# Ein Datenflussrechnerkonzeot für den Einsatz in eingebetteten Systemen

Autor(en): **Bührer, R.**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses**

Band (Jahr): **79 (1988)**

Heft 7

PDF erstellt am: **22.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-904015

# Ein Datenflussrechnerkonzept für den Einsatz in eingebetteten Systemen

R. Bührer

***Ein Forschungsprojekt an der ETH Zürich soll abklären, wieweit Datenflussrechner auch für eingebettete Systeme eingesetzt werden können. Dazu wird eine entsprechende Programmiersprache und ein Experimental-Multiprozessor entwickelt. Nach einer Beschreibung des Codeblock-Datenflusskonzepts wird gezeigt, wie die funktionellen und zeitlichen Aspekte eines Datenflussrechners auf einem experimentellen Multiprozessor emuliert werden können. Die Architektur und die mögliche Implementation eines derartigen Systems sowie einige experimentelle Aspekte werden näher beleuchtet und abschliessend einige Schlussfolgerungen angegeben.***

***Un projet de recherche à l'EPF de Zurich essaie de montrer dans quelle mesure les ordinateurs à circulation de donées (dataflow) peuvent être appliqués dans la réalisation de systèmes à temps réel. Un langage de programmation approprié et un multiprocesseur expérimental sont développés à cet effet. Après une description du concept de circulation de données codeblock, il est montré comment les aspects fonctionnels et temporels d'un tel ordinateur peuvent être émulés sur un multiprocesseur expérimental. L'architecture et l'implémentation possible d'un tel système, ainsi que quelques aspects expérimentaux sont mis en lumière et quelques conclusions tirées.***

**Adresse des Autors**

Dr. *Richard Bührer*, Institut für Elektronik, ETH-Zentrum, 8092 Zürich.

Unter den verschiedenen grundsätzlich neuen Parallelcomputerarchitekturen [1] scheinen die Datenflussrechner für den Einsatz in technisch-wissenschaftlichen Anwendungen in der höheren und höchsten Leistungsklasse besonders prädestiniert zu sein [2]. Der kommerzielle Durchbruch dürfte hingegen in den nächsten Jahren noch nicht erfolgen, da zahlreiche Probleme auf den Stufen *Applikationsprogrammierung* (Sprachdefinition, Bereitstellung von Software-Entwicklungswerkzeugen usw.), *Laufzeitsystem* (Prozessor- und Speicherverwaltung usw.) und *Implementation* (spezielle Speicherelemente hoher Kapazität, extrem leistungsfähige Netzwerke zur Datenübertragung zwischen den einzelnen Prozessoren usw.) noch nicht vollständig gelöst sind.

Während die Forschungsarbeiten auf dem Gebiet neuer Rechnerarchitekturen für Supercomputer-Anwendungen in den letzten Jahren bemerkenswerte Fortschritte verzeichnen konnten [3], wurden bei den entsprechenden Arbeiten für eingebettete Systeme (Echtzeitsysteme, Embedded Systems) vergleichsweise bescheidene Erfolge erzielt, obwohl auch bei solchen Anwendungen künftig sehr hohe Rechenleistung gefragt sein wird, z. B. beim Einsatz fortgeschrittener Industrieroboter oder bei der Überwachung und Steuerung komplexer Industrieanlagen oder Kraftwerke usw. Beim vorliegenden Forschungsprojekt geht es darum, die Eignung einer speziellen Datenflussrechnervariante (Grobkörniger oder Coarse-Grain-Datenfluss) im Hinblick auf einen Einsatz in künftigen eingebetteten Systemen abzuklären. Zu diesem Zweck wird die Entwicklung einer entsprechenden Programmiersprache sowie eines Emulations-Multiprozessor vorangetrieben. Die besonderen Aspekte einer derartigen Architektur und Implementation sollen möglichst flexibel und dennoch exakt abgeklärt werden.

## Architektur

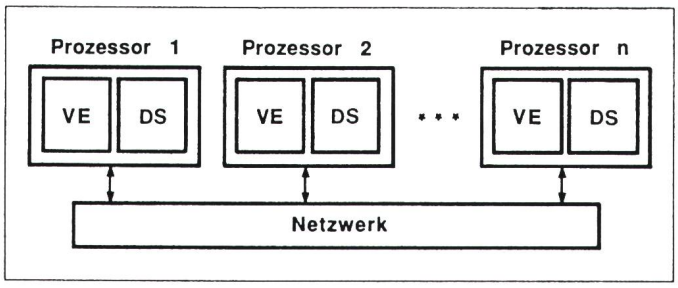
Einer der markantesten Vorzüge des Datenflusskonzepts liegt darin, dass unter Verwendung eines funktionalen Programmierstils die in einem Applikationsprogramm enthaltene Parallelität einfach extrahiert und in einer entsprechenden parallelen Rechnerarchitektur ausserordentlich gut ausgenützt werden kann [2], wodurch, zumindest theoretisch, extrem hohe Verarbeitungsgeschwindigkeiten möglich werden. Praktische Implementationsstudien haben jedoch gezeigt, dass infolge spezieller Beschaffenheit der einzelnen Datenflussprozessoren [4] die Rechenleistung stark reduziert wird, wenn die Programmparallelität eine gewisse Grenze unterschreitet. Da zudem die einzelnen Instruktionen ausschliesslich aufgrund der Verfügbarkeit der Daten abgearbeitet (gefeuert) werden, kann ein nach diesem Konzept aufgebautes eingebettetes System die strengen Antwortzeitbedingungen von Echtzeitsystemen nicht einhalten.

Die Entwicklung des nachstehend beschriebenen Codeblock-Datenflusskonzepts basiert auf Forschungsarbeiten [5], welche zum Ziel hatten, diese Nachteile zu eliminieren. Der wesentliche Unterschied gegenüber klassischen Lösungsversuchen wie [4] oder [6] besteht darin, dass die einzelnen Knoten in einem Datenflussgraphen [2] nicht mehr einzelne (einfache) Maschineninstruktionen sondern ganze Blöcke solcher Instruktionen (d. h. Codeblocks) beinhalten. Derartige Codeblocks weisen etwa die Struktur von Prozeduren oder Loops auf, wie sie von höheren Programmiersprachen her bekannt sind. Die Vorteile des Datenflusskonzepts können somit übernommen werden, während konventionelle Von-Neumann-Rechnertechniken bei der Abarbeitung der einzelnen Codeblocks die Eliminierung der erwähnten Nachteile ermöglicht.

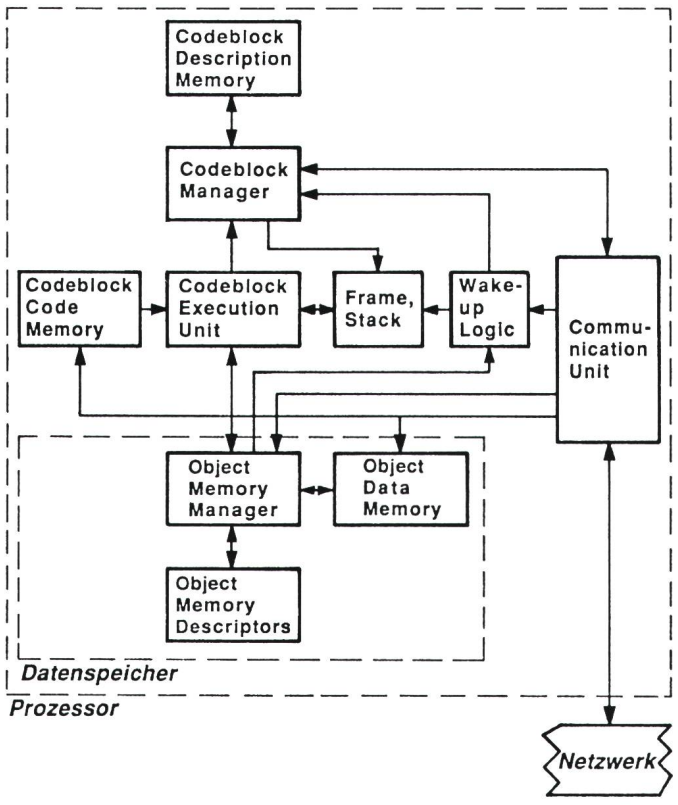
Der konzipierte Datenflussrechner

(Fig. 1) besteht aus einer Vielzahl einzelner Prozessoren, die sich aus einer Verarbeitungseinheit (VE) und einem verteilten Datenspeicher (DS) zusammensetzen und durch ein leistungsfähiges (z. B. Packed-Switched) Netzwerk verbunden sind. Der Aufbau eines einzelnen Prozessors ist in Figur 2 dargestellt.

Grosse Datenstrukturen wie Arrays, Matrizen usw. werden auf die einzelnen Speicherblöcke der verschiedenen Prozessoren verteilt; die Grundidee dazu wurde von den in [4] beschriebenen I-Strukturen übernommen. Da bei der von-Neumann-artigen Abarbeitung einzelner Codeblocks Referenzen auf Daten vorkommen, welche in nichtlokalen Speicherblöcken abgelegt sind, muss verhindert werden, dass das durch das eigentliche Datenflusskonzept elegant gelöste Memory-Latency-Problem [2] erneut auftritt und die Prozessoren während der Daten-Fetch-Phase für längere Zeit blockiert werden. Die Lösung besteht darin, dass in jedem Prozessor mehrere lauffähige Codeblocks vorliegen, welche ihrem momentanen Status entsprechend zur Abarbeitung gelangen. Jeder einzelne Codeblock verfügt dabei über einen eigenen Stack-Bereich zur Ablage temporärer Daten. Referenzen auf die Daten des verteilten Speichers erfolgen anhand spezieller Instruktionen (Load Global Address to Local Address), welche ein Data Request Token generieren, das zum Objekt Memory Manager (Figur 2) des das verlangte Datum enthaltenden Prozessors übertragen wird. Als Erwiderung sendet dieser eine Kopie des Datums in Form eines Data Tokens dem aufrufenden Prozessor zurück. Die das Datum benötigende Instruktion führt zur Ausführungszeit einen impliziten Test (Test Local Address) durch, um sich von der (rechtzeitigen) Ankunft desselben zu überzeugen. Stellt sich heraus, dass sich das verlangte Datum nicht am vorgesehenen lokalen Speicherplatz befindet, wird die Instruktion und der zugehörige aktive Codeblock suspendiert und ein anderer lauffähiger Codeblock zur Ausführung gebracht. Das nachträglich eintreffende Data Token wird vom Codeblock Manager als solches erkannt und der zugehörige Codeblock wird vom Zustand «suspendiert» in den Zustand «ausführbar» übergeführt. Die sonst üblichen zeitintensiven Context-Umschaltungen werden durch die Verwendung des Stackrechnerprinzips in den einzelnen Codeblock Execution Units auf ein absolutes Minimum reduziert.




**Figur 1**
**Konzept eines Datenflussrechners**
VE Verarbeitungseinheit
DS Datenspeicher




**Figur 2**
**Prozessor des Codeblock-Datenflussrechners**
Codeblock Execution Unit: Abarbeitung der Codeblocks
Codeblock Manager: Verwaltung der Codeblocks
Object Memory Manager: Verwaltung und Zugriffskontrolle des verteilten Datenspeichers
Communication Unit: Schnittstelle zum Netzwerk

Der Codeblock Manager ist neben der Verwaltung auch für die Zuweisung der auszuführenden Codeblocks zuständig. Während die Verwaltung der einem Prozessor zugewiesenen Codeblocks eine rein lokale Angelegenheit ist, verlangt die zur Laufzeit erfolgende Zuweisung neuer Codeblocks zu den einzelnen Prozessoren umfassenden Koordinationsmassnahmen zwischen allen Codeblock Managern des Systems. Es gilt, die momentane Auslastung der verschiedenen Prozessoren möglichst korrekt und konsistent zu erfassen und anhand dieser sowie anhand der anfallenden Last (Rechen- und Speicheraufwand für die Abarbeitung des Codeblocks) die Arbeit im Gesamtsystem möglichst optimal zu verteilen.

Ähnliche Aufgaben im Bereich der Speicherverwaltung werden durch die Gesamtheit der Object Memory Manager des Systems wahrgenommen.

## Programmierung des Codeblock-Rechners

Zur Programmierung des Codeblock-Datenflussrechners, der im vorliegenden Forschungsprojekte untersucht werden soll, wird zunächst eine bereits existierende, experimentelle, später eine neue, optimierte Sprache verwendet. In einer ersten Phase wird das Systemverhalten anhand verhältnismässig einfacher Testprogramme analysiert. Die zu diesem Zweck benötigten Applikationsprogramme werden in der seit einigen Jahren verfügbaren Sprache Sisal [7] geschrieben. Der zugehörige Compiler generiert

eine als IF1 bezeichnete Zwischenform, aus der ein im Rahmen dieses Projekts entwickelter Codegenerator den für den Codeblock-Datenflussrechner definierten Programmcode erzeugt. Als langfristige Lösung wird in einer zweiten Phase die Entwicklung einer neuen Applikationssprache vorangetrieben, welche unter anderem auch die nötigen Konstrukte für die Programmierung von Echtzeitanwendungen beinhalten wird.

## Emulation eines Codeblock-Datenflussrechners

Die Erforschung eines weitgehend neuartigen Rechnerkonzepts führt zur Erarbeitung zahlreicher Architektur- und Implementationsvarianten. Die Einflüsse der sich dabei herauskristallisierenden Hard- und Softwarelösungen müssen sorgfältig untersucht und optimiert werden. Da die Zahl der anfänglich freien Systemparameter sehr gross ist, muss der Bereitstellung einer flexiblen und leistungsfähigen Experimentierumgebung grösste Beachtung geschenkt werden.

Die gewählte Lösung stützt sich auf drei verschiedene, sich jedoch ergänzende Methoden:

1. Implementation einer sogenannten Metamaschine zur Emulation eines einzelnen Codeblockprozessors auf einem konventionellen Rechner (Macintosh II),
2. Verwendung spezieller Simulationspakete, welche detaillierte Simulationen einzelner Hardwarekomponenten des Datenflussrechners ermöglichen (eine detaillierte Simulation des vollständigen Multiprozessors ist selbst unter Verwendung von Höchstleistungsrechnern extrem zeitintensiv und zurzeit unrealistisch) und
3. Realisierung eines experimentellen Multiprozessors, mit dem Entwürfe von Codeblock-Multiprozessoren emuliert werden können.

Im folgenden wird auf den Emulations-Multiprozessor eingegangen, der momentan aufgebaut wird. Erfahrungen mit vergleichbaren Projekten haben gezeigt, dass ein Experimentator von einem derartigen Multiprozessor drei Grundvoraussetzungen erwartet. Die erste betrifft eine *komfortable Benutzerschnittstelle.* Dazu gehört eine gute Eingabemöglichkeit von Systemparametern (z. B. Speichergrösse, Busbandbreiten usw.), eine klare und umfassende Darstellung der zu untersuchenden Systemdaten in Form von Tabellen und Grafiken etc. und nicht zuletzt auch die Verfügbarkeit einer entsprechenden Programmiersprache mit Compiler für die Erstellung lauffähiger Testprogramme. Die zweite Voraussetzung betrifft die notwendige *Flexibilität* des Systems. Es muss möglich sein, den Einfluss verschiedener Systemparameter im Detail zu studieren, ohne dass der physikalische Aufbau des Rechners verändert werden muss. Die dritte Voraussetzung betrifft eine angemessene *Systemleistung;* einzelne Experimentierphasen (Abarbeitung von Testprogrammen mit unterschiedlichen Systemparametern) sollen keine untolerierbar lange Zeit in Anspruch nehmen. Diese verschiedenen Ansprüche sind teilweise kontradiktionär, und es gilt daher, einen entsprechenden Kompromiss auszuarbeiten[1].

Der Emulations-Multiprozessor soll neben den funktionellen Eigenschaften des theoretischen Codeblock-Multiprozessors auch dessen zeitbezogenen Eigenschaften exakt emulieren (nachbilden) können. Im Gegensatz zu nahezu allen existierenden Forschungs-Multiprozessoren hat der Experimentator dadurch die Möglichkeit, die Einflüsse sowohl von Software- wie auch von Hardwarekomponenten umfassend zu studieren. Bei der Konzeption des Emulationsrechners wird als erstes die gewünschte, auf das zu emulierende System bezogene, zeitliche Auflösung (Basic Time Step) aller Messungen festgelegt. Dieser Basiszeitschritt hat sowohl Einfluss auf die Genauigkeit aller Messresultate wie auch auf die Dauer der einzelnen Experimente, d. h. auf die Abarbeitungszeit entsprechender Testprogramme. Im vorliegenden Projekt wurde ein Basiszeitschritt gewählt, der der Abarbeitungszeit einer einfachen arithmetischen Operation entspricht. Basierend auf der Wahl dieser Grösse wird in einem nächsten Schritt definiert, welche Arbeit jede einzelne Rechnerkomponente während eines solchen Zeitschrittes erledigen kann.

Da der Codeblock-Prozessor gemäss Figur 2 aus einzelnen Blöcken (Subunits) besteht, welche über entsprechende (nicht gezeichnete) Entkopplungslatches oder First-in-First-out-Buffer gekoppelt sind, und alle Datentransfers in Form von Datenpaketen erfolgen, kann ein Emulationsalgorithmus gewählt werden, welcher zyklisch in jeder Subunit die Arbeit *eines* Basiszeitschrittes zur Ausführung bringt. Sobald alle Subunits eines Prozessors sowie alle Prozessoren des Systems die möglichen Aktivitäten des laufenden Zeitschritts erledigt haben, wird der gesamte Multiprozessor resynchronisiert und die Emulation des nächsten Zeitschritts initialisiert. Eine etwas vereinfachte Form dieses Algorithmus wird in Tabelle I aufgezeigt. Es fällt dabei auf, dass in einem ersten Schritt zuerst alle Eingangswerte der einzelnen Subunits eingefroren und anschliessend die möglichen Aktivitäten durchgeführt werden. Die von den Experimentator gewünschten Datenerfassungen für die laufende oder spätere Auswertung (Monitoring Tasks) werden ähnlich wie die Aktivitäten der Subunits eingefügt. Durch die spezielle Wahl des Emulationsalgorithmus wird die Abarbeitungszeit eines Testprogramms dadurch etwas verzögert; die Genauig-

[1] Das Schwergewicht der momentanen Projektphase liegt auf der Untersuchung der grösseren Zusammenhänge bzw. Einflüsse verschiedener Systemparameter. Zu diesem Zweck muss der theoretische Rechner zugunsten hoher Flexibilität und vernünftiger Experimentierzeiten entsprechend auf das wesentliche beschränkt werden. Fine-Tuning ist einer späteren Projektphase mit verbesserter Hardwareumgebung vorbehalten (z. B. Bau eines Prototyprechners).

**Tabelle 1**
**Emulationsalgorithmus des Emulationsrechners**
(Vereinfachte Darstellung)
max_number_of_steps: totale Anzahl Zeitschritte
max_number_of_subunit: totale Anzahl Subunits

```
for basic_time_step := 1 to max_number_of_steps do
    for subunit := 1 to max_number_of_subunits do
        {freeze input data of each subunit};
    for subunit := 1 to max_number_of_subunits do
        if input_data[subunit] = available then begin
            {do work of one basic time step};
            {perform monitoring tasks as necessary};
            {do output to connected subunit}
        end;
    {synchronize system-wide}
end.
```

keit der Messresultate wird hingegen in keiner Art und Weise beeinflusst.

In der praktischen Implementation wird der Emulationsalgorithmus von Tabelle I derart erweitert, dass auch Aktivitäten berücksichtigt werden können, welche mehr als einen ($k>1$) Basic Time Step beanspruchen. In diesen Fällen wird die entsprechende Aktivität der Subunit durchgeführt und diese anschliessend für weitere $k$-1 Zeitschritte gesperrt. Entsprechende Datenausgaben werden während des $k$-ten Zeitschrittes durchgeführt.

## Architektur und Implementation des Emulationsmultiprozessors

Zwei der wichtigsten Anforderungen beim Bau eines Experimentalrechners für Architekturuntersuchungen sind, wie erwähnt, dessen Flexibilität und Leistung. Flexibilität kann primär dadurch erreicht werden, dass die kritischen Module des Rechners in Software statt in Hardware realisiert werden. Änderungen oder Optimierungen führen dabei nur zu einfachen Softwareanpassungen. Die effektiv für die Emulation benötigte Hardware kann daher auf verhältnismässig einfache Komponenten wie konventionelle Prozessoren, Speicher, Interfaces und ein Netzwerk beschränkt werden, wie dies in Figur 3 dargestellt ist. Jeder der einzelnen Prozessoren (Macintosh II) emuliert dabei einen vollständigen Codeblock-Datenflussprozessor.

Leistungserhöhungen für den Fall weiterer, umfassenderer, Untersuchungen lassen sich mittels Hinzufügen zusätzlicher Emulationsprozessoren bewerkstelligen. Durch die Entkopplung der Prozessoren und des Netzwerks mittels Dual-Port-Speichern und definierten Protokollen bei der Datenübermittlung können aber auf einfache Art auch leistungsfähigere Prozessoren an das bestehende Netzwerk angeschlossen werden. Eine der Hauptauflagen wäre in diesem Fall, dass die in Modula-2 geschriebene Emulationssoftware weiter verwendet werden kann.

Das Kommunikationsnetzwerk des Emulations-Multiprozessors wird in einer ersten Phase (Anzahl Prozessoren $\leq 5$) mit Inmos-Transputern realisiert, wobei jeder einzelne Transputer mittels serieller Verbindungen mit allen anderen Transputern verbunden sein wird. Für mehr als 5 Prozessoren können hypercubeartige oder andere Transputer-Netzwerktopologien realisiert werden. Das in Figur 3 eingezeichnete Synchronisationsnetzwerk dient der im Emulationsalgorithmus von Tabelle I aufgeführten, nach jedem Zeitschritt erfolgenden Synchronisation des Gesamtsystems.

## Experimentelle Aspekte

Die zahlreichen Experimente auf dem Emulations-Multiprozessor haben zum Ziel, die grundsätzlichen Abläufe in einem Codeblock-Datenflussrechner zu verstehen sowie die einzelnen Komponenten eines solchen Rechners zu optimieren. Die Untersuchungen werden auf drei Ebenen durchgeführt und umfassen unter anderem folgende Teilaspekte:

### 1. auf der Compilerebene

- Art und Grösse der Codeblocks
- Statische Codeblock-Zuweisungsstrategien
- Strategien für das effiziente Kopieren von Daten
- Statische Codeblock-Prioritätsschemata zur Einhaltung vorgegebener Antwortzeiten.

### 2. auf der Laufzeitsystemebene

- Codeblock-Zuweisungsstrategien auf die verschiedenen Prozessoren
- Algorithmen für die Zuweisung von Daten auf den verteilten Speicher
- Dynamische Codeblock-Prioritätsschemata.

### 3. auf der Implementationsebene

- Vielfalt und Art der Maschineninstruktionen
- Beeinflussung der Rechenleistung durch Komponentenmodifikationen
- Einfluss unterschiedlicher Verbindungsnetzwerke zwischen den Prozessoren
- Einfluss unterschiedlicher Verbindungskonzepte innerhalb der Prozessoren
- Einfluss der Grösse von Speichern und Datenbuffern.

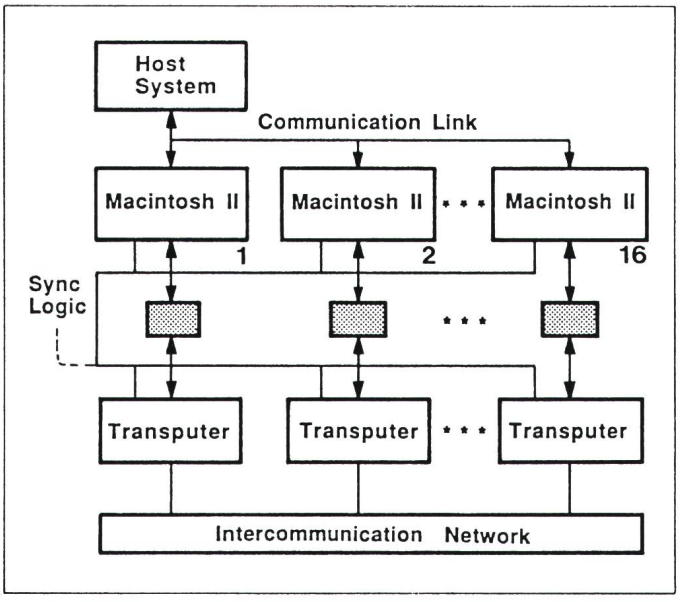
Im praktischen Experimentierbetrieb können auf übersichtliche Art und Weise die vorgesehenen Systemparameter mittels Bildschirm eingegeben und die gewünschten Datenaufzeichnungsarten und deren Darstellungsvarianten ausgewählt werden. Je nach Fall werden diese Werte laufend oder am Ende des Experimentes dargestellt. Die vollständige Experimentierumgebung, welche auch Daten für Off-Line-Simulationen zur Verfügung stellt (Trace Output), wird in Figur 4 dargestellt. Ein anderer wichtiger Aspekt der Untersuchungen betrifft die Erarbeitung von Software-Debugging-Techniken für Mehrprozessorsysteme.

## Schlussfolgerung und Ausblick

Die Resultate verschiedener ausländischer wie auch eigener Forschungsarbeiten deuten darauf hin, dass das Codeblock-Datenflusskonzept ein

**Figur 3**
**Struktur des Emulations-Multiprozessors**

Dual-Port-Speicher




# Wer zur Gesamtleistung Farbe bekennt, steckt den Horizont weiter.

**Zum Beispiel in der Tarifgestaltung**

Wer den Blick in die Zukunft richtet, wird die Möglichkeiten der zeit- und leistungsabhängigen Tarifgestaltung verstärkt nutzen. Zumal Ihnen Landis & Gyr technisch ausgereifte Tarifgeräte anbietet.

Mit Landis & Gyr haben Sie einen erfahrenen Partner, der Ihnen übergreifende Gesamtlösungen bietet. Und zwar für die Energiemessung, die Tarifgestaltung, die Datenerfassung und die Zählerprüfung.

Farbe bekennen zur Gesamtleistung eines Partners, der für die Zukunft gerüstet ist, eigentlich eine gute Sache. Und für uns ein willkommener Anlaß, Ihnen und allen anderen Kunden für die vertrauensvolle Zusammenarbeit zu danken.

## Landis & Gyr – der Partner für umfassende Lösungen

Bitte beachten Sie
zum Thema Tarifgestaltung
die Rückseite.

LANDIS & GYR

# Tarifgestaltung – ein wichtiger Bereich der Gesamtleistung

Die Verrechnung der elektrischen Energie erfolgt zunehmend mit Hilfe von Tarifen, welche die Kosten für die Bereitstellung von Leistung und Energie widerspiegeln. Sie berücksichtigen dabei einerseits den Zeitpunkt und andrerseits die Intensität des Energiebezugs. Tarifgeräte mit dementsprechenden Funktionen ergänzen die für die Energiemessung eingesetzten Zähler und liefern die erforderlichen Daten zur Verrechnung des Energiebezugs. Sie ermöglichen es den EW, den Bezug je nach Wunsch und Anforderung an die Tarifstrukturen zu verrechnen.

Die **zeitabhängige Tarifgestaltung** ist eine der verschiedenen Maßnahmen, die vorhandene Netzkapazität optimal auszunutzen. Was Ziel jedes EW ist. Dazu dienen

- mechanische Doppel- und Dreifachtarifzählwerke oder
- elektronische Geräte für Mehrfach-Energietarif

Rundsteuerung oder Schaltuhren übernehmen die zeitabhängige Steuerung.

Die **leistungsabhängige Tarifgestaltung** soll den Investitionsaufwand und die Betriebskosten für die Erzeugung und Lieferung der elektrischen Energie berücksichtigen. Die EW sind bestrebt, mittleren und größeren Abnehmern neben der bezogenen Energie auch die beanspruchte Leistung zu verrechnen.

Hierzu dienen Tarifgeräte, welche während einer definierten Meßperiode (z.B. 15 min) die mittlere Leistung ermitteln und den höchsten Wert (Maximum) während einer oder mehrerer Verrechnungsperioden festhalten. Diese Tarifgeräte sind

- elektromechanische Maximumzählwerke, teils mit Kumulierfunktion
- elektronische Tarifgeräte mit Mehrfach-Leistungs- und Energietarifen sowie Datenspeicher

**Datenerfassung**

Diesem Bereich ist unser nächster Beitrag gewidmet.

Für Ihr Interesse danken wir Ihnen schon jetzt.


LGZ Landis & Gyr Zug AG
Gubelstraße
CH-6301 ZUG
Tel. 042 - 24 11 24

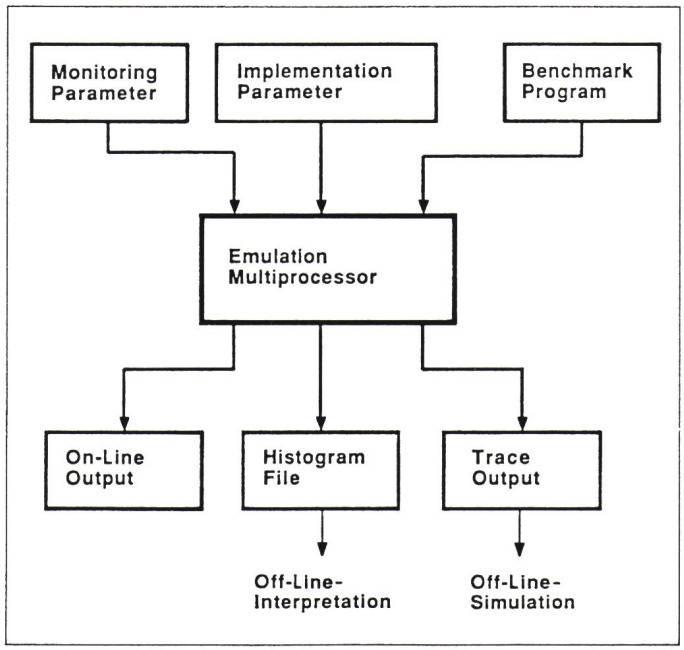
Elektrizitätszähler
Fernwirk- und Rundsteuertechnik
Heizungs- und Klimaregelung, Gebäudeleittechnik
Telefonie
Monetics

LANDIS & GYR

Z/D-CH 543

Gedruckt in der Schweiz

88801 KZ




**Figur 4**
**Experimentierumgebung**

vielversprechender Ansatz zur Realisierung künftiger eingebetteter Systeme darstellt. Sie versprechen eine hohe Rechenleistung sowie eine solide Programmierungsgrundlage. Die Einführung von Prioritätsschemata und die Verwendung von konventionellen Von-Neumann-Techniken zur Abarbeitung von Codeblocks bieten zudem die Grundlage für einen erfolgreichen Einsatz unter strengen Echtzeitbedingungen.

Die Zahl der noch offenen Fragen ist hingegen nicht klein. Die laufenden Forschungsarbeiten müssen die verschiedensten Aspekte abklären. Dazu gehören insbesondere das Anforderungsprofil potentieller Anwendungen sowie die Frage, ob eine mächtige Programmiersprache und ein leistungsfähiger Compiler, welcher effizienten und korrekten, den Spezifikationen entsprechenden Code zu generieren in der Lage ist, mit vernünftigem Aufwand realisierbar sind. Auf der Systemseite müssen für die Prozessor- und Speicherverwaltung günstige Strategien evaluiert und implementiert werden. Das Verhalten bei Ausfällen von Komponenten, allenfalls verbunden mit redundanter Verarbeitung, ist ebenso Gegenstand vertiefter Untersuchungen wie die Detailplanung entsprechender Zielhardware. Zur Problematik der Implementation gehören auch die physikalischen Randbedingungen, die für viele eingebettete Systeme gegeben sind, wie Grösse, Stromverbrauch, begrenzte Wartbarkeit usw.

Als vorteilhaft kann abschliessend der Aspekt gewertet werden, dass bei vielen Anwendungen vor der eigentlichen Inbetriebsetzung eine Optimierungsphase eingeschaltet werden kann, so dass bei eingebetteten Systemen verschiedene der erwähnten Probleme einen kleineren Stellenwert als bei allgemeinen Anwendungen von Datenflussrechnern (z. B. Supercomputer) einnehmen.

**Literatur**

[1] *P.C. Treleaven* and *I.G. Lima:* Future computers: Logic, data flow... control flow? IEEE Computer 17(1984)3, p. 47...58.

[2] *R. Bührer:* Datenflussrechner - Konzepte und Anwendungen. Bull. SEV/VSE 78(1988)7, S. 334...350.

[3] Proceedings of the Second Conference on Supercomputing. Vol. 1...3 St. Petersburg/Florida, International Supercomputing Institute Inc., 1987.

[4] *Arvind* a. o.: The tagged token dataflow architecture. Memo of the Computation Structures Group/Laboratory for Computer Science. Cambridge/Mass., Massachusetts Institute of Technology (MIT), 1983.

[5] *R. Buehrer* and *K. Ekanadham:* Incorporating data flow ideas into von Neumann processors for parallel execution. IEEE Tran. on Computers C 36(1987)12, p. 1515...1522.

[6] *J.R. Gurd, C.C. Kirkham* and *I. Watson:* The Manchester prototype dataflow computer. Communications of the ACM 28(1985)1, p. 34...52.

[7] *J. McGraw:* SISAL - Streams and iterations in a single-assignment language. Language reference manual. Livermore/California, Livermore National Laboratory, 1983.

[8] *R. Bührer:* Emulation of a parallel codeblock dataflow processor. Microprocessing and Microprogramming (The Euromicro Journal) 21(1987)- p. 319...324.