Realisierung einer Robotersteuerung mit PEARL

P.-J. Becker, K.H. Meisel, M. Salaba, W. Schill, G.Bonn IITB Karlsruhe

Zusammenfassung

Eine Robotersteuerung wurde vollständig auf einem Intel 3000 Mehrrechner System in der höheren Programmiersprache PEARL programmiert. Durch die Verwendung eines ausgereiften PEARL-Compilers, spezielle Betriebssystemanpassungen und die Einhaltung bestimmter Programmierregeln ist es gelungen, eine Abtastperiode der Regelung für einen 5-Achsen-Roboter von 13 msec zu erreichen. Dieser Wert liegt nur etwa 30 % über der Laufzeit vergleichbarer Assembler-Programme.

Summary

A robot control system has been implemented on a multicomputer system where the programs were completely written in the high level language PEARL (Process and Experiment Automation Realtime Language). The sampling-period of the feedback control loops for a five-axes-robot was 13 ms, about 30% longer than corresponding assembler programs. This was reached by the use of an optimised PEARL-compiler, some special functions in the operating system and by observing a few compiler and adapted operating system rules for time efficient programming.

1. Einleitung

Die Implementierungssprachen der meisten heutigen Steuerungen für Industrieroboter sind steuerungspezifische Assembler. Vereinzelt werden für zeitunkritische Teile einer Robotersteuerung einfache Hochsprachen (z. B. PLM) eingesetzt. Beim am IITB realisierten Einsatzfall "Roboter als Werkzeugmaschine" [1] wurden Hartmetalloberflächen mit einem KUKA IR 250 Roboter und einer im IITB entwickelten Steuerung gefräst. Dabei wurde die Steuerung vollständig in PEARL programmiert.

Wesentliche Komponenten einer Robotersteuerung sind:

- Regelung der Roboterachsen,
- Interpolation, Bahnberechnung,
- Koordinatentransformationen,
- Sensorschnittstellen,
- Schnittstellen zu übergeordneten Leitsystemen,
- Bedienerschnittstelle, "Programmierung" des Roboters.

Bei der Sensordatenverarbeitung und bei der Regelung werden hohe Echtzeitforderungen an eine Robotersteuerung gestellt. Aber auch Interpolation und Koordinatentransformation erfordern hohe Rechengeschwindigkeiten und damit eine sehr gute Ablaufeffizienz. Die Verschiedenartigkeit der Aufgaben, die von Robotern bearbeitet werden, erfordert in der Steuerungssoftware eine hohe Modularität. Die Komplexität, u. a. bei Einbeziehung von Sensorinformationen in den Arbeitsablauf, macht es notwendig, parallele Aktivitäten programmieren zu können.

Grundlage für die Modularität in der Steuerungssoftware ist die Programmierung in Hochsprache mit Echtzeitverarbeitung und ereignisgesteuerten parallelen Aktivitäten (Multitasking).

Dies bedeutet nicht, daß eine Steuerung, die in Maschinensprache programmiert ist, nicht auch modular aufgebaut sein kann. Sollte jedoch der Entwickler oder gar der Anwender im Stande sein, eine Robotersteuerung kurzfristig an andere Aufgaben-

stellungen oder Sensorsysteme anzupassen, so ist dies durch die Programmierung in einer Hochsprache wesentlich erleichtert.

Die oft diskutierten Probleme einer Hochsprache gegenüber ASSEMBLER-Programmen

- erhöhter Speicherbedarf,
- geringere Ablaufeffizienz gegenüber guten ASSEMBLER-Programmen

wurden durch die Vorteile

- gute Selbstdokumentation,
- schnelle Programmentwicklung,
- leichte Änderbarkeit und Erweiterungsmöglichkeit auch durch nicht bei der Entwicklung beteiligte Programmierer

aufgewogen, sofern z. B. die Verminderung der Ablaufeffizienz nicht die Aufgabenlösung verhindert.

Bei Verwendung von Hochsprachen ist die Beschaffenheit des Compilers und des Betriebssystems für die erreichbaren Geschwindigkeiten von mindestens genau so hoher Bedeutung wie der verwendete Prozessor.

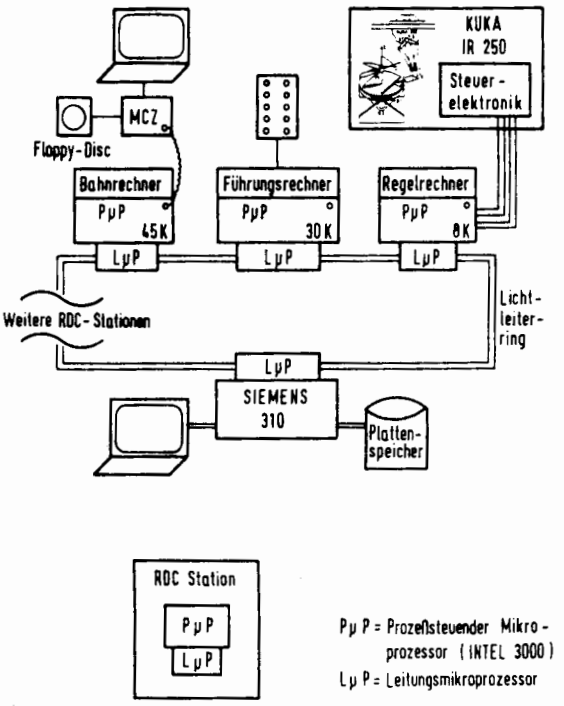

Bild 1: Gerätestruktur der realisierten Robotersteuerung.

2. Aufbau der Robotersteuerung

Die Robotersteuerung besteht aus drei im IITB auf Basis des Mikroprozessors INTEL 3000 entwickelten Prozessrechnern (RDC-Stationen) [2], die zusammen über einen Lichtleiterring kommunizieren. Jeder Rechner besteht aus zwei Prozessoren

- dem prozeßführenden Mikroprozessor PμP und
- dem Leitungsmikroprozessor LμP.

Die beiden Prozessoren arbeiten auf einem gemeinsamen Speicher mit der Kapazität von 256 K Worte (16 Bit), der in einen Rootsegment und 8 Kacheln aufgeteilt ist.

Die RDC-Stationen arbeiten als

- Reglerrechner,
- Führungsrechner und
- Bahnrechner.

An dem Bahnrechner ist über eine V24-Schnittstelle ein MCZ-Rechner angeschlossen, der als Bedienungsrechner arbeitet. Eine SIEMENS-310 mit Plattenlaufwerk benutzt man als Laderechner (B i l d 1).

Der Reglerrechner regelt direkt den KUKA IR 250-Roboter (B i l d 2). Er übermittelt

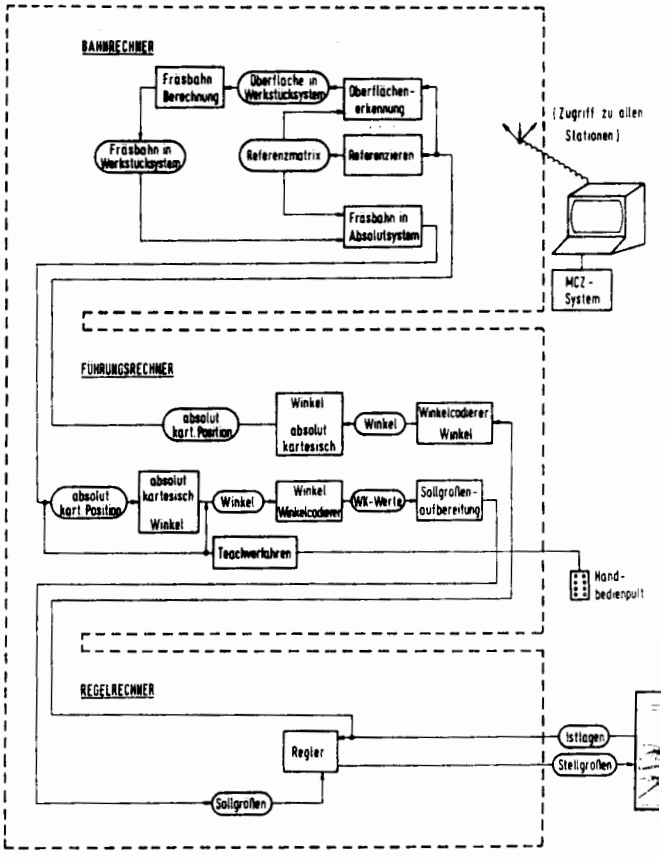

Bild 2: Datenfluß innerhalb der Robotersteuerung.

die Roboterlagen dem Führungsrechner und wird von ihm versorgt mit den Sollagen.

Der Führungsrechner führt den Roboter nach den Angaben des Bahnrechners, besorgt die Koordinatentransformationen Roboterkoordinaten - kartesische Koordinaten und umgekehrt und im Teach-Verfahren kann er den Roboter vom Handbedienpult aus führen.

Der Bahnrechner berechnet die Roboterbahn in kartesischen Koordinaten aus den eingelernten Daten, speichert die Daten im Teach-Verfahren und kommuniziert mit dem Bedienungsrechner. Die Fräsbahn wird in Werkstückkoordinaten berechnet, die durch das Referenzieren vor dem Einlernen in Bezug zu dem Roboterkoordinatensystem gebracht werden.

Alle Rechner bis auf den MCZ-Rechner sind in PEARL programmiert.

3. Optimierung im Compiler und in der Firmware

Für die Steuerung des Roboters KUKA IR 250 wurde der PEARL-Compiler der Firma Werum [3] für SIEMENS-Serie 300 verwendet. Bei der Programmentwicklung wurde der Weg über den Assembler SIEMENS-ASS300 gewählt, damit man während der Testphase Speicherabbild zur Verfügung hat, obwohl der Compiler den Maschinencode direkt generieren kann.

Mit dem PEARL-Testsystem PETSY des IITB steht inzwischen auch eine effiziente Testhilfe auf Hochsprachenebene zur Verfügung [4].

Die Roboterprogramme wurden mit der höchsten Optimierungsstufe des Compilers übersetzt. Diese Optimierungsstufe setzt u. a. im Code ab

- mikroprogrammierte Gleitkommabefehle anstatt assemblerprogrammierte Laufzeitroutinen,
- im Compilerlauf berechnete feste indizierte Adressen für eindimensionale Felder (keine Laufzeit-Indizierungsroutinen).

Zur Erhöhung der Programmeffizienz wurden im IITB folgende zusätzliche Maßnahmen getroffen.

Der Mikroprogrammspeicher wurde erweitert um eine komplett mikroprogrammierte Gleitkommaarithmetik mit Wandelbefehlen und Doppelwort-Ladebefehlen und um einige spezielle Befehle, die den Ablauf der PEARL-Programme beschleunigen, wie

- Laden/Speichern Byte-Gruppe von/auf EA-Adresse
- Wortgruppentransfer
- Wortgruppentransfer zwischen Kacheln
- Parameteradresse aus Parameterbeschreibung berechnen
- Freispeicher belegen
- Freispeicher freigeben.

Die ersten drei Befehlstypen wurden zur Verbesserung der Transferzeiten bei der Rechnerkommunikation geschaffen, die letzten drei Befehle zur Unterstützung der PEARL-Prozeduren, die wegen ihrer Reentrant-Eigenschaft ihre Variablen im Freispeicher dynamisch verwalten müssen.

Das Laufzeitsystem wurde um Kommunikationsprozeduren, Prozeduren zum Starten der Module und Kacheln und einige andere Prozeduren erweitert:

- ABSADR - Übergabe der absoluten Adresse einer Variablen
- HOLGM/ZUWGM - Holen-/Zuweis eines Datenfeldes im direkten Speicherzugriff aus/in einen anderen Rechner bzw. Kachel
- SENDF/RECEIV - Intertask- und Interrechnerkommunikation im Handshake-Verfahren mit Warten auf Telegramm
- GETIME/SETIME - Zeitzähler lesen/setzen
- GETKNR/SETKNR - Kachelregister lesen/setzen
- MODBEK/MODKIL - PEARL-Modul starten/beenden
- SEGBEK - den ersten Modul in einer Kachel starten
- MASKEE/DMASKE - Ebene maskieren/demaskieren

Eine sehr große Ersparnis an Speicherplatz und Laufzeit bringt eine Inkonsequenz in der Typ-Prüfung im Compiler. Man kann einen Pointer eines Typs auf eine Variable eines anderen Typs einstellen.

```
DCL  A  BIT(16);
DCL  P  REF FIXED;

P := A;
```

Der Compiler meldet an der Stelle nur eine WARNUNG und setzt den richtigen Code ab. Damit wird z. B. eine Wandlung TOFIXED eingespart, die wegen der Definition BIT(16) und FIXED(15) nur umständlich programmiert werden kann. Weiterhin wird damit ermöglicht, bei der Rechnerkommunikation eine Pufferkette zu verwenden, in die verschiedene Tasks Telegramme unterschiedlicher Typen abspeichern, die dann mit einer einzigen Sendetask übertragen werden.

Das Betriebssystem stellt eine Anwenderschnittstelle zur Verfügung, an die man besonders zeitkritische, im Assembler programmierte Routinen anhängen kann, wie z. B. die digitale Filterung der Eingangssignale oder die Behandlung von speziellen Baugruppen. Diese Schnittstelle wurde für die Roboteranwendung nicht genutzt.

Die Robotersteuerung verwendete das vorher geschilderte System ohne jede zusätzliche Änderung. Die einzige Anpassung des PEARL-Laufzeitsystems bestand aus speziellen schnellen TAKE- und SEND-Anweisungen, die sowieso der jeweiligen Ein-/Ausgabekonfiguration angepaßt werden müssen.

4. Effiziente Programmierung in PEARL

Um schnelle und kurze Programme in einer höheren Programmiersprache wie PEARL zu erzielen, muß man bestimmte durch die Sprachstruktur und die Befehlsstruktur des jeweiligen Rechners und Compilers bedingte Regeln einhalten. Im weiteren werden einige Regeln, die für den angewendeten Compiler gelten, vorgestellt, die jedoch z. T. für PEARL allgemeingültig sind.

- Strukturen anstatt indizierte Felder anwenden. Bei einem Feldelement wird die Elementadresse erst zur Laufzeit berechnet, auch dann, wenn der Index fest ist (Laufzeit ca. 500 µs pro Index). Eine Strukturelementadresse wird dagegen zu Compilerzeit berechnet (Zugriffszeit 5,6 µs).
- Eindimensionale Felder anstatt mehrdimensionale verwenden. Die Laufzeit für die Indexberechnung steigt mit der Anzahl der Dimensionen. Außerdem werden die Elementadressen bei eindimensionalen deklarierten Feldern mit festen Elementindizes bei entsprechender Compilereinstellung schon zu Compilerzeit berechnet.
- Bei indizierten Feldern Pointer verwenden. Die Pointer werden im Vorlauf auf häufig benutzte Feldelemente eingestellt, zur Laufzeit wird dann auf die Elemente über die Pointer zugegriffen.

  Beispiel:

```
MODULE;
SYSTEM;
        BINEIN (1:3):  DIINO(0)*(1:3)   <- ;
PROBLEM;
        SPC BINEIN () DATION IN BASIC;
        DCL BEINO REF DATION IN BASIC;
ANLAUF: TASK;
        BEINO := BINEIN (1);
        ALL 0.1 SEC ACTIVATE REGLER;
END;    /***   ANLAUF   ***/

REGLER: TASK;
        TAKE WKEIN (1)  FROM BEINO;
END;    /***   REGLER   ***/

MODEND; /***   MODULE   ***/
```

  Hier optimiert man den Zugriff zu der Binäreingabe, weil die Compileroptimierung die Indizierung im Systemteil nicht erfaßt, dagegen wird die Adresse WKEIN(1) vom Compiler berechnet. Ohne Optimierung würde die Prozedur OFFSET abgesetzt, die etwa 180 µs zusätzliche Laufzeit bringt.
- Die Deklarationen in die zeitkritischen Module legen. Auf die spezifizierten Variablen und Strukturen wird immer über deren Adresse (referenziert) zugegriffen, da die Adressen erst durch das Binden bekannt werden. Auf spezifizierte Felder wird über deren Feldbeschreibung zugegriffen.
- Bei Bit- und Charakter-Variablen Masken benutzen. Bei der normalen PEARL-Schreibweise

```
DCL BEREIT BIT(16);
IF BEREIT.BIT(15) THEN
```

wird eine langsame Bitselektor-Prozedur abgesetzt (Laufzeit ca. 250 µs).
Wenn man dagegen Masken verwendet,

```
DCL BEREIT BIT(16);
DCL BIT15 BIT(16) IN Γ('0001'B4);
IF (BEREIT AND BIT15) == BIT15  THEN
```

reduziert sich die Laufzeit der gleichen Abfrage auf ca. 20 µs.

In PEARL sind die Operationen auf Bit-Variablen, wie SHIFT, CSHIFT, Bitselektor A.BIT(3:11) sehr zeitaufwendig.

- In den Abfragen möglichst BIT(1)-Variablen benutzen.

  Die Abfrage

```
DCL B BIT(1);
IF B THEN
```

  wird in einen einzigen Assembler-Testbefehl und anschließenden Sprungbefehl umgewandelt. Alle anderen Abfragen kosten viel mehr Zeit und Platz.

- FLOAT-Arithmetik anstatt Doppelwort-FIXED-Arithmetik. Eine allgemeine Doppelwort- FIXED-Arithmetik ist zeitaufwendiger als eine FLOAT-Arithmetik. Zum Vergleich dauert eine Multiplikation in

  FIXED(31)-Assemblerroutine 280 µs
  FLOAT(23)-Assemblerroutine 65 µs.

  Die notwendige Normierung bei der FIXED-Arithmetik verlangsamt den Ablauf noch zusätzlich. Dies ist speziell bei der Robotersteuerung von Bedeutung, da man mit Achsen-Lagezählern rechnen muß, die meistens breiter als 16 Bit sind.

- Negative Konstanten statisch initialisieren. Im Ausdruck

```
IF DELTWK < -32767  THEN
```

  wird die positive Konstante 32767 abgespeichert und zur Laufzeit zuerst von Null abgezogen und dann verglichen. Besser ist:

```
DCL M32K FIXED INIT(-32767);
IF DELTWK < M32K  THEN
```

- Wenige Prozedurparameter verwenden.
  Jede Parameterübernahme in eine Prozedur kostet Zeit. Möglichst parameterlose Prozeduren, die mit auf Modulebene deklarierten Daten arbeiten, verwenden, oder möglichst viele Parameter in eine Struktur zusammenfassen und die Struktur als einen Parameter der Prozedur übergeben.

5. Ergebnisse

Bei den erreichten Ergebnissen ist zu berücksichtigen, daß der verwendete Compiler in einer sehr ausgereiften Form mit Optimierungsoptionen sowohl in Richtung Rechengeschwindigkeit als auch in Richtung Speicherbedarf zur Verfügung stand [2].

Das PEARL-Laufzeitsystem wurde durch spezielle mikroprogrammierte Befehle unterstützt und es wurden zusätzliche Funktionen, die nicht im Sprachumfang von PEARL enthalten sind, in das Laufzeitsystem aufgenommen. Die Programme wurden unter Beachtung der in Kap. 4 geschilderten Programmierregeln entwickelt, so daß z. B. alle Datenfelder als Strukturen oder eindimensionale Felder angelegt worden sind, die Prozeduren arbeiteten meistens auf festen, auf Modulebene deklarierten Daten. Die Regelung wurde linear ohne Schleifen programmiert.

Dagegen zeichnete sich der verwendete Prozessor in den benutzten RDC-Stationen [3] nicht durch hohe Rechengeschwindigkeit aus. (Befehlssatz SIEMENS PR310S, Arbeitsspeicher-Zykluszeit 900 ns, Mikrobefehlsausführungszeit 200 ns, Befehlsausführungszeit einfacher Befehle 4,4 - 5,6 µs)

Für die Regelung von 5 Achsen wurde eine Abtastperiode von ca. 13 msec erreicht, dabei betrug die reine Rechenzeit des Reglers ca. 9 msec, der Rest entfällt auf den Taskanlauf (ca. 1 msec) und auf die Kommunikation. Die Regelung lief unter der normalen PEARL-Betriebssystem-Steuerung. Damit war es möglich, bei Bahngeschwindigkeiten des Roboters von bis zu 30 mm/sec Oberflächen mit Restwelligkeiten ≤ 0,1 mm zu bearbeiten. Die Rechenzeit lag ca. 30 %

höher als von vergleichbaren ASSEMBLER-Programmen. Für die Koordinatentransformation für den KUKA-Roboter IR 250 von externen kartesischen in interne Roboterkoordinaten wurden ca. 80 msec benötigt. Die Programmentwicklung der IITB-Steuerung mit dem KUKA IR 250 (Fräsanwendung) betrug mit PEARL ab Vorlage der Algorithmen bis zur Erstellung von ca. 80 k Worte Programm ca. 1 Mann-Jahr.

Schrifttum

[1] Becker, P.-J.; K.H.Meisel; W. Schill; M. Salaba: Roboter als Werkzeugmaschine im Versuchsbetrieb. Mitteilungen aus dem Fraunhofer-Institut für Informations- und Datenverarbeitung (IITB) (1983).

[2] Heger, D.; H. Steusloff; M. Syrbe: Echtzeitrechnersystem mit verteilten Mikroprozessoren. Forschungsbericht DV 79-01, Datenverarbeitung (1979).

[3] Windauer, H.: The Portable PEARL-Programming System of WERUM. PEARL-Rundschau, Heft, 1, Bd. 3 (1982).

[4] Heine, P.; von Stülpnagel, F.: PETSY, ein Pearl-Testsystem zum Austesten von physikalisch verteilten Pearl-Programmen unter Realzeitbedingungen. PEARL-Rundschau, Band 2, Nr. 6 (1981).

Anschrift der Autoren

Dr. P.-J. Becker
Dipl.-Inform. G. Bonn
Dipl.-Inform. K.-H. Meisel
Dr. M. Salaba
Dipl.-Ing. W. Schill

Fraunhofer-Institut für Informations- und Datenverarbeitung (IITB)
Sebastian-Kneipp-Str. 12/14
7500 Karlsruhe 1

Tel. (07 21) 60 91-332
-301
-311
-272
-360