Visualisierung von Hazards und Races in Schaltnetzen

Studienarbeit

im Fachbereich Informatik der Universität Hamburg,
Arbeitsbereich TECH

vorgelegt von Stephan Siemsglüss,
im Februar 1997

Inhaltsverzeichnis

A Einleitung

Hazards und Races sind Laufzeiteffekte, welche in nicht-getakteten Schaltwerken auftreten können. Das Ignorieren dieser Erscheinungen kann unerwünschte Probleme mit sich bringen. Das dieser Arbeit zugrundeliegende Programm will versuchen, die Ursachen für das Entstehen von Hazards und den komplexeren Wettläufen pädagogisch sinnvoll dem Betrachter interaktiv darzustellen.

Diese begleitende Schrift wird kurz die Problematik der Laufzeiteffekte behandeln, der Schwerpunkt liegt jedoch einerseits darin, das gewählte Programmdesign zu begründen, andererseits die vorgenommene Implementation vorzustellen.

A.1. Hazards

Nach DIN-Norm 44300/93 ist ein Schaltnetz definiert als ‘ein Schaltwerk, dessen Wert am Ausgang zu irgendeinem Zeitpunkt nur vom Wert am Eingang zu diesem Zeitpunkt abhängt.’ Bei dieser Definition realisiert man die Netze mit idealen Verknüpfungsgliedern, die Eigenschaften realer Verknüpfungsglieder bleiben genauso unberücksichtigt wie die unter bestimmten Umständen nicht zu vernachlässigende Signallaufzeit auf den Leiterbahnen.

Für das fehlerfreie Funktionieren der zu entwerfenden Netze ist die allen realen Gliedern innewohnende Eigenschaft der Signallaufzeit jedoch äußerst wichtig. Deshalb dient diese Größe auch als Kennzahl integrierter Bausteine. Die Laufzeit kann Abweichungen der Schaltvariablen von den theoretischen Belegungen bei idealen Verknüpfungsgliedern verursachen. Diese Falsch-Belegungen der Schaltvariablen treten nach einer Änderung der Werte einer oder mehrerer Eingangsvariablen auf. In ihrer einfachsten Form bezeichnet man sie als ‘Hazards’. Die Gefahr des Auftretens von Fehlbelegungen durch Hazards hängt allein von den betreffenden Laufzeiten der verschiedenen Signalwege ab. Hazards lassen sich nach unterschiedlichen Aspekten klassifizieren, zum einen nach der Ursache ihres Auftreten, man spricht von Struktur- und Funktionshazards, weiter nach dem Erscheinungsbild der Hazards im

Signalverlauf, gebräuchlich sind 0- und 1-Hazards, statische und dynamische Hazards, und schließlich kann man Hazards auch in der Abhängigkeit zu ihrer Umgebung betrachten, also Hazards in Schaltnetzen und in Schaltwerken. Bei letzteren differenziert man Hazards und Races in Abhängigkeit der vorliegenden Rückkopplungen.

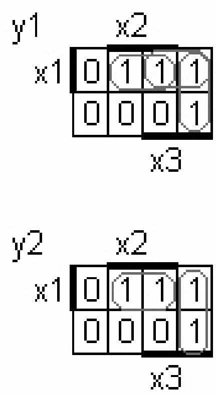
A.2. Strukturhazards

Besteht die Gefahr einer Fehlbelegung der Schaltvariablen durch den Wertewechsel an einem einzigen Eingang, bezeichnet man dieses als ‘Strukturhazard’. Das zugrundeliegende KV-Diagramm zeigt dann nicht-überlappende Schleifen, vgl. Abb.A.1. Der Hazard tritt also wegen der besonderen Struktur des Schaltnetzes auf und läßt sich entweder durch Hinzunahme eines zusätzlichen UND-Terms oder ODER-Terms, welcher die beiden Schleifen verbindet, eliminieren. Man kann sich dieses gut an den KV-Diagrammen in Abb.A.2 veranschaulichen. y2 ist dabei die vollständig minimierte Schaltfunktion der nebenstehenden Wertetafel, zur Realisierung werden ein ODER- und 2 UND-Gatter benötigt, wenn man Existenz von Eingängen voraussetzt, die sowohl das normale als auch das invertierte Signal aufnehmen können. y1 ist redundant gebildet, indem noch UND-Term x1x3 disjunktiv ergänzt wird. Es ist ohne weitere Erläuterungen klar, daß eine Schaltfunktionsbildung mit Redundanz die Schaltung zum einen teurer, zum zweiten komplizierter macht.

Abb.A.1 Notwendige Struktur für Struktur-hazards

Abb.A.2 Redundante Schleifenbildung zur Vermeidung von Struktur-hazards

x1	x2	x3	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



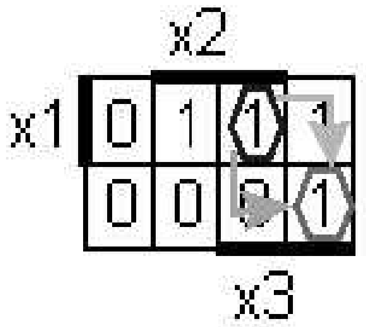
y1 = x1x2 v !x2x3 v x1x3
y2 = x1x2 v !x2x3

A.3. Funktionshazards

Verursachen gleichzeitige Wertewechsel an mindestens zwei Eingängen die Gefahr einer Fehlbelegung, so spricht man von einem ‘Funktionshazard’. Dabei konkurrieren mindestens zwei Umschaltreihenfolgen von Werten im KV-Diagramm miteinander, von denen wenigstens eine durch einen im Idealfall nicht vorgesehenen Falschwert führt. Durch strukturelle Maßnahmen läßt sich hier keine Abhilfe finden. Die unterschiedlichen Zustandswechsel bei gleichzeitigem Wechsel zweier Eingänge verdeutlicht Abb.A.4. Hier wird der Fall betrachtet, daß im Zustand x1=1, x2=1, x3=1 gleichzeitig x1 und x2 ihre Werte wechseln. Die beiden Pfeile zeigen die möglichen Wege durch die vom Wechsel berührten Zustände an. Wechselt die Schaltung durch den Zustand x1=0, x2=1, x3=1, tritt ein Hazardfehler auf, führt der Wechsel durch x1=1, x2=0, x3=1, kommt es zu keiner Fehlbelegung. Entscheidend darüber, welcher Weg genommen wird, sind die Laufzeiten über die betroffen Leiterbahnen und Gatter. Durch ein Schaltungsdesign, welches diese Art von Hazards berücksichtigt und Laufzeiten mit berücksichtigt, lassen sich Probleme aufgrund funktionaler Hazards vermeiden.

Abb.A.3 Notwendige Struktur für Funktionshazards

Abb.A.4 Mögliche Wege der Zustandswechsel bei Funktionshazards



A.4. Statische und dynamische 0- und 1-Hazards

Erwartet man am Ausgang durchgehend einen Nullpegel, bemerkt aber das ein- oder mehrfache Auftreten eines Einspegels, spricht man von einem statischen 0-Hazard. Erwartet man hingegen einen einmaligen Wechsel vom Null- auf den Einspegel, erhält aber zwei oder mehr Wechsel zwischen Null- und Einspegel vor dem abschließenden verbleiben auf Einspegel, so beobachtet man einen dynamischen 0-Hazard. Entsprechendes gilt für die 1-Hazards.

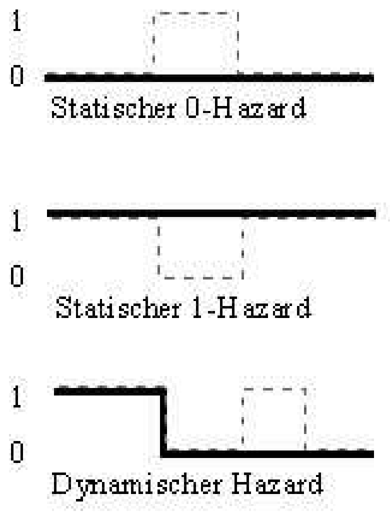


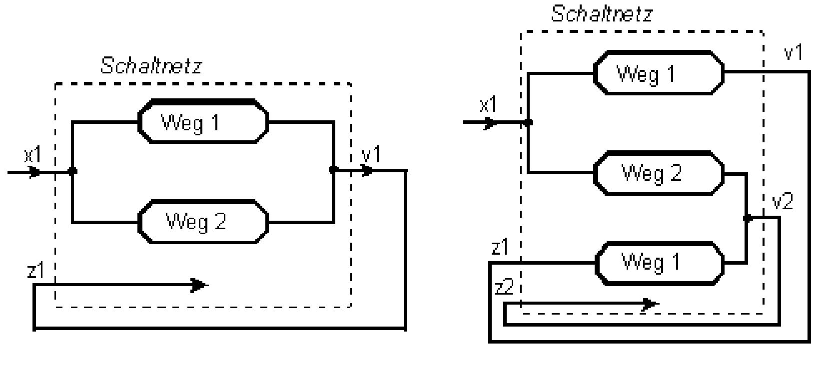
Abb.A.4 Signalverläufe bei Hazards

A.5. Schaltnetzhazards, inessentielle und essentielle Hazards

Verzweigen und vereinigen sich wieder die Wegbahnen einer einzelnen Variablen innerhalb eines Schaltnetzes, so kann man von einem Schaltnetzhazard sprechen, speziell ein durch Erweitern der Schaltfunktion entfernbarer Strukturhazard.

Im Schaltnetz entsteht ein Funktionshazard, wenn sich der Weg 2 erst über eine Rückkopplung mit Weg 1 vereinigt. In diesem Zusammenhang spricht man dann von einem essentiellen Hazard, falls eventuell eine Fehlbelegung der Schaltvariablen verursacht wird, andernfalls von einem nicht-, bzw. inessentiellem Hazard.

Abb.A.5 Notwendige Struktur für Schaltnetzhazards und essentielle Hazards



A.6. Races

Die komplizierteste Variante von Funktionshazards sind die sogenannten Races, bzw. Wettläufe. Sie kommen dadurch zustande, daß beide Wege über eine eigene Rückkopplung geführt sind. Falls der mögliche Hazard unerwünschte Belegungen verursachen kann, spricht man von critical Races, bzw. kritischen Wettläufen, andernfalls von unkritischen Wettläufen.

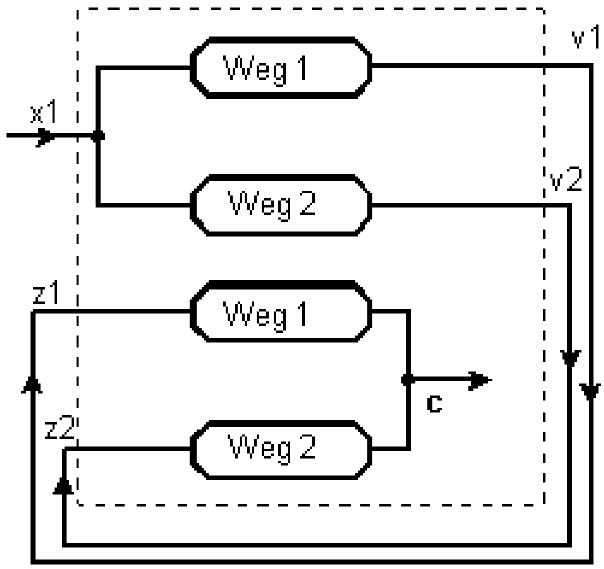


Abb.A.6 Notwendige Struktur für Wettläufe

B Tutorial

An dieser Stelle sollen exemplarisch die ersten Schritte beim Abspielen einer Animation beschrieben werden. Nach dem Start des Programms wird automatisch das Einführungskapitel gestartet. Nach einer kurzen Initialisierungsphase, in welcher einige programminterne Aufgaben wie z.B. das Laden der Animationsbilder durchgeführt werden, erscheint im Textfeld am unteren Rand des Applikationsfenster ein Text wie in Abb.B.1.

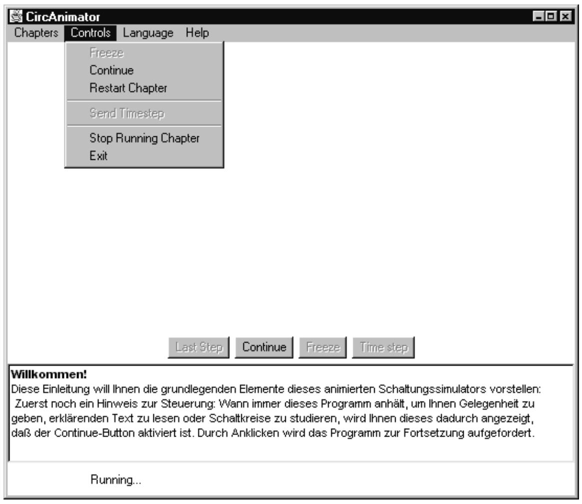


Abb.B.1
Programmstart

Gleichzeitig werden jene *Buttons* außer Funktion gesetzt (*ausgegraut*), deren Betätigung im Augenblick keinen Sinn ergeben würde. Dieses sind z.B. am Anfang der “Last Step”-, “Freeze”- und “Time step”-Button. Auch gleichlautende Menüpunkte werden temporär ausgeschaltet. Der einzige Button, der noch drückbar ist, ist jener “Continue”-Button, welcher dem Programm signalisiert, daß der Benutzer die Erläuterungen gelesen hat und den nächsten Schritt der Animation sehen, bzw. die nächste Erläuterung lesen möchte.

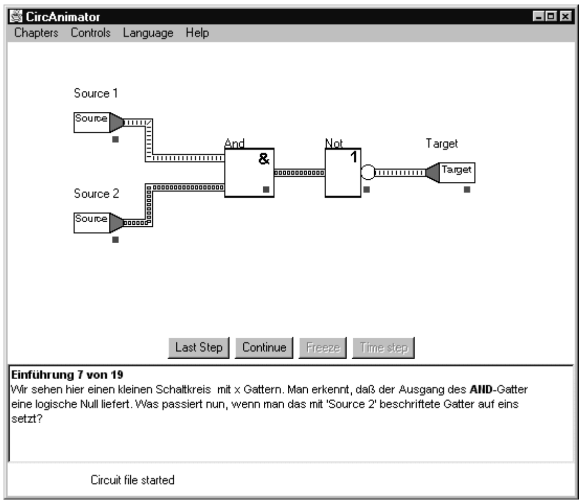


Abb.B.2
Schaltung

So klicke denn auch der Benutzer schließlich auf diese Schaltfläche. Ein neuer Text erscheint, welcher weitere Programmelemente vorstellt. Wenn sich der Benutzer dann bis zur Erläuterung Nr.7 vorgearbeitet hat, wird das erste Mal ein Schaltkreis geladen, um einige der schaltkreisspezifischen Elemente vorzuführen.

Wir sehen in Abb.B.2 schon fast alle wesentlichen Bestandteile: Gatter, Leiterbahnen, aktuell anliegende Signalwerte auf den Leiterbahnen und aktuelle Ausgangswerte der Signalgeneratoren (*sources*) und Signalempfänger (*targets*). Ein roter Punkt an einem Gatter bezeichnet hierbei eine ausgehende logische Eins, entsprechend ein grüner Punkt eine logische Null.

Wir bemerken weiterhin, daß an dieser Stelle nebem dem “Continue”- auch der “Last step”-Button anwählbar sind. Letztere Schaltfäche erlaubt dem Benutzer, die zuvor gezeigte Erläuterung noch einmal anzuschauen. Im weiteren Verlauf, der Benutzer möge sich davon überzeugen, kommt man bei Schritt 13 der Einführung zu einer automatischen Sequenz. Diese ist innerhalb eines Kapitels ein eigenständiger Abschnitt, in welchem zu bestimmten Zeitpunkten Ein-, bzw. Ausgangswerte an Gattern gesetzt und passende Texte ausgegeben werden. Während einer automatischen Sequenz kann der Ablauf durch Anwählen des “Freeze”-Buttons unterbrochen werden (Abb.B.3). Zwar wird der Animationseffekt auf den Leitungen nicht unterbrochen, schließlich hat die Implementierung dies*es Features* einige Mühen gekostet, doch wird die Uhr, welche den Ablauf steuert und

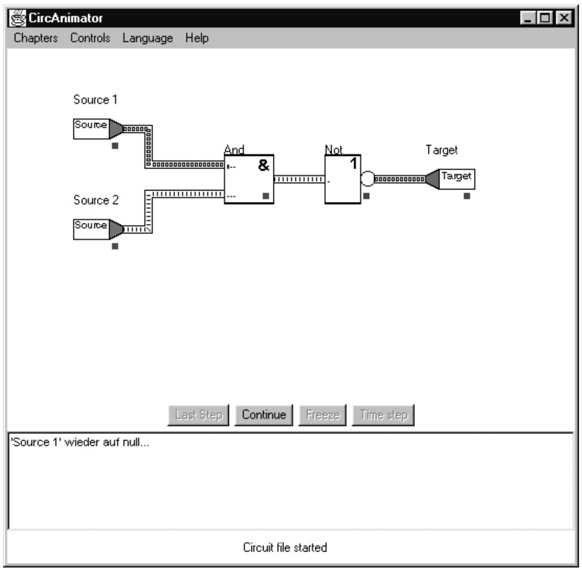


Abb.B.3 Automatische Sequenz

zeitabhängig die nächste Sequenzanweisung bearbeitet, angehalten. Die angehaltene Uhr läßt sich hier daran erkennen, daß einerseits der “Freeze”-Button ausgeschaltet, andererseits aber der “Continue”-Button eingeschaltet, d.h. wählbar ist. Während des Ablauf einer Sequenz sind nur der “Freeze”- und der “Last step”-Button für gewöhnlich anklickbar. Wobei noch zu ergänzen ist, daß “Last step” in einer Sequenz einen Neustart eben dieser Sequenz auslöst.

Ein weiteres wichtiges Detail, welches bisher noch nicht erwähnt wurde, ist auch in Abb.B.3 sichtbar, namentlich Gatterschaltverzögerungen. In der sogenannten Simulationsbetriebsart “time stepped mode”, bzw. Zeitschrittmodus genannt, wird in einer Sequenz automatisch etwa jede Sekunde ein Zeitsignal an die Simulations*engine* gesendet, welche daraufhin die logischen Signale durch den Schaltkreis errechnet, Signaländerungen den Gattern mitteilt und eine Änderung der Anzeigen bewirkt (eine genaue Beschreibung dieser Betriebsart findet sich in Abschnitt D.1.3.4.3). Was nun die Verzögerungen angeht, so simulieren diese, daß in realen Schaltungen sowohl Verzögerungen durch die Länge der Leiterbahnen als auch durch die Schaltzeit von Gattern auftreten. Grundsätzlich hat jedes Gatter wenigstens eine Verzögerung von einer Zeiteinheit (ZE), maximal kann sie 10 ZE betragen. Kleine Striche oder Quadrate geben innerhalb eines Gatters an, wie hoch die Verzögerung ist. Ändert sich der Eingang an einem Gatter, so daß der neue Wert zu einer Änderung des Ausgangswertes führen würde, zählt die Logikroutine bei jedem Zeitschritt die Verzögerung an dem geänderten Eingang bis null herab und zieht dann jenen Wert zur Ermittlung des Ausgangswertes heran. In Abb.B.3 können wir sehen, daß am oberen Eingang des Und-Gatters (eingestellte Verzögerung: 3 ZE) eine Signaländerung eingetreten ist und noch ein Zeitschritt fehlt, bis dieses Signal eine Änderung des Ausgangswertes von eins auf null bewirkt.

Die letzte wichtige Funktion, welche im Einführungskapitel vorgestellt wird, ist die des Signalplotters. Der Signalplotter ist in einem eigenen Fenster realisiert, wo über einer Zeitachse die zu einer bestimmten Zeit anliegenden Signale an verschiedenen Gattern graphisch dargestellt werden (Abb.B.4). Meist wird diese Aufzeichnungsfunktion in späteren Kapiteln verwendet, um Signalverläufe leicht modifizierter Schaltungen miteinander vergleichen zu können und daran Situationen mit und ohne Hazardgefahr vorzuführen.

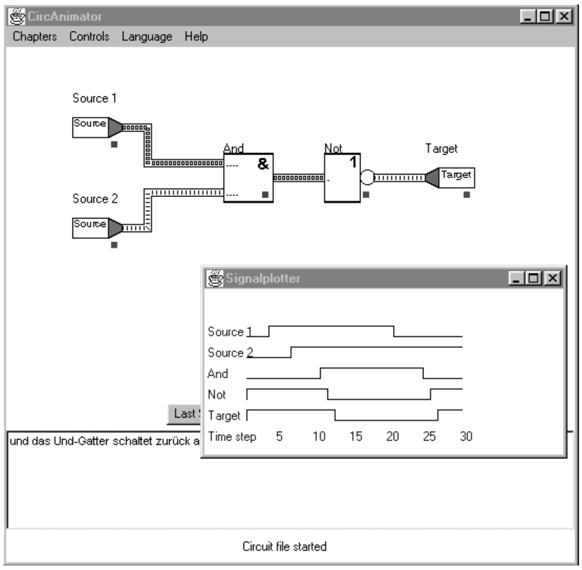


Abb.B.4
Signalplotter

Um ein laufendes Kapitel abzubrechen, kann der Benutzer aus dem “Controls”-*Menu* den Punkt “Stop running chapter” anwählen. Um das ganze Kapitel noch einmal vom Start durchzugehen, ist der Menupunkt “Restart chapter” anzuklicken. Aus dem “Chapters”-Menu schließlich lassen sich wie in Abb.B.5 gezeigt die einzelnen Kapitel direkt starten. Dabei besteht die Möglichkeit, durch Wahl der mit “Test ...” bezeichneten Kapitel mit den in den korrespondierenden Kapiteln benutzten Schaltungen zu experimentieren.

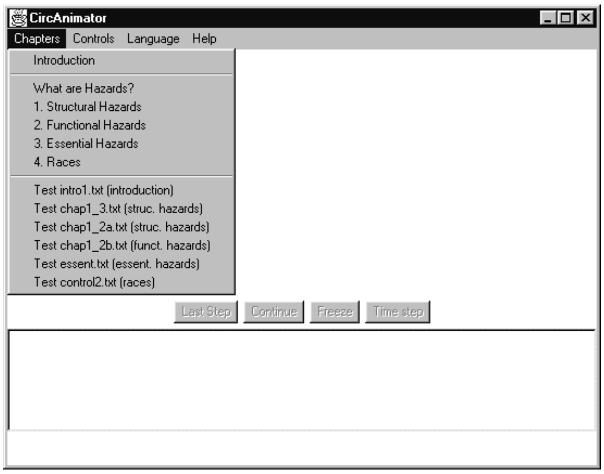


Abb.B.5
Chapters-Menu

Ist der Experimentiermodus eingeschaltet - darauf weist dann einerseits eine Statusanzeige direkt über den Schaltflächen hin, andererseits sollte auch bereits im Text ein Verweis gegeben worden sein - kann man Gatter anklicken. Das selektierte Gatter wird dann mit einem roten Rechteck markiert. Bei Signalgeneratoren kann dann durch wiederholtes Klicken das logische Ausgangssignal gewechselt werden, bei allen anderen Arten von Gattern die Laufzeitverzögerung. Dabei erhöht jeder Rechtsklick und erniedrigt jeder Linksklick die Gatterverzögerung (Abb.B.6).

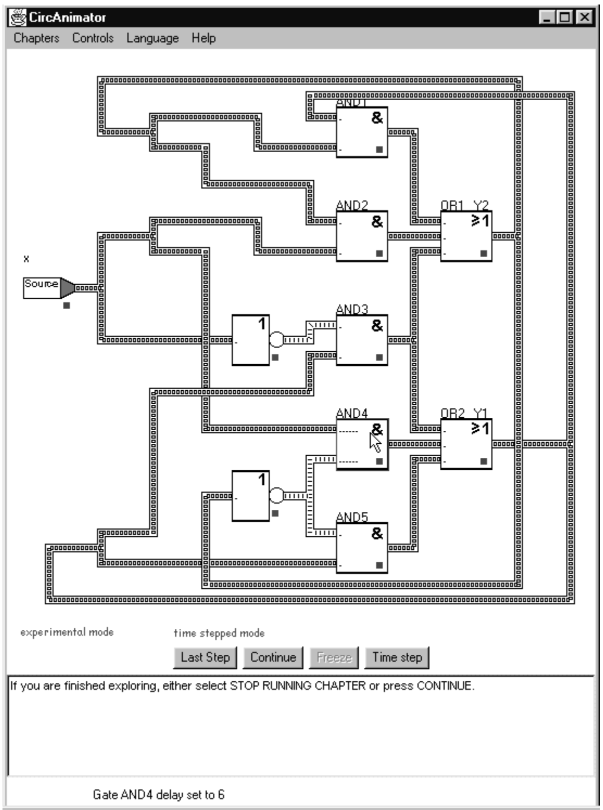


Abb.B.6 Experimentier-Modus

C Analyse

Zur didaktischen sinnvollen Darstellung von Hazards sind Animationsabläufe verständlicherweise besonders geeignet. Dabei müssen einerseits Signalverläufe über der Zeit an bestimmten Punkten im Schaltkreis dargestellt werden können, andererseits ist die Anzeige der Signalpegel direkt auf Leiterbahnen und in Gattern interessant, ermöglicht sie es doch, Signale auf ihren Wegen durch den Schaltkreis zu verfolgen.

Bei einem Erklärungsversuch dieser Laufzeiteffekte sind außerdem Darstellungen wie KV-Diagramme und zugehörige Schaltfunktionsterme beispielsweise zur Darstellung von Strukturhazards hilfreich. Aber auch andere Darstellungen können bei der Erklärung unterstützen. Es muß möglich sein, Signalwechsel zu bestimmten Zeiten an verschiedenen Eingängen im Voraus festzulegen, um gewünschte Effekte zu erzielen (interaktive Simulation).

Um den Unterschied zwischen kritischen und nicht kritischen Hazards, bzw. Races zu verdeutlichen, müssen die Laufzeitverzögerungen für bestimmte Gatter oder Schaltglieder im Netz bis zu einem gewissen Grad wählbar sein.

Es ist also ein Programm zur Visualisierung erwünscht, welches folgende Eigenschaften in sich vereint:

C.1. Flexible Schaltkreisdarstellung

Signalwerte an jedem Gatter und auf jeder Leiterbahn sollen darstellbar sein. Verschiedene Schaltkreise, von einfacheren zur Demonstration von Strukturhazards bis zu komplizierteren für Race-Erscheinungen, müssen vorführbar sein.

C.2. Schaltkreissimulation

Der dargestellte Schaltkreis soll sich logisch korrekt unter einem kleineren Regelsatz verhalten.

C.3. Möglichkeit zum Festlegen bestimmter Signalwechsel an Eingängen

Da Hazards Phänomene sind, die in Reaktion auf Signalwechsel auftreten, müssen die entscheidenen Signalwechsel vorher festlegbar sein, mit anderen Worten wie in einem Drehbuch festgelegt, *geskriptet*, werden können, um dem Lernenden diese Erscheinung vorzuführen.

C.4. Aufzeichnung bestimmter Signalpegel im Netz über der Zeit

Ein über viele Zeitschritte fortgeschriebener Signalverlauf kann mit anderen Verläufen verglichen werden, um einerseits Hazards festzustellen und andererseits die Auswirkungen von Schaltkreismodifikationen zu beobachten.

C.5. Darstellung bestimmter Schaltkreis-Eigenschaften

Hierunter fallen z.B. Darstellungen als KV-Diagramme, Funktionsterme, Funktionstabellen, Zustandscodierungen und Zustandsübergangsgraphen, all jene Merkmale also, die zum Veranschaulichen einzelner Laufzeiteffekte und ihrer möglichen Beseitigung benötigt werden. Dabei sind zur Darstellung von Races andere Mittel notwendig als bei der Visualisierung von Funktionshazards.

C.6. Interaktive Manipulation des Schaltkreises über die Zeit

Neben der vorgefertigten Sequenz von Eingangssignalverläufen bei einer bestimmten Anordnung von Laufzeitverzögerungen soll dem Benutzer gestattet sein, selbst Eingangswerte setzen oder löschen zu können und ferner auch die Wirkung unterschiedlicher Verzögerungen ausprobieren zu können. Ein gewisses Maß an Interaktivität ist also notwendig.

D Design und Implementierung

Die erste große Designentscheidung war im Hinblick auf eine flexible Schaltnetz-Darstellung zu treffen: sollten die Schaltnetze als ein Element in der Software verwirklicht werden? Oder sollte es besser sein, Schaltnetze von der Software zu trennen? Die Entscheidung fiel zugunsten der Trennung aus. Es schien sinnvoll, einen Toolbox-Entwurf zu verfolgen und alle notwendigen (unter Abschnitt B “Analyse” beschriebenen) Elemente durch die Entwicklung entsprechender Java-Klassen bereitzustellen. Dieser Ansatz ermöglichte es, die Implementierung der Animationssoftware von den Inhalten vollständig, auch zeitlich, zu trennen. Da die Zielsetzung des Programms die Funktionalität eines vollständigen Simulators für Schaltkreise verlangt, läßt sich so außerdem die Software zu anderen Zwecken als der reinen Darstellung der Hazardproblematik weiterverwenden.

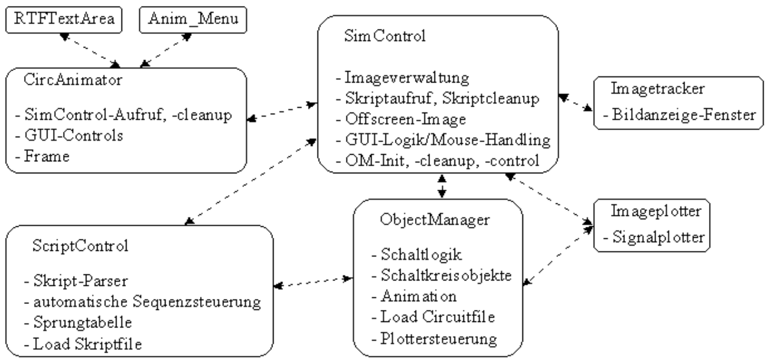


Abb.D.1 Klassen in “CircAnimator”

Die Software wurde mit einer Schnittstelle versehen, um die inhaltlichen Bestandteile aus speziellen Textdateien einlesen und drehbuchartig in bestimmter Reihenfolge ausgeben zu können. Zwei Arten von inhaltlichen Bestandteilen wurden unterschieden: Schaltkreis-Dateien und Animationsablauf-Dateien. Erstere sind Dateien, in denen Schaltkreise in einem speziellen Format beschrieben werden, welches die Software lesen und in einem zusätzlichen Hilfsprogramm, dem Schaltkreis-Editor, auch schreiben kann. Animationsablauf-Dateien, hiernach auch als Skripte bezeichnet, beschreiben in einem eigenen, den Anweisungen eines C-Programm ähnlichem Format die zeitlich getrennten Abläufe zu einem bestimmten Thema. Verschiedene Skripte erlauben so die nach Themen getrennte Darstellung der einzelnen Hazardformen. Diese Dateien lassen sich mit einem gewöhnlichen Texteditor bearbeiten, Änderungen in der Software sind normalerweise nicht notwendig, wenn Ablauf- oder Schaltkreis -Dateien modifiziert werden. Festgeschrieben in der Software sind lediglich die Namen der einzelnen Skripte.

Im folgenden werden nun sowohl die Benutzerschnittstelle als auch der Aufbau des Hauptprogramms “CircAnimator” und daran anschließend des Hilfsprogramms “CircEditor” zur Schaltkreis-Eingabe beschrieben.

D.1. Applikation “CircAnimator”

D.1.1. Überblick

Das Hauptprogramm “CircAnimator” besteht gewöhnlich aus einem Fenster, welches neben einer Menuleiste eine große grafische Anzeige für Schaltkreise und ein kleineres Feld zur Anzeige von Texten enthält. Schließlich gibt es noch einen Bereich am unteren Fensterrand, in welchem Statusmeldungen angezeigt werden.

Beim Ablauf einer Animation auf der Grundlage eines Skriptes können weitere Fenster automatisch geöffnet und geschlossen werden. Alle illustrierenden Grafiken werden genauso wie die Aufzeichnung von Signalabläufen in einem eigenen Fenster dargestellt. Der Benutzer kann diese inhaltlich erläuternden Darstellungen nach Belieben selbst durch Schließen der Fenster vom Bildschirm entfernen.

D.1.2. Benutzerschnittstelle

Es liegt in der Natur der Sache, daß die Schnittstelle zwischen Mensch und Software bei einem Programm, das primär mit graphischen Animationen und Text den Benutzer als “Zuschauer” durch die Zusammenhänge führt, möglichst einfach sein sollte.

Nach dem Programmstart wird automatisch ein Skript “intro1.skr” gestartet, welches eine Einführung in das Programm und die verwendeten Symbole und Terminologie gibt.

Zur Steuerung eines Skriptes gibt es unter dem Menu “Controls” die Menupunkte “Restart Chapter” und “Stop Running Chapter”. Dieses sind Funktionen zum Neustarten des laufenden Skriptes vom Anfang an und zum Beenden eines laufenden Skriptes. Ein laufendes Skript hält an vielen Punkten an, um dem Benutzer Zeit zu geben, die Erläuterungen zu lesen und die Zusammenhänge zu verstehen. Der Benutzer kann das Programm auch selbst innerhalb einer laufenden Demonstrationssequenz durch Wahl des Menupunktes “Freeze”, bzw. Drücken der entsprechenden Schaltfläche, bzw. Buttons, anhalten. Die Fortsetzung kann entsprechend durch Anwahl von “Continue” ausgelöst werden.

Nach dem Ende eines Skriptes, welches dem Benutzer durch eine Meldung in der Statuszeile signalisiert wird, kann ein neues Thema unter dem Menu “Chapters” ausgewählt werden. Das zugehörige Skript wird dann geladen und gestartet.

Das Programm kann jederzeit durch Schließen des Hauptfensters oder Wahl des “Exit”-Menupunktes beendet werden.

Die Anwahl des Menupunktes “Online Help” führt zum Anzeigen eines Textfensters, welches die Zielsetzung und die Grundlagen der Programmsteuerung erläutert, während die Anwahl von “About” unter dem “Help”-Menu ein Fenster mit Copyright- und Autorvermerk anzeigt.

D.1.3. Programmaufbau

Insgesamt liegen dieser Anwendung 11 Java-Klassen zugrunde. Es sind:

- Anim_Menu
- CircAnimator
- CircuitToken
- ImagePlotter
- ImageTracker
- LeadConn

• LeadPart

• ObjectManager

• RTFTextArea

• ScriptControl

• SimControl

Während RTFTextArea, ImageTracker und Anim_Menu keine applikationsspezifischen Aufgaben vornehmen, sondern lediglich zusätzliche GUI-Controls darstellen bzw. in der Java-Architektur notwendige Aufgaben vornehmen (und deshalb in diesem Rahmen auch nicht weiter erwähnenswert sind), erfüllen die anderen Klassen applikationsspezifische Aufgaben in großem (SimControl, ScriptControl, ObjectManager, ImagePlotter) und in kleinem Umfang (CircuitToken, LeadConn, LeadPart).

D.1.3.1.CircAnimator

Gestartet wird die Anwendung durch Aufruf der Frame-Klasse CircAnimator. Diese stellt als Java-Applikation ein eigenständiges Frame-Fenster mit Grafikbereich, Textfeldern, Buttons und Menüsystem zur Verfügung. Die ausgelösten Events werden, wenn anwendungsspezifisch, an entsprechende Methoden in SimControl übergeben.

D.1.3.1.1.Methodenübersicht

- action(): ruft bei bestimmten Events SimControl-Methoden wie startScript(), stopScript(), suspendScript() und resumeScript() auf.
- continueEnable(), disableAll(), disableChapterControls(), disableChapters(), disableLanguages(), disableTimestep() und die entsprechenden enable-Methoden dienen dazu, die Benutzerumgebung entsprechend des Skriptablaufes vernünftig zu steuern. So ist zum Beispiel der Freeze-Button nur aktiv, wenn die Animation in einer Phase ist, wo sie sinnvoll gestoppt werden kann.
- handleEvent(): typische *Frame*-Methode zum Schließen des Hauptfensters

- main(): Start-Methode, welche Instanzen der untergeordneten Klassen bildet. Als Kommandozeilen-Parameter können das voll qualifizierte Verzeichnis der Klassendateien und die *Sleep*-Zeit des Animations*threads* übergeben werden. Diese das ist jene Zeitdauer, in welcher der aktive *Thread* pausiert, um beispielsweise Rechenzeit auch anderen *Threads* und Prozessen im Rechner zur Verfügung zu stellen. Außerdem steuert dieser Parameter auch die Wartezeit zwischen Schritten in einer automatischen Sequenz.

- startIntro(): ruft SimControl.startScript() mit der als Einführungsskript benannten Datei als Parameter auf.

- showStatus(): ermöglicht Applet-ähnliche Status-Meldungen am unteren Rande des Frames. So können abhängige Klassen, die Meldungen an den Benutzer geben wollen, dieses per CircAnimator.showStatus("someText") tun.

- showText(): zeigt in dem von CircAnimator verwalteten TextArea-Control den zugesandten Text mit automatischem Zeilenumbruch an.

D.1.3.2. Drei kleine Hilfsklassen

CircuitToken, LeadConn und Leadpart sind drei kleine Klassen, welche definierte Einheiten sowohl der Simulations- und Animationsfunktionalität bilden.

D.1.3.2.1.CircuitToken

CircuitToken bildet eine Klasse, welche ein Schaltnetz-Gatter darstellt. Das Gatter besitzt hier verschiedene Eigenschaften wie Position auf dem Schaltkreis (posX, posY), eine identifizierende Nummer (seqNum), eine Laufzeitverzögerung zwischen 1 und 10 (specVal) sowie den aktuellen logischen Wert (out). Es existieren die typischen Selektor- und Accessor-Methoden, die auch entsprechend benannt sind: getOut(), getPropDelay(), toggleValue() etc. . Nachfolgend eine Beschreibung der über reine S. und A.- hinausgehenden Methoden:

D.1.3.2.1.1.Methodenübersicht von CircuitToken

- and(): Hilfsmethode, die ein logisches UND zweier Eingabewerte realisiert.

- checkOut(): Ausgangswert (out) auf Grundlage der verschiedenen Eingangssignale ermitteln.
- downDelay(): vermindert die eingestellte Verzögerung (specVal) um eins. Minimalwert ist 1, Maximalwert 10.
- getPendingTimer(int): intern werden für jeden Eingang Members verwaltet, die darüber entscheiden, wann ein neuer Eingabewert lange genug anliegt, um dieses Signal für das Ausgangssignal zu berücksichtigen.
- timeSignal(): zählt die PendingTimer-Variablen der einzelnen Eingänge bis null herab, wonach diese bis dato nicht weiterverarbeiteten Eingangssignale nunmehr zur Ermittlung des Ausgangssignals benutzt werden.
- upDelay(): erhöht die eingestellte Gatterverzögerung um 1, siehe downDelay().
- xor(): realisiert ein logisches Exklusiv-ODER für zwei Eingabewerte.
- or(): realisiert ein logisches ODER für zwei Eingabewerte.

D.1.3.2.2.LeadPart

Die LeadPart-Klasse realisiert ein Leitungssegment allein zur Erstellung der graphischen, u.U. animierten Anzeige. Mehrere LeadPart-Objekte sind als ein LeadConn-Objekt in einem Vektor organisiert. LeadPart-Objekte besitzen einen Typ (type), eine Position (posX, posY), eine Animationsrichtung (normal) und die Identifikations-Nummer der zugehörigen LeadConn, bzw. Leitung. Hauptaufgabe dieser Klasse ist, entsprechend ihres Typs und der Animationsphase in der getImage()-Methode das korrekte, d.h. zur Phase und Typ passende Bitmap in dieser Situation an die aufrufende Methode zurückzugeben.

D.1.3.2.2.1.LeadPart-Methodenübersicht

- getImage(): liefert ein Image aus dem Image-Array von SimControl zurück, in welchem sämtliche Animationsimages verwaltet werden. Dabei müssen Animationsphase und -richtung berücksichtigt werden.

D.1.3.2.3.LeadConn

Die LeadConn-Klasse bildet eine Einheit in der Schaltungssimulationslogik. Wie die Gatter (CircuitToken-Objekte) besitzt jedes LeadConn-Objekt einen logischen Wert. Weiterhin ist jedem Objekt bekannt, von welchem Gatter (start) diese Verbindung stammt und ihren Wert bezieht und zu welchem Gatter (end1) sie führt. Jedes instanzierte LeadConn-Objekt besitzt eine diese Verbindung identifizierende Nummer (ID). Schließlich hat jedes Objekt die Eigenschaft, animiert, d.h. angeschaltet zur Animation, bzw. nicht animiert zu sein (isAnimated()). Die Methoden dieser Klasse bestehen lediglich aus den typischen Selektoren und Accessoren.

D.1.3.3.SimControl

D.1.3.3.1.Funktion

Die Klasse SimControl hat mehrere Aufgaben: sie lädt und verwaltet alle Images, empfängt bestimmte GUI-Events und bearbeitet diese (Tastatur, Maus). Sie generiert zusätzliche Fenster zur Imageanzeige und zur Signalaufzeichnung. Sie instantiiert bei Skriptstart ein ObjectManager-Objekt und räumt dieses auf bei Skript-Ende. Außerdem verwaltet sie das Offscreen-Image, auf dem der OM zeichnet.

D.1.3.3.2.Methodenübersicht

- clearOffscreenImage(): löscht das Offscreen-Image, auf dem der Schaltkreis gezeichnet wird.
- getImage(): liefert die Images für alle Gatter-Teile
- leftOffsetType(): liefert den linken Versatz für das Zeichnen von Leitungsteilen auf dem Gatterraster.
- topOffsetType(): liefert den oberen Versatz für das Zeichnen von Leitungsteilen auf dem Gatterraster.
- pauseDisable(): schaltet den Pause-Button in CircAnimator aus.
- pauseEnable(): schaltet den Pause-Button in CircAnimator ein.

- removeBox(): löscht beim Ausschalten des Experimental-Modus das rote Highlight-Rechteck, welches zum Anzeigen der ausgewählten Cursorposition auf dem Schaltkreisraster verwendet wird.
- resetBoxCoordinates(): setzt die Koordinaten beim Ausschalten des Experimental-Modus zurück.
- run(): lädt alle zur animierten Darstellung eines Schaltkreises benötigten Bilder mit Hilfe der Variablen cdbPath durch eine MediaTracker-Instanz.
- suspendScript(): wird von CircAnimator auf ein Freeze-Event aufgerufen, hält den ScriptController-Thread und damit die weitere Abarbeitung des aktuellen Skriptes an.
- resumeScript(): setzt nach einem Continue-Event die Abarbeitung des ScriptController-*Threads* fort.
- scriptFinish(): schaltet die Kontrollen zur Steuerung, d.h. Restart-, Freeze-Buttons der Kapitel (Skripte) in CircAnimator nach Beendigung, bzw. Abbruch eines Skriptes aus und die Kapitel-Auswahlkontrollen wieder an.
- startScript(String): reinitialisiert das statische ObjectManager-Objekt, instantiiert ein ScriptController-Objekt mit dem übergebenen Dateinamen und startet jenes Objekt.
- stopScript(): bricht nach einem “Stop-current-chapter”-Menu*event* von CircAnimator die Abarbeitung des aktuellen Skriptes ab. Dieses kann nicht durch resumeScript() wieder aufgenommen werden.
- setSpeed(int): ruft in OM setAnimSpeed auf und setzt die *Sleep*-Zeit des Animationsthreads. Standardmäßig sind 100ms als *Sleep*-Zeit eingestellt. Eine andere *Speed*einstellung wird in erster Linie zum *Debugging* von neuen Skripten benutzt.
- showImage(String, String): instantiiert einen ImageTracker, lädt ein Image durch den übergebenen Filenamen und ruft den ImageTracker mit Übergabe des Images und des Fenstertitel-Textes auf.
- showImage2(Image, String): wie showImage(), aber erzeugt ein Duplikat des übergebenen Images. showImage2() wird zur Darstellung der Signalverläufe von Image-Plotter mit ImageTracker verwendet.

• simStatus(String): ruft CircAnimator.showStatus() zur Anzeige einer Statusmeldung im Hauptfenster von CircAnimator auf.

D.1.3.4.ObjectManager

D.1.3.4.1.Funktion

Die ObjectManager-Klasse ist das Kernstück von CircAnimator. Es beinhaltet sowohl die Animations- und Schaltkreis-spezifischen Zeichenroutinen als auch die Logikroutinen und die Simulationsverwaltung. Es lädt und baut intern die Schaltkreise auf unter Zuhilfenahme der mit dem Editorprogramm “CircEditor” erstellten Circuit-Dateien.

Im Zusammenhang mit der Zeitverwaltung werden im Time-Stepped-Modus auch für jene Gatter, die im ImagePlotter registriert wurden, die aktuellen Signale an das ImagePlotter-Objekt gesendet.

D.1.3.4.2.Animationsmethode

Um den Eindruck von in einer Leitung fließenden Nullen oder Einsen zu erreichen, werden pro Sekunde 4 Mal jeweils für jedes Leitungssegment Bitmaps der aktuellenAnimationsphase (0-3) entsprechend auf das Offscreen-Image gezeichnet und danach sofort ein repaint() ausgelöst. Die Animationsphasen werden in ObjectManager.run() bestimmt und synchron mit der Logikverwaltung geführt, indem nach jedem kompletten Zyklus ein Logikcheck durchgeführt wird. Da es notwendig ist, für eine weiche Animation alle Bitmaps im Speicher zu halten, sind die Bitmaps in einem 16-Farben-Gif-Format gespeichert. Dadurch ist die durchschnittliche Größe etwa 350 bytes pro Bild. Insgesamt gibt es 163 Animationsbitmaps, die zusammen 38 Kbyte Speicher belegen.

Der Schaltkreis wird auf einem max. 12x12 großen Raster dargestellt, jedes Element hat dabei höchstens die Abmessungen 48x48 pixel. Die Applikation "CircEd" ermittelt beim Abspeichern, ob die Richtung einer Leitung verlangt, daß ein Leitungssegment normal animiert, also bitmap0, bitmap1, bitmap2, bitmap3 hintereinander dargestellt werden soll, oder in einer entgegengesetzten Richtung. Dabei gibt es in der Gegenrichtung zwei verschiedene Sequenzen, einmal bitmap1, bitmap0, bitmap3, bitmap2 und das andere Mal bitmap3, bitmap2, bitmap1, bitmap0.

In der run()-Methode wird jeweils zwischen dem Umschalten der Phase und der repaint()-Aufforderung ein sleep() veranlaßt, die defaultmäßige Länge ist dabei 100ms, also 0,1 sec. Dieser Wert kann aber durch eine als zweiten Kommandozeilen-Parameter übergebene Zahl geändert werden.

D.1.3.4.3.Simulationsmodell der Schaltkreislogik

Bei der Erstellung eines Schaltkreises durch den Editor wird eine Textdatei angelegt, in der dieser durch drei Gruppen, Gatter, Leitungen und Leitungssegmente, beschrieben wird. In gleicher Form wird der Schaltkreis nach dem Einlesen auch in der ObjectManager-Instanz gespeichert. Dabei werden die Gatter in dem Vektor circGates, die Leitungen im Vektor leadConns und die Leitungsteile im Vektor leadParts als Sammlungen von CircuitToken-, LeadConn- und LeadPart-Instanzen zusammengefasst.

Das Gatter ist dabei die Einheit, welche die wesentliche Information trägt. Die CircuitToken-Klasse kennt dabei neben den Informationen über ID, Typ und Position auch *Member*variablen jedes Gatters für Verzögerungswert (specVal), aktuelles Ausgangssignal (out), wirkende logische Eingänge (in1, in2, in3), tatsächlich durch eine Leitung angeschlossene Eingänge (in1A, in2A, in3A), darüberhinaus auch, wie viele Zeitschritte es noch dauert, bis ein Signalwechsel auf einem Eingang auch seine logische Wirkung entfaltet (pendingTimer1 - 3). Ein Signalwechsel wird dabei durch die Flags pendingIn1 - 3 signalisiert.

Der Ablauf im Gatter bei der Logikermittlung ist wie folgt:

Die logicCheck()-Methode ruft die setInVal()-Methode eines Gatters auf. Falls das Ausgangsignal vom neuen Wert differiert, wird dort in Abhängigkeit von wirkender Zeitbetriebsart, also Instantan-Modus oder Zeitschritt-Modus, entweder der Ausgang verändert, oder das pendingIn-Flag eines Eingangs gesetzt und der Timer für diesen Eingang pendingTimer auf den eingestellten Verzögerungswert (specVal) des Gatters gesetzt. Im Instantan-Modus werden also die eingestellten Verzögerungswerte der Gatter nicht berücksichtigt. Signaländerungen an Ausgängen sollen so schnell wie möglich an nachfolgende Gatter weitergegeben werden, deswegen auch die Bezeichnung als *instantan*. Im Zeitschritt-Modus hingegen werden Verzögerungen genau berücksichtigt. Nur so, d.h. durch gezielte Manipulation (Verkürzung und Verlängerung) der Signalwege lassen sich Hazard-Situationen erzeugen und demonstrieren.

Aus dem run()-Thread des ObjectManager wird nach jedem kompletten Zyklus ein logicCheck() aufgerufen, welches als erstes alle Objekte im circGates-Vektor untersucht. Dazu schickt es der Reihe nach jedem Gatter eine timeSignal()- und checkOut()-Mitteilung. timeSignal() bewirkt, daß jeder pendingTimer eines Eingangs, der größer null ist, um eines reduziert wird. Wird dabei null erreicht, wird das Signal nunmehr als wirkende Eingang betrachtet, indem der Eingang (in0, in1, in2) das Signal am pendingIn-Eingang erhält. Das darauffolgende checkOut() läßt das CircuitToken-Objekt anhand seines Typs das Ausgangssignal berechnen. In einer Switch-Anweisung verzweigt das Programm so zu den Routinen für AND, OR, NAND etc. und führt dann diese Berechnung für die Zahl der anliegenden Eingänge (numIns) durch.

Nachdem die ObjectManager.logicCheck()-Methode sichergestellt hat, daß die Verzögerung in den Gattern reduziert und Signalwechsel bearbeitet wurden, wird in einem zweiten Schritt auch der leadConns-Vektor Leitung für Leitung durchgegangen. Bei jeder Leitung, also einem leadConn-Objekt, wird ermittelt, woher sie ihr Signal erhält. Dann wird geprüft, ob das aktuelle Signal der Leitung, LeadConn.getCurrentVal(), dem aktuellen Ausgangssignal des Gatters, CircuitToken.getOut() entspricht. Falls nicht, wird der Wert des Leitung entsprechend geändert und das Gatter, zu welchem die Leitung führt, über den Signalwechsel durch ein CircuitToken.setInVal() benachrichtigt. Das benachrichtigte Gatter indessen kann den Wechsel erst beim nächsten Aufruf von logicCheck() bearbeiten.

D.1.3.4.4.Methodenübersicht

- animateAllLeads(boolean): wird von ScriptControl-Befehlen 28 und 29 aufgerufen. Bewirkt das An-, bzw. Ausschalten der animierten Leitungen.
- animateLead(int, boolean): wird von ScriptControl-Befehlen 30 und 31 aufgerufen. Bewirkt das An-, bzw. Ausschalten der Animation jener durch den Integerwert bezeichneten Leitung.
- loadCircuitFile(String): wird von ScriptControl-Befehl 2 aufgerufen. Die bezeichnete Schaltkreisdatei wird geladen, die Gatter, Leitungen und Segmente im OM registriert, der Schaltkreis dann auf dem Bildschirm in gestopptem Zustand nichtanimiert dargestellt.
- start(): wird von ScriptControl-Befehl 3 aufgerufen. Animation und Logikengine werden über den run()-*Thread* in Betrieb gesetzt.
- stop(): wird von ScriptControl-Befehl 4 aufgerufen. Der *Thread* wird angehalten, die Animation ausgeschaltet, der Schaltkreis in neutraler Darstellung wiedergegeben.
- clearCircuit(): wird von ScriptControl-Befehl 12 aufgerufen. Löst das Löschen des Offscreen-Images aus und entfernt alle Bestandteile des bisher geladenen Schaltkreises aus dem OM.
- registerValues(Strings): wird von ScriptControl-Befehl 25 aufgerufen. Im Vektor circGates, wo alle Gatter registriert sind, wird geprüft, ob die übergebene Liste von Namen korrekte Alias-Bezeichnung von Gattern sind. Wenn ja, wird ein ImagePlotter instantiiert und diesem die Namensliste zur Registrierung übergeben.
- showPlotterImage(): wird von ScriptControl-Befehl 26 aufgerufen. Erstellt vom augenblicklichen Plotter-Verlauf ein Bitmap und übergibt dieses mittels SimControl.showImage2() an einen neuen ImageTracker.
- removePlotter(): wird von ScriptControl-Befehl 27 aufgerufen. Löscht das ImagePlotter-Objekt.
- setExperimentMode(boolean): wird von ScriptControl-Befehl 16 und 17 aufgerufen. Setzt oder löscht den Experimentier-Modus, in welchem der Benutzer Gatter-Werte und *Delays* durch Mausklicke setzen und Zeitsignale zur Steuerung der Simulation mit dem *TimeStep*-Button senden kann.

- setRecordingMode(boolean): wird von ScriptControl-Befehl 23 und 24 aufgerufen. Setzt und löscht den Recording-Modus, wo für alle im Plotter registrierten Gatter pro Zeitschritt die aktuellen Werte an ImagePlotter mittels ImagePlotter.timeSignal(Vector) geschickt werden.
- setInstantan(boolean): wird von ScriptControl-Befehl 14 und 15 aufgerufen. Schaltet den Instantan-Modus ein oder aus. Wenn eingeschaltet, berücksichtigt die Simulation etwaige Gatter-Verzögerungen nicht mehr, sondern gibt alle unverzüglich alle Änderungen an den Eingängen eines Gatter an den Ausgang und die damit verbundenen Gatter weiter. Im Instantan-Modus ist kein Recording-Modus möglich, da keine geregelte Zeit zur Logikermittlung herangezogen wird und deswegen auch etwaige eingestellte Verzögerungen an den Gattern nicht berücksichtigt werden.
- setGate(int, int, int): wird von ScriptControl-Befehl 13 aufgerufen. Setzt für das mit seiner ID-Nr. gekennzeichnete Gatter einen neuen Ausgangswert und einen neuen Verzögerungswert (min. 1, max. 10).
- setGate(String, int, int): wie setGate(int, int. int), nur wird hier das Gatter durch seinen Alias-Namen identifiziert.
- setShowAliasMode(boolean): wird von ScriptControl-Befehl 18 und 19 aufgerufen. Schaltet die Anzeige der Alias-Bezeichnungen von Gattern ein oder aus.
- init(): Löscht alle etwaigen vorherigen Daten der statischen OM-Instanz und schaltet *default*mäßig Instantan-Modus und ShowAlias-Modus ein.
- checkAndToggle(int, int, boolean): wird im Experimentier-Modus aus SimControl.MouseDown() aufgerufen. Sucht an den übergebenen Koordinaten nach einem editierbaren Gatter und invertiert bei einer Signalquelle das Ausgangssignal oder erhöht, bzw. erniedrigt in Abhängigkeit des dritten Parameters den Verzögerungswert für das betreffende Gatter (min.1, max.10).
- drawCircuit(): wird von SimControl.update() aufgerufen, zeichnet alle Gatter und Verzögerungsboxen auf das Offscreen-Image.
- getImage(): erfragte vom ImagePlotter das aktuell angezeigte Bild.

- logicCheck(): wird entweder von der Zeitkontrolle von ScriptControl oder aus dem run-Thread von OM aufgerufen. Geht alle Gatter durch und fordert diese auf, die interne Logik der Gatter zu verarbeiten. Danach werden alle Verbindungen daraufhin getestet, ob ihr aktueller Wert mit dem Quellgatter-Ausgangssignal übereinstimmt. Wenn nicht, wird ihr Wert entsprechend geändert.
- logicInit(): wie logicCheck(), nur wird ein zusätzlicher Prüflauf der Leitungssignale vor der Logikverarbeitung der Gatter durchgeführt.
- paintLead(): wird von SimControl.update() aufgerufen und zeichnet die Leitungen bestehend aus allen Segmenten (LeadPart-Instanzen) auf das Offscreen-Image. Dabei wird für jedes Segment in Abhängigkeit seines Typs, seiner eingestellten Animationsrichtung das notwenige Image ermittelt, welches dann von SimControl.getImage() angefordert wird.

D.1.3.5.ScriptControl

D.1.3.5.1.Funktion

Diese Klasse steuert die Skriptverarbeitung. Das bei der von SimControl ausgelösten Instantiierung bezeichnete Skript wird geladen und Befehl für Befehl geparst und abgearbeitet. Die Aufforderung zur Abarbeitung wird dabei aus dem run()-*Thread* von ScriptControl geschickt. Je nach Befehl wird nach Aufruf der zum Befehl zugehörenden Methode angehalten und der Continue-Button freigeschaltet oder der nächste Befehl abgearbeitet.

Bei Bedarf kann aus dem Skript heraus eine automatische Sequenz gestartet werden, die Befehle zu bestimmten Zeitpunkten ausführt, ohne daß der Benutzer mit dem Continue-Button seine Aufforderung zur Fortsetzung geben muß.

Neben Kommentaren in C++-Form durch '//' kennt die Skriptsprache 32 verschiedene Befehle:

D.1.3.5.2.Skriptbefehlsübersicht

- *showText(String)*: zeigt den übergebenen Text auf der Anzeige des Textfeldes von CircAnimator an. Das Textfeld bricht automatisch um. Eigene Absätze können mit '\n' bewirkt werden. Außerdem wird das Markieren von Wörten mit Hilfe von '<B>' zum Anschalten und '<-B>' zum Ausschalten unterstützt. Wegen nicht ganz fehlerfreier Funktion der Java-Parserklassen, speziell die Kommentarbehandlung von java.io.StreamTokenizer, dürfen weder normale *Slashes* '/' noch *Backslashes* '\' im Skript verwendet werden.
- *showStatus(String)*: zeigt einen kurzen übergebenen Text aus wenigen Worten in der Statuszeile von CircAnimator an.
- *loadCircuitFile(String)*: veranlaßt das Laden der unqualifiziert bezeichneten Schaltkreisdatei aus dem *Homedirectory* von CircAnimator. Der Schaltkreis wird dann in neutralem Zustand auf dem Bildschirm dargestellt.
- *startCircuit()*: startet die Animation und Logikverarbeitung des Schaltkreises.
- *stopCircuit()*: stoppt Animation und Logikverarbeitung.
- *setGate(int ID, int Wert, int Verzögerung)*: weist dem mit seiner ID (1. Parameter) bezeichneten Gatter einen bestimmten Ausgangswert und eine neue Verzögerung (min.1, max.10) zu.
- *setGateAlias(String Alias, int Wert, int Verzögerung)*: wie *setGate()*, nur wird das gewünschte Gatter durch seinen Alias-Namen bezeichnet.
- *beginSequence()*: veranlaßt ScriptControl, bis zum Erhalt von "*endSequence*" alle Befehle in eine Queue zu stellen, welche dann auf den Skript-Befehl "*startAutoTimer*" hin abgearbeitet wird.
- *atTime(int)*: weißt die ScriptControl-Verarbeitung an, alle bis zum nächsten "*atTime*" folgenden Befehle innerhalb einer Sequenz zum angegeben Zeitpunkt auszuführen.
- *endSequence()*: veranlaßt ScriptControl, nachfolgende Befehle wieder normal abzuarbeiten.
- *startAutoTimer()*: veranlaßt die automatische Abarbeitung der vorher durch *beginSequence()* und *endSequence()* registrierten Befehlskette.

- *manualTimeStep()*: bewirkt einen Logikcheck in OM und ein anschließendes repaint().
- *showBitmap(String)*: zeigt mit Hilfe von ImageTracker die unqualifiziert übergebene Bilddatei in einem eigenen Fenster an.
- *pauseUntilContinue()*: hält ohne weitere Aktion die Abarbeitung der Skriptdatei an. Durch ein Continue-*Event* wird die Verarbeitung fortgesetzt.
- *clearScreen()*: deregistriert den akutell in OM geladenen Schaltkreis und löscht das grafische Anzeigefenster, indem OM.clearCircuit() gerufen wird.
- *setInstantan()*: schaltet den Instantan-Modus ein. Etwaige Gatter-Verzögerungen werden nicht mehr berücksichtigt.
- *setTimeStepped()*: schaltet den Instantan-Modus aus. Gatter-Verzögerungen werden wieder bei der Logikverarbeitung berücksichtigt. Auch das Aufzeichnen von Werten durch ImagePlotter ist wieder möglich.
- *setExperimentModeOn()*: schaltet den Experimentier-Modus ein. Durch *Anklicken* zum Auswählen eines Gatters und weiteren Links- oder Rechtsklicken können Werte invertiert, bzw. Verzögerungen erhöht oder vermindert werden. Das aktuell selektierte Gatter ist durch ein rotes, auffälliges Rechteck markiert.
- *setExperimentModeOff()*: schaltet den Experimentier-Modus wieder aus.
- *showAliasOn()*: schaltet die Anzeige der Alias-Namen aller Gatter ein.
- *showAliasOff()*: schaltet die Anzeige der Alias-Namen aller Gatter aus.
- *registerValues(Strings)*: veranlaßt die Anzeige des ImagePlotters und registriert die mit ihren Alias-Namen bezeichneten Gatter im Plotter. Für diese Gatter werden im Recording-Modus zu jedem Zeitpunkt ihre aktuellen Werte übergeben.
- *startRecording()*: schaltet den Recording-Modus ein. Zu beobachtende Gatter sollten vorher mit *registerValues()* bezeichnet worden sein.
- *stopRecording()*: schaltet den Recording-Modus aus.
- *showPlotterImage()*: erzeugt aus der momentanen Anzeige des ImagePlotter ein ImageTracker-Objekt, so daß der ImagePlotter für eine weitere Aufzeichnung von Werten eines anderen Schaltkreises verwendet werden kann.

- *removePlotter()*: entfernt das ImagePlotter-Objekt vom Bildschirm.

D.1.3.5.3.Implementierungshinweise

Die meisten Methoden dieser Klasse werden aus der Methode ScriptControl.parseNextStatement(), bzw. der entsprechenden Methode ScriptControl.selectNextStatement() bei aktiven Autotimer-Modus innerhalb einer Sequenz aufgerufen. Der *Vektor* ScriptControl.funcNames mit allen bekannten Befehlsnamen wird durchsucht. Nach dem Erkennen des Befehlsnamen wird in einem *Switch-Statement* in die laut Tabelle zugehörige Methode verzweigt. Deshalb tragen diese Methoden, die jeweils genau einer Skriptanweisung entsprechen, Bezeichnung der Art "statement0()", "statement1()" etc. je nach ihrer Position in der Befehlstabelle.

Die anderen Methoden neben den üblichen Selektoren und Accessoren sind:

D.1.3.5.4.Methodenübersicht

- init(): setzt interne Timer und Zeit zurück, defaultmäßig werden Autotimer-Modus ausgeschaltet und das inSequence-Flag gelöscht.

- openScriptFile(String): öffnet das mit unqualifiziert Namen bezeichnete File und stellt in der verwendeten Parserklasse notwendige Variablen zum korrekten Verarbeiten der Datei ein.

- parseNextStatement(): veranlaßt das Einlesen des nächsten Skript-Befehls, d.h. alles bis zum nächsten Semicolon. Der ermittelte Befehlsname wird gegen eine Tabelle (Vektor funcNames) geprüft, in der alle gültigen Statements verzeichnet sind. Dann wird die zu diesem Befehl gehörende Routine (ScriptControl.statement0() bis ScriptControl.statement31()) ausgeführt. In der dabei ausgeführten Switch-Verzweigung ist angegeben, ob die Weiterverarbeitung nach diesem Befehl automatisch fortgesetzt oder bis zur Continue-Aufforderung angehalten werden soll.

- selectNextStatement(): wie parseNextStatement(), allerdings wird diese Methode nur im Autotimer-Modus ausgeführt, da das automatische Ausführen und Beachten der "*atTime()*"-Anweisungen besonderer Beachtung bedarf. Auch wird bei weit mehr Befehlen die Verarbeitung fortgesetzt als bei parseNextStatement().

- suspend(): durch ein *Freeze-Event* kann die Abarbeitung auch innerhalb einer Sequenz gestoppt werden, d.h. auch die Simulationsuhr wird gestoppt.
- resume(): löst die Weiterbearbeitung des Skriptes und das Wiederanlaufen der Simulationsuhr aus.

D.2. Applikation “CircEd”

D.2.1. Funktion

Die Java-Applikation “CircEd” dient dazu, Schaltkreise zu entwerfen und in dem Format abzuspeichern, welches “CircAnimator” lesen kann. Die manuelle Eingabe per Texteditor ost zwar möglich, aber wegen der zu erwartenden Fehleranfälligkeit nicht zu empfehlen.

“CircEd” besteht wie “CircAnimator” aus einer *Frame*-Klasse, CircAnimator2, der eigentlichen Hauptklasse circEditorApplet und den Hilfsklassen CircuitToken, LeadConn, Leadpart sowie DialogLayout und PropDialog. Die ersten drei Hilfsklassen sind fast identisch mit jenen von “CircAnimator”, die zwei Dialog-Klassen bringen lediglich einen Dialog zur Werteeingabe auf den Bildschirm.

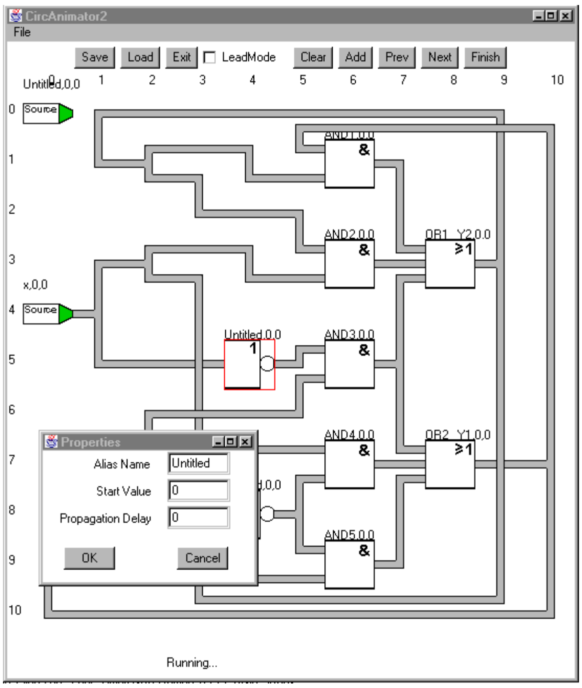


Abb.D.2 Applikation “CircEd”

D.2.2. Userinterface

“CircEd” wird durch Aufruf von der *Frame*-Klasse CircAnimator2 gestartet. Dabei wird ein Fenster aufgebaut, welches ein Koordinatenraster an den Rändern anzeigt und am oberen Rand eine Reihe von Buttons besitzt.

Auf der Rasterfläche wird durch ein rotes Rechteck das aktuelle gewählte Rasterfeld bezeichnet. Hier kann durch Links- und Rechtsklicke durch die Liste der verfügbaren Gatter geblättert werden. Ist ein Gatter eingestellt, kann ein Dialog zum Eingeben einer Alias-Bezeichnung, des Defaultwertes und der Defaultverzögerung aufgerufen werden. Nach dem Schließen dieses Dialoges werden dann die gesetzten Werte über dem aktuellen Gatter angezeigt.

Nachdem alle Gatter gesetzt wurden, kann durch Anwahl der Checkbox “LeadMode” die Betriebsart zur Eingabe der Leitungen angewählt werden.

Sind bisher keine Leitungen eingegeben worden, schaltet das Programm nach Anwahl der Checkbox automatisch in die Funktion zur Aufnahme einer neuen Leitung. Eine Statusmitteilung am unteren Rand des Fensters bestätigt, daß man sich im Modus zur Eingabe neuer Leitungen befindet.

Für eine Leitung sind Anfangs- und Endgatter zu markieren. Dabei wird zuerst das Feld des Startgatters ausgewählt und dann die Taste ‘s’ gedrückt. Auf die gleiche Weise wird das Endgatter der neuen Leitung selektiert und mit ‘e’ markiert. Über den beiden Gattern erscheint dann “Start”, resp. “Ende”. Danach kann der gewünschte Verlauf der Leitung durch Links- und Rechtsklicke der entsprechenden Rasterfelder gewählt werden. Es ist notwendig, daß vom Startgatter bis zum Endgatter eine geschlossene Verbindung markiert wird. Andere nicht verbundene Raster sollten keine Leitungsstücke enthalten. Ist erwünscht, daß die Leitung welche gerade eingegeben wird, defaultmäßig nicht animiert sein soll, kann durch Drücken der ‘a’-Taste zwischen nicht-animierter und animierter Leitung hin- und hergeschaltet werden. Über dem Startgatter gibt ein entsprechender Text darüber Auskunft.

Um die Leitung zu vollenden, ist der Finish-Button anzuwählen. Hat man keine Fehler gemacht, meldet das Programm "LeadConn hinzugefügt". Um die nächste Verbindung einzugeben, muß der Add-Button angeklickt werden. Befindet man sich im normalen Lead-Modus, können mit dem Next- und dem Prev-Button alle bisher eingegebenen Leitungen durchgeblättert werden. Soll eine Leitung gelöscht werden, ist auf den Clear-Button zu drücken.

Wenn alle Gatter und Leitungen eingegeben und wahlweise benannt wurden, kann diese Schaltung als Schaltkreisdatei durch Anwahl des Save-Button gesichert werden. Unter Windows bietet sich an, als Namens-Erweiterung ".txt" zu wählen. Wurde die Schaltung korrekt gesichert, erscheint in der Standardausgabe der gewählte Dateiname. Erscheint hingegen eine Fehlermeldung, liegt entweder ein Problem mit dem Dateinamen oder falschen Leitungen vor.

Soll eine bestehende Schaltung geändert werden, kann nach dem Starten auch durch Anwahl des Load-Buttons eine Schaltkreisdatei geladen werden.

D.2.3. Methodenüberblick

Neben den üblichen Zugriffs- und Auswahlmethoden finden sich folgende speziellere Methoden in circEditorApplet:

- action(): bearbeitet sämtliche von den Buttons ausgelösten Events: save, load, Exit, außerdem bei eingeschaltetem Lead-Modus Clear, Add, Previous Lead und Next Lead.
- closeDialog(): Hilfsfunktion, um den Property-Dialog nach Drücken des Ok-Buttons zu schließen, bzw. zu verbergen.
- createFDDialog(): Hilfsfunktion, um den FileSelect-Dialog aufzubauen.
- createPropertyDialog(): erstellt einen *Frame*, in welchem dann die Dialog-Klasse instantiiert wird.
- getImage(int): liefert das zu dem übergeben Typ zugehörige Image eines Gatters (Typ zwischen 0 und 8), bzw. Leitungssegment (Typ zwischen 11 und 28) zurück.
- getMarkers(): dient im Lead-Modus dazu, die Start- und Endmarkierungen der Gatter der aktuellen Verbindung als Text an die Gatter zu schreiben.

- keyDown(): behandelt die *Keyboard-Events*, speziell die Tasten ‘s’, ‘e’, ‘p’ und ‘a’, welche Gatter markieren, den Property-Dialog aufrufen oder den Animationsstatus der Leitung einstellen, die gerade eingegeben wird.
- leftOffsetType(int): gibt für den übergebenen Typ den linken Versatz zur korrekten Darstellung auf dem Schaltraster an.
- topOffsetType(int): gibt für den übergebenen Typ den oberen Versatz zur korrekten Darstellung auf dem Schaltraster an.
- mouseDown(): ermittelt bei Mausklick die Rasterkoordinaten. Wenn festgestellt wird, daß sich ein Gatter, bzw. bei aktivem AddLead-Modus kein Gatter an diesen Koordinaten befindet, wird das nachfolgende oder vorhergehende Gatter, bzw. Leitungssegment ausgewählt und dargestellt.
- paint(): geht den *Array* gateArray durch und zeichnet an den angegebenen Positionen Gatter und Leitungssegmente. Im LeadModus werden nur Leitungssegment der aktuell ausgewählten Leitung gezeichnet.
- resetGateArray(): löscht den *Array* gateArray, in welchem vor dem Speichern eines Schaltkreise Gatter und teilweise Leitungssegmente hinterlegt werden.
- resetMarkers(): löscht Start- und Endbezeichner. Wird verwendet, wenn der Lead--Modus ausgeschaltet wird.
- selectNextGate(int, int, boolean): Stellt fest, welcher Typ Gatter sich an den angegebenen Koordinaten befindet und wählt in Abhängigkeit des übergeben 3. Parameters entweder den nachfolgenden oder vorausgehenden Typ an.
- selectNextLead(int, int, boolean): Wird im AddLead-Modus aufgerufen. Stellt fest, welcher Typ Leitungssegment sich an den angegebenen Koordinaten befindet und wählt in Abhängigkeit des übergeben 3. Parameters entweder den nachfolgenden oder vorausgehenden Typ an.
- setGateValues(String, int, int): Setzt nach Drücken des Ok-Buttons im Property-Dialog die eben eingegebenen Werte für das selektierte Gatter.
- simStatus(String): zeigt im unteren Fensterrand des durch CircAnimator2 aufgespannten *Frames* eine Statusmeldung mit übergebenem Text an.

- run(): lädt bei nach Initialisierung von circEditorApplet und start() mit Hilfe eines MediaTrackers die zur Darstellung notwendigen Gatter und Leitungsteil-Bitmaps.

E Anhänge

E.1. Auszug aus einer Skriptdatei

Im folgenden ist die Skriptdatei chap4.skr in Auszügen mit einigen Kommentaren versehen aufgelistet:

// chap4.skr, Races
// ACHTUNG, ACHTUNG, ACHTUNG: Keine slashes oder backslashes außer für Zeilenvorschub verwenden, diese bringen den Parser aus dem Trab!!!!!
//

Kommentare werden mit zwei Schrägstrichen in C++-Manier markiert.

showStatus("Chapter 4 running ...");
showText("<B>Races 1 von 36<-B>\nBei Races oder Wettläufen handelt es sich um Situationen innerhalb eines Schaltwerkes, wo im Verlauf eines Übergangs von einem in einen anderen Zustand mehr als nur eine Zustandsvariable wechselt, bei diesem Wechsel also diese Zustandsvariablen einen Wettlauf miteinander führen, welche Variable als erste, zweite etc. den neuen Wert erreicht.");

Jeder andere Befehl endet mit einem Semicolon.

showText("<B>Races 2 von 36<-B>\nWenn das korrekte Verhalten im Schaltkreis allein davon abhängt, ob die beteiligten Zustandsvariablen ihren Wert in einer bestimmten Reihenfolge ändern, spricht man von einem <B>kritischen Wettlauf<-B>.");

Der `showText`-Befehl nimmt als Argument einen durch Anführungszeichen begrenzten String. Durch Formatiersteuerzeichen wie '\n' kann ein Zeilenumbruch veranlaßt werden, durch '<B>' und '<-B>' ein Wort fett markiert werden.

showText("<B>Races 3 von 36<-B>\nWenn es hingegen egal ist, in welcher Reihenfolge die Variablen wechseln und der Schaltkreis dennoch im gewünschten Zustand ankommt, nennt man dieses einen <B>unkritischen Wettlauf<-B>.");
showText("<B>Races 4 von 36<-B>\n Betrachten wir hierzu erst einmal einige Diagramme:");
showBitmap("race10.JPG","Abb.4.0");

Der `showBitmap`-Befehl zeigt ein Bild in einem eigenen Fenster mit dem als zweiten Parameter übegeben Titel.

showText("<B>Races 5 von 36<-B>\nBild 4.0 zeigt eine (minimierte) Flußmatrix eines kritischen Wettlaufs. Diese Matrix bezeichnet die stabilen Zustände eines Schaltwerks (rot unterlegte Ziffern) und den Ausgabewert <B>z<-B> in diesem Zustand.");	
'showText("<B>Races 6 von 36<-B>\n<B>y1<-B> und <B>y2<-B> bezeichnen die Zustandsvariablen.<B>x1<-B> und <B>x2<-B> sind die Eingänge.);	
showBitmapOnscreen("race10ct.JPG");	*`showBitmapOnscreen` zeigt ein Bild im Hauptfenster von CircAnimator an.*
showText("<B>Races 7 von 36<-B>\nOben sei das Schaltwerk als Schaltkreis angegegeben:");	
loadCircuitFile("control2.txt");	*Hier wird eine Schaltkreisdatei eingelesen.*
showText("<B>Races 30 von 36<-B>\nWegen der Rückkopplungen muß erst einmal der gewünschte Zustand <B>xC=01 y2y1=11<-B>, erreicht werden, ab dort schaltet sich der Signalplotter ein!");	
beginSequence();	*Hier wird der Blockanfang einer automatischen Sequenz markiert.*
atTime(1);	*Ein `atTime(x)` bedeutet, daß alle bis zum nächsten `atTime(y)` folgenden Befehle zu einer Zeit ausgeführt werden müssen.*
startCircuit();	*Startet die Simulations- und Logikengine für den durch `loadCircuitFile` eingelesenen Schaltkreis.*
setGateAlias("x",1,0");	
setGateAlias("C",1,0");	
atTime(3);	
setGateAlias("x",0,1);	*Dieser Befehl weist dem durch den Alias x bezeichneten Gatter den Ausgangswert null und die Standardverzögerung von 1ZE zu.*
atTime(10);	
showText("Wie man an den nicht vorhandenen Verzögerungen sieht, ist das System jetzt stabil, also los...");	
pauseUntilContinue();	*Veranlaßt die Sequenz, bis zum Drücken von Continue anzuhalten.*
atTime(11);	
registerValues("x","C","A1","A2","A3","Y1","Y2");	*Bereitet den Einsatz des Signalplotters vor. Jene Gatter, deren Werte aufgezeichnet werden sollen, weden als Argumente von `registerValues` übergeben.*
startRecording();	*Löst die Anzeige des Signalplotter-Fensters und den Start der Werteaufzeichnung aus.*

atTime(14);
showText(“Jetzt wechselt C von 1 auf 0...”);
atTime(16);
setGateAlias(“C”,0,0);
atTime(17);
showText(“ “);
atTime(32);	

stopRecording();	*Die Aufzeichnung wird beendet.*
showPlotterImage(“Signalverlauf 4a”);	*Das Plotterfenster wird geschlossen, die graphische Aufzeichnung als eigenes Fenster mit dem als Parameter übergebenen Titel angezeigt.*
atTime(33);	
stopTimer();	`stopTimer` *hält die Uhr an, welche den Ablauf der Sequenz steuert.*
stopCircuit();	*Die Simulations- und Logikengine wird angehalten, der Schaltkreis in neutraler Darstellung angezeigt.*
endSequence();	*Dieses Schlüsselwort kennzeichnet das Ende des Sequenzblocks.*
startAutoTimer();	*Hierdurch wird nun das Abspielen des Sequenzblocks gestartet.*
showText(“<B>Races 31 von 36<-B>\nWenn nun die verwendeten <B>UND<-B>-Gatter unterschiedliche Verzögerungen hätten, z.B. A3 als ein langsameres Gate von 4ZE realisiert ist, während die Gatter A1 und A2 weiter die Verzögerung von 2ZE haben, können wir einen Fehlverlauf wegen des hier vorliegenden kritischen Wettlaufs betrachten:”);	

E.2. Beschreibung einer Schaltkreisdatei

Nachfolgend ist die Schaltkreisdatei rsff.txt mit Kommentaren versehen abgedruckt. Bei dem darin beschriebenen Schaltkreis handelt es sich um ein RS-Flipflop.

Gatter:	*Das Schlüsselwort "Gatter" markiert alle bis zum nächsten Schlüsselwort "LeadConns" stehenden Zeilen als Gatterbeschreibungen.*
1,1,2,2,R,0,0	
2,1,2,5,S,0,0	
3,8,5,2,Nor 1,0,0	*Hier wird beispielsweise ein NOR-Gatter bezeichnet. Es ist Gatter Nr.3, hat den Typ Nr.8 (NOR), liegt an Position x=5 und y=2, besitzt den Alias "Nor 1" und als Startwert für den logischen Ausgang null, die Laufzeitverzögerung 1 ZE (ob der letzte Parameter 0 oder 1 ist, spielt keine Rolle).*
4,8,5,5,Nor 2,0,0	
5,6,8,1,Q,0,0	
6,6,8,6,Q_Strich,0,0	
LeadConns:	*In den nachfolgenden Zeilen werden Leiterverbindungen beschrieben*
0,1,3	*Die erste Zahl ist die ID-Nr. der Verbindung, die zweite gibt an, von wo, die dritte, wohin diese Verbindung führt. Hier also vom Source-Gatter "R" zum Nor-Gatter "Nor 1".*
1,2,4	
2,3,5	
3,3,4	
4,4,6	
5,4,3	
LeadParts:	*In dem dritten Abschnitt einer Schaltkreisdatei werden die zu einer Verbindung gehörenden Segmente aufgeführt. Dieses dient nicht der logischen Verarbeitugn, sondern nur der Animation.*
0,18,3,2,t	
0,18,4,2,t	
1,18,3,5,t	*Die erste Zahl bezeichnet die zugehörige Verbindung, die zweite den Segmenttyp, die dritte und vierte Zahl die x/y-Position, die letzte Zah schließlich nennt die Animationsrichtung (t: Standardrichtung, f: umgekehrt).*
1,18,4,5,t	
2,14,6,1,t	
2,23,6,2,t	
2,18,7,1,t	
3,14,4,3,f	
3,16,4,4,f	
3,22,4,5,t	
3,18,5,3,f	
3,23,6,2,t	
3,11,6,3,f	
4,23,6,5,t	
4,13,6,6,t	

4,18,7,6,t
5,15,4,2,t
5,16,4,3,t
5,13,4,4,f
5,18,5,4,f
5,12,6,4,f
5,23,6,5,t

Die dabei verwendeten Gatter-Typen sind folgende:

- 0: nicht belegt ("emptygate.gif")
- 1: Signalquelle ("source.gif")
- 2: Logisches UND ("and.gif")
- 3: Logisches ODER ("or.gif")
- 4: Logischer Inverter NICHT ("not.gif")
- 5: Logisches Exklusiv-ODER ("xor.gif")
- 6: Signalempfänger ("target.gif")
- 7: Logisches NICHT-UND ("nand.gif")
- 8: Logisches NICHT-ODER ("nor.gif")

Die übrigen Typen (Typ 9 bis 30) werden zur Bezeichnung der verschiedenen Leiterbahn-Segmente verwendet.

E.3. Verzeichnis der Ablaufskripte

Den Einträgen in “Chapters”-Menu sind von oben nach unten gelesen folgende Skripte zugordnet:

- Introduction: intro.skr. Dieses Skript hat die Aufgabe, dem Benutzer das Programm “CircAnimator” vorzustellen, in die Bedienung einzuweisen und die benutzten Symbole und Konventionen zu erläutern.
- What are Hazards: chap0.skr. Mit Hilfe dieser Datei werden die verschiedenen, später näher ausgeführten Hazardtypen kurz dargelegt. Schaltkreise werden hier nicht verwendet, lediglich eine Reihe von Bildern.
- 1.Structural Hazards: chap1.skr. Wie der Kapiteltitel andeutet, befaßt sich dieses Skript mit strukurellen Hazards. Zur Demonstration von Situationen mit strukturellen Hazards werden drei verschiedene Schaltkreise geladen.
- 2. Functional Hazards: chap2.skr. Funktionelle Hazards werden hier anhand eines Schaltkreises demonstriert, indem durch unterschiedliche Verzögerungswerte der Lauf von Signalen über verschiedene Wege demonstriert wird.
- 3. Essential Hazards: chap3.skr. Das zugrundeliegende Skript versucht mit Hilfe einer Zählerschaltung, die Problematik essentieller Hazards zu beleuchten. Aufgrund der dem Thema eigenen Komplexität werden die Grenzen der verwendeten Software leider auch deutlich.
- 4. Races: chap4.skr. Auch Schaltungen, die Wettläufe, bzw. Races beeinhalten, sind meist von der Art, daß die Komplexität verbietet, die Situation auf einen Blick zu erfassen. Dennoch ist es möglich, mit der in diesem Skript vorliegenden Schaltung einen Eindruck der Wettlauf-Problematik zu bekommen.

Die nachfolgenden Skripte dienen allein dazu, dem Benutzer die in den eigentlichen Skripten benutzten Schaltungen im Experimental-Modus zur eigenen Analyse und zum eigenen Austesten verfügbar zu machen. Jedes Skript stellt dabei eine Schaltung einmal im Instantan-, daraufhin zum Experimentieren meist sinnvolleren Zeitschritt-Modus vor.

- Test intro1.txt (introduction): exp1.skr
- Test chap1_3.txt (struc. hazards): exp2.skr
- Test chap1_2a.txt (struc hazards): exp3.skr
- Test chap1_2b.txt (func. hazards): exp4.skr
- Test essent.txt (essent. hazards): exp5.skr
- Test control2.txt (races): exp6.skr

F Literaturverzeichnis:

Java:

David Flanagan, Java in a Nutshell, O'Reilly & Assoc., 1996

Alexander Newman et al., Using Java - Special Edition, Que Corp., 1996

James Gosling et al., The Java Application Programming Interface Vol1/2, Sun Microsystems, 1996

Schaltungstheorie:

Stephen H. Unger, Asynchronous Sequential Switching Circuits, John Wiley & Sons, 1969

F.J.Hill's and G.R.Peterson, Introduction to Switching Theory And Logical Design, John Wiley & Sons, 1968,1974, 2nd edition

Klaus Lagemann, Rechnerstrukturen, Springer-Verlag, 1987