Inhalt

Stand: 30. 3. 1990
Release: 1

S.015-Dokumentationsbezeichnung: ADR.015

1. Überblick

Das System.015 (im folgenden: S.015) repräsentiert ein einheitliches Architekturkonzept für vielfältige Ausführungsformen von Mikrocontrollern, die vorzugsweise für den Einsatz in "embedded systems" vorgesehen sind. Wesentliche Anwendungsgebiete sind Steuerungen aller Art, Zusatzeinrichtungen (Accelleratoren) für herkömmliche Universalrechner (Personalcomputer, Mikroprozessorbaugruppen für standardisierte Bussysteme usw.) sowie Funktionseinheiten in komplexen Informationsverarbeitungssystemen.
Die Schaltungsstrukturen des S.015 sollen in ASIC-Zellenbibliotheken bereitgestellt werden. Alle Hardwarekomplexe haben reguläre Interfaces, über die sie gemäß dem jeweiligen Einsatzfall zusammengeschaltet werden können; es handelt sich gleichsam um eine "Hardware-Baukasten", aus dessen Komponenten die jeweilige Anwendungslösung aufgebaut wird, wobei neben dem Zusammenschalten nur noch - falls überhaupt erforderlich - vergleichsweise einfache Anpassungsschaltungen für die jeweilige Anwendungsumgebung zu entwerfen sind.
Im folgenden werden die Zielvorstellungen und Grundsätze für die Entwicklung der S.015-Architektur dargelegt.

Hinweis:
Die Hardware-Strukturen und -Interfaces werden in dieser Schrift nicht behandelt. Die Architektur des S.015 ist im Architekturhandbuch beschrieben (/7/).

2. Allgemeine Ziele

Bisher werden programmierbare Steuerungsysteme des oberen Leistungsbereichs aus mehreren Leiterplatten zusammengesetzt, die über Bussysteme untereinander verbunden sind. Demgegenüber bieten moderne ASIC-Technologien einen solch hohen Integrationsgrad, daß sich damit die eigentliche (d. h. informationsverarbeitende) Steuerelektronik wesentlich kompakter aufbauen läßt (oft in Form einer einzigen Leiterplatte bzw. Baugruppe). Das hat offensichtliche Vorteile:

1. Höhere Zuverlässigkeit durch geringere Anzahl an Bauelementen, Steckverbindern usw.

2. Physisch kompakte Steuerungsbaugruppen lassen sich oft unmittelbar in der zu steuernden Umgebung anordnen, z. B. in die betreffenden Maschinen einbauen, wodurch sich, um einen der Vorteile zu nennen, die Stellfläche verringert.

3. Sind Steuerungsbaugruppen kompakt, flexibel, leistungsfähig und kostengünstig, so kann man Gebrauchswerte schaffen, die sonst im Rahmen üblicher Aufwandsgrenzen nicht zu verwirklichen sind: man kann in den Endprodukten komplexere Funktionen vorsehen (den Automatisierungsgrad weiter erhöhen, Optimierungsrechnungen in Realzeit ausführen usw.); man kann aber auch fehlertolerante Systeme aufbauen, da die erforderliche Redundanz vergleichsweise preiswert ist.

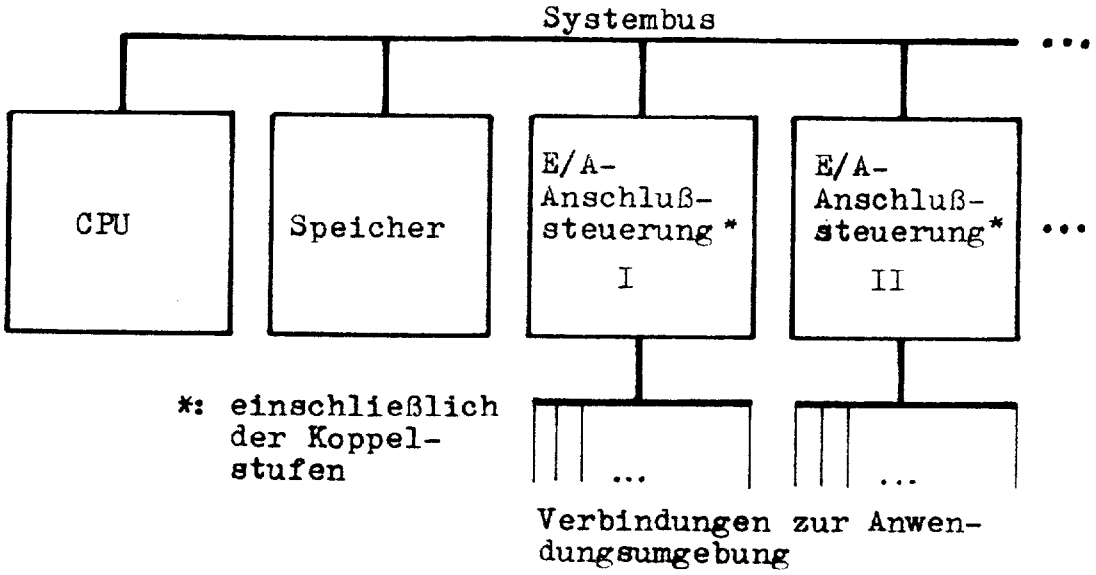
4. Hinsichtlich der vielfältigen Diagnose-, Fehlerlokalisierungs- und Reparaturprobleme läßt sich eine wirksame Abhilfe schaffen. Hat man anstelle vieler spezieller Funktionseinheiten nur wenige Typen kompakter Baugruppen, so ist es dem Anwender zumutbar, eine Reserve an Ort und Stelle vorrätig zu halten und bei Bedarf selbst zu wechseln. Der Anlaß dazu wird - wegen der an sich hohen Zuverlässigkeit - nur selten gegeben sein, und das Personal im Hause ist garantiert schneller "vor Ort" als der Wartungstechniker, der erst anreisen muß.

Hinweis:
Ein solcher Ansatz erfordert nicht nur Maßnahmen seitens der Architektur und der Betriebssoftware (z. B. zum Überwachen und Testen; für Fehlerbehandlung, Rekonfiguration und Wiederanlauf), sondern auch besondere Vorkehrungen bei der mechanischen Gestaltung der Baugruppen, Kundendokumentation usw., die hier nicht weiter behandelt werden.

Die genannten Vorteile sind nicht ohne weiteres realisierbar. Komplexe ASIC-Schaltungen lassen sich nicht im Rahmen herkömmlicher Entwicklungsabläufe verwirklichen (Spezifizieren - Entwerfen - Aufbauen - Erproben - Ändern unter industrieüblichen Vorstellungen zum Zeit- und Personalbedarf); man kann sich bei weitem nicht so viele Änderungszyklen leisten wie beim herkömmlichen Schaltungsaufbau. Werden, um den Schwierigkeiten aus dem Wege zu gehen, nur bewährte Schaltungslösungen in ASIC umgesetzt, so verschenkt man eine Vielfalt von Innovationsgelegenheiten.
Daraus folgt der Grundsatz: je komplexer eine bestimmte Funktionseinheit an sich ist, um so weniger Typen, Varianten usw. davon darf es geben. Das erzwingt, soviele funktionelle Eigenschaften wie möglich programmierbar zu gestalten. In den meisten Einsatzfällen soll es überhaupt nicht mehr nötig sein, spezifische Verknüpfungsschaltungen zu entwerfen, vielmehr sollen neben dem programmierbaren ASIC und seiner Speicherausstattung nur noch die unumgänglich notwendigen Schaltmittel zur Anpassung an die Anwendungsumgebung erforderlich sein (z. B. Pegelwandler, Optokoppler usw.).
Zum Aufbau komplexer Steuerungssysteme werden serielle Verbindungen zwischen den einzelnen Baugruppen bevorzugt. Vorteile: Zuverlässigkeit, einfache Verkabelung, praktisch beliebige Leitungslängen, elektrische Störsicherheit und Potentialtrennung (bei Glasfaser-Kabeln), einfache Fehlerlokalisierung.
Bild 1 veranschaulicht das wesentliche Anwendungsziel des S.015, Bild 2 zeigt die grundsätzliche Struktur eines S.015-Komplexes.

Ein solches Konzept bedeutet, daß die weitaus meisten funktionellen Eigenschaften der Anwendungslösung durch Software verwirklicht werden müssen. Die Architektur muß deshalb so ausgelegt sein, daß man umfangreiche und komplexe Software entwikkeln, austesten und effektiv abarbeiten kann, und sie muß dabei vielfältigen Gesichtspunkten der Praxis gerecht werden. Diese sind in den nachfolgenden Forderungen zusammengefaßt.

a) Herkömmliche Systemstruktur



b) Systemstruktur mit S.015

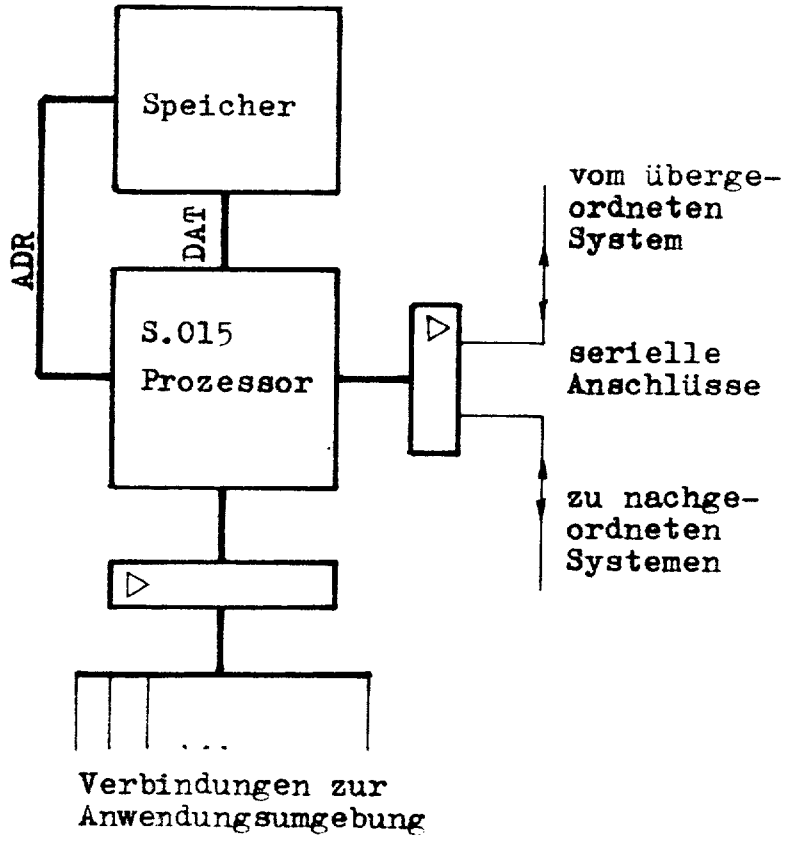


Bild 1 Anwendungsziel des S.015

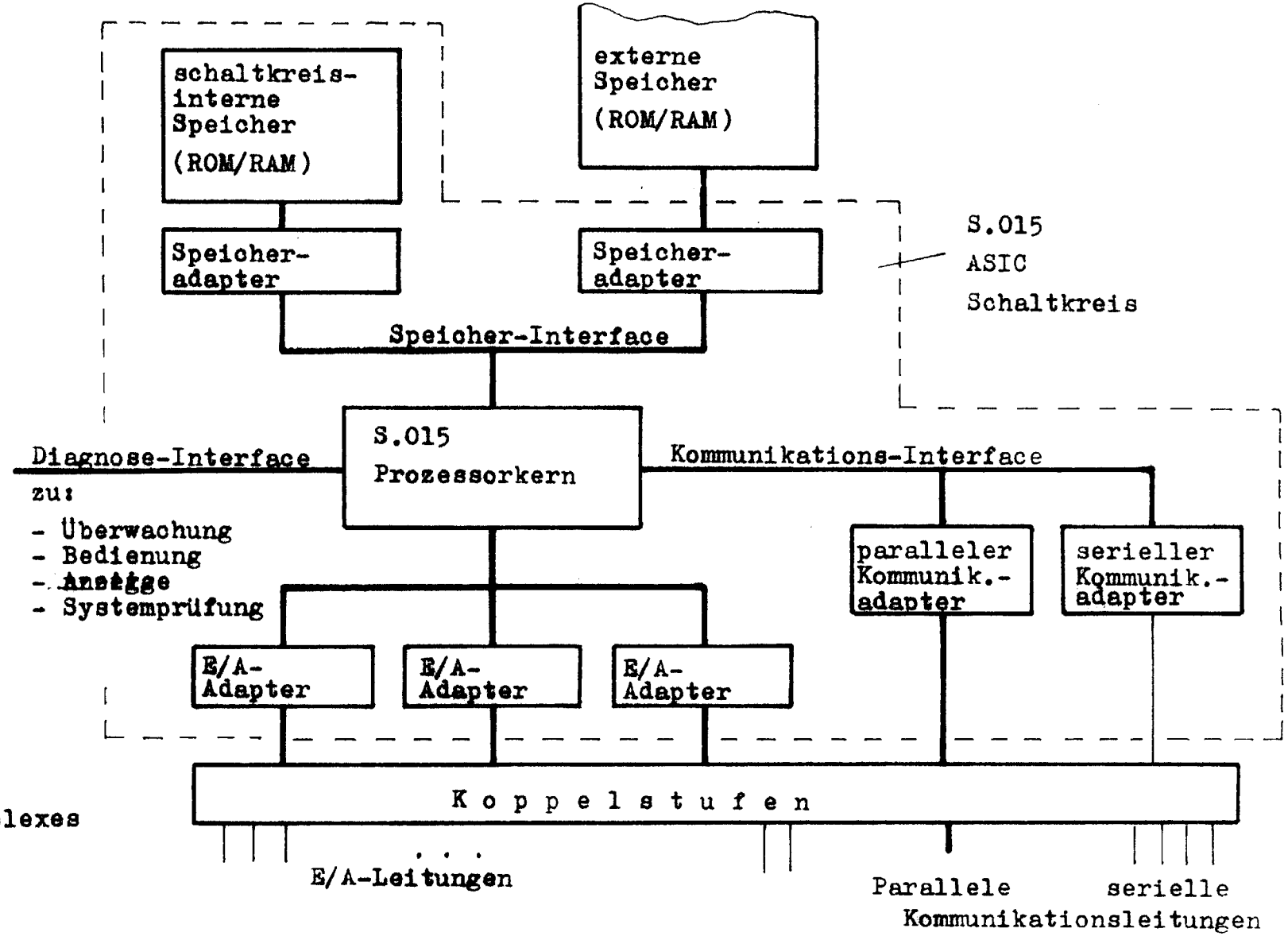


Bild 2

Struktur
eines
S.015-Komplexes

Forderung 1:
Es muß möglich sein, hardware- bzw. anwendungsnah zu programmieren (das betrifft beispielsweise die direkte programmseitige Zugänglichkeit von einzelnen E/A- Leitungen).

Forderung 2:
Das Realzeitverhalten muß genau vorhersagbar sein.

Forderung 3:
Wenn es darum geht, Endprodukte in hohen Stückzahlen kostengünstig zu fertigen, werden in der Praxis vielfältige Programmiertricks (wie z. B. Befehlsmodifikationen) verwendet, ungeachtet aller Lehren des "software engineering". Die Architektur des S.015 muß so etwas zulassen.

Forderung 4:
Im Interesse einer hohen Ausbeute ist die Siliziumfläche sparsam auszunutzen. Dazu muß die S.015-Architektur anwendungsspezifische Implementierungen zulassen, die mit dem jeweils niedrigsten Integrationsgrad auskommen. (Ein vergleichsweise niedriger Integrationsgrad hat zudem Vorteile in Hinsicht auf Störsicherheit, z. B. gegenüber Alpha-Strahlen.)

Forderung 5:
Die ASIC-Anschlüsse müssen praxisgerecht belegt sein. Das betrifft sowohl Speicher- als auch E/A- Anschlüsse. Übliche Speicherschaltkreise müssen sich ohne weitere Zusatzbeschaltung (mit Ausnahme eventuell notwendiger Treiberstufen) direkt anschließen lassen. E/A-Anschlüsse sollen so gestaltet sein, daß sie direkt mit indstrieüblichen Koppelstufen (z. B. Optokopplern, A/D- Wandlern usw.) verbunden werden können. Externe Decoder zum Ansteuern von "chip enable"- Eingängen, kombinatorische Verknüpfungen, "single shot"- Generatoren (zum Bilden von Impulsen), Latches (zum "Fangen" von Impulsen) usw. sollen weitgehend entbehrlich sein.
In manchen Fällen wird man trotz des innewohnenden Leistungsvermögens des S.015 nicht ohne leistungssteigernde Zusatzbeschaltung auskommen. Für wichtige Arten solcher Zusatzbeschaltungen (Coprozessoren, zusätzliche Steuerspeicher, die die Befehlswirkung außerhalb des Prozessors erweitern[1] usw.) sollen praxisgerechte Anschlußbelegungen vorgesehen werden.
Das ist im wesentlichen durch Bereitstellung geeigneter Adapterschaltungen zu lösen, aber auch die Architektur muß auf solche Praxisforderungen abgestimmt sein. Sie darf z. B. die Speicherzugriffsprinzipien nicht so spezifizieren, daß, um die Spezifikation zu erfüllen, unter allen Umständen ein komplizierter externer Memory-Controller notwendig ist. Ähnliches gilt sinngemäß für das Unterbrechungssystem: die Architektur darf nicht zwingend fordern, daß externe Schaltmittel die Priorität feststellen und den Interrupt-Vektor liefern müssen. Vielmehr sollte es ausreichen, einen Eingangsleitung zu erregen, um den Interrupt auszulösen.

---

1 Solche Zusatzbeschaltungen sind in /3/ beschrieben. Sie haben den Vorteil, mit einfachen Mitteln bestimmte steuerungstypische Funktionen wirksam zu beschleunigen.

Forderung 6:
Hochkomplexe Anwendungsfälle brauchen sehr viel Verarbeitungsleistung, und die Programme müssen in höheren Programmiersprachen entwickelt werden. Dafür sind Formen der architekturseitigen Unterstützung zu erwägen.

Forderung 7:
Die meisten S.015-Konfigurationen müssen eine jeweils überlegene Leistung bei industrieüblicher, bewußt kostengünstiger Auslegung des gesamten Hardware-Komplexes erreichen. Das heißt: billige Speicherschaltkreise (ROMs, DRAMs), schmale Datenwege, vergleichsweise geringe Taktfrequenz (und damit Flankensteilheit auf den Informationsleitungen im Interesse der dynamischen Störsicherheit), Aufbau auf Zweiebenen-Leiterplatten usw.
Es darf nicht sein, daß überlegene Leistung nur bei extrem hoher Taktfrequenz und mit schnellen statischen RAMs gewährleistet werden kann. Vielmehr ist die Überlegenheit der S.015-Prozessoren im wesentlichen durch architektur- und schaltungsseitige Konzepte anzustreben.

Forderung 8:
Fragen der Sicherheit im weitesten Sinne sind von Anfang an zu berücksichtigen. Das betrifft Selbsttest, Überwachung während des Betriebs, Fehlerbehandlung, Fehlertoleranz, Maßnahmen gegen Fremdstörungen sowie Unterstützung des "debugging" und des Ausbesserns von Softwarefehlern (mit denen erfahrungsgemäß selbst nach längerer Zeit noch zu rechnen ist).

Forderung 9:
Die S.015-Architektur muß es ermöglichen, eine gewisse Zahl leistungs- und aufwandsmäßig abgestufter Implementierungen so bereitzustellen, daß für jede Implementierung ein vernünftiges Verhältnis von Aufwand und Leistung gewährleistet ist. Mit "kleineren" Implementierungen sind die derzeitigen 8- bzw. 16-bit- Mikrocontroller leistungsmäßig bzw. im Preis/Leistungs-Verhältnis bezogen auf die Anwendungsaufgabe deutlich zu übertreffen. (Beispielsweise muß eine gegebene Anwendungsaufgabe mit deutlich weniger IC-Gehäusen, Leiterplattenfläche usw. lösbar sein.)
Bei "mittleren" Implementierungen gilt das sinngemäß gegenüber Anwendungslösungen, deren Prozessorkerne auf eingeführten 32-bit- Architekturen beruhen.
Schließlich muß die S.015-Architektur für Hochleistungs-Implementierungen geeignet sein, die den innewohnenden Parallelismus in den Programmen ausnutzen und die dazu mit mehreren Verarbeitungswerken ausgestattet sind.

Forderung 10:
Die S.015-Architektur muß das gleichzeitige bzw. zeitlich verschachtelte Abarbeiten mehrerer unabhängiger Programme (Multitasking) unterstützen. Dafür gibt es 3 wesentliche Gründe:

1. Viele der bisher notwendigen speziellen E/A- Anschlußschaltungen sollen durch frei programmierbare virtuelle E/A- Pro-

zessoren ersetzt werden (das bringt praktisch unbegrenzte Flexibilität und vermindert wesentlich die Hardware-Aufwendungen).

2. S.015-Implementierungen des oberen Leistungsbereichs sollen die bisherigen kleineren Multimikrorechnersysteme (mit 4...8 16- bzw. 32-bit- Mikroprozessoren) ablösen; anstelle mehrerer Mikrorechner, die über einen Systembus untereinander verbunden sind, soll ein S.015-Komplex zum Einsatz kommen (damit entfallen Systembus, Buskoppelschaltungen, Teile der Speicherausstattung usw.).

3. Gerade bei Steuerungsproblemen ist es eine natürliche Auffassung, unabhängige Teilaufgaben mit unabhängigen Mitteln zu lösen. Das Multitasking ist die direkte Umsetzung dieses Prinzips im Rahmen programmierbarer Universalrechner.

Forderung 11:
Die S.015-Architektur muß für den Aufbau vielfältig strukturierter Multiprozessorsysteme geeignet sein.
Komplexe Anwendungslösungen lassen sich nur noch mit Multiprozessorkonfigurationen verwirklichen. Dabei sind folgende Strukturen besonders wichtig:

1. verteilte, lose gekoppelte Systeme (vielfältige Verbindungsstrukturen, viele Prozessoren, serielle Verbindungen, Kommunikation über Software-Protokolle)

2. dicht gekoppelte Systeme (starre, einfache Verbindungsstrukturen, wenige Prozessoren, hardwareseitige Steuerung der Kommunikation).

Forderung 12:
Die S.015-Architektur muß zur Zeit ihrer Marktwirksamkeit ein sinnfälliges Ziel darstellen, sowohl für Compiler als auch für die künftigen Schaltkreis-Technologien.

Forderung 13:
Die S.015-Architektur muß eine ausgewogene Kombination innovativer und bewährter Lösungen darstellen. Grundsätzliche Innovationen sind nur dann vorzusehen, wenn hinsichtlich der folgenden Gesichtspunkte keine grundsätzlichen Schwierigkeiten oder Nachteile zu erwarten sind:

- Datenaustausch mit Fremdsystemen
- Programmentwicklung auf Fremdsystemen
- physische Kopplung mit Fremdsystemen bis hin zum unmittelbaren Einbau in diese
- Implementierung bewährter Modelle der Informationsverarbeitung (z. B. solcher, die durch Sprachumgebungen wie Fortran und C gekennzeichnet sind)
- Akzeptanz seitens des Praktikers (das S.015 muß auf der Grundlage des industrieüblichen Fachwissens nutzbar sein).

Eine solche Architektur bildet eine der entscheidenden Grundlagen, um folgendes Szenarium zur Lösung komplexer Steuerungsaufgaben zu verwirklichen:

1. Der gesamte Gebrauchswert wird in einer höheren Programmiersprache entwickelt, z. B. in Ada (unter Nutzung des "package"- Konzepts, der Multitasking- Vorkehrungen usw.); also zunächst rein funktionell spezifiziert.

2. Diese funktionelle Beschreibung wird mit üblichen Entwicklungsumgebungen compiliert und ausgetestet. Entsprechend leistungsfähige Entwicklungsumgebungen könnten in dieser Phase direkt mit der zu steuernden Umgebung gekoppelt werden.

3. Danach wird die Lösung in das konkrete Anwendungssystem umgesetzt, das heißt in eine angemessene Hardware-Software-Kombination auf Grundlage der S.015-Architektur (das könnte ein entsprechender "silicon compiler" weitgehend automatisiert erledigen).

3. Anforderungen an die Systemgestaltung

3.1. Funktionelle Anforderungen

Die Anforderungen an komplexe Steuerungs- bzw. Automatisierungssysteme sind so vielfältig, daß sie mit einer vergleichsweise einseitigen Orientierung auf bestimmte Funktionsabläufe, Operationen und Datenstrukturen nicht erfüllt werden können. Vielmehr ist eine universelle, völlig freizügig programmierbare Auslegung erforderlich. Nachfolgend werden die wesentlichen Funktionseigenschaften dargestellt, die in der S.015-Architektur vorzusehen sind.

3.1.1. Numerische Operationen

Es sind die 4 Grundrechenarten über binär codierte Zahlen vorzusehen. Das betrifft natürliche und ganze Zahlen sowie Zahlen in Gleitkommadarstellung.
In vielen Anwendungen gibt es begrenzte Zahlenbereiche (die z. B. in Ada durch range- Angaben deklariert werden). Daraus ergeben sich 2 Forderungen an die Architekturgestaltung:

1. Es ist wesentlich, auf Bereichsüber- bzw. Unterschreitungen sofort zu reagieren (beispielsweise wenn es sich um Temperaturen, Drücke, Drehwinkel, Drehzahlen usw. handelt). Die S.015-Architektur soll deshalb die Bereichsprüfung und die entsprechende Ausnahmebehandlung unterstützen.

2. In der Praxis können numerische Angaben oft mit natürlichen bzw. ganzen Binärzahlen codiert werden. Die S.015-Architektur muß es gestatten, beliebig lange Bitketten als solche Zahlen zu interpretieren, um diese in kompakter Form (mit so vielen

Bits, wie jeweils notwendig) speichern zu können.

Die Zahlenverknüpfungen sollen gemäß den üblichen Regeln der Mathematik ausgeführt werden (das betrifft die Behandlung von Vorzeichen, Überlauf, Divisionsrest usw.)[1], und sie sollen im Sinne eines Semimorphismus[2] implementiert werden, das heißt, bei jeder beliebigen Verknüpfung im Rahmen der Elementaroperationen darf zwischen dem Resultat und seiner maschineninternen Darstellung kein weiterer Wert der maschineninternen Darstellung liegen. Im Interesse eines überlegenen Gebrauchswertes sollen (wenigstens in den oberen Leistungsbereichen) die wichtigsten Operationsverkettungen des numerischen Rechnens[3] als Semimorphismen implementiert werden oder durch entsprechende Elementarbefehle in Form von Unterprogrammen implementierbar sein. Das betrifft beispielsweise das Skalarprodukt (SDOT) und die Vektor- "Triade" (SAXPY). Werden diese Anforderungen erfüllt, ist an sich jede beliebige Codierung der Zahlendarstellungen zulässig (die Datenstrukturen können nach Maßgabe der Zweckmäßigkeit festgelegt werden). Die Wandlung in allgemein übliche Darstellungsformen muß jedoch gewährleistet sein, das heißt, es muß mindestens gewährleistet sein, daß das S.015 solche Zahlendarstellungen akzeptiert und jenen Elementaroperationen unterziehen kann, die für die wechselseitige Konvertierung notwendig sind. Das betrifft: natürliche und ganze Zahlen in 2er-Komplement- Darstellung, Gleitkommazahlen nach IEEE 754 und gepackte binär codierte Dezimalzahlen.

3.1.2. Boolesche Operationen

Die üblichen bitweisen Verknüpfungen sind vorzusehen (Negation, Konjuktion, Disjunktion, Antivalenz). Die Architektur soll dafür Vorsorge treffen, solche Operationen selektiv für bestimmte Bitpositionen ausführen zu können (d. h. unter Steuerung einer entsprechenden binären Maske). Weiterhin ist vorzusehen, daß ein einziges Bit an alle Positionen eines der Operanden verteit werden kann. Das ist für parallele Verknüpfungen verschiedener Art und im besonderen für die Polynombe-rechnung (z. B. bei fehlererkennenden bzw. korrigierenden Codes) notwendig.
In Steuerungsanwendungen ist das Lösen Boolescher Gleichungen von entscheidender Bedeutung, das heißt die Bestimmung des binären Resultats auf Grund der aktuellen Variablenbelegung. Folgende Prinzipien sind dafür besonders wirkungsvoll:

1. direkte Abbildung durch Zugriff zu gespeicherten Wahrheitstabellen (äußerst schnell, aber nur für geringe Variablenzahlen brauchbar; vielleicht für 10...16 Variable)

---

1 Das betrifft die interne Arithmetik im S.015. Die 2er-Komplement- Arithmetik muß den Konventionen der anwendungspraktisch wichtigsten Fremdsysteme entsprechen.
2 Es ist den theoretischen Erkenntnissen gem. /2/ zu folgen.
3 Erfahrungsgrundlagen: numerische Anwendungen, Befehlslisten von Vektorrechnern.

2. Operatordarstellung durch passsende Befehle ("Bitprozessor")

3. Durchmusterung von Ternärvektorlisten (das ist im Gegensatz zur sequentiellen Interpretation einer Operatordarstellung vollständig parallelisierbar und für komplizierte Verknüpfungen bei größeren Variablenzahlen deutlich überlegen).[1]

Die Architektur muß es gestatten, alle 3 Prinzipien zu implementieren, so daß im konkreten Anwendungsfall das jeweils zweckmäßigste Prinzip gewählt werden kann. Das 1. Prinzip bedeutet, alle Variablenbelegungen vorab zu ermitteln und zu speichern. Die aktuelle Variablenbelegung wird dann als Bitadresse für den jeweiligen Wahrheitswert verwendet. Die Architektur muß es gestatten, solche Bitadressen auf einfache Weise aus den jeweiligen Variablenwerten zusammenzustellen. Das 2. und 3. Prinzip sollte für wenigstens 4096 Variable anwendbar sein. Das Durchmustern von Ternärvektorlisten (der "Test auf Orthogonalität") muß in der Hardware unmittelbar unterstützt werden. Man wird diese Datenstruktur verwenden, wenn es um besonders komplizierte Boolesche Gleichungen geht. Wegen der exponentiellen Abhängigkeit der Lösungszeit von der Variablenzahl müssen alle technisch sinnfälligen Möglichkeiten zur Ablaufbeschleunigung genutzt werden (die Aufwendungen für ausgesprochene Sonderhardware sind vergleichsweise unerheblich). Zudem ist der Durchmusterungsablauf als Grundlage leistungsbestimmender Operationen über relationale (in Tabellenform gespeicherte) Datenstrukturen nutzbar (Join, Intersection usw.). Im Interesse der Vielseitigkeit muß die Architektur Listen in modifizierter, speicherplatzsparender Form zulassen (z. B. als Mischung von binärere und ternärer Codierung).

Weitere anwendungspraktisch bedeutsame Funktionen sind:

1. Abfragen und Modifizieren von Einzelbits, aber auch von kurzen Bitfeldern (4...8 bit Länge). Steuerzustände werden oft in Einzelbits codiert, weil das zur gleichsam "naiven" Art und Weise gehört, wie aus der Anschauung heraus ein Problem verstanden und in Programmabläufe umgesetzt wird. Die Einzelbits können oft sinnvoll in Gruppen angeordnet werden, und es kommt häufig vor, daß in einer solchen Gruppe bestimmte Bits gleichzeitig ein- und auszuschalten sind.

2. Wenigstens eine Operation der Art "Test and Set" (für Synchronisationszwecke).

3. Die Ermittlung der niedrigstwertigen Eins in einer Bitkette. Das ist für Prioritätsabfragen wesentlich. Die Abfrage ist häufig mit dem Löschen der betreffenden Bitposition verbunden. Eine manchmal nützliche Abwandlung besteht darin, eine Resultat-Bitkette zu erzeugen, die nur noch die niedrigstwertige Eins enthält, und die betreffende Position in der Argument-Bitkette zu löschen.

---

1 Steuerungspraktische Anwendungen sind in /6/ dargestellt.

4. Die Ermittlung der Anzahl der Einsen in einer Bitkette ("Quersumme"). Ein Sonderfall dieser Operation ist die Paritätsprüfung (es wird ermittelt, ob die Quersumme gerade oder ungerade ist).

5. Die Ermittlung der höchstwertigen Eins in einer Bitkette, verbunden mit deren Löschung. Eine solche Funktion kann beispielsweise zur Beschleunigung arithmetischer Verknüpfungen (im besonderen Multiplikation und Division) genutzt werden.

3.1.3. Zeichenkettenoperationen

Zeichenkettenbewegungen und -umformungen sind mit entsprechenden Transport- bzw. Auswahloperationen auszuführen. Darüber hinaus sind Zeichenkettenvergleiche wesentlich. Die entscheidende Elementaroperation besteht im Feststellen der Gleichheit bzw. Ungleichheit zweier gleich langer Zeichenketten (ergänzend dazu ist die Möglichkeit zu schaffen, ungleich lange Zeichenketten für diesen Vergleich aufzubereiten). Alle komplexeren Vergleichs- bzw. Durchmusterungsabläufe sind auf diese Elementaroperationen zurückzuführen (mit entsprechender Adressenrechnung, Schleifensteuerung usw.). Die S.015-Architektur darf nicht von einem besonderen Zeichencode abhängen. Elementare Vergleiche müssen durch Hardware unterstützt werden; für komplexere Operationen sollte es die Architektur gestatten, Adressenrechnungen usw. parallel auszuführen.

3.1.4. Transport- und Auswahloperationen

Transporte beliebiger Datenstrukturen ohne weitere Informationswandlungen (Blocktransporte) sind durch Hardware zu unterstützen. Das gilt auch für das Auswählen bzw. selektive Einfügen von Bitfeldern. Die S.015-Architektur muß dies unmittelbar unterstützen, um in den Anwendungsprogrammen die vielen Befehle zu ersparen, die ansonsten notwendig sind, um aus dicht gepackt gespeicherten Bitfeldern die Verknüpfungs-Operanden aufzubereiten und die Resultate wieder entsprechend zu verpacken. Bild 3 veranschaulicht anhand eines einfachen Beispiels (der Daten, die zur Steuerung eines Druckers erforderlich sind) , was beabsichtigt ist: Oft bestehen die Daten eines Steuerprogrammes aus vergleichsweise wenigen Einzelbits und kurzen Bitfeldern, die kompakt gespeichert werden können (beispielsweise in Universalregistern). Eine solche Speicherbelegung erfordert aber bei den meisten Architekturen zusätzliche Befehle (hauptsächlich konjunktive und disjunktive Verknüpfungen sowie Verschiebungen), um die kompakt gespeicherten Angaben zur Verarbeitung aufzubereiten und die Resultate wieder einzufügen.

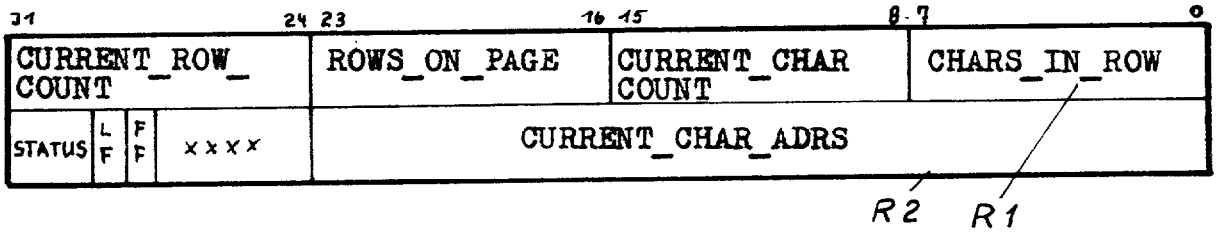
3.1.5. Adressierung

Die S.015-Architektur soll eine Adressierung bis aufs Bit vorsehen. Es sollen variabel lange Bitketten adressierbar

a) Deklaration der Datenstruktur (in Ada)

```
type PRINTER_CONTROL is record
     CHARS_IN_ROW:          INTEGER range 0..255;
     CURRENT_CHAR_COUNT:    INTEGER range 0..255;
     ROWS_ON_PAGE:          INTEGER range 0..255;
     CURRENT_ROW_COUNT:     INTEGER range 0..255;
     STATUS:                (READY, OUT_OF_PAPER, OFFLINE);
     LINE_FEED:             BOOLEAN;
     FORMS_FEED:            BOOLEAN;
     CURRENT_CHAR_ADDRS:    POINTER; -- 24 bit adrs
end record;
```

b) Kompakte Speicherung in 2 32-bit-Registern



c) **Verarbeitung mit elementaren Befehlen**
(Beispiel: Erhöhen von CURRENT_CHAR_COUNT)

```
LOAD IMMEDIATE Ra, X`0000FF00` -- Maske nach Hilfsreg. a
AND Ra, R1
LOAD IMMEDIATE Rb, X`00000100` -- Additionskonstante nach
ADD Ra, Rb                     -- Hilfsregister b
...weitere Befehle, z. B. für
   Vergleich mit CHARS_IN_ROW
LOAD IMMEDIATE Ra, X`FFFF00FF` -- Maske zum Rückschreiben
OR R1, Ra                      -- Rückschreiben
```

Bild 3 **Probleme der Behandlung kompakt gespeicherter Datenstrukturen in herkömmlichen Architekturen**

sein. Die Adressenangaben sollen jeweils so kompakt wie möglich codiert werden. Theoretisch sind für eine Auswahl unter n Angaben nur ceil (ld n) Adressenbits notwendig. Eine solche Codierung ist praktisch kaum zu verwirklichen. Die S.015-Architektur soll aber dem jeweiligen theoretischen Wert in angemessener Weise nahekommen.
Die Adressen für technische Speichermittel sollen auf einfache Weise gebildet werden. Kompliziertere Formen der Adressenumsetzung (z. B. virtuelle Speicher mit "paging"- Organisation) sind auf spezifische Adapterschaltungen auszulagern. Alle technischen Speichermittel sollen programmseitig direkt erreichbar sein.

3.1.6. Ein- und Ausgabe

Eine entscheidende Anforderung an die S.015-Architektur besteht darin, "Restlogik" zur Verbindung der S.015-Prozessoren mit der jeweils zu steuernden Umgebung weitgehend entbehrlich zu machen, sie in den meisten Anwendungsfällen völlig zu vermeiden. Dazu sind die ASIC- Anschlüsse so zweckmäßig zu belegen, daß sich die unumgänglichen Koppelstufen auf einfachste Weise anschließen lassen.

Für die Mehrfachnutzung von Anschlüssen gilt grundsätzlich:

Eingänge sind nicht mehrfach nutzbar, es sei denn, es werden externe Auswahlmöglichkeiten vorgesehen (Bild 4). Das erfordert aber zusätzliche Schaltmittel. Zudem sind nur die Eingänge der Verarbeitung zugänglich, die jeweils ausgewählt sind.

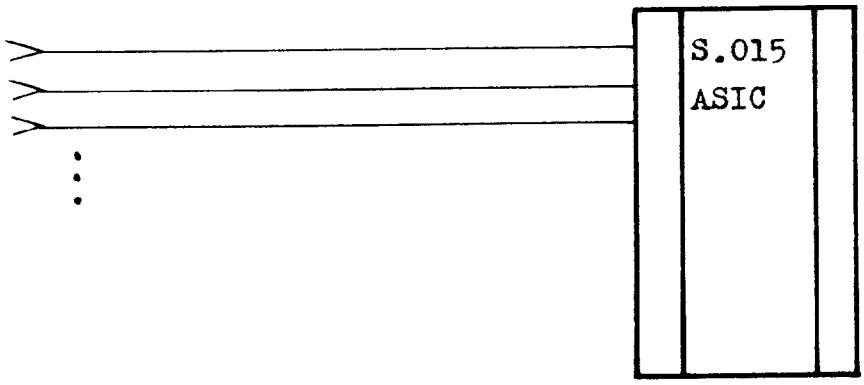
Ausgänge sind mehrfach nutzbar, sofern sie Datensignale führen, die zur Übernahme in externe Speichermittel, z. B. Register, bestimmt sind. Dafür sind zusätzliche Übernahmesignale zu liefern (Auswahl der jeweiligen Bestimmung, Taktimpulse). So können beispielsweise die an sich vorhandenen Adressen- und Datenanschlüsse, die zu den Speichermitten führen, als Datenleitungen benutzt werden (Bild 5).

Wesentliche Anforderungen im Sinne von Flexibilität und Leistungsvermögen sind:

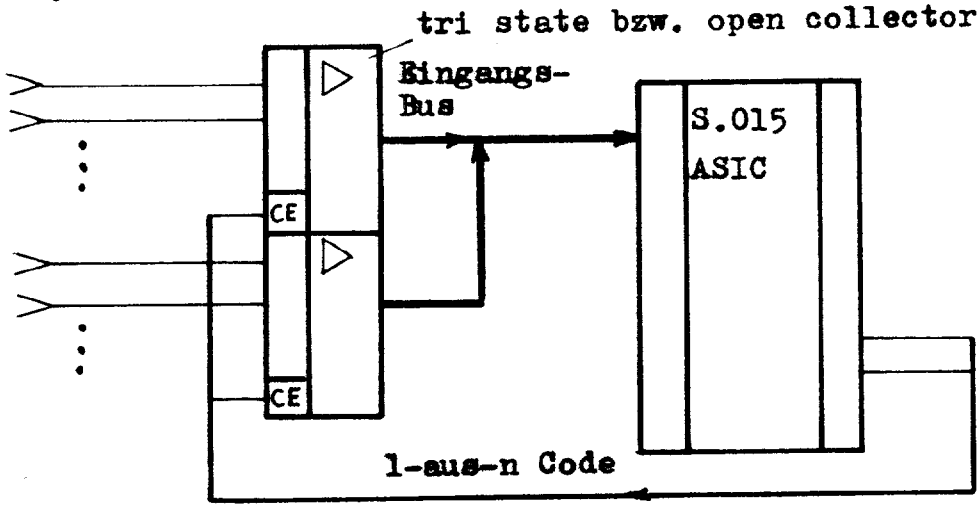
1. Möglichst viele Ausgangssignale müssen gleichzeitig erregbar und möglichst viele Eingänge gleichzeitig abfragbar sein. Eingänge erfordern oft eine Pegelwandlung, Ausgänge eine Leistungsverstärkung. Die Mehrfachnutzung von Ein- und Ausgängen soll weitgehend auf die entsprechenden Schaltmittel verlagert werden (das erfordert z. B. Pegelwandler mit tri-state-Ausgängen und selektiv ladbare Pufferregister vor Treiberstufen). Dafür müssen Auswahl- bzw. Ansteuersignale in praxisgerechter Form bereitgestellt werden (binär codiert, 1-aus-n-Code usw., je nach Sinnfälligkeit).

2. Der aktuelle Erregungszustand der Ausgänge muß programmseitig abfragbar sein, so daß diese Belegungen unmittelbar in Test- und Verknüpfungsoperationen (wie Erhöhen, Vermindern

a) Direktanschluß



b) Buskopplung



c) Auswahlschaltung

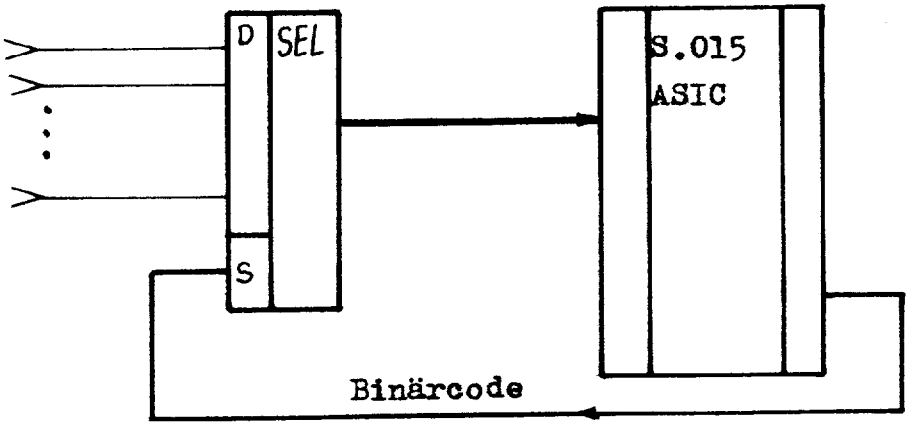


Bild 4 Alternativen der Eingangsbeschaltung

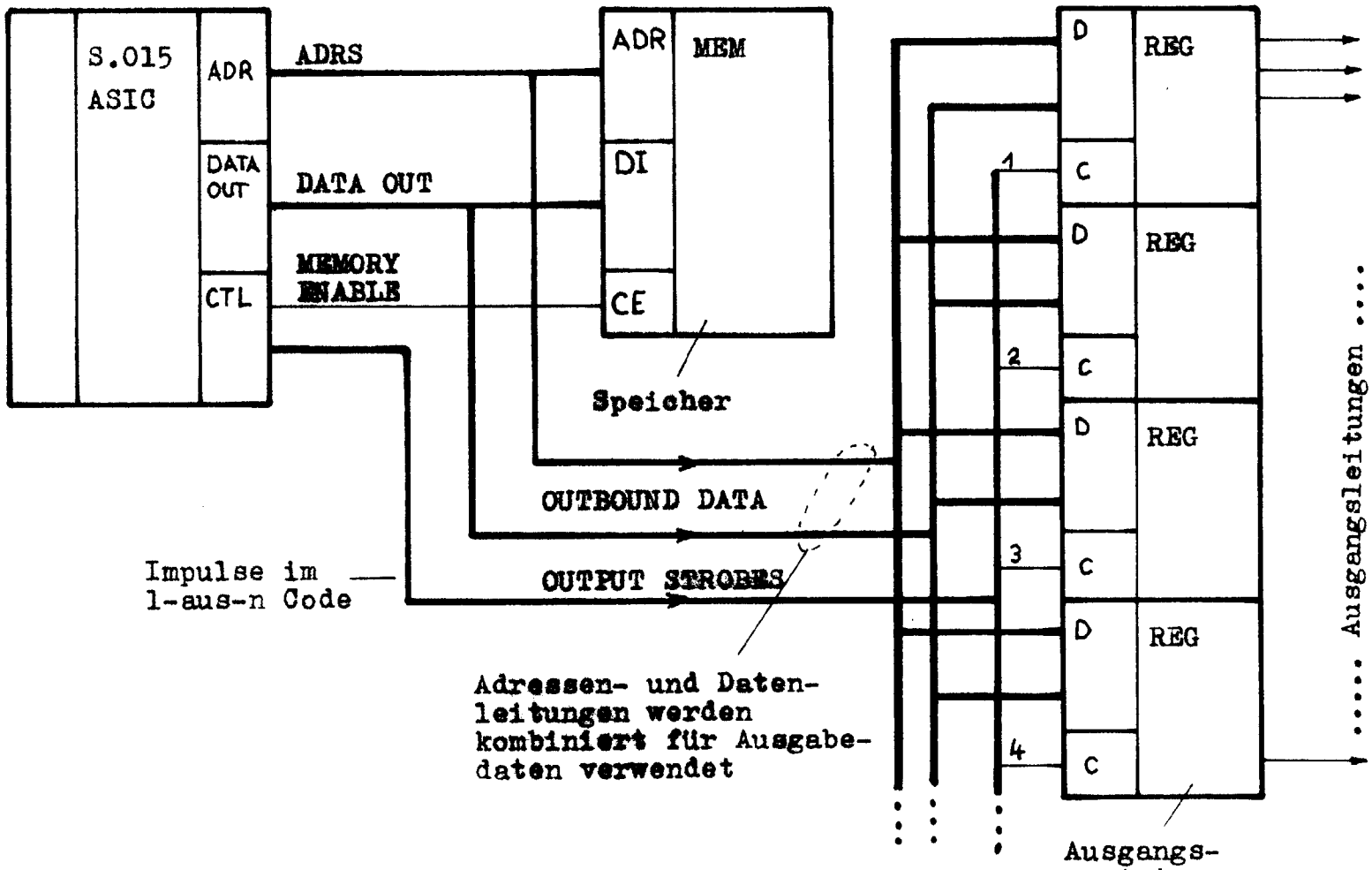


Bild 5 Beispiel für die Anschaltung von Ausgangsleitungen

usw.) einbezogen werden können. Das darf aber keine externen Schaltmittel zum Rücklesen der Belegungen erfordern.

3. Die Ausgangs-Anschlüsse des ASIC müssen für folgende Informations-Arten konfigurierbar sein:

a) Daten, die in externe Register übernommen werden (die Belegung ist nur in Zusammenhang mit einem zugehörigen Strobe-Impuls gültig)

b) statische Daten

c) selektive Strobe-Impulse, die durch interne Verknüpfungen von Datenbit-Positionen mit internen Strobes gebildet werden. So können "fertige" Strobe- Impulse im 1-aus-n- Code als Daten-Übernahmesignale geliefert werden, wodurch externe Decoder überflüssig sind (zudem wird es so möglich, gleiche Datenbelegungen gleichzeitig an mehrere Bestimmungen zu liefern, z. B. in Rücksetzabläufen).

d) interne Signale (z. B. Strobe- Impulse) des Prozessors zur externen Nutzung (z. B. zu Synchronisationszwecken)

e) Angaben für die Auswahl von Eingängen (in jeweils passender Codierung, z. B. binär oder 1-aus-n; für tri-state- Auswahl mit Austastlücke beim Umschalten; erforderlichenfalls mit besonderer Strobe- oder Bereitschaftsleitung).

4. Viele Eingangssignale müssen schnell und flexibel abfragbar sein, um kurze Reaktionszeiten für die programmseitige Verarbeitung zu gewährleisten. Es müssen Synchronisations- und Auffangschaltungen konfigurierbar sein, die vom Programmablauf unabhängig sind und den Realzeitbedingungen der zu steuernden Umgebung gerecht werden (z. B. extern triggerbare Halteregister). Wünschbar ist eine gewisse Vorverarbeitung von Eingangsbelegungen unabhängig vom Programmablauf. Entsprechende Schaltungen dürfen aber nicht zu sehr spezialisiert sein.
Dafür bieten sich folgende Prinzipien an:

a) programmierbare Logik (PLA-Schaltungen mit ladbarer Verknüpfungsbestimmung)

b) unmittelbare technische Umsetzung automatentheoretischer Vorstellungen, wobei die Zustands- und Ausgangszuordner mit ROM- bzw. RAM- Anordnungen implementiert werden

c) Beschreibung des Automatenverhaltens in Form einer Phasenliste[1], die gespeichert und von entsprechenden Schaltmitteln durchmustert wird. Die Phasenliste enthält für jede Kombination von Zustand und Eingangsbelegung den Folgezustand und die Ausgangsbelegung. Der Umfang solcher Listen kann durch folgen-

---

1 Zur Theorie siehe /1/, zu steuerungspraktischen Einzelheiten siehe /6/.

de Maßnahmen auf beherrschbare Größenordnungen gebracht werden:

- Die aktuellen Zustände werden nicht mit gespeichert; vielmehr wird für jeden Zustand eine Teilliste angelegt (dazu gehört noch eine Zeigerliste für alle Zustände).

- Die Eingangsbelegungen werden ternär codiert.

d) Beschreibung des Automatenverhaltens durch Boolesche Gleichungen, die durch kombinierte Anwendung programmseitiger und schaltungstechnischer Vorkehrungen interpretiert werden (vgl. Abschnitt 3.1.2.)

e) übliche Rechnerprogramme (auf heuristische Weise erstellt).

Alle Prinzipien müssen - je nach Zweckmäßigkeit - anwendbar sein.
Die Prinzipien a) und b) sind nur für beschränkte Variablenzahlen nutzbar, sie haben aber eine definierte (kurze) Ausführungszeit, die von der aktuellen Variablenbelegung unabhängig ist. Werden die Zustandsübergänge eines Steuerautomaten durch sequentielle Abläufe ermittelt, so hängt die Ausführungszeit von der Variablenbelegung ab. Das Prinzip c) liegt dabei zwischen der unmittelbaren Zuordnung und der programmseitigen Berechnung. Es ist üblichen Programmabläufen leistungsmäßig deutlich überlegen, sofern der Umfang der Phasenlisten akzeptabel ist und das Durchmustern schaltungstechnisch unterstützt wird.

5. Um komplexere Systeme konfigurieren zu können, sind serielle Kommunikationswege und entsprechende Steuerschaltungen vorzusehen. Die Kommunikationsprotokolle sollten allgemein akzeptierten Standards entsprechen bzw. wenigstens Teilmengen solcher Standards. Ist dies auf Grund der Markt- und Schutzrechtslage nicht durchführbar, wird ein eigener Standard definiert. (Man kann sich in der Praxis eines so aufwendigen Entwicklungsvorhabens, wie es das S.015 darstellt, nicht ausschließlich auf Standards stützen, die im Entwicklungszeitraum einem unbeeinflußbaren Änderungsgeschehen unterworfen sind.) Es ist aber für eine problemlose Konvertierbarkeit Sorge zu tragen: möglichst viele Schichten des Kommunikationsprotokolls sind in Software oder leicht änderbarer Firmware zu implementieren.

3.1.7. Parallelarbeit

Jeder S.015-Prozessor muß die Gelegenheit bieten, mehrere unabhängige Programme parallel bzw. zeitlich verschachtelt abzuarbeiten (Multitasking). Für eine gewisse Zahl von Tasks soll das Umschalten zwischen den Tasks hardwareseitig unterstützt werden (Anzahl: 2...16).

Bemerkungen:
1. Die Anzahl der Tasks ergibt sich aus Erfahrungswerten und Abschätzungen. Mit n Tasks hat man praktisch n virtuelle Prozessoren, deren Verarbeitungsleistung jeweils etwa das 1/n-fache der Leistung des realen Prozessors beträgt.
2. Die Haupanwendung dieser virtuellen Prozessoren besteht darin, spezielle Ein- und Ausgabeschaltungen zu ersetzen. Dazu muß der einzelne virtuelle Prozessor aber eine gewisse Mindest-Leistungsfähigkeit haben.
3. Eine weitere sinnvolle Anwendung besteht darin, Tasks für die Behandlung von Unterbrechungen, Ausnahmebedingungen und Maschinenfehlern zu reservieren. Dann wird der eigentlichen Verarbeitung praktisch keine Laufzeit entzogen (eine solche Task wird nur aktiv, wenn die jeweilige Bedingung auftritt). Für derartige Zwecke werden aber nur wenige Tasks benötigt (im einfachsten Fall sind nur eine Nutzer-Task und eine Sonderfall-Task erforderlich).
4. Jede Task braucht Speichermittel (Register) auf dem ASIC. Große Registeranordnungen belegen aber viel Siliziumfläche und erfordern einen längeren Maschinenzyklus. Somit ist es nicht sinnvoll, das hardware-gestützte Multitasking über die angegebene Größenordnung hinaus auszudehnen.

Es sind Organisationsformen der Umschaltung vorzusehen, die es gestatten, das Realzeitverhalten aller Tasks exakt vorauszusagen. Beispiel: zeitgesteuertes Umschalten ("time slicing"). Darüber hinaus muß es die Architektur gestatten, softwareseitig ein unbeschränktes Multitasking zu organisieren.

Bis zu 4 Prozessoren müssen sich zu einem Verbund zusammenfassen lassen, der anwendungsseitig wie ein Prozessor für maximal 64 Tasks erscheint, wobei bis zu 4 Tasks gleichzeitig Laufzeit zugeteilt sein kann.

Die Beschränkung auf 4 Prozessoren hat folgende Gründe:

1. Bei den voraussichtlichen Aufwendungen eines S.015-Prozessors sind 4 Prozessoren ein sinnfälliges Ziel für die Integration auf einem Schaltkreis.

2. In einer solchen Konfiguration ist die unmittelbare Kopplung, z. B. von Registerblöcken und Speicheradaptern, technisch beherrschbar und führt nicht bzw. kaum zu Leistungsverlusten.

3. In dieser Größenordnung ist die Organisation der Parallelverarbeitung noch überschaubar.

Es muß möglich sein, einem S.015-Prozessor ("Master") andere S.015-Prozessoren als E/A- ("Slave")- Prozessoren nachzuordnen. Für die Kommunikation zwischen Master- und Slave- Prozessoren soll das gleiche Software-Protokoll vorgesehen sein wie für die Kommunikation seriell gekoppelter S.015-Prozessoren (für die Anwendungssoftware muß es völlig transparent sein, wie die Prozessoren physisch gekoppelt sind).

Die Architektur muß dafür Vorsorge treffen, daß Hochleistungs-Implementierungen geschaffen werden können, die - im Einzelprozessor - mehrere Operationen parallel ausführen. Dabei ist es nicht erforderlich, den Parallelismus zur Laufzeit zu erkennen (diese Aufgabe kann den Compilern übertragen werden).

3.1.8. Unterbrechungssteuerung

Mittel und Wirkprinzipien der Unterbrechungssteuerung sind für kurze Reaktionszeiten auszulegen. Für deren Implementierung ist vorzugsweise die Taskumschaltung zu nutzen. Die Unterbrechungssteuerung darf, über die unmittelbare Auslösung hinaus, keine besonderen externen Schaltmittel (zur Prioritätsfeststellung, zum Liefern des jeweiligen Interrupt-Vektors usw.) erfordern.

3.1.9. Ausnahmebehandlung

Es ist die automatische Behandlung von Sonderbedingungen vorzusehen, die während des Programmablaufs auftreten, z. B. Überläufe, Bereichsüber- bzw. -unterschreitungen usw. Dabei muß die auslösende Ursache für das auswertende Programm zugänglich sein. Bei bestimmten Operationen muß eine unmittelbare Verbindung zum jeweiligen spezifischen Behandlungablauf herstellbar sein, so daß Zeitverluste, beispielsweise durch das Rufen eines allgemeinen Ausnahmebehandlers, vermieden werden.

3.1.10. Maschinenfehlerbehandlung

Die Korrektheit der statischen Objektprogramme ist durch Compiler bzw. anderweitige Entwicklungshilfen zu gewährleisten. In der Hardware sollen zur Laufzeit nur 2 Überwachungsmaßnahmen vorgesehen werden:

1. Kontrolle von Speicherinhalten (durch Paritätsbits oder Fehlerkorrekturcodes)

2. allgemeine Zeitkontrollen ("watchdog") für Programmabläufe und externe Reaktionen.

Weitere Überwachungsschaltungen für die Hardware sind nicht vorzusehen; stattdessen wrid bei kritischen Anwendungen die Duplikation ("Master-Checker- Prinzip") bzw. Triplikation des kompletten Prozessors bevorzugt.

Alle Kontrollmaßnahmen müssen programmseitig steuer- und auswertbar sein.

Es ist dafür Sorge zu tragen, daß die korrekte Arbeit eines S.015-Prozessors extern mit einfachen Mitteln überwacht werden kann.

Besonderer Wert ist auf die Testbarkeit der S.015-Hardware zu legen (Abarbeiten von Testfolgen nach dem Rücksetzen, Nutzung des "scan/set"- Diagnosepfades des ASIC usw.). Diese Vorkehrungen müssen neben den Prozessorkernen auch die verschiedenen Adapter, Speichermittel und Zusatzschaltungen umfassen. Bedienung und Fehleranzeige sind so zu organisieren, daß dafür industrieübliche Mittel (z. B. RS 232- Terminals) verwendbar sind. In größeren S.015-Konfigurationen soll es möglich sein, wenigstens einen Prozessor als Prüfprozessor zu konfigurieren.

3.1.11. Wartungs- und "debugging"- Hilfen

Es muß möglich sein, Pogrammabläufe zu verfolgen. Entsprechende Schaltmittel dürfen extern vorgesehen werden, sofern der Anschluß die normale Anwendungsumgebung nicht beeinträchtigt und in der Handhabung praktikabel ist.

Zum Einbringen von Programmänderungen gibt es grundsätzlich 2 Möglichkeiten:

1. Die Anwendungsprogramme werden stets aus RAM-Bereichen abgearbeitet. Dann ist das Einbringen der Änderungen eine Angelegenheit des Anfangsladeprogramms. Wird aus ROM-Bereichen geladen, so ist zu gewährleisten, daß ein üblicher ROM-Schaltkreis ohne weiteres einzeln gelesen werden kann (z. B. byteweise). Dieser Schaltkreis nimmt die Änderungen in der Form "Adresse - Länge - neuer Inhalt" auf. Diese Angaben können vom Anfangsladeprogramm zum Einbringen der Änderungen genutzt werden.

2. Die Programme werden aus externen ROM-Bereichen abgearbeitet (das ist vorzugsweise in kleineren Konfigurationen der Fall). Es ist dann zu geährleisten, daß handelsübliche "ROM-Patches" ohne weiteres angeschlossen werden können.

Um fehlerhafte Programmteile von der Verarbeitung ausschließen bzw. durch geänderte Programmstücke ersetzen zu können, sind folgende Vorkehrungen bei der Befehlsgestaltung zu treffen:

1. ein Verzweigungsbefehl mit vollständiger Absolutadresse des Sprungzieles, unabhängig von veränderlichen Registerinhalten der jeweiligen Task

2. ein entsprechender Unterprogrammaufruf

3. ein NO OP- Befehl.

Diese Befehle dürfen den Maschinenzustand nicht verändern (z. B. keine Bedingungscodes beeinflussen).

3.2. Datenstrukturen

Es ist wesentlich, daß die Datenstrukturen mit Fremdsystemen verträglich sind bzw. auf einfache Weise wechselseitig gewan-

delt und ausgetauscht werden können. Als grundlegende Datenstruktur wird deshalb das 32-bit- Wort (im folgenden als Wort bezeichnet) vorausgesetzt. Innerhalb des S.015 werden - im Rahmen der Wort-Organisation - die Datenstrukturen ausschließlich nach Gesichtspunkten der Zweckmäßigkeit, also ohne Rücksicht auf Fremdsysteme und -konventionen, festgelegt. Die S.015-Architektur muß dafür Vorsorge treffen, daß alle intern zu speichernden Angaben in kompakter Form codiert werden können (es sollen jeweils nur soviele Bits belegt werden, wie unbedingt notwendig), und sie muß die erforderlichen Informationswandlungen für den Datenaustausch unterstützen.

3.2.1. Numerische Datenstrukturen

1. Natürliche und ganze Binärzahlen
Grundlage aller arithmetischen Operationen ist das 32-bit-Wort. Natürliche Zahlen umfassen 32 Binärstellen, ganze Zahlen 31 sowie das Vorzeichen. Die Architektur muß es ermöglichen, beliebig kurze Bitketten als natürliche bzw. ganze Binärzahlen zu interpretieren, wobei jede Operanden-Bitkette auf das Wortformat erweitert wird. Aus den Resultat-Worten müssen sich beliebige Bitketten zuweisen lassen.
Die Architektur muß es zulassen, natürliche und ganze Zahlen mit mehr als 32 Binärstellen zu verarbeiten.

2. Gleitkommazahlen
Solche Zahlen sind erforderlich:

a) für schnelle Rechnungen vergleichsweise geringer Genauigkeit über einen großen Zahlenbereich. Dafür wird das 32-bit-Format nach IEEE 754 vorgesehen.

b) für genaue Rechnungen über einen großen Zahlenbereich. Dafür wird intern ein 96-bit-Format vorgesehen. Dieses entspricht dem 80-bit-Format nach IEEE 754, ist aber um 16 Mantissenstellen erweitert, so daß 3 32-bit-Worte voll ausgenutzt sind.

Die Architektur muß es darüber hinaus erlauben, das 64-bit-Format nach IEEE 754 zu verarbeiten und Gleitkommadarstellungen in extrem lange ganze Binärzahlen zu wandeln.

3. Binär codierte Dezimalzahlen
Es werden 32-bit-Worte unterstützt, die gepackte binär codierte Dezimalzahlen enthalten. Dies betrifft nur die Addition bzw. Subtraktion und ist vorwiegend für Zwecke der Informationswandlung bzw. des Datenaustauschs vorgesehen.

Bemerkung:
Es wird empfohlen, Dezimalzahlen als rationale Zahlen der Form a + b/c darzustellen, wobei a, b, c ganze bzw. natürliche Binärzahlen sind. Diese werden in den meisten S.015-Implementierungen in 32 Stellen parallel verarbeitet, Implementierungen des oberen Leistungsbereichs gestatten überdies die Parallelausführung mehrerer Operationen.

3.2.2. Boolesche Datenstrukturen

Die elementare Datenstruktur ist der Binärvektor. Die Architektur muß es gestatten, sehr lange Binärvektoren zu verarbeiten (wenigstens 4096 bit). Im besonderen müssen kurze Binärvektoren von 1...32 bit Länge unterstützt werden.
Darüber hinaus muß die Architektur für ternäre Strukturen (Variable, Vektoren und Listen) eingerichtet sein.

3.2.3. Zeichenketten

Die S.015-Architektur muß Zeichencodes beliebiger Länge zwischen 1 und 32 bit zulassen. Daraus müssen sich Zeichenketten durch Aneinanderreihung bilden lassen, und zwar ohne Rücksicht auf Wortgrenzen.

3.2.4. Beschreibende Datenstrukturen

Beschreibende Datenstrukturen sind Adressen, Längenangaben usw. Sie treten in Form von Registerinhalten, Steuerworten bzw. -tabellen und als Teile von Befehlen auf. Sie müssen folgenden Kriterien genügen:

1. Sie müssen eine reguläre, übersichtliche Struktur aufweisen.

2. Erweiterungsvorkehrungen müssen vorgesehen sein.

3. Sie müssen so gestaltet sein, daß sie durch Programme und Schaltmittel effektiv interpretiert werden können.

4. Es muß möglich sein, alle Datenstrukturen bis aufs Bit zu adressieren.

5. Technische Mittel, wie z. B. Adapterschaltungen, müssen auf einfache und reguläre Weise programmseitig zugänglich sein.

6. Adressen sollen grundsätzlich als natürliche Binärzahlen codiert werden.

Bemerkung:
Eine Konsequenz besteht darin daß es keine vorzeichenbehafteten Offsets gibt; verschiebliche Adressen sind somit nur in der Form "Basis + Displacement" darstellbar (als Basisadresse ist die niedrigste Adresse der jeweiligen Informationsstruktur zu verwenden).

7. Die S.015-Architektur soll das Auswählen allgemeiner Informationsstrukturen über Ordinalzahlen unterstützen. Das heißt, es muß möglich sein, in einer geordneten Menge von Informationsstrukturen jede einzelne durch Angabe ihrer Ordinalzahl zu selektieren (z. B. das einzelne Zeichen in einer Zeichenkette).

Bemerkungen:
1. Das ist die Grundlage der objektorientierten Zugriffsorganisation.
2. Ordinalzahlen sind oft deutlich kürzer als Adressenangaben (sie betreffen nur die jeweilige Menge, nicht den gesamten Speicheradressenraum).
3. Auf einer solchen Grundlage lassen sich weitere Adressierungsschemata aufbauen, u. a. auch eine Byteadressierung.

3.3. Befehlsgestaltung

Die Befehle sind so zu gestalten, daß die erforderlichen Steuer- und Auswahlangaben (Operationscodes, Adressen usw.) kompakt gespeichert werden können, das heißt, daß jeder einzelne Befehl möglichst viele zur Lösung der Anwendungsaufgabe unmittelbar dienende Angaben enthält. Die der Datenkompression oft entgegenstehende, aber grundsätzlich ebenso wichtige Forderung ist die nach effektiven Schaltungsstrukturen: die Befehlsformate müssen so gestaltet sein, daß sich beherrschbare Schaltungsstrukturen zu deren Interpretation angeben lassen, die kurze Zykluszeiten ermöglichen (die "Schaltungstiefe" der Netzwerke darf nicht zu groß werden). Es ist mithin zwischen Datenkompression (d. h. "Inhalt" der Befehle) und Zykluszeit ein Optimum zu finden, wobei - im Vergleich zu üblichen Architekturen - eher eine größere Schaltungstiefe zu bevorzugen ist (s. Forderung 7 in Abschnitt 2; d. h. die Optimierung kann auf anwendungspraktisch sinnfällige Zykluszeiten bezogen werden, wobei anzustreben ist, die technologisch mögliche Schaltungstiefe sinnvoll zu nutzen). Die Befehle sind so zu gestalten, daß bereits mit größenordnungsmäßig 256...4096 Befehlen nichttriviale Anwendungsaufgaben lösbar sind; elementare Funktionen der Ein- und Ausgabe bzw. innerste Schleifen leistungsbestimmender Abläufe sollten mit 4...8 Befehlen formuliert werden können.

Bemerkung:
256...4096 Befehle können in technologisch ohne weiteres beherrschbaren schaltkreisinternen Speichern untergebracht werden, 4...8 Befehle sogar in Registern. So lassen sich für E/A-Aufgaben optimierte Prozessoren implementieren, deren Anschlüsse praktisch vollständig für Ein- und Ausgabezwecke belegbar sind. Des weiteren kann die Verarbeitungsleistung erhöht werden, weil in leistungsbestimmenden Abläufen das Befehlslesen entfallen kann.

Man kann davon ausgehen, daß Programme für das S.015 in der Regel mit Compilern oder anderen Entwicklungshilfen erstellt werden. Es ist deshalb nicht notwendig, daß die Befehlsformate Bequemlichkeiten für Assembler-Programmierer bieten. Im Interesse einer effektiven Compilierung sollen aber alle Steueranagben auf sinnfällige Weise in den Befehlsformaten "verpackt" sein. (Aus logischer Sicht ist eine reguläre Befehlsgestaltung anzustreben, für die bitweise Codierung braucht man hingegen nur die Hardware-Struktur zu berücksichtigen.)

Die Befehlsformatierung muß sich in die Struktur der Maschinenworte einpassen.
Es ist zu berücksichtigen, daß der gesamte Befehlsvorrat in Teile (Subsets) zerlegbar ist. Für alle S.015-Implementierung ist eine einheitliche elementare Befehlsliste vorzusehen (RISC.015). Namentlich in kleineren S.015-Implementierungen muß es möglich sein, komplexere Befehlswirkungen zu emulieren. Dafür können folgende Methoden eingesetzt werden:

1. Der komplexe Befehl verbleibt im Programm. Gelangt er zur Wirkung, wird automatisch eine Unterprogramm aufgerufen.

2. Der Compiler fügt "inline"- Code oder Unterprogrammaufrufe ein, um fehlende Hardwarefunktionen zu ersetzen.

Entsprechende Vorkehrungen betreffen die Operationscodes, die Unterprogramm-Aufrufprinzipien, die programmseitige Zugänglichkeit (Abfragbarkeit) von Befehlen, die Registernutzung und die Organisation der Bedingungsabfrage.

Die Befehlsgestaltung ist auf Grundlage des Ressourcen_Paradigma[1] auszuarbeiten, muß aber von bewährten Konzepten ausgehen.
Gemäß dem Ressourcen-Paradigma bestimmen die Hardware-Ressourcen, also die Verarbeitungsschaltungen, Speichermittel, Datenwege usw. entsprechend ihrer Gestaltung und Anzahl in erster Linie die Verarbeitungsleistung. Befehle sind codierte Steuerangaben für die zeitsequentielle Nutzung der Ressourcen. In diesem Sinne müssen sie so gestaltet sein, daß mehrere und in sich leistungsfähige Ressourcen soweit wie möglich mit nützlicher, also zur Lösung der Anwendungsaufgabe unmittelbar beitragender, Arbeit beschäftigt werden können.

Bemerkung:
Die Ressourcen-Ausstattung der S.015-Implementierungen des oberen Leistungsbereichs muß über herkömmliche Mikroprozessoren hinausgehen, da nur so die erforderliche Überlegenheit zu erreichen ist (das bedeutet z. B. mehrere Verarbeitungswerke, schaltungstechnische Implementierung komplexer, leistungsbestimmender Operationen usw.).

Die bewährten bzw. aus der Erfahrung heraus wünschenswerten Konzepte betreffen:

1. Universalregisterprinzip. Eine gewisse Anzahl von Universalregistern (wenigstens 16) in Wortbreite sollte für jede Task zur Verfügung stehen. Die feste Belegung bestimmter Register (z. B. als Stackpointer) ist dabei zulässig.

2. sequentielle Befehlsadressierung (durch Befehlszählung)

3. Verzweigungen und Unterprogrammaufrufe mit hinreichender Sprungdistanz (bzw. mit Sprungziel im gesamten Speicheradressenraum)

---

1 Zu den Grundlagen siehe /4/ bzw. /5/.

4. gekoppelte Operations- und Verzweigungs- bzw. Übersprungbefehle mit kurzer Sprungdistanz (es kommt oft vor, daß auf Grund eines Verknüpfungsresultats nur wenige Folgebefehle zu überspringen sind; das betrifft beispielsweise viele Entscheidungen der Art IF...THEN)

5. Mehrfachverzweigungen für wenigstens 4 Verzweigungsrichtungen und Verzweigungen über Tabellen (in Steuerungsanwendungen ist es oft notwendig, schnelle Entscheidungen auf Grundlage mehrerer Bedingungen zu treffen; mit herkömmlichen Sprungbefehlen allein werden solche Abläufe vergleichsweise langsam).

6. bewährte Zugriffsprinzipien (Stack-Organisation, Basis + Displacement usw.)

7. kurze Auswahlangaben (im Interesse der Speicherplatzersparnis)

8. effektiver Unterprogrammaufruf

9. Unabhängigkeit der Programme voneinander

10. keine Datentyparchitektur. Daten soll aus der Sicht der Hardware lediglich Bitketten sein. Deren Interpretation wird ausschließlich durch die Befehle bestimmt.

Bemerkung:
Es darf nicht verhindert werden, Laufzeitsysteme mit Typprüfung effizient zu implementieren (z. B. für Smalltalk). Auf Grund der künftigen Bedeutung solcher Systeme sind die elementaren leistungsbestimmenden Abläufe in der S.015-Architektur mit universell nutzbaren Befehlen zu unterstützen (das betrifft z. B. verkettete Tabellenzugriffe).

4. Literatur

/1/ Bochmann, D.: Automatengraphen. Akademie-Verlag, Berlin 1982.

/2/ Kulisch, U. W.; Miranker, W. L.: Computer Arithmetic in Theory and Practice. Academic Press, New York - London 1981.

/3/ Matthes, W.: Ergänzungsschaltungen für Mikroprozessoren. Institut für Informatik und Rechentechnik, Berlin 1990 (in Vorbereitung).

/4/ -: Hardware Resources: A Generalizing View on Computer Architectures. Institut für Informatik und Rechentechnik, Berlin 1990 (Preprint in Vorbereitung).

/5/ -: Leistungsoptimierte Einzelprozessorarchitekturen. Institut für Informatik und Rechentechnik, Berlin 1990 (Preprint in Vorbereitung).

/6/ -: Die Ternärverktorliste als Datenstruktur für Binärsteuerungen - neue Konzepte für Steuerungshardware. In Vorbereitung.

/7/ -: System.015 (Vorläufiges) Architekturhandbuch (S.015-Dokumentationsbezeichnung: ARM.015). In Vorbereitung.