahl der während einer Zeitfortschaltung abgesuchten Zeitschlitze, die nur gelöschte Notizen haben, und

Ec , die durchschnittliche Anzahl der in jedem dieser Zeitschlitze gelöschten Notizen. Ein Xc mit Wert ungleich Null würde aber nur in Systemen mit kleinem Wert von p*N und ungleichmäßiger Ereignisverteilung vorkommen. Wir lassen diese Möglichkeit beiseite, auch weil der Einfluß von Xc auf den Zeitverbrauch gering wäre. Falls Xc=0 gilt, hat der Parameter Ec keinen Sinn.

4.6.5 Behandlung der Erkennungskonflikte in der Instanzennetzmodellierung

Im Abschnitt 4.3 haben wir die Gatterinstanz so definiert, daß sie falsche aufgrund von Erkennungskonflikten erzeugte Zwischenzustände noch rechtzeitig wiedergutmachen kann. Wir haben gesehen, daß diese Zwischenzustände mit der Aufrufreihenfolge der Instanzen zusammenhängen. Nach unserer spezifischen Implementierung der Rufliste haben wir im Abschnitt 2.7.5 gezeigt, daß zwei gleichzeitige Ereignisse für die Eingangssignale eines Gatters zwei Aufrufe aufgrund von Umgebungsänderungen nicht bewirken können. Wir brauchen uns dann für die Messung des Simulationszeitverbrauches nicht mit dieser Situation zu befassen, obwohl der Konstrukteur des Instanzennetzmodells diese Möglichkeit auch einkalkulieren muß, weil er die Implementierung des Koordinators nicht kennt.

Für eine Gatterinstanz A, deren Eingänge mit den Signalen (=Instanzen) S1, S2, ... verbunden sind, können trotzdem zwei Fälle vorkommen, die Einfluß auf den Zeitverbrauch haben und auf Erkennungskonflikte zurückzuführen sind.

FALL 1
Gleichzeitige Ereignisse sind für A und Si geplant, und der Koordinator ruft Si vor A auf. Die Änderung an Si würde die Eintragung von A in die Rufliste veranlassen. Jedocht liegt A aufgrund einer Zeitmeldung schon in der Rufliste. Der Parameter

h stellt die Rate der aufgrund von effektiven Zeitmeldungen (d.h. die keiner gelöschten Ereignisnotiz entsprechen) aufzurufenden Instanzen dar (d.h. % von E), für die sich zwei Aufrufgründe überlappen, nämlich die Zeitmeldung selbst und eine zu diesem Zeitpunkt stattgefundene Umgebungsänderung.

Damit verringert sich nach W - h die durchschnittliche Anzahl der aufgrund einer Änderung eines Instanzenausgangs in die Rufliste einzutragenden Instanzennamen (vgl. W im Abschnitt 4.6.2).

FALL 2
Gleichzeitige Ereignisnotizen sind für Si, A und Sj in dieser Reihenfolge vorhanden. A wird zweimal während derselben Systemüberführung in Aktion gebracht. Der erste

Aufruf geschieht aufgrund einer Zeitmeldung und einer Änderung an Si, der zweite Aufruf wegen einer Änderung an Sj. Zweimal merkt die Gatterinstanz eine Eingangsänderung und muß ihre logische Funktion auswerten. Wir definieren

i als die Rate der aufgrund von effektiven Zeitmeldungen aufzurufenden Instanzen (% von E), die ein zweites Mal zum selben Modellzeitpunkt aktiviert werden und für die in beiden Aufrufen Eingangsänderungen stattgefunden haben, von denen aber der erste Aufruf keine Auswirkung (Eintragung oder Löschen einer Ereignisnotiz) hat,

j wie i, wobei jetzt aber der erste Aufruf eine Auswirkung hat, und

b als die Wahrscheinlichkeit, daß wir die Auswirkung des ersten Aufrufes im zweiten Aufruf annullieren müssen.

Wir werden sehen, daß die Parameter i, j und b keine große Bedeutung für unsere Analyse haben. Wir werden uns deswegen nicht bemühen, die Definitionen für mehrere gleichzeitige Eingangsänderungen auszuweiten. Ihre Wertebereiche sind auch nicht notwendig. Für b wäre ein Zusammenhang mit dem Parameter a feststellbar.

Aus den Definitionen geht leicht eine Wertehierarchie h >= i >= j hervor. Diese Werte hängen stark von p*N ab. Je größer der Wert von p*N, desto größer auch die Wahrscheinlichkeit, daß gleichzeitige Ereignisnotizen für den Ausgang und einen oder mehrere Eingänge derselben Instanz vorhanden sind. Der Wert von h kann erst ab E = 2 größer als Null sein, und die Werte von i und j erst ab E = 3. Es ist dann zu erwarten, daß

h ≠ 0 ab E = E1 (>> 2),
i ≠ 0 ab E = E2 > E1, und
j ≠ 0 ab E = E3 > E2.

Wir haben uns für einen Wertebereich

h = 0 .. 0,2

willkürlich entschieden.

4.6.6 Andere Parameter

Wir stellen hier drei Parameter zusammen, die mit der Anzahl der für ein Gatter geplanten Ereignisse zu tun haben. Wir definieren

f als die Rate aller aufgrund von Umgebungsänderungen aufgerufenen Instanzen (% von W*E), die mindestens ein geplantes Ereignis für einen späteren Modellzeitpunkt haben,

g = 0 .. 1 als die Rate aller ausgeführten Ereignisnotizen (% von E), die die einzigen Notizen für die entsprechenden Gatter waren (dieser Parameter hat keinen Sinn für das Zeitmodell G1, weil dort immer g=1 gilt), und

d als eine Rate von E, so daß d*E die durchschnittliche Anzahl der Instanzen ist, die zu einem Modellzeitpunkt aufgrund einer Zeitmeldung aufgerufen werden und die die entsprechenden Ereignisnotizen in ihren internen Zuständen gelöscht, aber andere zukünftige Ereignisse schon geplant

hatten. Dementsprechend ist d ein Anteil von c, und so hängt sein Wertebereich mit dem von c zusammen.

Eine eindeutige Beziehung zwischen den Parameter f und c ist feststellbar, so daß wir den Wertebereich für f nicht willkürlich festlegen können.

4.6.7 Abgeleitete Parameter für die Instanzennetzmodellierung

Weil der Koordinator Gatterinstanzen aus mehreren Gründen in Aktion bringen kann, diese Gründe sich in einem Aufruf überlappen können, und eine eindeutige Zuordnung jeder Instanzenprozedursektion zu einem dieser Gründe unmöglich ist, ist es für die Übersichtlichkeit der Zeitverbrauchsausdrücke sinnvoll, Parameter einzuführen, die sich als Summen von elementaren Gliedern erklären lassen und eine einfache und eindeutige Zuordnung zu den Instanzenprozedursektionen erlauben. Wir definieren

k als die durchschnittliche Rate der Gatterinstanzen, die zu einem Modellzeitpunkt aufgerufen werden,

l als die durchschnittliche Rate der Gatterinstanzen, die zu einem Modellzeitpunkt aufgerufen werden und dazu mindestens ein geplantes Ereignis haben, und

m als die durchschnittliche Rate der Gatterinstanzen, die zu einem Modellzeitpunkt aufgrund von Umgebungsänderungen aufgerufen werden.

Aufgrund dieser Definitionen und der schon eingeführten Parameter leiten wir folgende Ausdrücke für diese Hilfsparameter ab:

$$k*N = E*(1 + c + W - h), \qquad (4.5)$$

$$l*N = E*(1 + d + (W - h)*f) \quad \text{und} \qquad (4.6)$$

$$m*N = E*(W + i). \qquad (4.7)$$

Für den Ausdruck (4.6) haben wir angenommen, daß f sowohl eine Rate von E*W sein darf , wie aus der Definition, als auch von E*(W-h), weil kein offensichtilicher Zusammenhang zwischen f und h besteht.

4.6.8 Gleichungen der Systemdynamik

Nach den Parameterdefinitionen wissen wir, daß zu jedem Modellzeitpunkt im Durchschnitt

E*W Gatter von einer Eingangsänderung betroffen werden, und

a*E*W Gatter aufgrund dieser Änderungen einen neuen Ausgangswert haben, der sich vom aktuellen Wert (bzw. vom Wert in der letzten Ereignisnotiz) unterscheidet. Diese a*E*W Gatter müssen für

- die Eintragung von E effektiven Ereignisnotizen,
- die Eintragung von c*E Notizen, die später gelöscht werden, und
- das Löschen von c*E Notizen

verantwortlich sein. Daraus folgt die Gleichheit

$$a*E*W = E + c*E + c*E,$$

die wir vereinfachen können, und es ergibt sich

$$a*W = 1 + 2*c \quad . \tag{4.8}$$

Diese Gleichung gilt für beide Zeitmodelle. Der Wertebereich von c ist daraus ableitbar. Weil per Definition $a =< 1$ ist, gilt

$$1 + 2*c =< W,$$

und daraus folgt

$$c =< (W-1)/2 \quad . \tag{4.9}$$

Wenn zum Beispiel W = 2, haben wir $c =< 0,5$, d.h. jede E Ereignisse sind an 2*E Gattereingängen beobachtbar, und damit diese Ereignisse die Planung von mindestens E neuen effektiven Ereignissen verursachen (sonst würde die Ereignisaktivität im Laufe der Zeit abschwächen), dürfen höchstens 0,5 * E zu löschende Notizen eingetragen und entsprechend 0,5 * E Notizen gelöscht werden.

Eine zweite Gleichung können wir für das Zeitmodell G1 feststellen. Dieses Modell verlangt, daß eine Notiz gelöscht werden muß, wenn eine Eingangsänderung für das entsprechende Gatter stattfindet, die eine Ausgangsänderung bewirkt. Wir können diese Eigenschaft folgendermaßen zum Ausdruck bringen:

E*W Gatter haben im Durchschnitt zu jedem Modellzeitpunkt eine Eingangsänderung,
f*E*W Gatter haben diese Änderung und dazu ein geplantes Ereignis für einen späteren Zeitpunkt, und
a*f*E*W Gatter ändern ihre Ausgänge aufgrund dieser Eingangsänderung, so daß wir das Ereignis für ungültig erklären müssen.

Da wir c*E Notizen per Definition zu jedem Zeitpunkt löschen, muß

$$a*f*E*W = c*E$$

gelten, und daraus folgt

$$c = a*f*W \quad . \tag{4.10}$$

Aus (4.8) und (4.10) leiten wir

$$f = c/(1+2*c) \tag{4.11}$$

ab. Wir brauchen die Wertebereiche für c und f nicht festzulegen, weil wir ihre Werte eindeutig durch (4.9) und (4.11) berechnen.

Für das Zeitmodell G2 gilt (4.11) nicht, aber es muß

$$c*E =< f*E*W$$

gelten, d.h., nur diejenige Gatter, die aufgrund einer Umge-

bungsänderung aufgerufen werden und mindestens eine Notiz haben, können das Löschen einer Notiz veranlassen. Es muß dann

$$f >= c/W \qquad (4.12)$$

gelten.

4.6.9 Grundsystem

Für die Untersuchung der Sensibilität des Simulationszeitgewinnes vom Gatter- gegenüber dem Instanzennetzsimulator bezüglich der Parameter brauchen wir ein Grundsystem als Ausgangspunkt. Wir haben uns für ein System mit langen Abständen zwischen den Signaländerungen entschieden, so daß das Löschen von Ereignisnotizen aufgrund des Tiefpassverhaltens (im Modell G1) oder des Unterschiedes zwischen minimaler und maximaler Verzögerung (im Modell G2) nicht nötig ist. Außerdem nehmen wir eine gleichmäßige Zeitverteilung der Ereignisse mit einem willkürlich ausgewählten Wert p*N = 4 an. Das Grundsystem hat dann folgenden Satz von Parameterwerten, der die festgestellten Beziehungen erfüllt:

N = 400, p = 0,01 (p*N = 4)	(vgl. Abs. 4.6.2)
X = 0, Xc = 0, Ec nicht definierbar	(vgl. Abs. 4.6.4)
q = 1, Me = Ms = 0 (optimale Werte für p*N = 4)	(vgl. Abs. 4.6.3)
Z1 = 5, Z2 = 15, Y = 2,3, U = 2,5, a = 0,5	(vgl. Abs. 4.6.1)
W = 2, c = 0	(vgl. Abs. 4.6.2)
h = i = j = 0, b nicht definierbar	(vgl. Abs. 4.6.5)
f = 0, d = 0, g = 1	(vgl. Abs. 4.6.6)

4.7 Zeitverbrauchsausdrücke

Um vergleichbare Zeitmessungen zu haben, müssen wir ein **Meßintervall** definieren, für dessen Grenzpunkte die Gatter- und die Instanzennetzsimulation identische Zustände zeigen. Wir definieren deswegen das Ende eines Intervalls (und gleichzeitig den Anfang des nächsten) als den Simulationszeitpunkt, zu dem alle einem Modellzeitpunkt entsprechenden Aktivitäten schon durchgeführt worden sind und die Modellzeitfortschaltung noch nicht angefangen hat.

Da die zu vergleichenden Simulatoren verschiedene Zeitfortschaltungsmechanismen verwenden, ist jedoch eine explizitere Definition erforderlich. Das Meßintervall muß immer genau eine Modellzeit erfassen, die mindestens eine nicht gelöschte Ereignisnotiz hat. Für den Gattersimulator definieren wir damit ein eindeutiges Simulationsintervall, weil der Simulator gelöschte Notizen erkennt und den nächsten Zeitschlitz mit effektiven Notizen immer findet. Für den Instanzennetzsimulator würde diese Definition aber bedeuten, daß einem Meßintervall mehrere Modellzeiten entsprechen könnten, weil der Koordinator auch Instanzen aufruft, für die (in ihren internen Zuständen) die

Ereignisnotizen gelöscht sind. Da wir aber

- für Systeme mit p*N < 1 nur eine teilweise Analyse machen, für die c = 0 immer gilt, lediglich um den Nachteil des "Fixed-Step"-Mechanismus auf diesem Bereich zu zeigen, und

- für Systeme mit p*N > 1 nur gleichmäßige Ereigniszeitverteilungen betrachten, so daß alle Modellzeiten mindestens eine effektive Ereignisnotiz haben,

brauchen wir uns nicht mit dieser Unangemessenheit der Definition zu befassen. Die Definitionen E=1 und D=1 für p*N < 1 bzw. (c+1)*p*N < 1 sind dann zweckmäßig, weil das Meßintervall genau eine Ereignisnotiz hat, falls p*N < 1.

4.7.1 Zeitverbrauch des Gattersimulators

Die erste Spalte der Tabelle 4.6 zeigt die Verarbeitungssequenz des Gattersimulators (siehe Abb. 4.3) während des Meßintervalls. Eine Bemerkung ist zu dieser Tabelle zu machen: Für die Auswertung einer Gatterfunktion, beispielweise

```
NEWVALUE <- SIGNAL [ FANINLISTE [ P ]] AND
            SIGNAL [ FANINLISTE [ P+1 ]] ,
```

teilen wir den Zeitverbrauch in zwei Faktoren c.1 und c.2 auf. Den Zeitverbrauch des Zugriffes auf die Eingangswerte können wir mit Hilfe der im Abschnitt 3.4 eingeführten Zugriffszeit Ta bestimmen, und er ist in diesem Fall gleich 2*Ta. Damit umfaßt der Parameter Z nur den Zeitverbrauch der Zuweisung an NEWVALUE und der Auswertung der logischen Funktion.

Der Algorithmus des Gattersimulators ist in Sektionen aufgeteilt, denen Ausführungszeiten T51 bis T63 zugeordnet sind (siehe Abb. 4.3 und 4.4). Die Festlegung dieser Sektionen ist für die definierten Parameter zweckmäßig. Punkte b.1, c.3, c.4 und c.5 der Tabelle 4.6 sind zeitmodellabhängig und haben entsprechende Ausführungszeiten

T64 (Aktualisierung von SIGNAL und EVENTP) und T65 bis T67 für das Zeitmodell G1 (Abb. 4.6), und

T68 bis T74 für das Zeitmodell G2 (Abb. 4.9 und 4.10).

Gemäß den Definitionen der Parameter und der Sektionen resultiert der in der zweiten Spalte der Tabelle 4.6 gezeigte Zeitverbrauch. Die Summe dieser Teilzeiten ergibt den gesamten Zeitverbrauch ZVS(G1) (Zeitmodell G1) bzw. ZVS(G2) (Zeitmodell G2) der Gattersimulation. Dafür werden die Zeiten T durch die Werte ersetzt, die wir für sie unter Berücksichtigung der Ausführungszeiten der Algorithmenschritte bekommen. Wir haben die Ausdrücke unter der Voraussetzung

D = E*(1+c)

vereinfacht, die für die uns interessierenden Fällen p*N > 1 und p*N < 1, c = 0 gilt.

Aktivitätsbeschreibung	Zeitverbrauch
a) Modellzeitfortschaltung bis zum Zeitschlitz, der mindestens eine effektive Ereignisnotiz hat	T51 + T52*X + T52*Xc + T53*Ec*Xc
b) Für jede in diesem Zeitschlitz vorhandene effektive Ereignisnotiz	
b.1) Ausführung der Ereignisnotiz	G1: T64*E G2: T68*E + T69*g*E
b.2) Zugriff auf den Anfang der Teil-FANOUTLISTE dieses Gatters	T54*E
b.3) Durchlauf der FANOUTLISTE	T55*U*E
b.4) Eintragung der Namen der Gatter, die FLAG=FALSE haben, in die RL	T56*W*E
b.5) Zugriff auf die nächste Notiz, falls vorhanden	T57*(D-1) + T58
c) FÜr jedes Gatter, dessen Name in der Rufliste eingetragen ist	
c.1) Zugriff auf die Werte der Eingangssignale durch die FANINLISTE	Ta*Y*W*E
c.2) Auswertung der logischen Funktion	Z*W*E
c.3) Modellabhängiger Test für die Entscheidung, ob Eintragung oder Löschen einer Ereignisnotiz notwendig ist	G1: T65*W*E G2: T70*W*E + T71*f*W*E + T73*(W*E - c*E)
c.4) ggf. Eintragung einer Notiz	G1: (T66 + T61)*D + T62 + T63*(D-1) G2: (T74 + T61)*D + T62 + T63*(D-1)
c.5) ggf. Löschen einer Notiz	G1: T67*c*E G2: T72*c*E
c.6) Zugriff auf den nächsten Gatternamen, falls vorhanden	T59 + T60*W*E

Tabelle 4.6 - Verarbeitungssequenz des Gattersimulators in einem Meßintervall

$$ZVS(G1) = 28 + 27*X + Xc*(27 + 32*Ec) + E*(131 + 16*U + W*(49 + Z1 + 5*Y) + 115*c) \quad (4.13)$$

$$ZVS(G2) = 28 + 27*X + Xc*(27 + 32*Ec) + E*(150 + 16*U + W*(49 + Z2 + 5*Y + 10*f) + 9*g + 117*c) \quad (4.14)$$

4.7.2 Zeitverbrauch der Instanzennetzmodellierung

Tabelle 4.7 zeigt die Verarbeitungssequenz des Koordinators des Instanzennetzsimulators während eines Meßintervalls. Die den Sektionen des Koordinators entsprechenden Ausführungszeiten kennen wir schon aus dem Abschnitt 3.4. Aufgrund der Definitionen der Parameter und der Sektionen bekommen wir den in der zweiten Spalte der Tabelle gezeigten analytischen Zeitverbrauch.

Aktivitätsbeschreibung	Zeitverbrauch
a) Zeitfortschaltung	
a.1) Falls die nächste Notiz eine Scheinnotiz ist, Entfernung und neue Eintragung	T8*q + T9*Ms*q
a.2) Erhöhung der Modellzeit und Vorbesetzung der Pointers auf die Rufliste	T7
a.3) Eintragung der Namen der Gatter, die eine Notiz für die neue Zeit haben, in die RL	T10*D
b) Verarbeitung der Rufliste für alle aufzurufenden Gatterinstanzen (Hauptschleife)	T1*k*N
c) Eintragung von Notizen in die EL für die Instanzen, die eine Zeitmeldung abgeben	T2*D + T3*Me*D
d) Für jede Gatterinstanz, die eine Umgebungsänderung gemeldet hat	
d.1) Zugriff auf den Anfang der Teil-UMGLISTE für diese Instanz	T4*E
d.2) Durchlauf der UMGLISTE	T5*U*E
d.3) Eintragung der Namen der Instanzen, die FLAG = FALSE haben, in die RL	T6*(W-h)*E

Tabelle 4.7 - Verarbeitungssequenz für den Koordinator des Instanzennetzsimulators in einem Meßintervall bei der Gatterebenen-Simulation

Außerdem müssen wir die Durchführung aller in einem Meßintervall aufgerufenen Gatterinstanzenprozeduren betrachten. Die Verarbeitungssequenz in den Gatterinstanzen ist in der Tabelle 4.8 gezeigt. Da die Zeitmodelle G1 und G2 unterschiedliche Instanzen verlangen, haben wir die Sektionausführungszeiten
T20 bis T29 (Abb. 4.5) und T30 bis T31 (Abb. 4.7) für das Modell G1, und
T23 bis T26 und T32 bis T42 (Abb. 4.12) für das Modell G2.
Aus den Definitionen der Parameter und der Sektionen ergibt sich der Zeitverbrauch, der in der zweiten (Modell G1) bzw. dritten Spalte (Modell G2) der Tabelle 4.8 angegeben ist.

Es ist zu beachten, daß die wirklich benutzten Ausführungszeiten Ti nicht direkt den in der Arbeit gezeigten Ablaufdiagrammen zu entnehmen sind, weil diese Diagramme den direkten Zugriff auf die Anschluß- und Zustandsvariablen nicht verdeutlichen (siehe Abschnitt 3.3).

Die Summe aller Teilzeiten der Tabelle ergibt den gesamten Simulationszeitverbrauch ZVI(G1) (Modell G1) bzw. ZVI(G2) (Modell G2) für ein Meßintervall, wobei wir die Vereinfachung D=E*(1+c) nochmals ausgenutzt haben.

```
ZVI(G1) = 20 + q*( 116 + 15*Ms ) +
   + E*( 305 + 16*U + W*( 104 - 4,5*Y + 9*f ) + 287*c +
       + 17*Y*(1+c+W-h) + 15*Me*(1+c) + (8*Y + Z1)*(W+i) +
       + j*(5 + 14*b) + 21*i - h*(83 + 9*f) + 9*d )    (4.15)

ZVI(G2) = 20 + q*( 116 + 15*Ms ) +
   + E*( 343 + 16*U + W*( 143 - 4,5*Y + + 22*f ) + 309*c +
       + 17*Y*(1+c+W-h) + 15*Me*(1+c) + (8*Y + Z2)*(W+i) +
       + j*(32 + 17*b) + (i-j)*(35 + 14*f) - h*(108 + 8*f) +
       + 8*d )                                          (4.16)
```

4.8 Vergleich zwischen dem Gattersimulator für das Zeitmodell G1 und der entsprechenden Instanzennetzmodellierung

Für unseren Vergleich suchen wir das Verhältnis

$$R = \frac{ZVS\ (G1)}{ZVI\ (G1)} .$$

Der Wert R ist ein Maßstab für den Simulationszeitgewinn des Gattersimulators gegenüber dem Instanzennetzsimulator. Ein Wert R = 0,4 beispielweise bedeutet, daß der Gattersimulator nur 40% der Simulationszeit braucht, die der Instanzennetzsimulator für dasselbe System verbraucht.

Absicht dieser Untersuchung ist es, die Sätze von Parameterwerten für zwei Systeme S2 und S3 herauszufinden, die für den Gattersimulator den größten bzw. den kleinsten Zeitgewinn gegenüber dem Instanzennetzsimulator aufweisen, d.h.

Aktivitätsbeschreibung	Zeitverbrauch für G1	Zeitverbrauch für G2
e) Testen, ob die Instanz ein Ereignis für diese Modellzeit hat	T20*k*N + T21*l*N	T32*k*N + T33*l*N
f) Ausführung der Ereignisnotiz	T22*E	T34*E
g) Zugriff auf die Eingangsvariablen	T23*Y*k*N	
h) Testen, ob eine Eingangsänderung stattgefunden hat	T24*Y*E*(1+c-h) + (T24/2)*Y*W*E	
i) Eingangsänderung stattgefunden		
i.1) Zustandsvariablen aktualisieren	T25*Y*m*N	
i.2) Auswertung der logischen Funktion	Z1*m*N	Z2*m*N
i.3) Testen, ob die Instanz zu diesem Zeitpunkt eine Ereignisnotiz schon eingetragen oder gelöscht hat	T26*m*N	
i.4) Falls nicht		
i.4.1) Testen, ob Eintragung oder Löschen einer Notiz notwendig ist	T30*(m*N-j*E)	(m*N-j*E)*(T35 + T36*(1-f) + T37*f + T38*f) -T38*c*E
i.4.2) Eintragung einer Notiz (im internen Zustand)	T27*D	T39*D
i.4.3) Löschen einer Notiz (im internen Zustand)	T28*c*E	T40*c*E
i.5) Falls schon		
i.5.1) Testen, ob Eintragung bzw. Löschen annulliert werden muß	T31*j*E	T41*j*E
i.5.2) Annullierung der Eintragung bzw. des Löschens	T29*b*j*E	T42*b*j*E

Tabelle 4.8 - Verarbeitungssequenz für alle während eines Meßintervalls aufgerufenen Gatterinstanzen

die den minimalen bzw. den maximalen Wert von R liefern. Eine ausführlichere Erklärung der Gründe für die mit diesen Systemen S2 und S3 verbundenen Werte von R erfolgt erst in Kapitel 6.

4.8.1 Sensibilität von R in Bezug auf die Parameter h, i und j

Diese drei Parameter kennzeichnen die Behandlung der Auswirkung von Erkennungskonflikten durch die Gatterinstanzen und tauchen deswegen nur im Ausdruck ZVI auf. Wir nehmen an, daß wir einen Wert Ro für h=i=j=0 erhalten. Damit wir einen Wert R < Ro erreichen, ist es notwendig, daß

$$j*(5 + 14*b) + i*(21 + 8*Y + Z1) - h*(83 + 9*f + 17*Y) > 0 , \quad (4.17)$$

d.h. daß die Summe aller in ZVI(G1) von h, i und j abhängigen Glieder größer als Null sein muß. Wir sehen, daß h > 0 die Instanzennetzmodellierung begünstigt, weil wir Aufrufe aufgrund von Umgebungsänderungen sparen, während i>0 und j>0 dem Gattersimulator zugute kommen, weil diese Werte bedeuten, daß Gatterinstanzen mehrmals die Auswirkung gleichzeitiger Eingangsänderungen betrachten.

Um die Erfüllung von (4.17) zu erreichen, müssen wir die größten erlaubten Werte für b (=1) und Z1 (=15) und den kleinsten erlaubten Wert für f (=0) nehmen. Y wird sowohl mit i als auch mit h multipliziert, aber weil i =< h immer gilt, muß Y*(8*i - 17*h) immer < 0 sein, so daß wir den kleinsten erlaubten Wert Y = 1,7 nehmen. Damit bekommen wir aus (4.17)

$$19*j + 48*i - 108,5*h > 0 . \quad (4.18)$$

Da j =< i, versuchen wir (4.18) zu erfüllen, indem wir den größten Wert für j (=i) nehmen. Wir haben endlich

$$57*i - 108,5*h > 0, \text{ oder}$$
$$i > 1,9*h.$$

Wir können diese Ungleichung nicht erfüllen, weil die Definitionen i =< h gewährleisten. Wir schließen daraus, daß wir den kleinsten Wert für R, bezüglich der Werte von h, i und j, immer dann erreichen, wenn wir h = i = j = 0 haben.

4.8.2 Sensibilität von R bezüglich der anderen Parameter

Wir haben im Abschnitt 4.6.9 einen Satz von Parameterwerten für ein Grundsystem S1 ausgewählt. Unter Verwendung von diesem Satz finden wir

ZVS(G1) = 1236,
ZVI(G1) = 2921,6 und
Rg(S1) = 42,31 %,

wo Rg(S1) der Grundwert von R für das System S1 ist. Wir müssen jetzt für jeden Parameter herausfinden, welche Tendenz R zeigt, wenn wir diesen Parameter ändern. Dafür vergleichen wir Rg mit einem Grenzwert von R, den wir erreichen, wenn der untersuchte Parameter den Wert plus unendlich annimmt (bzw. Null, wenn damit der Grenzwert von R leichter auffindbar ist). Wir definieren

$$Ru(P) = \lim_{P \to \infty} R \quad \text{und}$$

$$Ro(P) = \lim_{P \to 0} R \ .$$

Wenn beispielweise Ru(P) größer als Rg(Si) ist und wir den Wert von R ab diesem Arbeitspunkt Si aufgrund einer Wertänderung von P erhöhen möchten, müssen wir einen Wert von P nehmen, der größer ist als der im Arbeitspunkt Si festgelegte Wert von P.

Diese Art von Analyse ist hier möglich, weil wir für jeden Parameter P

$$R = \frac{K1 + K2*P}{K3 + K4*P}$$

schreiben können, wobei K1,...,K4 Konstanten sind, deren Werte von den übrigen Parametern abhängen.

Wir versuchen festzustellen, ob diese Tendenz von R bezüglich einer Variation von P sich für alle anderen möglichen Arbeitspunkte verallgemeinern läßt. Dies ist möglich, indem wir Ru(P) oder Ro(P) mit dem (später) erzielten Wertebereich von R vergleichen. Wir dürfen schon im voraus erwähnen, daß der Wert von R für beliebige Werte der Parameter zwischen 40% und 50% liegen wird.

Für die ganze Untersuchung nehmen wir Xc=0 an, so daß nur gleichmäßige Ereignisverteilungen in Betracht kommen, und h=i=j=0, weil wir hauptsächlich das Minimum von R suchen, d.h. das System, das am ungüstigsten für die Instanzennetzsimulation ist.

Im folgenden führen wir für jeden Parameter, der den Zeitverbrauch der Gatter- und/oder der Instanzennetzsimulation beeinflußt, eine Untersuchung der Sensibilität von R bezüglich dieses Parameters durch.

Untersuchung U1: E (=p*N)

Wir schreiben R als eine Funktion von p*N. Da die für die Zeitfortschaltung kennzeichnenden Parameter q, Me, Ms und X abhängig von p*N sind, müssen wir sie auch als Variablen betrachten. Die übrigen Parameter übernehmen ihre Werte aus S1. Wir finden

$$R(p*N) = \frac{28 + 27*X + 302*E}{20 + q*(116 + 15*Ms) + E*(696{,}4 + 15*Me)} \ . \qquad (4.19)$$

Ein lineares uniformes Verhalten von R bezüglich des Wertes von p*N gibt es hier nicht, wegen der streckenweise Abhängigkeit der Parameter q, Me und Ms von p*N. Deswegen ist eine Analyse aufgrund von Ru(p*N) oder Ro(p*N) unmöglich. Wir berechnen R(p*N) für verschiedene Werte von p*N und versuchen das Verhalten von R(p*N) dadurch festzustellen. Für X, q, Me und Ms gelten:

X=0 falls p*N >= 1, sonst ist sein Wert der Gl. 4.4 zu entnehmen,

q=1/5 falls p*N < 1, sonst ist sein Wert der Tab. 4.5 zu entenhmen, und

Me=Ms=1 falls p*N < 1, sonst sind ihre Werte der Gl. 4.1 zu entnehmen.

Tabelle 4.9 stellt die Ergebnisse zusammen. Sie bestätigt unsere Behauptung, daß der "Fixed-Step"-Mechanismus erst dann für den Gattersimulator vorteilhaft ist, wenn p*N > 1 ist. Wir begrenzen deswegen unsere Analyse auf p*N >= 1. Aus der Tabelle ist auch sichtbar, daß R ein Minimum bei E=4 hat (wir beschränken uns auf ganzzahlige Werte für p*N).

p*N	E	X	q	Me=Ms	R
0,1	1	9	1/5	1	75,6 %
0,2	1	4	1/5	1	57,8 %
0,5	1	1	1/5	1	47,1 %
1	1	0	1/5	1	43,6 %
2	2	0	1/3	1	42,52 %
3	3	0	1/2	3/4	42,33 %
4	4	0	1	0	42,31 %
5	5	0	1	0	42,51 %
8	8	0	1	0	42,8 %
16	16	0	1	0	43,1 %

$$R = \frac{ZVS(G1)}{ZVI(G1)}$$

Tabelle 4.9 - Sensibilität von R bezüglich des Parameters p*N

Untersuchung U2: U

Ru(U) = 100 %, d.h., wir würden die ganze Simulationszeit bei dem Zugriff auf die UMG- bzw. FANOUTLISTE verbrauchen, falls U den Wert unendlich hätte, und dies ist eine beiden Simulatoren gemeinsame Aktivität. Da der Wert von R nicht wesentlich über 50% gebracht werden kann und Ru(U) eine Konstante ist, gilt Ru(U) > R für beliebige Werte der anderen Parameter. Deswegen erreichen wir das Minimum für R unter Verwendung des kleinst möglichen Wertes von U.

Untersuchung U3: Z1

Ru(Z1) = 100 %, d.h., beide zu vergleichende Simulatoren verbrauchen die gleiche Zeit für die Auswertung der logischen Funktion eines Gatters. Die Folgerung für die Tendenz von R bezüglich des Wertes von U gilt auch für Z1.

Untersuchung U4: Y

$$Ru(Y) = \frac{5*W}{20,5*W + 17*(1+c)} \qquad (4.20)$$

Weil wir einen Wertebereich W = 1,5 .. 3 festgelegt haben, ergibt sich für c=0 Ru(Y) = 15,7% .. 19,1%. Falls c ungleich Null wäre, würden wir einen noch kleineren Wert von Ru(Y) bekommen. Da wir R nicht wesentlich unter 40% bringen können, gilt immer R > Ru(Y). Das Minimum von R wird mit dem größten Wert von Y erreicht.

Untersuchung U5: W

$$Ru(W) = \frac{49 + Z1 + 5*Y}{104 + Z1 + 20,5*Y} \qquad (4.21)$$

Für die Parameterwerte von S1 haben wir Ru(W) = 41,95 % < Rg(S1). Eine allgemeingültige Aussage über die Sensibilität von R in Bezug auf den Wert von W ist aber nicht machbar, weil Ru(W) um den Wertebereich von R liegt und von Z1 und Y abhängig ist.

Untersuchung U6: d

Der Parameter d taucht nur in ZVI(G1) auf, so daß wir das Minimum von R mit dem größten Wert von d erreichen. Diesen Wert können wir jedoch nicht willkürlich festlegen, weil die Definition d =< c verlangt.

Untersuchung U7: f

Wie d taucht f nur in ZVI(G1) auf. Sein Wert hängt auch von c gemäß der Gleichung 4.11 ab.

Untersuchung U8: c

$$Ru(c) = \frac{115}{326,1 + 15*Me} \qquad (4.22)$$

Da der maximale Wert von Ru(c) 35,3 % (Wert für Me=0) ist und R für beliebige Werte der Parameter nicht wesentlich unter 40% gebracht werden kann, haben wir immer Ru(c) < R. Der größte Wert von c, der jedoch von W gemäß (4.9) abhängt, bringt uns den kleinsten Wert von R. Da ein großer Wert von c große Werte für d und f mit sich bringt, können die drei Anforderungen für einen minimalen Wert von R, nämlich maximale Werte für c, d und f, gleichzeitig erfüllt werden.

Der Ausdruck (4.22) gilt nur, weil wir immer Xc=0 annehmen. Ungleichmäßige Ereignisverteilungen würden Xc ungleich Null ergeben, falls c ungleich Null und der Wert von E klein wäre. Daraus würde einen nichtlinearen Ausdruck für R(c) folgen. Wir hätten dann, wie für die Untersuchung U1, das Verhalten von R in Bezug auf Variationen von c durch die Auswertung von R(c) für verschiedene Werte von c feststellen müssen.

4.8.3 Maximaler Gewinn für den Gattersimulator

In diesem Abschnitt suchen wir den Satz von Parameterwerten für das System S2, das den kleinsten Wert von R liefert, d.h., für das der Gattersimulator den höchsten Effizienzgewinn gegenüber der Instanzennetzmodellierung aufweist. Aufgrund der Ergebnisse der Untersuchungen U1-U8 können wir behaupten, daß für S2

E=4 sein muß (und damit X=0, q=1, Me=Ms=0), wobei unerheblich ist, welche Werte p und N haben,
U und Z1 minimale Werte, und
Y, c, d und f maximale Werte haben müssen.

Eine Aussage über den Wert von W können wir noch nicht machen. Da der Wert des Parameters c von W abhängt, der Wert von d kleiner als c sein muß und der Wert von f abhängig von c ist, übernehmen wir vorerst die Werte aus S1
W=2, c=0, d=0 und f=0.

Die Parameter h, i und j müssen den Wert Null haben, wie wir schon bewiesen haben. Für die anderen Parameter legen wir gemäß den entsprechenden Wertebereichen folgende Werte fest:
U=2, Z1=4 und Y=2,7.

Für diesen neuen Satz S2' von Parameterwerten bekommen wir
Rg(S2') = 40,75 %.

Es ist jetzt notwendig, die Sensibilität von R bezüglich des Parameters W ab dem neuen Arbeitspunkt S2' zu untersuchen. Der Ausdruck (4.20) für Ru(W) gilt immer noch, und daraus bekommen wir
Ru(W) = 40,71 % < Rg(S2').
Wir sollten W erhöhen, um R zu erniedrigen. Das steht aber im Gegensatz zur gewünschten Verminderung von U, weil U>=W sein muß. Die sehr nahe beieinander gelegenen Rg(S2') und Ru(W) besagen aber, daß der Einfluß der Erhöhung von W auf R für S2 sehr gering ist. Stattdessen haben wir Ru(U) = 100%, so daß ein kleiner Wert von U mehr zu unserem Ziel beiträgt als ein großer Wert von W. Wir legen deshalb für W den größten Wert fest, der durch U=2 erlaubt ist, nämlich
W=2.

Damit können wir auch die Werte für c, d und f festlegen. Aus (4.9) nehmen wir c = 0,5, und aus (4.11) folgt dann der Wert 0,25 für f. Da d =< c gelten muß, nehmen wir d = 0,5.

Der Satz von Parameterwerten für S2 ist jetzt komplett, und mit ihm bekommen wir
Rg(S2) = 39,23 %.

4.8.4 Minimaler Gewinn für den Gattersimulator

Aufgrund der Untersuchungen U1-U8 und der Kenntnis über h, i und j wissen wir, daß für das System S3, das den minimalen Zeitgewinn für den Gattersimulator gegenüber der Instanzen-

netzmodellierung aufweist,
U, Z1 und h maximale Werte, und
Y, i, j, c, d und f minimale Werte haben müssen.

Eine Aussage über W können wir noch nicht machen. Am Anfang übernehmen wir den Wert W=2 aus S1.

Das in der Tabelle 4.9 gezeigte Verhalten von R(E) läßt uns nicht erkennen, mit welchem Wert von E wir den maximalen Wert von R erreichen. Sowohl E=1 als auch der maximale Wert von E kommen in Frage. Wir führen deswegen zwei Versuche durch, einmal für E=1, einmal für E=15. Wir merken schon, daß der Wert E=1 im Widerspruch zur Anforderung nach einen möglichst großen Wert für h steht, weil h ≠ 0 erst ab E=2 sein kann. Es ist vorauszusehen, daß der Wert E=15 vorteilhafter für einen möglichst großen Wert von R ist.

Für die übrigen Parameter legen wir folgende Werte fest, die von dem Wert von E unabhängig sind:
U=3, Z1=10, Y=1,7,
i=0, j=0,
c=0, d=0 und f=0.

Versuch 1: E=1
Aus E=1 ergeben sich X=0, q=1/5, Me=Ms=1, h=0. Damit bekommen wir
Rg(S3') = 46,17 %.

Der Ausdruck (4.21) gilt wieder, und daraus finden wir
Ru(W) = 45,3 % < Rg(S3'),
Wir müssen W dann erniedrigen, um R zu erhöhen. Wir nehmen den kleinst möglichen Wert W=1,5.

Der Satz von Parameterwerten für S3' ist jetzt endgültig definiert, und mit ihm bekommen wir
Rg(S3') = 46,30 %.

Versuch 2: E=15
E=15 ergibt X=0, q=1 und Me=Ms=0. Da E wesentlich größer als 1 ist, können wir den maximalen Wert für h nehmen, nämlich 0,2. Wir finden
Rg(S3'') = 47,41 %,
und da Ru(W) = 45,3%, wie für den Versuch 1, müssen wir für W auch hier den kleinsten Wert 1,5 nehmen. Damit haben wir den endgültigen Satz für S3'', der
Rg(S3'') = 47,67 %
ergibt.

Der Versuch 2 hat einen größeren Wert für R geliefert als der Versuch 1, und dies ist sicherlich auf den Wert h=0,2 zurückzuführen. Wir wählen deswegen den Satz von Parameterwerten von S3'' für das gewünschte System S3 aus.

4.8.5 Sensibilität von R bezüglich der Ausführungszeiten der Algorithmenschritte

In diesem Abschnitt führen wir eine Untersuchung der Sensibilität von R in Bezug auf die in Kapitel 2 eingeführten Ausführungszeiten S der elementaren Aktivitäten der algorithmischen Ebene. Ziel dieser Untersuchung ist es, die Werte der einzelnen Zeiten S so zu modifizieren, daß zwei Sätze A2 und A3 von Ausführungszeiten entstehen, die den kleinsten bzw. den größten Wert für R aufweisen. Wir versuchen diese Sätze so zu gestalten, daß sie den gewünschten Grenzwert für die drei hier gebildeten Systeme S1, S2 und S3 darstellen.

Im Gegensatz zur Suche der Sätze von Parameterwerten für S2 und S3, wo wir die Beziehungen zwischen den Parametern berücksichtigt haben, ändern wir jetzt die Ausführungszeiten unabhängig voneinander, weil wir sonst die Abbildung dieser Zeiten in maschinen- und sprachabhängigen Aktivitäten betrachten müßten.

Wie in Kapitel 2 schon erwähnt, haben wir uns für eine Veränderung von 50% für jede Ausführungszeit entschieden. Diese Veränderungen scheinen schon genügend übertrieben zu sein, um die Vermutung zu verstärken, daß A2 und A3 wirklich unerreichbare Grenzfälle seien.

Für jede Ausführungszeit Sij können wir

$$R = \frac{Gs\ (\text{Systemparameter und alle Skl, } k \neq i \text{ und } j \neq 1) + Sij * Fs\ (\text{Systemparameter})}{Gi\ (\text{Systemparameter und alle Skl, } k \neq i \text{ und } j \neq 1) + Sij * Fi\ (\text{Systemparameter})} \qquad (4.23)$$

schreiben, wobei Gs, Gi, Fs und Fi Funktionen mit den obengenannten Argumenten sind und Fs oder Fi Null sein können, falls Sij in dem Zeitverbrauchsausdruck für den Gatter- bzw. den Instanzennetzsimulator nicht auftaucht. In diesem Fall haben wir eine triviale Lösung für die Sensibilitätsuntersuchung. Ansonsten führen wir die Untersuchung wie bei der Suche nach S2 und S3 durch, d.h. wir leiten Ru(Sij) ab und vergleichen diesen Wert mit dem möglichen Wertebereich von R, der jetzt jedoch breiter ist als der vorher erreichte Wertebereich 39% (S2) bis 48% (S3). Für einige Ausführungszeiten S liefert ein analytischer Vergleich eine eindeutige Lösung, die die ganze Bandbreite aller Parameter umfaßt, während für andere wir Ru für S1, S2 und S3 berechnen müssen, weil der Vergleich zwischen Ru(Sij) und R von dem Arbeitspunkt abhängt.

Eine genaue Beschreibung der Untersuchungsschritte ist für uns unwichtig. Tabelle 4.10 faßt die Ergebnisse zusammen. Sie zeigt, praktisch unabhängig vom simulierten System, daß der Effizienzgewinn eines Gattersimulators gegenüber der entsprechenden Instanzennetzsimulation sich um nicht mehr als 20 % von seinem hier festgelegten Mittelwert nach oben

oder nach unten durch irgendeinen neuen Satz von Ausführungszeiten ändern läßt.

	Rg(S1) V	Rg(S2) V	Rg(S3) V
Satz A1	42,3 %	39,2 %	47,7 %
Satz A2	35,7 % -18,5 %	32,6 % -20,1 %	40,3 % -18,2 %
Satz A3	53,8 % +21,3 %	51,1 % +23,3 %	59,9 % +20,4 %

$$R = \frac{ZVS(G1)}{ZVI(G1)}$$

V = Prozentuelle Veränderung von R gegenüber seinem Wert für den Satz A1

Tabelle 4.10 - Sensibilität von R bezüglich der Ausführungszeiten der Algorithmenschritte

4.9 Vergleich zwischen dem Gattersimulator für das Zeitmodell G2 und der entsprechenden Instanzennetzmodellierung

Wir können für das Zeitmodell G2 eine Untersuchung durchführen, die sehr ähnlich mit der Untersuchung für G1 ist. Wir beschreiben deswegen die Einzelheiten hier nicht wieder, sondern beschränken uns auf die wesentlichen Ergebnisse. Das Verhältnis R ist jetzt als

$$R = \frac{ZVS(G2)}{ZVI(G2)}$$

definiert, wobei ZVS(G2) und ZVI(G2) den Ausdrücken (4.14) bzw. (4.16) im Abschnitt 4.7 zu entnehmen sind.

Aus der Untersuchung der Sensibilität von R in Bezug auf h, i und j ergibt sich, daß

$$j*(17*b - 14*f -3) + i*(35 + 14*f + 8*Y + Z2) - h*(108 + 8*f + 17*Y) > 0 \qquad (4.24)$$

sein muß, damit wir R < Ro erreichen, wobei Ro das mit h=i=j=0 erzielte Verhältnis ist. Die günstigsten erlaubten Werte der Parameter b, f, Z2 und Y liefern uns

$$14*j + 77*i - 135,5*h > 0. \qquad (4.25)$$

Damit diese Bedingung erfüllt ist, müssen wir für den günstigsten Fall j=i

$$i > 1,47*h$$

haben, was allerdings unmöglich ist. Das bedeutet, daß wir auch für G2 den minimalen Wert von R mit h=i=j=0 erreichen.

Der Satz von Parameterwerten für das Grundsystem S1 ergibt
ZVS(G2) = 1508,
ZVI(G2) = 3465,6 und
Rg(S1) = 43,51 %.
Wegen dem komplexeren Zeitmodell wurden sowohl ZVS als auch ZVI um ca. 20% von seinen Werten für G1 erhöht, aber Rg(S1) hat sich nicht wesentlich geändert.

Untersuchungen U1-U4, U6 und U8 bestätigen, daß
- der minimale Wert von R mit E=4 erreicht wird,
- Ru(U) = Ru(Z2) = 100 %,
- Ru(Y) = 15,7% .. 19,1% für c=0,
- d nur in ZVI auftaucht, und
- $Ru(c) = \frac{117}{348,1 + 15*Me} =< 33,6\%$,

so daß Ru(c) für beliebige Werte von Me kleiner als R ist.

Für W ergibt sich jetzt

$$Ru(W) = \frac{59 + Z2 + 5*Y + 10*f}{143 + Z2 + 20,5*Y + 22*f} , \qquad (4.26)$$

und für S1 haben wir Ru(W) = 41,7%. Wie für G1 liegt Ru(W) um den Wertebereich von R. Wegen der Abhängigkeit des Wertes Ru(W) von den Parametern Z2, Y und f könnten wir keine allgemeingültige Aussage über die Sensibilität von R in Bezug auf W machen.

Die Untersuchung U7 sieht nun anders aus, weil f auch in ZVS(G2) auftaucht. Wir bekommen

$$Ru(f) = \frac{10*W}{22*W - 8*h} >= 45,5\% \text{ (Wert für h=0).} \qquad (4.27)$$

Da Ru(f) um den Wertebereich von R liegt, können wir wie für W keine endgültige Aussage über die Tendenz von R bezüglich einer Variation von f machen.

Für das Zeitmodell G2 taucht noch der Parameter g auf. Wir führen deswegen eine zusätzliche Untersuchung U9 durch, die allerdings trivial ist, weil g nur in ZVS(G2) vorhanden ist, so daß wir den minimalen Wert von R unter Verwendung von g=0 erreichen.

Basierend auf den Ergebnissen dieser Untersuchungen suchen wir zwei Systeme S4 und S5, die den minimalen bzw. den maximalen Wert von R für das Zeitmodell G2 bringen. Wir können schon sagen, daß für S4

- E=4 sein muß (und damit X=0, q=1, Me=Ms=0),
- U, Z2, g, h, i und j minimale Werte und
- Y, c und d maximale Werte haben müssen.

Aussagen über W und f machen wir noch nicht, und da der Parameter c von W abhängt (und der Parameter d wiederum von c), übernehmen wir vorerst

W=2, c=0, d=0 und f=0

aus dem System S1. Für die übrigen Parameter legen wir gemäß ihren Wertebereichen folgende Werte fest:

U=2, Z2=10, Y=2,7 ,
g=0, h=0, i=0 und j=0.

Mit diesem partiellen Satz bekommen wir

Rg(S4') = 40,61 %.

Es besteht hierbei ein Kompromiß zwischen c und f, weil Ru(f) > Rg(S4), so daß f zu erniedrigen wäre. Da f >= c/W gelten muß, steht die Verminderung von f im Widerspruch zu einer gleichzeitigen Erhöhung von c. Wir haben für diesen Satz S4'

$$R(c,d,f) = \frac{1416 + 468*c + 80*f}{3486,4 + 1419,6*c + 176*f + 32*d} . \qquad (4.28)$$

Dieser Ausdruck hilft uns, den obengenannten Kompromiß zu schließen. Für W=2 haben wir

- falls c = maximaler Wert = 0,5, dann f = minimaler Wert = 0,25, d = maximaler wert = 0,5 und Rg(S4'') = 39,24 %, und
- falls f=0, dann c=0, d=0 und Rg(S4''') = 40,61 %.

Wir nehmen die erste Möglichkeit als unseren neuen Arbeitspunkt. Aus (4.26) haben wir dann

Ru(W) = 39,75 % > Rg(S4''),

so daß wir W erniedrigen sollten. Ein Wert von W kleiner als 2 würde aber nach (4.9) einen Wert von c kleiner als 0,5 verlangen und im Widerspruch zum festgelegten Wert c=0,5 stehen. Aufgrund der sehr ähnlichen Werte von Ru(W) und Rg(S4'') erkennen wir, daß der Einfluß der Verminderung von W auf R sehr gering und der Verlust wegen der kleineren Wert von c nicht auszugleichen wäre. Wir legen deswegen für S4 endgültig

W=2, c=0,5, f=0,25 und d=0,5

fest, was

Rg(S4) = 39,24 %

ergibt.

Für das System S5 müssen wir

- maximale Werte für U, Z2, g und h und
- minimale werte für Y, c, d, i und j haben.

Wie für G1 können wir zwei Versuche durchführen, einmal mit E=1 und einmal mit E=15, um den maximalen Wert für R festzustellen. Wiederum ist die Tatsache, daß h=0 für E=1 gelten muß, ausschlaggebend dafür, daß dieser Wert von R mit E=15 erreicht wird. Wir legen zunächst

U=3, Z2=30, Y=1,7,
c=0, d=0, g=1,
h=0,2, i=0 und j=0

fest, was

Rg(S5') = 49,42 %
bei verbleibenden W=2 und f=0 ergibt. Wir finden von diesem Punkt S5' aus
Ru(W) = 46,9 % (aus (4.26)) und
Ru(f) = 47,2 % (aus (4.27)),
so daß wir die niedrigsten Werte für W und f nehmen müssen. Damit ändert sich f nicht, während für W der Wert 1,5 genommen wird. Mit dem endgültigen Satz für S5 haben wir
Rg(S5) = 49,78 %.

4.10 Ergebnisinterpretation

Die wichtigste Schlußfolgerung dieses Kapitels können wir durch die Bandbreite des Effizienzgewinns des Gattersimulators gegenüber der Instanzennetzsimulation zusammenfassen:

	Zeitmodell G1	Zeitmodell G2
minimaler Wert von R	39,23 %	39,24 %
maximaler Wert von R	47,67 %	49,78 %

Diese Ergebnisse besagen, daß der Gattersimulator für alle denkbaren Gatterebenen-Systeme zwischen 40% und 50% der Simulationszeit einer entsprechenden Instanzennetzsimulation verbraucht.

Die Sätze von Parameterwerten für S2 und S4, die den minimalen Wert von R für G1 bzw. G2 liefern, sind praktisch identisch, ausgenommen unwichtige Parameter. Das gleiche gilt für die Sätze der Systeme S3 und S5, die den maximalen Wert von R für G1 bzw. G2 liefern. Das bedeutet, daß die ausgewählten Zeitmodelle geringen Einfluß auf den Effizienzgewinn haben. Dieser Gewinn ist hauptsächlich eine Funktion topologischer und dynamischer zeitmodellunabhängiger Eigenschaften des zu simulierenden Systems. Dies kann auch so verstanden werden, daß die zeitmodellabhängigen Teile der Simulation eine kleine Rolle spielen. Die meiste Simulationszeit wird in Aktivitäten verbraucht, die allen denkbaren Zeitmodellen gemeinsam sind, insbesondere die Verwaltung der Ereignis- und der Rufliste.

Die Zeitmodelle G1 und G2 scheinen, wenn wir "Zero-Delay"-Simulation nicht berücksichtigen, genug weit auseinander zu sein, um diesen Eindruck zu bestätigen. Für "Zero-Delay"-Simulation verschwindet praktisch der Effizienzunterschied zwischen dem Gatter- und dem Instanzennetzsimulator. Dies ist genau der Fall der Verknüpfungslogik in der Simulation der Register-Transferebene, wie wir im nächsten Kapitel sehen werden.

Wenn wir die Zeitverbrauchsausdrücke für beispielweise das Zeitmodell G1 vereinfachen, indem wir die Parameter, die geringen Einfluß auf den gesamten Zeitverbrauch haben, nicht berücksichtigen, bekommen wir aus (4.13) und (4.15) folgende Ausdrücke:

$$ZVS(G1) = E*(131 + 16*U + W*(49 + Z1 + 5*Y) + 115*c) \quad (4.29)$$

$$ZVI(G1) = E*(305 + 16*U + W*(104 + Z1 + 20,5*Y + 9*f) + 17*Y + c*(287 + 17*Y)) \quad (4.30)$$

Diese Ausdrücke ermöglichen uns zwei wichtige Folgerungen. Erstens ist R praktisch unabhängig von E, was schon durch die Untersuchung U1 festgestellt wurde. Das bedeutet, daß es unerheblich ist, ob wir Systeme mit großer oder kleiner Ereignisaktivität haben. Der Vorteil des Gattersimulators wird davon nicht beeinflußt.

Zweitens erzielen wir die kleine Gewinnspanne durch Variation der Parameter

U, W, und Y, die reine topologische Eigenschaften widerspiegeln,

Z1, dessen Wert Folge der im System vorhandenen logischen Funktionen ist, und

c und f, die mit dem Verhältnis zwischen den Verzögerungen und den Zeitunterschieden zwischen den Signalübergängen zusammenhängen.

Keiner von diesen Parametern hat allerdings einen entscheidenden Einfluß auf den Gewinn, und die Parameterwerte für die Sätze, die die verschiedenen Grenzfälle liefern, zeigen keinen besonderen Zusammenhang, der auf eine besondere Klasse von Gatterebenen-Systeme hinweisen könnte.

Obwohl der Gattersimulator beispielweise für Systeme mit großen Werten von c und f seinen größeren Vorteil gegenüber dem Instanzennetzsimulator zeigt, ist der Einfluß dieser Parameter fast bedeutungslos. Das können wir anhand eines extremen Beispiels zeigen. Das System S1, wo c=f=0 gilt, weist R = 42,3% auf. Falls wir nur c und f so ändern, daß sie ihre größten Werte annehmen, bekommen wir R = 40,52 %, d.h. der Gewinn wurde nur um 4,2 % seines Anfangswertes erhöht.

Zusammenfassend können wir sagen: Keine spezielle Klasse von Gatterebenen-Systemen scheint durch die Instanzennetzmodellierung weder begünstigt noch benachteiligt zu werden.

Obwohl die Gatterinstanzen etwas kompliziert aussehen (siehe Abb. 4.5), insbesondere durch die Behandlung der Auswirkung von Erkennungskonflikten, haben wir gezeigt, daß diese Behandlung fast reine Formsache ist, weil die auf sie bezogenen Parameter h, i und j keinen bedeutsamen Einfluß auf den Zeitverbrauch haben. Solche Auswirkungen sprechen allerdings eher für als gegen die Effizienz der Instanzennetzsimulation. Wir haben durch (4.17) und (4.24) bewiesen, daß die für die Effizienz negative Seite dieser Auswirkungen, die durch i und j dargestellt wird, wenn überhaupt vorhanden, dann doch immer kleiner sein wird als die positive Seite, die von h widergespiegelt wird.

Es ist auch zu beachten, daß wir für den Gattersimulator eine sehr günstige Annahme gemacht haben, nämlich, daß die Zeitverteilung der Verzögerungen auf eine kleine Spanne beschränkt ist. Ist dies nicht der Fall, muß die Ereignisliste im Gattersimulator ein Überfluß-Intervall haben /SZY75/, das die Ereignisse auffängt, die außerhalb des normalen Zeitintervalls ZSP der TIMEWHEEL fallen. Die Verwaltung dieses Überfluß-Intervalls (lineare Eintragung von Ereignisnotizen und Holen von Notizen, die aufgrund einer Zeitfortschaltung nun im Zeitintervall ZSP liegen) würde die Effizienz des Simulators sicherlich beeinträchtigen.

5. Simulation der Register-Transferebene

Im Gegensatz zur Gatterebenen-Simulation finden wir in der Literatur nicht viel über Algorithmen für die Simulation von auf der Register-Transferebene beschriebenen Systemen (der Kürze halber als Register-Transfer-Systeme bezeichnet), und die wenige Beschreibungen geben keine ausführliche Einsicht in die verwendeten Algorithmen (z.B. /HEM75/). Diese Feststellung können wir dadurch erklären, daß die Register-Transferebene eine Abstraktion der Gatterebene ist, in der wir digitale Systeme als die Vereinigung zweier kommunizierenden Teile modellieren, nämlich Operations- und Steuerwerk /WEN74/. Diese Trennung ist nicht unbedingt explizit in einer Beschreibung erkennbar, aber sie erklärt, warum so viele Register-Transfer-Sprachen (d.h. Sprachen, die digitale Systeme auf der Register-Transferebene beschreiben) sich entwickelt haben (siehe z.B. Einführungen in verschiedene Sprachen in /CD74/). Jede Sprache stellt verschiedene Kontrollstrukturen zur Verfügung /LEI80/, mit denen wir letzten Endes die Kausalstruktur, d.h. das für die Steuerwerkstruktur bestimmende Ablaufdiagramm, und die Kommunikation zwischen Operations- und Steuerwerk beschreiben können. Die Forschung richtet sich deswegen hauptsächlich auf die Definition von Sprachen, d.h., inwieweit eine Beschreibung von dieser Trennung geprägt ist und welche die Kontrollstrukturen sind, und dies spiegelt sich in der Literatur wider. Die Grundlinien für den einer Register-Transfer-Sprache zugeordneten Simulationsalgorithmus sind dann normalerweise direkt aus den Kontrollstrukturen ableitbar. Weitere Einzelheiten über diese Algorithmen bleiben jedoch unerwähnt.

Im weiteren bezeichnen wir nur den spezifischen Simulator für die Klasse der Register-Transfer-Systeme als Register-Transfer-Simulator, obwohl auch der aus der Instanzennetz-modellierung von Register-Transfer-Systemen resultierende Simulator diese Bezeichnung verdient.

5.1 Beschreibung der Register-Transferebene

Wir wollen hier eine möglichst allgemeine Untersuchung durchführen, die insbesondere nicht sprachbezogen ist. Je mehr Kontrollstrukturen wir voraussetzen würden, desto ferner wären wir von diesem Ziel. Allgemeine Begriffe für die Register-Transferebene sind hardwarenah, und so müssen wir möglichst wenige Abstraktionen machen, d.h., möglichst wenige Kontrollstrukturen in unseren Beschreibungen von Register-Transfer-Systemen haben.

Wir beschränken uns deswegen auf zwei Abstraktionen in Bezug auf die Gatterebene. Erstens führen wir als unteilbare Primärelemente der Register-Transferebene Einheiten ein, die aus mehreren Gattern zusammengesetzt und in zwei Klassen eingeteilt sind: Register, die einen internen Zustand aufweisen (Schaltwerke), und Verknüpfungsglieder, die

keinen internen Zustand haben (Schaltnetze). Der Ausgang eines Primärelementes ist b Bits breit, wobei eine Breite b=1 selbstverständlich erlaubt ist.

Zweitens deklarieren wir ein Signal als Takt, wobei Mehrphasentakt möglich ist. Alle Register werden durch diesen zentralen Takt geladen, so daß Taktung durch andere im System abgeleitete Signale nicht modellierbar ist. Taktausblendung für einzelne Register ist möglich, indem jedes Register einen ENABLE-Eingang haben kann. Dieser Eingang darf mit dem Ausgang eines beliebigen Verknüpfungsgliedes im System verbunden sein.

Das Repertoire von Verknüpfungsgliedertypen ist nicht definiert. Damit wir uns mit ihm nicht beschäftigen müssen, betrachten wir den Zeitverbrauch der Funktionsauswertung der Verknüpfungsglieder wie in der Gatterebenen-Simulation als einen Parameter. Allen Verknüpfungsgliedern ist eine Nullverzögerung zugeordnet. Diese Entscheidung läßt sich mit dem Hauptziel der Simulation der Register-Transferebene vereinbaren, nämlich die Überprüfung der logischen Funktion und der Kausalstruktur eines digitalen Systems. Die Taktphase ist dann der kleinste Zeitschritt der Simulation. Es ist deshalb zu verstehen, warum asynchrone Registertaktung nicht erlaubt werden kann: Sie würde eine andere Definition für die Zeitfortschaltung verlangen. Wir nehmen an, daß Zeitbedingungen, die mit den Gatterverzögerungen zusammenhängen, in einer anderen Entwurfsphase überprüft werden. Auch deswegen benutzen wir hier nur zweiwertige Logik.

Um die Simulationsalgorithmen durchsichtlicher zu machen, betrachten wir alle Register als vorderflankengesteuert /WEN74/. Rückflanken- und zweiflankengesteuerte Register lassen sich sowohl in dem Register-Transfer-Simulator als auch in der Instanzennetzmodellierung leicht einführen. Latchregister bringen zwar eine gewisse Komplexität für die Simulation, aber wir schliessen ihre Betrachtung aus, weil sie keine wesentlichen Erkenntnisse für die Untersuchung darstellen.

Die Aufteilung der Systemelemente in zwei Klassen (Schaltnetze und Register) bedeutet, daß ein Register-Transfer-System einem synchronen Automaten entspricht (siehe Abb. 5.1). Wir können deswegen in einem solchen System zwei getrennte Blöcke erkennen, nämlich den **Register-** und den **Verknüpfungsblock**. Dieser Automat muß zwei Rückkopplungsbedingungen erfüllen, um richtig definiert zu sein /WEN74/:

1) Die Entscheidungsintervalle für die Register dürfen erst anfangen, wenn der Schaltnetzausgang stabil ist; und
2) Ein Schaltnetzausgang darf sich erst ändern, wenn alle Entscheidungsintervalle beendet sind.

Wir haben schon gesagt, daß das Ziel der Register-Transfer-Simulation keine Überprüfung von Zeitbedingungen ist. Trotzdem müssen die zwei obengenannten Bedingungen durch die Simulation eingehalten werden, weil sonst ein logischer

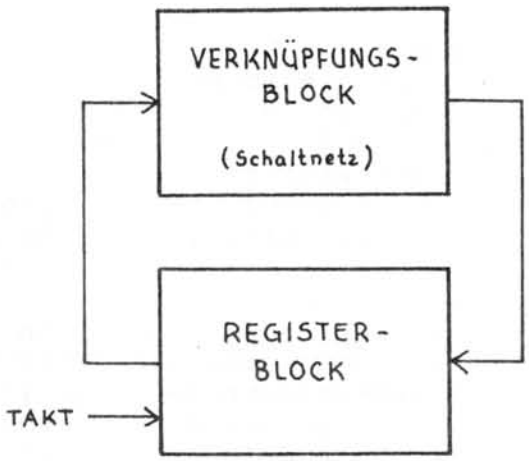


Abb. 5.1 - Register-Transferebenen-System als einen synchronen Automaten

Fehler auftreten kann. Die Erfüllung dieser Bedingungen wird dann automatisch durch die Simulatoren gewährleistet. Allen hier behandelten Modellierungen der Register-Transferebene sind folgende Vereinfachungen über das Verhalten digitaler Systeme gemeinsam:

a) Da eine Taktphase der kleinste Zeitschritt in der Simulation ist, das Schaltnetz Nullverzögerung hat (es reagiert sofort auf neue Registerinhalte) und die Register erst in der nächsten Taktphase wieder geladen werden, ist es sicher, daß die erste Rückkopplungsbedingung immer erfüllt ist; und

b) Die Simulation einer Taktphase (d.h. eine Systemüberführung) führen wir immer in zwei getrennte nacheinander folgenden Teilschritte aus, nämlich das Laden der Register und den Durchlauf der Änderungen durch das Schaltnetz. Aus der Sicht der Verknüpfungsglieder gewährleisten wir damit, daß alle Register parallel geladen werden, so daß die zweite Rückkopplungsbedingung teilweise erfüllt ist.

Dennoch besteht eine nicht erfüllte Bedingung. Ein Registerausgang darf mit einem Registereingang direkt verbunden sein, ohne daß ein Verknüpfungsglied dazwischen geschaltet ist. In diesem Fall besteht das Teilschaltnetz zwischen den Registern aus einem Draht, für den die zweite Rückkopplungsbedingung durch die schon eingeführten Modellierungseigenschaften nicht garantiert ist. Es darf nicht vorkommen, daß durch die Simulation ein neuer Registerinhalt den Inhalt eines anderen Registers noch in der gleichen Taktphase beeinflußt (Latchregister sind ausgeschlossen !). Auf die Erfüllung dieser Bedingung müssen wir deswegen in den verschiedenen Modellierungen achten.

Wir entwickeln zwei Algorithmen für den Register-Transfer-Simulator. Das erste Modell ist für alle Register-Transfer-Systeme anwendbar, die mit der bisherigen Beschreibung zu vereinbaren sind, und ergibt den im weiteren genannten Simulator RT1. Das zweite Modell, aus dem der Simulator RT2 resultiert, setzt gewisse Eigenschaften der Systemtopologie voraus, so daß eine bestimmte Klasse von Register-Transfer-Systemen ausgeschlossen ist, aber dafür haben wir eine größere Effizienz gegenüber dem Simulator RT1.

Diese Eigenschaften haben keinen Einfluß auf die Instanzennetzmodellierung von Register-Transfer-Systemen. Wir stellen aber auch zwei Instanzennetzmodellierungen von Register-Transfer-Systemen vor, einmal mit explizitem Taktgeber und ein andermal ohne Taktgeber. Die Einführung dieser beiden Modelle wird dadurch gerechtfertigt, daß es schwierig vorauszusehen ist, welche von beiden die größere Effizienz zeigt. Basierend auf den Ergebnissen eines Vergleichs zwischen ihnen entscheiden wir uns für eine der Modellierungen und verwenden diese als Muster für den Vergleich mit den Simulatoren RT1 und RT2.

5.2 Register-Transfer-Simulatoren

5.2.1 Gemeinsame Eigenschaften

Beide Register-Transfer-Simulatoren besitzen getrennte Ruflisten für die Register und die Verknüpfungsglieder. In die Rufliste der Verknüpfungsglieder tragen wir die Namen der Glieder ein, die durch Änderungen an Register- (im ersten Teilschritt einer Systemüberführung) und Verknüpfungsgliederausgängen (im zweiten Teilschritt) betroffen werden. Infolge der dem Schaltnetz zugeordneten Nullverzögerung wertet der Simulator die Funktionen der eingetragenen Elemente noch in derselben Taktphase aus. Genauso bilden wir Register-Ruflisten für jede Taktphase, wobei die für das Laden jedes Registers maßgebliche Taktphase bekannt sein muß. Die Register, deren Namen in der Rufliste eingetragen sind, laden wir jedoch selbstverständlich erst in einer späteren Taktphase, falls die entsprechenden ENABLE-Eingänge den Wert TRUE aufweisen. Damit gewährleisten wir die Erfüllung der ersten Rückkopplungsbedingung.

Die Modellzeitfortschaltung erfolgt implizit durch die Verarbeitung beider Ruflistentypen. Das Ende einer Systemüberführung wird erreicht, wenn die Rufliste für die Verknüpfungsglieder leer ist. Um wieviele Zeiteinheiten die Modellzeit erhöht wird, ist auf dieser Abstraktionsebene unerheblich. Die Modellzeit schrumpft zu der Zählung der schon durchgeführten Taktperioden.

Der Simulator wertet erst die Funktionen der Verknüpfungsglieder aus, wenn die der laufenden Taktphase entsprechende Register-Rufliste leer ist. Somit teilen wir die Systemüberführung in die schon erwähnten Teilschritte auf und gewährleisten die Teilerfüllung der zweiten Rückkopplungs-

bedingung.

Wie im Abschnitt 3.3 diskutiert, nehmen wir für die Hardware-Simulatoren an, daß sie kein Repertoire von Funktionen dem Benutzer zur Verfügung stellen. Allgemeine Prozeduren für die Verknüpfungsgliederfunktionen gibt es deswegen nicht. Mit den Konstrukten der Register-Transfer-Sprache darf der Benutzer trotzdem beliebige Funktionen für die Verknüpfungsglieder definieren. Aus Effizienzgründen verlagern wir die entsprechenden Auswertungen der logischen Funktionen in die Hauptschleife des Algorithmus und bilden damit einen systemabhängigen Teil des Simulators, wie wir es für den Instanzennetzsimulator gemacht haben. Prozeduraufrufe sind jedoch in den Register-Transfer-Simulatoren nicht notwendig.

Die den Register entsprechenden Aktivitäten sind auch von vornherein bekannt (Testen des ENABLE-Eingangs und ggf. Laden), und deswegen sind Register-Prozeduren ebenfalls unnötig.

Im Gegensatz zur Gatterebene können die Verknüpfungsglieder auf der Register-Transferebene mehrere Ausgänge haben. Typische Beispiele dafür sind der Decoder, der Volladdierer und der Entbündeler. Wenn wir die Ausgangswerte der Verknüpfungsglieder in einem Array SW speichern, ist eine direkte Verwendung des Gliedernamens als Index für SW unmöglich. Eine dreistufige Hierarchie, wie für den Instanzennetzsimulator, ist hier erforderlich. Um den Vergleich deutlicher zu machen, behalten wir die Namen PARAMLISTE und PARAMVERWEIS für die anderen notwendigen Hierarchiestufen. Ausgenommen die Tatsache, daß ein einziger Datentyp, nämlich Vektoren von booleschen Variablen, auf der Register-Transferebene vorhanden ist, ist die Datenstruktur für die Anschlußvariablen der Verknüpfungsglieder ebenfalls der Abb. 2.5 zu entnehmen.

Da die Register eine Sonderklasse bilden, die in allen Register-Transfer-Systemen vertreten sein muß und deren Elemente immer dieselbe einfache und bekannte Anschlußstruktur haben, können wir für sie eine zweistufige Datenhierarchie verwenden. Wir ersetzen die PARAMLISTE durch drei besonderen Arrays EINGANG, AUSGANG und ENABLE und lassen damit PARAMVERWEIS entfallen. ENABLE[i] hat den Wert "leer", falls für das entsprechende Register i der Takt nicht ausblendbar ist.

Damit die von einer Signaländerung betroffenen Systemelemente in die jeweiligen Ruflisten eingetragen werden können, ist eine Datenstruktur mit FANOUTPOINTER und -LISTE notwendig, wie wir sie schon für die Simulation der Gatterebene benötigt haben.

5.2.2 Modell RT1

Das erste Modell für Register-Transfer-Simulation verwendet eine universelle Lösung für die noch benötigte Teilerfüllung der zweiten Rückkopplungsbedingung. Diese Lösung setzt keine besonderen Eigenschaften bezüglich der Topologie des zu simulierenden Systems voraus.

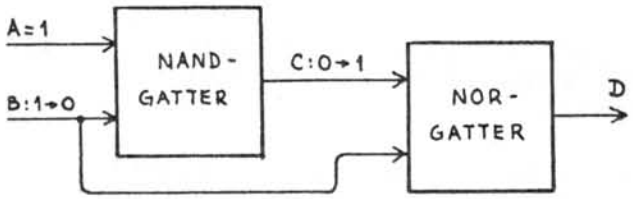
Innerhalb des Registerblocks können wir deswegen eine beliebige Topologie haben. Damit neue Registerinhalte das Laden anderer Register in derselben Taktphase nicht beeinflussen, teilen wir diesen Ladenprozess in zwei nacheinander folgende Teilschritte auf, die dem Entscheidungs- und dem Übergangsintervall entsprechen.

Dafür sind zwei separate Ruflisten notwendig: Eine, die wir während der vergangenen Taktschritte gebildet haben und die dazu dient, die Eingangswerte der Register einer Zwischenvariable zuzuweisen, und die zweite für die Zuweisung des Wertes dieser Zwischenvariablen zu den Registerausgängen. Zweckmäßigerweise benennen wir sie die "Master"- und die "Slave"-Rufliste. Eine Master-Rufliste ist für jede einzelne Taktphase nötig, während eine einzige Slave-Rufliste genügt, weil sie immer während einer einzigen Taktphase neu belegt und voll geleert wird.

Die Werte der Zwischenvariablen können wir in einem von SW getrennten Array MASTER speichern, denn nur die entsprechenden Register greifen auf sie zu.

Auch für die Verarbeitung des Verknüpfungsblocks machen wir keine Voraussetzung über die innere Blocktopologie und verwenden deswegen die universelle Lösung des Instanzennetzsimulators für die Bildung der Rufliste. Sie wird dynamisch während der Systemüberführung nach den Signalwertänderungen und mit Hilfe der FANOUTLISTE gebildet. Mehrmalige Eintragungen des Namens eines Elementes während derselben Systemüberführung aufgrund von gleichzeitigen Eingangsänderungen werden ebenfalls vermieden.

Zwei Folgerungen ergeben sich aus dieser Vorgehensweise. Erstens können Pulse mit Nulldauer (= Hazardpulse) infolge der Erkennungskonflikte und der Nullverzögerungen auftreten, wie Abb. 5.2 veranschaulicht. Zweitens können Oszillationen auftreten, falls Schleifen innerhalb des Blocks vorhanden sind. Wir nehmen aber an, daß diese Oszillationen entweder nie auftreten, oder der Simulator so ausgestattet ist, daß sie nach endlicher Zeit identifiziert werden. Deshalb dürfen wir die Rufliste der Verknüpfungsglieder auf eine vernünftige endliche Größe begrenzen, ohne sie zyklisch organisieren zu müssen. Welche Methode die Erkennung der Oszillationen erlaubt, diskutieren wir nicht, weil sie das Wesen des Algorithmenentwurfs nicht beeinflußt und nur im Rahmen einer bestimmten Benutzerfreundlichkeit relevant ist.



(a) Beispiel eines Teilsystems

Auswertungs-reihenfolge	Wertereihenfolge für D
C , D	1
D , C , D	1 → 0 → 1

(b) mögliche Wertereihenfolgen fürs Signal D

Abb. 5.2 - Beispiel eines Null-(Hazard-)Pulses bei der Simulation des Verknüpfungsblocks eines Register-Transfer-Systems nach dem Modell RT1

Die Nullpulse haben dagegen einen bedeutsamen Einfluß auf unseren Vergleich. Die Erkennungskonflikte können hier unterschiedliche Zwischenzustände während der Verarbeitung des Verknüpfungsblocks veranlassen. Da die Modellierung aber implizit annimmt, daß die Taktperiode genügend groß ist, und da die Oszillationen ausgeschlossen sind, erreicht das System am Ende einer Systemüberführung immer denselben Zustand, der für die Belegung der Register in den nächsten Taktphasen entscheidend ist, unabhängig davon, welche oder wieviele Hazardpulse stattgefunden haben. Für die Simulationseffizienz bleiben jedoch zwei Auswirkungen: Wir können die Funktionen bestimmter Verknüpfungsglieder mehrmals im Laufe der Verarbeitung einer Taktphase auswerten und einige Registernamen unnötigerweise in die Rufliste eintragen.

Für alle erwähnten Ruflisten verwenden wir eine FIFO-Organisation, die am einfachsten zu realisieren ist. Für jede Liste sind dann die Pointer REAR (wo das nächste Element einzutragen ist) und FRONT (woher das nächste zu verarbeitende Element zu holen ist) vorhanden.

Jedem Element ist eine Variable ID zugeordnet. Ein Wert von ID gleich Null kennzeichnet ein Verknüpfungsglied. Ein Wert ungleich Null kennzeichnet ein Register und gleichzeitig die Taktphase, in der das Register geladen wird. Diese Variable ist notwendig, damit der Name eines Elementes aufgrund einer Eingangsänderung in die richtige Rufliste eingetragen wird.

Abb. 5.3 zeigt den Simulationsalgorithmus für das Modell RT1.

5.2.3 Modell RT2

Das Modell RT2 setzt voraus, daß sowohl innerhalb des Register- als auch des Verknüpfungsblocks keine Schleifen vorhanden sind. Diese Beschränkung in der Systemtopologie erlaubt eine Vereinfachung der Implementierung der Ruflisten und bringt dadurch eine Effizienzerhöhung.

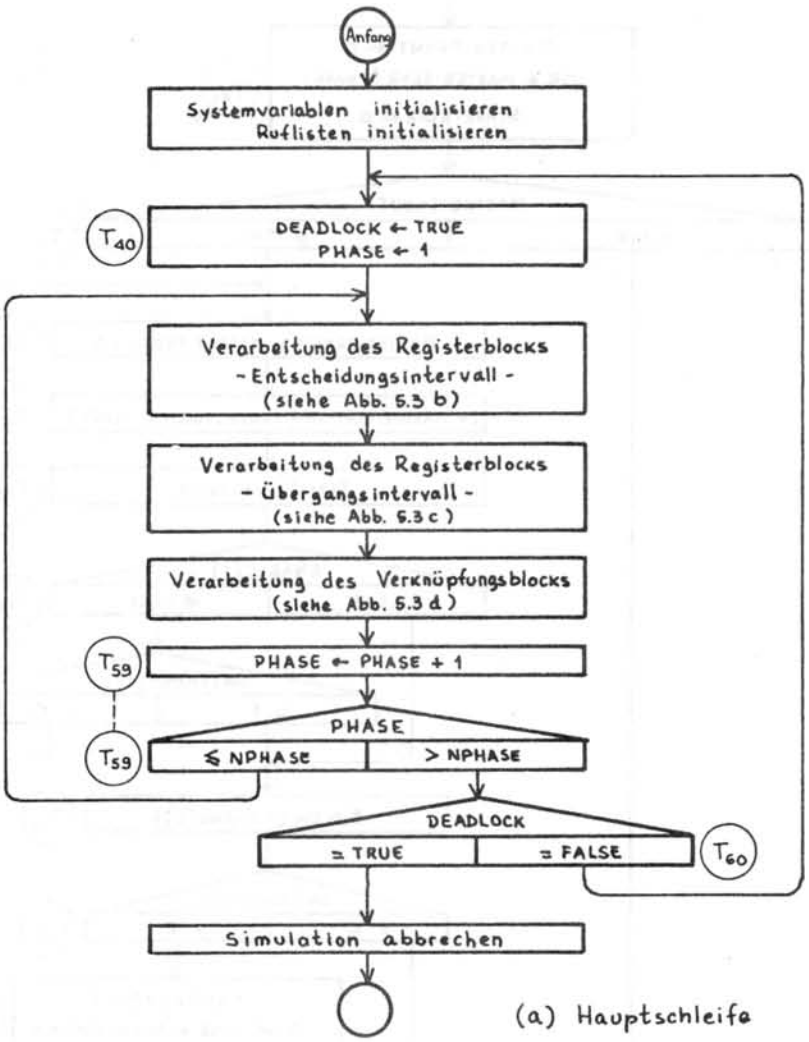
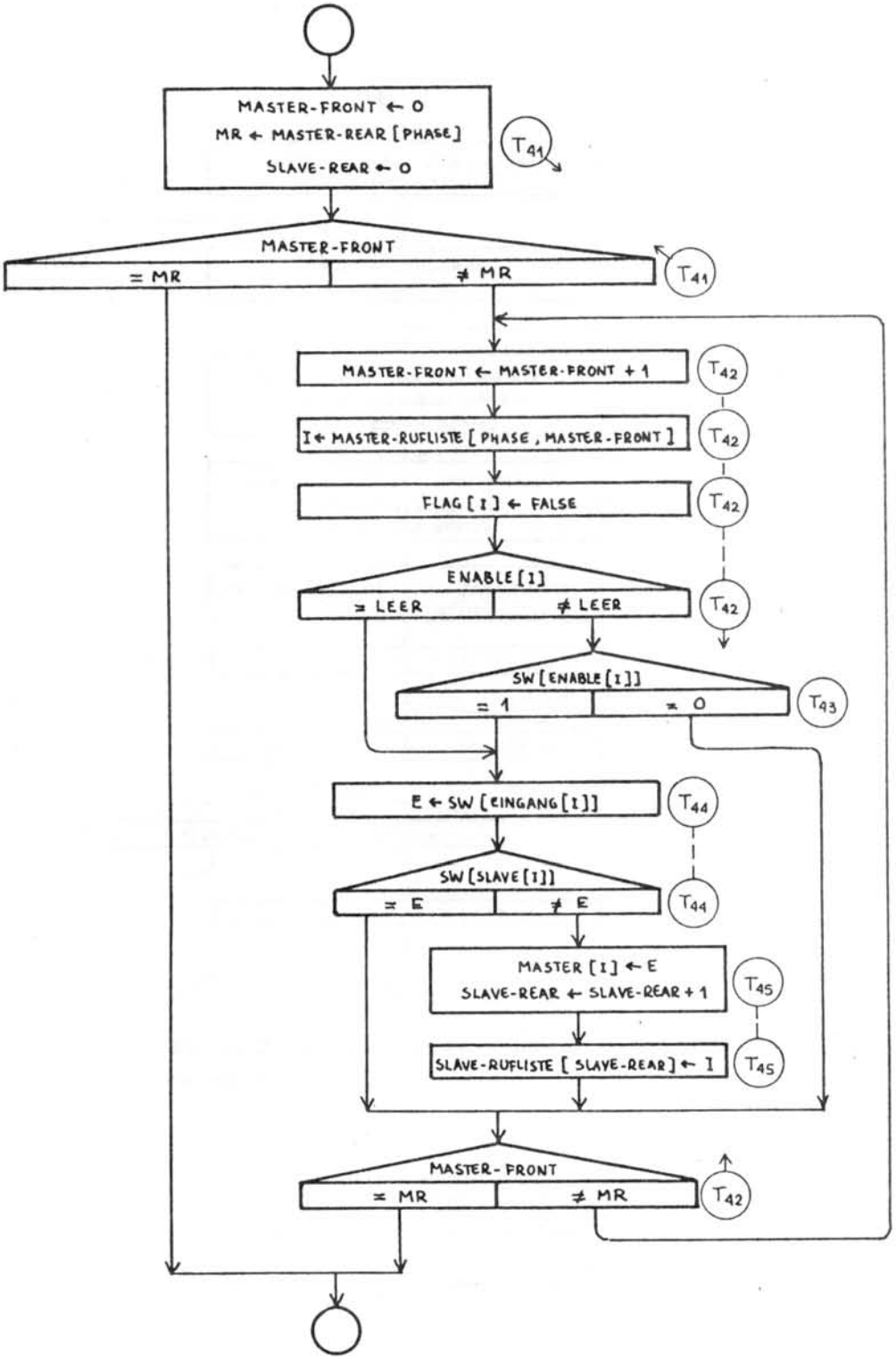


Abb. 5.3 - Algorithmus für den Register-Transfer-Simulator, Modell RT1
(Fortsetzung auf den nächsten Seiten)

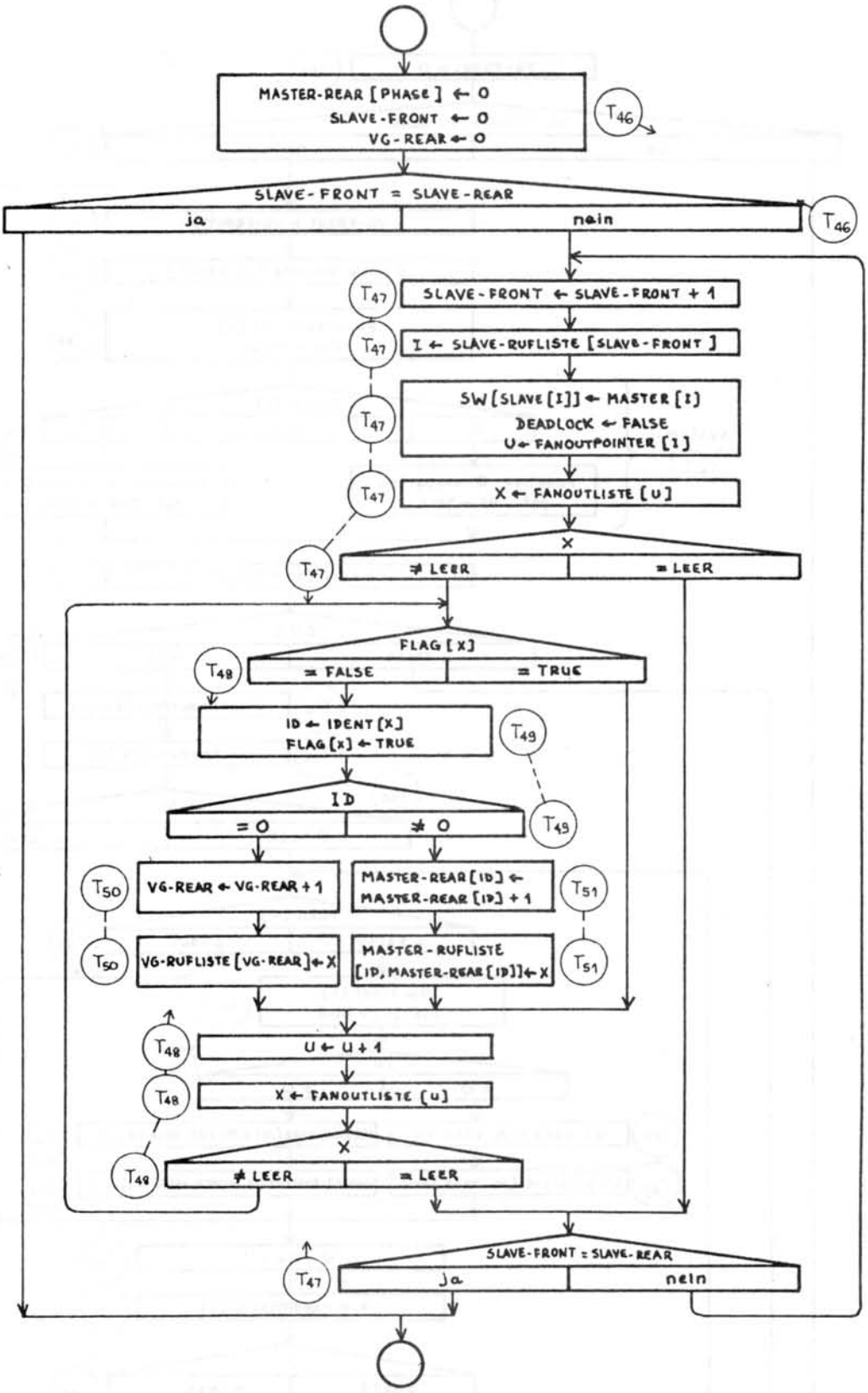
Die Abwesenheit von Schleifen und die Tatsache, daß Register-Transfer-Systeme gerichtet sind, erlauben uns, von einer **Speisefolge** zu sprechen. Ein Element wird von einem anderen gespeist, wenn einer seiner Eingänge mit einem Ausgang des anderen Elementes verbunden ist.

Wir bilden die Rufliste jetzt nicht mehr dynamisch während einer Systemüberführung. Es gibt vielmehr statische Listen, die die Namen aller Elemente enthalten und nach der direkten Speisefolge (Verknüpfungsglieder) bzw. nach der umgekehrten Speisefolge (Register) gebildet sind, wie Abb. 5.4 zeigt. Wie für das Modell RT1 haben wir eine Register-Rufliste für jede Taktphase und eine einzige Rufliste für die Verknüpfungsglieder. Damit das Prinzip der selektiven Simulation der Elemente, deren Eingänge geändert worden sind, eingehalten wird, besteht in der Tat jede Stelle der



(b) Verarbeitung des Registerblocks - Entscheidungsintervall -

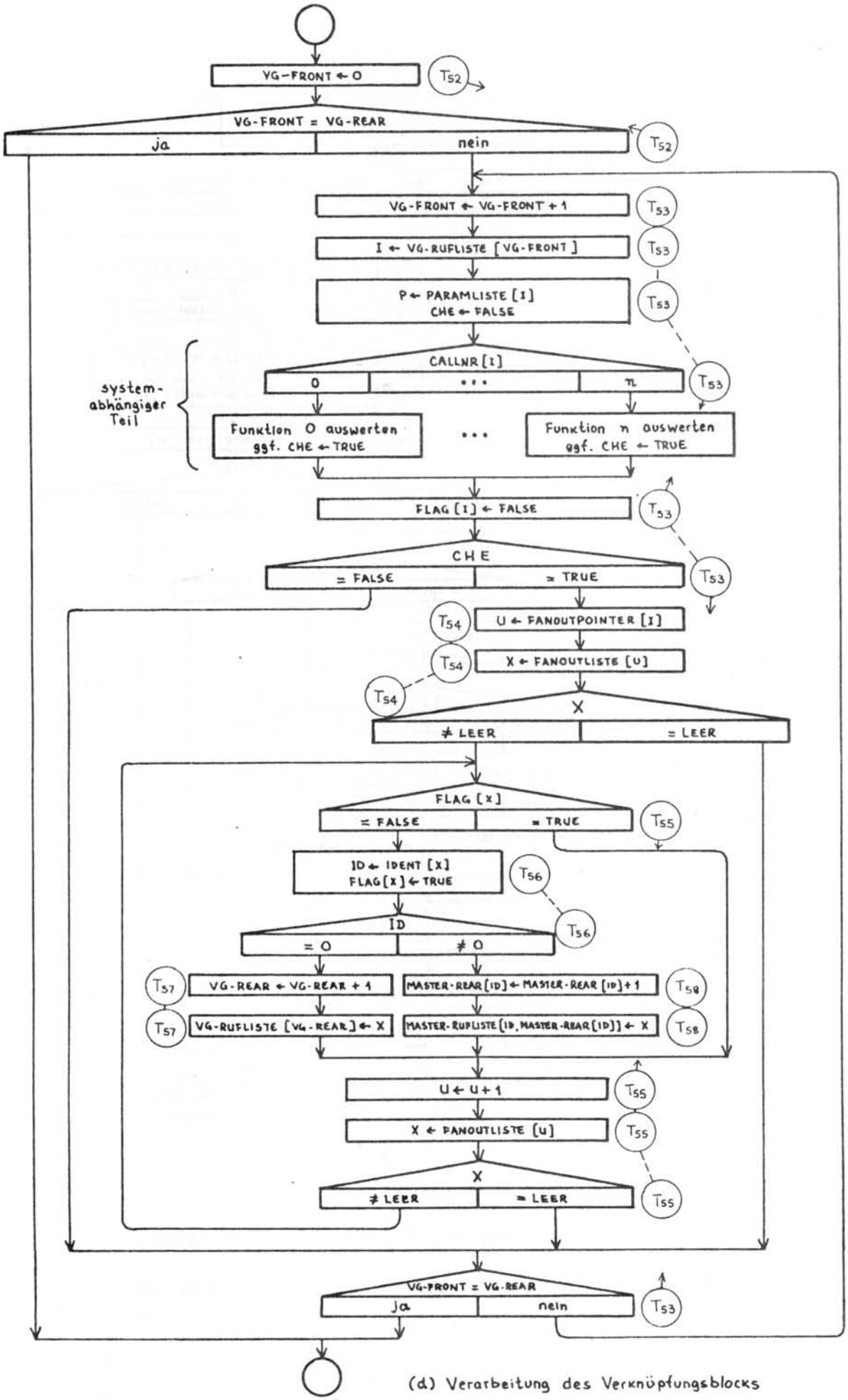
Abb. 5.3 - Algorithmus für den Register-Transfer- Simulator , Modell RT1



(c) Verarbeitung des Registerblocks - Übergangsintervall -

Abb. 5.3 - Algorithmus für den Register-Transfer-Simulator, Modell RT1

(Fortsetzung auf der nächsten Seite)



(d) Verarbeitung des Verknüpfungsblocks

Abb. 5.3 - Algorithmus für den Register-Transfer-Simulator, Modell RT1

Ruflisten aus einer booleschen Variable CALLBIT, die angibt, ob der Simulator die Funktion des Elementes demnächst auswerten muß. Jedem Register bzw. Verknüpfungsglied ist ein eindeutiger Index für die Liste (= sein Name) zugeordnet.

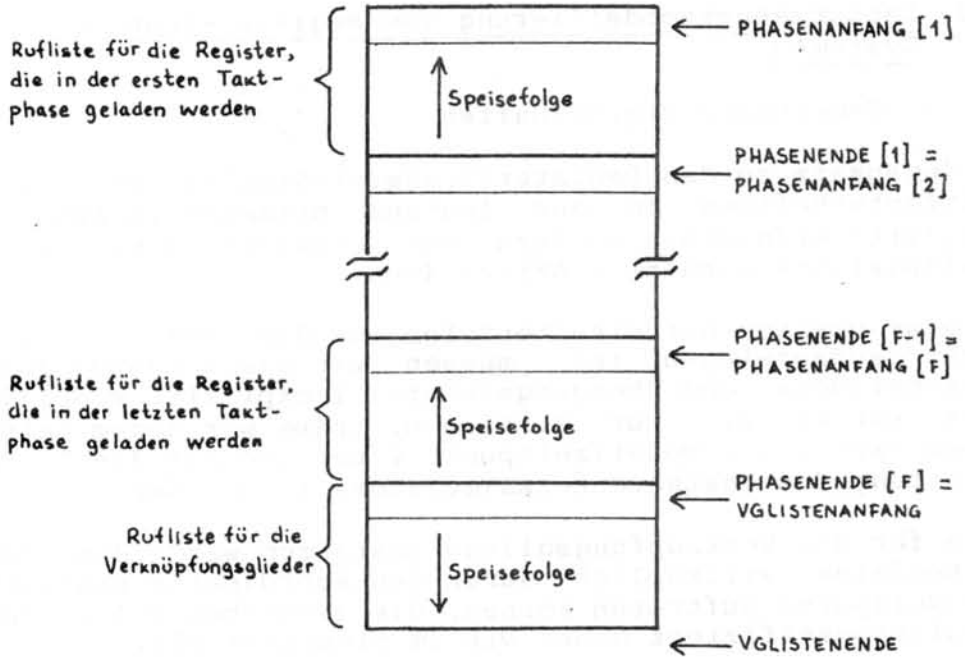


Abb. 5.4 - Statische Ruflisten für den Register-Transfer-Simulator, Modell RT2

Die Eintragung des Namens eines Elementes in die Rufliste schrumpft zum Setzen des jeweiligen CALLBITs. Für die Verarbeitung eines Blocks ist dann notwendig, die ganze dazugehörige Rufliste durchzulaufen und die einzelnen CALLBITs abzuprüfen.

Im Gegensatz zum Modell RT1 haben wir nun eine einzige Register-Rufliste je Taktphase, so daß die Entscheidungs- und Übergangsintervalle nicht mehr eindeutig getrennt sind. Aufgrund der umgekehrten Speisefolge in der Register-Rufliste laden wir allerdings ein gespeistes Register immer vor den speisenden Registern. Das bedeutet, daß das Entscheidungsintervall des gespeisten Registers immer vor den Übergangsintervallen der speisenden Register kommt. Damit ist die zweite Rückkopplungsbedingung erfüllt.

Für das Schaltnetz garantieren wir mit der Funktionsauswertung nach der direkten Speisefolge, daß die Funktion eines Elementes, dessen Eingänge sich geändert haben, nur einmal in dieser Taktphase ausgewertet wird. In der Tat lösen wir mit dieser Organisation der Rufliste die Erkennungskonflikte immer zugunsten der Elemente, die in der Speisefolge vorher kommen. Damit wirken sich diese Konflikte nie in Form von Nullpulsen aus.

Abb. 5.5 zeigt den Simulationsalgorithmus für das Modell RT2. Dieses Modell ist eine Anpassung der im Simulationssystem BORIS eingeführten "implizit getakteten Netze" /WEN79b/.

5.3 Instanzennetzmodellierung von Register-Transfer-Systemen

5.3.1 Gemeinsame Eigenschaften

Im Gegensatz zu den Register-Transfer-Simulatoren kann die Zeitfortschaltung in der Instanzennetzmodellierung nicht implizit erfolgen, sondern nur explizit durch in der Ereignisliste vermerkte Aktivitäten.

Da eine gewisse Aufrufreihenfolge für die Registerinstanzen nicht aufzuzwingen ist, müssen wir die Trennung zwischen Entscheidungs- und Übergangsintervall explizit modellieren. Dies können wir nur erreichen, indem wir jedem Intervall einen expliziten Modellzeitpunkt (bezogen auf die Zeitfortschaltung des Instanzennetzsimulators) zuordnen.

Auch für die Verknüpfungsgliederinstanzen wird die Aufrufreihenfolge willkürlich durch den Koordinator bestimmt, so daß Nullpulse auftreten können, die dieselben Folgen für die Simulationseffizienz haben wie im Simulator RT1.

Eine sehr wichtige Eigenschaft der Instanzennetzmodellierung von Register-Transfer-Systemen besteht darin, daß wegen der Nullverzögerung die Verknüpfungsgliederinstanzen nur aufgrund von Umgebungsänderungen aufgerufen werden. Im Gegensatz zur Modellierung der Gatterebene ist deswegen Sequenzermarkenverwaltung für diese Instanzen überhaupt nicht nötig. Dies bedeutet, daß genau dieselbe Prozeduraufschreibung der Instanzennetzmodellierung anwendbar ist für die Funktionsauswertung der Verknüpfungsglieder in den Register-Transfer-Simulatoren. Wir brauchen deshalb diese Aufschreibungen nicht zu berücksichtigen. Der ganze Zeitverbrauch der Durchführung einer Verknüpfungsgliederinstanzenprozedur wird durch einen Parameter erfaßt und ist mit dem Zeitverbrauch der entsprechenden Auswertung der Gliederfunktion in den Simulatoren RT1 und RT2 identisch. Dies gilt, weil Parameterübergabe nicht nötig ist (siehe Abschnitt 3.3) und der Rückkehr aus der Prozedur im Zeitverbrauch des Koordinators berücksichtigt wird.

5.3.2 Modellierung mit explizitem Taktgeber

Abb. 5.6 zeigt das Instanzennetzmodell eines Register-Transfer-Systems, für den Fall, daß wir das Taktsignal explizit als Ausgang einer Taktgeberinstanz modellieren. Wir nehmen an, daß es einen Taktgeber je Taktphase gibt, dessen Taktausgang nur mit den Registern verbunden ist, die in der jeweiligen Taktphase geladen werden. Der Koordinator bringt die Taktgeberinstanzen in der notwendigen Reihenfolge in

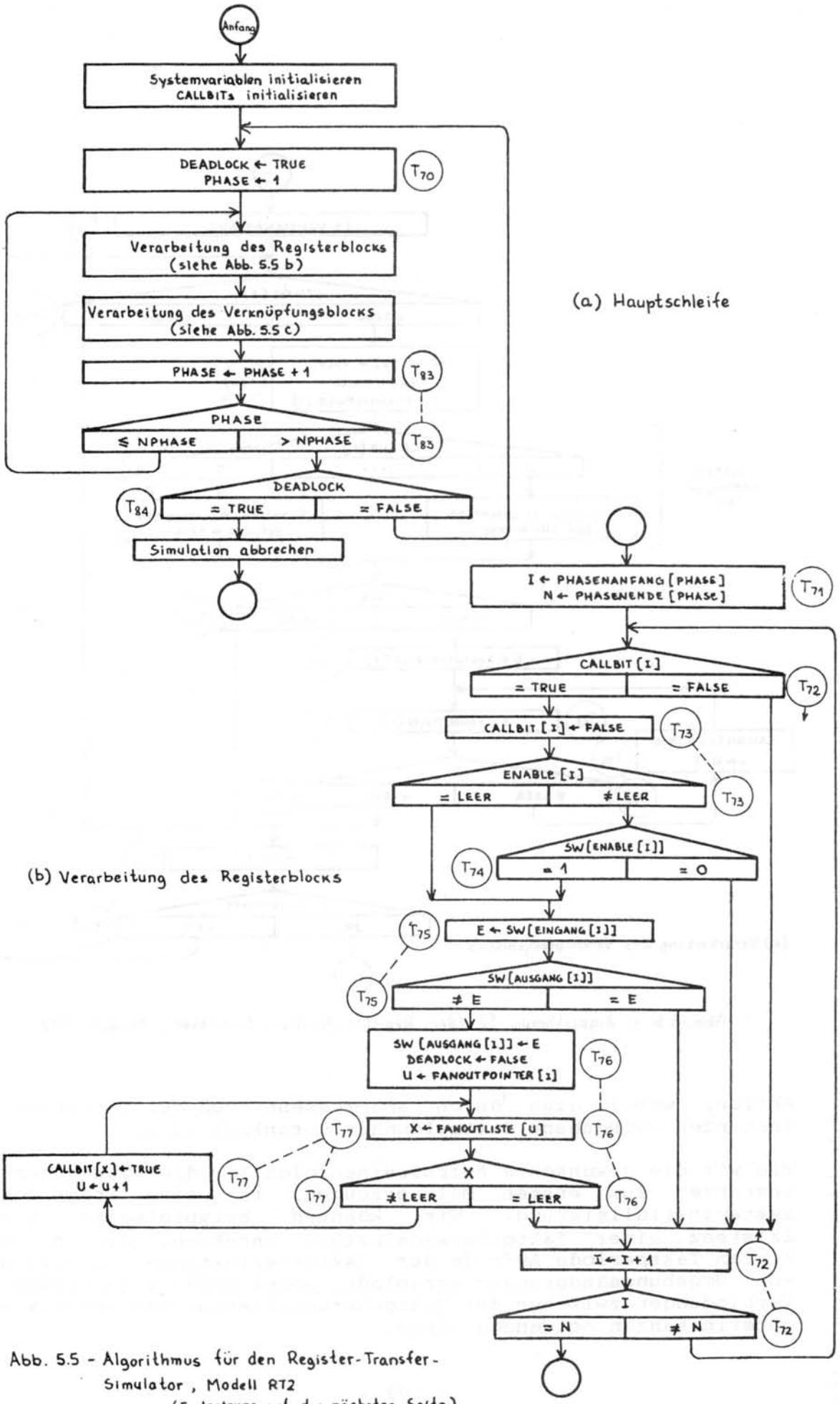
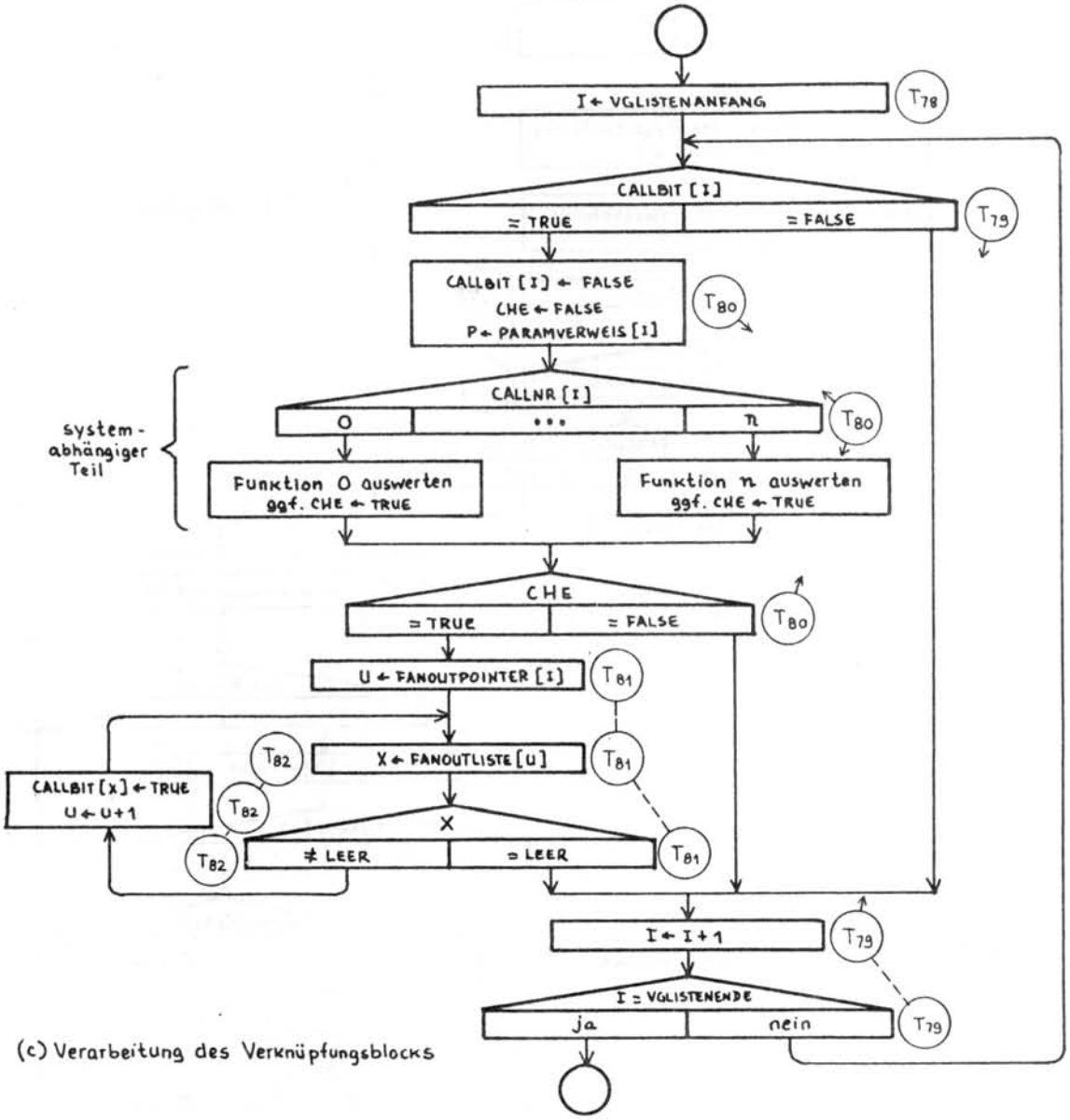


Abb. 5.5 - Algorithmus für den Register-Transfer-Simulator, Modell RT2

(Fortsetzung auf der nächsten Seite)



(c) Verarbeitung des Verknüpfungsblocks

Abb. 5.5 - Algorithmus für den Register-Transfer-Simulator, Modell RT2

Aktion, wobei dies durch angemessene von den Taktgeberinstanzen abgegebene Zeitmeldungen veranlaßt wird.

Wie wir die gewünschte Aufrufreihenfolge für die Taktgeberinstanzen zum ersten Mal erzeugen, ist eine Frage der Systeminitialisierung. Wir könnten beispielweise die Existenz einer Taktgeberweckinstanz annehmen, die in der ersten Taktperiode Aufrufe der Taktgeberinstanzen aufgrund von Umgebungsänderungen veranlaßt, wobei dafür verschiedene Verbindungen zwischen der Taktgeberweckinstanz und den Taktgeberinstanzen notwendig wären.

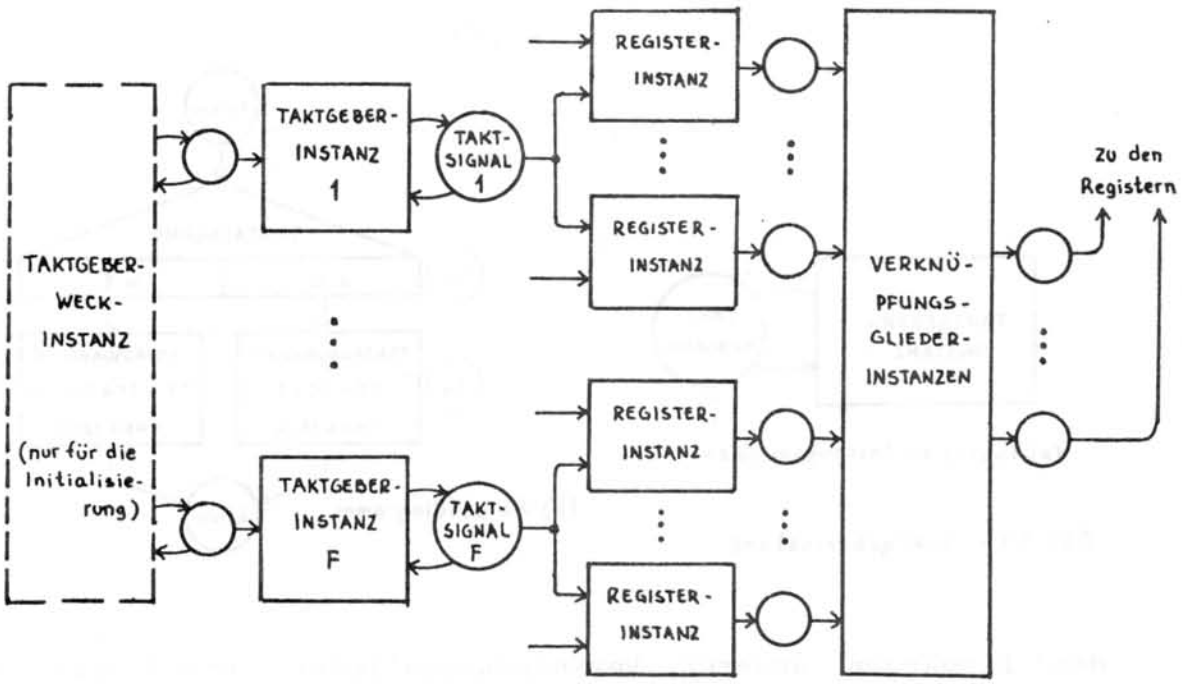
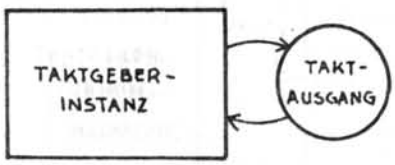


Abb. 5.6 - Instanzennetzmodellierung eines Register-Transfer-Systems mit explizitem Taktgeber

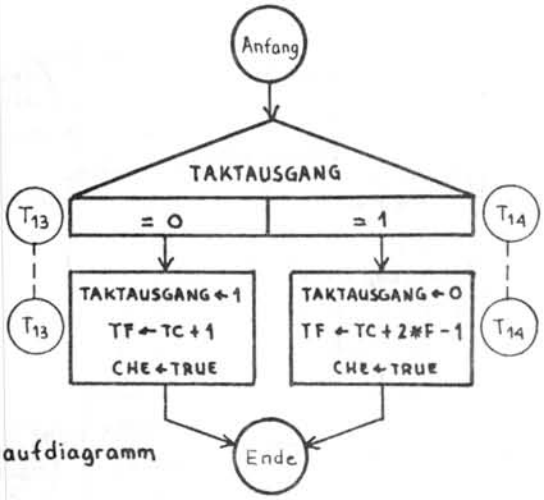
Um Entscheidungs- und Übergangsintervalle für jede Taktphase voneinander zu trennen, ruft der Koordinator jede Taktgeberinstanz in zwei nacheinander folgenden Modellzeitpunkten auf. Jedesmal ändert sich der Taktausgang, so daß alle mit ihm verbundenen Registerinstanzen aufgrund dieser Umgebungsänderung aktiviert werden. Diese Instanzen erkennen durch den Zustand der Taktvariable, ob die "Master"- (Takt = 1) oder die "Slave"-Funktion (Takt = 0) durchzuführen ist. Abb. 5.7 und 5.8 zeigen die Strukturen der Taktgeber- bzw. Registerinstanzen.

Die im Modell RT1 verwendete Zwischenvariable MASTER ist hier durch die Zustandsvariable ZDATA der Registerinstanzen implementiert. Eine zusätzliche Zustandsvariable ZTAKT ist noch erforderlich, und sie, zusammen mit dem Takteingang, ermöglicht die für die Registerinstanz notwendige Sequenzermarkenverwaltung. Die Instanz kann unter drei verschiedenen Umständen aufgerufen werden: Aufgrund einer Änderung des Takteingangs, wenn er von 0 bzw. 1 auf 1 bzw. 0 geht, oder aufgrund einer Änderung des Daten- oder des ENABLE-Eingangs.

Während des ersten Schrittes einer Taktphase, wenn TAKT den Wert 1 hat, werden die Zwischenvariablen ohne weitere Auswirkungen geladen. Während des zweiten Schrittes bekommen die Registerausgänge ihre neuen Werte, und diese Änderungen haben Auswirkung auf das Schaltnetz. Verknüpfungsgliederinstanzen können dann aufgrund einer Umgebungsänderung in Aktion gebracht werden, und ihre Ausgangsänderungen sind an



(a) Zugang zu Anschlußvariablen



(b) Ablaufdiagramm

Abb. 5.7 - Taktgeberinstanz

den Eingängen anderer Verknüpfungsglieder- und Registerinstanzen beobachtbar. Bei diesem möglichen zweiten Aufruf innerhalb des zweiten Schrittes einer Taktphase haben jedoch die Registerinstanzen keine Aktivität durchzuführen. Diese Situation unterscheidet sich vom ersten Aufruf während dieses Schrittes durch den Wert 1 der Variable ZTAKT.

Es ist zu beachten, daß diese Modellierung im Gegensatz zu den Register-Transfer-Simulatoren nicht gewährleistet, daß alle Übergangsintervalle der Register schon beendet sind, wenn der Koordinator die erste Verknüpfungsgliederinstanz aufgrund einer Umgebungsänderung aktiviert. Diese Tatsache hängt nicht mit den Rückkopplungsbedingungen zusammen und könnte nur Nullpulse an den Ausgängen der Verknüpfungsglieder auslösen, die mit zwei Registerausgängen verbunden sind. Dies kommt in unserer konkreten Implementierung des Instanzennetzsimulators jedoch nicht vor, weil die Rufliste zunächst durch die Umgebungsänderungsmeldung der Taktgeberinstanz alle Registernamen bekommt, und erst dann, durch Meldungen der Registerinstanzen, die Namen von Verknüpfungsgliederinstanzen.

5.3.3 Modellierung ohne expliziten Taktgeber

Die Trennung zwischen Entscheidungs- und Übergangsintervall machen wir hier, indem wir die Master- und die Slave-Funktion eines Registers explizit durch zwei getrennten Instanzen modellieren, die zu zwei nacheinander folgenden Modellzeitpunkten (bezogen auf die Zeitfortschaltung des Instanzennetzsimulators) aufgrund von Zeitmeldungen aufgerufen werden.

Während des zweiten Schrittes einer Taktphase, wenn die Schaltnetzausgänge sich ändern, ruft der Koordinator aufgrund von Umgebungsänderungen die Master-Instanzen auf, die

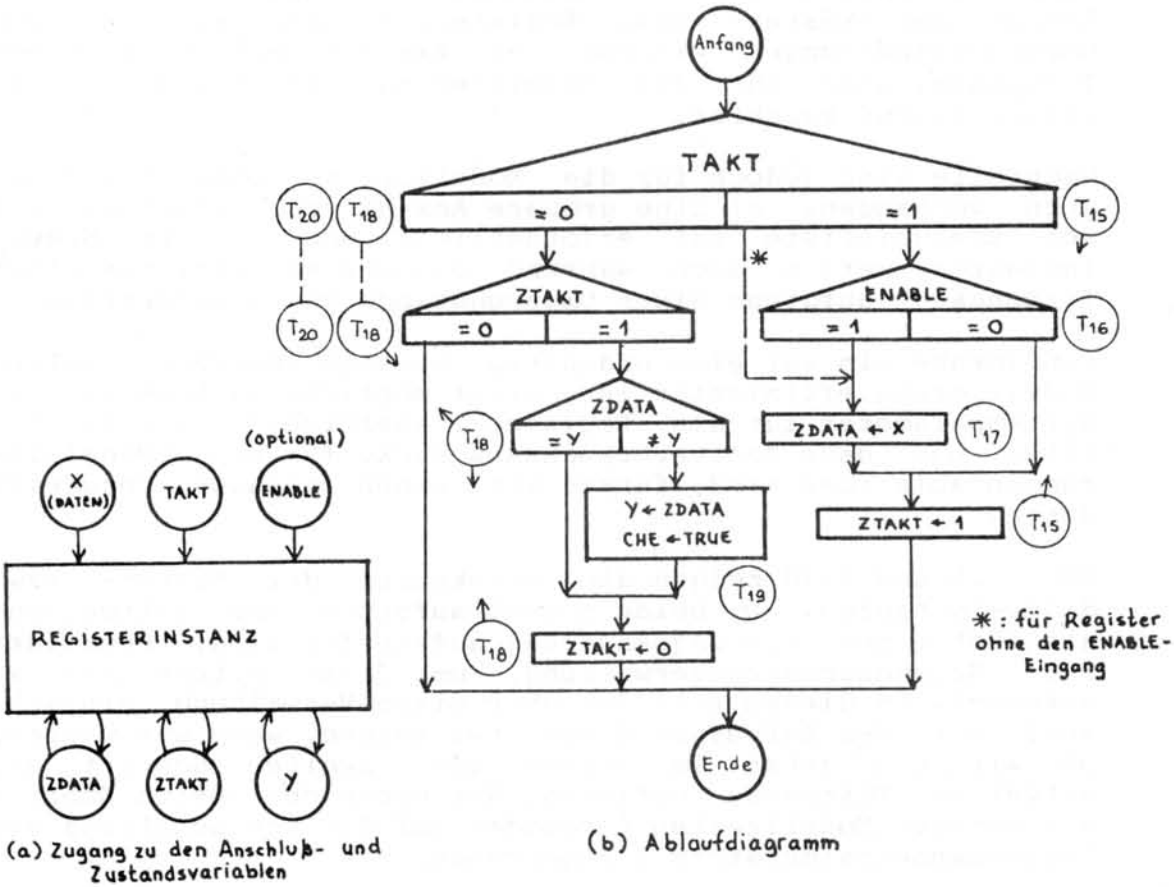


Abb. 5.8 - Registerinstanz für die Modellierung mit explizitem Taktgeber

von einer Änderung betroffen werden. Diese Instanzen geben dem Koordinator eine Zeitmeldung für den Modellzeitpunkt ab, zu dem das entsprechende Register geladen werden muß. Während des ersten Schrittes der nächsten Taktphase aktiviert dann der Koordinator nur diejenige Master-Instanzen, die eine Zeitmeldung für diesen Zeitpunkt abgegeben haben. Falls ihr ENABLE-Eingang den Wert 1 und ihr Dateneingang sich geändert hat, meldet jede dieser Master-Instanzen eine Umgebungsänderung, die am Eingang der entsprechenden Slave-Instanz beobachtbar ist. Die Slave-Instanzen geben dem Koordinator ihrerseits eine Zeitmeldung für den nächsten Zeitpunkt ab, d.h. für den zweiten Schritt dieser Taktphase, um dann ihre Ausgänge ändern und die Verknüpfungsgliederinstanzen aufgrund von Umgebungsänderungen in Aktion bringen zu können.

In Bezug auf die Modellierung mit Taktgeber sind jetzt folgende Vorteile aufzuweisen: a) Die Taktgeberinstanzen fallen weg; b) Während des ersten Schrittes jeder Taktphase werden nur die Master-Instanzen aufgerufen, für die sich mit Sicherheit vorher der Daten- oder der ENABLE-Eingang geändert hat; und c) Während des zweiten Schrittes werden

nur die Slave-Instanzen aktiviert, die sicherlich ihre Ausgänge ändern werden. Beide Modellierungen bewirken den Aufruf der Master- bzw. Registerinstanzen aufgrund von Umgebungsänderungen während des zweiten Schrittes einer Taktphase, aber für die Modellierung mit Taktgeber ist dieser Aufruf zwecklos.

Nachteile sind jedoch für die Modellierung ohne Taktgeber auch vorhanden: a) Eine größere Anzahl von Operationen auf der Ereignisliste ist erforderlich; und b) Die Slave-Instanzen werden auch während des ersten Schrittes einer Taktphase (aufgrund einer Umgebungsänderung) aufgerufen.

Von vornherein ist eine endgültige Aussage darüber, welche Modellierung effizienter ist, nicht möglich. Nachdem wir die Systemparameter für die Register-Transferebene vorgestellt haben und dann Zeitverbrauchsausdrücke für beide Modellierungen ableitbar sind, führen wir einen genauen Vergleich durch.

Abb. 5.9 und 5.10 zeigen die Strukturen der Master- bzw. Slave-Instanzen. Da beide sowohl aufgrund einer Zeitmeldung als auch einer Umgebungsänderung aufzurufen sind, benötigen sie Sequenzermarkenverwaltung, um diese Situationen zu erkennen. In diesem Fall ist aber diese Verwaltung einfach, weil wir den Aufrufgrund ableiten können, wenn wir wissen, ob wir uns jetzt im ersten oder zweiten Schritt der aktuellen Taktphase befinden. Wir haben dem ersten Schritt die geraden Modellzeiten (bezogen auf die Fortschaltung des Instanzennetzsimulators) zugeordnet.

Die Master-Instanz braucht eine zusätzliche Zustandsvariable ZFLAG, die angibt, ob die Instanz eine Zeitmeldung für die nächste dem Register entsprechende Ladephase dem Koordinator abgegeben hat. Die Variable wird rückgesetzt, wenn die Instanz aufgrund einer Zeitmeldung aktiviert wird. Damit vermeiden wir, daß eine Master-Instanz, aufgrund mehrerer innerhalb einer Taktperiode stattgefundenen Eingangsänderungen, mehrere Zeitmeldungen dem Koordinator abgibt.

Die Master-Instanzen müssen außerdem den Modellzeitpunkt berechnen, der ihrer nächsten Ladephase entspricht. Dafür hat jede Master-Instanz eine Zustandsvariable RLZ mit konstantem Wert, die die relative (auf eine allgemeine Taktperiode bezogene) Ladezeit des entsprechenden Registers angibt. RLZ ist eine gerade Zahl zwischen 0 und 2*F-1, wobei F die Anzahl der Taktphasen in einer Taktperiode ist. Bei einem Aufruf aufgrund einer Umgebungsänderung berechnet die Master-Instanz die relative (auf die laufende Taktperiode bezogene) Modellzeit RAZ des Aufrufes. Ein Vergleich zwischen RLZ und RAZ klärt, ob die Ladephase des Registers in der aktuellen Taktperiode schon vorbei ist oder nicht. Dementsprechend veranlaßt die Instanz die Eintragung einer Ereignisnotiz für die nächste bzw. die laufende Taktperiode.

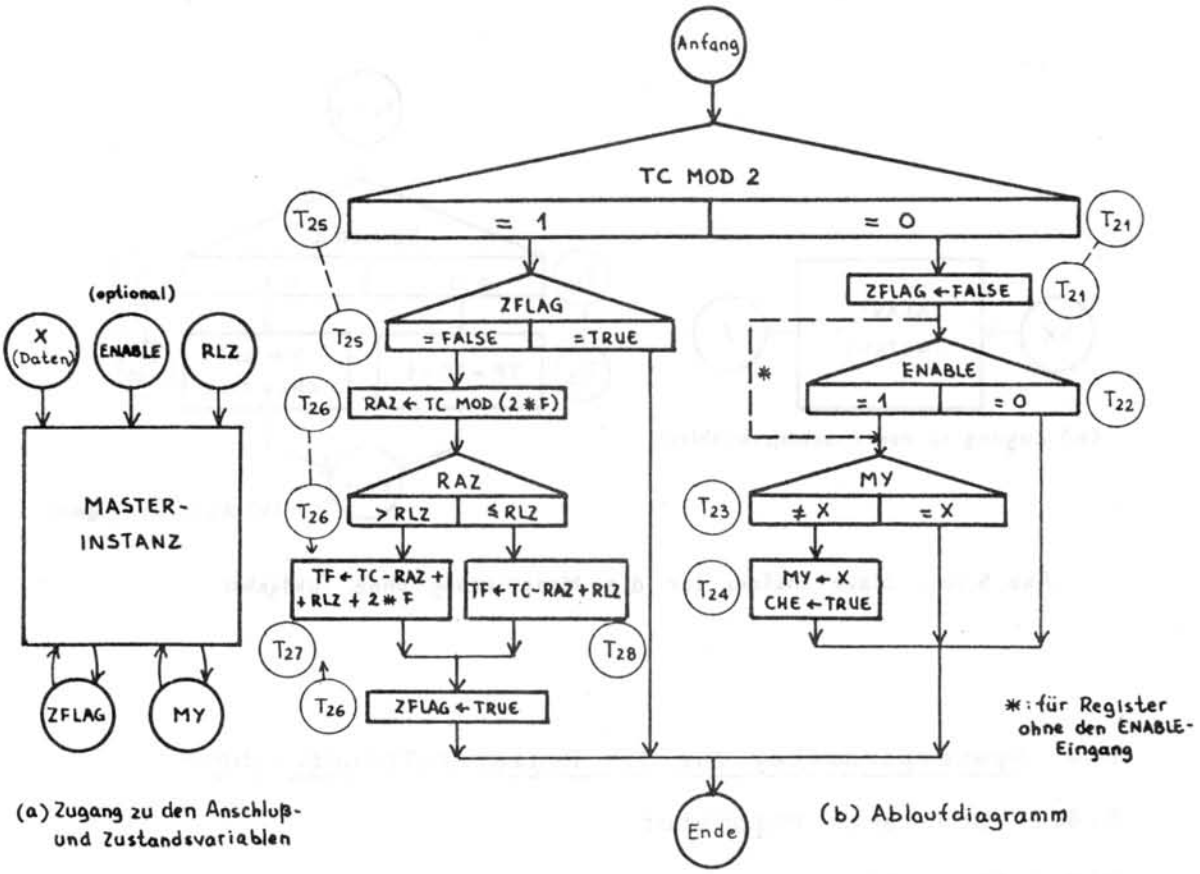


Abb. 5.9 - Master-Instanz für die Modellierung ohne Taktgeber

Wie in der Modellierung mit Taktgeber schließt diese Modellierung Erkennungskonflikte nicht aus, die es zwischen Aufrufen von Slave-Instanzen aufgrund von Zeitmeldungen und Aufrufen von Verknüpfungsgliederinstanzen aufgrund von Umgebungsänderungen geben kann. Wie auch dort sind diese Konflikte in der Praxis allerdings ausgeschlossen, weil der Koordinator alle Aufrufe aufgrund von Zeitmeldungen bearbeitet, bevor er irgendeine Instanz aufgrund einer Umgebungsänderung aktiviert.

Wie bereits gesagt, muß die Implementierung des Instanzennetzsimulators unberücksichtigt bleiben. Deswegen muß der Modellierer die mögliche Existenz solcher Konflikte in Betracht ziehen. Wir haben auch schon gesagt, daß Erkennungskonflikte das Endergebnis der Simulation nicht beeinflussen, und daß durch diese Konflikte nur die Effizienz beeinträchtigt wird. Könnte der Modellierer die Implementierung berücksichtigen, hätten wir die Trennung zwischen Entscheidungs- und Übergangsintervall nicht durch zwei verschiedene Modellzeiten schaffen müssen, weil der Koordinator zunächst alle Master-Instanzen aufgrund von Zeitmeldungen aktiviert und erst dann die Slave-Instanzen

aufgrund von Umgebungsänderungen. Die Effizienz unserer Modellierung würde damit bedeutend größer sein.

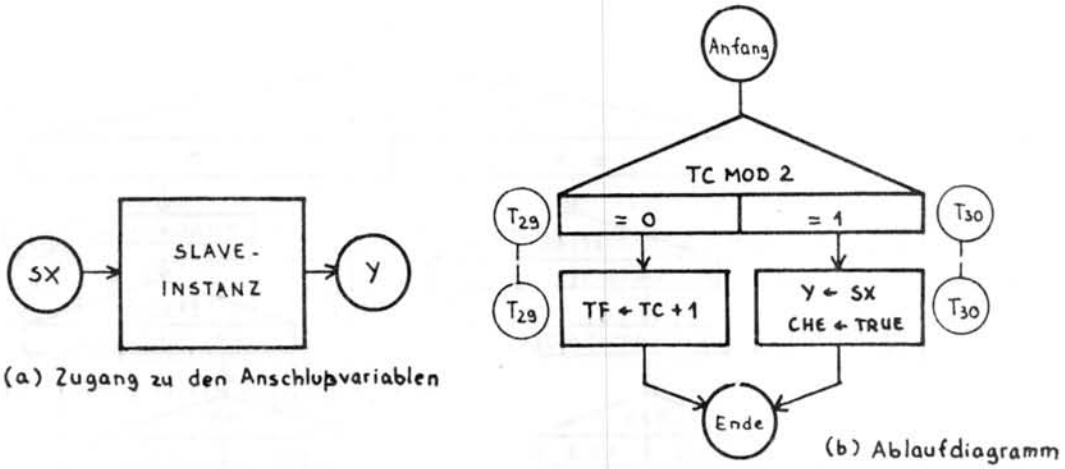


Abb. 5.10 - Slave-Instanz für die Modellierung ohne Taktgeber

5.4 Systemparameter für die Register-Transferebene

5.4.1 Statische Parameter

Wir definieren
F = 1..3
als die Anzahl der Taktphasen in einer Taktperiode. Obwohl ein größerer Wert von F denkbar wäre, werden wir zeigen, daß für den Vergleich der Instanzennetzmodellierung mit dem Simulator RT1 der Einfluß von F gering ist, während RT2 ein Modell ist, das nur für F=1 sinnvoll ist.

Nr = 10 .. 40
ist die Anzahl der Register und
Ng = 50 .. 400
die Anzahl der Verknüpfungsglieder im System. Eine wichtige Rolle in der Untersuchung spielt das Verhältnis
Ng/Nr = 5 .. 20.

Die durchschnittliche Anzahl der mit einem Registerausgang verbundenen Elemente ist als
Ur = 2 .. 3
definiert. Dementsprechend gilt die Definition
Ug = 2 .. 3
für die Verknüpfungsglieder.

Wir führen die Parameter
gr und gg
ein, die die Wahrscheinlichkeiten darstellen, daß ein Fan-Out-Element eines Registers bzw. eines Verknüpfungsgliedes ein Verknüpfungsglied ist. Wir vereinfachen unsere Analyse,

indem wir immer gr = gg = g annehmen. Folglich ist (1-g) die Wahrscheinlichkeit, daß ein Fan-Out-Element eines beliebigen Elementes ein Register ist. Aus reinem statistischen Gesichtspunkt ist zu vermuten, daß im Durchschnitt

$$g = Ng / (Ng + Nr) \qquad (5.1)$$

gilt.

Die Rate der mit einem ENABLE-Eingang versehenen Register ist

$e = 0 .. 1.$

Da die Parameter F, Nr, Ng, Ur, Ug, g und e topologische Eigenschaften darstellen, die von der Simulation unabhängig sind, sind ihre Werte für alle Modellierungen identisch.

Wir definieren weiter

$Z = 10 .. 40$

als den durchschnittlichen Zeitverbrauch der Auswertung der logischen Funktion eines Verknüpfungsgliedes, und

$Yg = 3 .. 6$

als die durchschnittliche Anzahl der Zugriffe auf die Anschlußvariablen in der Datenstruktur während dieser Auswertung. Wie in der Untersuchung für die Gatterebene betrachten wir den Zeitverbrauch der Zugriffe selbst getrennt von Z. Da wir auch für die Instanzennetzmodellierung datenstrukturabhängige Instanzenprozeduren verwenden, wird es für alle Modellierungen nur auf die für eine bestimmte Auswertung notwendigen Anschlußvariablen zugegriffen. Damit sind die Werte von Z und Yg für alle Modellierungen identisch. (Siehe auch Bemerkung über die Prozeduraufschreibung der Verknüpfungsgliederinstanzen am Ende des Abschnittes 5.3.1 .)

5.4.2 Dynamische Parameter

Wir definieren alle dynamischen Parameter in Bezug auf das reale zu simulierende System. Für die einzelnen Modellierungen berücksichtigen wir dann, inwieweit sich der Simulationsverlauf vom realen Geschehen unterscheidet.

Die Parameter q, Me und Ms, die die Operationen auf der Ereignisliste für den Instanzennetzsimulator widerspiegeln, behalten ihre schon angegebenen Definitionen (siehe Abschnitt 4.6.3).

5.4.2.1 Registerblock

In diesem Abschnitt führen wir die Parameter ein, die die Änderungen innerhalb des Registerblocks widerspiegeln.

Wir definieren

$r = 0,2 .. 0,8$

als die durchschnittliche Rate der Register, für die minde-

stens eine Eingangsänderung (Daten oder ENABLE) während einer Taktperiode stattfindet,

s = 0,2 .. 0,64

als die durchschnittliche Rate der Register, deren Ausgänge sich während einer Taktperiode ändern, und

z

als die durchschnittliche Rate der Register, die einen ENABLE-Eingang mit Wert 1 in der Taktphase haben, in der sie geladen werden.

Aus diesen Definitionen ergibt sich, daß r >= s immer gelten muß. Es ist sinnvoll, die drei eingeführten Parameter als Funktionen des Parameters e und von elementaren Raten b1 bis b7 zu definieren, wie das Schema in Abb. 5.11 zeigt. Mit ihrer Hilfe können wir Beziehungen zwischen diesen Parametern leichter erkennen. Es ist auch zweckmäßig, r und z als

$$r = r1 + r2, \tag{5.2a}$$
$$\text{wobei } r1 = e * (b1 + b2 + b3) \tag{5.2b}$$
$$\text{und } r2 = (1 - e) * b6 \tag{5.2c}$$

$$\text{und } z = z1 + z2, \tag{5.3a}$$
$$\text{wobei } z1 = e * (b1 + b2) \tag{5.3b}$$
$$\text{und } z2 = e * b4 \tag{5.3c}$$

zu definieren. Es ist leicht zu sehen, daß

$$s = e*b1 + (1-e)*b6 \tag{5.4}$$

gilt.

<table>
<tr><td rowspan="5">Register, die einen ENABLE-Eingang haben
e * Nr</td><td rowspan="3">ENABLE- oder Daten-Eingang wurde während einer Taktperiode geändert
$r_1 * Nr$</td><td>ENABLE = 1, Daten geändert
$b_1 * e * Nr$</td><td rowspan="3">$\sum_{i=1}^{5} b_i = 1$</td></tr>
<tr><td>ENABLE = 1, Daten ungeändert
$b_2 * e * Nr$</td></tr>
<tr><td>ENABLE = 0
$b_3 * e * Nr$</td></tr>
<tr><td rowspan="2">Weder der ENABLE- noch der Daten-Eingang haben sich geändert</td><td>ENABLE = 1
$b_4 * e * Nr$</td></tr>
<tr><td>ENABLE = 0
$b_5 * e * Nr$</td></tr>
<tr><td rowspan="2">Register, die keinen ENABLE-Eingang haben
(1-e) * Nr</td><td colspan="2">Daten-Eingang hat sich während einer Taktperiode geändert</td><td>$b_6 * (1-e) * Nr$ oder $r_2 * Nr$</td><td rowspan="2">$\sum_{i=6}^{7} b_i = 1$</td></tr>
<tr><td colspan="2">Daten-Eingang hat sich nicht geändert</td><td>$b_7 * (1-e) * Nr$</td></tr>
</table>

Abb. 5.11 - Beziehungen zwischen den Parametern, die die Änderungen innerhalb des Registerblocks widerspiegeln

Ein zusätzlicher Parameter ist im Zusammenhang mit dem Einfluß der Erkennungskonflikte notwendig. Stellen wir uns vor, daß der Eingang eines Registers R mit dem Ausgang eines Verknüpfungsgliedes G verbunden ist. Falls der Ausgang von G in zwei verschiedenen Taktphasen sich ändert, die zwischen zwei aufeinanderfolgenden Ladephasen von R liegen, wird die Registerinstanz R in beiden Phasen in der Instanzennetzmodellierung aufgrund von Umgebungsänderungen aufgerufen, obwohl sie nur einmal eine Zeitmeldung dem Koordinator abgibt. Unter Verwendung der Register-Transfer-Simulatoren wird R jedoch nur einmal in die Rufliste eingetragen. Dementsprechend wird seine Funktion nur einmal ausgewertet. Wir definieren

v' = 0 .. 0,6

als die durchschnittliche Rate der in einer Taktperiode von Eingangsänderungen betroffenen Register (% von r*Nr), deren Eingängen sich sich in mehr als einer Taktphase ändern. Um die Aufschreibung der Zeitverbrauchsausdrücke einfacher zu machen, verwenden wir immer

v = v' + 1.

5.4.2.2 Verknüpfungsblock

Wir definieren

t = 0,35 .. 0,6

als die durchschnittliche Rate der Verknüpfungsglieder, für die mindestens eine Eingangsänderung während einer Taktperiode stattfindet, und

p = 0,25 .. 0,5

als die durchschnittliche Rate der Verknüpfungsglieder, die im Laufe einer Taktperiode ihre Ausgänge ändern. Diese Definitionen beziehen sich auf eine Taktperiode und nicht auf eine Taktphase. Da die Verknüpfungsglieder im Gegensatz zu den Registern in jeder Phase Eingangs- und Ausgangsänderungen haben können, müssen wir für die Auswertung der Parameter t und p ein Verknüpfungsglied mehrmals mitzählen, wenn es mehrmals Anschlußänderungen in derselben Taktperiode hat.

Der Parameter p auf der Register-Transferebene ist direkt mit dem gleichnamigen Parameter auf der Gatterebene vergleichbar, aber sein Wert muß bedeutsam größer sein. Wenn x Gatter ein Verknüpfungsglied bilden, müßte rein statistisch gesehen p(RT) = x*p(G) gelten.

Den schon in der Gatterebenen-Simulation eingeführten Parameter a verwenden wir auch hier, mit einer leichten Anpassung. Wir definieren

a = 2/3 .. 5/6

als die durchschnittliche Wahrscheinlichkeit, daß der Ausgang eines Verknüpfungsgliedes sich aufgrund einer oder mehrerer gleichzeitiger Eingangsänderungen ändert. Obwohl a auch hier eine Funktion der Anzahl der Eingänge ist, ist eine exakte mathematische Formel wegen der komplexeren logischen Funktionen der Verknüpfungsglieder schwieriger zu bekommen. Wir können aber sagen, daß der Wert von a für

Register-Transfer-Systeme wesentlich höher liegen muß als für Gatterebenen-Systeme: Wenn von 8 Gattern sich nur ein Ausgang ändert, haben wir a=1/8 auf der Gatterebene, aber falls auf der Register-Transferebene diese 8 Gatter in einem einzigen komplexen Verknüpfungsglied zusammengestellt sind, hat a den Wert 1.

Aufgrund der Definitionen erhalten wir die Gleichung

$$p = a * t \qquad (5.5)$$

In den Modellierungen, die Nullpulse an den Ausgängen von Verknüpfungsgliedern aufgrund von Erkennungskonflikten erzeugen können, werden die Funktionen einiger Verknüpfungsglieder zwei- oder sogar mehrmals im Laufe einer Taktphase ausgewertet. Wir führen den Parameter

$$k = 1 \ldots 1{,}6$$

ein, so daß k*p die durchschnittliche Rate der Verknüpfungsglieder ist, die im Laufe der Simulation einer Taktperiode eine Ausgangsänderung haben, falls der Simulator die Nullpulse nicht unterdrückt.

Wir nehmen an, daß der Parameter k sich auch über t anwenden lasse, so daß k*t die durchschnittliche Rate der Verknüpfungsglieder ist, für die im Laufe der Simulation einer Taktperiode eine Eingangsänderung stattfindet. Wegen der Verbindungen zwischen dem Schaltnetz und den Registern muß das nicht unbedingt der Fall sein, aber es ist eine annehmbare Annäherung, weil Nr wesentlich kleiner als Ng ist. Wenn wir dann zwei Parameter p' und t' einführen, die die Entsprechungen von p und t für die Simulatoren sind, die Nullpulse nicht unterdrücken, gelten die Beziehungen

$$p' = k * p \,, \qquad (5.6)$$

$$t' = k * t \qquad (5.7)$$

und

$$p' = a * t'. \qquad (5.8)$$

5.4.2.3 Auswirkung von Ausgangsänderungen

Ähnlich wie in der Simulation der Gatterebene definieren wir

$$Wr \text{ und } Wg$$

als die durchschnittliche Anzahl der mit dem Ausgang eines Registers bzw. Verknüpfungsgliedes verbundenen Elemente, deren Namen aufgrund einer Änderung an diesem Ausgang in die Rufliste gebracht werden, d.h., die wir während der Verarbeitung der UMG- bzw. FANOUTLISTE mit FLAG = FALSE finden. Auch hier muß $W =< U$ gelten. Diese Parameter sind nur für das Modell RT1 und die Instanzennetzmodellierung sinnvoll, denn im Modell RT2 ist die "Rufliste" anders gebildet, und die Tatsache, ob das CALLBIT gesetzt ist oder nicht, den Simulationsverlauf nicht beeinflußt.

Für RT1 teilen wir Wr und Wg in jeweils zwei Parameter folgendermaßen auf:

$$Wr = Wrr + Wrg, \qquad (5.9)$$

wobei Wrr (bzw. Wrg) die durchschnittliche Anzahl der Register (bzw. Verknüpfungsglieder) ist, die von einer

Ausgangsänderung eines Registers betroffen werden, und

$$Wg = Wgr + Wgg, \qquad (5.10)$$

wobei Wgr (bzw. Wgg) die Entsprechungen von Wrr und Wrg für die Ausgangsänderung eines Verknüpfungsgliedes sind.

Die Erzeugung der Rufliste und die Verarbeitung der eingetragenen Elemente sind für die Verknüpfungsglieder identisch im RT1 und in der Instanzennetzmodellierung. Deswegen müssen die Signaländerungen an den Eingängen derselben Verknüpfungsglieder beobachtbar sein:

$$Wrg\ (IN) = Wrg\ (RT1) \qquad (5.11)$$

und $$Wgg\ (IN) = Wgg\ (RT1). \qquad (5.12)$$

Dies ist für die betroffenen Register nicht mehr gültig. Die Namen der Register, die in zwei Phasen einer Taktperiode eine Eingangsänderung haben, werden im RT1 nur einmal in die Rufliste eingetragen. Bei der zweiten Änderung hat die Variable FLAG eines betroffenen Registers noch den Wert TRUE. In der Instanzennetzmodellierung dagegen werden diese Namen zweimal in die Rufliste eingetragen, und der Koordinator ruft die entsprechenden Registerinstanzen zweimal auf. Deswegen gelten

$$Wrr\ (IN) = v * Wrr\ (RT1) \qquad (5.13)$$

und $$Wgr\ (IN) = v * Wgr\ (RT1), \qquad (5.14)$$

wobei wir annehnmen, daß v gleichermaßen Wrr und Wgr beeinflußt.

Nach den Definitionen ist es notwendig, daß die Bedingungen

$$Wrg =< g * Ur,$$
$$Wrr =< (1-g) * Ur,$$
$$Wgg =< g * Ug \qquad (5.15)$$

und $$Wgr =< (1-g) * Ug$$

eingehalten werden.

Im weiteren Verlauf dieser Arbeit bezieht sich eine Referenz auf einen Parameter W, falls nicht anders vermerkt, immer auf den Wert dieses Parameters für das Modell RT1.

5.4.3 Durch Nullpulse betroffene Register

Eigentlich sollten wir noch einen Parameter x einführen, der die durchschnittliche Rate der Register darstellt, die einen aufgrund eines Erkennungskonflikts erzeugten Nullpuls an ihren Eingängen haben. Diese Tatsache bedeutet sowohl für den Simulator RT1 als auch für die Instanzennetzmodellierung eine Erhöhung der durch Umgebungsänderungen betroffenen Register gegenüber dem Simulator RT2 und folglich auch eine irrtümliche Erhöhung der Anzahl der Register, die in ihren zugehörigen Taktphasen als ladbar bezeichnet werden. Auf die Richtigkeit des Simulationsverlaufes hat diese Möglichkeit allerdings keinen Einfluß.

Wie groß ist der Wert dieses Parameters? Im schlimmsten Fall könnten wir uns vorstellen, daß alle aufgrund der Auswirkung von Erkennungskonflikten mehrfach verarbeiteten Verknüpfungsglieder einen Nullpuls an ihren Ausgängen haben:

p*Ng Verknüpfungsglieder haben eine Ausgangsänderung in jeder Taktperiode,
(k-1)*p*Ng werden mehrfach aufgrund der Auswirkung von Erkennungskonflikten verarbeitet, und
Wgr*(k-1)*p*Ng Register werden von diesen Änderungen betroffen.

Wenn wir x als eine Rate von r*Nr definieren, muß diese Anzahl der betroffenen Register gleich x*r*Nr sein, und daraus ergibt sich

$$x = \frac{Wgr*(k-1)*p*Ng}{r*Nr} . \qquad (5.16)$$

Für alle hier untersuchten Systeme finden wir den Wert 0,3 als Obergrenze für den Ausdruck (5.16).

Die Realität ist aber anders. Von allen aufgrund der Auswirkung von Erkennnungskonflikten mehrfach verarbeiteten Verknüpfungsgliedern hat nur ein winziger Teil einen Nullpuls am Ausgang. Wenn wir ein Gatter mit zwei Eingängen betrachten, das zwei Eingangsänderungen hat und dadurch zweimal den Ausgang ändert, liefern aus reinem statistischen Gesichtspunkt nur 2 von 8 möglichen Ausgangskombinationen einen Nullpuls. Nennen wir dies die Wahrscheinlichkeit Wa. Wenn wir darüber hinaus b Bit breite Verknüpfungsglieder der Register-Transferebene betrachten, ist die Wahrscheinlichkeit, daß zwei gleichzeitige Eingangsänderungen dasselbe Ausgangsbit ändern, auch sehr gering. Nennen wir dies die Wahrscheinlichkeit Wb. Die Multiplikation des Ausdrucks (5.16) mit diesen zwei Wahrscheinlichkeiten Wa und Wb läßt für den Parameter x einen sehr kleinen Wert, den wir im weiteren Verlauf dieser Arbeit vernachlässigen.

5.4.4 Gleichungen der Systemdynamik

Gleichungen (5.17) und (5.18) spiegeln die Gleichheit zwischen der Anzahl der von Eingangsänderungen betroffenen Verknüpfungsglieder (bzw. Register) und der Anzahl der aufgrund dieser Änderungen verarbeiteten Verknüpfungsglieder (bzw. Register) im Laufe einer ganzen Taktperiode wider. Sie gelten nur für den Simulator RT1 und die Instanzennetzmodellierung. Für den Simulator RT2 sind die Parameter W nicht definierbar, aber ähnliche Gleichungen unter Verwendung von p und t wären ableitbar.

$$Wrg*s*Nr + Wgg*p'*Ng = t'*Ng \qquad (5.17)$$

$$Wrr*s*Nr + Wgr*p'*Ng = r*Nr \qquad (5.18)$$

Wir können die Gleichung (5.17) folgendermaßen interpretieren: Nach den Definitionen der Parameter haben wir für eine Taktperiode
s*Nr Register, die ihre Ausgänge ändern,
Wrg*s*Nr Verknüpfungsglieder, die von diesen Änderungen

betroffen werden,
p'*Ng Verknüpfungsglieder, die ihre Ausgänge ändern, und
Wgg*p'*Ng Verknüpfungsglieder, die von diesen Änderungen betroffen werden.
Die Summe der betroffenen Verknüpfungsglieder muß identisch sein mit t'*Ng, die die Anzahl der verarbeiteten Verknüpfungsglieder darstellt.

In ähnlicher Weise läßt sich (5.18) interpretieren. In der Tat lautet sie für die Instanzennetzmodellierung
v*Wrr*s*Nr + v*Wgr*p'*Ng = v*r*Nr,
wo Wrr und Wgr die Werte für den Simulator RTl haben. Da v allen Gliedern der Gleichung gemeinsam ist, gilt (5.18) auch für die Instanzennetzsimulation.

5.4.5 Beziehungen, die nicht mathematisch erfaßt sind

Es gibt andere Parameterbeziehungen, die wir intuitiv erklären können, aber deren mathematische Erfassung entweder sehr schwierig ist oder unsere Vergleichsuntersuchung unnötigerweise erschweren würde. Während unserer Suche nach Grenzsystemen durch Variationen der Parameterwerte versuchen wir trotzdem, diese Beziehungen zu berücksichtigen.

Die Parameter Z, Yg und a sind Funktionen der im System vorhandenen Verknüpfungsgliedertypen und ihrer Anteile in Ng:
Z und Yg sind desto größer, je komplexer die Verknüpfungsglieder sind;
a ist desto größer, je größer die Breite b der Eingänge ist; und
a ist desto kleiner, je größer die Anzahl der Eingänge ist.

Es ist auch zu erwarten, daß, je komplexer die Verknüpfungsglieder sind, d.h., je höher die Abstraktion der Register-Transferebene bezüglich der Gatterebene ist, desto kleiner Ng/Nr ist. Es besteht deswegen auch eine Beziehung zwischen Ng/Nr und den Parameter Z,Yg und a.

Für eine bestimmte Anzahl s*Nr von aktiven Registern ist der Parameter t eine Funktion der Anzahl F der Taktphasen, wie Abb. 5.12 veranschaulicht. Ein ganz einfaches Register-Transfer-System hat Nr=2, s=1 und Ng=3. Falls das System eine einzige Taktphase hat, hat t dann den Wert 1. Falls aber jedes Register in einer anderen Taktphase geladen wird, ändert sich der Eingang von V2 zweimal in einer Taktperiode, und deswegen ist t gleich 4/3.

Auch der Parameter v ist eine Funktion der Anzahl der Taktphasen. Aus der Definition folgt unmittelbar, daß v den Wert 1 haben muß, falls F gleich 1 ist. Es ist zudem zu erwarten, daß v mit F wächst.

Zuletzt ist der Faktor k eine Funktion der Änderungswahrscheinlichkeit a, und deswegen auch der Komplexität der Verknüpfungsglieder, wie Abb. 5.13 anhand eines einfachen

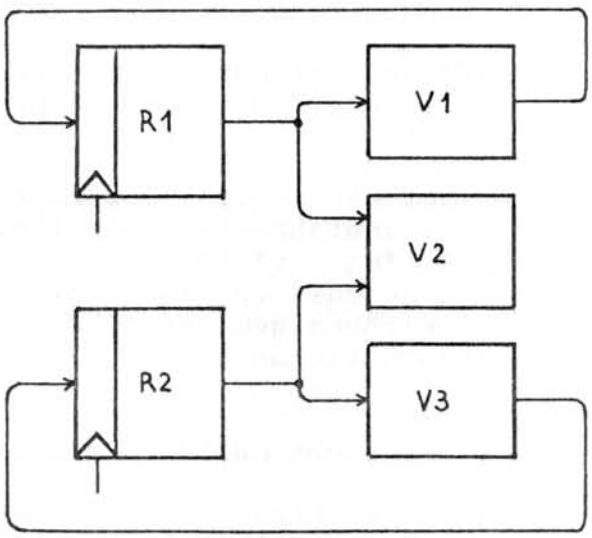


Abb. 5.12 - Beispiel eines Register-Transfer-Systems für die Erläuterung der Abhängigkeit des Parameters t von der Anzahl F der Taktphasen

Beispiels veranschaulicht. Der Simulator RT2 verwendet sicherlich die Auswertungsreihenfolge 1-2-3, und die Anzahl der Verknüpfungsglieder, deren Funktionen ausgewertet werden, ist 1+2*a, wobei angenommen wird, daß der Simulator die Funktion des Elementes V1 immer auswertet und alle Elemente dieselbe Änderungswahrscheinlichkeit haben. Für das Modell RT1 und die Instanzennetzmodellierung können wir behaupten, daß die Reihenfolgen 1-2-3 und 1-3-2-3 gleich möglich seien. Damit ist die Anzahl der Verknüpfungsglieder, deren Funktionen ausgewertet werden, gleich

$(1 + 2*a)/2 + (1 + 2*a + a^2)/2.$

Daraus entsteht

$$k = \frac{\text{Anzahl der im RT2 verarbeiteten Glieder}}{\text{Anzahl der im RT1 verarbeiteten Glieder}} = 1 + \frac{a^2}{2 + 4*a}$$

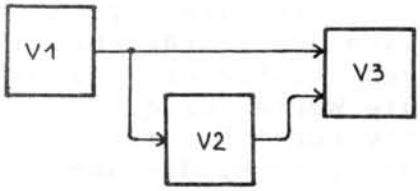
Wir sehen, daß, je größer a ist, d.h. je komplexer die Verknüpfungsglieder sind, desto größer auch der Wert des Parameters k ist.

5.4.6 Grundsystem

Das hier durch einen Satz von Parameterwerten zu definierende Grundsystem S1 ist der Anfangsarbeitspunkt für die Untersuchung der Sensibilität (bezüglich der Parameter) des Gewinns der Register-Transfer-Simulatoren gegenüber dem Instanzennetzsimulator.

Wir fangen an mit einer willkürlichen Festlegung der Werte der statischen Parameter:

F=2, Nr=18, Ng=180, so daß Ng/Nr=10,



(a) Beispielsystem

Auswertungs-reihenfolge	Wahrscheinlichkeit für		
	V1	V2	V3
V1-V2-V3	1	a	a
V1-V3-V2-V3	1	a	$a+a^2$

(b) Wahrscheinlichkeit, daß die Funktion eines Verknüpfungsgliedes ausgewertet wird

a	K
0,7	1,102
0,8	1,123
0,9	1,145

(c) $K=f(a)$

Abb. 5.13 - Abhängigkeit des Parameters K von der Änderungswahrscheinlichkeit a

Ur=Ug=2,2 , g=10/11 (nach (5.1)),
Z=20, Yg=4, e=0,5.

Ebenfalls willkürlich wählen wir die Werte der Raten b aus, die die Änderungen innerhalb des Registerblocks widerspiegeln:

Register mit ENABLE
- Eingang geändert 0,4
  - ENABLE=1, Daten geändert (b1=0,2)
  - ENABLE=1, Daten ungeändert (b2=0,1)
  - ENABLE=0 (b3=0,1)
- Eingang ungeändert 0,6
  - ENABLE=1 (b4=0,3)
  - ENABLE=0 (b5=0,3)

Register ohne ENABLE
- Daten geändert (b6=0,2)
- Daten ungeändert (b7=0,8)

Aus diesen Werten und den Gleichungen (5.2) und (5.4) entnehmen wir
r=0,3 (r1=0,2 und r2=0,1),
s=0,2 und
z=0,3 (z1=0,15 und z2=0,15).

Für die Parameter, die die Aktivitäten innerhalb des Verknüpfungsblocks darstellen, können wir auch eine willkürliche Entscheidung treffen:
a=0,8 , t=0,25 und k=1,4.
Aus (5.5)-(5.8) folgt dann
p=0,2 , p'=0,28 und t'=0,35.

Wir können jetzt Werte für die W-Parameter finden, die die Gleichungen (5.17) und (5.18) lösen. Da wir 2 Gleichungen aber 4 Variablen haben, sind Freiheitsgrade für die Festlegung der Werte vorhanden. Wir nehmen
Wrr(RT1)=0,1 , Wrg=1,4 , Wgr(RT1)=0,14 und Wgg=1,15.
Diese Werte erfüllen die Bedingungen (5.15).

Da F > 1 ist, können wir v > 1 auswählen. Wir legen
v=1,3
fest, und aus (5.13) und (5.14) resultieren
Wrr(IN)=0,13 und Wgr(IN)=0,182,
die die Bedingungen (5.15) ebenfalls erfüllen.

Wir können die Werte der Parameter q, Me und Ms noch nicht festlegen. Im nächsten Abschnitt untersuchen wir die aus der Instanzennetzmodellierung von Register-Transfer-Systemen resultierende Belegung der Ereignisliste des Instanzennetzsimulators und stellen Beziehungen zwischen diesen Parametern fest. Dadurch können wir die Werte für q, Me und Ms finden, die den minimalen Zeitverbrauch für die Aktivitäten der Instanzennetzsimulation liefern, die auf die Ereignisliste bezogen sind.

Für unsere Untersuchungen sind viele der Parameter unwichtig. Aufgrund der dynamischen Gleichungen werden wir z.B. die W-Parameter aus den Zeitverbrauchsausdrücken verschwinden lassen. Andere Parameter haben geringen Einfluß auf den Zeitverbrauch. Wichtig sind F, k, v und die Quantitäten, die die Systemaktivität in einer Taktperiode widerspiegeln:

r*Nr = 5,4 ist die durchschnittliche Anzahl der betroffenen Register,
s*Nr = 3,6 die Anzahl der Register, deren Ausgänge sich ändern,
t*Ng = 45 die Anzahl der betroffenen Verknüpfungsglieder, und
p*Ng = 36 die Anzahl der Verknüpfungsglieder, deren Ausgänge sich ändern (Nullpulse ausgeschlossen).

5.5 Zeitverbrauch der Instanzennetzsimulation

5.5.1 Modellierung mit Taktgeber

Wir nehmen eine Taktperiode als unser Meßintervall. In der Tabelle 5.1 ist die Verarbeitungssequenz des Instanzennetzsimulators während eines Meßintervalls gezeigt, wenn wir Register-Transfer-Systeme mit expliziten Taktgeberinstanzen modellieren. Einem Meßintervall entsprechen die Sektionausführungszeiten

T1-T10 und Ta (Parameterzugriffszeit, siehe Abschnitt 3.4) im Koordinator (siehe Abb. 2.10),
T13-T14 in der Taktgeberinstanz (Abb. 5.7) und
T15-T20 in der Registerinstanz (Abb. 5.8).

Gemäß den definierten Parametern finden wir den in der zweiten Spalte der Tabelle angegebenen Zeitverbrauch.

Da wir die Ereignisliste am Anfang des ersten und zweiten Schrittes jeder Taktphase mit unterschiedlichen Belegungen finden, haben wir für Punkte 7 und 10 der Tabelle Ms' und Me' eingeführt, die sich von Ms bzw. Me in Punkten 1 und 4 unterscheiden.

Es ist anzumerken, daß die Aufschreibung der Instanzen in den Abbildungen 5.7 und 5.8 formale Parameternamen für die Anschluß- und Zustandsvariablen verwendet, während die tatsächlich zu implementierenden Instanzen direkt auf die Variablen in der Datenstruktur zugreifen.

Aktivitätsbeschreibung	Zeitverbrauch
ERSTER SCHRITT EINER SYSTEMÜBERFÜHRUNG	
1) Koordinator schaltet die Modellzeit fort	F*(T7 + q*(T8 + T9*Ms))
2) Koordinator findet in jeder Taktphase eine der jeweiligen Taktgeberinstanz entsprechende Notiz und ruft diese Instanz auf	F*(T10 + T1)
3) Taktgeberinstanz bringt den Taktausgang von 0 auf 1, meldet Umgebungsänderung und gibt eine Zeitmeldung ab	T13*F
4) Koordinator trägt als Folge dieser Zeitmeldung eine neue Notiz für die nächste Modellzeit in die Ereignisliste ein	F*(T2 + T3*Me)
5) Koordinator trägt die Namen aller Nr Registerinstanzen (verteilt auf die Taktphasen) in die Rufliste ein und ruft die Instanzen auf	T4*F + T5*Nr + T1*Nr
6) Registerintanzen führen ihre Master-Funktionen aus, wobei (1-e+z)*Nr von ihnen ENABLE=1 bzw. keinen ENABLE-Eingang haben und ihre Zwischenvariablen laden	T15*Nr + T16*e*Nr + + T17*(1-e+z)*Nr
ZWEITER SCHRITT EINER SYSTEMÜBERFÜHRUNG	
7) wie 1)	F*(T7 + q*(T8 + T9*Ms'))
8) wie 2)	F*(T10 + T1)
9) Taktgeberinstanz bringt den Taktausgang von 1 auf 0, meldet Umgebungsänderung und gibt eine Zeitmeldung ab	T14*F
10) Koordinator trägt als Folge dieser Zeitmeldung eine neue Notiz für die nächste Taktperiode in die Ereignisliste ein	F*(T2 + T3*Me')
11) wie 5)	T4*F + T5*Nr + T1*Nr

Tabelle 5.1 - Verarbeitungssequenz des Instanzennetzsimulators in einem Meßintervall bei der Simulation der Register-Transferebene nach dem Modell mit Taktgeber (Fortsetzung auf der nächsten Seite)

12) Registerinstanzen führen ihre Slave-Funktionen aus (ZTAKT=1), wobei s*Nr von ihnen ZDATA ≠ Y haben und Umgebungsänderung melden	T18*Nr + + T19*s*Nr
13) Wegen dieser Umg.-Änd. sucht der Koordinator die UMGLISTEn der Registerinstanzen ab, findet Ur*s*Nr Fan-Out-Namen und bringt Wr*s*Nr davon in die RL	T4*s*Nr + T5*Ur*s*Nr + + T6*Wr*s*Nr
14) Wegen der Umgebungsänderungen der Register- und der Verknüpfungsgliederinstanzen werden t'*Ng Verknüpfungsgliederinstanzen aufgerufen	T1*t'*Ng
14.1) Die Instanzen greifen auf Anschlußvariablen zu	Ta*Yg*t'*Ng
14.2) Die Instanzen werten ihre logischen Funktionen aus	Z*t'*Ng
15) p'*Ng Verknüpfungsgliederinstanzen haben Umg.-Änd. gemeldet; Deshalb sucht der Koordinator ihre UMGLISTEn ab, findet Ug*p'*Ng Fan-Out-Namen und bringt Wg*p'*Ng davon in die Rufliste	T4*p'*Ng + T5*Ug*p'*Ng + T6*Wg*p'*Ng
16) Wegen der Umg.-Änd. der Verknüpfungsglieder- und der Registerinstanzen werden v*r*Nr Registerinstanzen aufgerufen	T1*v*r*Nr
16.1) Sie alle haben ZTAKT=0 und deshalb keine Aktivität durchzuführen	T20*v*r*Nr

Tabelle 5.1 - Verarbeitungssequenz des Instanzennetzsimulators in einem Meßintervall bei der Simulation der Register-Transferebene nach dem Modell mit Taktgeber (Fortsetzung)

Die Summe dieser Teilausführungszeiten ergibt den gesamten Zeitverbrauch in einer Taktperiode, wenn die Zeiten T unter Berücksichtigung der Ausführungszeiten der Algorithmenschritte ihre Werte erhalten:

$$
\begin{aligned}
ZVIm(RT) = {} & F^* (538 + 15^*Me + 15^*Me' + q^*(232 + 15^*Ms + 15^*Ms')) + \\
& + Nr^*(247 - 5^*e + 13^*z) + 73^*v^*r^*Nr + \\
& + s^*Nr^* (23 + 16^*Ur + 15^*Wr) + t'^*Ng^* (57 + 5^*Yg + Z) + \\
& + p'^*Ng^* (13 + 16^*Ug + 15^*Wg)
\end{aligned}
\qquad (5.19)
$$

Es ist jetzt notwendig, die Ereignisliste zu dimensionieren. Wir müssen eine Intervallgröße DT=1/q festlegen, so daß wir aufgrund dieses Wertes und der durch ihn bestimmten Werte für Me und Ms einen minimalen Zeitverbrauch ZVIm(RT) finden.

In der Instanzennetzmodellierung mit Taktgeber hat die Ereignisliste eine sehr einfache Belegung, weil nur die F Taktgeberinstanzen, und immer nach einer festen Reihenfolge,

Eintragungen von Ereignisnotizen veranlassen.

Während des zweiten Schrittes einer Taktphase plant der Koordinator ein Ereignis für eine Modellzeit (in der nächsten Taktperiode), die sicherlich größer ist als alle anderen schon vorgekommenen Planzeiten, so daß Me'=0 ist, weil die Intervalle zwischen Scheinnotizen immer rückwärts verfolgt werden. Da F eine kleine Zahl ist und die Zahl der Intervalle dagegen um 30 liegt, haben wir bei der Wiedereintragung einer Scheinnotiz (immer am Ende der Ereignisliste !) Ms=0 und Ms'=0.

Nur während des ersten Schrittes der Taktphase, wenn eine Taktgeberinstanz eine Ereignisnotiz für die nächste Zeit braucht und die anderen Taktgeberinstanzen Ereignisnotizen für die nächste Taktperiode schon haben, finden wir Me ungleich Null. Aus einer Analyse aller möglichen Belegungen der Ereignisliste ergibt sich für einen Wert von q, der gleich 1/x ist, wo x eine beliebige gerade Zahl ist,

$$\text{falls } F >= x/2, \quad M = \sum_{i=0}^{\frac{x}{2}-1} i \quad \text{und} \quad Me = 2*M/x, \text{ und} \qquad (5.20)$$

$$\text{falls } F < x/2, \quad M = \sum_{i=0}^{F-1} i + (x/2-F)*(F-1) \quad \text{und} \quad Me=2*M/x. \qquad (5.21)$$

Wir müssen einen Kompromiß zwischen q und Me finden, damit die von q und Me abhängigen Glieder in (5.19) einen minimalen Zeitverbrauch liefern. Das Ergebnis dieses Kompromisses ist in der Tabelle 5.2 zusammengefaßt.

Anzahl F der Taktphasen	optimaler Wert	
	für q	für Me
1	1/32	0
2	1/32	15/16
3	1/32	29/16
4	1/32	42/16
>= 5	1/8	3/2

Tabelle 5.2 - Kompromiß zwischen q und Me für minimalen ZVIm(RT)

Der Ausdruck (5.19) läßt sich vereinfachen, indem wir
- Me' = Ms = Ms' = 0 immer annehmen dürfen und
- Wr und Wg unter Verwendung von (5.9) und (5.10) und dann von (5.17) und (5.18) eliminieren.

Wir bekommen

$$\begin{aligned} ZVIm(RT) = {} & F^* (538 + 15*Me + 232*q) + Nr*(247 - 5*e + 13*z) + \\ & + 88*v*r*Nr + s*Nr*(23 + 16*Ur) + t'*Ng*(72 + 5*Yg + Z) + \\ & + p'*Ng*(13 + 16*Ug) \end{aligned} \qquad (5.22)$$

5.5.2 Modellierung ohne Taktgeber

Einer Taktperiode entspricht die in der Tabelle 5.3 gezeigte Verarbeitungssequenz für den Instanzennetzsimulator, wenn wir Register-Transfer-Systeme ohne explizite Taktgeberinstanzen modellieren. Die Ausführungszeiten
T1-T10 und Ta im Koordinator (Abb. 2.10),
T21-T28 in der Master-Instanz (Abb. 5.9) und
T29-T30 in der Slave-Instanz (Abb. 5.10)
sind für eine Taktperiode maßgeblich. Nach den Parameterdefinitionen ergibt sich der in der zweiten Spalte der Tabelle angegebene Zeitverbrauch.

Wegen der unterschiedlichen Belegungen der Ereignisliste während der Eintragung von Notizen für die Master- und die Slave-Instanzen haben wir Mem und Mes als Ersatz für Me eingeführt. Auch für die Zeitfortschaltung müssen wir zwischen Ms und Ms' unterscheiden, die für die Fortschaltung nach dem ersten bzw. zweiten Schritt einer Taktphase maßgeblich sind.

Die Summe aller Sektionausführungszeiten ergibt folgenden gesamten Zeitverbrauch, wenn die Zeiten T nach den Ausführungszeiten der Algorithmenschritte berechnet werden:

$$
\begin{aligned}
ZVIo(RT) = \\
&F*(40 + q*(232 + 15*Ms + 15*Ms')) + \\
&+ r*Nr*(256 + 68*v + 15*Mem) + Nr*(8*r1 + 17*r2 + 17*z1) + \\
&+ s*Nr*(363 + 15*Mes + 16*Ur + 15*Wr) + \\
&+ t'*Ng*(57 + 5*Yg + Z) + p'*Ng*(13 + 16*Ug + 15*Wg)
\end{aligned}
\qquad (5.23)
$$

Eine Analyse der Belegung der Ereignisliste während der Eintragung von Master- und Slave-Ereignisnotizen ergibt die in der Tabelle 5.4 aufgeführte Abhängigkeit der Parameter Mem und Mes von F, q und r*Nr. Diese Analyse wurde vereinfacht, indem wir nur Zweier-Potenzen für DT=1/q und Werte kleiner als 3 für F berücksichtigen, was gerechtfertigt ist, weil, wie später gezeigt, größere Werte von F keine neuen Erkentnisse bringen.

Ziel der Dimensionierung von q ist die Minimierung des Zeitverbrauchs ZVE der Operationen auf der Ereignisliste (Zeitfortschaltung, Suche und Eintragung von Notizen). Dieser Zeitverbrauch beträgt

$$
\begin{aligned}
ZVE = &F*(40 + 232*q) + r*Nr*(95 + 15*Mem) + \\
&+ s*Nr*(95 + 15*Mes),
\end{aligned}
\qquad (5.24)
$$

wobei Ms=Ms'=0 angenommen wird, was wir nachher beweisen werden. Wenn wir die in der Tabelle 5.4 zusammengefaßte Kenntnis über Mes und Mem verwenden, zeigt sich ZVE als eine Funktion von F, q, r*Nr und s*Nr. Für einen festen Wert von F hängt der optimale Wert von q dann von r*Nr und vom Verhältnis zwischen r und s ab. Tabelle 5.5 faßt die optimalen Werte von q für einige in dieser Arbeit typische verwendete Werte von s/r zusammen. Für F=1 können wir immer

Aktivitätsbeschreibung	Zeitverbrauch
1) Der Koordinator schaltet die Modellzeit fort	F*(T7 + q*(T8 + T9*Ms))
2) Der Koordinator findet Notizen für r*Nr Master-Instanzen (verteilt auf die Taktphasen) und ruft diese Instanzen auf	(T10 + T1)*r*Nr
3) Die r*Nr Master-Instanzen stellen fest, daß eine Zeitmeldung der Aufrufgrund war; r1*Nr von ihnen haben ENABLE-Eingang; (r2+z1)*Nr haben ENABLE=1 oder keinen ENABLE-Eingang; s*Nr weisen den Ausgangswert X dem Ausgang MY zu und melden Umgebungsänderung	T21*r*Nr + T22*r1*Nr + T23*(r2+z1)*Nr + T24*s*Nr
4) Wegen dieser Umg.-Änd. sucht der Koordinator die Teil-UMGLISTEn der s*Nr Master-Instanzen ab, findet und bringt s*Nr Slave-Instanzennamen in die RL und ruft diese Instanzen auf	(T4 + T5 + T6 + T1)*s*Nr
5) Die s*Nr Slave-Instanzen stellen fest, daß eine Umg.-Änd. der Aufrufgrund war, und geben eine Zeitmeldung an den Koordinator ab (Ereignis für die nächste Modellzeit)	T29*s*Nr
6) Der Koordinator trägt entsprechende Ereignisnotizen in die Ereignisliste ein	(T2 + T3*Mes)*s*Nr
7) Der Koordinator schaltet die Modellzeit fort	F*(T7 + q*(T8 + T9*Ms'))
8) Der Koordinator findet Notizen für s*Nr Slave-Instanzen (verteilt auf die Taktphasen) und ruft diese Instanzen auf	(T10 + T1)*s*Nr
9) Die s*Nr Slave-Instanzen stellen fest, daß eine Zeitmeldung der Aufrufgrund war, weisen den Eingangswert SX dem Ausgang Y zu und melden Umgebungsänderung	T30*s*Nr
10) Wie 13) in der Tabelle 5.1	(T4 + T5*Ur + T6*Wr)*s*Nr
11) Wie 14) in der Tabelle 5.1	(T1 + Ta*Yg + Z)*t'*Ng
12) Wie 15) in der Tabelle 5.1	(T4 + T5*Ug + T6*Wg)*p'*Ng

Tabelle 5.3 - Verarbeitungssequenz des Instanzennetzsimulators während eines Meßintervalls bei der Simulation der Register-Transferebene nach dem Modell ohne Taktgeber (Fortsetzung auf der nächsten Seite)

13) Wegen der Umgebungsänderungen der Verknüpfungsglieder- und der Registerinstanzen werden v*r*Nr Registerinstanzen aufgerufen	(T1 + T25)*v*r*Nr
13.1) Von ihnen haben r*Nr Registerinstanzen FLAG= FALSE und geben an den Koordinator eine Zeitmeldung ab, wobei angenommen wird, daß	T26*r*Nr
13.2) eine Hälfte davon ein Ereignis noch für die laufende Taktperiode brauchen (d.h., ihre entsprechenden Taktphasen noch nicht erreicht wurden), und	T28*r*Nr/2
13.3) die andere Hälfte ein Ereignis für die nächste Taktperiode brauchen	T27*r*Nr/2
14) Der Koordinator trägt die entsprechenden Ereignisnotizen in die Ereignisliste ein	(T2 + + T3*Mem)*r*Nr

Tabelle 5.3 - Verarbeitungssequenz des Instanzennetzsimulators während eines Meßintervalls bei der Simulation der Register-Transferebene nach dem Modell ohne Taktgeber (Fortsetzung)

F	Mem	Mes
1	0	0
2	0, falls q >= 1/2 (1-2*q)*r*Nr/16, falls q =< 1/4	(1-q)*r*Nr/4
3	0, falls q >= 1/2 (5-12*q)*r*Nr/54, falls q =< 1/4	0, falls q = 1 r*Nr/9, falls q = 1/2 7*r*Nr/36, falls q = 1/4 (1-2*q)*r*Nr/3, falls q =< 1/8

Tabelle 5.4 - Abhängigkeit der Parameter Mem und Mes von F, q und r*Nr in der Instanzennetzsimulation von Register-Transfer-Systemen (nach dem Modell ohne Taktgeber)

den kleinst möglichen Wert von q nehmen, weil Mem und Mes unabhängig von q den Wert Null haben. Da wir uns auf Zweier-Potenzen beschränkt haben und eine Änderung des Wertes von q geringen Einfluß auf den gesamten Zeitverbrauch hat, wenn q eine Bruchteilzahl ist, haben wir 1/32 willkürlich als kleinst möglichen Wert für q ausgewählt.

Nach den festgelegten Wertebereichen der Parameter ist die maximale Anzahl r*Nr + s*Nr der gleichzeitig eingetragenen Ereignisnotizen immer kleiner als 60 und größer als 6. Unter

F	s/r	Wert von x = r*Nr , damit q(opt)		
		=Minimum (1)	= 1/2	= 1
2	1/2	< 11,12	11,12 =< x < 15,73	>= 15,73
	2/3	< 10,30	10,30 =< x < 13,62	>= 13,62
	4/5	< 9,76	9,76 =< x < 12,44	>= 12,44
3	1/2	< 9,14	9,14 =< x < 20,43	>= 20,43
	2/3	< 8,34	8,34 =< x < 17,70	>= 17,70
	4/5	< 7,84	7,84 =< x < 16,15	>= 16,15

Bemerkung: (1) in der Praxis q=1/32 verwendet

Tabelle 5.5 - Optimale Werte von q in Abhängigkeit von F, r*Nr und s/r in der Instanzennetzsimulation von Register-Transfer-Systemen (nach dem Modell ohne Taktgeber)

diesen Umständen und der Verwendung der Tabelle 5.5 ist nachweisbar, daß die Anzahl der benutzten Intervalle der Ereignisliste immer gering ist, so daß die letzten Intervalle immer leer bleiben. Daraus folgt Ms=Ms'=0, was grundlegend für die Analyse des optimalen Wertes von q war. Wir können jetzt den Ausdruck (5.23) vereinfachen, indem wir
- immer Ms=Ms'=0 annehmen,
- rl durch r-r2 ersetzen und
- Wr und Wg wie in der Modellierung mit Taktgeber eliminieren, und bekommen

$$ZVIo(RT) = F*(40 + 232*q) + r*Nr*(264 + 83*v + 15*Mem) + Nr*(9*r2 + 17*z1) + s*Nr*(363 + 15*Mes + 16*Ur) + t'*Ng*(72 + 5*Yg + Z) + p'*Ng*(13 + 16*Ug) \quad (5.25)$$

5.5.3 Vergleich zwischen den Instanzennetzmodellierungen

Unser Ziel besteht jetzt darin, beide Modellierungen im ganzen Spektrum der Register-Transfer-Systeme zu vergleichen und eine globale Entscheidung darüber zu treffen, welche Modellierung effizienter ist. Der Vergleich basiert auf dem Verhältnis

$$I = \frac{ZVIo(RT)}{ZVIm(RT)},$$

das ein Maßstab für den Gewinn einer der Modellierungen gegenüber der anderen ist. Da die Verarbeitung des Verknüpfungsblocks identisch für beide Modellierungen ist, beschränken wir uns zunächst auf die Analyse von

$$Ir = \frac{ZVIo'(RT)}{ZVIm'(RT)} ,$$

wo die in (5.22) und (5.25) von t' und p' abhängigen Glieder vernachlässigt werden. Wir berücksichtigen später den Einfluß der Verknüpfungsglieder.

Für die ganze Untersuchung übernehmen wir die Werte e=0,5 , r2=r/3 , z1=r/2 , z=1/4 + r/6 , s=2*r/3 und Ur=2,2 aus dem Grundsystem S1. Andere Werte dieser Parameter ändern zwar die numerischen Ergebnisse unserer Untersuchung, aber nicht ihre wesentlichen Eigenschaften.

Wegen der Nichtlinearität der Belegungen der Ereignisliste müssen wir getrennte Untersuchungen für F=1,2,3 führen.

Für F=1 gelten immer
v=1,
q'=1/32 , Me=0 (Modellierung mit Taktgeber),
q''=1/32 , Mem=0 und Mes=0 (Modellierung ohne Taktgeber),
und wir bekommen

$$Ir = \frac{47,25 + 623,967*r*Nr}{545,25 + Nr*(247,75 + 128,967*r)} . \qquad (5.26)$$

Damit Ir >= 1, d.h., die Modellierung mit Taktgeber besser ist, muß

$$498 + 247,75*Nr =< 495*r*Nr \qquad (5.27)$$

gelten. Werte von r und Nr, die diese Ungleichung erfüllen, sind in der Tabelle 5.6 exemplarisch aufgezeigt. Wir erkennen, daß die Modellierung mit Taktgeber nur besser abschneidet für Systeme mit sehr großen Werten von r oder Nr. Wieviel dieser Vorteil im Verhältnis zur Modellierung ohne Taktgeber ausmacht, zeigt Tabelle 5.7 zum Beispiel für r=0,8 : Bezüglich der Verarbeitung des Registerblocks verbraucht die Modellierung ohne Taktgeber höchstens 37% mehr Simulationszeit als die Modellierung mit Taktgeber, was der Fall für Nr=40 ist.

r	Nr, damit Ir >= 1	r	Nr, damit Ir >= 1
< 0,50	nicht vorhanden	0,7	>= 5,0
0,51	>= 106	0,8	>= 3,4
0,55	>= 20,3	1,0	>= 2,0
0,60	>= 10,1		

Tabelle 5.6 - Werte von r und Nr, die die Modellierung mit Taktgeber begünstigen

Damit eine der beiden Modellierungen den maximalen Gewinn gegenüber der anderen hat (d.h. I seinen maximalen bzw. minimalen Wert hat), muß die Verarbeitung des Verknüpfungsblocks den geringsten Einfluß auf die Simulationszeit haben, was wir erreichen, wenn

Ng/Nr, Yg, Z und Ug

minimale Werte haben. Die Werte von t' und p' stehen allerdings über die dynamischen Gleichungen in Zusammenhang mit r und s und können keine willkürlich festgelegten Werte haben. Unter günstigsten Umständen beträgt der Zeitverbrauch ZV(VG) der Verarbeitung des Verknüpfungsblocks immerhin mehr als 55% des gesamten Zeitverbrauchs der Modellierung mit Taktgeber, was I wesentlich in Bezug auf Ir vermindert, wie die Tabelle 5.7 zeigt.

Nr	Ir - 100 %	ZV(VG)/ZVIm(RT)	I - 100 %
10	24,3 %	55,5 %	10,8 %
20	32,6 %	57,2 %	14,0 %
40	37,2 %	58,1 %	15,6 %

Tabelle 5.7 - Ir und I für r=0,8 bei minimalem Einfluß der Verknüpfungsglieder

Dies ist nicht der Fall, wenn wir den Gewinn der Modellierung ohne Taktgeber gegenüber der Modellierung mit Taktgeber betrachten, z.B. für r=0,2 (siehe Tabelle 5.8). Der kleine Wert von r ist für die Modellierung mit Taktgeber schlecht. Ihr Nachteil liegt hauptsächlich

- in der Verarbeitung der Taktgeber-bezogenen Aktivitäten (Faktor 538*F in ZVIm(RT)!), die unabhängig von r und Nr sind und für kleine Werte von r, s, t' und p' eine große relative Bedeutung haben, und

- in dem Aufruf aller Registerinstanzen, ein nur von Nr abhängiger Faktor.

In diesem Fall weist die Modellierung ohne Taktgeber einen großen Gewinn auf. Bezüglich der Verarbeitung des Registerblocks (Taktgeber eingeschlossen) verbraucht die Modellierung mit Taktgeber bis 153% mehr Simulationszeit als die Modellierung ohne Taktgeber. Dieser Gewinn wird durch die Verarbeitung des Verknüpfungsblocks nicht wesentlich vermindert, weil die Parameter t' und p' kleine Werte haben. Dieser Verarbeitung entsprechen höchstens 44% des gesamten Zeitverbrauchs der Modellierung mit Taktgeber. Insgesamt verbraucht diese Modellierung mindestens 64% mehr Simulationszeit als die Modellierung ohne Taktgeber, wie Tabelle 5.8 zeigt.

Nr	1/Ir - 100 %	ZV(VG)/ZVIm(RT)	1/I - 100 %
10	153,3 %	38,5 %	77,6 %
20	136,6 %	42,0 %	68,5 %
40	128,0 %	44,0 %	63,9 %

Tabelle 5.8 - Ir und I für r=0,2 bei minimalem Einfluß der Verknüpfungsglieder

Abb. 5.14 faßt diese Ergebnisse graphisch zusammen. Die Punkte der Kurven zeigen, welche Werte von r und Nr notwendig sind, damit der jeder Kurve entsprechende Faktor I erreicht wird, wenn die Verarbeitung des Verknüpfungsblocks minimalen Einfluß auf den Zeitverbrauch hat. Wir haben den festgelegten Wertebereich von r*Nr in der Abbildung hervorgehoben. Nur wenn r einen Wert über 0,52 bis 0,60 hat (je nachdem welchen Wert Nr hat), ist die Modellierung mit Taktgeber besser, und die Modellierung ohne Taktgeber verbraucht höchstens 15% mehr Simulationszeit. Dagegen kann die Modellierung ohne Taktgeber bis auf nur 60% der Simulationszeit der Modellierung mit Taktgeber verbrauchen, wenn r den Wert 0,2 hat.

Die Untersuchungen für F=2 und F=3 sind wegen der streckenweisen Linearität von ZVIo(RT) (Bereiche mit q=1/32, 1/2 und 1) etwas komplizierter, aber wir bekommen ähnliche Kurven, jedoch mit anderen numerischen Ergebnissen. In der Abb. 5.14 sind die Kurven von I=1 für F=2 und F=3 ebenfalls gezeigt.

Als Schlußfolgerung dieser Untersuchung wählen wir die Modellierung ohne Taktgeber für den Vergleich mit den Simulatoren RT1 und RT2. Aus der Analyse der Ergebnisse geht hervor:

- Für nicht sehr große Werte von r sind die Taktgeberbezogenen Aktivitäten und der Aufruf aller Registerinstanzen von großem Nachteil für die Modellierung mit Taktgeber, und

- für große Werte von r muß die Mehrheit der Registerinstanzen sowieso aktiviert werden, so daß in diesem Fall die Operationen auf der Ereignisliste in der Modellierung ohne Taktgeber vom Nachteil sind. Da die Werte von t' und p' jedoch entsprechend groß sein müssen, macht die große relative Bedeutung der beiden Modellierungen gemeinsamen Verarbeitung des Verknüpfungsblocks diesen Nachteil unwichtig.

Wir müssen das Grundsystem S1 noch vervollständigen. Da wir uns für die Instanzennetzmodellierung ohne Taktgeber entschieden haben und für S1 r*Nr = 5,4 und s/r = 2/3 gelten, nehmen wir aus der Tabelle 5.5
q=1/32

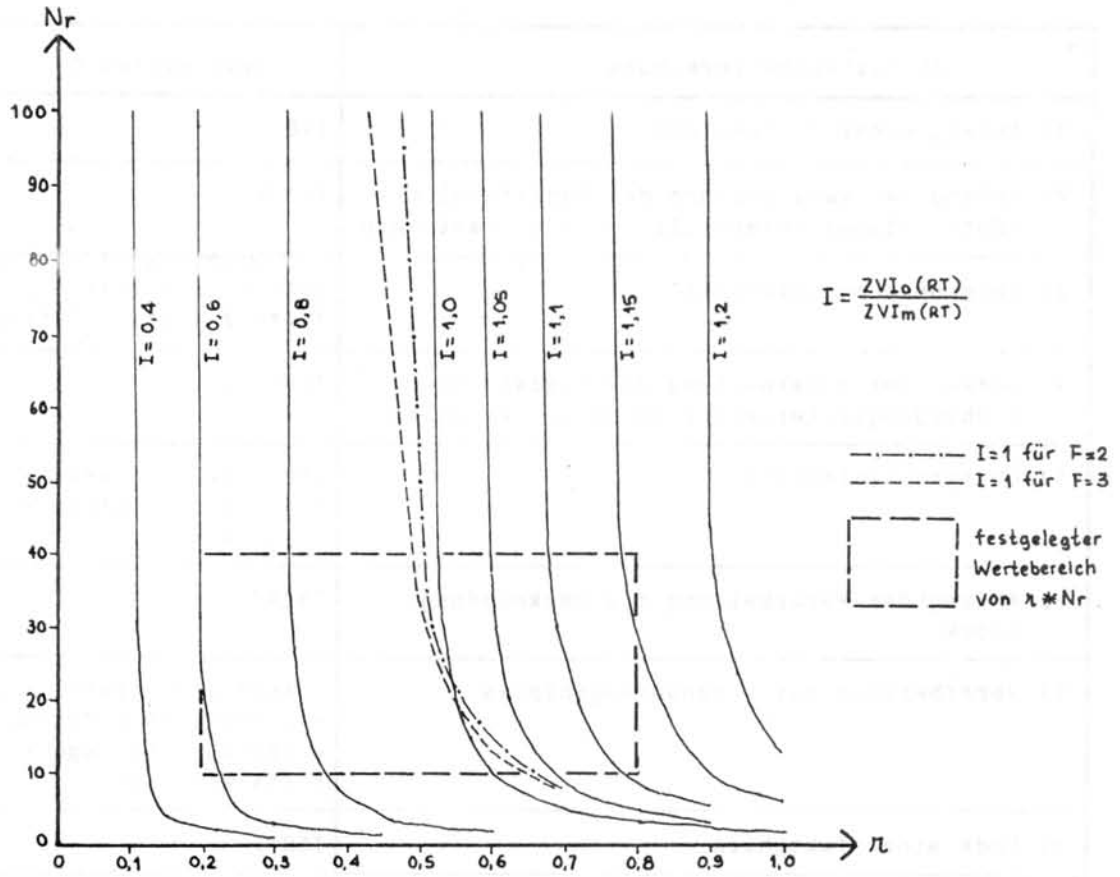


Abb. 5.14 - π * Nr für konstante Werte von I bei F = 1

und aus der Tabelle 5.4
Mem=0,316 und Mes=1,308.

5.6 Vergleich zwischen dem Simulator RT1 und der Instanzennetzmodellierung

5.6.1 Zeitverbrauch des Simulators RT1

Da es sich hier um die Ausführung eines Algorithmus handelt, den wir vollständig in Abb. 5.3 sehen und leicht interpretieren können, beschreiben wir die Verarbeitungssequenz des Simulators während eines Meßintervalls nicht so ausführlich, wie wir es für die Instanzennetzmodellierung gemacht haben. Diese Verarbeitungssequenz und der dazugehörige Zeitverbrauch sind in der Tabelle 5.9 gezeigt. Für ein Meßintervall sind die Ausführungszeiten T40-T60 (Abb. 5.3) und Ta (siehe Abschnitt 3.4) maßgeblich.

Aktivitätsbeschreibung	Zeitverbrauch
1) Anfang einer Taktperiode	T40
2) Anfang der Verarbeitung des Registerblocks (Entscheidungsintervall) in jeder Taktphase	T41*F
3) Entscheidungsintervall	Nr*(T42*r + T43*r1 + T44*(r2 + z1) + T45*s)
4) Anfang der Verarbeitung des Registerblocks (Übergangsintervall) in jeder Taktphase	T46*F
5) Übergangsintervall	s*Nr*(T47 + T48*Ur + + T49*Wr + T50*Wrg + + T51*Wrr) (1)
6) Anfang der Verarbeitung des Verknüpfungs-blocks	T52*F
7) Verarbeitung des Verknüpfungsblocks	t'*Ng*(T53 + Ta*Yg + Z) + p'*Ng*(T54 + T55*Ug + + T56*Wg + T57*Wgg + + T58*Wgr) (2)
8) Ende einer Taktphase	T59*F
9) Ende einer Taktperiode	T60

Hinweise: (1) Jedes der s*Nr Register hat Ur Fan-Out-Elemente; Davon haben Wr FLAG=FALSE, Wrg sind Verknüpfungsglieder und Wrr sind Register

(2) Jedes der p'*Ng Verknüpfungsglieder hat Ug Fan-Out-Elemente; Davon haben Wg FLAG=FALSE, Wgg sind Verknüpfungsglieder und Wgr sind Register

Tabelle 5.9 - Verarbeitungssequenz des Simulators RT1 während eines Meßintervalls

Wenn wir die Sektionausführungszeiten T gemäß den Ausführungszeiten der Algorithmenschritte auswerten, bekommen wir folgenden gesamten Zeitverbrauch für ein Meßintervall:

$$ZVS(RT1) = 7 + 38*F + Nr*(25*r + 8*r1 + 17*r2 + 17*z1) + + s*Nr*(50 + 16*Ur + 20*Wrg + 28*Wrr) + t'*Ng*(33 + 5*Yg + + Z) + p'*Ng*(13 + 16*Ug + 20*Wgg + 28*Wgr) \quad (5.28)$$

Unter Verwendung von r1=r-r2 und von (5.17)-(5.18) läßt sich dieser Ausdruck zu

$$ZVS(RT1) = 7 + 38*F + Nr*(61*r + 9*r2 + 17*z1) + s*Nr*(50 + 16*Ur) + t'*Ng*(53 + 5*Yg + Z) + p'*Ng*(13 + 16*Ug) \quad (5.29)$$

vereinfachen.

5.6.2 Sensibilität von R in Bezug auf die Parameter

Die Untersuchung basiert auf dem Verhältnis

$$R = \frac{ZVS(RT1)}{ZVIo(RT)} .$$

Die Grundlagen für die Rechtfertigung der folgenden Untersuchungen der Sensibilität von R bezüglich der Parameter wurden im Abschnitt 4.8.2 schon besprochen.

Aus (5.25) und (5.29) finden wir für das Grundsystem S1
ZVS(RT1) = 9069,5 ,
ZVIo(RT) = 13179,888 und
Rg(S1) = 68,81 %.

Untersuchung U1: Z
Ru(Z) = 100%, weil beide Modellierungen den Faktor Z*t'*Ng haben, so daß wir den minimalen Wert von R unter Verwendung des kleinsten Wertes von Z erreichen.

Untersuchung U2: Yg
Ru(Yg) = 100%, Folgerung wie für Z.

Untersuchung U3: v
Der Parameter v beeinflußt nur die Instanzennetzmodellierung, so daß wir das Minimum von R erreichen, wenn wir den größten Wert von v nehmen.

Untersuchung U4: Ur und Ug
Ru(Ur) = Ru(Ug) = 100%, weil beide Modellierungen den Faktor 16*Ur*s*Nr bzw. 16*Ug*p'*Ng haben. Die Folgerung ist wie für Z und Yg.

Untersuchung U5: Änderungsraten und Anzahl der Elemente
Wenn wir die Ausdrücke ZVS(RT1) und ZVIo(RT) betrachten, merken wir, daß alle Glieder, die eine **Änderungsrate** (r, s, t' und p') und/oder eine Anzahl von Elementen (Nr und Ng) besitzen, vom Typ "Rate * Anzahl" sind. Wir können deswegen eine Untersuchung durchführen, in der wir diese Produkte variieren, wobei unerheblich ist, ob wir die Rate, die Elementenanzahl oder beide ändern.

Gleichungen (5.17) und (5.18) besagen, daß eine gleichzeitige proportionale Änderung der Raten r, s, t' und p' möglich ist, ohne daß die übrigen Parameter beeinflußt werden.

In dieser Untersuchung übernehmen wir die Verhältnisse
  s = 2*r/3 , r2 = r/3 , z1 = r/2 , t'*Ng = 35*r*Nr/3 und
  p'*Ng = 140*r*Nr/15
aus dem System S1 und variieren alle Produkte "Änderungsrate * Anzahl" proportional zu r*Nr.

In der Tabelle 5.5 stellen wir fest, daß eine streckenweise lineare Funktion R(r*Nr) entsteht, die Wendepunkte in den Übergängen des Parameters q von 1/32 auf 1/2 und von 1/2 auf 1 hat. Die Parameter Mem und Mes (Anzahl der durchlaufenen Ereignisnotizen während der Eintragung einer neuen Notiz) sind Funktionen von q und r*Nr, die wir der Tabelle 5.4 entnehmen können.

Wir bekommen folgenden Ausdruck:

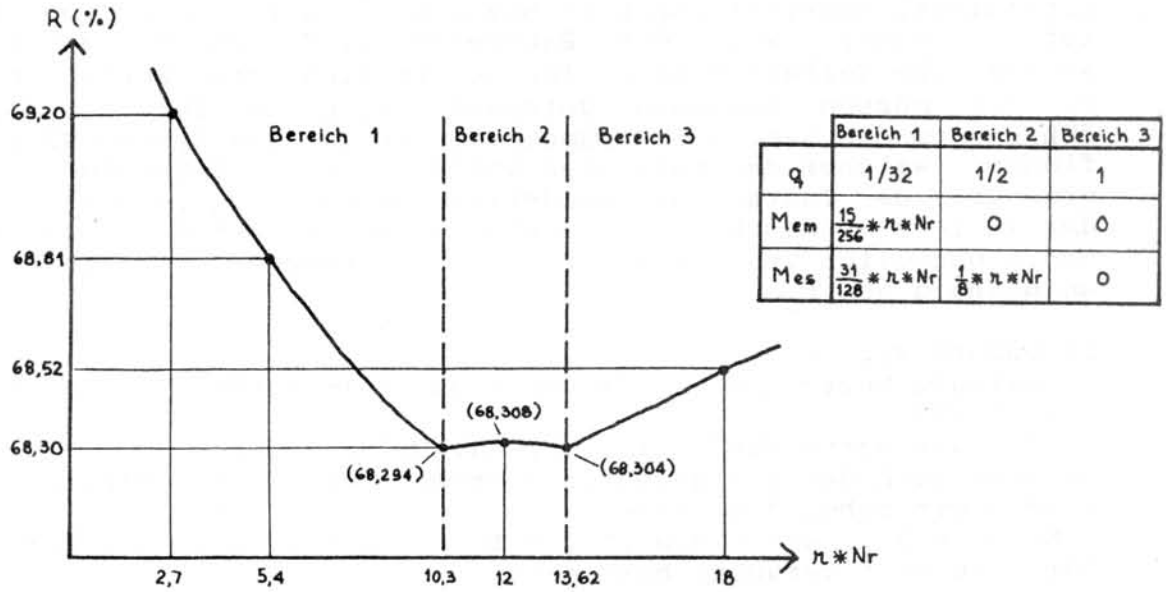
$$R(r*Nr)=\frac{83 + 1664{,}167*r*Nr}{80 + 464*q + r*Nr*(2405{,}4 + 15*Mem + 10*Mes)} \quad . \quad (5.30)$$

Nach diesem Ausdruck leiten wir die Kurve R x r*Nr der Abb. 5.15 ab. Wir sehen, daß ein Minimum mit dem ersten Wendepunkt zusammenfällt. In der Nähe der beiden Wendepunkte ist die Gewinnvariation allerdings sehr gering, weil die von r*Nr abhängigen Glieder fast keinen Einfluß haben und der Wert von R immer um Ru(r*Nr) = 69,18 % bleibt. Diese Eigenschaft bestätigt sich auch für alle anderen untersuchten Systeme, obwohl wir die Minima jeweils in verschiedenen Punkten finden. Wichtige erzielte Erkenntnis ist, daß der Gewinn praktisch unabhängig von r*Nr ist.

Untersuchung U6: Verhältnis zwischen dem Zeitverbrauch der Verarbeitung des Register- und des Verknüpfungsblocks

Es ist klar, daß der Hauptvorteil des Modells RT1 gegenüber der Instanzennetzmodellierung in der effizienteren Verarbeitung des Registerblocks liegt. Die Verarbeitung des Verknüpfungsblocks ist für beide Simulatoren fast identisch, was sich in
  Ru(p') = 100% und
  Ru(t') = 83%
widerspiegelt. Dieser letzte Wert erklärt sich nur durch die kompliziertere Hauptschleife der Verarbeitung der Rufliste in dem Instanzennetzsimulator. Das bedeutet, daß wir den minimalen Wert von R erreichen, indem wir die relative Bedeutung der Verarbeitung des Registerblocks für den Zeitverbrauch erhöhen. Dies erzielen wir durch
  größere Werte von s/r, s und r im Verhältnis zu p' und t' und
  kleinere Werte von a=p'/t' und Ng/Nr.

Insbesondere das Verhältnis Ng/Nr hat einen großen Einfluß auf R, wie die folgenden Zahlen veranschaulichen. Für eine verbleibende Summe Nr + Ng = 198 finden wir:



	Bereich 1	Bereich 2	Bereich 3
q	1/32	1/2	1
Mem	$\frac{15}{256} * r * Nr$	0	0
Mes	$\frac{31}{128} * r * Nr$	$\frac{1}{8} * r * Nr$	0

Abb. 5.15 - R x (r*Nr) für den Vergleich zwischen der RT1- und der Instanzennetz-modellierung (Arbeitspunkt = System S1)

Ng/Nr	=>	R
17		81,3 %
10		68,8 %
5		56,0 %

Untersuchung U7: F

In den Tabellen 5.4 und 5.5 erkennen wir, daß q, Mem und Mes abhängig sind von F, so daß wir R(F) als

R(F) = Funktion (F, q, Mem, Mes)

schreiben müssen. Für einen gleichbleibenden Wert r*Nr = 5,4 (damit gilt q=1/32 für F=1,2,3) finden wir

F	=>	R
1		69,28 %
2		68,81 %
3		68,69 %

Für andere Werte von r*Nr kann der erforderliche Wert von q jedoch unterschiedlich sein für unterschiedliche Werte von F, so daß wir eine allgemeingültige Folgerung über die Tendenz von R bezüglich von Variationen von F nicht feststellen können. Wir stellen trotzdem fest, daß falls die Anzahl der Elemente im System nicht zu klein ist, R praktisch unabhängig von F ist und sein Wert um den Grenzwert Ro(F) = 69,24 % liegt.

5.6.3 Maximaler Gewinn für den Simulator RT1

Eine Nichtlinearität von R in Bezug auf Variationen von F ist vorhanden, weil die Parameter q, Mem und Mes unterschiedliche Verhalten haben für unterschiedliche Werte von F. Wir müssen deswegen getrennte Versuche für F=1,2,3 führen, um den Satz von Parameterwerten für das System S2 zu finden, welches den maximalen Vorteil für den Simulator RT1 gegenüber der Instanzennetzmodellierung bringt. Die Mehrheit der im letzten Abschnitt gemachten Aussagen über die Tendenz von R bezüglich von Variationen der Parameter ist jedoch unabhängig von F.

So müssen wir
maximale Werte für v, s/r und s und r im Verhältnis zu t' und p' und
minimale Werte für Z, Yg, Ur, Ug, Ng/Nr und a
nehmen. Nach den festgelegten Wertebereichen der Parameter können wir schon die Werte
Ng/Nr = 5 , s/r = 0,8 , Z = 10 , Yg = 3 und a = 2/3
für alle drei Versuche bestimmen.

Versuch 1: F=1
Per Definition ist v=1 falls F=1, so daß wir in diesem Fall den angestrebten höchsten Wert für v nicht annehmen dürfen. Da für F=1 Mem und Mes immer gleich Null sind, ist keine Nichtlinearität von R in Bezug auf r*Nr vorhanden, und wir können den minimalen Wert q=1/32 festlegen. Wenn die Funktion R(Parameter P) linear ist, ist die Untersuchung durch den Vergleich von Ru(P) oder Ro(P) mit dem möglichen Wertebereich von R anwendbar (siehe Abschnitt 4.8.2). In diesem Fall finden wir Ro(r*Nr) = 95,24 % > R im ganzen erzielten Wertebereich von R, so daß eindeutig ist, daß ein größerer Wert von r*Nr R nach unten drückt. Wir nehmen deswegen die größten erlaubten Werte
r = 0,8 und Nr = 40 (r*Nr = 32),
woraus
s = 0,64 und Ng = 200
folgt.

Der Einfluß von e, r2 und z1 ist unbedeutend. Wir nehmen wieder e=0,5 und, um die Beziehungen zwischen r, s und e zu erfüllen (siehe Abb. 5.11), r2=0,32 und z1=0,4.

Wie im Abschnitt 5.4.5 dargestellt, sind niedrigere Werte von t' im Verhältnis zu r bei F=1 möglich, was einen kleineren Wert von R noch mehr begünstigt. Wir erniedrigen das Verhältnis t'/r von 7/6 auf 0,675, und haben damit
t' = 0,54 und p' = 0,36.

Wir können jetzt die Werte der W-Parameter finden, die die Gleichungen (5.17) und (5.18) lösen. Wir finden
Wrr = 0,3275 , Wrg = 1,125 , Wgg = 1,1 und Wgr = 0,328.

Die Bedingungen (5.15) müssen auch erfüllt werden, und wenn wir das Verhältnis (5.1) beibehalten, gibt uns (5.15) die kleinst möglichen Werte für Ur und Ug, in diesem Fall

Ur = Ug = 2,1.

Der Satz von Parameterwerten für S2 ist nun fertig, und mit ihm bekommen wir
Rg(S2) = 45,90 %.

Versuche 2 und 3

Ähnliche Vorgänge für die Suche nach S2 verwenden wir auch für F=2 und F=3. Es gibt nämlich nur folgende Unterschiede zum Versuch 1:
a) R(r*Nr) ist streckenweise linear, und wir müssen eine Untersuchung durchführen, wie sie in Abb. 5.15 dargestellt ist;
b) Die Ausdrücke (5.13) und (5.14) und die Erfüllung der Bedingungen (5.15) zeigen uns, daß es unmöglich ist, v zu erhöhen, was jetzt durch F ungleich eins möglich ist, und Ur und Ug gleichzeitig zu erniedrigen, so daß wir einen Kompromiß zwischen diesen Parametern finden müssen; und
c) t'/r muß bei F=2 höher und bei F=3 noch höher liegen als bei F=1.

Wir ersparen uns die ausführliche Beschreibung dieser Vorgänge, weil wir für F=2 Rg(S2') = 46,0 % und für F=3 Rg(S2'') = 47,4 % finden. Der endgültige Satz für S2 ist der, den wir mit F=1 festgelegt haben.

Wichtigste Schlußfolgerung dieser Untersuchung ist es, daß der Gewinn des Simulators RT1 gegenüber der Instanzennetzsimulation fast unabhängig von F und von proportionalen Variationen der Änderungsraten und der Anzahl der Elemente ist. Ein größerer Gewinn ist erst dann zu erzielen, wenn die relative Bedeutung der Verarbeitung des Registerblocks in der Simulation größer ist, d.h. wenn wir
Ng/Nr, Z und Yg erniedrigen und
s und r im Verhältnis zu t' und p' erhöhen.

5.6.4 Minimaler Gewinn für den Simulator RT1

Auch hier sind getrennte Untersuchungen für F=1,2,3 durchzuführen. Wir stellen den Vorgang exemplarisch für F=3 dar, weil am Ende dieser Fall das gewünschte System S3 liefert, das den maximalen Gewinn des Simulators RT1 gegenüber dem Instanzennetzsimulator zeigt. Allgemeingültige Aussage sind es, daß

minimale Werte für v, s/r, s und r im Verhältnis zu t' und p' und
maximale Werte für Z, Yg, Ur, Ug, Ng/Nr und a

für S3 festzulegen sind. Zunächst können wir schon die Werte
Ng/Nr = 20, s/r = 0,5 , Z = 40, Yg = 6, a = 5/6 und
Ur = Ug = 3
für alle drei Versuche bestimmen. Da wir nun, im Gegensatz zu S2, Ur und Ug erhöhen und v erniedrigen wollen, besteht kein Widerspruch mehr zu den Bedingungen (5.15).

Versuch 1: F=3

Für F=3 nehmen wir den Wert 1,2 als angemessene Untergrenze für v. Größere Werte von t'/r sind bei F=3 möglich, was einen größeren Wert von R glücklicherweise begünstigt. Wir nehmen für das Verhältnis t'/r den Wert 1,875. Unter Verwendung eines willkürlich gewählten Anfangswertes r=0,32, zusammenhängender Werte r2=0,08 und z1=0,16 (für e=0,5) und Beibehaltung der Verhältnisse s/r, t'/r und p'/t' finden wir

Rg(S3') = 84,88 % .

Um diesen Satz S3' variieren wir jetzt r*Nr, ohne die Proportion zwischen r*Nr und den anderen Produkten "Änderungsrate * Elementenanzahl" zu ändern. Daraus folgt die in der Abb. 5.16 gezeigte Kurve für R x r*Nr. Innerhalb des erlaubten Wertebereichs von r*Nr liegt das Maximum von R auf einen der beiden Endpunkten. Mit r*Nr=32 bekommen wir einen Wert R = 84,77 % und mit r*Nr=2 R = 84,93 % , so daß wir das Maximum unter Verwendung des kleinsten Wertes von r*Nr haben. Obwohl Systeme mit r*Nr=2 für F=1 noch denkbar wären, ist das für F=3 sicherlich schon unmöglich. Da der Wert von r*Nr sowieso kaum Einfluß auf den Wert von R hat, entscheiden wir uns für den Wert

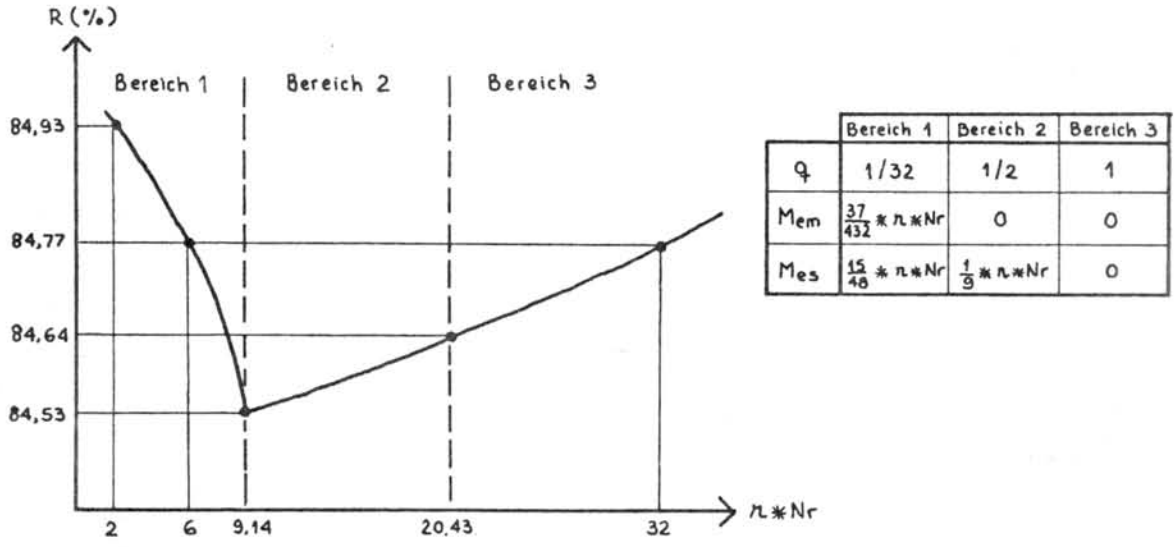
r*Nr = 6 (r = 0,4 und Nr = 15 willkürlich gewählt),

der

Rg(S3) = 84,77 %

liefert. Alle anderen Parameter sind folglich auch festgelegt:

s = 0,2 , t' = 0,75 , p' = 0,625 , Ng = 300 ,
r2 = 0,1 und z1 = 0,2 (für e = 0,5).



	Bereich 1	Bereich 2	Bereich 3
q	1/32	1/2	1
Mem	$\frac{37}{432} * r * Nr$	0	0
Mes	$\frac{15}{48} * r * Nr$	$\frac{1}{9} * r * Nr$	0

Abb. 5.16 - R x (r* Nr) für den Vergleich zwischen der RT1- und der Instanzennetz-modellierung (Arbeitspunkt = System S3')

Folgende W-Werte lösen (5.17) und (5.18) und erfüllen (5.15) unter Beachtung von (5.1):
Wrr = 0,03125 , Wrg = 1,25 , Wgr = 0,0315 und Wgg = 1,18.

Für r*Nr = 6 gelten
q = 1/32, Mem = 0,514 und Mes = 1,875.

Versuche 2 und 3
Die Versuche für F=1 und F=2 unterscheiden sich vom Versuch 1, indem:
a) t'/r bei F=2 niedriger und bei F=1 noch niedriger als bei F=3 liegen muß, was einen größeren Wert von R benachteiligt; und
b) v bei F=2 kleiner als 1,2 und bei F=1 gleich 1 sein kann, was einen größeren Wert von R begünstigt.

Ansonsten finden wir für beide Versuche den minimalen Wert von R, auch wenn wir den kleinsten Wert von r*Nr verwenden, wie im Versuch 1. Da wir für F=1 Rg(S3'') = 84,08 % und für F=2 Rg(S3''') = 84,49 % bekommen, wählen wir den Satz von Parameterwerten für F=3 aus als endgültigen Satz für S3.

Wir stellen wiederum fest, daß der Vorteil des Simulators RT1 fast unabhängig ist von F, r*Nr und entsprechenden proportional bleibenden Werten von s*Nr, t'*Ng und p'*Ng. Der kleinere Gewinn für das System S3 ist nur durch
größere Werte von Ng/Nr, Z und Yg
erklärbar, d.h., durch eine größere relative Bedeutung der Verarbeitung des Verknüpfungsblocks in der Simulation. In diesem Fall stellt die Verarbeitung des Registerblocks einen so kleinen Anteil des Zeitverbrauchs dar, daß sogar das Verhältnis t'/r fast unwichtig ist, wie die folgenden Zahlen für S3 zeigen:

t'/r =>	R
1,875	84,77 %
1,5	83,69 %
1,25	82,55 % .

5.7 Vergleich zwischen dem Simulator RT2 und der Instanzennetzmodellierung

5.7.1 Zeitverbrauch des Simulators RT2

Da auch hier der Algorithmus im Ganzen (Abb. 5.5) zu sehen ist, ist die Interpretation der Verarbeitungsssequenz des Simulators wesentlich leichter, und wir beschränken uns in der Tabelle 5.10 auf eine grobe Beschreibung dieser Sequenz. Einem Meßintervall entsprechen die Ausführungszeiten T70-T84. Der Zeitverbrauch der einer Taktperiode zugeordneten Aktivitäten ist ebenfalls in der Tabelle 5.10 angegeben.

Wichtig für das Verständnis dieser Teilausführungszeiten ist die Kenntnis über die Verarbeitungssequenz für die Ruflisten (d.h. CALLBIT-Arrays), die die Abb. 5.4 veranschaulicht

Aktivitätsbeschreibung	Zeitverbrauch
1) Anfang einer Taktperiode	T70
2) Anfang der Verarbeitung des Registerblocks in jeder Taktphase	T71*F
3) Verarbeitung des Registerblocks	Nr*(T72 + T73*r + T74*r1 + + T75*(r2 + z1) + T76*s + T77*Ur*s)
4) Anfang der Verarbeitung des Verknüpfungsblocks in jeder Taktphase	T78*F
5) Verarbeitung des Verknüpfungsblocks	Ng*(T79*F + T80*t + Ta*Yg*t + + Z*t + T81*p + T82*Ug*p)
6) Ende einer Taktphase	T83*F
7) Ende einer Taktperiode	T84

Tabelle 5.10 - Verarbeitungssequenz des Simulators RT2 während eines Meßintervalls

hat. Während jedes Register-CALLBIT in jeder Taktperiode nur einmal abgeprüft wird (deswegen T72*Nr), wird die Schaltnetz-Sektion des Arrays einmal in jeder Taktphase abgesucht (deswegen T79*F*Ng).

Nach Einsetzung der Werte für die Zeiten T unter Berücksichtigung der Ausführungszeiten der Algorithmenschritte bekommen wir folgenden gesamten Zeitverbrauch für ein Meßintervall, wobei wir in dem Ausdruck die Vereinfachung r1=r-r2 schon verwendet haben:

$$ZVS(RT2) = 7 + 20*F + Nr*(12 + 17*r + 9*r2 + 17*z1) + + s*Nr*(23 + 15*Ur) + 12*F*Ng + t*Ng*(23 + 5*Yg + Z) + + p*Ng*(13 + 15*Ug) \quad (5.31)$$

5.7.2 Sensibilität von R in Bezug auf die Parameter

Wir untersuchen hier die Sensibilität von

$$R = \frac{ZVS(RT2)}{ZVIo(RT)}$$

in Bezug auf Variationen der Systemparameter.

Untersuchung U1: F

Wie in der Untersuchung U7 bei dem Vergleich zwischen dem Simulator RT1 und der Instanzennetzmodellierung müssen wir R(F) als eine Funktion von F, q, Mem und Mes angeben. Für gleichbleibenden Wert r*Nr = 5,4, wie im System S1, bekommen wir

F	=>	R
1		55,61 %
2		71,54 %
3		87,56 % .

Die Ursache dieses Verhaltens ist klar: Durch das Abprüfen aller Verknüpfungsglieder-CALLBITs in jeder Taktphase ist ZVS(RT2) sehr empfindlich bezüglich des Wertes von F*Ng, insbesondere wenn die Änderungsraten t und p niedrig sind. Deswegen sind Systeme mit F > 1 untypisch für das Modell RT2. Im weiteren Verlauf dieser Untersuchung beschränken wir uns auf Systeme mit F=1.

Untersuchung U2: k

Nach dem Modell RT2 wird die Funktion eines Verknüpfungsgliedes, welches im realen System in einer Taktphase eine oder mehrere gleichzeitige Eingangsänderungen hat, nur einmal in dieser Phase ausgewertet. Der Faktor k in dem Zeitverbrauchsausdruck für die Instanzennetzmodellierung bringt die Tatsache zum Ausdruck, daß in dieser Modellierung mehrere Auswertungen möglich sind. Deswegen ist ein größerer Wert von k immer vorteilhafter für RT2 im Vergleich zur Instanzennetzsimulation. Dieser Vorteil ist desto größer, je größer die relative Bedeutung der Verarbeitung des Verknüpfungsblocks in der Simulation ist.

Untersuchung U3: v

Der Parameter v beeinflußt nur die Instanzennetzmodellierung, weil er die Registerinstanzen widerspiegelt, die mehrmals in einer Taktperiode aufgrund von Umgebungsänderungen sofort in derselben Taktphase aufgerufen werden, in der sie von der Änderung betroffen werden. Wir würden den minimalen Wert von R unter Verwendung des größten Wertes von v erreichen. Da wir aber nur F=1 berücksichtigen, muß v den Wert 1 haben.

Aufgrund der Existenz von Faktoren in dem Zeitverbrauchsausdruck für den Simulator RT2, die abhängig von Nr und Ng aber unabhängig von den Änderungsraten sind, ist hier eine einheitliche Analyse der Produkte "Rate * Elementenanzahl" unmöglich. Daher müssen wir getrennte Untersuchungen für Nr+Ng und für die Änderungsraten durchführen.

Untersuchung U4: Nr + Ng

$$Ro(Nr+Ng) = \frac{7 + 20*F}{F*(40 + 232*q)} \qquad (5.32)$$

Da wir uns nur mit F=1 befassen und für diesen Fall q immer gleich 1/32 ist, führt (5.32) zu einer Konstanten Ro(Nr+Ng) = 57,14 % . Das ist der Grenzgewinn, wenn die Anzahl der Elemente auf ein Minimum reduziert ist und der Gewinn nur durch die effizientere Zeitfortschaltung von RT2 entsteht. Da wir erwarten können, daß der minimale Wert von R unter den in U1 für F=1 erreichten 55% liegt, müssen wir Nr+Ng erhöhen, um R zu erniedrigen.

Untersuchung U5: Ur

Ru(Ur)=15/16, d.h., beide Simulatoren führen fast dieselben gemeinsamen Anweisungen bezüglich des Parameters Ur aus, und, je größer Ur ist, desto größer ist auch R.

Untersuchung U6: Änderungsraten

$$Ro(\text{Änderungsraten}) = \frac{7 + 20*F + 12*Nr + 12*F*Ng}{F*(40 + 232*q)} \qquad (5.33)$$

Da wir nur F=1 berücksichtigen, wird dieser Ausdruck zu

$$Ro(\text{Änderungsraten}) = \frac{27 + 12*(Nr + Ng)}{47,25} . \qquad (5.34)$$

Für den minimalen Wert Nr+Ng = 60 haben wir immerhin Ro(Änderungsraten) = 1581 % ! Die Erklärung ist einfach: Der größte Nachteil des Simulators RT2 liegt in dem Abprüfen aller CALLBITs. Je kleiner die Änderungsraten sind, desto auffallender ist dieser Nachteil. Um R zu erniedrigen, müssen wir deshalb immer große Änderungsraten nehmen.

Bevor wir Aussagen über die anderen Parameterwerte machen können, die zu einen minimalen Wert von R beitragen, müssen wir einen Konflikt lösen. Der Simulator RT2 verarbeitet sowohl den Register- als auch den Verknüpfungsblock effizienter als der Instanzennetzsimulator. Im Gegensatz zum Vergleich zwischen dem Simulator RT1 und der Instanzennetzsimulation ist der Zeitverbrauchsgewinn nun auch in der Verarbeitung des Verknüpfungsblocks aufgrund des Faktors k beträchtlich. Für die Suche nach dem System S4, welches den größten Vorteil für RT2 bringt, müssen wir zunächst feststellen, wo RT2 am meisten Gewinn gegenüber der Instanzennetzmodellierung bringt. Dementsprechend müssen wir die relative Bedeutung der Verarbeitung des Register- bzw. des Verknüpfungsblocks in der Simulation erhöhen.

Dazu berechnen wir unabhängig voneinander zwei Teilverhältnisse Rr für die Verarbeitung des Registerblocks und Rs für die Verarbeitung des Verknüpfungsblocks, indem wir nur die von Nr bzw. Ng abhängigen Glieder der Zeitverbrauchsausdrücke berücksichtigen. Unsere Entscheidung basiert dann auf einem Vergleich zwischen den minimalen Werten von Rr und Rs, die wir unter Variierung der zutreffenden Parameter erzielen.

Für Rr finden wir das Minimum bei höchsten Werten der Änderungsraten (r = 0,8), höchstem Verhältnis s/r (0,8) und kleinstem Wert von Ur (2,0). Die Parameter r2 und z1 haben geringen Einfluß auf den Zeitverbrauch und bekommen die Werte 0,32 bzw. 0,4, die die Bedingungen der Abb. 5.11 für e=0,5 erfüllen. Wir finden Rr(min) = 12,81 % .

Für Rs ist das Minimum bei höchsten Werten der Änderungsraten (t=0,6), höchstem Verhältnis a=p/t (0,9), größtem Wert von k (1,6 , und somit t'=0,96 und p'=0,864) und kleinsten Werten von Z (10), Yg (3) und Ug (2) zu finden. Daraus folgt Rs(min) = 48,50 % .

Das bedeutet, daß der größte Vorteil von RT2 gegenüber der Instanzennetzmodellierung in der Verarbeitung des Registerblocks liegt, wie auch für RT1 der Fall war. Diese Erkenntnis ermöglicht die Folgerungen der Untersuchungen U7 und U8.

Untersuchung U7: Verhältnis zwischen dem Zeitverbrauch der Verarbeitung des Register- und des Verknüpfungsblocks

Um die relative Bedeutung der Verarbeitung des Registerblocks in der Simulation zu erhöhen, müssen wir
hohe Werte von s/r, s und r im Verhältnis zu p und t, und
niedrige Werte von Ng/Nr und a=p/t
nehmen.

Untersuchung U8: Z, Yg und Ug
Um die relative Bedeutung der Verarbeitung des Verknüpfungsblocks zu erniedrigen, müssen wir
niedrige Werte für Z, Yg und Ug
nehmen, weil diese Parameter sich nur auf den Verknüpfungsblock beziehen. Diese Entscheidung können wir auch durch
Ru(Z) = Ru(Yg) = 1/k , Ru(Ug) = 15/(16*k)
erklären. Für den maximalen Wert von k (=1,6) haben wir
Ru(Z) = Ru(Yg) = 62,5 % und Ru(Ug) = 58,6 %,
und für F=1 liegt Rg(S4) sicherlich unter diesen Werten. Schon für die Parameterwerte von S1, aber mit F=1, haben wir 55,6 % bekommen (siehe Untersuchung U1).

5.7.3 Maximaler Gewinn für den Simulator RT2

In den vorherigen Untersuchungen haben wir alle Informationen bereitgestellt, die für die Festlegung des Satzes von Parameterwerten für das System S4 notwendig waren, welches maximalen Gewinn für den Simulator RT2 gegenüber der Instanzennetzsimulation aufweist. Wir müssen

F=1 (und damit v=1, q=1/32, Mem=Mes=0),
kleinste Werte für Ng/Nr, a=p/t, Z, Yg, Ur, Ug und t'/r, und
größte Werte für Nr+Ng, s/r und die Änderungsraten insgesamt nehmen.

Wir legen deswegen
  Ng/Nr = 5, Nr = 40 (folglich Ng = 200),
  s/r = 0,8 , t'/r = 0,675 , p/t = 2/3 , k = 1,6 ,
  r = 0,8 (folglich s = 0,64 , t' = 0,54 , p' = 0,36 ,
    t = 0,3375 und p = 0,225),
  Z = 10 und Yg = 3
fest.

Um schon bekannte Bedingungen zu erfüllen, legen wir weiter
  r2=0,32 , z1=0,4 (für e=0,5),
  Wrr=0,3275 , Wrg=1,125 , Wgr=0,328 und Wgg=1,1
fest. Nach diesen W-Werten leiten wir die minimalen Werte
  Ur = Ug = 2,1
ab, die die Bedingungen (5.15) unter Beibehaltung von (5.1) erfüllen.

Der Satz von Parameterwerten für S4 ist somit komplett und liefert
  Rg(S4) = 29,49 % .

Dieser Wert bestätigt unsere Behauptung in der Untersuchung U4 über die Tendenz von R in Bezug auf eine Variation von Nr+Ng. Es ist bemerkenswert, daß dieser Satz identisch ist mit dem von S2, mit Ausnahme von k, p und t, die dort nicht definierbar sind. Obwohl RT2 andere dynamische Eigenschaften als RT1 hat, die sogar für RT2 einen Zeitverbrauchsgewinn im System S4 (=S2) von 35 % gegenüber RT1 ermöglichen, sind sowohl für RT2 als auch für RT1 dieselben Bedingungen notwendig für den maximalen Gewinn gegenüber der Instanzennetzmodellierung. Insbesondere wichtig ist die möglichst große relative Bedeutung der Verarbeitung des Registerblocks in der Simulation.

5.7.4 Minimaler Gewinn für den Simulator RT2

Wir suchen den Satz von Parameterwerten für das System S5, das den minimalen Vorteil für den Simulator RT2 gegenüber der Instanzennetzsimulation bringt.

Ausgenommen die Tatsache, daß wir uns auf Systeme mit F=1 beschränken, weil sie die typischen Anwendungen für RT2 darstellen, müssen wir für die Festlegung dieses Satzes bis auf eine einzige Ausnahme genau die umgekehrten Entscheidungen in Bezug auf die Festlegung von S4 treffen.

Die Ausnahme bildet nämlich die Tendenz von R bezüglich einer Variation von Nr+Ng. Es ist zu erwarten, daß Rg(S5) höher als die in U1 erreichten 55% liegen wird. Damit gilt Ro(Nr+Ng) < Rg(S5), und wir müssen Nr+Ng erhöhen, um einen höheren Wert von R zu erreichen. Es gibt keinen Widerspruch in der Tatsache, daß wir Nr+Ng sowohl für die Festlegung von S4 als auch von S5 erhöhen müssen. Im ersten Fall bewirkt die Erhöhung von Nr+Ng eine Verminderung der Bedeutung der Zeitfortschaltung in der Simulation. Die Zeitfortschaltung bringt für den Simulator RT2 einen Gewinn gegenüber der Instanzennetzmodellierung, der kleiner ist als der aufgrund

der Verarbeitung des Register- und des Verknüpfungsblocks erzielte Gewinn. Im zweiten Fall bewirkt die Erhöhung von Nr+Ng eine größere Bedeutung für die Faktoren 12*Nr und 12*F*Ng, die bei sehr niedrigen Änderungsraten den Hauptnachteil von RT2 darstellen. Im ersten Fall bringen diese Faktoren keinen bedeutsamen Verlust, weil die Änderungsraten sehr groß sind.

Wir legen dann
  Ng/Nr = 20, Nr = 20 (folglich Ng = 400),
  s/r = 0,5 , t'/r = 1,8 , p/t = 5/6 , k = 1,2 ,
  r = 0,2 (folglich s = 0,1 , t'= 0,36 , p' = 0,3 , t = 0,3
    und p = 0,25),
  Z = 40, Yg = 6 und Ur = Ug = 3
fest.

Um die schon bekannten Bedingungen zu erfüllen, wählen wir noch
  r2=0,05 , z1=0,1 (für e=0,5),
  Wrr = 0,02 , Wrg = 1,2 , Wgr = 0,033 und Wgg = 1,18
aus.

Mit diesem Satz von Parameterwerten bekommen wir den Wert
  Rg(S5) = 74,08 % ,
der die Sensibilität von R bezüglich der Summe Nr+Ng bestätigt.

5.7.5 Sensibilität von R bezüglich der Ausführungszeiten der Algorithmenschritte

Wir haben auch für den Vergleich zwischen dem Simulator RT2 und dem Instanzennetzsimulator eine Untersuchung der Sensibilität von R in Bezug auf die Ausführungszeiten S der elementaren Aktivitäten der algorithmischen Ebene durchgeführt. Den Vorgang haben wir schon im Abschnitt 4.8.5 dargestellt, und da seine Einzelheiten keine besonderen Erkenntnisse bringen, stellen wir nur die erzielten Ergebnisse in der Tabelle 5.11 vor. Wir haben keine Untersuchung für das System S1 gemacht, weil es untypisch für das Modell RT2 ist.

Das Ergebnis ist nicht so schlecht für unsere Arbeit, wie es scheint. Wir müssen uns daran erinnern, daß die Zahlen lediglich Grenzwerte darstellen, die erst durch eine Variierung aller Ausführungszeiten S erreichbar ist, die mögliche Beziehungen zwischen den Ausführungszeiten nicht berücksichtigt. Außerdem haben wir die Ausführungszeiten um 50% geändert, die höchst unwahrscheinlich sind.

5.8 Ergebnisinterpretation

Um die wichtigsten Eigenschaften der untersuchten Simulationsalgorithmen zu verdeutlichen, vereinfachen wir die Zeitverbrauchsausdrücke, indem wir
- die wenig zeitverbrauchende Zeitfortschaltung weglassen

	Rg(S4) V	Rg(S5) V
Satz A1	29,5 %	74,1 %
Satz A2	20,4 % -44,7 %	60,4 % -22,6 %
Satz A3	39,4 % +25,1 %	86,5 % +14,4 %

$R = \frac{ZVS(RT2)}{ZVIo(RT)}$ V = Prozentuelle Veränderung von R gegenüber seinem Wert für den Satz A1

Tabelle 5.11 - Sensibilität von R bezüglich der Ausführungszeiten

und
- mittlere Werte r2=r/3, z1=r/2, Ur=Ug=2,5, Z=20 und Yg=4 als Konstanten verwenden.
Wir finden

$$ZVIo(RT) = r*Nr*(275,5 + 83*v) + 403*s*Nr + 112*t'*Ng + 53*p'*Ng \quad (5.35)$$

$$ZVS(RT1) = 72,5*r*Nr + 90*s*Nr + 93*t'*Ng + 53*p'*Ng \quad (5.36)$$

$$ZVS(RT2) = 12*Nr + 28,5*r*Nr + 60,5*s*Nr + 12*F*Ng + 63*t*Ng + 50,5*p*Ng \quad (5.37)$$

Diese Ausdrücke sind nun fast ausschließlich Funktionen der Änderungsraten und der Anzahl der Elemente. Wir erkennen ganz deutlich folgende Eigenschaften der Ausdrücke:

1) Die multiplikativen Konstanten für alle Produkte "Änderungsrate * Anzahl" nehmen in der Reihenfolge
Instanzennetzmodellierung -> RT1 -> RT2
ab;

2) Die größte Verminderung dieser Multiplikatoren in der obengenannten Reihenfolge erfolgt für die auf den Registerblock bezogenen Produkte r*Nr und s*Nr, und nicht für die auf den Verknüpfungsblock bezogenen Produkte t*Ng und p*Ng;

3) Das Modell RT2 hat einen zusätzlichen Vorteil gegenüber der Instanzennetzmodellierung und dem Modell RT1, nämlich den Faktor k = t'/t = p'/p, der die Abwesenheit einer eindeutigen Auswertungsreihenfolge für die Verknüpfungsglieder in dem Simulator RT1 und in dem Instanzennetzsimulator widerspiegelt. Sowieso liegt auch für RT2 der meiste Gewinn gegenüber der Instanzennetzmodellierung in der Verarbeitung des Registerblocks;

4) Das Modell RT2 hat dafür den Nachteil, daß sein Zeitverbrauch die Faktoren 12*Nr und 12*F*Ng (Abprüfen aller Register-CALLBITs in jeder Taktperiode und aller Verknüpfungsglieder-CALLBITs in jeder Taktphase) enthält, die bei kleinen Änderungsraten ausschlaggebend sind; und

5) Die Instanzennetzmodellierung hat den zusätzlichen Nachteil, daß ihr Zeitverbrauch vom Parameter v beeinflußt wird, der die mehrmalige Aufrufe einer Registerinstanz aufgrund von Umgebungsänderungen während einer Taktperiode widerspiegelt.

Obwohl das Modell RT2 anscheinend besser als das Modell RT1 ist, hat es zwei wesentlichen Nachteile:
1) Von der Modellierung her dürfen wir mit ihm nur Systeme ohne Schleifen innerhalb des Register- und des Verknüpfungsblocks simulieren, und
2) Von der Effizienz her ist es nur für Systeme mit Einphasentakt geeignet.

Die Reichweite der Gewinne der Simulatoren RT1 und RT2 gegenüber der Instanzennetzsimulation ist in der Tabelle 5.12 gezeigt. Diese Zahlen bestätigen die größere Effizienz von RT2. Sie besagen aber hauptsächlich, daß das Verhältnis Ng/Nr die Gewinnspanne entscheidend beeinflußt. Für ein digitales System gilt, daß, je abstrakter die Register-Transfer-Modellierung in Bezug auf die Gatterebene ist, d.h., je größer die Anzahl der Gatter in jedem Verknüpfungsglied ist, desto kleiner das Verhältnis Ng/Nr und der Effizienzgewinn der Simulatoren RT1 und RT2 sind. Dies passiert, weil die Verarbeitung des Verknüpfungsblocks, ausgenommen der Parameter k und die einfachere Verwaltung der Rufliste durch die Simulatoren RT1 und RT2, identisch ist für alle Modellierungen. Zur Erinnerung: Die Prozeduraufschreibung der Verknüpfungsgliederinstanzen gleicht die Aufschreibung der Funktionsauswertung der Verknüpfungsglieder in den Register-Transfer-Simulatoren. Effizienzverlust in der Instanzennetzmodellierung durch Sequenzermarkenverwaltung kommt nur in den Registerinstanzen vor. Ein wesentlicher Effizienzgewinn für RT1 und RT2 entsteht erst dann, wenn Ng/Nr genügend klein ist.

	Minimaler Wert von R (Maximaler Wert von 1/R)	Maximaler Wert von R (Minimaler Wert von 1/R)
RT1	45,9 % (2,18)	84,8 % (1,18)
RT2	29,5 % (3,39)	74,1 % (1,35)
Ng/Nr	5	20

Tabelle 5.12 - Gewinnspanne für den Vergleich der Simulatoren RT1 und RT2 (nur für F=1) mit der Instanzennetzmodellierung

Daß die in der Tabelle 5.12 aufgezeigten Zahlen wirklich Grenzfälle darstellen, sehen wir aus der Tatsache, daß sie erst durch eine Änderung der Parameterwerte erreicht wurden, die einige Beziehungen zwischen den Parametern nicht berücksichtigt haben. Den größten Gewinn zum Beispiel erzielen wir mit widersprüchlichen kleinst möglichen Werten für Ng/Nr, d.h. komplexeren Verknüpfungsgliedern, und Z und Yg, d.h. simpleren Verknüpfungsgliedern (siehe Abschnitt 5.4.5).

6. Gewinninterpretation

Wir haben uns zwei Ziele gesetzt. Das erste Ziel war die Messung des Effizienzgewinns der für die ausgewählten Systemklassen spezifischen Simulatoren gegenüber der Instanzennetzsimulation und die Suche zweier Grenzsysteme für jeden Vergleich, nämlich die, welche den größten bzw. den kleinsten Effizienzgewinn aufweisen. Dieses Ziel haben wir in den Kapiteln 4 und 5 erreicht.

Bei der Identifizierung der Grenzfälle für die Register-Transfer-Simulatoren haben wir schon grob dargestellt, was den Gewinn hauptsächlich verursacht, nämlich die bessere Verarbeitung des Registerblocks. Diese Feststellung wurde aufgrund der sehr großen Gewinnspanne möglich: Ein Vergleich zwischen den extremen Fällen verdeutlicht die günstigsten Bedingungen für die Register-Transfer-Simulatoren. Das ist nicht der Fall für die Gatterebenen-Simulation, weil die Gewinnspanne sehr klein ist und kein Parameter ausschlaggebenden Einfluß auf den Gewinn hat.

In diesem Kapitel versuchen wir nun, das zweite Ziel unserer Arbeit zu erreichen. Wir möchten ganz ausführlich feststellen, wodurch ein bestimmter Gewinn entsteht. Wir teilen dafür den Zeitverbrauch eines spezifischen Simulators und der entsprechenden Instanzennetzsimulation in **Faktoren** auf. Wir versuchen, in beiden zu vergleichenden Simulatoren gemeinsame oder ähnliche Simulationsaktivitäten während eines Meßintervalls zu erkennen und einem einzigen Faktor zuzuordnen. Als Beispiele für solche Simulationsaktivitäten, die sowohl im spezifischen als auch in dem Instanzennetzsimulator zu erkennen sind, können wir

- den Zugriff auf die UMG- bzw. FANOUTLISTE,
- die Eintragung von Namen in die Rufliste und
- die Zeitfortschaltung

erwähnen. Zwischen Faktoren in beiden zu vergleichenden Simulatoren muß eine der folgenden Beziehungen bestehen:

1) Ein Faktor ist beiden Modellierungen gemeinsam, d.h., beide Simulatoren müssen genau dieselbe Simulationsaktivität durchführen und verbrauchen dabei die gleiche Simulationszeit, so daß der spezifische Simulator bezüglich dieses Faktors keinen Gewinn gegenüber der Instanzennetzmodellierung aufweist;

2) Einer Simulationsaktivität entsprechen Faktoren in beiden Modellierungen, aber bei der Durchführung dieser Aktivität verbraucht der spezifische Simulator wenig Simulationszeit als der Instanzennetzsimulator; oder

3) Ein Faktor ist vorhanden, der nur in der Instanzennetzmodellierung erkennbar ist, d.h., die entsprechende Simulationsaktivität ist in dem spezifischen Simulator nicht notwendig.

Allgemein gesagt, ergibt sich ein Gewinn für den spezifischen gegenüber dem Instanzennetzsimulator, weil der spezifische Simulator für eine bestimmte Systemklasse zugeschnitten ist, während der Instanzennetzsimulator universelle Eigenschaften besitzt.

Wenn eine der beiden letzten obengenannten Beziehungen besteht, können wir sagen, daß die betrachtete Systemklasse in Bezug auf die allgemeinen Instanzennetze zusätzliche einschränkende Eigenschaften hat, die entweder eine effizientere Implementierung des zugehörigen spezifischen Simulators erlauben (zweite Beziehung) oder gewisse Aktivitäten der Instanzennetzsimulation gar unnötig machen (dritte Beziehung).

Wir möchten deshalb diesen Faktoren ganz bestimmte Eigenschaften der untersuchten Systemklassen zuordnen und darüber hinaus feststellen, wieviel Prozent des gesamten Zeitverbrauchs jedes Simulators und des gesamten Zeitgewinns des spezifischen Simulators jedem Faktor zuzuschreiben sind.

Die Faktoren bilden keine vollständige Partition des Zeitverbrauchs, vielmehr überschneiden sie sich manchmal. Diese Definition der Faktoren mit Überschneidungen ist notwendig, weil wir jeden Faktor einem einzigen eindeutigen Grund für den daraus resultierenden Zeitgewinn zuordnen möchten. Oft passiert es aber, daß erst die Existenz zweier Eigenschaften einer Systemklasse eine effizientere Implementierung des entsprechenden spezifischen Simulators ermöglichen. Der sich daraus ergebende Zeitgewinn muß deswegen in zwei Faktoren auftauchen.

Wir führen diese Untersuchung für drei Systemklassen durch: Gatterebenen-Systeme, Zeitmodell G1, und Register-Transferebenen-Systeme, Modelle RT1 und RT2.

6.1 Restriktionsgruppen

Die zusätzlichen Eigenschaften der Systemklassen können wir auch als Restriktionen in den allgemeinen Eigenschaften der Instanzennetze erklären.

Als wir in Kapitel 2 den Koordinator des Instanzennetzsimulators (und implizit seine Kommunikation mit den Instanzen und die den Instanzen zugeordneten Aufgaben) Schritt für Schritt entwickelt haben, haben wir immer jede Implementierungsentscheidung als Folge gewisser Eigenschaften gerechtfertigt. Diese allgemeinen Eigenschaften können wir in fünf **Gruppen** einteilen, die im folgenden beschrieben sind.

A) Fortschaltungsmodell: Dieser Gruppe ordnen wir die Eigenschaften zu, die uns erklären, wie der Koordinator entscheidet, welche Aktivität er zunächst durchzuführen hat. (Zum Beispiel welche Aufrufreihenfolge er für die Instanzen bestimmt, wann er die Modellzeit fortschaltet.)

B) Datenstruktur: Dieser Gruppe gehören die Eigenschaften, die die Organisation für die Speicherung der und Zugriff auf die Anschluß- und Zustandsvariablen bestimmen.

C) Topologie: In dieser Gruppe ordnen wir die Eigenschaften über die Systemtopologie ein, d.h., wie die Instanzen miteinander verbunden sein können.

D) Repertoire: Hierzu gehören die Annahmen über die vorhandenen Instanzentypen und ihre möglichen Auswirkungen.

E) Zeitverteilung der Ereignisse

Für die Gruppen C und D z.B. dürfen wir keine besondere Eigenschaft voraussetzen, falls wir allgemeine Instanzennetze betrachten. Da wir die Topologie nicht kennen, folgt eine nicht voraussehbare Anzahl der zu einem Modellzeitpunkt durch Umgebungsänderungen betroffenen Instanzen, und dementsprechend müssen wir die Rufliste dimensionieren. Da auch das Instanzenrepertoire unbekannt ist, muß der Koordinator annehmen, daß beliebige Auswirkungen einer Instanzenaktion (Umgebungsänderung, Zeitmeldung) vorkommen können.

Wir können alle zusätzlichen Eigenschaften der untersuchten Systemklassen auf diese fünf Restriktionsgruppen zurückführen. Jede konkrete Eigenschaft ergibt eine Untergruppe von Faktoren, die gesondert beschrieben wird, so daß wir den durch jede Eigenschaft entstandenen Gewinnanteil erkennen können.

6.2 Gatterebenen-Simulation, Zeitmodell G1

Im folgenden beschreiben wir die Eigenschaften von Gatterebenen-Systemen, falls das Zeitmodell G1 (siehe Abschnitt 4.4) verwendet wird, die den Aufbau eines optimalen Gattersimulators ermöglichen. Dazu zeigen wir, wie diese Eigenschaften sich auf Faktoren des Simulationszeitgewinnes des Gattersimulators gegenüber der entsprechenden Instanzennetzsimulation auswirken. Tabelle 6.1 faßt die Ergebnisse zusammen. Sie zeigt, für die in Kapitel 4 abgeleiteten Systeme S1 (Grundsystem), S2 (höchster Gewinn für den Gattersimulator) und S3 (kleinster Gewinn), wieviel Prozent des Simulationszeitverbrauchs des Gatter- und des Instanzennetzsimulators jedem Faktor entspricht, und wieviel Prozent des gesamten Zeitgewinns jedem Faktor zuzuschreiben ist.

Für jeden Faktor können wir einen analytischen Zeitverbrauchsausdruck ableiten, der eine Funktion der Systemparameter ist und alle Algorithmenschritte umfaßt, die sich auf diesen Faktor beziehen. Weder diese Ausdrücke noch der daraus resultierende absolute Zeitverbrauch werden jedoch angegeben, weil sie keine Erkenntnisse bringen.

Gruppe	Faktor	SYSTEM S1			SYSTEM S2			SYSTEM S3		
		% von ZVS (G1)	% von ZVI (G1)	% vom Gewinn (1)	% von ZVS (G1)	% von ZVI (G1)	% vom Gewinn (1)	% von ZVS (G1)	% von ZVI (G1)	% vom Gewinn (1)
A1	SM	0	10,9	19,0	0	11,4	18,8	0	8,9	17,0
	Psm	0	7,7	13,3	0	8,5	14,0	0	6,1	11,6
	Pek	0	2,7	4,7	0	3,3	5,4	0	3,0	5,6
	EK	0	1,8	3,1	0	2,0	3,2	0	1,9	3,6
	ZE	0	0	0	0	0,4	0,7	0	0	0
Summe A1		0	23,1	40,1	0	25,6	42,1	0	19,9	37,8
A2	AL	0	0	0	0	7,2	11,9	0	0	0
Summe A		0	23,1	40,1	0	32,8	54,0	0	19,9	37,8
D1	ZM	0	17,3	29,9	0	14,5	23,8	0	16,3	31,2
D2	Ruä	13,9	13,4	13,1	11,9	10,7	9,8	11,3	12,4	13,5
	ZR	0	2,1	3,6	0	1,8	3,0	0	1,8	3,5
Summe D2		13,9	15,5	16,7	11,9	12,5	12,8	11,3	14,2	17,0
Summe D		13,9	32,8	46,6	11,9	27,0	36,6	11,3	30,5	48,2
E1	S	21,4	13,7	8,1	27,6	16,3	9,0	23,7	16,9	10,7
	H	9,8	7,1	5,2	12,8	8,5	5,7	11,2	8,8	6,6
	Z	2,8	4,7	6,0	2,4	3,7	4,5	0,8	1,5	2,2
Summe E		34,0	25,5	19,3	42,8	28,5	19,2	35,7	27,2	19,5
B1	Pt	8,4	6,8	5,7	8,3	6,3	4,9	7,6	7,2	6,8
Fakt. ohne Gewinn oder mit Verlust (O+V)	F	13,1	5,5	0	9,7	3,8	0	16,3	7,8	0
	U	11,8	5,0	0	9,4	3,7	0	11,7	5,2	-0,8
	Euä	7,4	3,1	0	7,5	2,9	0	4,5	2,2	0
	T	4,5	1,9	0	3,9	1,5	0	3,7	1,8	0
	EA	3,6	1,1	-0,7	3,1	0,9	-0,5	3,9	1,4	-1,0
	GF	3,2	1,4	0	2,2	0,9	0	5,3	2,5	0
	L	0	0	0	1,1	0,1	-0,5	0	0	0
Summe O+V		43,6	18,0	-0,7	36,9	13,8	-1,0	45,4	20,9	-1,8
	P	8,4	18,6	26,1	8,3	13,9	28,2	7,6	10,3	26,6

(1) Siehe Bemerkung (3) in der Tabelle 6.4

Tabelle 6.1 - Faktoren im Gewinn des Gattersimulators gegenüber der Instanzennetzsimulation (nach dem Zeitmodell G1)

6.2.1 Einschränkungen in dem Fortschaltungsmodell

Zwei Eigenschaften von Gatterebenen-Systemen sind dieser Restriktionsgruppe zuzuordnen. Erstens haben wir angenommen,

daß Signaländerungen immer gleichzeitig erfolgen (auf die Modellzeitpunkte bezogen). Wir haben deswegen die Systemüberführung in dem Gattersimulator in zwei Phasen aufgeteilt, eine für die Aktualisierung der Gatterausgänge aufgrund von Ereignissen und andere für den Durchlauf dieser Änderungen durch das System. Es gibt eine deutliche Trennung zwischen zwei Situationen: Erstens werden die Gatter durch Ereignisse "aufgeweckt", und wir müssen ihre Ausgänge ändern. Erst dann werden sie von Eingangsänderungen "aufgeweckt", und wir werten ihre logischen Funktionen aus und planen ggf. neue Ereignisse bzw. machen Ereignisse rückgängig. In der Instanzennetzmodellierung erfolgt diese Trennung nicht automatisch durch den Simulator, so daß die Instanzen die Trennung durch Sequenzermarkenverwaltung nachbilden müssen. Die Nichtexistenz der automatischen Trennung in der Instanzennetzsimulation spiegelt sich auf dem Zeitgewinn des Gattersimulators durch 5 verschiedene Faktoren wider, die die Untergruppe A1 in der Tabelle 6.1 bilden und nur in der Instanzennetzmodellierung vorhanden sind (siehe Gatterinstanz in der Abb. 4.5 für ein besseres Verständnis der Faktoren).

Faktor SM - Sequenzermarkenverwaltung -
- Die Gatterinstanz muß testen, ob der Aufrufgrund eine Zeitmeldung (EVENT = TRUE? und EVENTTIME = TC?) oder eine Umgebungsänderung (Xi = ZXi?) ist, und die für diese Tests notwendigen Zustandsvariablen EVENT (bei der Eintragung und dem Löschen von Ereignisnotizen), EVENTTIME (bei der Eintragung von Notizen) und ZXi (bei Aufrufen aufgrund von Umgebungsänderungen) verwalten.

Faktor Psm - Parameterzugriff für die Sequenzermarkenverwaltung -
- Wir führen den Zugriff auf die Variablen EVENT, EVENTTIME und ZXi getrennt vom Faktor SM, damit die relative Bedeutung der Zugriffszeit ausgezeichnet wird.

Faktor EK - Auswirkung von Erkennungskonflikten -
- Die Nichtexistenz der vorher genannten Trennung bewirkt mögliche Verdopplungen von Aufrufen einer Gatterinstanz aufgrund von Umgebungsänderungen zu einem einzigen Modellzeitpunkt. Mögliche falsche Voraussagen für den Gatterausgang beim ersten Aufruf müssen wir beim zweiten Aufruf wiedergutmachen. Dafür ist die Verwaltung der Zustandsvariablen LASTACTION und LASTEFFECTIVETIME erforderlich. Dieser Faktor umfaßt selbstverständlich auch die Widergutmachung.

Faktor Pek - Parameterzugriff für den Faktor EK -
- Dieser Faktor umfaßt die Zugriffe auf die Zustandsvariablen LASTACTION und LASTEFFECTIVETIME und auf andere Variablen (EVENT und Y, siehe auch Abb. 4.7), die für die Behandlung der Auswirkungen von Erkennungskonflikten notwendig sind.

Faktor ZE - Mehrere zukünftige Ereignisse -
- Einige Instanzen können aufgerufen werden und zwar ein geplantes Ereignis haben (EVENT = TRUE), das aber für eine andere künftige Modellzeit geplant ist (EVENTTIME ≠ TC). Diesem Faktor ZE entspricht der im Punkt e) einer Instanzenprozedurausführung (siehe Tabelle 4.4) durch die Parameter d und (W-h)*f (im 1*N) gedeckten Zeitverbrauch.

Eine zweite Eigenschaft des Gattersimulators, die wir dieser Gruppe zuordnen müssen, ist seine Fähigkeit, Ereignisnotizen direkt auf der Ereignisliste zu löschen. Der Koordinator des Instanzennetzsimulators hat diese Fähigkeit nicht. Das Löschen selbst und die Erkennung, daß die Planung eines Ereignisses rückgängig gemacht worden ist, erfolgen durch Sequenzermarkenverwaltung in dem internen Instanzenzustand. Ein Faktor AL entsteht, der nur in der Instanzennetzmodellierung vorhanden ist.

Faktor AL - Ausführung der Instanzen, deren Aufrufe einer gelöschten Ereignisnotiz entsprechen -
- Der Koordinator bringt den Instanzennamen aus der Ereignis- in die Rufliste und aktiviert die Instanz. Sie testet, ob der Aufrufgrund eine Zeitmeldung oder eine Umgebungsänderung ist (dafür greift sie auf einige Anschluß- und Zustandsvariablen zu), aber findet keinen Aufrufgrund.

6.2.2 Einschränkungen in dem Instanzenrepertoire

Die Kenntnis über die Gatterinstanzen und ihre Auswirkungen ist konkret durch zwei Eigenschaften dargestellt. Erstens wissen wir, daß die Gatterinstanzen Ereignisse planen, um dann bei dem Aufruf aufgrund der Zeitmeldung immer dieselbe Aktivität durchzuführen, nämlich ihre Ausgänge zu ändern. Diese Aktivität übernimmt in dem Gattersimulator der Simulationskern selbst. In der Instanzennetzmodellierung gibt es dafür einen Faktor ZM, der keine Entsprechung im Gattersimulator findet.

Faktor ZM - Aufrufe der Gatterinstanzen aufgrund von Zeitmeldungen -
- Der Koordinator bringt die Instanzennamen aus der Ereignis- in die Rufliste und ruft die Instanzen auf, und jede Instanz testet, ob der Aufrufgrund eine Zeitmeldung oder eine Umgebungsänderung ist.

Die zweite Eigenschaft bezieht sich auf die Gatter, die von Eingangsänderungen betroffen werden. Zwei Gewinnfaktoren ZR und Ruä, die die Untergruppe D2 bilden, ergeben sich aus den Tatsachen, daß die Anzahl der logischen Funktionen begrenzt und ihre Typen bekannt sind, und daß auch die Auswirkungen von Auswertungen der Gatterfunktionen bekannt sind, nämlich mögliche Planungen von Ereignissen, aber nie unmittelbare Ausgangsänderungen.

Faktor ZR - Zyklische Organisation der Rufliste -
- Da die von Eingangsänderungen betroffenen Gatter nie eine unmittelbare Ausgangsänderung bewirken, ist die Anzahl der Gatter, deren Namen wir aufgrund von Umgebungsänderungen in die Rufliste bringen müssen, natürlicherweise auf die gesamte Anzahl der Gatter im System begrenzt. In der Instanzennetzmodellierung schaffen wir die Begrenzung der Größe der Rufliste erst durch die zyklische Organisation.

Faktor Ruä - Verarbeitung der Rufliste -
- Zusätzlich zu den Aktivitäten, die auch für den Gattersimulator notwendig sind (Eintragung und Rückgewinnung der Instanzennamen, zyklische Organisation unberücksichtigt), ist in der Instanzennetzmodellierung noch nötig, die Instanzen aufzurufen und die Variablen TF und CHE zu verwalten.

6.2.3 Einschränkungen in der Zeitverteilung der Ereignisse

In den Abschnitten 2.7.4 und 4.2 haben wir schon die Organisation der Ereignisliste für den Instanzennetz- bzw. den Gattersimulator diskutiert. Die optimale TM-Organisation für den Gattersimulator ist Folge einer besonderen Zeitverteilung der Ereignisse und spiegelt sich in drei Gewinnfaktoren wider, die die Gruppe E bilden: S (Eintragung von Ereignisnotizen), H (Holen der nächsten Notiz) und Z (Zeitfortschaltung).

6.2.4 Einschränkungen in der Datenstruktur

Jedem Gatter sind im Zeitmodell G1 eine Ausgangsvariable SIGNAL und zwei Zustandsvariablen EVENTP und DEL zugeordnet. Ein Gewinnfaktor Pt für den Gattersimulator ist Folge des Zugriffs auf diese Variablen.

Faktor Pt - Zugriff auf die Anschluß- und Zustandsvariablen-
- Der Gattersimulator hat 3 Arrays SIGNAL, EVENTP und DEL, die wir direkt (einstufiger Zugriff) durch den Gatternamen indizieren. In der Instanzennetzmodellierung verwenden wir eine allgemeine Datenstruktur, die einen dreistufigen Zugriff benötigt.

6.2.5 Faktoren ohne Gewinn

Der Vollständigkeit wegen müssen wir noch die übrigen Faktoren vorstellen, die in beiden Modellierungen vorhanden sind und keinen Gewinn für den Gattersimulator bzw. sogar einen kleinen Gewinn für die Instanzennetzmodellierung aufweisen.

Faktor F - Zugriff auf den Anfang der Teil-FANOUT- bzw. UMGLISTE -

Faktor U - Eintragung der Namen der Fan-Out-Elemente in die Rufliste (zyklische Organisation der Rufliste in der Instanzennetzmodellierung, die vom Faktor ZR gedeckt wird, unberücksichtigt) -
- Falls der Parameter h einen Wert ungleich Null hat, ist die Anzahl der in der Instanzennetzsimulation in die Rufliste aufgrund von Umgebungsänderungen einzutragenden Elementennamen sogar kleiner als im Gattersimulator.

Faktor Euä - Zugriff auf die Eingangsvariablen für die Auswertung der Gatterfunktion, wenn das Gatter von einer Eingangsänderung betroffen wird -

Faktor T - Zeitmodellbedingte Entscheidungen -
- Aufgrund der Variablen EVENTP und DEL (bzw. ihre Entsprechungen in der Gatterinstanz) und des neu berechneten Ausgangswertes muß die Entscheidung darüber fallen, ob die Eintragung oder das Löschen einer Ereignisnotiz notwendig ist.

Faktor EA - Ausführung einer Ereignisnotiz (unmittelbare Systemzustandsänderung aufgrund einer Ereignisnotiz) -

Faktor GF - Auswertung der logischen Funktion eines Gatters-

Faktor L - Löschen einer Ereignisnotiz -
- Diesem Faktor entspricht in der Instanzennetzmodellierung lediglich das Rücksetzen der Zustandsvariable EVENT, während der Gattersimulator einen wirklichen Zugriff auf das Feld CANCEL der zu löschenden Ereignisnotiz benötigt.

6.2.6 Schlußfolgerungen

Den Zahlen in der Tabelle 6.1 können wir wichtige Erkenntnisse über den Gewinn eines Gattersimulators gegenüber einer entsprechenden Instanzennetzsimulation entnehmen.

Zunächst möchten wir wissen, woran dieser Gewinn hauptsächlich liegt. Wir können drei Anteile nennen, die zusammen für 75,8 % (System S2) bis 81,5 % (System S1) des gesamten Zeitgewinns verantwortlich sind:
- Die Sequenzermarkenverwaltung samt den dafür notwendigen Parameterzugriffen ergibt 28,6 % bis 32,8 % des Gewinns;
- Die Aktivierung von Instanzen aufgrund von Zeitmeldungen erklärt 23,8 % bis 31,2 % des Gewinns; und
- Die Operationen auf der Ereignisliste ergeben 19,2 % bis 19,5 % des Gewinns.

Anders gesagt sind folgende Eigenschaften der Klasse der Gatterebenen-Systemen, die die Implementierung eines Gattersimulators ermöglichen, der eine wesentliche Verbesserung im Verhältnis zur Instanzennetzsimulation aufweist:
- Wir teilen die Systemüberführung in zwei Phasen auf und können jeder Phase verschiedene feste Instanzenaktivitäten zuordnen;
- Die Instanzenaktivitäten in der daraus resultierenden

ersten Phase sind trivial und bekannt und können vom "Koordinator" übernommen werden, ohne daß er die Instanzennamen aus der Ereignis- in die Rufliste bringen muß; und
- Eine bestimmte Zeitverteilung der Ereignisse wird angenommen.

Die zweitwichtigste Erkenntnis kommt aus dem Zeitverbrauch der Faktoren, die beiden Modellierungen gemeinsam sind (O+V in der Tabelle 6.1). Sie sind für 13,8 % bis 20,9 % des Zeitverbrauchs der Instanzennetzsimulation verantwortlich. Das bedeutet, daß die betrachtete Systemklasse bestimmte Eigenschaften hat, die genau mit Eigenschaften der Instanzennetze übereinstimmen. Die diesen Eigenschaften entsprechenden Faktoren verlangen deshalb von beiden Simulatoren genau dieselben Simulationsaktivitäten, die den obengenannten Anteil des Zeitverbrauchs der Instanzennetzsimulation erklären.

Dank der spezifischen Eigenschaften unserer Systemklasse lassen sich zudem zwischen 38 % (System S3) und 49,1 % (System S2) der in der Instanzennetzmodellierung notwendigen Aktivitäten im Gattersimulator völlig vermeiden. Der restliche Anteil der Aktivitäten hat zwar noch eine Entsprechung im Gattersimulator, aber eine Optimierung aufgrund zusätzlicher Eigenschaften ist möglich.

Diese beide Zahlen, nämlich der niedrigste Prozentsatz der gemeinsamen und der höchste Prozentsatz der vermeidbaren Aktivitäten, erklären, warum der Gattersimulator immer mindestens 40 % der für die Instanzennetzsimulation benötigten Simulationszeit verbraucht (siehe Abschnitt 4.10).

Wichtig ist auch zu erkennen, für welche Systeme der Gattersimulator den meisten Effizienzgewinn zeigt. Das System S2 unterscheidet sich von den Systemen S1 und S3 durch die Anwesenheit des Faktors AL, der für 11,9 % des Gewinns verantwortlich ist. Das bedeutet, daß wir den größten Gewinn des Gattersimulators für sehr schnelle Systeme haben, die das Löschen vieler Ereignisnotizen benötigen. Dies ist auf die Tatsache zurückzuführen, daß der Instanzennetzsimulator die ganze Verarbeitung der Rufliste samt Instanzenaufrufen auch für diejenige Instanzen braucht, deren Aufrufe gelöschten Ereignisnotizen entsprechen. Wir weisen jedoch nochmals darauf hin, daß die Gewinnspanne trotzdem nicht groß ist (siehe Abschnitt 4.10).

Letztlich möchten wir die relative Bedeutung der Parameterzugriffe und der dreistufigen Zugriffsstruktur in der Instanzennetzmodellierung von Gatterebenen-Systemen hervorheben. Aufgrund dieser Struktur und der Notwendigkeit vieler Parameter für die Sequenzermarkenverwaltung und für die Behandlung von Erkennungskonflikten erklären diese Zugriffe 26,1 % bis 28,2 % des gesamten Verlustes der Instanzennetzsimulation gegenüber dem Gattersimulator.

6.3 Simulation der Register-Transferebene, Modell RT1

Tabelle 6.2 zeigt die Gruppen von Restriktionen in den allgemeinen Instanzennetzeigenschaften, die die Register-Transfer-Systeme als Sonderfälle von Instanzennetzen erklären, wenn wir das Modell RT1 verwenden. Dazu sehen wir die Gewinnfaktoren, ihre Anteile im Zeitverbrauch des Simulators RT1 und des Instanzennetzsimulators und ihre Anteile im gesamten Effizienzgewinn des Simulators RT1 gegenüber der Instanzennetzsimulation. Im folgenden beschreiben wir diese Eigenschaften und ordnen ihnen die Faktoren zu.

6.3.1 Einschränkungen in dem Instanzenrepertoire

Folgende Kenntnis über die Instanzen eines Register-Transfer-Systems nach dem Modell RT1 ist im voraus vorhanden, wenn wir den Simulator RT1 entwickeln:

- Die Master- und die Verknüpfungsgliederinstanzen können eine Umgebungsänderung melden;
- Die Slave-Instanzen melden immer eine Umgebungsänderung; und
- Keine Instanz gibt eine Zeitmeldung ab.

Außerdem nehmen wir an, daß keine allgemeine Instanzenprozeduren dem Benutzer zur Verfügung stehen. Diese Eigenschaften ergeben die 4 folgenden Gewinnfaktoren für den Register-Transfer-Simulator, die die Gruppe D3 bilden.

Faktor Rg - Verarbeitung der Rufliste für die Verknüpfungsgliederinstanzen -
- Zusätzlich zu den Aktivitäten, die im Simulator RT1 auch vorhanden sind, sind in der Instanzennetzsimulation der Instanzenaufruf und die Verwaltung der Variablen TF erforderlich.

Faktor Rm - Verarbeitung der Rufliste für die Master-Instanzen -
- Zusätzlich sind in der Instanzennetzsimulation der Instanzenaufruf und die Verwaltung der Variablen TF und CHE notwendig.

Faktor Rs - Verarbeitung der Rufliste für die Slave-Instanzen -
- Wie Rm

Faktor As - Ausführung der Slave-Funktion -
- Zusätzlich zu der Zuweisung des neuen Wertes an den Registerausgang ist in der Instanzennetzsimulation das Setzen der Variable CHE nötig.

6.3.2 Einschränkungen in dem Fortschaltungsmodell

Drei Eigenschaften von Register-Transfer-Systemen erlauben Optimierungen des Simulators RT1 in Bezug auf das Fortschal-

Gruppe	Faktor	SYSTEM S1			SYSTEM S2			SYSTEM S3		
		% von ZVS(RT1)	% von ZVIo(RT)	% vom Gewinn (5)	% von ZVS(RT1)	% von ZVIo(RT)	% vom Gewinn (5)	% von ZVS(RT1)	% von ZVIo(RT)	% vom Gewinn (5)
D3	Rg	19,4	23,4	28,1	18,5	14,9	11,0	15,7	23,3	52,2
	Rm	1,2	2,0	3,3	3,9	4,4	4,5	0,3	0,6	1,9
	Rs	0,6	1,3	2,7	2,2	3,5	4,3	0,1	0,3	1,2
	As	0,12	0,14	0,15	0,5	0,4	0,2	0,02	0,03	0,07
Summe D		21,3	26,8	34,2	25,1	23,2	20,0	16,1	24,2	55,4
A3	Em	0	6,6	18,6	0	14,1	24,3	0	2,0	10,7
	Muä	0	4,1	11,5	0	9,0	15,5	0	1,3	6,6
	SMm	0	1,3	3,6	0	2,5	4,3	0	0,4	2,0
	Mv	0	0,8	2,3	0	0	0	0	0,17	0,9
	Z	0,9	0,7	0,2	0,3	0,13	0,01	0,6	0,6	0,4
Summe A3 (1)		0,9	12,7	33,8	0,3	24,2	41,3	0,6	4,2	19,3
A4	Es	0	4,8	13,5	0	11,3	19,4	0	1,1	5,7
	Suä	0	1,7	4,9	0	4,6	7,9	0	0,4	2,1
	SMs	0	0,16	0,5	0	0,4	0,7	0	0,04	0,2
Summe A4 (2)		0	6,7	18,7	0	16,1	27,7	0	1,5	7,9
A5	ZRg	0	2,9	8,0	0	1,8	3,1	0	2,9	14,9
Summe A (3)		0,9	22,7	61,7	0,3	43,2	73,9	0,6	8,7	42,8

Tabelle 6.2 - Faktoren im Gewinn des Simulators RT1 gegenüber der Instanzennetzsimulation
(Fortsetzung auf der nächsten Seite)

C1	FUm	0,3	1,1	2,5	1,3	3,0	4,1	0,06	0,3	1,1
	ZRm	0	0,4	1,2	0	0,8	1,4	0	0,13	0,7
Summe C1		0,3	1,5	3,7	1,3	3,8	5,5	0,06	0,4	1,8
C2	ZRs	0	0,2	0,7	0	0,6	1,1	0	0,06	0,3
Summe C		0,3	1,7	4,4	1,3	4,4	6,6	0,06	0,5	2,1
B1	P1	0,16	0,6	1,4	0,6	1,5	2,1	0,03	0,16	0,7
B2	P2	0,9	1,0	1,0	3,1	2,3	1,4	0,2	0,3	0,5
Summe B (4)		1,0	1,3	1,4	3,7	3,0	2,1	0,2	0,3	0,7
Faktoren	Ug	19,8	9,8	- 10,7	18,5	6,1	- 4,2	18,0	11,5	- 19,9
mit	Ur	2,3	1,0	- 1,4	6,5	1,6	- 2,5	0,8	0,5	- 0,9
Verlust	Am	0,9	0,5	- 0,4	3,0	1,1	- 0,5	0,2	0,13	- 0,3
Summe V		23,0	11,3	- 12,5	28,0	8,8	- 7,2	19,0	12,1	- 21,1
Faktoren	Fg	20,7	14,2	0	15,9	7,3	0	21,5	18,3	0
ohne	Pg	17,4	12,0	0	13,2	6,1	0	19,7	16,7	0
Gewinn	Ag	13,9	9,6	0	6,6	3,0	0	22,5	19,1	0
0	Fs	1,5	1,0	0	5,7	2,6	0	0,3	0,3	0
Summe 0		53,5	36,8	0	41,4	19,0	0	64,0	54,4	0
Summe 0 + V		76,5	48,1	- 12,5	69,4	27,8	- 7,2	83,0	66,5	- 21,1
	EL	0,9	12,1	32,3	0,3	25,5	43,7	0,6	3,7	16,8
	P	1,0	2,5	5,0	3,7	5,6	6,6	0,2	0,7	2,7

Bemerkungen:

(1) Überschneidung zwischen SMm und Muä, SMm und Mv
(2) Überschneidung zwischen SMs und Suä
(3) Überschneidung zwischen ZRm und Muä, ZRm und Mv
(4) Überschneidung zwischen P1 und P2
(5) Siehe Bemerkung (3) in Tabelle 6.4

Tabelle 6.2 - Faktoren im Gewinn des Simulators RT1 gegenüber der Instanzennetzsimulation (Fortsetzung)

tungsmodell. Die erste Eigenschaft ist die Aufteilung einer Taktperiode in Taktphasen und die Zuordnung eines Registers zu einer bestimmten Phase. Der Simulator RT1 nutzt diese Eigenschaft aus, indem er unterschiedliche Master-Ruflisten für jede Taktphase einer Taktperiode hat. Daraus folgen 5 Gewinnfaktoren, die die Untergruppe A3 bilden und Aktivitäten in der Instanzennetzsimulation entsprechen, die im Simulator RT1 keinen Vergleich finden (Ausnahme: Faktor Z).

Faktor Em - Eintragen und Holen von Ereignisnotizen für die Master-Instanzen -

Faktor Muä - Aufruf der Master-Instanzen aufgrund von Umgebungsänderungen noch in derselben Taktphase, in der die Eingangsänderungen stattgefunden haben -

Faktor SMm - Sequenzermarkenverwaltung der Master-Instanzen- - Diese Verwaltung ist vorhanden, damit die Instanzen den Aufrufgrund (Zeitmeldung oder Umgebungsänderung) und die mehrmaligen Aufrufe aufgrund von Umgebungsänderungen in einer einzigen Taktperiode erkennen können.

Faktor Mv - Aufruf der Master-Instanzen aufgrund von Umgebungsänderungen, für die Instanzen die mehrmals in einer Taktperiode in Aktion gebracht werden -
- (Siehe Parameter v.) Der erste Aufruf in der Taktperiode ist durch den Faktor Muä gedeckt.

Faktor Z - Zeitfortschaltung -
- In der Instanzennetzsimulation wird die Zeitfortschaltung durch eine Sequenz von Ereignisnotizen geschaffen, während der Simulator RT1 die Modellzeit implizit durch die Verarbeitungssequenz der Ruflisten vorrückt.

Die zweite Eigenschaft des Modells RT1, die das Fortschaltungsmodell betrifft, ist die Aufteilung der Simulation einer Taktphase in zwei Schritte, einer für das Entscheidungsintervall (Master-Funktionen) und ein anderer für das Übergangsintervall (Slave-Funktionen). Der Simulator RT1 schafft diese Aufteilung durch die Implementierung zweier getrennter Ruflisten, die nacheinander bearbeitet werden. In der Instanzennetzmodellierung ist die Aufteilung erst durch explizite getrennte Modellzeiten und dazugehörige Ereignisse und Modellzeitfortschaltung möglich. Drei Gewinnfaktoren, die die Untergruppe A4 bilden, ergeben sich aus entsprechenden, nur in der Instanzennetzsimulation vorhandenen Aktivitäten.

Faktor Es - Eintragen und Holen von Ereignisnotizen für die Slave-Instanzen -

Faktor Suä - Aufruf der Slave-Instanzen aufgrund von Umgebungsänderungen der Master-Instanzen -

Faktor SMs - Sequenzermarkenverwaltung der Slave-Instanzen -
- Die Instanzen müssen den Aufrufgrund (Zeitmeldung oder Umgebungsänderung) erkennen.

Die dritte auf das Fortschaltungsmodell bezogene Eigenschaft von Register-Transfer-Systemen gemäß dem Modell RT1 ist die Abwesenheit von Oszillationen in der Verknüpfungslogik. Diese Eigenschaft erklärt den Gewinnfaktor ZRg.

Faktor ZRg - Zyklische Organisation der Rufliste für die Verknüpfungsglieder -
- Während der Instanzennetzsimulator diese allgemeine Organisation verwenden muß, ist sie für den Simulator RT1 nicht nötig, weil die Anzahl der Verknüpfungsglieder, deren Funktionen auszuwerten sind, natürlicherweise begrenzt ist.

6.3.3 Einschränkungen in der Topologie

Zwei Tatsachen sind über die Topologie von nach dem Modell RT1 nachgebildeten Register-Transfer-Systemen bekannt. Erstens hat jede Master-Instanz ein einziges Fan-Out-Element, welches eine Slave-Instanz mit dem gleichen Namen ist. Zwei Gewinnfaktoren (Untergruppe C1) lassen sich durch diese Eigenschaft erklären.

Faktor FUm - FANOUT- bzw. UMGLISTE-Verarbeitung für die Master-Instanzen -
- Im Simulator RT1 tragen wir nach einer Ausgangsänderung eines Masters den Namen des betroffenen Slaves unmittelbar in die Slave-Rufliste ein. In der Instanzennetzsimulation müssen wir die allgemeine Struktur UMGVERWEIS + UMGLISTE verwenden, um diesen Namen zu finden.

Faktor ZRm - Zyklische Organisation der Rufliste für die Master-Instanzen -
- Der Simulator RT1 braucht diese Organisation nicht: Da wir keinen Master-Namen aufgrund einer Umgebungsänderung eines anderen Masters in die Rufliste eintragen müssen, ist die Größe einer Master-Rufliste natürlicherweise auf die Anzahl der dieser Taktphase entsprechenden Master begrenzt.

Zweitens wissen wir, daß keine Slave-Instanz wiederum eine Slave-Instanz als Fan-Out-Element hat. Diese Eigenschaft erklärt den Faktor ZRs, der nur in der Instanzennetzsimulation vorhanden ist.

Faktor ZRs - Zyklische Organisation der Rufliste für die Slave-Instanzen -

6.3.4 Sonstige Faktoren

Aufgrund der für Register-Transfer-Systeme spezifischen optimalen Datenstruktur erscheinen zwei Gewinnfaktoren, eine für die Variablen, auf die wir im Simulator RT1 durch eine einstufige Struktur zugreifen können, und eine andere für

die Variablen, die durch eine zweistufige Struktur zugreifbar sind. In der Instanzennetzsimulation verwenden wir eine dreistufige Datenstruktur für alle Anschluß- und Zustandsvariablen.

Faktor P1 - Zugriff auf die Anschlußvariable zwischen dem Master-Ausgang und dem Slave-Eingang -
- Das ist ein einstufiger Zugriff im Simulator RT1, weil er ein gesondertes Array für diese Variable hat, das durch die Registernamen indiziert wird. Dies ist möglich, weil nur der Master- und der Slave-Teil des zutreffenden Registers Zugang zu einer dieser Variablen brauchen, und beide denselben Namen haben.

Faktor P2 - Zugriff auf die Anschlußvariablen ENABLE und X (Dateneingang) der Master-Instanzen und Y (Datenausgang) der Slave-Instanzen -
- Auf diese Variablen greifen wir im Simulator RT1 durch eine zweistufige Struktur zu. Wir sehen für jeden dieser Anschlußtypen ein Sonderarray von Verweisen auf die Variablen vor, und dieses Array wird direkt durch den Registernamen indiziert.

Folgenden Faktoren entsprechen Aktivitäten, für die sowohl der Simulator RT1 als auch der Instanzennetzsimulator genau denselben Zeitverbrauch aufweisen.

Faktor Fg - Zugriff auf den Anfang der Teil-FANOUT- bzw. UMGLISTE eines Verknüpfungsglieds (Eintragung der Namen der Fan-Out-Elemente in die Rufliste unberücksichtigt) -

Faktor Pg - Zugriff auf die Anschlußvariablen der Verknüpfungsglieder -
- Der Simulator RT1 benutzt auch eine dreistufige Struktur (siehe Abschnitt 5.2.2).

Faktor Ag - Auswertung der Funktionen der Verknüpfungsglieder -
- Siehe Abschnitt 5.3.1 für die Erklärung, warum beide Modellierungen dieselbe Simulationszeit für diese Aktivität verbrauchen.

Faktor Fs - Zugriff auf den Anfang der Teil-FANOUT- bzw. UMGLISTE der Slaves (Eintragung der Namen der Fan-Out-Elemente in die Rufliste unberücksichtigt) -

Um die Aufteilung des Zeitverbrauchs zu vervollständigen, müssen wir noch drei Faktoren einführen, nämlich
Ug (Eintragung der Namen der Fan-Out-Elemente der Verknüpfungsglieder in die Rufliste),
Ur (Eintragung der Namen der Fan-Out-Elemente der Register bzw. der Slave-Instanzen in die Rufliste) und
Am (Ausführung der Master-Funktionen).
Diese drei Faktoren, falls isoliert betrachtet, weisen sogar einen Effizienzverlust des Simulators RT1 gegenüber der Instanzennetzsimulation auf. Dieser Verlust ist jedoch trügerisch, weil die diesen Faktoren entsprechenden Aktivi-

täten des Simulators RT1 in der Tat wesentliche Gewinne in anderen Faktoren ermöglichen.

Faktoren Ug und Ur -
- Bevor wir im Simulator RT1 den Namen eines Fan-Out-Elementes in die Rufliste eintragen können, müssen wir feststellen, von welchem Typ dieses Element ist, um die richtige Rufliste identifizieren zu können. Falls es sich um ein Register handelt, ist die Rufliste sogar eine Matrix, die eine kompliziertere Eintragung verlangt. Dafür aber ermöglichen die getrennten Ruflisten eine wesentlich einfachere Zeitfortschaltung als in der Instanzennetzmodellierung.

Faktor Am -
- In der Instanzennetzmodellierung können wir spezifische Instanzenprozeduren für diejenigen Register schreiben, die keinen ENABLE-Eingang haben. Da wir für den Simulator RT1 eine einheitliche Behandlung für alle Register haben wollen, die vom "Koordinator" übernommen wird, muß der Koordinator überprüfen, ob das gerade verarbeitete Register diesen Eingang hat. Dafür entfällt der Instanzenaufruf.

6.3.5 Schlußfolgerungen

Eine Analyse der Zahlen in der Tabelle 6.2 bringt uns die gewünschte Erkenntnis, woran der Gewinn des Simulators RT1 gegenüber dem Instanzennetzsimulator hauptsächlich liegt. Fünf Faktoren sind für 73,3 % (System S2) bis 91,0 % (System S3) des Gewinns verantwortlich, wobei ihre relativen Bedeutungen stark davon abhängen, wie sich die gesamte Anzahl der Elemente zwischen Registern und Verknüpfungsgliedern aufteilt. Tabelle 6.3 veranschaulicht diese Tatsache.

	S2	S3
Rg + ZRg	14,1 %	67,1 %
Em + Muã + Mv + Es	59,2 %	23,9 %
Ng / Nr	5	20
1 - R	54,1 %	15,2 %

$$R = \frac{ZVS(RT1)}{ZVIo(RT)}$$

Tabelle 6.3 - Einfluß von Ng/Nr auf die relativen Bedeutungen der Register- und Schaltnetz-bezogenen Faktoren im Gewinn des Simulators RT1 gegenüber der Instanzennetzsimulation

Der Faktor Rg stellt keine wesentliche Modellierungsverbesserung des Simulators RT2 dar. Er spiegelt lediglich die Tatsache wider, daß im Simulator RT1 der Instanzenaufruf und die Verwaltung der Variablen TF entfallen. Analytisch haben wir für Rg die Zeitverbrauchsausdrücke 28*t'*Ng (Simulator RT1) und 49*t'*Ng (Instanzennetzsimulation). Das bedeutet, daß RT1 eine Schleife für die Verarbeitung der Rufliste der Verknüpfungsglieder hat, die 1,75mal schneller ist als die Hauptschleife des Koordinators des Instanzennetzsimulators. Die große Bedeutung des Faktors Rg kommt hauptsächlich aus der Verarbeitungshäufigkeit dieser Schleife.

Dagegen stellen die Faktoren Em, Es und Muä+Mv wesentliche Optimierungen dar, weil sie Aktivitäten entsprechen, die im Simulator RT1 ganz entfallen. Jeder von diesen Faktoren, falls isoliert betrachtet, bringt deswegen einen Gewinn von 100% für den Simulator RT1. Daß sein Vorteil gegenüber dem Instanzennetzsimulator insbesondere in der Verarbeitung des Registerblocks liegt, hatten wir schon im Abschnitt 5.8 angedeutet.

Die Faktoren Rg und ZRg haben eine große Bedeutung, gerade wenn der Vorteil des Simulators RT1 am kleinsten ist. Dagegen gewinnen die Faktoren Em, Es und Muä+Mv an Wichtigkeit, genau wenn dieser Vorteil größer wird.

Die relative Bedeutung der Parameterzugriffe, und somit der Datenstruktur in der Instanzennetzmodellierung von Register-Transfer-Systemen ist deutlich vermindert gegenüber der Instanzennetzmodellierung von Gatterebenen-Systemen, was in der letzten Zeile der Tabelle 6.2 zu erkennen ist. Der Grund dafür ist die wesentlich einfachere Gestaltung der Instanzen in der Modellierung von Register-Transfer-Systemen (fast keine Sequenzermarkenverwaltung !), sowie der kleine Anteil der Register in der gesamten Anzahl von Elementen, weil wir nur für die Register eine Optimierung der Zugriffe auf die Anschlußvariablen gegenüber der Instanzennetzmodellierung erzielen können.

Dagegen haben die Operationen auf der Ereignisliste einen wichtigeren Gewinnanteil übernommen (bis 43,7 %), obwohl sie jetzt im Durchschnitt sogar einen kleineren Anteil des Zeitverbrauchs der Instanzennetzsimulation ausmachen: 12,1 % für das Register-Transfer-System S1, gegenüber 25,5 % in der Instanzennetzsimulation der Gatterebene. Dies ist verständlich, weil die Register-Transfer-Simulatoren keine Ereignisliste brauchen, während für die Gattersimulatoren die Ereignisliste sehr wichtig ist.

6.4 Simulation der Register-Transferebene, Modell RT2

Tabelle 6.4 zeigt alle Faktoren, die den gesamten Gewinn des Simulators RT2 gegenüber der Instanzennetzmodellierung von Register-Transfer-Systemen bilden. Die Gewinnfaktoren der Untergruppe A3 ergeben sich aus der gleichen Eigenschaft, die für diese Faktoren auch im Simulator RT1 verantwortlich

ist, und wurden im letzten Abschnitt schon besprochen. Im weiteren erklären wir die Gründe für die anderen Faktoren.

Gruppe	Faktor	SYSTEM S4			SYSTEM S5		
		% von ZVS(RT2)	% von ZVIo(RT)	% vom Gewinn (3)	% von ZVS(RT2)	% von ZVIo(RT)	% vom Gewinn (3)
C3	EK	0	14,5	19,9	0	15,4	43,8
	Ug	4,7	7,0	7,6	5,2	13,0	25,9
	Hg	22,9	4,6	- 3,0	21,5	7,2	- 25,0
Summe C3 (1)		27,6	23,8	21,4	26,7	33,2	38,0
C4	S	0	24,5	33,6	0	2,3	6,4
	Ur	0,9	1,8	2,1	0,3	0,5	0,8
	Hr	4,6	2,7	1,9	1,1	0,4	- 1,1
Summe C4		5,5	29,0	37,6	1,4	3,2	6,1
Summe C		33,1	52,8	59,0	28,1	36,4	44,1
A3	Em	0	14,1	19,4	0	2,1	5,9
	Muä	0	9,0	12,3	0	1,3	3,8
	SMm	0	2,5	3,5	0	0,4	1,1
	Mv	0	0	0	0	0	0
	Z	0,3	0,13	0,1	0,1	0,16	0,2
Summe A (2)		0,3	24,2	33,0	0,1	3,7	10,3
D4	Rg	11,6	11,2	10,7	9,7	17,7	30,0
	Rm	1,7	3,3	3,9	0,1	0,5	1,2
Summe D		13,3	14,5	14,6	9,8	18,2	31,2
B2	Pr	4,9	1,9	0,6	0,2	0,2	0,2
	Fg	15,5	7,3	3,8	20,7	18,4	8,7
	Pg	12,9	6,1	3,1	18,9	16,8	7,9
	Ag	6,4	3,0	1,6	21,5	19,2	9,1
O + V	Fs	8,8	2,6	0	0,4	0,3	0
	Am	4,7	1,1	- 0,4	0,3	0,13	- 0,15
	EL	0,3	34,9	25,6	0,1	3,3	9,1
	P	17,8	11,7	8,8	19,1	17,5	9,5

Bemerkungen: (1) Überschneidung zwischen EK und Ug
(2) Überschneidung zwischen SMm und Muä, SMm und Mv
(3) % des entstandenen Gewinnes, wenn die Faktoren, die Verlust für den spezifischen Simulator bringen, aus ZVS und ZVI ausgeschlossen sind

Tabelle 6.4 - Faktoren im Gewinn des Simulators RT2 gegenüber der Instanzennetzsimulation

6.4.1 Einschränkungen in der Topologie

Die grundlegende Idee für das Modell RT2 ist die Abwesenheit von Schleifen innerhalb des Register- und des Verknüpfungsblocks. Diese Eigenschaften erklären den meisten Gewinn des Simulators RT2 und bilden die Untergruppen C3 und C4 der Restriktionsklassen. In jeder Untergruppe gibt es einen Faktor (Hg bzw. Hr), der allein betrachtet einen Verlust für den Simulator RT2 gegenüber der Instanzennetzsimulation darstellt. Erst seine Existenz ermöglicht jedoch den Gewinn der anderen Faktoren der jeweiligen Untergruppe.

Die Abwesenheit von Schleifen innerhalb des Verknüpfungsblocks erlaubt eine Optimierung in der Implementierung des Simulators RT2, nämlich eine feste Auswertungsreihenfolge für die Verknüpfungsglieder gemäß einer statischen "Rufliste", die sich in folgenden Faktoren widerspiegelt:

Faktor EK - Zusätzliche Aufrufe von Verknüpfungsgliederinstanzen in der Instanzennetzsimulation im Verhältnis zum Simulator RT2 aufgrund der Auswirkung von Erkennungskonflikten (Parameter k > 1) -

Faktor Ug - Eintragung der Namen der Fan-Out-Elemente, die Verknüpfungsglieder sind, in die Rufliste (schrumpft im Simulator RT2 auf das Setzen des CALLBITs) -

Faktor Hg - Suche der Namen der Verknüpfungsglieder, deren Funktion auszuwerten ist, in der Rufliste -
- Dies bedeutet für den Simulator RT2 das Abprüfen aller CALLBITs, während die Rufliste in der Instanzennetzsimulation nur die Namen derjenigen Verknüpfungsgliederinstanzen enthält, deren Funktionen wirklich ausgewertet werden müssen.

Die Abwesenheit von Schleifen innerhalb des Registerblocks ermöglicht für den Simulator RT2 die Bearbeitung des Entscheidungs- und des Übergangsintervalls in einem einzigen Schritt, gemäß der umgekehrten Speisefolge der Register. Das bringt gegenüber der Instanzennetzsimulation folgende Gewinnfaktoren:

Faktor S - Slave-Funktion -
- Eine Reihe von Aktivitäten in der Instanzennetzsimulation, die der Trennung zwischen Slave- und Master-Funktion dienen, haben keine Entsprechung im Simulator RT2, nämlich die Eintragung der Namen der Slave-Instanzen in die Rufliste aufgrund der Umgebungsänderung der Master-Instanzen, der daraus folgende Aufruf der Slave-Instanzen, die Eintragung von Notizen in die Ereignisliste für die Slave-Instanzen, die Zeitfortschaltung, das Holen der vorher eingetragenen Notizen aus der Ereignisliste und der entsprechende Aufruf der Slave-Instanzen aufgrund der Zeitmeldung.

Faktor Ur -
- Er entspricht dem Faktor Ug, diesmal für die Register.

Faktor Hr -
- Er entspricht dem Faktor Hg, diesmal für die Register. Aus den analytischen Zeitverbrauchsausdrücke für diesen Faktor in beiden Modellierungen, 12*Nr für den Simulator RT2 und 15*(v+1)*r*Nr für den Instanzennetzsimulator, sehen wir, daß Hr sogar Gewinn für RT2 bringen kann, falls 15*(v+1)*r > 12 ist, was der Fall für das System S4 ist.

6.4.2 Einschränkungen in dem Instanzenrepertoire

Die Kenntnis über die Instanzen und deren Auswirkungen erlaubt Optimierungen in der Verarbeitung der Rufliste. Wir haben die daraus resultierenden Gewinnfaktoren in die Untergruppe D4 zusammengelegt. Diese Faktoren erfassen die Suche nach dem nächsten Element in der Rufliste nicht, weil diese Aktivität in die Faktoren Hg bzw. Hr verlagert wurde.

Faktor Rg - Verarbeitung der Rufliste für die Verknüpfungsglieder -
- Der Simulator RT2 braucht das Rücksetzen des CALLBITs und die Verwaltung der Variable CHE, während in der Instanzennetzsimulation der Instanzenaufruf, die Verwaltung der Variablen CHE und TF und das Rücksetzen der Variable FLAG erforderlich sind.

Faktor Rm - Verarbeitung der Rufliste für die Register (im Simulator RT2) bzw. für die Master-Instanzen -
- Wie Rg bis auf die Variable CHE, die der Simulator RT2 nicht braucht.

6.4.3 Sonstige Faktoren

Die spezifische Datenstruktur des Simulators RT2 erklärt den Gewinnfaktor Pr, der den Zugriff auf die Anschlußvariablen der Register (im RT2) bzw. der Master-Instanzen darstellt.

Faktor Pr -
- Der Zugriff auf die Anschlußvariablen erfolgt für den Simulator RT2 durch eine zweistufige, in der Instanzennetzsimulation durch eine dreistufige Datenstruktur.

Die Faktoren Fs, der keinen Gewinn für den Simulator RT2 bringt, und Am, der sogar einen kleinen Gewinn für die Instanzennetzsimulation bringt, haben wir schon bei der Interpretation des Gewinns des Simulators RT1 vorgestellt.

Damit alle Faktoren, die den Zeitverbrauch der beiden Simulatoren bilden, in der Tabelle 6.4 auftauchen, enthält die Tabelle noch die Faktoren
Fg (Zugriff auf den Anfang der Teil-FANOUT- bzw. UMGLISTE der Verknüpfungsglieder),
Pg (Zugriff auf die Anschlußvariablen der Verknüpfungsglieder) und
Ag (Ausführung der Funktionen der Verknüpfungsglieder).
Den durch diese Faktoren entstandenen Gewinn deckt jedoch

schon der Faktor EK ab.

6.4.4 Schlußfolgerungen

Wie für das Modell RT1 hängen die relativen Bedeutungen der verschiedenen Gewinnfaktoren stark davon ab, welchen Anteil der Elemente die Register und die Verknüpfungsglieder darstellen. Diese Tatsache bringen wir in der Tabelle 6.5, die die wichtigsten Gewinnfaktoren zusammenfaßt, zum Ausdruck. Genau wie für RT1 haben die Faktoren, die der Verarbeitung des Verknüpfungsblocks entsprechen, nur dann eine größere relative Bedeutung, wenn der gesamte Gewinn des Simulators RT2 gegenüber der Instanzennetzsimulation geringer ist. Trotzdem müssen wir hier den Faktor EK als Ergebnis einer wesentlichen Modellierungsverbesserung auszeichnen. Auf die Wichtigkeit der Faktoren Em und Muä hatten wir schon beim Modell RT1 hingewiesen. Jetzt kommt S dazu als wichtigster Faktor, der für den meisten Gewinn genau dann verantwortlich ist, wenn das Modell RT2 seinen Vorteil gegenüber der Instanzennetzmodellierung am meisten zeigt.

	S4	S5
EK + Ug + Rg	38,2 %	99,7 %
S + Em + Muä	65,3 %	16,1 %
Ng / Nr	5	20
1 - R	70,5 %	25,9 %

$$R = \frac{ZVS(RT2)}{ZVIo(RT)}$$

Tabelle 6.5 - Einfluß von Ng/Nr auf die relativen Bedeutungen der Register- und Schaltnetz-bezogenen Faktoren im Gewinn des Simulators RT2 gegenüber der Instanzennetzsimulation

Zusammenfassend können wir sagen, daß:
- Die feste Verarbeitungsreihenfolge für Register und Verknüpfungsglieder die wesentlichste Modellierungsverbesserung des Simulators RT2 ist (die Untergruppen C3 und C4 ergeben zwischen 44,1 % und 59,0 % des gesamten Gewinns gegenüber der Instanzennetzsimulation); und
- Als zweitwichtigste Optimierung das im Abschnitt 6.3.2 schon behandelte implizite Fortschaltungsmodell durch Master-Ruflisten genannt werden soll (die Untergruppe A3 erklärt 10,3 % bis 33,0 % des gesamten Gewinns).

7. Schlußfolgerungen

7.1 Gültigkeit der Vergleichsmethode

Im Abschnitt 3.1 haben wir unsere Vergleichsmethode dadurch gerechtfertigt, daß eine Komplexitätsanalyse O(n) unmöglich war, weil für die zu vergleichenden Algorithmen

- viele wichtige Parameter vorhanden sind, die wir nicht vernachlässigen können, und

- eine identische Abhängigkeit des Zeitverbrauchs beider Algorithmen von den jeweiligen Parametern besteht, wobei der Unterschied nur durch die verschiedenen multiplikativen Konstanten bemerkbar ist.

Diese Behauptungen können wir jetzt bestätigen, indem wir z.B. die vereinfachten Zeitverbrauchsausdrücke für alle Modellierungen betrachten: (4.29) und (4.30) im Abschnitt 4.10 für den Gattersimulator und die entsprechende Instanzennetzmodellierung, (5.36), (5.37) und (5.35) im Abschnitt 5.8 für die Register-Transfer-Simulatoren RT1 und RT2 und die entsprechende Instanzennetzmodellierung.

Aufgrund dieser Tatsache war es unvermeidbar, die Analyse durch die exakten Zeitverbrauchsausdrücke durchzuführen, und Annahmen über die Ausführungszeiten der Algorithmenschritte zu machen, um die Untersuchung überhaupt zu ermöglichen. Trotzdem scheinen uns die erzielten Ergebnisse allgemeingültig und insbesondere von dieser Annahme über die Ausführungszeiten völlig unabhängig zu sein, was die Untersuchungen über die Sensibilität des Effizienzgewinns bezüglich dieser Zeiten jetzt im Rückblick sogar überflüssig macht. Dafür sprechen zwei Eigenschaften unserer Analyse.

Erstens haben wir darauf hingewiesen, daß zwischen den Parametern Beziehungen bestehen, die wir formal nicht haben erfassen können. Als Folge dieser Lücke haben wir für die Parametervariierung einen Freiheitsgrad angenommen, der gar nicht vorhanden ist. Die Grenzgewinne wurden dadurch erreicht, daß wir extrem günstige (bzw. ungünstige) Parameterwerte festgelegt haben. Hätten wir diese nicht formal erfaßbaren Beziehungen berücksichtigen können, wären diese Sätze von Parameterwerten für die Grenzgewinne gar nicht entstanden. Das bedeutet, daß reale Grenzfälle näher beieinander liegen, als wir hier festgestellt haben. Ein Beispiel dafür ist das System S4, welches den höchsten Gewinn des Simulators RT2 gegenüber der Instanzennetzsimulation aufweist. Der Wert Ng/Nr = 5 deutet auf ein System mit komplexen Elementen hin, die mit den Werten Z = 10, Yg = 3 und k = 1,6 höchstwahrscheinlich nicht zu vereinbaren sind.

Zweitens hat das Kapitel 6 bewiesen, daß die berechneten Gewinne sich aus ganz identifizierbaren Eigenschaften der Simulationsalgorithmen ergeben. Wir haben nie mit bestimmten

Werten der Ausführungszeiten argumentieren müssen, um Effizienzgewinne zu erklären. Es ist sehr wohl möglich, daß ein anderer Satz von Ausführungszeiten andere Prozentsätze für die Gewinnfaktoren bewirken würde, aber das Wesen unserer Schlußfolgerungen wäre damit sicherlich nicht gestört.

7.2 Gewinn von Hardware-Simulatoren gegenüber der Instanzennetzmodellierung

Tabelle 7.1 faßt die wesentlichen numerischen Ergebnisse unserer Untersuchung zusammen, wobei 1/R bedeutet, wieviel langsamer die Instanzennetzsimulation im Verhältnis zum entsprechenden Hardware-Simulator ist.

	1 / R	
	höchster Wert	niedrigster Wert
G1	2,55	2,10
RT1	2,18	1,18
RT2	3,39	1,35

Tabelle 7.1 - Gewinnspanne der Hardware-Simulatoren gegenüber der Instanzennetzsimulation

Wir müssen zwei Eigenschaften dieser Zahlen überprüfen:
1) Stellen die aufgezeigten Zahlen wirklich Grenzfälle dar? und
2) Lassen sich diese Zahlen für alle Hardware-Simulatoren (auf der Register-Transfer- und Gatterebene selbstverständlich) verallgemeinern?

Zur ersten Frage haben wir schon im vorigen Abschnitt angedeutet, daß einerseits die Grenzfälle vielleicht sogar näher beieinander liegen. Andererseits aber hängen diese Zahlen von den von uns festgelegten Wertebereichen der Parameter ab, und wir können keinen Beweis dafür präsentieren, daß diese Wertebereiche tatsächlich mit der Realität der möglichen Systeme übereinstimmen. Wenn wir aber alle Untersuchungen der Sensibilität der verschiedenen Gewinne bezüglich der Parameter durchlaufen, können wir feststellen, daß das Verhalten immer asymptotisch ist und Parameterveränderungen über die festgelegten Wertebereiche hinweg keine bedeutsame Änderung der Zahlen bringen, bis auf einen einzigen Fall, nämlich das Verhältnis Ng/Nr bei der Register-Transfer-Simulation. Ein Wert von Ng/Nr wesentlich kleiner als 5 scheint uns jedoch sehr unwahrscheinlich.

Zur zweiten Frage müssen wir die beiden Hardwaremodellierungsebenen getrennt betrachten. Für die Gatterebenen-Simulation haben wir gezeigt, daß der Einfluß des

Zeitmodells sehr gering ist. Es scheint uns, daß die Grenzen höchstwahrscheinlich für alle Simulatoren stimmen, die eine Verzögerung den Gattern zuordnen. Nehmen wir dagegen "Zero-Delay"-Simulatoren, gilt nun die Analyse der Verknüpfungslogik auf der Register-Transferebene. Wird eine feste Auswertungsreihenfolge für die Gatterfunktionen verwendet, die Auswirkungen von Erkennungskonflikten vermeidet, liegt die höhere Grenze für 1/R bei k, was sicherlich eine Zahl kleiner als 2 ist. Wird diese Optimierung nicht realisiert, ist keine wesentliche Verbesserung gegenüber der Instanzennetzsimulation zu erwarten.

Die Register-Transferebene bietet keine große Auswahl von Simulationsmodellen, wenn wir die Simulation von Systemen ausschließen, die durch prozedurale Register-Transfer-Sprachen /BAR75/ beschrieben werden. Eine Variierung des Elementenrepertoires haben wir schon in der Untersuchung durch den Parameter Z berücksichtigt. Falls Verzögerungen für die Verknüpfungsglieder vorgesehen sind, gelten unsere Modelle RT1 und RT2 nicht mehr. Ein Simulator mit Ereignisliste wäre unbedingt erforderlich, die Zeitfortschaltung müßte aussehen wie auf der Gatterebene, und wir würden vielleicht eine dieser Ebene ähnliche Gewinngrenze erreichen. Die Einbeziehung von zweiflankengetakten Registern bringt keine wesentliche Änderung für die Algorithmen. Latchregister dagegen würden einen ganz anderen Simulationsmechanismus benötigen, und die Folgen für den Gewinn eines Register-Transfer-Simulators gegenüber einer entsprechenden Instanzennetzsimulation sind unseren Ergebnissen nicht zu entnehmen.

7.3 Verallgemeinerung der Gründe für den Effizienzgewinn von spezifischen Simulatoren

Wir können aus den Schlußfolgerungen des Kapitels 6 allgemeingültige Aussagen über den Effizienzgewinn von spezifischen Simulatoren für beliebige Systemklassen gegenüber entsprechenden Instanzennetzmodellierungen gewinnen.

Ein wesentlicher Gewinn ergibt sich, wenn wir für die Systemelemente bestimmte Aktivitäten zu allen Modellzeitpunkten erkennen können, und eine eindeutige Verarbeitungsreihenfolge für diese Aktivitäten besteht, so daß die Durchführung einer neuen Aktivität erst dann anfängt, wenn für alle Elemente die unmittelbar vorhergehende Aktivität schon durchgeführt worden ist. In einem spezifischen Simulator läßt sich diese Eigenschaft sehr leicht implementieren. Die Sequentialisierung dieser Aktivitäten in einem Instanzennetzmodell dieses Systems ist dagegen nur durch explizite Aufteilung einer Modellzeit in Schritte aufzwingbar, die den einzelnen Aktivitäten entsprechen. Der Effizienzverlust der Instanzennetzsimulation gegenüber dem spezifischen Simulator kommt dann aus der Notwendigkeit, Ereignisse planen und eine explizite Zeitfortschaltung durchführen zu müssen, um diese Aufteilung nachzubilden.

Wir können auf die Aufteilung verzichten und die Instanzen so gestalten, daß durch Sequenzermarkenverwaltung die bei dem Aufruf erforderliche Aktion erkannt wird. Es besteht aber dann auch die Gefahr, daß sich Aufrufgründe überlappen und Erkennungskonflikte zwischen den Aktivierungen von Instanzen entstehen, die aus verschiedenen Gründen aufzurufen sind. Aufgrund der Erkennungskonflikte können Instanzen mehrfach zu einem Modellzeitpunkt aktiviert werden und folglich falsche Aktionen durchführen, die erkannt und korrigiert werden müssen. Erkennungskonfliktbehandlung und Sequenzermarkenverwaltung bringen dann den meisten Effizienzverlust für die Instanzennetzsimulation.

Ein zweiter wichtiger Gewinngrund ist die Eigenschaft von spezifischen Simulatoren, Umgebungsänderungseffekte in künftige bekannte Modellzeiten verlagern zu können, ohne dafür Notizen in eine Ereignisliste eintragen zu müssen. Dies ist möglich, wenn wir wissen, daß die durch die Umgebungsänderungen betroffenen Instanzen erst zu diesem späteren Zeitpunkt eine Aktion aufgrund dieser Änderungen durchzuführen haben, und wenn wir diese Instanzenaktionen spezifischen Modellzeiten eindeutig zuordnen können. Der spezifische Simulator für diese Systemklasse braucht nur verschiedene Ruflisten für die verschiedenen möglichen künftigen Zeiten, eine Elementenaktion erfolgt nicht zum Zeitpunkt der für sie verantwortliche Umgebungsänderung. In der Instanzennetzmodellierung ist diese Effektverzögerung von Umgebungsänderungen nicht möglich. Die Instanzen müssen unmittelbar zu dem Zeitpunkt aktiviert werden, zu dem die Umgebungsänderung stattgefunden hat, und durch Zeitmeldung neue Aufrufe zu der gewünschten Zeit veranlassen.

Teilen wir die Instanzen dieses Systems in Klassen ein, die aus den Instanzen bestehen, die einem selben Aufrufzeitpunkt zugeordnet sind. Nehmen wir an, daß wir für eine bestimmte Instanzenklasse wissen, daß der Anteil der zu einem beliebigen Zeitpunkt zu aktivierenden Instanzen immer sehr groß ist. Es lohnt sich dann schon, die Planung von Ereignissen entfallen zu lassen, die die Folge einer Aktion einer Instanz dieser Klasse wäre, wenn die Instanz aufgrund einer Umgebungsänderung aufgerufen wird. Vielmehr bringen wir alle Instanzen der Klasse zu diesem künftigen Zeitpunkt in Aktion. Dafür ist eine zusätzliche Instanz notwendig, die mit allen Instanzen der betrachteten Klasse verbunden ist. Diese Instanz plant ein Ereignis für den gewünschten Zeitpunkt und als Auswirkung ihrer Aktion meldet sie eine Umgebungsänderung, die die Aktivierung aller Instanzen der Klasse bewirkt. Der zusätzliche Simulationszeitverbrauch aufgrund dieser neuen Instanz ist bei weitem kleiner als der, den wir sparen, weil Aufrufe aufgrund von Umgebungsänderungen deutlich weniger Zeit verbrauchen als Aufrufe aufgrund von Zeitmeldungen, die Operationen auf der Ereignisliste voraussetzen.

Der Effizienzverlust der Instanzennetzsimulation gegenüber dem spezifischen Simulator besteht jedoch immer, sei es durch den unmittelbaren Aufruf aufgrund der Umgebungsände-

rung und die notwendigen Operationen auf der Ereignisliste, sei es durch die mit dieser speziellen zusätzlichen Instanz verbundenen Aktivitäten.

Ein dritter Gewinngrund für die spezifischen Simulatoren entsteht, wenn die Topologie der Systemklasse bekannt ist und sie eine optimale Aufrufreihenfolge für alle Instanzen bzw. die Instanzen eines bestimmten Typs erlaubt. Die Aufzwingung einer gewissen Aufrufreihenfolge ist in der Instanzennetzmodellierung praktisch unmöglich. Durch eine angebrachte Modellierung müssen wir dafür sorgen, daß mit der Instanzennetzsimulation dieselben Ergebnisse wie beim spezifischen Simulator erzielt werden, obwohl zu erwarten ist, daß während einer Systemüberführung die Instanzennetzsimulation Zwischenzustände zeigt, die mit den Zwischenzuständen des spezifischen Simulators nicht übereinstimmen. Eine allgemeine Lösung können wir hier nicht angeben, weil sie von der Natur des Problems abhängt.

Ein vierter nennenswerter Gewinngrund ist die Implementierung einer Organisation für die Ereignisliste, die für eine bekannte Zeitverteilung der Ereignisse zugeschnitten ist. Ein bedeutsamer Gewinn gegenüber der Instanzennetzsimulation folgt selbstverständlich erst dann, wenn eine große Ereignisaktivität erforderlich ist. Die Suche nach einer optimalen allgemeinen Organisation der Ereignisliste für den Instanzennetzsimulator scheint uns gerechtfertigt, nicht nur um einen größeren Verlust gegenüber den spezifischen Simulatoren zu vermeiden, wenn die Systemklasse von Natur her eine große Ereignisaktivität benötigt. Wir müssen uns daran erinnern, daß die Ereignisnotizen in der Instanzennetzsimulation den einzigen Mechanismus darstellen, der die Auswirkung einer Instanzenaktion in die Zukunft erlaubt. Beispielweise braucht ein Register-Transfer-Simulator keine Ereignisnotizen, und wir können bis 43 % seines Gewinns gegenüber der Instanzennetzsimulation den Operationen auf der Ereignisliste zuschreiben.

Letztlich möchten wir die Kenntnis über das Instanzenrepertoire als Gewinngrund auszeichnen. Schon die einfache Tatsache, daß die in der Hauptschleife (nämlich Verarbeitung der Rufliste) des Koordinators des Instanzennetzsimulators vorhandenen Verwaltung der Variablen TF und Instanzenaufruf unnötig sind für einen Register-Transfer-Simulator, bedeutet für diesen Simulator bei jedem Schleifendurchlauf einen Gewinn von 43 % des Zeitverbrauchs des Instanzennetzsimulators. Diese Optimierung kann bis 52 % des gesamten Gewinns des Register-Transfer-Simulators gegenüber der Instanzennetzsimulation erklären. Die Verarbeitung der Rufliste (d.h. Instanzennamen in die Rufliste eintragen und später abholen) kann sogar ganz entfallen, wenn der Kern des spezifischen Simulators die Durchführung der Instanzenaktion übernehmen kann, unmittelbar nach der Erkennung einer neuen Ereignisnotiz.

Wir können anhand dieser Untersuchung selbstverständlich nicht behaupten, daß alle Verluste, die die Instanzennetzmodellierung gegenüber einem spezifischen Simulator aufweist, unbedingt durch die hier erwähnten Gründen erklärbar sind. Die Arbeit hat sich zum Beispiel auf gerichtete Netzsysteme beschränkt. Ungerichtete Systeme stellen möglicherweise neue Vergleichsbedingungen, die hier unbeachtet bleiben. Vielmehr sollen wir diese Gründe als Beispiele annehmen, und die Untersuchung dient insbesondere dazu, dem Verlust der Instanzennetzsimulation gegenüber spezifischen Simulatoren Grenzen zu setzen. Aus dieser Arbeit scheint uns ganz klar hervorzugehen, daß dieser Verlust vernünftigerweise begrenzt ist, solange wir als Instanzennetze Systeme modellieren, die typische Operandenkommunikation zwischen den Instanzen benötigen. Verlustfaktoren zwischen 1 und 5 sind möglich, vielleicht etwas mehr, aber wesentlich höhere Zahlen sollen nicht vorkommen.

Schließlich möchten wir nochmals darauf hinweisen, daß ein fairer Vergleich erst möglich ist, wenn der gleiche Benutzerkomfort in beiden zu vergleichenden Simulatoren vorausgesetzt ist. Da wir uns nur mit den Eigenschaften befassen wollten, die unbedingt für den Instanzennetz- bzw. den spezifischen Simulator notwendig sind, haben wir auf jegliche Komfortbetrachtungen verzichtet. Wenn ein Komfort aber dabei wäre, und selbstverständlich in beiden Simulatoren, würde sich dieser Verlustfaktor angesichts des erhöhten Anteils gemeinsamer Aktivitäten höchstwahrscheinlich verringern.

ANHANG A. Literaturverzeichnis

/ALI78/ Alia,G. , P.Ciompi und E.Martinelli, "LSI Components Modeling in a Three-Value Functional Simulation", Proceedings of the 15th Design Automation Conference, Jun 1978, S. 428-438

/BAR75/ Barbacci,M., "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", IEEE Transactions on Computers, Vol. C-24, Feb. 1975, S. 137-150

/BRE76/ Breuer,M.A. und A.D.Friedman, Diagnosis & Reliable Design of Digital Systems, Computer Science Press Inc., California, 1976

/CD74/ Sonderausgabe der Zeitschrift "Computer Design" über Register-Transfer-Sprachen, Vol. 7, Dez. 1974

/CHA71/ Chappel,S.G. und S.S.Yau, "Simulation of Large Asynchronous Logic Circuits Using an Ambiguous Gate Model", AFIPS Conference Proceedings, Fall Joint Computer Conference, 1971, S. 651-661

/COM79/ Comfort,J.C., "A Taxonomy and Analysis of Event Set Management Algorithms for Discrete Event Simulation", Proceedings of the 12th Annual Simulation Symposium, Tampa, Florida, März 1979, S. 115-146

/DAC/ Proceedings der verschiedenen "Design Automation Conferences"

/EIC65/ Eichelberger,E.B., "Hazard Detection in Combinational and Sequential Switching Circuits", IBM Journal of Research & Development, Vol, 9, Nr. 2, März 1965, S. 90-99

/FRA77/ Franta,W.R. und K.Maly, "An Efficient Data Structure for the Simulation Event Set", Communications of the ACM, Vol. 20, Aug. 1977, S. 596-602

/GOR78/ Gordon,G., System Simulation, zweite Ausgabe, Prentice-Hall Inc., Englewood Cliffs, N.J., 1978

/HEM75/ Hemming,C.W. und S.A.Szygenda, "Register Transfer Language Simulation", in Breuer,M.A., Digital System Design Automation: Languages, Simulation and Data Bases, Computer Science Press Inc., Cal., 1975

/KIV69/ Kiviat,P.J., "Digital Computer Simulation: Computer Programming Languages", The Rand Corporation, Memorandum RM-5883-PR, Santa Monica, Cal., Jan. 1969

/KNU73/ Knuth,D., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, zweite Ausgabe, Addison-Wesley Publ. Co., Reading, Mass., 1973

/LEI80/ Leinwand,S.M. und T.Lamdan, "Models of Control at Register Transfer Level", Information Processing 80 (Proccedings of the IFIP Congress 1980), Hrsg. S. Lavington, S. 163-168

/NAN71/ Nance,R.E., "On Time Flow Mechanisms for Discrete System Simulation", Management Sciences: Theory, Vol. 18, Nr. 1, 1971, S. 59-73

/POT82/ Pott,E.G., "Klassifikation von Simulationskonzepten für Instanzennetze", Dissertation, Universität Kaiserslautern, Fachbereich Elektrotechnik, 1982

/SZY70/ Szygenda,S.A. , D.Rouse und E.W.Thompson, "A Model and Implementation of a Universal Time Delay Simulator for Large Digital Nets", AFIPS Conference Proceedings, Spring Joint Computer Conference, 1970, S. 207-216

/SZY75/ Szygenda,S.A. und E.W.Thompson, "Digital Logic Simulation in a Time-Based, Table-Driven Environment, Part 1: Design Verification", Computer, Vol. 8, März 1975, S. 24-36

/THO75/ Thompson,E.W. und S.A.Szygenda, wie oben, "Part 2: Parallel Fault Simulation", S. 38-49

/ULR69/ Ulrich,E.G., "Exclusive Simulation of Activity in Digital Networks", Communications of the ACM, Vol. 12, Nr. 2, Feb. 1969, S. 102-110

/VAU75/ Vaucher,J.G. und P.Duval, "A Comparison of Simulation Event List Algorithms", Communications of the ACM, Vol. 18, Nr. 4, Apr. 1975, S. 223-230

/WEN74/ Wendt,S., Entwurf komplexer Schaltwerke, Springer-Verlag, Berlin, 1974

/WEN79a/ Wendt,S., "The Programmed Action Module: An Element for System Modelling", Digital Processes, Vol. 5, 1979

/WEN79b/ Wendt,S., "Blockorientiertes interaktives Simulationssystem BORIS", Informationsschrift, Universität Kaiserslautern, Fachbereich Elektrotechnik, Dez. 1979

/WEN82/ Wendt,S., "Einführung in die Begriffswelt allgemeiner Netzsysteme", Regelungstechnik, 30.Jahrgang, Heft 1, 1982

/WEN83/ Wendt,S., "Grundlegende systemtechnische Begriffe und Modelle der Informationstechnik", Bericht 83 B-14, Universität Kaiserslautern, Fachbereich Elektrotechnik, Jan. 1983

/WUL81/ Wulf,W.A. et al., Fundamental Structures of Computer Science, Addison-Wesley Publ. Co., Reading, Mass., 1981

/WYM75/ Wyman,F.P., "Improved Event-Scanning Mechanisms for Discrete Event Simulation", Communications of the ACM, Vol. 18, Nr. 6, Jun. 1975, S. 350-353

Tabellarischer Lebenslauf

10.1.1953	geboren in Porto Alegre (Brasilien) als Sohn des Ehepaars Helio A. S. Wagner und Maria Emilia R. Wagner
1959-1963	Besuch der Hauptschule "Joao Wesley" in Porto Alegre
1964-1970	Besuch des Gymnasiums "Colegio Israelita Brasileiro" in Porto Alegre
1971-1975	Studium der Elektrotechnik, Vertiefung Elektronik, an der "Universidade Federal do Rio Grande do Sul" (UFRGS) in Porto Alegre, mit Abschluß als Dipl.-Ing.
1975-1977	Fortbildung in der Informatik, Vertiefung Digitale Systeme, an der "Universidade Federal do Rio Grande do Sul" mit Abschluß als M.Sc.
seit 1976	Assistent-Professor für Informatik an der "Universidade Federal do Rio Grande do Sul" mit Lehr- und Forschungstätigkeiten auf dem Gebiet "Digitale Systeme"
seit 1980	Aufenthalt im Fachbereich Informatik der Universität Kaiserslautern mit dem Ziel der Promotion, als Stipendiat des CNPq ("Conselho Nacional de Desenvolvimento Cientifico e Tecnologico"), Forschungsbeirat der brasilianischen Regierung