Phasenübergreifende Entwicklung von Realzeitsystemen mit der Software-Entwicklungsumgebung proMod / Darts

J.Schmitz , GEI Aachen

Abstract:

Werkzeuge zur Unterstützung der Softwareentwicklung haben in der Vergangenheit eine weite Verbreitung gefunden. Viele Werkzeuge sind nur auf spezielle Phasen des Life Cycles zugeschnitten. Mit proMod/DARTS wird eine Softwareentwicklungsumgebung für die Entwicklung von Echtzeitsystemen vorgestellt, die den gesamten Life Cycle unterstützt.

Tools supporting software development came to be an accepted part of software production in the last years. Many of them provide support for a special phase of the life cycle only. The software engineering environment proMod/DARTS is introduced dealing with support for the development of real time systems over the whole life cycle.

Keywords:

Structured Analysis, Structured Design, abstrakte Datentypen, Information Hiding, Pseudocode, Parallele Prozesse, Synchronisation, Projektbibliothek, Interaktivität, Transformation, Durchgängigkeit, Softwareentwicklungsumgebung

Structured Analysis, Structured Design, Abstract Data Types, Information Hiding, Pseudocode, Tasks, Synchronization, Project Data Base, Interactiv, Transformation, Continual, Software Engineering Environment

Die industrielle Softwareproduktion erlebt in den letzten Jahren einen drastischen Wandel. Durch die seit einigen Jahren intensiv betriebene Forschung auf dem Gebiet "Software Engineering" entstand eine Vielzahl von Methoden für die verschiedenen Phasen (3) der Softwareentwicklung und auch Werkzeuge zur Automatisierung der Methoden (1), (4), (5), (7), (8). Meist sind diese Werkzeuge und Methoden Insellösungen im Software Life Cycle, da sie auf eine bestimmte Phase im Life Cycle zugeschnitten sind. Es genügt nicht, mehrere für verschiedene Phasen konzipierte Werkzeuge aneinanderzureihen, um eine effiziente phasenübergreifende Unterstützung zu schaffen. Die Kontinuität der Entwicklung wird beeinträchtigt, wenn beim Übergang zwischen den Methoden ein Umdenken erforderlich ist; Arbeitsergebnisse können verlorengehen, wenn die Werkzeuge nicht auf den Ergebnissen der vorangegangenen Phasen aufbauen können, sondern ein Neuaufsetzen erforderlich machen.

proMod ist eine Softwareentwicklungsumgebung, die für den gesamten Life Cycle rechnergestützte, aufeinander abgestimmte Methoden umfaßt. Die Methoden und zugeordneten Werkzeuge sind so verträglich miteinander, daß die in einer Phase erzielten Ergebnisse durch Transofrmation ihrer Struktur in das Denkschema der folgenden Phase umgesetzt werden und als Ausgangspunkt der weiteren Bearbeitung dienen können.

Für die Systemanalyse stellt proMod die Methode Structured Analysis und Werkzeuge zur Verfügung, die die Grundbausteine von SA, nämlich Datenflußdiagramme (DFD), Transformationsbeschreibungen (Minispecs) und Datenbeschreibungen (in einem Datenlexikon) interaktiv

- erfassen (DFD´s graphisch)
- gestalten,
- syntaktisch und semantisch prüfen,
- verwalten,

dokumentieren.

Ziel von Structured Analysis ist die Bildung eines Systemmodells aus hierarchisch geordneten Datenflußdiagrammen (2). Requirements an das Realzeitverhalten des Systems werden in den Minispecs spezifiziert oder sind implizit in der Netzstruktur der Datenflußdiagramme enthalten.

Der Übergang zum Grobentwurf wird von proMod durch eine Transformation der Datenflußnetze in eine Modulhierarchie, der Übernahme der Datenbeschreibungen und der Projektion der Minispecs als Designvorgaben in die Funktionsrümpfe der erzeugten Moduln vollzogen. Das Transformationsergebnis kann durch entsprechende Werkzeuge nach den Konzepten von Structured Design, abstrakten Datentypen und Information Hiding weiter verfeinert werden (6). Dafür stehen Werkzeuge mit einem zu den SA-Werkzeugen analogen Leistungsumfang zur Verfügung. Die Transformation generiert in den erzeugten Funktionsrümpfen Verweise auf alle von ihr "benutzten" anderen Funktionen. Ziel dieser Phase ist die Gestaltung einer Hierarchie von Moduln mit minimalen Schnittstellen untereinander und möglichst wenig Voraussetzungen für ihre gegenseitige Benutzung. So sollten Synchronisationsvorgänge soweit als möglich nur innerhalb von Moduln und nicht zwischen Moduln stattfinden.

Der Übergang zum Feinentwurf erfolgt durch Transformation der Modulhierarchie einschließlich aller Schnittstellenspezifikationen in DARTS Pseudocode.

Es folgt die Spezifikation der Algorithmik der Funktionen und des Realzeitverhaltens. DARTS stellt Sprachmittel für die Beschreibung paralleler Prozesse und ihrer Synchronisation zur Verfügung. Die Requirements für die Spezifikation der Funktionen sind aus den vorangegangenen Phasen übernommen und tauchen als Vorgaben in den Funktionsrümpfen auf.

Für die Implementierung hat sich die Verzahnung von Code und Pseudocode in einer Quelle bewährt; diese verhindert das Auseinanderlaufen von Code und Dokumentation. Verzahnung und Trennung von Code und Design nehmen zwei entsprechende Werkzeuge vor.

Die Verzahnung von Code und Design wird umso strikter sein, je bessere Implementationsmöglichkeiten für die Strukturen der Designsprache durchdie Zielsprache zur Verfügung gestellt werden. Wegen der Darstellungsmöglichkeiten für das Echtzeitverhalten eines Systems in DARTS ist PEARL für eine Verzahnung mit DARTS besonders geeignet. Für die Implementation eines DARTS-Designs durch eine weniger hochstehende Zielsprache ist vor der Codierung eine weitere Verfeinerung des DARTS-Designs angezeigt, durch die die nicht in der Zielsprache unterstützten Konstruktionen modelliert werden.

Mittelpunkt und Voraussetzung für die Durchgängigkeit von proMod ist die Projektbibliothek, eine zentrale Datenbank, die - für den Benutzer unsichtbar - die Verwaltung aller Objekte und Teildokumente, die in den verschiedenen Phasen anfallen, übernimmt (B i l d 1). Durch die Projektbibliothek werden die Beziehungen zwischen den einzelnen Dokumenten aufrechterhalten und organisiert.
Äußerliches Merkmal der Durchgängigkeit von proMod ist die Benutzerschnittstelle. Der Benutzer arbeitet mit dem proMod-System interaktiv an einem Terminal. Er gibt Kommandos an das System ab und erhält als Antwort Informationen aus dem System (oder genauer: aus der Projektbibliothek).

Alle Kommandos sind als Kombination von Kommandonamen und Objektbezeichner nach dem gleichen Schema aufgebaut. Die Auswahl der einzelnen für die Bearbeitung einer Aufgabe benötigten Werkzeuge bleibt dem Benutzer verborgen, er sieht das System allein durch dessen logischen Grundfunktionen wie Erfassen, Anzeigen, prüfen, Dokumentieren.

B i l d 1 vermittelt einen Überblick über die physikalischen Werkzeuge von proMod. Den methodenspezifischen Grundbausteinen wie Datenflußdiagramme, Daten, Funktionen, Moduln sind jeweils Werkzeuge zugeordnet, die das Erfassen, Prüfen, Formatieren und Verwalten dieser Bausteine unterstützen. Die als Analysatoren bezeichneten Werkzeuge sowie der DARTS-Prozessor sorgen für eine übergreifende Prüfung der Spezifikation in den entsprechenden Phasen und melden Verstöße gegen methodische Standards, wie z.B. mangelnde Vollständigkeit, Widersprüche bei hierarchischen Zerlegungen, unerlaubte Benutzung von Funktionen und Daten. Darüber hinaus generieren diese Werkzeuge eine Dokumentation der gesamten Spezifikation oder von Teilen.

Die einzelnen Schritte bei der Arbeit mit proMod werden im folgenden in Ausschnitten an einem Beispiel demonstriert. Das Beispiel behandelt im Meßwerterfassungssystem, bei dem das interruptgesteuerte Einlesen der Meßwerte und ihre Verarbei-

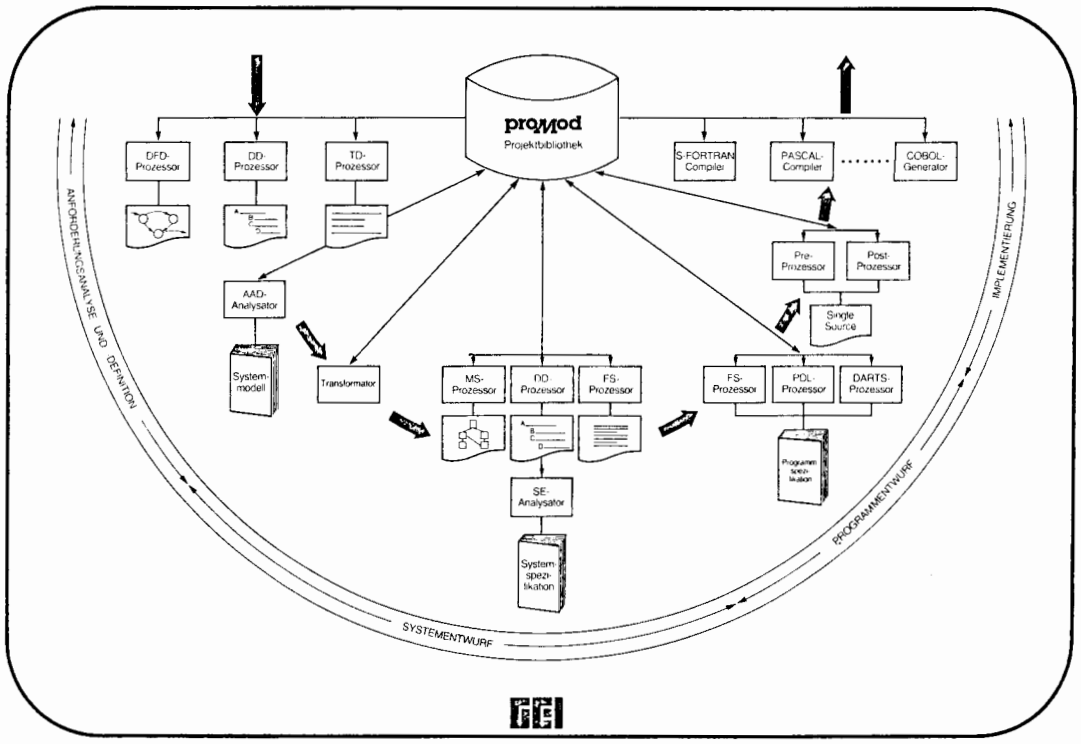
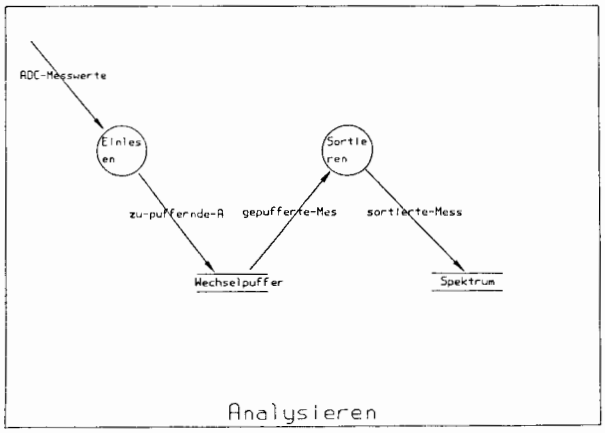


Bild 1. Die Werkzeuge von proMod

IN : ADC_Messwerte
OUT : sortierte_Messwerte



1.1 Einlesen

Die Task Einlesen liest Daten von einem ADC und schreibt sie in einen Wechselpuffer . Die Task kann nur in jeweils einen der beiden Puffer schreiben, wenn dieser leer ist. Sie erklaert einen Puffer fuer voll, nachdem sie ihn beschrieben hat. Starten und Stoppen der Task erfolgt ueber die (z. B. von einer Bedientask generierten) Signale Analysestart und Analysestop bzw. Totalstop

1.2 Sortieren

Die Sortiertask entnimmt abwechselnd Daten aus den beiden Puffern eines Wechselpuffers, fuehrt Berechnungen mit diesen durch und legt sie im Spektrum ab. Die Task kann nur aus den beiden Puffern Puffer1 oder Puffer2 des Wechselpuffers lesen. Sie erklaert sie fuer leer, nachdem sie den jeweiligen Inhalt ausgewertet hat. Starten der Task erfolgt durch das (z. B. von einer Bedientask erzeugte) Signal Analysestart , Stoppen durch Totalstop . Das Signal Analysestop bewirkt einen geordneten Abbruch des Sortiervorgangs, d. h., alle bis dahin in den Wechselpuffer geschriebenen Messwerte werden noch analysiert, auch wenn der jeweilige Puffer noch nicht fuer voll erklaert worden ist. Der Analysestop kann durch den Bediener oder den Ausfall des ADC generiert werden.

Bild 2. Funktionale Vorgaben

tung parallel ablaufen; die Kopplung von Einlesen und Verarbeitung soll über einen Wechselpuffer erfolgen.

B i l d 2 enthält den entsprechenden Ausschnitt des einschließlich Graphik mit proMod erstellten Pflichtenheftes.

Im nächsten Schritt werden die graphischen Datenflußdiagramme durch den Transformator von proMod in eine Modulhierarchie umgewandelt. Getreu dem Prinzip der abstrakten Datentypen ist der Wechselpuffer zu einem eigenen Modul mit Zugriffsfunktionen für Lesen und Schreiben geworden (B i l d 3).

Die funktionalen Vorgaben aus dem Pflichtenheft sind automatisch als Programmiervorgaben in die Funktionsrumpfe der generierten Funktionen eingesetzt und gemäß den Vorgaben der Datenflußdiagramme Funktionsaufrufe generiert worden (B i l d 4). Die Modulstrukturen sind anschließend zu verfeinern (B i l d 5) und durch proMod zu prüfen. Die auf diese Weise entstandene Modulhierarchie ist noch offen für verschiedene Formen einer zeitlichen Steuerung wie Sequenzialisierung, Parallelität, Quasiparallelität.

Nach dem Prinzip des Information Hiding sollten Synchronisationsvorgänge möglichst innerhalb von Moduln versteckt sein (z.B. bei Modul Wechselpuffer).

```
Copyright(c) 1983, GEI mbH, Albert-Einstein-Str.61, 5100 Aachen
**************************************************************
*  proMod V1.1      26-07-83     16:24            -PAGE  12 -   *
*                                                              *
*  GEI              PROJECT :    DPP          Design listing    *
**************************************************************

MODULE Wechselpuffer

EXP
     Read_Wechselpuffer, ----------------------------------->  13

     Write_Wechselpuffer ----------------------------------->  14
```

Bild 3. Ergebnis der Modultransformation des Speichers "Wechselpuffer"

```
Copyright(c) 1983, GEI mbH, Albert-Einstein-Str.61, 5100 Aachen
**************************************************************
*  proMod V1.1      26-07-83     16:24            -PAGE   3 -   *
*                                                              *
*  GEI              PROJECT :    DPP          Design listing    *
**************************************************************

FUNCTION Einlesen

PURPOSE
 Die Task Einlesen liest Daten von einem ADC und schreibt sie
 in einen Wechselpuffer. Die Task kann nur in jeweils einen
 der beiden Puffer schreiben, wenn dieser leer ist. Sie erklaert
 einen Puffer fuer voll, nachdem sie ihn beschrieben hat.
 Starten und Stoppen der Task erfolgt ueber die (z. B. von einer
 Bedientask generierten) Signale Analysestart und Analysestop
 bzw. Totalstop.
ENDPURPOSE

 1   Write_Wechselpuffer ----------------------------------->  14
```

Bei Transformation generierte Programmvorgaben und Funktionsaufrufe

```
Copyright(c) 1983, GEI mbH, Albert-Einstein-Str.61, 5100 Aachen
**************************************************************
*  proMod V1.1      28-07-83     10:08            -PAGE  14 -   *
*                                                              *
*  GEI              PROJECT :    DPP          Design listing    *
**************************************************************

MODULE Wechselpuffer

PURPOSE
 Das Modul realisiert den abstrakten Datentyp Wechselpuffer.
 Nach aussen praesentiert sich der Wechselpuffer wie ein FIFO -
 Puffer, in den sukzessive einzelne Werte geschrieben werden,
 die in gleicher Reihenfolge - allerdings als Liste - ausgelesen
 werden. Intern besteht der Wechselpuffer aus zwei Puffern, die
 alternierend beschrieben werden. Ist ein Puffer voll, so wird
 ein entsprechendes Signal Puffer1_voll bzw. Puffer2_voll als
 Freigabesignal zum Lesen generiert und das Einschreiben im
 jeweils anderen Puffer fortgesetzt, sofern dieser durch die
 Signale Puffer1_leer bzw. Puffer2_leer als ' gelesen '
 freigegeben worden ist. Diese Signale werden vom lesenden
 Zugriff erzeugt. Durch diese Synchronisation kann parallel
 gelesen und geschrieben werden.
ENDPURPOSE

EXP
     Initialisiere_Wechselpuffer, ------------------------->  15

     Read_Rest_Wechselpuffer ------------------------------->  16
          (OUT Werteliste),

     Read_Wechselpuffer ------------------------------------>  17
          (OUT Werteliste),

     Write_Wechselpuffer ----------------------------------->  18
          (IN zu_puffernder_Wert),

DATA
     Puffer1_leer, Puffer1_voll, Puffer2_leer, Puffer2_voll,
     Wechselpuffer, Werteliste, zu_puffernder_Wert
```

Bild 5. Verfeinertes Modul "Wechselpuffer"

Die Gestaltung des Echtzeitverhaltens erfolgt im nächsten Schritt nach automatischer Transformation der Modulhierarchie in DARTS-Pseudocode.

Diejenigen Funktionen, die parallel ablaufen sollen, werden als "ACTIVITY" deklariert und ihre Aufrufe durch Taskaktivierungen bzw. Aborts im Pseudocode ersetzt (B i l d 6, 7, 8). Aus den Schlüsselworten zur Beschreibung des Echtzeitverhaltens leitet DARTS spezielle Informationen über das Zusammenspiel von Tasks und Synchronisationsmitteln ab (B i l d 9).

Das Konzept der Durchgängigkeit hat sich im bisherigen praktischen Einsatz von proMod/DARTS bewährt. Besonders auffällig war die gute Akzeptanz die auf leichte Erlernbarkeit und schnelle Beherrschung von proMod/DARTS zurückzuführen ist. Die bisherigen Erfahrungen vermitteln die Sicherheit, daß auch die geplanten Erweiterungen von proMod um

- Requirementverfolgung,
- Coderahmengeneratoren,
- Versionshaltung und Configuration Management,
- Projektmanagement

homogen unter Wahrung des Prinzips "Durchgängigkeit" in proMod einzubetten sind.

Literaturverzeichnis:

(1) C a i n e, S.H.; G o r d o n, E.W.: A Tool for Software Design. In: AFIPS, Proc. NCC, Vol. 44, 1975.

(2) D e M a r c o, T: Structured Analysis and System Specification. Yourdon Press, 1979.

(3) K e r o l a, P.; F r e e m a n, P.: A Comparison of Life Cycle Models. In: IEEE Proc. 5th ICSE, p. 90-99, 1981.

(4) K e u t g e n, H.: DARTS - Design Aid for Real Time Systems. KfK-PFT 17, Januar 1982, Kernforschungszentrum Karlsruhe.

(5) L a u b e r, R.; et al.: EPOS - A Specification and Design Technique For Computer Controlled Real-Time Automation Systems. In: IEEE Proc. 4th ICSE, p. 245-250, 1980.

(6) P a r n a s, D.L.: Information Distribution Aspects of Design Methodology. Proc. IFIP Congress 71, North Holland, 1971.

(7) T e i c h r o e w, D.; H e r s h e y, E.A. III: PSL/PSA: A Computer-Aided Technique For Structured Documentation and Analysis of Information Processing Systems. IEEE Trans. on SE, Vol. 3, No. 1, Jan. 1977.

(8) W i l l i s, R.R.: Computer Aided Design of Software Systems. In: Proc. 4th ICSE, p. 116-125, 1980.

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A                                                                                 A  29. 7.83 A
A            proMod                                                               A   13:32   A
A   G E I    4 Modul Doppelpufferproblem                                          A SEITE  18 A
A             4. 4 Analysieren                                                    A           A
A                                                                                 A     BEZUG A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

       CALL Initialisiere_Wechselpuffer----------------------------------------------->( 37)
       CALL Init_Spektrum------------------------------------------------------------->( 27)
       ACTIVATE Einlesen=============================================================>( 10)
       ACTIVATE Sortieren============================================================>( 12)

   ? WAITFOR Einlesen_Ende AND Sortieren_Ende

       Hier koennen ggf. Anschlussarbeiten durchgefuehrt werden.



       WHENEVER Totalstop
   + :  ABORT Analysieren                                                           ( 18)
   + :  ABORT Einlesen                                                              ( 10)
   + :  ABORT Sortieren                                                             ( 12)
       ENDWHEN
```

Bild 6. Pseudocode für Aktivierung der Task "Einlesen"

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A                                                                                 A  29. 7.83 A
A            proMod                                                               A   13:32   A
A   G E I    3 Modul Analysieren                                                  A SEITE  10 A
A             3. 4 Einlesen                                                       A           A
A                                                                                 A     BEZUG A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

       DO zyklisch
       :  ADC in zu_puffernder_ADC_Messwert einlesen
       :  CALL Write_Wechselpuffer---------------------------------------------------->( 43)
       :  IN zu_puffernder_ADC_Messwert
       ENDDO



       WHENEVER Analysestop
   + :  ABORT Einlesen                                                              ( 10)
     :  ADC abschalten
   ! :  SIGNAL Einlesen_Ende
       ENDWHEN
```

Bild 7. Pseudocode der Task "Einlesen"

```
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
M                                                                                 M  29. 7.83 M
M            proMod                                                               M   13:32   M
M   G E I    6 Modul Wechselpuffer                                                M SEITE  43 M
M            6. 11 Write_Wechselpuffer                                            M           M
M                                                                                 M     BEZUG M
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

      DATA
      IN zu_puffernder_Wert
      ENDDATA

      CASE Write_Selektor zeigt auf
   Puffer1:
     :  IF Wechselpuffer.aktuelle_Schreibposition zeigt auf ersten Platz
   ? :  :  WAITFOR Puffer1_leer
     :  ENDIF
     :  schreibe zu_puffernder_Wert in Puffer1 auf den durch Wechselpuffer.aktuelle_Schreibposition
     :       definierten Platz
     :  Inkrementiere Wechselpuffer.aktuelle_Schreibposition
     :  IF Pufferende erreicht
   ! :  :  SIGNAL Puffer1_voll
     :  :  Setze Write_Selektor auf Puffer2
     :  :  Setze Wechselpuffer.aktuelle_Schreibposition auf Anfang von Puffer2
     :  ENDIF
   Puffer2:
     :  IF Wechselpuffer.aktuelle_Schreibposition zeigt auf ersten Platz
   ? :  :  WAITFOR Puffer2_leer
     :  ENDIF
     :  schreibe zu_puffernder_Wert in Puffer1 auf den durch Wechselpuffer.aktuelle_Schreibposition
     :       definierten Platz
     :  Inkrementiere Wechselpuffer.aktuelle_Schreibposition
     :  IF Pufferende erreicht
   ! :  :  SIGNAL Puffer2_voll
     :  :  Setze Write_Selektor auf Puffer1
     :  :  Setze Wechselpuffer aktuelle_Schreibposition auf Anfang von Puffer1
     :  ENDIF
      ENDCASE
      Return
```

Bild 8. Zugriff auf Wechselpuffer mit Synchronisation

```
SFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSF
S                                                                                                     F  29. 7.83 F
S           proMod                                                                                    F   13:32   F
S   G E I   7 Strukturreport                                                                          F SEITE  56 F
S           7.11 Beziehungen zwischen den Aktivitaeten                                                F         F
S                                                                                                     F         F
SFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSF
```

```
 1   Liste der Aktivitaeten
 2
 3   Seite   Name
 4
 5      10   Einlesen
 6      12   Sortieren
 7      18   Analysieren
 8      20   Darstellen
 9      22   Doppelpufferproblem
10
11   ZNR |  5|  6|  7|  8|  9|
12   ----------------------
13     5 | +|   |   |   |   |
14   ----------------------
15     6 |  |   |   |   |   |
16   ----------------------
17     7 |> +|> +|  +|   |   |
18   ----------------------
19     8 |  |   |   |   |   |
20   ----------------------
21     9 |  |   |>  |>  |  +|
22   ----------------------
```

```
SFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSF
S
S           proMod
S   G E I   7 Strukturreport
S           7. 2 Aufrufgraph mit der Wurzel: Einlesen
S
SFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSFSF
```

```
 1    Einlesen
 2   |   Write_Wechselpuffer
 3
 4   folgende Synchronizer und Interrupts werden benutzt:
 5
 6    Typ    Seite  Zeile   Name
 7
 8   IRPT        5      3   Analysestop
 9   SYNCH       5      2   Einlesen_Ende
10   SYNCH       5      1   Puffer1_leer
11   SYNCH       5      1   Puffer1_voll
12   SYNCH       5      1   Puffer2_leer
13   SYNCH       5      1   Puffer2_voll
```

Bild 9. Analyse des Echtzeitverhaltens

Schmitz, Jürgen
c/o GEI-GESELLSCHAFT FÜR ELEKTRONISCHE
INFORMATIONSVERARBEITUNG mbH
Albert-Einstein-Str. 61

5100 Aachen

Tel. 02408/130