Vorwort

Dumm dürft Ihr sein, aber laßt Euch was einfallen ...
H. Ford, M. Grundig, E. Heinkel, S. P. Koroljow,
W. Messerschmitt, H. Nixdorf, D. Packard u. v. a. m.

There are nine and sixty ways of constructing tribal lays,
And every single one of them is right!
Rudyard Kipling

Keinen Reimer wird man finden,
Der sich nicht den besten hielte,
Keinen Fiedler, der nicht lieber
Eigne Melodien spielte.
Johann Wolfgang Goethe

Es ist immer von Vorteil, über eine gut gefüllte Werkzeug- und Trickkiste zu verfügen und nicht nur über einen einzigen Hammer. In diesem Sinne ist das vorliegende Buch dazu gedacht, den Werkzeugkasten aufzufüllen, der vorgesehen ist, um mit Schaltungen und Programmen Steuerungsaufgaben zu lösen.
Oftmals wird man fertige Plattformen einsetzen, beispielsweise speicherprogrammierbare Steuerungen (SPS, PLCs) oder Industrie-PCs. Kommt so etwas nicht in Betracht, muß man also die Plattform selbst entwickeln, wird man zuerst wohl an Computermodule und Mikrocontroller denken, vielleicht auch an IP Cores und FPGAs. Man kann fertige Prozessorkerne als IP Cores einsetzen oder die Entwurfsaufgabe mit sequentiellen Schaltungen lösen. Es ist im Grunde die Wahl zwischen Software und Hardware. Beides hat Vorteile, Nachteile, Probleme und Entwurfsschwierigkeiten. Es gibt aber weitere Alternativen. Manche haben sich schon vor Jahrzehnten bewährt. Andere könnten vielleicht mit den heutigen Technologien so implementiert werden, daß man sie in der Praxis verwenden kann.
Zu den bewährten Grundsatzlösungen gehört das Prinzip der Mikroprogrammsteuerung. Ein Mikroprogrammsteuerwerk kann nicht rechnen, sondern nur Steuerwirkungen ausüben und verzweigen. Mikroprogramme bestehen aus Mikrobefehlen, die in Formatgestaltung und Wirkungsweise an die zu steuernden Einrichtungen angepaßt sind. Wenn man vor allem steuern, aber gar nicht oder nur wenig rechnen muß, ist das Mikroprogrammsteuerwerk dem üblichen Universalprozessor deutlich überlegen. Es braucht weniger Ressourcen (Siliziumfläche, Logikzellen) als ein ansonsten vergleichbarer RISC-Prozessor und reagiert schneller. Wenn man Maschinenprogramme durch Mikroprogramme ersetzt, ergibt sich ein Geschwindigkeitsgewinn (Speedup) von – erfahrungsgemäß – etwa 2:1 bis 40:1, je nachdem, was ablaufen soll.
Aus den Prinzipien der Mikroprogrammsteuerung können sich alternative Lösungsansätze ergeben. Es ist eine Art dritter Weg, eine Mischung von Hardware und Software, mit der man oftmals Vorteile vereinigen und Nachteile abmildern kann. Mikroprogrammsteuerwerke sind überschaubare Schaltungsstrukturen. Sie sind leichter zu entwerfen als komplexe Spezialschal-

tungen. Ein Mikroprogrammsteuerwerk kann in jedem Taktzyklus Signale abfragen und Steuerwirkungen ausüben. Somit ergeben sich geringe Latenz- und Reaktionszeiten. Wird das Steuerwerk kompromißlos auf Leistung ausgelegt, kann es alle Steuersignale auf einmal erregen, alle Bedingungen gleichzeitig auswerten und in jedem Taktzyklus in mehrere Richtungen verzweigen – wenn es sein muß, aufgrund von Bedingungen, die sich im selben Taktzyklus ergeben haben. Unterprogrammaufrufe, Interrupts und und Programmumschaltungen kann man so implementieren, daß sie keine zusätzlichen Maschinenzyklen (Overhead) kosten. Wie beim universellen Prozessor wird die funktionelle Komplexität aus der Schaltung in einen Speicherinhalt verlagert. Die meisten Änderungen sind keine Schaltungsänderungen, sondern Programmänderungen, die meisten Entwurfsfehler sind keine Schaltungsfehler, sondern Programmierfehler. Die Verfahren und Hilfsmittel der Fehlersuche entsprechen denen der üblichen Maschinenprogrammierung.
Deshalb dürfte es sich lohnen, diese Prinzipien wieder zu beleben und mit heutigen Mitteln zu implementieren. Es liegt nahe, solche Steuerwerke zunächst als Soft-Cores zu entwerfen.
Wir wollen uns auf den zeitgemäßen Stand der Technik beziehen. Es geht nicht darum, die Spar- und Tricklösungen der Vergangenheit wieder aufleben zu lassen. Die Entwicklungsgeschichte betrachten wir nur, um uns in die Grundlagen einzuarbeiten und etwas zu lernen. Worauf wir uns stützen wollen, sind zwei bewährte Prinziplösungen, der Zustandsautomat und der Universalprozessor. Das Mikroprogrammsteuerwerk schiebt sich gleichsam dazwischen, es hat von beiden etwas. Die Mikroprogrammierung als Philosophie und das Mikroprogrammsteuerwerk als Schaltungslösung kommen für mehrere Anwendungsbereiche in Betracht:

- Das Mikroprogrammsteuerwerk als Plattform, als Alternative zum fertigen Mikrocontroller. Es braucht weniger Ressourcen und ermöglicht komplette Schaltungslösungen aus einem Guß, als Verbund von Steuerung, Verarbeitung und Peripherie.
- Das Mikroprogrammsteuerwerk als Steuereinheit in universellen oder speziellen Maschinen. Es ist eine bewährte Grundsatzlösung. Die Hardware ist leicht zu entwerfen, die Anwendungslösung ist im wesentlichen “soft“ und immer wieder zu ändern, ohne die Schaltung jedesmal neu synthetisieren zu müssen.
- Mikroprogrammierte Zusätze und Maschinen als Ergänzung gegebener Prozessoren, beispielsweise als Akzelerator, E-A-Prozessor oder programmierbare Peripherie.
- Die externe Erweiterung von Maschinenbefehlen gegebener Prozessorkerne. Diese Design-Idee ergibt praktisch eine Vereinigung der üblichen Befehlswirkungen und der Mikroprogrammierung. Der Prozessor kann so zeitweilig zum spezialisierten Mikroprogrammsteuerwerk werden.
- Die Prinzipien der Mikroprogrammsteuerung als Grundlage neuer Architekturlösungen (Micro-Architecture). Die Mikrobefehle sind die elementaren universellen Maschinenbefehle.
- Die Prinzipien und Schaltungslösungen als Quelle von Anregungen für unkonventionelle Architekturlösungen. Sie seien hier mit den Begriffen Ressourcenvektormaschine und Ressourcen-Algebra angedeutet.

Der Inhalt im Überblick

Der Zweck des vorliegenden Buches ist – aufs Ganze gesehen – die Horizonterweiterung. Die Grundlagen der Mikroprogrammsteuerung sollen so dargestellt werden, daß sie als Startpunkt eigener Entwicklungen nutzbar sind. Hierzu werden sowohl bewährte Lösungen als auch Entwicklungsvorschläge beschrieben. Teils sind es Prinzipien und theoretische Ansätze aus der Entwicklungsgeschichte, die neu ventiliert werden, teils Problemlösungen und Lösungsvorschläge, die sich im Laufe der Zeit ergeben haben. Worauf wir hinauswollen, ist das Mikroprogrammsteuerwerk als Computer im Computer, als ganz elementarer Prozessor, der schnell entworfen ist und dabei an die Anforderungen des jeweiligen Einsatzfalls angepaßt werden kann. Es ist oftmals eine Alternative zu herkömmlichen Mikrocontrollern und Prozessorkernen. Da die Wirkprinzipien und Schaltungsstrukturen einfach und überschaubar sind, kann man es sich eher leisten, eigene Schaltungen zu entwickeln und somit unabhängiger werden[1]. Womöglich ergeben sich aus der Wiederbelebung solcher Ideen auch Anregungen zur grundsätzlichen Weiterentwicklung der Rechnerarchitektur.

Kapitel 1: Grundlagen

Am Anfang steht der Gedanke der freien Programmierbarkeit. Auf dieser Grundlage wollen wir Steuerungsaufgaben lösen. Dieses grundsätzliche Entwickungsziel wird vorgestellt und begründet. Im Laufe der Entwicklungsgeschichte hat sich die Frage ergeben, ob man eine anwendungsspezifische Schaltung entwirft oder eine Universalmaschine (Prozeßrechner, Minicomputer, Mikroprozessor, Mikrocontroller) beschafft, an die Anwendungsumgebung adaptiert und zur Lösung der Anwendungsaufgabe programmiert. Diese traditionelle Frage – Hardware oder Software – stellt sich aber nicht mehr in dieser Gegensätzlichkeit, denn mittlerweile ist alles programmierbar, auch die Schaltungslösung. Man kann sie erzeugen, indem man Programme schreibt (Verhaltensbeschreibung) und so oft ändert, bis sie zufriedenstellend funktioniert. Um aber solche Programme in echte Schaltungen umzusetzen, muß die Schaltungssynthese durchlaufen werden. Sie beruht auf Booleschen Algorithmen hoher Komplexität. Darin liegt der entscheidende Unterschied zwischen Schaltungsprogrammierung und Speicherprogrammierung. Daß man mit sozusagen gewöhnlichen Programmen, die in den Speicher einer fertigen Maschine geladen werden, auch sehr ehrgeizige Aufgaben erledigen kann, ist eine Erfahrungstatsache. Somit liegt es nahe, das Prinzip der Speicherprogrammierung auch zur Steuerung der innersten Abläufe in den Schaltungen nutzbar zu machen. Der Grundgedanke: Wenn gespeicherte Programme steuern, wie die Funktionseinheiten der Maschine zusammenwirken, kann sich die Schaltungssynthese auf die einzelnen Funktionseinheiten beschränken. Man kann eine solche Maschine entwerfen, indem man sie in überschaubare Funktionseinheiten zerlegt, die sich bequem mittels Verhaltensbeschreibung erfassen lassen[2]. Wie sie zusammenarbeiten, steuert das Mikroprogramm. Es wirkt als eine Art Dirigent.

1. Von Herstellern, Anbietern, Lizenzkosten und Lizenzbedingungen.
2. Es gibt Entwicklungssysteme, die ein solches Vorgehen unterstützen. Sie können fertig synthetisierte Funktionseinheiten auf dem Schaltkreis placieren und die Verbindungen zwischen ihnen erzeugen.

Kapitel 2: Zustandsautomaten und Steuerautomaten

Wir haben die Mikroprogrammierung als eine Art dritter Weg angesprochen, als eine Verbundlösung von Hard- und Software. Um grundsätzliche Ideen zu finden, nach denen man solche Maschinen entwickeln kann, sollten wir bestrebt sein, von beiden Alternativen etwas zu lernen, von den als Schaltung entworfenen Steuerautomaten und von den Universalrechnern, die für Steuerungsaufgaben eingesetzt werden. Die Kapitel 2 und 3 sind überblicksmäßige Darstellungen, um Grundbegriffe zu erläutern und Information zusammenzutragen[3]. Es gehört seit längerem zum üblichen Fachwissen, Steuerschaltungen als Zustandsautomaten aufzufassen und als solche zu entwerfen. Viele Steuerungsaufgaben können mit wenigen elementaren Mustern von Zustandsübergängen erledigt werden, die man mit einfachen Schaltungen implementieren kann. Eine mittlerweile übliche Alternative besteht darin, das gewünschte Automatenverhalten zu formulieren (Verhaltensbeschreibung) und eine entsprechende Schaltung synthetisieren zu lassen. Wie sehen die so erzeugten Schaltungen grundsätzlich aus? Womöglich kann man Mikroprogrammsteuerwerke so gestalten, daß sie typische Zustandsübergänge unterstützen und dabei kaum langsamer sind als synthetisierte Schaltungen.

Nun legen schon die gleichsam naiven Blockschaltbilder der Automatentheorie nahe, die Überführungsfunktion und die Ausgabefunktion mit Speichern zu implementieren. Kleine Automaten kann man auf diese Weise auch tatsächlich bauen (speicherbasierte Zustandsautomaten). Sollten die Speicher zu groß werden, kann man die Automatenfunktionen in mehreren Schritten erledigen; die Zustandsübergänge sind dann keine kombinatorischen Zuordnungen mehr, sondern Algorithmen. Man spricht deshalb vom algorithmischen Zustandsautomaten (Algorithmic State Machine, ASR). Je einfacher die Grundmuster der Zustandsübergänge, desto mehr Aufwand wird gespart, desto mehr Schritte sind aber auch auszuführen. Die einfachsten Übergangsmuster sind das Verzweigen in zwei Richtungen (Branch Sequencer) und das Weiterzählen der Zustandscodierung (Count Sequencer). Durch Kombinieren solcher Prinziplösungen ergeben sich anwendungsbrauchbare speicherbasierte Automaten mit fließendem Übergang zum Mikroprogrammsteuerwerk.

Die letzten Abschnitte des Kapitels betreffen Boolesche Steuerautomaten, universelle algorithmische Automaten und die Turing-Vollständigkeit. Es sind Konzepte der Theorie, die als Anregungen in Betracht kommen oder zum Grundlagenwissen gehören.

Boolesche Steuerautomaten sind – als sog. Bitprozessoren – gelegentlich gebaut worden, auch in der industriellen Steuerungstechnik. Mittlerweile ist es aber möglich, über solche Einfachlösungen weit hinauszugehen und auch in der Anwendungsprogrammierung mit komplexen Booleschen Ausdrücken zu arbeiten, die in der Maschine unterstützt werden (funktioneller Digitalsimulator).

3. Dabei geht es vor allem darum, für die Mikroprogrammsteuerung etwas zu lernen und womöglich auf neue Ideen zu kommen. Deshalb lassen wir vieles weg, was zum allgemein bekannten Fachwissen und zum Stand der Technik gehört.

Universelle algorithmische Automaten bestehen aus Steuer- und Operationsautomaten. Es ist ein theoretisches Konzept, das schon vor Jahrzehnten entwickelt wurde, ein verallgemeinertes Prinzip der Mikroprogrammsteuerung, das über die üblichen Wirkprinzipien der Mikroprogrammsteuerwerke hinausgeht. Daraus lassen sich weiterführende Grundsatzlösungen zur Steuerung von Operationsautomaten und Universalmaschinen ableiten.
Den Begriff der Turing-Vollständigkeit brauchen wir, wenn wir universelle Maschinen bauen wollen. Das Problem ergibt sich u. a. dann, wenn man Mikrobefehle zur System- und Anwendungsprogrammierung verwenden will (Micro-Architecture).

Kapitel 3: Universalrechner als Steuerautomaten
Der Universalprozessor ist mittlerweile zur bevorzugten Plattform geworden. Die meisten Anwendungsaufgaben wird man durch Programmieren lösen. Über anwendungsspezifische Schaltungslösungen wird man erst dann nachdenken, wenn es nicht anders geht.
Wer hier nach grundsätzlichen Verbesserungen strebt, muß bis zu den ganz elementaren Grundlagen zurückgehen und sich Fragen zuwenden, die man üblicherweise längst nicht mehr als Problem ansieht. Das beginnt mit der Überlegung, wozu man eigentlich Universalrechner einsetzt und wie man sie einsetzt. Der Universalprozessor wurde ursprünglich als Rechenmaschine entwickelt. Die meisten Maschinen wurden verwendet, um mit einem Programm aus Eingaben und gespeicherten Daten Ausgaben und neue gespeicherte Daten zu erstellen. Diesen grundsätzlichen Einsatzfall nach dem Schema Eingabe – Verarbeitung – Ausgabe bezeichnen wir mit dem traditionellen Begriff als elektronische Datenverarbeitung (EDV). Praktisch alle Universalprozessoren – einschließlich der Mikrocontroller – die sich am Markt durchgesetzt haben, sind im Grunde EDV-Maschinen. In den weitaus meisten Einsatzfällen sind aber gar keine EDV-mäßigen Aufgaben zu bearbeiten. Vielmehr dient der Prozessor letzten Endes dazu, die Funktionen einer anwendungsspezifischen Schaltung zu erbringen. Die Funktionsblöcke werden zu Programmen, an die Stelle der anwendungsspezifischen Signalwege treten programmierte Wertzuweisungen und Datentransporte. Der Universalrechner wird so zum programmierbaren Emulator. Daß man EDV-Maschinen für solche Zwecke einsetzen kann, liegt eigentlich nur an der Universalität = Turing-Vollständigkeit (so daß es grundsätzlich funktioniert) und an der hohen Arbeitsgeschwindigkeit (so daß die Maschinen ein zufriedenstellendes Realzeitverhalten aufweisen). Das legt die Frage nahe, ob man nicht zu noch besseren Grundsatzlösungen kommen könnte, wenn man die Maschinenarchitektur von Anfang an zur Nutzung als programmierbarer Emulator, funktioneller Digitalsimulator usw. auslegen würde.
Es entspricht dem Stand der Technik, die Universalprozessoren so zu nehmen wie sie am Markt angeboten werden, und bedarfsweise mit Schaltungslösungen in FPGAs zu ergänzen (Hardware-Software-Co-Design). Die Frage ist, wann sich der Übergang vom Prozessor auf die programmierbare Logik lohnt. Auch kommt man womöglich zu überraschenden Einsichten, wenn man über den grundsätzlichen Unterschied zwischen Software und Hardware genauer nachdenkt. Wer ein Problem durch Programmieren lösen will, ist letzten Endes auf einen starren und sehr beschränkten Funktions- und Ressourcenvorrat angewiesen, wie er durch die Befehlsliste und die architekturseitige Ausstattung der Maschine gegeben ist. Wenn man hingegen die Frei-

heit hat, die Schaltung selbst zu gestalten, lassen sich manche knifflige Programmierprobleme ohne weiteres lösen, gelegentlich sogar auf einfachste Weise.
Es liegt nahe, Universalmaschinen mit Beschleunigungseinrichtungen zu ergänzen. Dafür gibt es mehrere Prinziplösungen und viele Beispiele. Sie unterscheiden sich darin, welchen Aufwand man treibt, nach welchen Prinzipien man die zusätzlichen Funktionseinheiten gestaltet und wo man sie an die Universalmaschine anschließt. Die grundsätzliche Frage aber ist, ob es sich lohnt. Ist ein auf Leistung entworfener großer Universalprozessor besser oder eine Maschine, die aus einem vergleichsweise leistungsschwachen Prozessorkern und Beschleunigungsschaltungen im FPGA besteht? Dieses Problem sollte man von beiden Seiten betrachten.
Die letzten Abschnitte stellen Prinziplösungen vor, die seit längerem bekannt sind. Es geht um den Entwurfsgedanken, einen Prozessor wie einen x-beliebigen Schaltkreis zu behandeln, den man in die Anwendungsschaltung gleichsam einwerfen kann, und um das Prinzip des Multitasking in der Hardware, das auf dem zyklischen Umschalten von Registern beruht. Beides ist als Anregung für weiterführende Entwicklungen gedacht. So könnte man an an Prozessoren als IP Cores in FPGAs denken, die von Grund auf zum Erweitern mit einfachen Zusatzschaltungen vorgesehen sind. Auch liegt der Gedanke nahe, eine Art frei programmierbares CPLD oder FPGA zu schaffen, einen kleinen funktionellen Digitalsimulator, der sozusagen live – ohne jedesmal ein Boolesche Schaltungssynthese zu durchlaufen – mit beliebigen Schaltfunktionen programmiert werden kann. Das Hardware-Multitasking ist eine oftmals effektivere Alternative zum Pipelining, vor allem auch in Mikroprogrammsteuerwerken der oberen Leistungsklassen.

Kapitel 4: Grundlagen der Mikroprogrammsteuerung
Am Anfang der Entwicklungsgeschichte stand das Steuerwerk des Universalprozessors. Mikroprogrammierung bedeutet, die Steuersignale, die in den einzelnen Taktzyklen benötigt werden, als Bitmuster aus einem Steuerspeicher zu entnehmen. Die Befehlsablaufsteuerung wird damit programmierbar, es ergibt sich gleichsam ein Computer im Computer. Nun wird man heutzutage Prozessoren nur sehr selten selbst entwerfen, sondern zumeist fertig kaufen. Infolge dessen kann man das Fachwissen, wie Steuerwerke aufgebaut sind, nicht mehr in der Tiefe voraussetzen, die hier erforderlich ist. Deshalb beginnt das Kapitel mit einer entsprechenden Einführung. Anschließend werden die Grundlagen der Mikroprogrammsteuerung näher erläutert. Hierbei geht es nicht um einen historischen Rückblick, sondern um die Erläuterung von Grundlagen als Anregung zum Entwickeln, wenn nicht gar zum Erfinden. Wir betrachten das Mikroprogrammsteuerwerk als Computer im Computer und – darüber hinausgehend – als Plattform zum Steuern von Funktionseinheiten, als eine Art Dirigent. Womöglich ist es auch zweckmäßig, komplexere Funktionseinheiten mit eigenen Mikrobefehlen zu steuern. Die Funktionseinheit wird so zum Operationsautomaten. In den Bereich solcher Entwurfsideen gehört auch, die Ausführung der Mikrobefehle mit einem weiteren speicherprogrammierten Steuerwerk zu steuern (Nanoprogrammsteuerung). Zudem erläutern wir typische Leistungsprobleme der Mikroprogrammsteuerung, stellen die grundsätzlichen Mikrobefehlsformate vor und diskutieren den Gedanken, Mikrobefehle zur Grundlage der Maschinenarchitktur zu machen (Micro-Architecture).

Kapitel 5: Mikroprogrammsteuerwerke
Dieses Kapitel behandelt tiefere Einzelheiten der Auslegung von Mikroprogrammsteuerwerken. Nun wird man Entwurfsaufgaben oftmals erledigen, indem man das gewünschte Verhalten formal beschreibt, also nicht, indem man Schaltsymbole auf dem Bildschirm miteinander verbindet. Aber nach wie vor sollten Schaltbilder und Impulsdiagramme am Anfang stehen, denn man muß man erst einmal wissen, was man will, ehe man ein Verhalten so präzise beschreiben kann, wie es notwendig ist, um eine funktionierende Schaltung synthetisiert zu bekommen. Auch müssen wir uns mit dem Taktsystem etwas genauer beschäftigen, da man typischerweise bestrebt sein wird, vollsynchron zu entwerfen und die Takterzeugungs- und Taktverwaltungsschaltungen der FPGAs gut auszunutzen.

Kapitel 6: Maschinen mit Mikroprogrammsteuerung
Was sich hier darstellen läßt, sind nur Beispiele einfacher Maschinen. Sie sollen aber auch praxisbrauchbar sein, beispielsweise als Soft Cores in FPGAs oder als Prozessorkerne alternativer Mikrocontroller. Wir folgen dabei den Erläuterungen von Kapitel 2 und beginnen mit der einfachen Zeitplansteuerung (Sequencer). Der Steuerautomat wird universell, wenn er auf Bedingungen reagieren kann (Branch Sequencer). Eine solche Maschine kann aber keine Daten speichern, nichts entscheiden, berechnen usw. Es liegt nahe, den Steuerautomaten zum Universalprozessor zu erweitern, damit er diese Aufgaben selbst erledigen kann. In vielen Steuerungsanwendungen ist aber die echte Universalität = Turing-Vollständigkeit gar nicht erforderlich; es genügen sog. algorithmische Steuerautomaten, die nur jene universellen Operationen unterstützen, die für die jeweilige Anwendung erforderlich sind. In unseren Erläuterungen bilden sie die Vorstufe zur echten Universalmaschine. Wir beginnen mit Einadreßmaschinen, die sich durch systematisches Weiterentwickeln aus den algorithmischen Steuerautomaten ergeben. Dann diskutieren wir die Mikroprogrammsteuerung kleiner (= überschaubarer) Universalregistermaschinen. Abschließend betrachten wir Beispiele einfacher Maschinen, deren Mikrobefehle zugleich die elementaren Maschinenbefehle sind (Micro-Architecture).

Kapitel 7: Der Prozessorkern als Mikroprogrammsteuerwerk
Mikroprogrammsteuerwerke haben typische Vorteile. Heutige Projekte sind aber durch einen großen Umfang an Software gekennzeichnet. Deshalb wird man zumeist auf verbreitete Prozessorarchitekturen und Entwicklungsumgebungen zurückgreifen. Die Verbundlösung aus Prozessor und Mikroprogrammsteuerwerk (als Akzelerator oder Coprozessor) liegt nahe, hat aber ihre eigenen Probleme. Hier wird als Alternative vorgeschlagen, die Befehle außerhalb des Prozessors zu verlängern. Im Innern wird nichts geändert. Dem Programmspeicher wird ein Erweiterungs- oder Steuerspeicher hinzugefügt. Die Verlängerungen lösen zusätzliche Steuerwirkungen aus. Sie können auch anweisen, wie der Befehl auf dem Wege vom Speicher zum Prozessorkern verändert oder ersetzt wird. Der Prozessor wird so zeitweilig zum spezialisierten Mikroprogrammsteuerwerk. Alle Programme, die mit diesen Funktionen nichts zu tun haben, bleiben wie sie sind.

Kapitel 8: Schaltungseinzelheiten
Dieses Kapitel enthält eine Einführung in tiefere Einzelheiten, die man kennen sollte, wenn man ein Mikroprogrammsteuerwerk tatsächlich entwerfen möchte. Zunächst betrachten wir den Steuerspeicher sowie Vorkehrungen zur Diagnose (der Hardware) und zum Debugging (der Mikroprogramme)[4]. Zu den grundsätzlichen Entwurfsentscheidungen gehört auch, ob man die Maschine mit schmalen Datenwegen, geringer Verarbeitungsbreite usw. baut und mit extrem hohen Taktfrequenzen betreibt oder man längere Taktzyklen vorsieht, in diesen aber mehr leistet. Ergänzende Darlegungen betreffen Schaltungslösungen zum Darstellen von Zeitintervallen, das Synchronisieren, das Erzeugen von Taktphasen und die Nutzung des Universalprozessors als Schaltungsbaustein.

Kapitel 9: Historische Beispiele
Technikgeschichte ist immer interessant. Wir können aber aus der Vergangenheit nur Anregungen entnehmen; in den Einzelheiten sieht es heute anders aus. Deshalb beschränken wir uns auf einige wenige Beispiele aus den Bereichen der Mainframes, der kleineren Maschinen und der Schaltkreise.

Anhang 1: Eigene Maschinen programmieren
Es versteht sich von selbst, zu fragen, wie eine Maschine mit neuartiger, womöglich unkonventioneller Architektur in der Anwendungspraxis programmiert werden soll. Diese Thema kann hier nicht näher behandelt werden. Im Anhang beschränken wir uns auf Einfachlösungen. Für den Anfang – wo sozusagen zu Fuß programmiert werden muß – genügt jeder einigermaßen leistungsfähige Makroassembler. Darüber hinaus werden einige weiterführende Überlegungen angedeutet.

Anhang 2: Elementare Operationen
Manche Blockschaltbilder zeigen Verarbeitungswerke mit ihren Registern und Verknüpfungsschaltungen, vor allem mit Arithmetik-Logik-Einheiten (ALUs). Wir beschreiben aber keine Einzelheiten. Der grundsätzliche Gedanke ist, von den bewährten Lösungen der typischen Prozessorkerne zu lernen und das zu implementieren, was jeweils zweckmäßig ist, bedarfsweise aber auch Register und Verknüpfungsschaltungen für anwendungsspezifische Operationen vorzusehen. Hier geben wir einen Überblick über Operationen und Registermodelle elementarer universeller Operationswerke, wie sie in vielen Prozessoren zu finden sind.

4. Praxistip: Beides (Diagnose und Debugging) von Anfang an einplanen, nicht erst später hineinbasteln ...