Wirtschaftliche Implementierung von PEARL auf Mikroprozessoren: INTEL RMX86 -PEARL

P. Heine, F. Kaiser, IITB Karlsruhe; **R. Schneider,** INTEL München

Zusammenfassung

Es wird eine Implementierung von PEARL auf einer INTEL 8086/87-basierenden Hardware-Konfiguration vorgestellt. Unter dem Gesichtspunkt der Wirtschaftlichkeit wurden dabei, soweit als möglich, bereits existierende Programm-Pakete benutzt.

Summary

We present an implementation of PEARL on an INTEL 8086/87-based hardware configuration. For economic reasons as far as possible we explicitely selected already existing program packages to be parts of the system.

1. Einleitung

Für viele der derzeitig verfügbaren Mikro-Rechner existieren leistungsfähige Betriebssysteme, die alle einen vergleichbaren Satz an multi-tasking and realzeit-Eigenschaften als Minimum enthalten. Auch kristallisiert sich für einige Hochsprachen heraus, daß sie künftig zur Standard-Ausrüstung solcher Rechner gehören werden. Es liegt damit nahe, daß das Ausnutzen dieser beiden Tatsachen eine äußerst wirtschaftliche Implementierung von PEARL auf Mikro-Rechnern ermöglichen kann.
Deshalb wurden bei der hier vorgestellten Implementation von PEARL auf einem Entwicklungssystem und Single-board-computer der Fa. INTEL nach Möglichkeit bereits existierende Standard-Bausteine, -Schnittstellen und -Hochsprachen benutzt.

Die Schnittstellen des Compilers, sowie der Laufzeit-unterstützenden Prozeduren zu den darunter liegenden Schichten, können somit weit oberhalb der Maschinen-Ebene angelagert werden. Die darunter liegenden Schichten sind dann durch die lokalen Implementierungen verwirklicht. Im vorliegenden Fall wurden der existierende INTEL PASCAL-Compiler: PASC86 [1] und das RMX86-Betriebssystem [2] bei der PEARL-Implementierung miteinbezogen.

Bei diesem Vorgehen wurde ein Verlust an Leistungsfähigkeit dadurch vermieden, daß wenige, aber für die Laufzeit wesentliche Anteile klar abgetrennt und in Assembler-Sprache codiert wurden. Die Abb. 1 gibt einen Überblick über das gesamte Programmsystem, das im folgenden näher beschrieben wird.

2. PEARL-Compiler

Der PEARL-Compiler ist von der Fa. Werum erstellt worden.

Sein Sprachumfang ist in [3] festgelegt. Er geht weit über BASIC-PEARL hinaus. Der Code-Generator des PEARL-Compilers erzeugt PASCAL-Quellcode als Zwischensprache. Diese wird zum Erzeugen des Maschinencodes, als letzter Schritt des Übersetzens, vom PASC86-Compiler weiterverarbeitet. Der Sprachumfang dieses Compilers ist eine Obermenge des ISO-Normungsvorschlags für STANDARD PASCAL. Der PEARL-Compiler selbst liegt in PASC86-Code vor. Er läuft auf dem INTEL MDS SERIES III Entwicklungssystem [4] als Overlay-Programm unter der Kontrolle des standard ISIS-Betriebssystems [7]. Zugriff zum ISIS-Betriebssystem erfolgt über die standardisierte INTEL-interne UDI-Schnittstelle [5]. Dadurch ist der PEARL-Compiler auch unter künftigen INTEL-Entwicklungssystemen ablauffähig, die der UDI-Schnittstelle genügen.

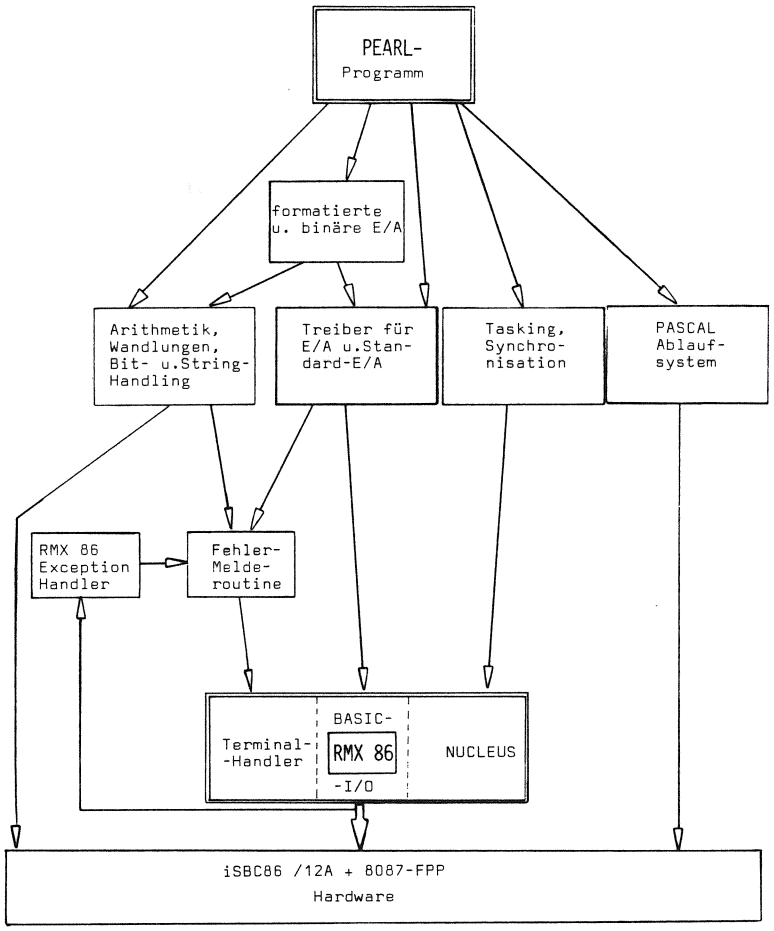

Abb. 1: Ablaufschema eines PEARL-Jobs unter dem RMX86-Betriebssystem

Wesentliche Abbildungen der Strukturen PEARL- nach PASC86-Moduln sind:

- Ein PEARL-Modul wird in einen PASC86-non-main-Modul abgebildet.
- Eine PEARL-Task wird in ein PASC86-Prozedur abgebildet. Diese wird dann später, zur Laufzeit, dem Betriebssystem als Task bekannt gemacht.
- Jeder PASCAL-Modul enthält eine Initialisierungs-prozedur. Diese erledigt alle Modul-spezifischen Initialisierungen. So macht sie z.B. die Tasks des Moduls dem Betriebssystem bekannt oder erzeugt Objekte, die zur Verwaltung von Warteschlangen dienen.
- Der PEARL-Compiler generiert eine INCLUDE-Anweisung für ein File, in dem die Typ-Definitionen für alle eigen erzeugten Objekte stehen wie Task-, Interrupt-, Semaphor- und Bolt-Kontrollblöcke.

Die Initialisierung dieser Struktur der generierten PASC86-Moduln ist in Abb. 3 skizziert. Sie wird in Abschnitt 6 näher behandelt.

3. Betriebssystem

Ein gebundenes PEARL-Programm läuft unter der Kontrolle des INTEL RMX86-Betriebssystem ab. Dieses Betriebssystem bietet die wesentlichen Funktionen, die bis auf wenige Ausnahmen (BOLT-Variablen, SIGNALs, zeitliche- und Unterbrechungs-Einplanungen von Tasks) eine einfache Abbildung der PEARL-auf den RMX86-Funktionen gestatten.

So sind z.B. die PEARL-Anweisungen ACTIVATE, SUSPEND, CONTINUE, RESUME, REQUEST, RELEASE nahezu unmittelbar auf RMX86-Aufrufe abbildbar; für Zeit- und Interrupt-Einplanungen, sowie für die Behandlung von BOLT-Variablen müssen eigene Warteschlangen aufgebaut werden. Die vorliegende Implementierung des RMX86 ist eine Konfiguration bestehend aus NUCLEUS, BASIC I/O und TERMINAL-HANDLER.

4. Laufzeit-Unterstützung

Die PEARL-Laufzeitroutinen dienen der Abarbeitung in PASC86 nicht vorhandener Operationen wie der Behandlung von Bit- und Zeichen-Ketten,

dem Aufruf von RMX86-Funktionen zur Manipulation von RMX86-Objekten oder der Manipulation eigener Objekte.

Im einzelnen lassen sich die PEARL-Laufzeitroutinen wie folgt klassifizieren:

- PEARL-Arithmetik
- Formatierte und binäre PEARL-Ein-/Ausgabe
- PEARL-Ein-/Ausgabe-Treiber
- PEARL-Tasking und -Synchronisation
- Fehlerbehandlung
- Standard INTEL PASC86- und 8087-floating-point-prozessor-Unterstützung

4.1 PEARL-Arithmetik

Diese Routinen enthalten die in PASCAL nicht vorhandenen arithmetischen und logischen Operationen mit PEARL-Variablen vom Typ BIT, CHARACTER, Doppelwort Integer FIXED (31) u.ä., sowie Typwandlungen.

Diese Routinen sind in Assembler-Sprache geschrieben.

4.2 Formatierte und binäre PEARL-Ein-/Ausgabe

Diese Gruppe von Laufzeitroutinen bearbeitet die vielfältigen Möglichkeiten der Ein-/Ausgabe, die in PEARL geboten werden und bildet diese auf eine Geräteunabhängige Schnittstelle zum Ein-/Ausgabetreiber ab.

Diese Routinen sind in PEARL geschrieben und stellen den umfangreichsten Teil des Laufzeitsystems dar.

4.3 PEARL-Ein-/Ausgabe-Treiber

Der Ein-/Ausgabetreiber stellt die Verbindung von formatierter und binärer Ein-/Ausgabe zum RMX86 Basic I/O Betriebssystemanteil dar. Unterstützt werden alle Geräte, die auch vom Basic I/O System unterstützt werden und die vom File-Typ 'named' oder 'physical' sind.

Die Aufgabe des Ein-/Ausgabetreibers ist die Datenübertragung aus einem bzw. in einen von der formatierten und binären Ein-/Ausgabe bereitgestellten Datenpuffer, das Einstellen der aktuellen Schreib-/Leseposition sowie das Öffnen und Schließen von Files.

Die Abbildung der PEARL-Ein-/Ausgabe auf die Basic I/O des RMX86 ist auf diesem Niveau der Treiber stark dadurch vereinfacht, daß beide Seiten der Abbildung nach dem Konzept der Geräteunabhängigkeit gestaltet sind, d.h. die Struktur der Funktionsaufrufe ist dieselbe für alle Geräte.

Der Ein-/Ausgabetreiber ist vorwiegend in PLM86 programmiert.

4.4 PEARL-Tasking und -Synchronisation

Der Zugriff zu PEARL-Tasks, um unbedingte-, Zeit- oder Ereignis-gesteuerte Zustandsänderungen zu bewirken, erfolgt über diese Laufzeitroutinen.

Unbedingte Aufrufe, wie z.B. ACTIVATE, SUSPEND, CONTINUE bewirken unmittelbar die entsprechenden RMX86-NUCLEUS-Aufrufe.

Von den PEARL-Synchronisationsobjekten Semaphor- und Bolt-Variablen sind nur die Semaphoren direkt auf das RMX86 abbildbar. Für die Bolt-Variablen müssen eigene Warteschlangen aufgebaut und verwaltet werden. Ebenso werden Task-Einplanungen für externe Interrupts und zeitgesteuerte Ereignisse von eigenen Warteschlangen verwaltet. Das Laufzeitsystem enthält dazu pro externem Interrupt eine Interrupt-Task, welche die Laufzeitroutinen für die zugehörigen Einplanungen aufruft. Und insbesondere existiert eine Timer-Task, die externe Interrupts bedient und die Warteschlangen für zeitliche Task-Einplanungen bearbeitet. Ganz allgemein werden für die Interrupt-Behandlung die Möglichkeiten genutzt, die das RMX86 zur Verfügung stellt, nämlich die 2-stufige Behandlung eines Interrupts über einen Interrupt-Handler und eine Interrupt-Task (Abb. 2). Die Routinen für Tasking und Synchronisation sind überwiegend in PASCAL programmiert.

4.5 Fehlerbehandlung

Werden innerhalb der Laufzeitroutinen oder der RMX86-Operationen Fehler entdeckt, so folgt über eine zentrale Fehlerroutine eine Fehlermeldung auf dem Bildschirm. Ein Abbruch der fehlerhaften Task ist durch den Aufruf einer entsprechenden Routine möglich.

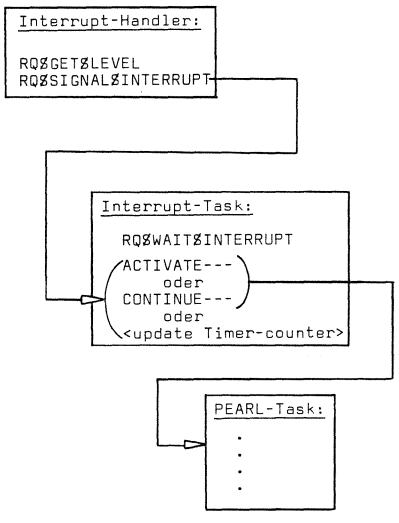

Abb. 2: Ablaufschema eines externen Interrupts

4.6 Standard INTEL PASC86- und 8087-Floating-point-prozessor-Unterstützung

Die von PASC86 benötigten Laufzeitroutinen werden aus den dazugehörigen Bibliotheken nach Bedarf dazugebunden. Es werden allerdings lediglich Routinen zur dynamischen Speicherplatzverwaltung (PASCAL-Anweisung NEW) und solche für einige Einbaufunktionen (z.B. SIN, COS) aus dem PASC86-Laufzeitsystem benötigt.

Der PASC86-Compiler nutzt voll die Möglichkeiten des 8087-Floating-point-Prozessors [6] aus. Und dies gilt damit auch für PEARL-Programme. Hierzu sei bemerkt, daß die Benutzung des 8087-Floating-point-Prozessors nicht nur einen erheblichen Laufzeitgewinn bringt, sondern auch das Einhalten des IEEE Floating point Standards garantiert. Der PASC86-Compiler erzeugt direkten Code für den 8087-Prozessor. Lediglich eine Laufzeitroutine muß zur Initialisierung der 8087-Register während der Initialisierungsphase einer PEARL-Task aufgerufen werden.

5. Binder

Die Möglichkeit, Moduln verschiedener Sprachen zusammenbinden zu können, war eine wesentliche Voraussetzung dafür, wirtschaftlich u n d Laufzeit-effizient sein zu können. So konnten kleine, laufzeitintensive Teile in Assembler geschrieben werden, maschinenabhängige größere Teile in PLM86 und der Hauptteil als portable PASC86-Moduln; und wie unter 2. beschrieben, werden aus den PEARL-Anwenderprogrammen ebenfalls PASC86-Programme generiert.

Für das Binden, Verabsolutieren und Einbringen in eine Bibliothek aller benötigter Laufzeit-Routinen werden die Standard-Dienstprogramme LINK86, LOC86, LIB86 [7] benutzt.

Gebundene und verabsolutierte PEARL-Moduln bilden einen RMX86-Job. Zusammen mit den vorkonfigurierten RMX86-Jobs 'Terminal Handler' und 'Basic I/O' ist der PEARL-Job ein Abkömmling des RMX86-root-Jobs. Nach Binden und Verabsolutieren des PEARL-Jobs muß deshalb dieser root-Job noch erzeugt werden. (Dabei wird die Endadresse des PEARL-Jobs in das Konfigurationsfile des root-jobs eingetragen.) Danach müssen der RMX86 Nucleus, die RMX86-Jobs, der PEARL-Job und der root-Job in einer Bibliothek bereitgestellt werden, um ein ablauffähiges Programmsystem zu erhalten.

All dies erfolgt automatisch, gesteuert von einem Dialog-Programm, das die Schnittstelle zum Benutzer darstellt. Der Benutzer ist somit davon befreit, über die Techniken der Implementierung informiert sein zu müssen. Als Ergebnis des Dialogs werden Kommandoprozeduren erzeugt, die dann die oben beschriebenen Funktionen ausführen. Im einzelnen werden dabei die folgenden Schritte durchlaufen:

-- Erzeugen eines verabsolutierten PEARL-Jobs:
  - Binden der PEARL-Job Initialisierungs-Task
  - Binden der PEARL-Moduln
  - Binden der PEARL-Laufzeitroutinen
  - Verabsolutieren des Gebundenen

-- Erzeugen eines verabsolutierten root-jobs:
  - Eintragen der Endadresse des PEARL-Jobs in das Konfigurationsfile des root-jobs
  - Binden und Verabsolutieren des root-jobs

-- Erzeugen einer ladefähigen Bibliothek aus:
  - nuclus
  - basic I/O
  - terminal-handler
  - PEARL-job
  - root-job

6. Initialisierung

Die Initialisierung des gesamten PEARL-Programmsystems erfolgt durch die Initialisierungs-Task des PEARL-jobs. Diese Task wird ebenfalls als Ergebnis des unter 5. beschriebenen Dialogs generiert.

Im einzelnen werden dabei die folgenden Initialisierungen durchgeführt:

- pro PEARL-Task
  (z.B. Initialisieren des 8087-floating-point-prozessors)
- pro PEARL-Modul
  (z.B. Erzeugen von RMX86- und PASC86-Objekten entsprechend den PEARL-Objekten wie Tasks, Semaphoren etc.)
- global für alle PEARL-Moduln
  (z.B. Initialsieren des PEARL-Timers)
- global für das gesamte PASC86-Laufzeitsystem
  (z.B. Aufrufen der PASC86-Laufzeitroutine TQ001 zum Initialisieren der dynamischen Speicherverwaltung in PASC86)
- global für das Gesamtsystem:
  Aktivieren der als PEARL-Start-Task bestimmten Task(s)

Diese Struktur der Initialisierung ist in Abb. 3 skizziert.

7. Implementierungserfahrung

Wie bereits in der Einleitung betont, wurden bei der vorliegenden Implementierung von PEARL auf einem INTEL Mikrocomputer System soweit wie möglich die Funktionen des bereits existierenden Betriebssystems ausgenutzt. Weiterhin wurde eine höhere Programmiersprache bei der Übersetzung als Zwischensprache wie auch als System-Implementationssprache verwendet. Die Erfahrungen, die bei der Implementierung gewonnen wurden, betreffen im wesentlichen die Verwendung dieser Programmpakete.

Das RMX86 ist ein mächtiges, modulares und bis in feine Strukturen konfigurierbares Realzeit- und Multitasking-Betriebssystem. Um bei der Implementierung der Laufzeitfunktionen von PEARL möglichst wirtschaftlich sein zu können, mußten für die Verwendbarkeit des RMX86 wesentliche Betriebssystem-Funktionen zur Task- und Speicherplatz-Verwaltung sowie Standard-Ein-/Ausgabe einfach abbildbar bereits darin vorhanden sein. Die feinkörnige Konfigurierbarkeit des RMX86 half dabei, ein Betriebssystem zu erzeugen, das genau den gewünschten Satz an Funktionen enthält. Daß diese minimale Konfiguration dennoch einen

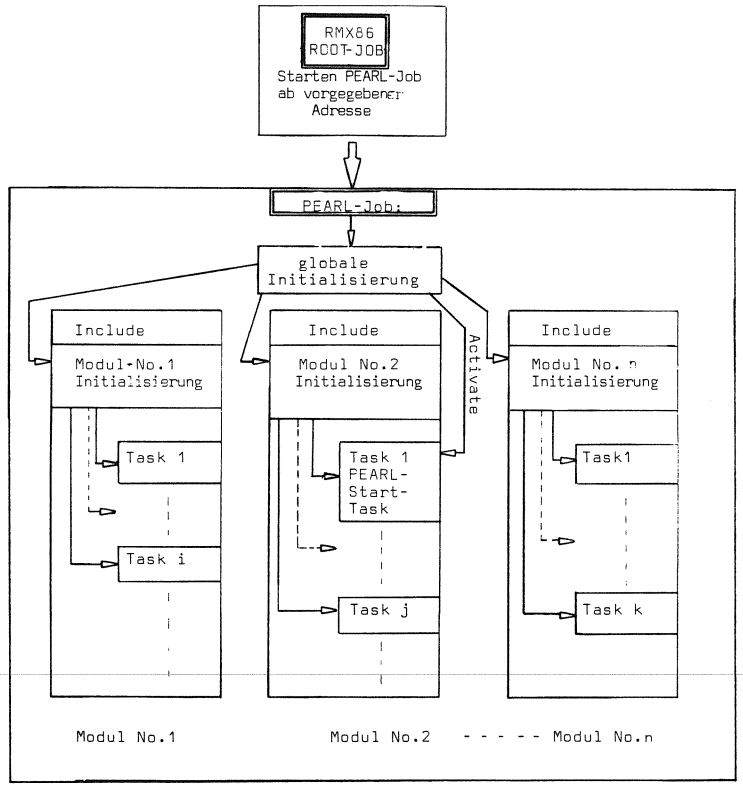

Abb. 3: Initialisierung des PEARL-Jobs

nach bisherigen Maßstäben hohen Bedarf an Speicherplatz erfordert (75 Kbyte), ist eine Eigenschaft des RMX86. Allerdings ist dieser hohe Bedarf an Speicherplatz bei einer Adressierbarkeit bis zu 1 Mbyte von geringerer Bedeutung. Dahingegen bleiben dem Benutzer sämtliche Funktionen erhalten, die durch die Struktur des RMX86 ermöglicht werden. So kann er z.B. für den Anschluß neuer Gerätetreiber voll die vom RMX86 dafür zur Verfügung gestellten Funktionen ausnutzen.

Bei der Implementierung der Laufzeitroutinen erwies sich als äußerst nützlich, daß standardisierte Aufrufkonventionen es ermöglichen Moduln verschiedener Sprachen problemlos zusammenzubinden. Dadurch konnten Funktionen entsprechend ihren Leistungsanforderungen in der geeignetsten Sprache implementiert werden z.B. wurden PEARL-Funktionen, die nicht unmittelbar auf RMX86-Funktionen abbildbar sind, in PASC86 portabel verwirklicht. An dieser Stelle sei zur Verwendung von PASCAL als System-Implementationssprache noch folgendes bemerkt:

PASCAL kristallisiert sich gegenwärtig nicht nur als eine der Standard-Sprachen für Mikrorechner heraus, sondern es erwies sich auch aufgrund seiner vorzüglichen Eigenschaften bzgl. Rekursivität als sehr geeignet für die Verwaltung eigener Warteschlangen, wie sie z.B. für die Implementierung von Zeiteinplanungen und BOLT-Variablen erforderlich waren. In der kombinierten Verwendung mit anderen Programmiersprachen, die für Hardwarenahe Programmierung wesentlich geeigneter sind, hat sich damit PASCAL auch für die Systemimplementierung als recht brauchbar erwiesen.

System-nahe Programmteile, wie z.B. die Abbildung der PEARL Ein-/Ausgabe auf RMX86 BASIC I/O Aufrufe wurden in der höheren INTEL-Systemimplementierungssprache PLM86 programmiert. Sie erlaubt für die betriebssystemabhängigen Anteile dennoch die Vorteile einer höheren Sprache ausnutzen zu können. Ansprüche auf Portabilität wurden bei diesen Betriebssystem-nahen Programm-Anteilen nicht gestellt, da bisher keine umfassenden standardisierten Schnittstellen für Tasking., realzeit- und Ein-/Ausgabe-System-Aufrufe existieren. Allerdings sind die UDI-Schnittstelle und das Konzept der Geräteunabhängigkeit ein erster Schritt in diese Richtung.

Für wenige zeitkritische Anteile wurde die ASM86-Assembler-Sprache benutzt.

Zusammengefaßt ist nach unserer Erfahrung die Forderung, problemlos Moduln verschiedener Sprachen zusammenbinden zu können, notwendig für eine leistungsfähige System-Implementierung mit vorgefertigten Programmteilen.

Als eine weitere Möglichkeit, portabel zu sein, wurde neben der Verwendung einer höheren Programmiersprache die Beschränkung auf die standardisierte INTEL-interne UDI-Schnittstelle angesehen. Sie wurde bei der Implementierung des PEARL-Compilers berücksichtigt. Dadurch ist dieser nicht nur unter der Kontrolle des ISIS-Betriebssystems ablauffähig, sondern auch unter allen künftigen Systemen, die der UDI-Schnittstelle genügen.

Die Berücksichtigung der UDI-Schnittstelle bei der Implementierung des PEARL-Laufzeit-Systems wurde jedoch nicht als vorteilhaft angesehen, da diese Schnittstelle erstens nur eine Untermenge der vom PEARL-Laufzeit-System benötigten und vom RMX86-Betriebssystem zur Verfügung gestellten Funktionen beinhaltet und zweitens eine weit umfangreichere Konfiguration des RMX86 erfordert.

Die Verwendung von PASCAL als Zwischensprache bei der PEARL-Übersetzung erwies sich als guter Kompromiß zwischen einer verlängerten Übersetzungszeit auf der einen und der Portabilität sowie der Sicherheit gegen Generierungs-Fehler auf der anderen Seite. So wurden die meisten Fehler, die zwangsläufig bei einer Installation eines neuen Compilers entstehen, bereits zur Übersetzungszeit durch den PASC86-Compiler abgefangen. Dieses Argument ist nach unserer Erfahrung für die Wirtschaftlichkeit der Implementierung von Bedeutung.

Zum PASC86-Compiler bleibt noch zu bemerken, daß bei der vorgestellten Implementierung nicht die offizielle Version verwendet wurde, sondern eine von der Fa. INTEL zur Ver-

fügung gestellte nicht-offizielle Version. Diese erlaubt auch, PASCAL-Programme zu bearbeiten, die sehr große Namenslisten erfordern; Beispiele dafür sind die einzelnen Pässe des PEARL-Compilers, die in PASC86-Code vorliegen.

8. Fazit

Zusammenfassend ist zu sagen, daß beim derzeitigen Stand der auf Mikrorechner existierenden Software, eine wirtschaftliche und dennoch leistungsfähige Implementierung von PEARL in wachsendem Maße möglich wird, durch Ausnutzung der Funktionen leistungsfähiger Realzeit-Betriebssysteme, sowie standardisierter Schnittstellen und Hochsprachen.

Literaturverzeichnis

[1] INTEL, PASCAL-86, USER's Guide Manual

[2] INTEL, Introduction to the RMX86 Operating System Manual

[3] W. Werum, H. Windauer: PEARL, Vieweg-Verlag 1978

[4] Intel, INTELLEC Series III, Microcomputer development console Operating Instructions Manual

[5] INTEL, Runtime Support Manual for iAPX86, 88 Applications

[6] INTEL, The 8086 Family Users Manual Numerics Supplement, July 1980

[7] INTEL, ISIS-II Users Guide Manual