Ein quellenbezogenes Testsystem für PEARL auf einem Prozeßrechner

Dipl.-Math. Karlotto Mangold, Konstanz

Zusammenfassung:
Das quellbebezogene PEARL-Testsystem QPTS für einen 16-Bit Prozessrechner erlaubt den Test von PEARL-Anwendungssystemen ohne Kenntnis der Hardware und des Betriebssystems. Es werden Aufbau und Funktion beschrieben, sowie der Test exemplarisch dargestellt.

Summary:
The source-code related PEARL-Debugger QPTS implemented on a 16-bit process control computer allows debugging of PEARL-application-systems without knowledge of hardware and operating-system. Structure and functions are described and an example is given.

1. Einleitung

Mit der Entwicklung der höheren Programmiersprachen, die es erlaubten, zunehmend problemorientiert und immer weniger maschinenorientiert zu programmieren, kam auch die Forderung nach quellbezogenen Testhilfsmitteln auf.

Bereits Mitte der sechziger Jahre gab es die ersten Maschinen (z.B. TR4) mit Post-Mortem-Dump auf Quellebene für FORTRAN <1>.

Dieser Ansatz wurde im Laufe der Zeit immer weiter vervollständigt und führte zu recht benutzerfreundlichen sprachbezogenen Testhilfen im Dialog- und Batchbetrieb mit statisch oder dynamisch definierten Haltepunkten <2>. Diese Entwicklung bezog sich jedoch einerseits auf die herkömmlichen Programmiersprachen, andererseits setzte sie grosse Rechensysteme voraus. Auch der erste mir bekannte Ansatz eines Testsystems für PEARL führte noch die Forderung nach einem Grossrechner im Titel <3>.

Während es mit PEARL wesentlich einfacher wurde, auch komplizierte Anwendungssysteme zu implementieren, musste das Austesten zunächst noch auf der Ebene des erzeugten Codes erfolgen. Diese Vorgehensweise erzwang beim Programmierer detaillierte und fundierte Kennntnisse der Objektstruktur und des vom Compiler erzeugten Codes und brachte damit keine wesentliche Erleichterung gegenüber einer Implementierung in Assemblersprache.

Inzwischen gibt es eine Reihe von allgemein anerkannten Anforderungen an ein solches sprachbezogenes Testsystem <4,5>, und die ersten Realisierungen sind verfügbar <6>. Es sollen nun hier, ausgehend von den Anforderungen an ein solches Testsystem, Aufbau, Wirkungsweise, Funktionen, Benutzerschnittstelle und ein Beispiel dargestellt werden.

2. Anforderungen

An das zu entwickelnde Testsystem wurden folgende Anforderungen gestellt, wobei die Reihenfolge die Wichtigkeit angibt.

- Einsatz auf dem operationellen Rechner
- Keine Code- und Laufzeitverlängerung für ausgetestete Objekte
- Modularer Aufbau, sodass auf einzelne Funktionen verzichtet werden kann, wenn die zugehörige Leistung nicht gebraucht wird.
- Erstellung und Änderung eines Testplans ist ohne Änderung eines bereits geladenen Anwendersystems möglich
- Unterstützung beim Test von Realzeit Operationen im PEARL-Programm
- Möglichkeit der Simulation der PEARL Anweisungen durch Testkommandos
- Eingeschränkte Testmöglichkeit in Hauptspeichersystemen

Nach den ersten Erfahrungen mit der Realisierung auf Prozessrechnern des Typs AEG 80-20 M bzw. ATM 80-30 kann heute festgestellt werden, dass die Anforderungen erfüllt werden konnten.

3. Aufbau und Struktur

Das Testsystem ist in die vorhandene Systemumgebung und das Programmiersystem eingebettet. Zur Ablage von Testhilfeinformationen und Testplänen werden Dateien der allgemeinen Dateiverwaltung eingesetzt. Falls die Protokollausgabe nicht direkt auf ein Gerät erfolgen soll, wird diese in einer Datei gepuffert.

Der Aufbau des Testsystems ist in Abbildung 1 dargestellt.

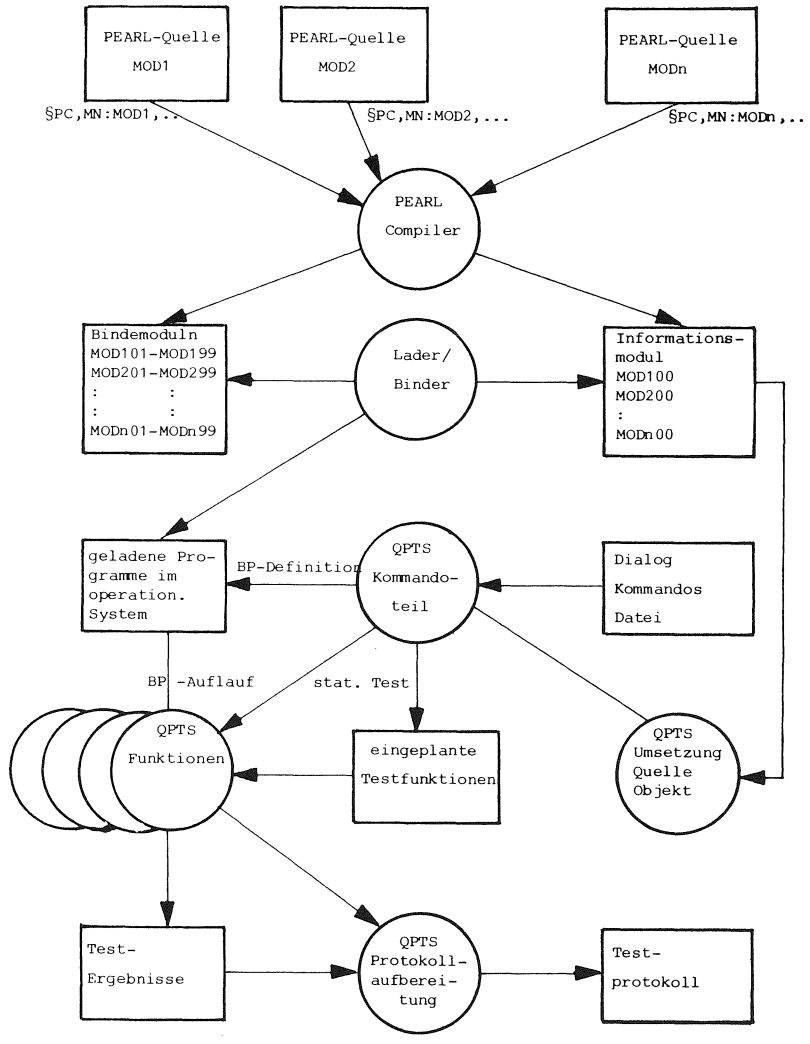

Abb.1 Aufbau und Struktur des Testsystems

Aus einem oder mehreren PEARL-Quellmoduln erzeugt der PEARL-Compiler neben den Bindemoduln noch je einen Informationsmodul, der u.a. die Zuordnungen zwischen der Quelle und dem generierten Objekt enthält. In diese Informationsmoduln werden bei der Systemgenerieung oder beim Laden der Bindemoduln weitere Informationen über die programmrelative Ablage der Bindemoduln eingetragen.

Das eigentliche QPTS besteht aus mehreren Teilen, deren Funktion kurz skizziert werden soll. Der QPTS-Kommandoteil bezieht vom Dialog oder aus einem Testplan die ihn steuernden Testanweisungen. Die Quellbezüge des Benutzers werden vom QPTS-Umsetzer mit Hilfe des zugehörigen Informationsmoduls in Maschinenadressen umgesetzt.

Die Ausführung der einzelnen Leistungen erfolgt durch die eigentlichen QPTS-Funktionen. Sie werden vom Kommandoteil beauftragt und gesteuert. Insgesamt sind dies etwa fünfzehn Moduln, die das gesamte Leistungsspektrum abdecken. Bei Verzicht auf einzelne Funktionen kann der zugehörige Modul bei der Systemgenerierung weggelassen werden, so dass dadurch eine einfache Anpassung an den gewünschten Leistungsumfang möglich ist.

Die Testergebnisse werden entweder direkt, oder nach Zwischenablage in einer Datei, durch die QPTS-Protokollfunktion aufbereitet und als Testprotokoll ausgegeben.

4. Funktionsbeschreibung

4.1 Allgemeines

Das quellbezogene PEARL-Testsystem QPTS, das hier vorgestellt werden soll, erlaubt den Test von PEARL-Programmen auf Quell ebene, d.h. QPTS ermittelt aus den vom Benutzer eingegebenen Bezeichnern von Programmgrössen (zulässig sind Nummern von Quellzeilen und Namen) die zugehörigen Code- und Datenadressen im Speicher.

Der Benutzer teilt seine Wünsche in Form eines sogenannten Testplans dem Testsystem mit. Der Testplan ist eine Folge von Testkommandos und kann sowohl im Dialog eingegeben, als auch in einer Datei vordefiniert werden. Im interaktiven Modus ist es jederzeit möglich, vom QPTS aus in ein laufendes Anwendersystem einzugreifen. Wie bei Testsystemen für herkömmliche Sprachen können dann Datenobjekte ausgegeben und ggf. verändert, sowie der

Programmablauf dokumentiert (einschränkbarer Trace) werden. Darüberhinaus können auch Programm- und Systemzustände dokumentiert werden. Schliesslich lassen sich durch eine Vielzahl von "Realzeit-Testkommandos" die Taskabläufe beeinflussen.

Die gewünschten Testleistungen können sofort erbracht werden - wir wollen dies als statischen Test im Dialogmodus bezeichnen - oder durch Definition eines Breakpoints im Testling eingeplant werden. Letzteres bezeichnen wir als dynamischen Test, da die Testfunktionen erst dann ausgeführt werden, wenn der Testling während seines dynamischen Laufes einen solchen Breakpoint erreicht.

Um durch das Testsystem das Realzeitverhalten des Anwendungssystems möglichst wenig zu stören, läuft der Testling zwischen den Haltepunkten mit normaler Geschwindigkeit. Ausserdem besteht die Möglichkeit, am Eingang in das Testsystem die Systemuhr anzuhalten. Das Protokoll des Tests kann wahlweise auf ein Gerät oder in eine Datei ausgegeben werden.

4.2 Datenzugriffe

Da PEARL blockorientiert ist, sind Gültigkeit und Lebensdauer von Datenobjekten durch den Block bestimmt, in dem sie definiert sind. Beim Eintritt in einen Block existieren automatisch alle in ihm definierten Datenobjekte. Sind Blöcke ineinander verschachtelt, so kann an einer beliebigen Stelle im Block nicht nur auf die lokalen Daten des aktuellen Blocks, sondern auch auf alle Datenobjekte der umfassenden Blöcke zugegriffen werden. Dabei gilt jeweils der Modul als "äusserster Block". Für die von QPTS erreichbaren Datenobjekte des zu testenden Quellmoduls gilt nun als aktuell innerster Block derjenige, welcher den Breakpoint (Quellzeile) unmittelbar enthält, der zur aktuellen Aktivierung des Testsystems geführt hat. Beim externen Start des Testsystems wird die Modulebene des jeweiligen Quellmoduls von QPTS zugreifbar, wenn nur der entsprechende Modul eingestellt ist. Beim Zugriff auf Daten des Testlings wird bei Referenzobjekten immer automatisch dereferenziert.

4.3 Testkommandos

Die Testkommandos genügen einer einfachen Syntax. Sie beginnen mit dem aus maximal zwei Zeichen bestehenden Kommandonamen, gefolgt von den Kommandoparametern. Zwischen dem Namen und dem ersten Parameter, sowie zwischen weiteren Parametern steht als Trenner eine Folge von Leerzeichen und/oder ein Komma.

Da die Quellbezüge immer nur innerhalb eines Quellmoduls eindeutig sind, wird mit dem Kommando MN (ModulName) immer genau ein Modul eingestellt. Bei Wechsel des Quellmoduls muss der neue Testling mit dem Kommando MN neu eingestellt werden.

Die Testkommandos werden unterschieden in Definitions- und Funktionskommandos.

4.3.1 Definitionskommandos

Die Definitionskommandos erbringen testsystemglobale Leistungen und werden stets sofort (statisch) ausgeführt. Ihre Ausführung wird nicht quittiert. Im einzelnen gibt es folgende Definitionen:

- zur Beschreibung der notwendigen Dateien. Dazu gehören Kommando-Eingabe-, Protokoll-Ausgabe- und Testinformationsdatei.
- zur Identifizierung des zu testenden Quellmoduls
- zum Setzen eines Breakpoints in einer bestimmten Quellzeile. Dabei ist eine Einschränkung der Aktivierungshäufigkeit möglich. Es gibt:
  - die einmalige Aktivierung nach genau n-maligem Erreichen,
  - die zyklische Aktivierung nach jeweils n-maligem Erreichen,
  - die ständige Aktivierung von n-maligem Erreichen an.
- zur Aktivierung, Passivierung und zum Löschen von Breakpoints
- zur Beendigung der dynamischen Testplaneingaben.

4.3.2 Funktionskommandos

Die Funktionskommandos spezifizieren die vom Testsystem zu erbringenden Leistungen. Sie werden im Dialogmodus sofort (statisch) ausgeführt, können aber auch in einem Testplan einem Breakpoint zugeordnet werden. In diesem Fall werden sie erst dann (dynamisch) ausgeführt, wenn der Testling diesen Punkt erreicht. Die einzelnen Funktionen sind:

- Manipulation der Systemuhr, insbesondere Anhalten der Systemuhr während des Laufes des Testsystems, um zeitliche Einplanungen und Weckaufträge nicht durch Aktionen des Testsystems zu stören, sowie Ausdrucken der Systemzeit beim Auflaufen auf einen Breakpoint.
- Typgerechter Dump von Ausdrücken und höheren Objekten.
- Typgerechter Dump von Werten einfacher Variabler und höherer Objekte mit anschliessender Änderung.

Bei diesen beiden Funktionen wird beim Zugriff Indizierung und Selektion zugelassen.

- Umschalten von Datei-Eingabe in den Dialogmodus.
- Beenden des einem Breakpoint zugeordneten Testplans und Fortsetzen des Testlings hinter dem Breakpoint oder an einer beliebigen Stelle in demselben Block. Optional kann gleichzeitig der Breakpoint gelöscht werden.
- Bedingte Ausführung eines Funktionskommandos in Abhängigkeit des Vergleichs zweier Ausdrücke vom Typ fixed, float, clock, dur, char(1) oder bit.

Neben diesen Funktionen gibt es eine Reihe von Funktionskommandos, die Operationen mit Tasks, Semas, Interrupts und Alarmen ausführen, wie sie in PEARL definiert sind. Damit besteht die Möglichkeit, Taskabläufe vom Testsystem aus zu beeinflussen. Im einzelnen sind dies:

- die Taskoperationen ACTIVATE, TERMINATE, PREVENT, SUSPEND und CONTINUE, jeweils mit einem Taskbezeichner als Parameter
- die Semaoperationen RELEASE und REQUEST
- die Interruptoperationen TRIGGER, ENABLE und DISABLE
- die Signaloperation INDUCE, sowie die Definition von Responses. Durch den Anschluss der Responseverwaltung an das Testsystem bekommt dieses im Fehlerfall die Regie und bietet dann dem Benutzer die Möglichkeit Umgebung und Ursache des Fehlers zu analysieren.

Ausser diesen Operationen gibt es eine Reihe von Auskunftsdiensten, die den Status von Tasks, Semas und Schedules ausgeben. Dabei können:

- der Wert der Zählgrösse eines Semas
- die Folgeaktivierungen einer Task
- die Einplanungen einer Task
- der Zustand einer Task

protokolliert werden. Mögliche Zustände sind dabei aktiv, suspendiert, beendet, wartend auf Sema, im Breakpoint und wartend auf Testsystem.

Daneben gibt es weitere Funktionen zur Ablaufüberwachung und -dokumentation des Testlings. Hier sind besonders zu erwähnen:

- Zeilentrace: Dynamische Protokollierung des Testlings auf Quellzeilenebene. Dieser Trace ist auf Zeilenbereiche einschränkbar.
- Prozedurtrace: dynamische Protokollierung von Prozeduraufrufen mit Ausgabe des Modulnamens und der Zeilennummer
- Tasktrace: dynamische Protokollierung aller Taskoperationen und Einplanungen für die jeweils spezifizierte Task mit Quell-Lokalisierung der auslösenden Anweisung.
- Schleifenkontrolle: Bei Laufanweisungen werden Anfangswert, Schrittweite und Endwert protokolliert, bei jedem Schleifendurchlauf wird die Laufvariable und die Bedingung ausgegeben.
- Verzweigungskontrolle: Beim dynamischen Erreichen von Programmverzweigungen (IF, CASE) wird die Verzweigungsentscheidung (THEN- oder ELSE-Zweig bzw. die Nummer der Alternative bei CASE) protokolliert.

Im Gegensatz zu den Definitionskommandos werden die Funktionskommandos stets auf dem eingestellten Protokoll-Ausgabemedium protokolliert. Dabei werden auch ggf. auszugebende Werte dokumentiert.

5. Beispiel

An einem relativ einfachen Beispiel, das jedoch die wesentlichen Sprachelemente von PEARL enthält, soll nun der Dialog und die Benutzerschnittstelle des Testsystems dargestellt werden.

Zunächst das Quellprogramm:

```
  5  MODULE;
 10  SYSTEM;
 15    FSR:FSR101;
 20    INT1:SWIRPT(1);
 25    INT2:SWIRPT(2);
 30  PROBLEM;
 35    SPC FSR DATION INOUT ALPHIC  DIM(,)
 36        TFU MAX CONTROL (ALL) GLOBAL;
 40    SPC (INT1,INT2) IRPT GLOBAL;
 45    TYPE EL STRUCT(/ (S1,S2) FIXED,
 46        (S3,S4) BIT(8),S5 FLOAT/);
 50    DCL SEM SEMA;
 55    DCL (FL1,FL2) FLOAT
 56         INIT (3.14159,14.5678);
 60    DCL  (FIX1,FIX2) FIXED INIT(20,50);
 65    DCL B16 BIT(16) INIT('AAAA'B4);
 70    DCL B8 BIT(8)   INIT('FF'B4);
 75    DCL (TM1,TM2) CLOCK;
 80    DCL DU DUR      INIT(20 MIN);
 85    DCL FELD(5) FIXED
 86           INIT((0,0,0,0,0));
 90    DCL STRU EL;
 95    DCL (ZLRT1,ZLRT2) FIXED INIT(0);
100
105    P:PROC;
110      DCL (A,B) FIXED INIT(0);
115      A:=A+1;
120      B:=A;
125    END  /* P */;
130
135    T1:TASK;
140      ZLRT1:=ZLRT1+1;
145      TM1:=NOW;
150      PUT 'START TASK T1:',TM1 TO FSR
151         BY A,D(25,3),SKIP;
155      FELD:=(1,2,3,4,5);
160      STRU:=
161         (/10,20,'08'B4,'80'B4,1.123/);
165      CALL P;
170      WHEN INT1 AFTER 2 SEC
171                ACTIVATE T1;
175      WHEN INT2 ACTIVATE T2;
180      AFTER 1 SEC CONTINUE;
185      SUSPEND;
190      TRIGGER INT2;
195      REQUEST SEM;
200    END  /*  T1  */;
205    T2:TASK;
210      ZLRT2:=ZLRT2+1;
215      TM2:=NOW;
220      PUT 'START TASK T2:',TM2 TO FSR
221         BY A,D(25,3),SKIP;
225      CALL P;
230      RELEASE SEM;
235      TRIGGER INT1;
240    END  /*  T2  */;
245  MODEND;
```

Das Quellprogramm liege in der Datei PRLPQU-PRLOBJ. Der PEARL-Compiler wird nun mit dem Kommando

```
$PC,DI:PRLPQU,MN:PRLP,TS:J;
```

gestartet. Die Parameter geben die Datei-Identifikation, und den ModulNamen an. Ausserdem veranlassen sie den Compiler Information für das Testsystem abzulegen. Er erzeugt die Bindemoduln

PRLP01,PRLP02,PRLP03,PRLP51,PRLP52,PRLP53

und den Informationsmodul PRLP00, der u.a. die Testinformation enthält.

Die Moduln seien geladen, und die beiden Anwender-Tasks, sowie die Modul-Task seien katalogisiert.

In die Datei CIFILE-PRLOBJ wird nun folgender Testplan eingetragen:

```
  5  BP 150
 10  IF ZLRT1 > 1
 15  GO 155
 20  BP 220
 25  GO 225
 30  AF 2 BP 185
 35  DP ZLRT1
 40  ST T1
 45  SH T1
 55  DP TM1
 60  DE FIX1,ZLRT1
 65  DP 'T1 -> SUS'
 70  DE FIX1,1.0
 75  IF ZLRT1 == 3
 80  WI
 85  AF 2 BP230
```

```
  90    DP ZLRT2
  95    ST T1
 100    DP FIX2
 105    DP 'REL SEM'
 110    IF ZLRT1 >= 3
 115    GO
 120    BL AL
 125    AC T1
 130    Z
```

Die Erläuterung obiger Testkommandos erfolgt nun in Form von Kommentaren in dem bei der Ausführung erzeugten Testprotokoll. In dem verwendeten System war als kleinste Zeiteinheit 250 msec eingestellt.

```
 $PQ; /* Benutzerstart des QPTS */
 START ATM 80 PEARL-QETS(A05.00)
  /*Anfangs-Meldung des QPTS */
 : CI CIFILE  /* Eingabe der Kommandos
                 aus der Datei CIFILE
 PAR-TYP UNZULAESSIG (SNR=70)
 /* Einbau von Breakpoints BP in den Zeilen
150, 220, 185, 230 mit den zugehörigen
dynamischen Testplänen. Das kommando "DE
FIX1   1.0" ist fehlerhaft, da versucht
werden soll, einer FIXED-Grösse ein
FLOAT-Wert zuzuweisen. Anschliessende
Ausführung des Kommandos BL AL (Auflisten
aller BPs).
 BP 150         PRLP ON
   IF ZLRT1 > 1
   GO 155
 BP 220         PRLP ON
   GO 225        /*Fortstart in Zeile 225 */
 AF 2 BP 185    PRLP ON
   DP ZLRT1      /* DumP ZLRT1 */
   ST T1         /* STatus Task T1 */
   SH T1         /* ScHedules Task T1 */
   DP TM
   DE FIX1,ZLRT1 /* Dump-Exchange */
   DP 'T1->SUS'
   IF ZLRT1 == 3
   WT            /*WaiT, Warten auf Dialog*/
 AF 2 BP 230    PRLP ON
   DP ZLRT2
   ST T1
   DP FIX2
   DP 'REL SEM'
   IF ZLRT1 >= 3
 GD              /* Go Delete BP */
   AC  T1         /*  Quittung des statisch
ausgeführten Funktionskommandos ACTIVATE T1
(Start einer Task aus dem zu testenden
Modul) */
 ENDE  PEARL-QETS  /* Ende-Meldung nach dem
Kommando Z.  */
/* Die Aktivierung der Task T1 führt nun
zum Auflaufen des Testlings auf die
definierten Breakpoints und damit in den
dynamischen Testmodus. Das Testkommando IF
wird nicht quittiert.  Damit ergibt sich
das folgende Testprotokoll:  */ Dazwischen
findet sich noch der Startausschrieb der
Task T1 */
 BP  150   PRLP ON  12:16:13.500
  START TASK T1: 12 HRS 16 MIN 13.500 SEC

 BP  220   PRLP ON  12:16:16.750
  GO 225
 BP  150   PRLP ON  12:16:18.750
  GO 155
/* Es folgt nun der Auflauf auf die BPs in
den Zeilen 185, 220, 230, 150, 185 und die
Ausführung der vordefinierten
Funktionskommandos. */
  AF 2 BP 185  PRLP ON 12:16:18.750
   DP ZLRT1     2 /* Wert von ZLRT1 =2 */
   ST T1      IM BP
   SH T1      AC,CO
   DP TM1     12:16:18.750
   DE FIX1,ZLRT1     20
   DP 'T1->SUS'
  BP 220        PRLP ON 12:16:19.750
   GO 225
  AF 2 BP 230   PRLP ON 12:16:19.750
   DP ZLRT2      2
   ST T1      WARTET AUF SEMA
   DP FIX2      50
   DP 'REL SEM'
  BP 150        PRLP ON 12:16:21.750
   GO 155
  AF 2 BP 185   PRLP ON 12:16:21.750
   DP ZLRT1      3
   ST T1      IM BP
   SH T1      AC,CO
   DP TM1     12:16:18.750
   DE FIX1,ZLRT1    2
   DP 'T1->SUS'
/*  Da nun die Bedingung ZLR == 3 erfüllt
ist, wird das Kommando WT ausgeführt und
das Testsystem geht in den Dialogmodus.
Nun werden Funktionskommandos eingegeben,
die sofort ausgeführt werden: */
 :DI INT1 /* DIsable INT1 */
  DI INT1  /* Quittung */
 :GD       /* Go Delete BP Fortstart */
/*  Nachdem nun bei diesem Fortstart der BP
gelöscht wurde, erfolgt nur noch der
Auflauf auf die BPs in den Zeilen 220 und
230. Durch die Anweisung DI INT1 wird ein
erneuter Start verhindert.  */
 BP 220         PRLP ON  12:16:22.500
  GO 225
 AF 2 BP 230    PRLP ON  12:16:22.750
  DP ZLRT2     3
  ST T1        WARTET AUF SEMA
  DP FIX2     50
  DP 'REL SEM'
  GD
$PQ; /* Benutzerstart von QPTS */
START ATM 80 PEARL-QETS (A5.00)
 :BL /* BPs aufListen */
 BP  150  PRLP ON
 BP  220  PRLP ON
 :NM  /* NorMiere Testsystem */
 :EN INT1  /* ENable INT1  */
   EN INT1
 :Z
ENDE PEARL QETS
```

6. Literatur

<1> - : Der FORTRAN-Dump-Operator, in TR4 FORTRAN, TELEFUNKEN,1968, Konstanz

<2> Krainer,H.: Testmöglichkeiten im Teilnehmer-Rechensystem TR 440, Beiträge 10, TELEFUNKEN COMPUTER, 1972, Konstanz

<3> - : PEARL-Testsystem für einen Grossrechner, Pflichtenheft, 31.10.76, mbp, Dortmund

<4> Gmeiner,L.; Hommel,G.(Hrsg.): Testen und Verifizieren von Prozessrechner-Software,KfK-PDV 179, Dez 1979, KfK, Karlsruhe

<5> - : Requirements for Ada Programming Support Environments, "STONEMAN", Febr. 1980, DoD, Washington D.C.

<6> - : PC30/PEARL 300, Kap.18, 1979 SIEMENS A.G.,München

Verfasser: Karlotto Mangold
AEG-TELEFUNKEN
A16/E-KN
Postfach 2154
775 Konstanz