KfK 2714 B
Januar 1979

Vergleich verschiedener Sprachen für die Prozeßprogrammierung anhand der Mehrfachimplementierung eines Überwachungssystems

M. Swieczkowski
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 2714 B

Vergleich verschiedener Sprachen für die Prozeßprogrammierung anhand der Mehrfachimplementierung eines Überwachungssystems

von

Michael Swieczkowski

Diplomarbeit eingereicht bei der Fakultät für Informatik der Universität Karlsruhe

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Zusammenfassung

Für VARIAN-Prozeßrechner der V7Ø-Serie wird versucht, die verfügbaren Sprachen Assembler DASMR, BASIC, FORTRAN, IFTRAN und PASCAL hinsichtlich ihrer Eignung für die Prozeßprogrammierung zu untersuchen.
Es wurden Kriterien erstellt, mit denen die Sprachen auf ihre syntaktischen Eigenschaften, Real-Zeit-Möglichkeiten und ihre Anwendbarkeit objektiv verglichen werden können.
Für den praktischen Teil des Vergleichs wird ein Überwachungssystem für einen technischen Prozeß in den verschiedenen Sprachen implementiert.
Das Grundkonzept ist sprachunabhängig mit Hilfe von SADT erstellt.
Basierend auf einem subjektiven Bewertungsmaßstab, mit dem die erstellten Kriterien gemäß ihrer Wichtigkeit für die Prozeßprogrammierung eingestuft werden, ergibt sich, daß PASCAL am besten für Lösungen von Aufgaben aus der Prozeßprogrammierung geeignet ist. Es folgen IFTRAN, FORTRAN, DASMR und BASIC in absteigender Reihenfolge.

Comparison of Different Languages for Real-Time Programming by Multiple Implementation of a Monitor System

Abstract

The aim of this paper is to examine the languages assembler DASMR, BASIC, FORTRAN, IFTRAN and PASCAL regarding to their suitability for process programming.
The languages are implemented on a VARIAN computer of the V7Ø-series and are running with the operating system VORTEX II F.
For this examination criteria were formulated which allow to compare the syntactic characteristics, the real-time-features and the general suitability in an objective manner.
For the practical part of the comparison a control system for a technical process was implemented in the several languages.
The basic concept was done by SADT.
Basing on a subjective scale of evaluation, by which the criteria were classified according to their importance for process programming, the following results were obtained:
PASCAL fit best the requirements for process programming.
It is succeeded by IFTRAN, FORTRAN, DASMR and BASIC.

1. EINLEITUNG 1
1.1 Motivation und Zielsetzung 1
1.2 Randbedingungen 3
1.3 Anforderungen des Ingenieurs an eine Programmiersprache 7

2. KRITERIEN FÜR EINEN SPRACHVERGLEICH 8
2.1 Problematik eines Sprachvergleichs 8
2.2 Kriterien für die Sprachmittel 1Ø
2.2.1 Elementare Datentypen 1Ø
2.2.2 Datenstrukturen 1Ø
2.2.3 Kommunikationsbereiche 11
2.2.4 Vereinbarungen 11
2.2.5 Unterprogramme 12
2.2.6 Grundoperationen 12
2.2.7 Ablaufsteuerung 12
2.2.8 Ein-/Ausgabe 14
2.3 Spezielle Kriterien für die Prozeßprogrammierung 15
2.4 Allgemeine Beurteilungskriterien für die Sprachanwendung 16
2.4.1 Modularität 16
2.4.2 Erlernbarkeit 16
2.4.3 Effizienz 16
2.4.4 Implementierungsaufwand 17
2.4.5 Testhilfen 17
2.4.6 Zuverlässigkeit, Sicherheit 17
2.4.7 Dokumentation, Lesbarkeit 18
2.4.8 Wartung 19

3. DEFINITION DES PROZESSMODELLS 21
3.1 Der reale Prozeß 21
3.2 Das Modell der Differenzdichte 24
3.3 Systemanalyse mit SADT 26
3.4 Das dynamische Modell der Prozeßüberwachung 3Ø

4. IMPLEMENTIERUNG 32
4.1 Abdeckung der Vergleichskriterien 32
4.2 Programmphilosophie 33
4.3 Detailkonzept zur Realisierung der Prozeßüberwachung 35
4.4 Das Simulationssystem 38

5. VERGLEICH DER SPRACHMERKMALE 44
5.1 Datentypen 44
5.2 Datenstrukturen 44
5.3 Kommunikationsbereiche 44
5.4 Vereinbarungen 45
5.5 Unterprogramme 46
5.6 Grundoperationen 46
5.7 Ablaufsteuerung 46
5.8 Ein-/Ausgabe 48
5.9 Überprüfung der Sprachmittel für die Prozeßprogrammierung 48
5.1Ø Möglichkeiten des Assemblers DASMR 49

6. VERGLEICH DES AUFWANDS WÄHREND DER IMPLEMENTIERUNGS- UND TESTPHASE 52
6.1 Vorbemerkungen 52
6.2 Änderungen des Konzepts bei PASCAL 53
6.3 Aufwand während der Implementierungs- und Testphase 54
6.4 Allgemeine Vorarbeiten und Zusatzprogramme 6Ø

7. VERGLEICH DER SPRACHQUALITÄTEN 61
7.1 Modularität 61
7.2 Erlernbarkeit 61
7.3 Effizienz 62
7.4 Implementierungsaufwand 63
7.5 Testhilfen 63
7.6 Zuverlässigkeit, Sicherheit 64
7.7 Dokumentation, Lesbarkeit 65
7.8 Wartung 67

8. BEWERTUNG DER SPRACHEN 68
8.1 Der Bewertungsmaßstab 68
8.2 Begründung der Wichtungen 69
8.3 Ergebnis des Sprachvergleichs 71

9. KRITISCHE ANMERKUNGEN ZU DEM SPRACHVERGLEICH 73

Literaturliste 75

ANHANG:

A Entwurf der Prozeßüberwachung und Simulation in SADT 77
A.1 Prozeßüberwachung 78
A.2 Simulation 96

B Das dynamische Modell 1Ø2

C Die Bewertungstabellen 1Ø3

D Das statische Programmprofil 1Ø6

1. EINLEITUNG

1.1 Motivation und Zielsetzung

Für die Entwicklung und Implementierung von rechnergeführten Prozeßüberwachungssystemen existieren im Rechnerversuchsfeld des IDT VARIAN-Prozeßrechner der V7Ø-Serie. Auf der VARIAN sind Sprachen für einen universellen Einsatzbereich implementiert:

Assembler DASMR, BASIC, FORTRAN, IFTRAN und PASCAL.

"Eigentliche" Prozeßsprachen sind zur Zeit nicht verfügbar. Es soll untersucht werden, welche der vorhandenen Sprachen sich am besten für die Lösung von Aufgaben aus der Prozeßautomatisierung eignet. Eine Antwort darauf soll durch einen Vergleich der Sprachen gegeben werden.
Die detailliertere Formulierung des Sprachvergleichs führt zur folgenden Vorgehensweise:

- Zunächst müssen geeignete Kriterien aufgestellt werden, die eine objektive Untersuchung der Sprachen ermöglichen.
- Da sich der Vergleich nicht nur auf die theoretischen Aspekte der Sprachen beschränken soll, werden in einem praktischen Einsatz die Aussagen über die Sprachen überprüft. Dazu muß ein geeigneter technischer Prozeß ausgewählt werden und ein Konzept für die rechnergestützte Überwachung aufgestellt werden.
- Das Modell der Prozeßüberwachung ist in den verschiedenen Sprachen auf dem Rechner zu implementieren.
- Die so gewonnenen Erfahrungen werden anhand der aufgestellten Kriterien analysiert und bewertet.
- Für Testzwecke wird der reale Prozeß durch ein Simulationssystem auf dem Rechner nachgebildet. Dadurch ist es möglich, erweiterte Aussagen über die Sprachen zu erhalten.

Parallel dazu wird in einer weiteren Arbeit /1/ das Verhalten der Sprachen zur Laufzeit untersucht.

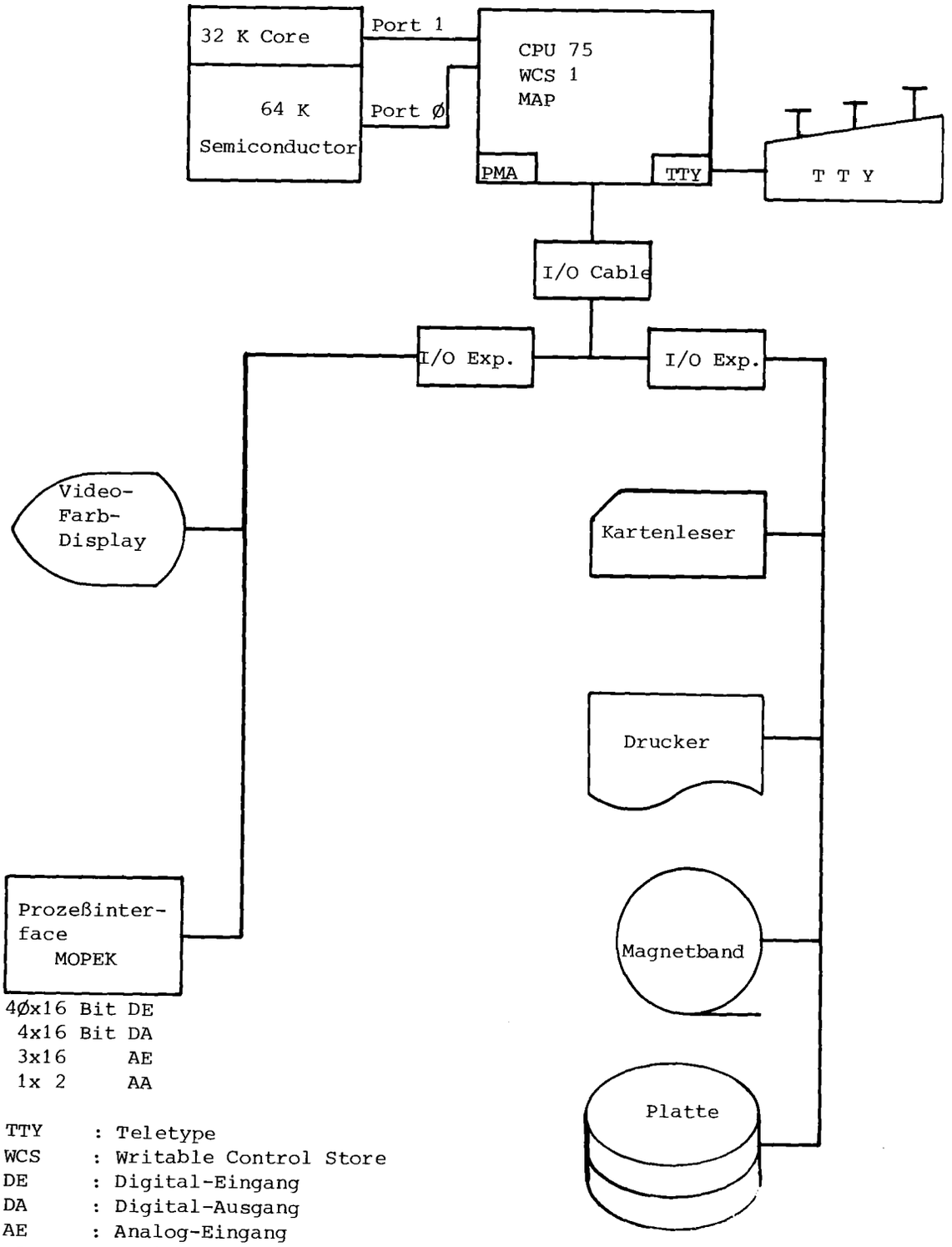
Der hier durchgeführte Vergleich erfolgte im Rahmen einer Diplomarbeit. Die damit verbundene zeitliche Beschränkung erlaubte es nicht, weitere interessante Aspekte zu verfolgen, wie zum Beispiel der Vergleich der verfügbaren Sprachen mit der Sprache PEARL, die speziell für Real-Zeitaufgaben konzipiert wurde.

1.2 Randbedingungen

Im IDT steht folgende Rechnerkonfiguration zur Verfügung:

Rechner: VARIAN-Rechner der V7Ø-Serie: V75 /3/
Die V75 hat eine Wortlänge von 16 Bit. Sie besitzt 16 Register. Zur Zeit stehen insgesamt 96 K Worte Speicher zur Verfügung. Der Speicher wird virtuell adressiert. Die V75 ist mikroprogrammierbar (Rechnerkonfiguration siehe Abb.1).

Prozeßinterface: MOPEK /7/, /8/
(Modulares Peripherie-Koppelelement)
Das Interface ermöglicht die direkte Kopplung des Rechners mit einer technischen Anlage und deren Überwachung und Lenkung im "On-line"-Betrieb. Der gerätetechnische und logische Aufbau von MOPEK ist vergleichbar mit dem des CAMAC-Systems.
MOPEK bietet in der vorliegenden Ausbaustufe folgende Anschlußmöglichkeiten:
4Øx16 Digitaleingänge, davon 8x16 mit Alarmgenerierung (die Änderung eines Eingangssignals erzeugt einen Interrupt) und 3x16 Analogeingänge. Ausgabe ist durch 4x16 Digitalausgänge und 1x2 Analogausgänge möglich. Hinzu kommt eine rechnerunabhängige, interne Uhr mit der ebenfalls Alarme generiert werden können (Intervallzeitgeber: IVZG).



4Øx16 Bit DE
4x16 Bit DA
3x16 AE
1x 2 AA

TTY : Teletype
WCS : Writable Control Store
DE : Digital-Eingang
DA : Digital-Ausgang
AE : Analog-Eingang
AA : Analog-Ausgang
I/O-Exp: E/A-Kabel-Ausweitung
MAP : Memory Allocation Processor
PMA : Priority Memory Access

Abb 1: Schema der Rechnerkonfiguration

Betriebssystem: VORTEX II F /2/

(VARIAN-Omni-Task-Real-Time-Executive)

VORTEX ist ein "Single-user"-System mit Eigenschaften für die Echtzeitprogrammierung. Aufträge an das Betriebssystem werden in zwei Klassen unterteilt: "Background"- und "Foreground"-Aufträge. Im Background werden alle Job-Control-Befehle, die Dateiverwaltung sowie alle zeitunkritischen Programme abgewickelt (Batchbetrieb), der Foreground ist für Echtzeitprogramme vorgesehen.

Sprachen:

- VARIAN-ASSEMBLER DASMR /3/, /4/

  Der ASSEMBLER der VARIAN besitzt Makroeigenschaften. Es stehen 8 Register für die Programmierung zur Verfügung. Die Adressierung von Speicherzellen kann auf 4 verschiedene Arten erfolgen: direkt, indirekt, relativ und indiziert.

  DASMR hat einen Befehlsumfang von 2Ø1 Befehlen. Das Befehlsformat besteht aus 1 oder 2 Worten.

- BASIC /5/

  BASIC ist auf der V75 durch einen Interpreter realisiert. Es entspricht in seinem Sprachumfang in etwa dem "Minimal BASIC" /1Ø/.

- FORTRAN /6/

  Der auf der V75 verfügbare 1-Pass-Compiler für FORTRAN IV stellt eine Erweiterung des ANSI-Standards für FORTRAN dar.

- IFTRAN /11/, /12/, /13/

  Zur Verfügung steht die Version IFTRAN-3. IFTRAN ist einer von vielen Vorübersetzern für FORTRAN und bietet zusätzlich zu FORTRAN Mittel für eine strukturierte Programmierung an.

Der IFTRAN-Preprozessor setzt das IFTRAN-Programm in einem ersten Übersetzungslauf in ein äquivalentes FORTRAN-Programm um, das in einem zweiten Lauf normal weiterbearbeitet wird.

- PASCAL /14/, /15/
  Auf der VARIAN existiert ein PASCAL-System, das sich aus drei Komponenten zusammensetzt:
  a. PASCAL-Compiler für Sequential PASCAL /16/
     7-Pass-Compiler, in PASCAL geschrieben
  b. PASCAL-Interpreter
     Virtuelle Maschine, in DASMR geschrieben
  c. PASCAL-I/O-Interface
     Schnittstelle zur Ein-/Ausgabe in VORTEX II, in DASMR geschrieben.

1.3 Anforderungen des Ingenieurs an eine Programmiersprache

Von einer Prozeßprogrammiersprache werden -aus dem Blickwinkel des anwendenden Ingenieurs- vor allem folgende Eigenschaften gefordert /9/:

- Ausreichender Sprachumfang, um beliebige Automatisierungsaufgaben und -Algorithmen formulieren zu können.
- Weitgehende Selbstdokumentation zur Gewährleistung einer einfachen Programmpflege.
- Erlernbarkeit und effiziente Anwendbarkeit durch Nicht-Programmierer (Anwendungsingenieure) bzw. "Selten-Programmierer".
- Hohe Zuverlässigkeit während des Compilierens, d.h. die Eigenschaft, daß ein großer Teil der möglichen Programmierfehler durch Compilerprüfungen entdeckt werden kann und daß die Wahrscheinlichkeit für noch verbleibende Fehler in den compilierten Anwenderprogrammen gering ist.
- Ausreichende Möglichkeiten für die Echtzeit-Programmierung.
- Hinreichende Effizienz der compilierten Automatisierungsprogramme bezüglich Rechenzeit, Speicherbedarf und Speicherzugriffe (hauptsächlich Plattenzugriffe).

Dem Anwender stehen bisher meist nur universell definierte Sprachen zur Verfügung: reine Prozeßsprachen sind noch zu wenig verbreitet und nur für einige spezielle Rechner implementiert. Damit der Anwendungsingenieur Entscheidungen hinsichtlich der Sprachauswahl treffen kann, sind die verfügbaren Universalsprachen nach den oben genannten Anforderungen zu bewerten.

2. KRITERIEN FÜR EINEN SPRACHVERGLEICH

2.1 Problematik eines Sprachvergleichs

Bei einem Vergleich von Programmiersprachen treten zunächst allgemeine Probleme auf:

- Welche Eigenschaften der zu untersuchenden Sprachen sollen miteinander verglichen werden?
- Wie kann man so unterschiedlich ausgelegte Sprachen wie zum Beispiel PASCAL und BASIC oder ASSEMBLER überhaupt miteinander vergleichen?
- Nach welchen Kriterien soll das Anwendungsbeispiel für den praktischen Teil des Vergleichs ausgesucht werden? Für jede Sprache kann eine Aufgabe gestellt werden, für deren Lösung sie optimale Ergebnisse liefert, genauso leicht ist es möglich, daß es überhaupt keine Realisierungsmöglichkeit gibt.
- Welche Mittel bietet eine Sprache, und welche Mittel können in einem vertretbaren Aufwand vom Anwender zusätzlich geschaffen werden?

Auch die Umgebung, die man beim Einsatz der Datenverarbeitung vorfindet, beeinflußt die Verwendung der Programmiersprachen. So existieren wesentliche Unterschiede zwischen der Prozeßprogrammierung und der allgemeinen kommerziellen DV.
Aufgrund dieser Problematik muß bei der Durchführung eines Sprachvergleichs eine Vorgehensweise gefunden werden, durch welche die aufgeworfenen Fragen in sachlicher Form beantwortet werden können.
Diese Grundlage wird durch die folgende Vorgehensweise erreicht:

- Es werden nur die Eigenschaften der Sprachen miteinander verglichen, die für die Prozeßprogrammierung wichtig sind.

- Für die praktische Erprobung beim Einsatz wird ein kleiner, realitätsnaher technischer Prozeß ausgewählt. Der Entwurf zur rechnergeführten Überwachung des Prozesses soll in einer sprachunabhängigen Form durchgeführt werden.
- Die Auswahl des Prozesses ist so vorzunehmen, daß möglichst viele typischen Eigenschaften der Prozeßprogrammierung durch ihn abgedeckt werden.

2.2 Kriterien für die Sprachmittel

Es folgt hier eine Aufzählung der Anforderungen an die allgemeinen Sprachmittel, die bei der Prozeßprogrammierung wichtig sind. Jedes über diesen Forderungskatalog hinaus existierende Sprachmittel bietet die Möglichkeit, bessere Lösungen für eine Aufgabe zu finden. Sie können jedoch nicht in der Aufzählung berücksichtigt werden, da sonst keine allgemeingültigen Aussagen mehr möglich sind (siehe 2.1). Eine Zusammenfassung aller untersuchten Sprachmittel und die darüber gewonnenen Erfahrungen ist in Anhang C gegeben.
In den nachfolgenden Abschnitten werden einige Begriffe verwendet, deren Bedeutung hier kurz erklärt ist /17/:
Ein Objekt ist eine Nachricht N zusammen mit der ihr zugeordneten Information J. Es gilt also:
Ein Objekt ist ein Paar (N,J) mit $N \overset{\alpha}{\longmapsto} J$, dabei wird die Information J Wert des Objekts, die Nachricht N Bezeichnung des Objekts genannt. Man sagt, die Bezeichnung N besitzt den Wert J.

2.2.1 Elementare Datentypen

Als einfache Datentypen sollten die Typen

INTEGER: positive und negative ganze Zahlen
REAL : positive und negative reelle Zahlen
CHAR : endliche Menge von "Zeichen"
BIT : Binärwerte

in einer Prozeßsprache vorhanden sein. Darüberhinaus sind spezielle Typen wünschenswert, die eine einfache Handhabung der Begriffe Zeitpunkt, Zeitdauer und Interrupt erlauben ("Prozeßtypen").

2.2.2 Datenstrukturen

Unter Datenstrukturen werden hier Felder und Verbunde verstanden. Nach Bauer, Goos /17/ sind diese wie folgt definiert:

Felder sind zusammengesetzte Objekte mit gleichartigen Komponenten, deren Indizes INTEGER-Werte sind. Mit Indizes kann numerisch gerechnet werden. Als Mindestanforderung für die Dimension genügt der Grad 3.

Verbunde (engl. RECORD) sind zusammengesetzte Objekte, deren Komponenten im Gegensatz zu Feldern verschiedene Typen besitzen. Sie bieten die Möglichkeit, verflochtene Datenstrukturen aufzubauen.

Verbunde sind für die Prozeßprogrammierung besonders interessant, weil mit ihnen Komplexe Strukturen zur Verwaltung der anfallenden Daten (Meßdatenblöcke, Eichwertsätze und ähnliches) aufgebaut werden können.
Im Gegensatz dazu werden Felder mehr bei numerischen Problemen benötigt.

2.2.3 Kommunikationsbereiche

Ein Hauptmerkmal der Prozeßprogrammierung ist das Zerlegen eines Problems in mehrere Programmeinheiten ("Multitasking" siehe 2.3). Für einen Informationsaustausch zwischen diesen Programmeinheiten sind gemeinsame Datenbereiche notwendig.
Daneben ist die Möglichkeit einer globalen Variablendeklaration wichtig (Variable werden an einer Stelle im Programm definiert und existieren in der gesamten Programmeinheit. Eine genauere Erläuterung der Begriffe erfolgt in 2.2.7.b: Blockstruktur).

2.2.4 Vereinbarungen

Neben einer üblichen expliziten Variablendeklaration wird auch eine entsprechende Deklaration für Konstanten gefordert.
Implizite Typdeklarationen sind möglichst auszuschließen, da dies eine Beeinträchtigung der Programmsicherheit (siehe 2.4.6) bedeuten könnte. Implizite Deklarationen sind höchstens bei Schleifenzählern und Indizes aus Bequemlichkeitsgründen sinnvoll.

2.2.5 Unterprogramme

Funktionen und Prozeduren sind unerläßliche Sprachhilfsmittel. Bei Prozeduren und Funktionen muß eine Schachtelung möglich sein, wobei die Schachtelungstiefe -ebenso wie die Anzahl der Funktions- bzw. Prozedurparameter- in gewissen realistischen Grenzen nicht beschränkt sein sollte.
Die aktuellen Parameter sollen zur Compilierzeit hinsichtlich Anzahl und Typ mit der formalen Parameterliste verglichen werden.

2.2.6 Grundoperationen

Arithmetische und Vergleichs-Operatoren müssen in vollem Umfang für jegliche Form von Algorithmen zur Verfügung stehen.
Logische Operatoren sind für den Datentyp BIT gefordert (siehe 2.2.1 und 2.3: Bitmanipulation).

2.2.7 Ablaufsteuerung

In diesem Abschnitt werden Sprachmittel betrachtet, die eine Kontrolle über den Programmablauf ermöglichen.

a. Operatorhierarchie:
   Die Definition der Priorität eines Operators (und damit die Reihenfolge der Abarbeitung) sollte mit der mathematischen Definition übereinstimmen. Wichtiger jedoch ist es, mit Hilfe von Klammern die Abarbeitungsreihenfolge selbst festlegen zu können.
b. Blockstruktur:
   Im Sinne eines klar konzipierten Programms ist eine Blockstruktur unerläßlich. Dabei ist unter einem Block folgendes zu verstehen /17/:
   Ein Block besteht aus einer Anzahl von Anweisungen (Rechenvorschrift). Dazu kommen die Deklarationen für benötigte Hilfsbezeichnungen (lokale Variable). Der Block selbst wird begrenzt durch spezielle Kennungen (BEGIN-END-Symbole).

Blöcke können geschachtelt werden.
Weitere Begriffe, die einen Block charakterisieren, sind die Begriffe Gültigkeitsbereich und Lebensdauer. Der Gültigkeitsbereich einer Bezeichnung umfaßt den Block, in dem sie vereinbart ist, mit Ausnahme all der inneren Blöcke, die eine weitere Vereinbarung derselben Bezeichnung enthalten.
Die Lebensdauer eines Objekts ist definiert durch die Zeit, in welcher der Block dynamisch (zur Laufzeit) existiert.
Gültigkeitsbereich einer Bezeichnung und Lebensdauer des durch sie bezeichneten Objekts stimmen im allgemeinen nicht überein (z.B. bei geschachtelten Blöcken); die Lebensdauer umfaßt jedoch stets den Gültigkeitsbereich.

c. Schleifen:
Schleifen sind -auch in der Prozeßprogrammierung- ein unerläßliches Strukturierungsmittel.
Eine Schleife sollte nur an einer fest definierten Stelle verlassen werden, entweder am Schleifenanfang oder am -ende.

d. Sprünge:
Sprünge können durch entsprechende Sprachmittel vermieden werden. Dennoch kann es -abhängig vom jeweiligen Problem- wünschenswert sein, Sprungmöglichkeiten in einer Sprache zu besitzen.

e. Bedingte Anweisungen:
Die bedingte Anweisung ist ähnlich wie die Schleife eine unerläßliche Forderung in der Prozeßprogrammierung. Dabei soll aus Gründen der Strukturierung und der Programmsicherheit die vollständige Form mit beiden Alternativen (IF..THEN..ELSE) verfügbar sein. Über diese einfache Form hinaus muß die Möglichkeit der Fallunterscheidung einschließlich eines Fehlerausgangs gegeben sein.

2.2.8 Ein-/Ausgabe

Neben der Ein-/Ausgabe mit dem technischen Prozeß über dafür spezielle elektronische Geräte (Prozeß-interface, siehe 2.3) wird eine ausreichende Kommunikation mit den Standard-E/A-Geräten gefordert. Dabei werden folgende Übertragungsformen benötigt:

- gepuffert/ungepuffert
- direkt/sequentiell
- binär/alphanumerisch

Das Adressieren der Geräte sollte innerhalb des Programms durch Namen erfolgen (logische Adressierung), um eine gewisse Hardwareunabhängigkeit des Programms zu erreichen. Dazu notwendig ist ein globaler Programmteil, in dem die Zuordnung der physikalischen Geräteadressen zu den Namen erfolgt (im Sinne des Systemteils bei PEARL).

Ein abgeschlossener E/A-Auftrag sollte auf seine korrekte Durchführung überprüft werden können (Fehlerausgang bei inkorrekter Abarbeitung eines E/A-Auftrages).

2.3 Spezielle Kriterien für die Prozeßprogrammierung

Für Aufgaben aus dem Bereich der Prozeßautomatisierung werden von einer Sprache zusätzliche Mittel gefordert, die über die allgemeinen Anforderungen hinausgehen (siehe auch /1/):

- Parallele Aktivitäten
  Eine der wichtigsten Charakteristika in der Prozeßprogrammierung ist das Aufteilen eines Problems in mehrere -zur Laufzeit unabhängige- Programmeinheiten (Multitasking). Dabei muß gewährleistet sein, daß eine Task zum richtigen Zeitpunkt gestartet, beendet, verzögert, unterbrochen und wiederaktiviert wird. Eine Task muß innerhalb von dem Problem vorgegebener Zeitschranken eine Antwort auf eine Anforderung erarbeiten (Erfüllen einer Bedingung in Realzeit). Der Zugriff auf Datenbereiche, über welche die Tasks miteinander kommunizieren, muß durch geeignete Synchronisationsmechanismen geregelt sein.
- IT-Verarbeitung
  Signale vom Prozeß, die zu unvorhergesehenen Zeitpunkten im Rechner eintreffen (Alarme), müssen erkannt und vorrangig verarbeitet werden können.
- Prozeß-Ein-/Ausgabe
  Die Kommunikation mit dem technischen Prozeß wird über dafür speziell ausgelegte E/A-Geräte abgewickelt.
- Bitmanipulation
  Operationen auf entsprechenden Datentypen (siehe 2.2.1 und 2.2.6).
- Simulationsmöglichkeiten für die Testphase
- Zugriff auf Bibliotheksprogramme

Die Wechselwirkung der Forderungen für die Prozeßprogrammierung mit dem Betriebssystem ist in /1/ näher erläutert.

2.4 Allgemeine Beurteilungskriterien für die Sprachanwendung

Die hier aufgestellten Forderungen stellen qualitative Kriterien für die Bewertung von Programmiersprachen dar. Die nachfolgenden Eigenschaften werden allgemein von einer Programmiersprache gefordert, nicht nur in der Prozeßprogrammierung, so daß in diesem Abschnitt eine etwas allgemeinere Aussage über die Brauchbarkeit und Eignung einer Sprache gemacht werden kann.

2.4.1 Modularität

Ein Programm heißt modular aufgebaut, wenn es so in "Programmoduln" zerfällt, daß man sich von der Richtigkeit eines Moduls überzeugen kann, ohne auf seine Einbettung in größere Einheiten zu achten.
Modularität ist eine wesentliche Voraussetzung für strukturiertes Programmieren. Sprachmittel, mit welchen ein Programm modular aufgebaut werden kann, sind: Unterprogramme (Prozeduren, Funktionen), Blöcke, Schleifen, Fallunterscheidungen und bedingte Anweisungen.

2.4.2 Erlernbarkeit

Unter Erlernbarkeit einer Sprache soll der Aufwand verstanden sein, den ein Anwendungsingenieur (mit der üblichen mathematischen Vorbildung) betreiben muß, um ein Programm mittlerer Schwierigkeit (Programmumfang ca. 1ØØØ Anweisungen) in dieser Sprache schreiben und auf einem Rechner implementieren zu können.

2.4.3 Effizienz

Effizienz bedeutet die bestmögliche Nutzung der vorhandenen Hilfsmittel zur Erfüllung der Aufgabenstellung (gewöhnlich Rechenzeit und Speicher).
Zeiteffizienz und Speichereffizienz sind meist nicht miteinander verträglich.

2.4.4 Implementierungsaufwand

Der Aufwand während der Programmier- und Testphase soll möglichst klein gehalten werden.
Dabei erfolgt eine Unterteilung in den Detail-Entwurf, der eine Verfeinerung der Systemanalyse für die Codierung darstellt, die Codierungsphase selbst, welche die Umsetzung des Entwurfs in die Programmiersprache beinhaltet, Testzeiten während der Übersetzungsphase und die Zeiten, in der das Programm zur Laufzeit getestet wird.

2.4.5 Testhilfen

In den Aufwand für die Implementierung fließen wie oben erwähnt die Zeiten für die Testphase mit ein. Diese Zeiten sind durch entsprechende Hilfsmittel klein zu halten, der Test selbst sollte möglichst umfassend sein.
Es können mehrere verschiedene Testhilfen unterschieden werden: Testhilfen, die das System anbietet, Testhilfen, die von der Sprache bereitgestellt werden und Testhilfen, welche der Programmierer selbst definiert und in das Programm einbaut.

2.4.6 Zuverlässigkeit, Sicherheit

Nach DIN 4ØØ41 wird unter Zuverlässigkeit die Fähigkeit eines Systems verstanden, innerhalb eines vorgegebenen Zeitraums die gestellten Anforderungen zu erfüllen (Aussage über das Nichtauftreten von Störungen).
Erweitert man diese Aussage auf Programmiersprachen, so kann hier durch die speziellen Anforderungen für Sprachen eine detailliertere Definition gegeben werden: Unter dem Begriff Zuverlässigkeit werden alle Eigenschaften zusammengefaßt, welche die Fehlersicherheit eines Programms beeinflussen. Dazu gehören interne Programmfehler, Fehler bedingt durch eine Störung des Grundsystems und Fehler beim Bedienen des Programms (falsche Eingabedaten und ähnliches).

Das Abfangen von Fehlern sollte zu einem frühestmöglichen Zeitpunkt erfolgen: während des Übersetzens. Dazu gehört die Überprüfung von Parametern bei Unterprogrammen auf Anzahl und Typ, das Überwachen von Indexgrenzen bei Feldern und ähnliches.
Eine weitere Forderung, welche die Fehlersicherheit eines Programms erhöht, sind explizite Variablendeklarationen. Bei impliziter Deklaration werden auch solche Namen akzeptiert, die aus Schreibfehlern resultieren.
Weitere Fehlersicherheit kann erreicht werden durch den Einbau von Fehlerbehandlungsprogrammen, mit denen Fehler zur Laufzeit erkannt und abgefangen werden können. Zusätzliche Verbesserungen sind durch modular aufgebaute Programme zu erzielen. Fehler sind dadurch leichter einzugrenzen und somit schneller zu beheben.
Verbunden mit der Sicherheit, speziell der Ausfallsicherheit, ist der störungsfreie Lauf des Rechners. Fehler in der Hardware (Speicher, Peripherie, Paritätsfehler) oder Fehler in der Systemsoftware (Betriebssystem) dürfen das korrekte Arbeiten des Programms nicht unmöglich machen sondern höchstens behindern.
Der erreichbare Grad der Zuverlässigkeit und Sicherheit ist vor allem eine Frage des Aufwandes, der in das Programm bzw. die Sprache gesteckt werden muß, um sämtliche Möglichkeiten auszutesten.
Umfangreiche Programmsysteme können nie absolut sicher gegen Fehler gemacht werden, da es unmöglich ist, jede Programmvariante mit Tests abzudecken.

2.4.7 Dokumentation, Lesbarkeit

Zu jedem Programm gehört eine ausreichende Dokumentation, damit eine korrekte Bedienung und Modifizierung auch von anderen Personen als dem Entwerfer gewährleistet ist.

Eine Programmdokumentation muß so angelegt sein, das verschiedene Zielgruppen in ihren Wünschen befriedigt werden: angefangen von einem groben Überblick über die Funktionsweise des gesamten Systems bis hin zur exakten Detailbeschreibung einzelner Programmabschnitte. Somit gehören zu einer Dokumentation auch Beschreibungsmittel, die über die Möglichkeiten einer Sprache hinausgehen, wie zum Beispiel Struktugramme, Ablaufdiagramme usw.
Von der Sprache selbst wird gefordert, daß sie Kommentarzeilen an beliebigen Stellen im Programm in beliebigem Umfang zuläßt.
Zusätzlich zu dieser Dokumentationsform sollte ein Programm ein hohes Maß an Lesbarkeit besitzen. Dazu gehören Eigenschaften wie:

- Ausreichende Länge von Bezeichnern für eine verständliche Erklärung ihrer Verwendung (signifikante Characters).
- Annäherung der Schreibweise für Befehle an mathematische Konventionen bzw. an eine natürliche Sprache, z.B. englisch.
- Sprachmittel, die ein modulares Programmieren ermöglichen, um damit einen Überblick über größere Programmeinheiten zu bekommen.
- Strukturierungsmöglichkeiten des Listings (Einrücken), Bezeichnen von Blöcken (Schachtelungstiefe) sowie Kennzeichnen externer Variable.

2.4.8 Wartung

Die Programmwartung dient der Programmpflege, der Aufrechterhaltung der vereinbarten Programmleistung, sowie der Programmverbesserung, der Anpassung an veränderte Umweltbedingungen.
Diese Eigenschaften sind in der Prozeßprogrammierung besonders relevant. Einmal erstellte Automatisierungsprogramme werden in der Regel sehr lange eingesetzt.

In dieser Zeit können Änderungen im Prozeß bzw. am Rechner durchgeführt werden, was im Programm eventuell entsprechend berücksichtigt werden muß.
Die Wartungsfreundlichkeit eines Programms ist sehr eng verknüpft mit den Begriffen Dokumentation, Lesbarkeit und Modularität, aber auch Aussagen über Zuverlässigkeit und Sicherheit können die Wartung erheblich beeinflussen (siehe 2.4.1, 2.4.6 und 2.4.7).

3. DEFINITION DES PROZESSMODELLS

3.1 Der reale Prozeß

Für den praktischen Teil des Vergleichs sollte ein Problem ausgewählt werden, welches folgende Forderungen erfüllt:

- Die zu lösende Aufgabe ist in der Realität tatsächlich vorhanden und bildet ein abgegrenztes Problem.
- Der Prozeß beinhaltet typische Eigenschaften, die bei der Prozeßprogrammierung eine wichtige Rolle spielen.

Der nach diesen Forderungen ausgewählte Prozeß ist dem chemischen Bereich entnommen. Die durch den Prozeß gestellte Aufgabe ist zwischen der Prozeßüberwachung und der Prozeßsteuerung angesiedelt.
Es handelt sich dabei um ein Gerät zur Messung von Urankonzentrationen nach dem Prinzip der Differenzdichtemessung /18/. Das Meßgerät ist im In-Line-Betrieb direkt in einen größeren Prozeß aus dem Bereich der Wiederaufbereitung eingegliedert und ermöglicht eine laufende Kontrolle der Konzentrationen.
Die Differenz-Dichtemeßstelle besteht aus einer Handschuhbox, in der die Meßanordnung sowie die nötigen Rohrleitungen untergebracht sind (Abb.2).
Gemessen wird die beladene Phase und als Bezugspunkt die unbeladene Phase. Das eigentliche Meßsignal erhält man durch die Differenzbildung der beiden Meßwerte.
Über Rohrleitungen werden die Meßlösung bzw. die Referenzlösung in zwei Behälter gefüllt (II). Die Niveauregelung übernehmen zwei metallische Schwimmkörper (IV). Diese werden durch die Lösungen emporgehoben und lösen bei einer bestimmten Höhe die induktiven Meßgeber (V) aus. Aufgrund dieser Signale muß veranlaßt werden, daß keine weitere Lösung mehr angesaugt wird.

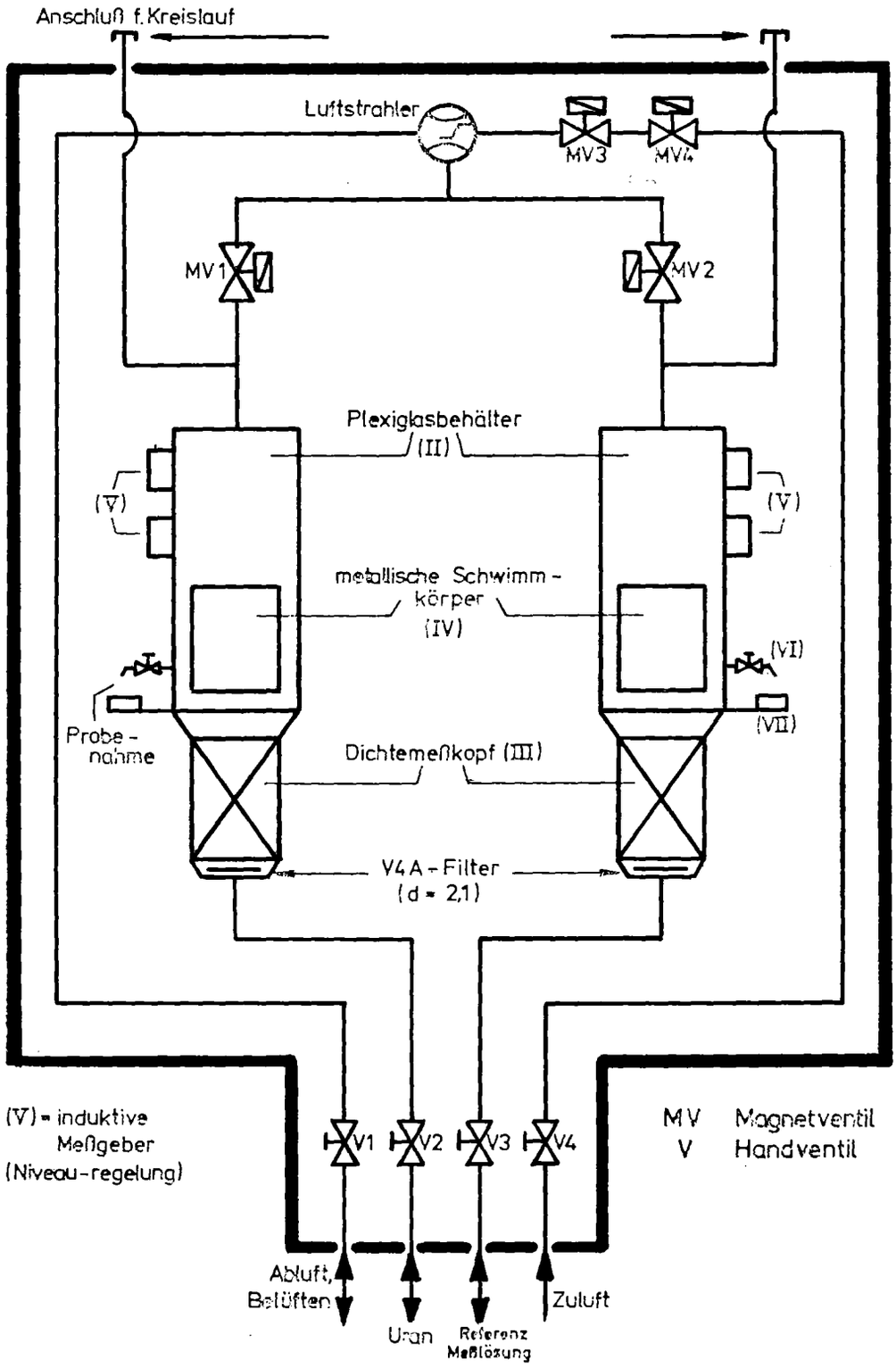


Abb 2: Fließschema zum doppelten Dichtemeßgerät /18/

Aus Sicherheitsgründen sind die Meßgeber doppelt ausgelegt, um beim Ausfall des unteren zu verhindern, daß die Lösung weiter angesaugt wird. In diesem Fall wird das Magnetventil MV4 geschlossen, das sonst normalerweise immer offen ist. Über zwei Schwingungsdichtemesser (III) werden die Dichten gemessen, anschliessend werden die Lösungen wieder abgelassen, und ein erneuter Meßvorgang kann gestartet werden.
Das Füllen und Leeren der Behälter wird durch die Magnetventile (MV1-4) gesteuert.
Die Meßvorgänge wiederholen sich zyklisch. Ein Meßzyklus läßt sich dabei in 3 Phasen zerlegen:

1. Ansaugphase (Füllen der Behälter)
2. Meßphase (Messen der Dichten)
3. Auslaufphase (Leeren der Behälter)

Die einzelnen Phasen sind durch die Stellungen der Magnetventile gekennzeichnet.

3.2 Das Modell der Differenzdichte

Das Differenzdichte-Meßgerät wird charakterisiert durch eine gewisse Anzahl von physikalischen Signalen und Daten. Für die Überwachung und Steuerung des Meßgerätes mit Hilfe eines Prozeßrechners sind jedoch nicht alle Werte relevant. In einem ersten Abstraktionsschritt wird deshalb eine Reduzierung der Werte auf die tatsächlich benötigten Signale vorgenommen. Dazu kommen die Signale, die vom Prozeßinterface MOPEK zusätzlich angeboten werden und eine verbesserte Kontrolle des Meßablaufs ermöglichen. Mit diesen Signalen wird der technische Prozeß in einer Form beschrieben, die als Grundlage für weitere Entwurfsbetrachtungen dient. Modell siehe Abb.3.
Die Signale haben folgende Bedeutung:
Zur Ansteuerung der Magnetventile werden vier Signale benötigt: MV1-MV4. Diese Signalwerte liegen im Prozeßinterface über eine längere Zeit hinweg konstant an und können zur Kontrolle rückgelesen werden: Statuswerte ST-MV1 - ST-MV4.
Der Prozeß erzeugt bis zu vier Signale, die Auskunft über den Füllstand der Behälter geben. Diese vier Werte werden im Interface dynamisch abgefragt (siehe 1.2). Wertänderungen erzeugen einen Interrupt. Die Interrupts können maskiert werden. IT-IMG1 - IT-IMG4, Maske 3.
Die beiden Meßwerte über die Dichten der Flüssigkeiten werden in analoger Form an das Interface abgegeben und dort digitalisiert: AW1, AW2.
Zur Überwachung des zeitlichen Ablaufs des Programms (siehe 4.2) wird der Intervallzeitgeber (IVZG) verwendet. Er kann gesetzt werden (Setzen IVZG) und zählt in vorgegebenen Zeitintervallen zurück auf Null. Die Null im IVZG erzeugt einen Interrupt (IT-IVZG).

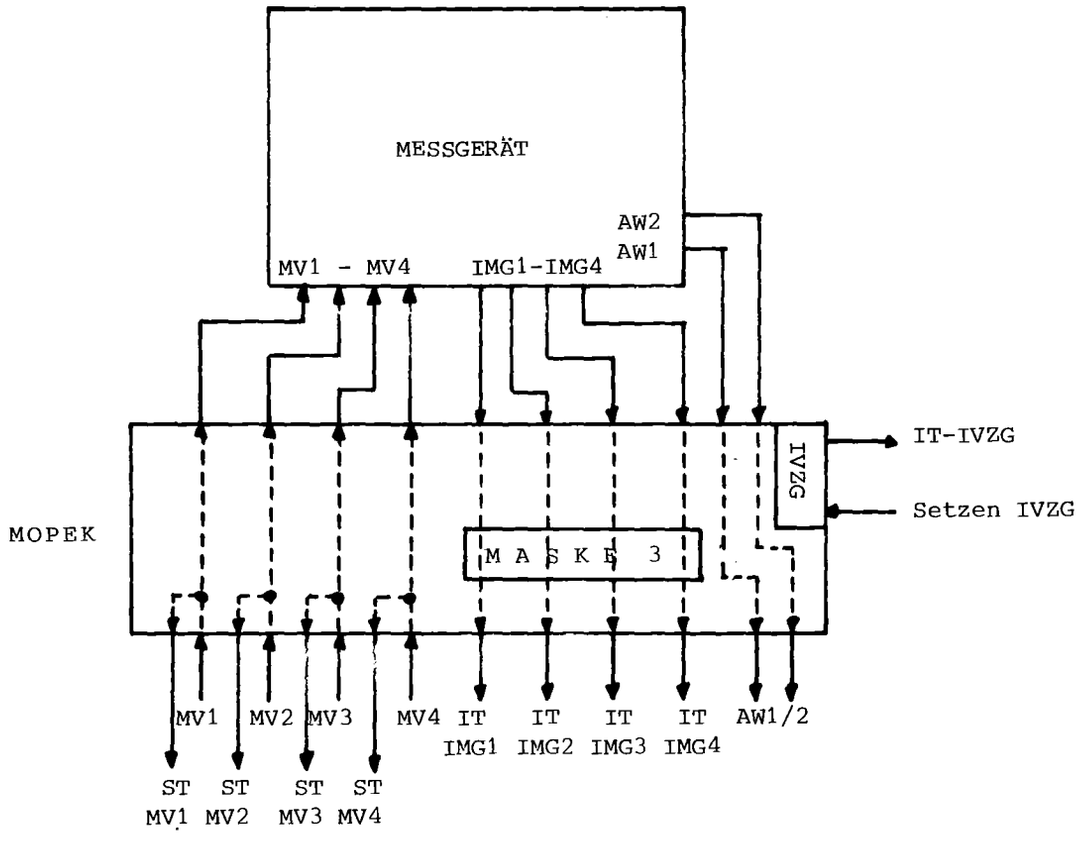


Abb.3 Das statische Modell der Differenzdichte

3.3 Systemanalyse mit SADT

Auf der Grundlage des erstellten Modells (siehe 3.2) kann nun die eigentliche Systemanalyse der Prozeßüberwachung durchgeführt werden. Um einen möglichst objektiven Sprachvergleich zu gewährleisten, ist es notwendig, das Problem (Programm) während der Entwurfsphase in einer sprachunabhängigen Form zu definieren (siehe 2.1).
Die Systemanalyse wird in SADT (Structured Analysis and Design Technique, /19/, /2Ø/) durchgeführt.
SADT ist eine graphische Beschreibungsform. Als Sprachmittel stehen zur Verfügung:

- rechteckige Kästchen
- Pfeile

In SADT sind zwei Beschreibungsarten möglich:
Actigramme und Datagramme
Mit Actigrammen wird der eigentliche Aktionsablauf beschrieben. Die Kästchen bilden dabei die Aktionen, die ausgeführt werden, die Pfeile zwischen den Kästchen repräsentieren die Daten, die zwischen den Aktionen ausgetauscht werden.
Bei Problemen, die große Datenmengen beinhalten, besteht die Möglichkeit, diese mit Datagrammen zu beschreiben. Dabei wird die Bedeutung von Kästchen und Pfeilen gegenüber den Actigrammen umgekehrt:

	Actigramm	Datagramm
Kästchen	Aktion	Daten
Pfeile	Daten	Aktion

Im weiteren wird nur auf die Actigramme eingegangen, da nur sie zum Entwurf herangezogen werden. Die Datenflüsse sind so geartet, daß auf Datagramme verzichtet werden kann.

Bei einem Actigramm haben Pfeile unterschiedliche Bedeutung, je nachdem, auf welcher Seite sie in ein Kästchen einmünden bzw. es verlassen (Abb.4):

- Aktivieren einer Aktion
- Eingabedaten für eine Aktion
- Ausgabedaten einer Aktion
- Ein-/Ausgabe mit externen Geräten

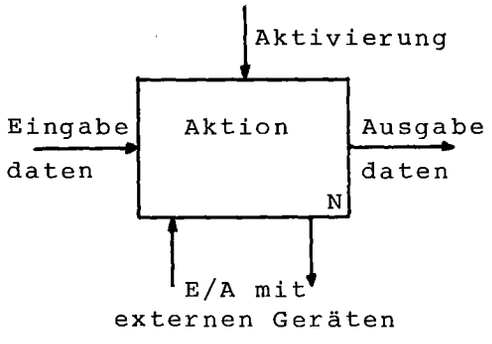


Abb.4 Sprachelemente in SADT

Mit SADT wird eine Systemanalyse nach der "Top-Down"-Methode durchgeführt. Diese Methode erlaubt eine modulare Detaillierung des Entwurfs. Dies bedeutet eine Strukturierung des Problems. Jede Aktion kann in weitere Aktionen zerlegt werden. Diese Aktionen liefern eine detailliertere Aussage des übergeordneten Begriffs. Zur eindeutigen Identifizierung einer Aktion (Kästchen) bzw. für den Bezug der einzelnen Detaillierungsstufen werden die Aktionen numeriert:
Alle Kästchen auf einer Stufe erhalten eine fortlaufende Nummer N (siehe Abb.4). Eine Verfeinerungsstufe wird gekennzeichnet durch das Aufführen des Namens der übergeordneten (verfeinerten) Aktion und der zugehörigen Nummer.

Die Anzahl der Nummern liefert die Verfeinerungsstufe. Die oberste Stufe wird mit Ø numeriert; diese Null taucht nicht in den einzelnen Verfeinerungen auf.
Im folgenden wird das Modell zur Prozeßüberwachung näher erläutert.
Die ausführliche SADT-Beschreibung ist in Anhang A gegeben. Eine Grenze der Detaillierung nach unten ist gegeben durch die Forderung, daß die Sprachunabhängigkeit des Entwurfs gewahrt bleibt.
Das System zur Prozeßüberwachung gliedert sich in fünf Teile:

- die Initialisierung der einzelnen Programmeinheiten
- die eigentliche Prozeßablaufsteuerung
- die Abarbeitung von Alarmen aus dem Prozeß
- die Ausgabe von Daten, Statuswerten und Ventilstellungen
- die Ausgabe von Fehlermeldungen

Die Prozeßablaufsteuerung untergliedert sich ihrerseits in fünf Teile, wobei die beiden ersten Teile (Grundstellung Meßgerät und Überwachung Meßzyklusanzahl) eine vorbereitende bzw. kontrollierende Funktion innehaben. Die weiteren drei Teile spiegeln die einzelnen Phasen eines Meßzyklus des Gerätes wider (siehe 3.1). Eine Übersicht der Stufen gibt Tabelle 1 wieder.

AØ Prozeßüberwachung

- A1 Initialisierung
- A2 Prozeßablaufsteuerung
  - A21 Grundstellung Meßgerät
  - A22 Überwachung Meßzyklusanzahl
  - A23 Ansaugphase
    - A231 1.Takt
    - A232 2.Takt
    - A233 3.Takt
  - A24 Meßphase
    - A241 Meßdatenerfassung
    - A242 Meßdatenaufbereitung
  - A25 Auslaufphase
    - A251 Öffnen MV1/2
    - A252 Statuskontrolle
- A3 Alarmsystem
- A4 Daten-/Status-/Ventilstellung-Ausgabe
- A5 Fehlerausgabe

Tabelle 1: Stufen des SADT-Modells zur Prozeßüberwachung

3.4 Das dynamische Modell der Prozeßüberwachung

Ein Entwurf nach SADT allein erlaubt nur sehr eingeschränkte Aussagen über das exakte zeitliche Verhalten des dargestellten Systems. Grundsätzlich jedoch gilt: Eine Aktion wird erst dann aktiviert, wenn die "darüberliegende" Aktion fertig abgearbeitet wird. Aktionen können allerdings auch übersprungen werden, was mit SADT nicht immer feststellbar ist (siehe Abb.5).

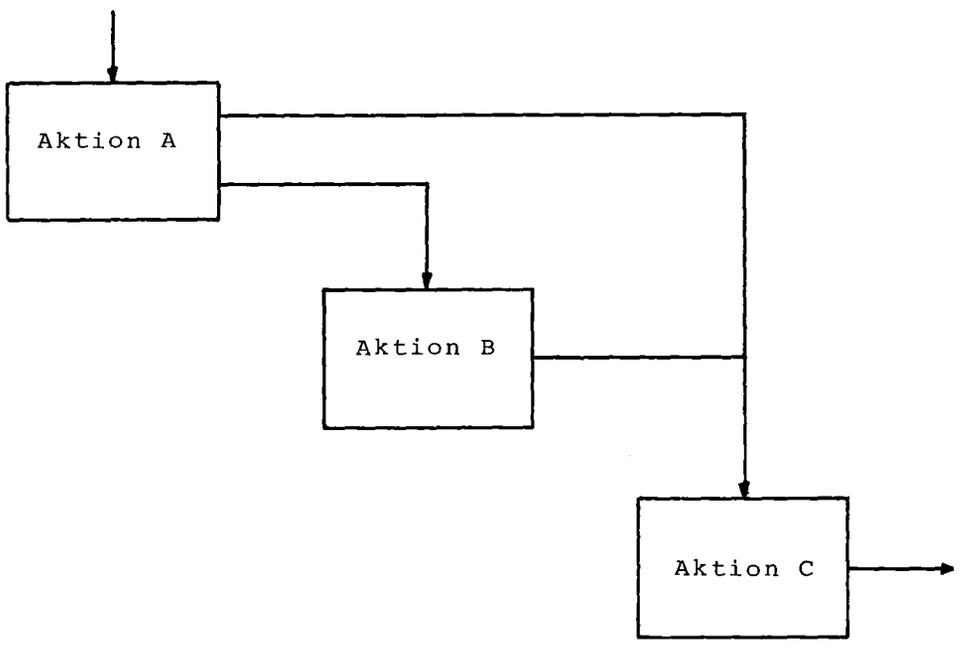


Abb.5 Überspringen einer Aktion in SADT
Eine Aktion B wird abhängig vom Ergebnis einer Aktion A ausgeführt oder übersprungen. Das Ergebnis von A ist nicht erkennbar.

Aufgrund dieser Gegebenheiten ist es nötig, eine zusätzliche Beschreibung zu erstellen, welche das dynamische Verhalten des SADT-Modells widerspiegelt. Der dynamische Ablauf ist hier durch ein Signaldiagramm realisiert. Es enthält sämtliche für die Prozeßüberwachung relevanten Signale in Form von Pfeilen auf einer Zeitachse aufgetragen (siehe Anhang B).
Die Signale sind bei diesem Modell auf verschiedenen Ebenen eingetragen. Diese Ebenen stellen einen ersten Schritt zur Realisierung hin dar (siehe 4.1). Jede Ebene entspricht einer Einheit aus dem zu entwickelnden Programmsystem. Die jeweiligen Signale sind diesen Programmeinheiten bzw. dem Prozeß zugeordnet. Die Signale zur Zeitsteuerung (IVZG) sind aus der Ablaufsteuerung herausgenommen und werden gesondert in dem Diagramm aufgeführt.
Die Zerlegung des Modells in mehrere Ebenen liefert einen genauen Überblick über das zeitlich verzahnte Auftreten und über das Zusammenwirken der Signale.
Mit Hilfe des SADT-Entwurfes und des dynamischen Modells kann nun mit relativ geringem Aufwand die Implementierung (Codierung) des Überwachungssystems durchgeführt werden.

4. IMPLEMENTIERUNG

4.1 Abdeckung der Vergleichskriterien

Um eine möglichst große Anzahl der unter 2 aufgestellten Kriterien abzudecken, werden an die Realisierung des Überwachungssystems gewisse Anforderungen gestellt:

- Aufteilen des Systems in mehrere unabhängige Programmeinheiten (Multitasking).
- Kommunizieren mit dem Prozeß über das Prozeßinterface MOPEK.
- Realisieren der Programmeinheiten als IT-Tasks (siehe 4.3).
- Simulieren des realen Prozesses während der Testphase.
- Ausschöpfen der Sprachmöglichkeiten.

Ein weiterer Gesichtspunkt ist die Verwendung der Standard-E/A-Geräte, wie sie bei der Prozeßprogrammierung eingesetzt werden:

Operateur-Konsole
Drucker
Platte
Band
Sichtgerät (Nachbildung der Prozeßwarte mit Hilfe eines halbgraphischen Video-Farb-Displays)

Eine Aufzählung der verschiedenen Anweisungen (Sprachmittel) und der Aufträge an die einzelnen E/A-Geräte erfolgt in Anhang D.

4.2 Programmphilosophie

Das Programmsystem zur Überwachung des technischen Prozesses kann zunächst in drei Hauptblöcke unterteilt werden:

- Initialisierung des Systems
- Prozeßablaufsteuerung
- Bearbeitung der Prozeßalarme

Nach der Initialisierung des Programmsystems (Übernahme von Parametern, Aktivieren der beiden weiteren Blöcke) erfolgt die Überwachung des Prozesses. Dabei ist folgende Strategie zugrundegelegt:
Die Prozeßablaufsteuerung ist ereignisgesteuert, das heißt, das Überwachungssystem ist nur dann aktiv, wenn Signale vom Prozeß abgegeben wurden, die eine Reaktion vom Programm erfordern (Übernahme von Statuswerten, Alarmbehandlung, Meßdatenverarbeitung).
Die Ereignissteuerung wird vom System zeitlich überwacht. Damit besteht die Möglichkeit zu überprüfen, ob gewisse Signale innerhalb vorgegebener Zeitspannen vom Prozeß eingetroffen sind (Time-out-Kontrolle). Bleibt ein Signal aus, so kann durch eine entsprechende Fehlerbehandlung verhindert werden, daß das System in einen undefinierten Zustand übergeht.
In der übrigen Zeit ist das System inaktiv und gibt den Rechnerkern für andere Aufgaben frei.
Die Länge der Time-out-Zeitspannen ist über Parameter frei wählbar. Die Summe dieser Zeiten ergibt die sogenannte maximale Zykluszeit.
Neben der Zeitüberwachung wird die Ablaufsteuerung und die Alarmbehandlung durch eine Programmvariable synchronisiert, der Ablaufvariablen. Diese Variable ändert ihren Wert an festen Programmstellen, so daß damit überprüft werden kann, wie weit ein Meßzyklus fortgeschritten ist.

In der Ablaufsteuerung werden überwacht:

- die Ablaufvariable (AV)
- der Prozeßzustand (Statuswerte)
- der Zeitpunkt des Starts der Ereignisbehandlung (Time-out-Kontrolle)

Die Alarmbehandlung selbst benötigt nur den Wert der Ablaufvariablen zum Zeitpunkt des Eintreffens eines Alarms. Die Überwachung dieser drei Werte erfolgt immer dann, wenn die Ablaufsteuerung nach einer Unterbrechungsphase wieder aktiviert wird. Während der Unterbrechung könnten durch Eingriffe von außen die Signalwerte verändert sein. Die ständige Kontrolle dieser wesentlichen Werte für das Programm ermöglicht ein genaues Verfolgen des Prozesses und erlaubt im Fehlerfall einen gezielten Eingriff in das System. Dies bedeutet eine Erhöhung der Programmzuverlässigkeit (siehe auch 2.4.6 und 7.6).

4.3 Detailkonzept zur Realisierung der Prozeß-überwachung

Aufgrund der Forderungen in 4.1 ist das Prozeß-überwachungssystem folgendermaßen realisiert:
Das System ist zerlegt in fünf verschiedene Tasks:

- INPAS : Start der Prozeßüberwachung.
  Unterscheidung, ob Simulation- oder Realbetrieb.
  Start der entsprechenden Aktivierungstasks (INABST und/oder INITSS).
- INABST: Anschließen der Ablaufsteuerung (ALARM1) und der Alarmbehandlung (ALARM3) an IVZG und Alarmausgang des MOPEK-Interface.
- PAREIN: Übernahme der Parameter zur Steuerung der Meßzyklen.
  Start der Ablaufsteuerung.
- ALARM1: Ablaufsteuerung im engeren Sinne.
- ALARM3: Behandlung der Prozeßalarme.

Dazu kommen:

- INITSS: Anschließen des Simulationssystems (ALARM2) an das MOPEK-Interface.
- ALARM2: Simulationssystem zur Nachbildung des realen Prozesses während der Testphase (siehe 4.4).

Die Tasks ALARM1, ALARM2 und ALARM3 sind als sogenannte Interrupt-Tasks realisiert /1/ und an verschiedene MOPEK-Funktionseinheiten (dynamische Digitaleingänge, MOPEK-Uhr) angeschlossen. Die Änderung eines Wertes in einer dieser Funktionseinheiten erzeugt einen Interrupt, worauf die zugehörige Task vom Betriebssystem aktiviert wird.
Das Zusammenspiel aller Tasks ist in Abb.6 gegeben (siehe auch Anhang B, das dynamische Modell).

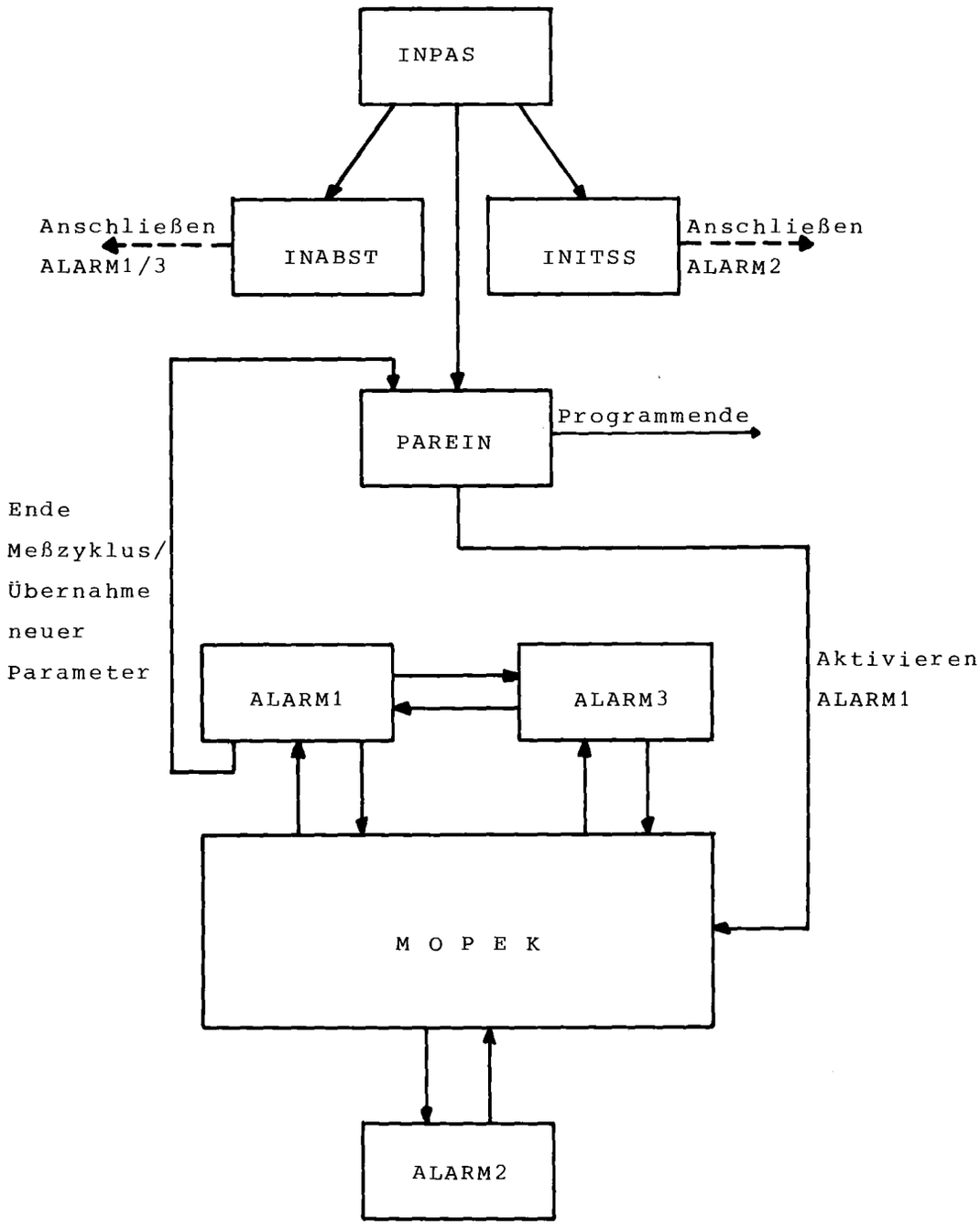


Abb.6 Zusammenspiel der Tasks zur Prozeßüberwachung

Während der eigentlichen Prozeßüberwachung sind nur die Tasks ALARM1 und ALARM3 aktiv sowie im Falle der Prozeßsimulation die Task ALARM2. Nach Ablauf einer vorgegebenen Anzahl von Meßzyklen wird die "Parameter-task" PAREIN erneut aktiviert, um eventuell neue Zyklusparameter einzulesen oder das Programmsystem zu beenden.
Das hier vorgestellte System zur Prozeßüberwachung ist mit Ausnahme von BASIC voll implementiert (Gründe hierfür siehe 6.1). In BASIC existiert nur ein Teil der Task ALARM1 (Takt1-Takt3 der Ansaugphase, siehe auch 3.3, Tabelle 2). Der implementierte Teil stellt die eigentliche kritische Phase der Überwachung dar. ALARM1 wartet an dieser Stelle auf die Signale der Alarmbehandlung ALARM3. Zu diesem Zeitpunkt muß ein exakt synchroner Ablauf der beiden Tasks gewährleistet sein, da sonst ein Fehlerfall angenommen wird (siehe 4.2).

4.4 Das Simulationssystem

Während der Testphase ist im allgemeinen der reale Prozeß nicht verfügbar. Er muß deshalb durch ein Simulationssystem im Rechner nachgebildet werden. Das Modell für die Simulation ist mit den gleichen Entwurfsmitteln wie die Prozeßüberwachung erstellt (Modelle siehe Anhang A und B).
Die Prozeßüberwachung selbst darf durch die Simulation auf dem Rechner nicht beeinflußt werden, es ist eine scharfe Trennung zwischen den beiden Systemen notwendig. Diese Trennung wird erreicht, indem die Schnittstelle in das Prozeßinterface MOPEK verlegt wird (Schnittstelle in der Hardware). Signale vom und zum Simulationssystem werden über zusätzliche MOPEK-Ein-/Ausgänge mit den Ein-/Ausgängen der Prozeßüberwachung über Kabel verbunden.
Die Entkopplung der Software ist wie folgt gesichert:

- Das Programm zur Simulation des realen Prozesses wird wie die Prozeßüberwachung als IT-Task an einen Alarmeingang des Prozeßinterface angeschlossen. Es erhält die höchste Priorität, um so die realen Gegebenheiten nachzubilden.
- Signale vom und zum Simulationssystem werden nur über die Rückkopplung im MOPEK ausgetauscht.

Ein Berührungspunkt in der Software ist nur während der Initialisierungsphase gegeben, wo das Simulationsprogramm über Parameter als MOPEK-IT-Task aktiviert wird.
Die Kopplung der Signale ist in Abb.7 festgehalten.
Die Loslösung vom realen Prozeßgeschehen erlaubt es darüber hinaus, Aussagen über eine zeitlich bedingte Synchronisationsgrenze der einzelnen Sprachen zu erhalten (Zeitraffung):
Durch Verringern der maximalen Zykluszeit (siehe 4.2) wird eine untere Grenze sichtbar, bei der ein synchroner Ablauf des Überwachungssystems noch garantiert

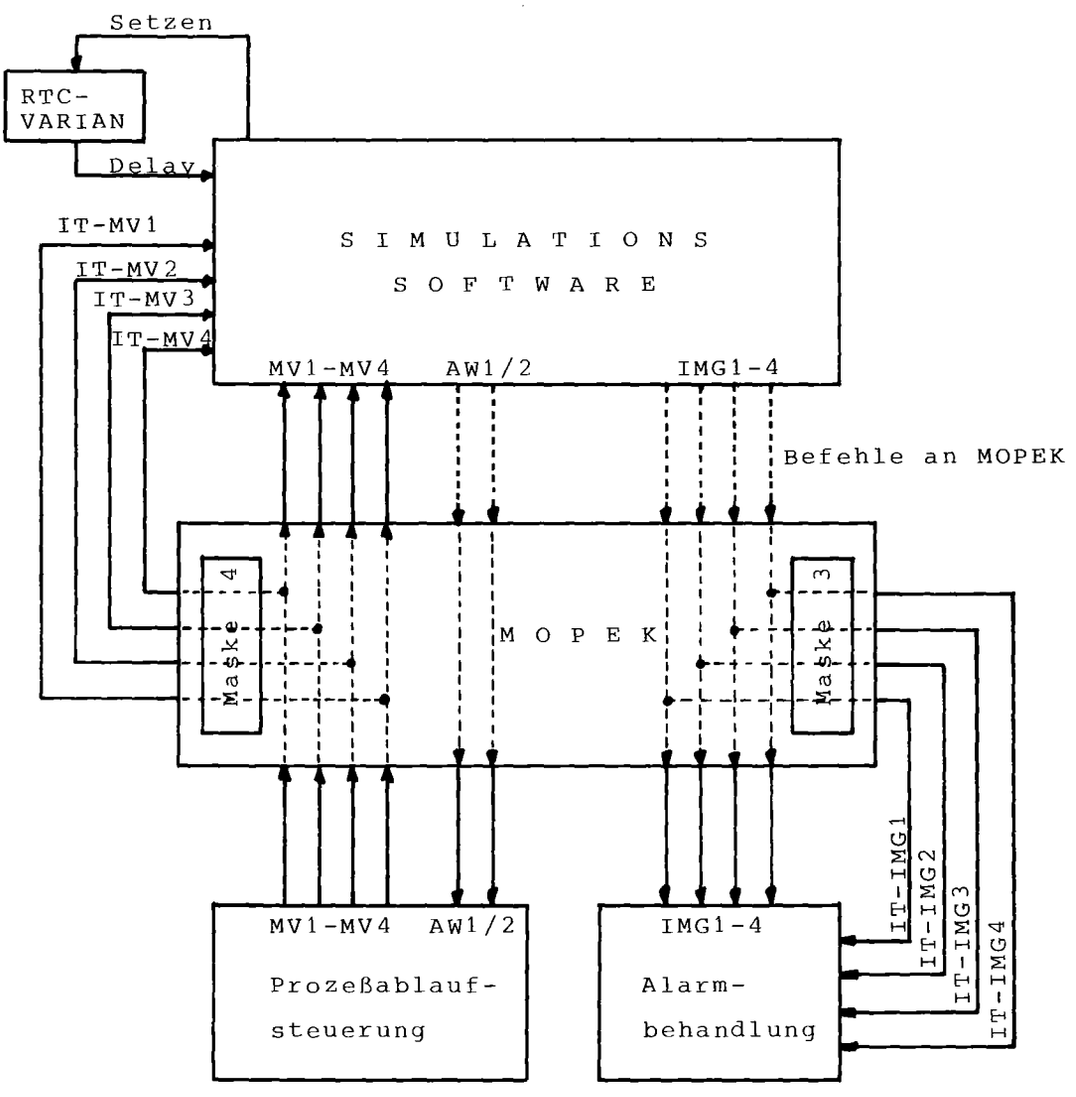
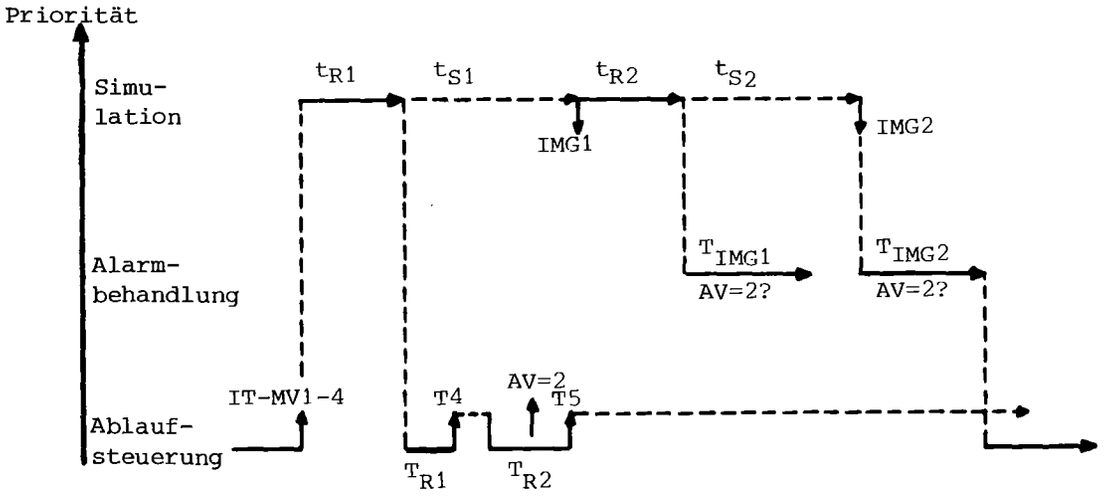


Abb.7 Kopplung der Signale im Prozeßinterface MOPEK während der Simulation

ist. Diese so gefundene minimale Zykluszeit ermöglicht eine grobe Aussage über die Verarbeitungszeit von Alarmen:
Innerhalb einer bestimmten Time-out-Zeitspanne muß die Alarmbehandlung ein Signal vom Prozeß (Simulationssystem) erkannt und verarbeitet haben. Dieses Signal stammt von den induktiven Meßgebern (siehe 3.1, Abb.2) und zeigt an, daß der Behälter gefüllt ist. Die Ausgabe dieser Signale vom Simulationssystem erfolgt zunächst zu beliebigen Zeitpunkten. Dies entspricht jedoch nicht dem "normalen" realen Prozeßgeschehen; hier werden diese Signale nur an ganz bestimmten Stellen im zeitlichen Prozeßablauf erzeugt. Das "Wohlverhalten" des Simulationssystems wird erreicht, indem auf die für dieses Geschehen spezifischen Time-out-Zeitspannen in der Prozeßüberwachung zugegriffen wird. Dies bedeutet eine Kopplung der Software von Prozeßüberwachung und Simulation.
Die Signalausgabe muß innerhalb dieser Zeitspanne erfolgen, um so einen weiteren "richtigen" Ablauf der Prozeßüberwachung zu gewährleisten.
Es ergeben sich somit folgende Zeitverhältnisse (die Aussagen sind gemacht anhand der kritischen Phase in Takt 2 während des Ansaugens, siehe auch Anhang B):



Zeitlicher Ablauf während der kritischen Phase in Takt 2

mit t_{R1}, t_{R2}	Rechen- und BS-Zeiten für Simulation
t_{S1}, t_{S2}	Verzögerungszeiten für Ausgabe IMG1, IMG2 (Simulationstask suspendiert)
IMG1, IMG2	Signale an Alarmbehandlung
IT-MV1-4	Start der Simulation
T_{R1}, T_{R2}	Rechen- und BS-Zeiten für Ablaufsteuerung
T4, T5	Time-out-Zeiten für Ablaufsteuerung (Ablaufsteuerung suspendiert)
T_{IMG1}, T_{IMG2}	Bearbeitungszeiten für die Alarme (sprachunabhängige Zeiten)

Die Alarmbehandlung erfolgt zeitgerecht, wenn zunächst gilt:

$$T_{IMG1} + T_{IMG2} \leq T5$$

Der "richtige Ablauf des Systems ist gewähleistet, wenn zum Zeitpunkt des Starts der Alarmbehandlung die Ablaufvariable AV den Wert 2 annimmt. Das heißt, die Verzögerungszeit t_{S1} muß mindestens so gewählt werden, daß die Anweisung AV=2 abgearbeitet werden kann:

$$T_{R1} + T4 + T_{R2}' \leq t_{S1} \qquad (1)$$

mit $T_{R2} = T_{R2}' + T_{R2}''$

T_{R2}' ist die Teilzeit, bis AV=2 gesetzt ist.

T_{R2}'' ist die Restzeit

Bei Erfüllung von (1) kann eine weitere Beziehung abgeleitet werden:

$$t_{R2} + t_{S2} + T_{IMG2} + T_{IMG1}' \leq T5 + T_{R2}'' \qquad (2)$$

mit $T_{IMG1} = T_{IMG1}' + T_{IMG1}''$

T_{IMG1}' ist die Restzeit von T_{IMG1}, falls $t_{S2} < T_{IMG1}$

$$\text{Falls} \left\{ \begin{matrix} t_{S1} & T_{R1} + T4 + T_{R2} \\ t_{S2} & T_{IMG1} \end{matrix} \right\}, \text{ dann gilt: } \begin{matrix} T_{R2}'' = \emptyset \\ T_{IMG1}' = \emptyset \end{matrix} \qquad (3)$$

Unter Berücksichtigung von (3) kann (2) aufgelöst werden:

$$T_{IMG2} \leq T5 - t_{S2} - t_{R2} \qquad (4)$$

mit t_{R2} = const = Ø,Ø5ms (theoretisch berechnet)

$T5, t_{S2} = F(T)$ $\qquad T = \sum_i TO_i$

TO_i = Einzel-Time-out-Zeiten

Beispiel: $t_{S1} = Ø.2T_A$ $\qquad T_A = Ø.2T$

$t_{S2} = Ø.1T_A$ $\qquad$ = Ansaugzeit

$T4 = Ø.1T_A$

$T5 = Ø.7T_A$

=> $t_{S1} = Ø.Ø4T$ $\qquad T4 = Ø.Ø2T$

$t_{S2} = Ø.Ø2T$ $\qquad T5 = Ø.14T$

nach(3) gilt: $T_{R1} + T_{R2} \leq Ø.Ø2T$

$T_{IMG1} \leq Ø.Ø2T$

=> für (4): $T_{IMG2} \leq Ø.12T$

für T = 1.5sec (gemessene kritische Zeit für FORTRAN, siehe 6.3, Tabelle 3) folgt daraus:

$T_{IMG2} \leq Ø.18$ sec

Nach /1/ teilt sich die Gesamtrechenzeit auf in:

9Ø % Betriebssystem - Rechenzeit

1Ø % eigentliche Rechenzeit

=> $T_{IMG2} \leq 18$ msec

Innerhalb dieser Zeit werden durchgeführt (Anhang D):

- Einlesen des Alarmmusters (MOPEK-Zugriff)
- Analyse der Ablaufvariablen
- Setzen der Ablaufvariablen
- Auswahl der Signale für den Prozeß
- Ausgabe des Signals (MOPEK-Zugriff)
- Nachricht an Ablaufsteuerung (E/A-Auftrag)

5. VERGLEICH DER SPRACHMERKMALE

In diesem und in den beiden nachfolgenden Kapiteln werden die Sprachen an den unter Kapitel 2 aufgestellten Kriterien gemessen, hier speziell an denen unter 2.2 und 2.3.
Eine tabellarische Zusammenfassung dieses Vergleichs ist in Anhang C gegeben.

5.1 Datentypen

Mit Ausnahme von BASIC sind in allen Sprachen die Typen INTEGER und REAL vorhanden, BASIC kennt nur den REAL-Typ. Darüberhinaus wird von PASCAL CHAR angeboten. Der Typ BIT kann jedoch durch die PASCAL-Fähigkeit der expliziten Typdefinition zusätzlich geschaffen werden.
Das gleiche gilt für die Typen zur speziellen Behandlung von Zeiten und Interrupt (Prozeßtypen), die in keiner Sprache standardmäßig vorhanden sind. Allerdings ist die Handhabung solcher selbstdefinierter Typen problematisch, da die dazu nötigen Operationen ebenfalls selbst geschaffen werden müssen (siehe 2.2.6 und 5.6).

5.2 Datenstrukturen

Jede der hier untersuchten Sprachen kennt Deklarationen für Felder. Eine Einschränkung bei der Dimensionszahl gibt es bei BASIC. Es sind maximal nur zweidimensionale Felder möglich.
Eine Verbunddeklaration ist nur in PASCAL gegeben.

5.3 Kommunikationsbereiche

Das Betriebssystem stellt einen globalen Speicherbereich, den "Blank-COMMON" zur Verfügung (siehe auch Aussagen in /1/). Dieser Bereich ist direkt nur von FORTRAN/IFTRAN (COMMON-Definition in FORTRAN) und

von DASMR aus ansprechbar.
Eine globale Variablendeklaration ist in jeder Sprache möglich:

BASIC : Ein BASIC-Programm wird "sequentiell" als eine Einheit niedergeschrieben. Somit ist jede im Programm eingeführte Variable im weiteren Verlauf bekannt und verfügbar.

FORTRAN/IFTRAN: In FORTRAN/IFTRAN gibt es zunächst keine globalen Variablen. In Anlehnung an die COMMON-Definition, die eine Kommunikation zwischen mehreren Tasks ermöglicht, können Variable einer Task als global deklariert werden, indem sie in einem "benannten" COMMON-Bereich definiert sind. Der benannte COMMON stellt einen Speicherbereich dar,auf den alle Programmoduln eines Programms zugreifen können.

PASCAL : Ein PASCAL-Programm kann in zwei Teile gegliedert werden: den "Prefix", der alle externen Prozeduren anschließt und für das System relevante Werte beinhaltet, und das PASCAL-Programm. Ein PASCAL-Programm wird als eine spezielle Anweisung definiert (Block). Alle Variablen, die für diesen Block deklariert sind, gelten für das gesamte Programm (detailliertere Angaben über Blockstruktur sind in 5.7.1 zu erhalten).

5.4 Vereinbarungen

In BASIC wird die Variablendeklaration implizit durchgeführt.
FORTRAN/IFTRAN erlauben implizite und explizite Typdeklarationen.
PASCAL verlangt eine explizite Definition jedes Namens.
Konstantendeklaration ist nur in PASCAL möglich.

5.5 Unterprogramme

Prozeduren sind in jeder Sprache bekannt. Eine Einschränkung gibt es bei BASIC: es können explizit keine Funktionen definiert werden, es sind nur einige vordefinierte mathematische Funktionen verfügbar.
PASCAL erlaubt über die Forderungen hinaus, Prozeduren rekursiv aufzurufen.
Eine Aussage über die maximale Parameteranzahl ist nur sehr schwer zu machen -ebenso für die Schachtelungstiefe-, da die realen Grenzen nicht ausgelotet sind.
In diesem Vergleich sind Versuche mit bis zu 2Ø Parametern erfolgreich verlaufen.

5.6 Grundoperationen

Alle Sprachen erfüllen voll die Anforderungen, BASIC kennt jedoch keine logischen Operatoren.
Neben den existierenden Operatoren ist in keiner Sprache die Möglichkeit vorgesehen, für entsprechende Datentypen neue Operationen zu definieren.

5.7 Ablaufsteuerung

1. Operatorhierarchie:
   In jeder Sprache werden die Operatoren in mathematischer Reihenfolge abgearbeitet. Klammerungen sind in allen Sprachen im beliebigen Umfang zugelassen.
2. Blockstruktur:
   Eine Struktur im Sinne der Definition in 2.2.7.b ist im vollen Umfang in keiner Sprache vorhanden.
   PASCAL bietet eine BEGIN-END-Klammerung an, die sogenannte "Compound-Anweisung". Ein Unterschied zur Definition besteht unter anderem bei Deklarationen:
   Innerhalb eines mit BEGIN-END geschachtelten "Blocks"

ist keine Typdeklaration erlaubt. Blockstruktur ist jedoch in dem Sinne vorhanden, daß BEGIN-END-Blöcke geschachtelt werden können. Lokale Variable sind nur -im Kopf- bei Funktionen und Prozeduren möglich. Dabei gelten die Aussagen über Lebensdauer und Gültigkeitsbereich einer Variablen bei geschachtelten Funktionen/Prozeduren im Sinne der Definition von 2.2.7.b. Diese Aussagen gelten auch für die globalen Variablen, wobei diese Variablen als lokal für diejenige BEGIN-END-Schachtel, die das gesamte Programm umklammert, angesehen werden.

3. Schleifen:
   IFTRAN und PASCAL entsprechen voll der Forderung, BASIC und FORTRAN besitzen jeweils nur einen Schleifentyp mit Ausgang am Ende.
   Die allgemeine Definition von Schleifen bei IFTRAN bringt die Möglichkeit mit sich, Schleifen an beliebigen Stellen zu verlassen (mit Hilfe der EXIT-Anweisung /12/). Die Verwendung dieser EXIT-Anweisung steht jedoch im Widerspruch zu den Forderungen in 2.2.7.c.
4. Sprünge:
   In jeder Sprache sind generell Sprünge erlaubt. Das Sequential PASCAL auf der V75 erlaubt im Gegensatz zur Wirth'schen Definition keine Sprünge.
5. Bedingte Anweisungen:
   Die volle IF..THEN..ELSE-Konstruktion ist nur in IFTRAN und PASCAL möglich. IFTRAN geht dabei über die Forderung noch hinaus, in dem es mit der ORIF-Konstruktion /12/ eine erweiterte IF-Anweisung ermöglicht.
   In BASIC und FORTRAN ist nur das eingeschränkte IF..THEN (ohne ELSE) verfügbar.
   Die geforderte Fallunterscheidung ist nur eingeschränkt vorhanden: durch berechnete Sprünge in BASIC, FORTRAN und IFTRAN; durch die CASE-Anweisung

in PASCAL. Alle diese Konstruktionen sehen jedoch keinen Fehlerausgang vor.
Allerdings kann IFTRAN mit der ORIF-Erweiterung der IF-Anweisung die voll geforderte Fallunterscheidung nachbilden (der Fehlerausgang wird duch ELSE realisiert).

5.8 Ein-/Ausgabe

Die minimalste Form einer E/A ist in BASIC implementiert: eine Kommunikation erfolgt nur mit der Operateurkonsole. Alle anderen Sprachen entsprechen im vollen Umfang den Forderungen aus 2.2.8, wobei in PASCAL keine ungepufferte E/A möglich ist.
Die Überprüfung auf eine korrekt durchgeführte E/A ist in den einzelnen Sprachen wie folgt realisiert:

BASIC: keine Möglichkeiten

FORTRAN/IFTRAN: Durch Erweiterung der READ/WRITE-Anweisungen kann das korrekte oder fehlerhafte Ende einer Ein-/Ausgabe überwacht werden. Daneben ist mit Hilfe eines Unterprogrammaufrufs (IOCHK) der Status eines abgeschlossenen E/A-Auftrages überprüfbar.

PASCAL: Bei satzweiser binärer E/A kann die Anzahl der übertragenen Worte abgefragt werden.

5.9 Überprüfung der Sprachmittel für die Prozeßprogrammierung

Da keine der untersuchten Sprachen für die Echtzeit- bzw. Prozeßprogrammierung konzipiert ist, werden die geforderten Merkmale im allgemeinen von keiner Sprache unmittelbar bereitgestellt.
Ausnahme: BASIC bietet die Möglichkeit an, mittels der WAIT-Anweisung eine Programmverzögerung durchzuführen /5/.
Die vom Betriebssystem VORTEX II zur Verfügung gestellten Real-Zeit-Dienste können in keiner Sprache auf

Sprachebene verwendet werden, sondern sie sind über CALL-Aufrufe als ASSEMBLER-Unterprogramme an die Sprachen angeschlossen. Das gleiche gilt auch für die Kommunikation mit dem Prozeßinterface MOPEK und mit einem als Prozeßwarte definierten Video-Farb-Display (VFD).
Synchronisationsmechanismen sind weder in den Sprachen vorgesehen, noch werden sie von VORTEX unterstützt. Wie in Kapitel 4.3 erwähnt besteht die Möglichkeit, ein Prozeßprogramm, das über MOPEK mit einem technischen Prozeß kommuniziert, als IT-Task zu realisieren. Dies ist jedoch kein spezielles Sprachmerkmal, sondern eine Möglichkeit, die von VORTEX und MOPEK angeboten wird.
Weit wichtiger in der Realzeitprogrammierung ist jedoch die Reaktionszeit beim Auftreten eines Interrupts. Diese Zeitbetrachtungen können in dieser Arbeit nur sehr grob durchgeführt werden (siehe 4.4).

5.1Ø Möglichkeiten des ASSEMBLERS DASMR

Mit einem ASSEMBLER vom Umfang von DASMR ist es prinzipiell möglich, jede aufgestellte Forderung mit entsprechendem Aufwand zu erfüllen. Allerdings erfolgt dies auf einer sprachlich niedrigeren Ebene und mit dadurch bedingten primitiveren Mitteln. Dieser Mehraufwand bei der Implementierung muß allerdings bei einer Bewertung entsprechend berücksichtigt werden.
Um DASMR mit in den Vergleich einbeziehen zu können, sollen seine vorhandenen Sprachmittel an den Forderungen von 2.2 und 2.3 gemessen werden. Alle als nicht vorhanden aufgeführten Sprachmittel können, wie oben erwähnt, entsprechend nachgebildet werden.

1. Datentypen:
   Beim ASSEMBLER wird die Realisierung der Werte auf der Maschine sichtbar. Jeder Wert wird zunächst auf seine binäre Darstellung zurückgeführt. Die Bitmuster können je nach der Art der Operation als INTEGER, REAL ("Floating-Point-Number"), CHAR (Character-Konstante), BIT oder als "Prozeßtyp" interpretiert werden.
2. Datenstrukturen:
   Felder und Verbunde sind als Sprachmittel nicht bekannt. Sie können jedoch mit Hilfe von Indizierung (bei Feldern) und indirekter Adressierung (bei Verbunden) einfach nachgebildet werden.
3. Kommunikationsbereiche:
   Der vom Betriebssystem VORTEX zur Verfügung gestellte COMMON-Bereich ist von DASMR aus direkt ansprechbar.
   Globale Variablen sind in DASMR prinzipiell verfügbar.
4. Vereinbarungen:
   Jeder Bezeichner in DASMR muß vereinbart werden. Dabei besteht die Möglichkeit, Variablen (mit der DATA-Anweisung) und Konstanten (mit der EQU-Anweisung, /3/, /4/) zu deklarieren.
5. Unterprogramme:
   DASMR bietet zwei Arten von Unterprogrammen an:
   - Makros
   - Subroutinen

   Der Aufruf eines Makros (durch Aufführen seines Namens) entspricht dem offenen und wiederholten Einbau, der Aufruf einer Subroutine (durch CALL-Aufruf) entspricht dem geschlossenen Einbau von Unterprogrammen in die aufrufende Programmeinheit.

6. Grundoperationen:
   Neben den mathematischen Operationen (4 Grundrechenarten) sind noch die logischen Operationen (Bitmanipulation) bekannt.
   Vergleiche sind nur im Zusammenhang mit Sprüngen (bedingte Sprünge) möglich. Operatoren können über Makros definiert werden.
7. Ablaufsteuerung:
   a. Operatorhierarchie: nicht bekannt
   b. Blockstruktur: nicht bekannt
   c. Schleifen: nicht bekannt
   d. Sprünge: Es sind verschiedene Arten von Sprüngen möglich: Unbedingte, bedingte, indirekte und markierte Sprünge.
   e. Bedingte Anweisungen: Bedingte Anweisungen sind reduziert auf bedingte Sprünge.
8. Ein-/Ausgabe:
   DASMR kann jede Art der E/A, die von VORTEX unterstützt wird,mit Hilfe von Makros ausführen (entspricht voll den Forderungen von 2.2.8) Genauere Aussagen sind in /1/ nachzulesen.
9. Prozeßprogrammierung:
   Alle Forderungen für die Echtzeit-Programmierung sind in DASMR erfüllt (ansprechbar über Makros).
   Die Kommunikation mit dem Prozeßinterface und der Prozeßwarte wird auf die Standard-E/A zurückgeführt.

Eine Zusammenfassung der DASMR-Fähigkeiten ist in Anhang C gegeben.

6. VERGLEICH DES AUFWANDS WÄHREND DER IMPLEMENTIERUNGS- UND TESTPHASE

6.1 Vorbemerkungen

Das System zur Überwachung des Differenzdichte-Meßgerätes wurde in den Sprachen FORTRAN, IFTRAN, PASCAL und DASMR auf der VARIAN V75 in der obigen Reihenfolge implementiert. In BASIC konnte kein lauffähiges Programm erstellt werden, da zu viele Voraussetzungen, die bei der Prozeßprogrammierung gefordert sind, zur Zeit der Implementierung nicht erfüllt waren:
Der BASIC-Interpreter arbeitet im Background /1/, /2/, /3/. Das bedeutet unter anderem, daß immer nur ein BASIC-Programm zu einem Zeitpunkt abgearbeitet werden kann (andere Möglichkeit siehe bei PASCAL). Somit ist in BASIC kein Multitasking möglich. Dieser Umstand wiegt jedoch nicht so schwer wie die Tatsache, daß keine ASSEMBLER-Unterprogramme (zum augenblicklichen Zeitpunkt) angeschlossen werden können. Das heißt, es ist keine Kommunikationsmöglichkeit mit dem Prozeß über MOPEK vorhanden, ebenso kann die geforderte umfangreiche E/A mit den Standardgeräten und der Prozeßwarte nicht durchgeführt werden.
Darüberhinaus besteht keine Möglichkeit, ein BASIC-Programm für spätere Aktivierungen auf einem geeigneten Speichermedium abzulegen.
In BASIC wurde aus diesen Gründen nur ein kleiner Teil des Programmsystems exemplarisch realisiert, um zumindest Aussagen über die Verwendung der Sprachmittel bzw. über die Sprachstrukturen zu erhalten.

6.2 Änderungen des Konzeptes bei PASCAL

Bei PASCAL wird das Programm wie bei BASIC durch einen Interpreter abgearbeitet. Dieser Interpreter wird aber im Gegensatz zu BASIC zu dem PASCAL-Programm hinzugebunden. Ein solches Programm ist im Foreground lauffähig und kann auf die VORTEX-Realzeit-Dienste über ASSEMBLER-Programme zugreifen.
Das Stack- und Heap-(Keller- und Halde-) Prinzip in PASCAL, das eine dynamische Speicherbelegung zur Laufzeit bedingt, erfordert, daß das Überwachungssystem durch eine Task realisiert wird:
Prozedurparameter werden im PASCAL-Stack übergeben, bei Generierung von Verbunden wird auf dem Heap Speicherplatz für den Verbund reserviert. Die jeweils aktuelle Länge des Stacks sowie der benötigte Speicherplatz für Verbunde sind zur Übersetzungszeit nicht bekannt. Aus diesem Grund wird der Speicher erst während der Laufzeit dynamisch zugewiesen oder wieder freigegeben ("pulsierender" Speicher). Sind gleichzeitig mehrere PASCAL-Programme aktiv, so versuchen diese, sich Speicherbereiche zuzuordnen, was zu Überschneidungen und somit zu Fehler und Abbruch der Tasks führt. Aus diesem Grund ist die alarmbehandelnde Task als Prozedur fest in die Prozeßüberwachung eingebaut. Dies bedeutet allerdings eine zeitlich eingeschränkte Reaktion auf Alarmsignale vom Prozeß, da der Prozeduraufruf nur an festdefinierten Stellen im Programm möglich ist.
Die oben erwähnte Einschränkung erfordert weiterhin, daß bei den in PASCAL geschriebenen Initialisierungstasks darauf zu achten ist, daß erst eine Task terminiert ist, bevor eine weitere PASCAL-Task gestartet werden kann.

6.3 Aufwand während der Implementierungs- und Testphase

1. Detail-Entwurf und Codierung

Die Verfeinerung des SADT-Modells wurde nur soweit durchgeführt, wie Sprachunabhängigkeit des Entwurfs gewahrt blieb. Im Hinblick auf die Codierung bedeutet dies, daß das Modell jeweils sprachspezifisch verfeinert werden muß. Die Detaillierung des Entwurfs und das Codieren wurden iterativ durchgeführt, das heißt Erfahrungen, die während der Codierung gemacht wurden, sind wiederum in den Detail-Entwurf eingeflossen. Umgekehrt natürlich beeinflußte die Verfeinerung die jeweilige Codierung. Durch diese Arbeitsweise ist es nicht möglich, den Aufwand für Detail-Entwurf und Codierung getrennt anzugeben:
Für Detail-Entwurf und Codierung wurden folgende Zeiten benötigt (ohne Übertragen des Programms auf endgültige Datenträger):

FORTRAN	IFTRAN	PASCAL	DASMR
5 Std	8 Std	22 Std	27 Std

Die Prozeßüberwachung in FORTRAN ist weniger in Programmoduln zerlegt als bei den anderen Sprachen (bedingt durch häufige Verwendung von Sprung-Anweisungen). Somit konnte in FORTRAN ohne großen Organisationsaufwand für das Zusammenfügen der Moduln das schnellste Umsetzen des SADT-Modells in Programmcode erreicht werden.
Weniger Organisation bedeutet aber auch geringeren Programmumfang. Der Umfang wird hier in Anzahl von Lochkarten (LK) angegeben. Diese Werte entsprechen nicht der Anzahl der Anweisungen, da eine Anweisung über mehrere Lochkarten verteilt sein kann.

1424 LK	1485 LK (2Ø97 LK)	2Ø91 LK	3168 LK

Der Wert in Klammern bei IFTRAN ist der Umfang des äquivalenten FORTRAN-Programms.
Diese Zahlen beinhalten auch die für ein Programm notwendigen Kommentarkarten. Der Kommentar hat folgenden Umfang:

FORTRAN	IFTRAN	PASCAL	DASMR
446 LK	415 LK	541 LK	4Ø7 LK
	(635 LK)		

Das äquivalente FORTRAN-Programm enthält zusätzlich als Kommentare alle eigentlichen IFTRAN-Befehle: 22Ø LK.
Somit existieren an "reinem" Programmcode:

978 LK	1Ø7Ø LK	155Ø LK	2761 LK
	(1462 LK)		

DASMR enthält zusätzlich Kommentare direkt auf den Befehlskarten. Diese werden hier nicht berücksichtigt.
Der hier aufgeführte Programmumfang beinhaltet nicht die Anschlußprogramme (Laufzeitsystem, siehe 6.4) und die Programmteile, welche aus der Standardbibliothek entnommen werden (MOD, INT, IFIX, FLOAT, usw.).

2. Testaufwand

Bevor das eigentliche Problem in den einzelnen Sprachen gelöst werden konnte, waren gewisse Vorarbeiten zu leisten. Der dafür benötigte Aufwand wird gesondert in Kapitel 6.4 behandelt.
Beim Testen von Programmen treten zwei Arten von Fehler auf: Syntaktische Fehler und logische Fehler.
Syntaktische Fehler beruhen zumeist auf Schreibfehlern und werden schon während der Übersetzung abgefangen.
Die logischen Fehler machen sich erst zur Laufzeit bemerkbar, entweder durch einen Laufzeitfehler (führt zu Programmabbruch) oder durch einen fehlerhaften Lauf des Programms. Letztere Fehler sind -abhängig vom jeweiligen Komplexitätsgrad des Programms- meist nur sehr schwierig zu lokalisieren und zu beheben.

Aufgrund der Vorarbeiten (6.4) traten nur sehr wenige Laufzeitfehler auf, so daß diese im Vergleich nicht weiter berücksichtigt werden. Das fehlerhafte Programmverhalten konnte wie folgt korrigiert werden: Programmoduln, welche keinen zeitkritischen Anforderungen unterworfen sind (z.B. Meßdatenaufbereitung, Ausgabeprogramme und ähnliches), wurden getrennt vom Programm in einer jeweils eigenen Testumgebung überprüft. Für die Tests der zeitkritischen Programmteile wurde darauf zurückgegriffen, daß von der Programmphilosophie (4.2) her schon sehr umfangreiche Kontrollen im Programm eingebaut sind (Ablaufvariable, Status). Mit diesen dicht über das Programm verteilten Kontrollen konnten Fehler im Programmablauf sehr eng lokalisiert und ohne zusätzliche Testhilfeprogramme behoben werden.
Für die oben erwähnte Art der Fehlersuche bei fehlerhaftem Programmablauf wurden folgende Testzeiten benötigt:

FORTRAN	IFTRAN	PASCAL	DASMR
64 Std	48 Std	32 Std	48 Std

Diese Zeiten lassen sich weiter aufteilen in Zeiten für die Überprüfung des gesamten Programmsystems mit Hilfe der fest installierten Kontrollen, sowie in Zeiten für die Überprüfung der getrennt testbaren Programmteile (prozentuale Verteilung und entsprechende absolute Zeitwerte):

	Programmsystem	Programmteile
FORTRAN	7Ø % - 44,8 Std	3Ø % - 19,2 Std
IFTRAN	8Ø % - 38,4 Std	2Ø % - 9,6 Std
PASCAL	5Ø % - 16,Ø Std	5Ø % - 16,Ø Std
DASMR	3Ø % - 14,4 Std	7Ø % - 33,6 Std

Die Zahlen für die Programmteile enthalten neben den Modultests auch die Zeiten für die Erstellung einer jeweils geeigneten Testumgebung.

Die Tabelle ist wie folgt zu interpretieren:
Die Programme in IFTRAN, PASCAL und DASMR sind stärker modularisiert als das FORTRAN-Programm. Bei der Implementierung des Problems in IFTRAN traten deshalb durch die größere Modularisierung mehr Fehler im Programmablauf auf als eigentlich zu erwarten war. Zusätzlich sind manche Moduln in FORTRAN und IFTRAN identisch, so daß hier für IFTRAN keine weiteren Testzeiten anfielen.
Der verhältnismäßig hohe Aufwand in PASCAL bei den Programmteilen gegenüber IFTRAN resultiert aus dem aufwendigeren Erstellen von Testumgebungen für ein Modul.
Bei DASMR ist der Testaufwand für die Moduln am größten. Dies ist bedingt durch das Ansteigen der Fehler in den Programmteilen für die Ausgabe von Meßdaten, Statuswerten und Ventilstellungen auf Drucker, Magnetband, Prozeßwarte und Platte. Die meisten Fehler entstanden durch die geforderte detailliertere Beschreibung eines E/A-Auftrags in DASMR gegenüber der E/A-Notation in einer höheren Programmiersprache.
Nicht mit aufgenommen in die Betrachtung sind die Zeiten für die Systemanalyse (ca. 2.5 Mann-Monate für Einarbeiten in das Problem, Beherrschen der SADT-Technik und Erstellen der Modelle) und das Simulationssystem (ca. 1 Mann-Monat mit dazugehörigem Entwurf).
Während der Testphase wurde eine untere zeitliche Grenze für einen "richtigen" Programmablauf ausgelotet. Mit Hilfe der Gleichungen aus 4.4 besteht dabei die Möglichkeit, ungefähre Aussagen über die Reaktionszeit auf Alarme in den einzelnen Sprachen zu geben. Bei einer jeweils minimalen Zykluszeit wurden für die einzelnen Sprachen folgende Werte ermittelt:

	FORTRAN	IFTRAN	PASCAL	DASMR
Zykluszeit	1.5 sec	1.5 sec	1.5 sec	1.Ø sec
Reaktionszeit	Ø.18 sec	Ø.18 sec	Ø.18 sec	Ø.12 sec

Die Reaktionszeit setzt sich zusammen aus BS-Zeiten und eigentliche Rechenzeit (siehe /1/).
Die gewonnenen Ergebnisse zeigen, daß die Reaktionszeiten im Gegensatz zu der Annahme in 4.4 sprachunabhängig sind (genauere Aussagen siehe /1/).

Eine Zusammenfassung der hier aufgeführten Zahlen ist in Tabelle 2 gegeben. Die Werte der Lademoduln werden in 7.3 näher erläutert.

	FORTRAN	IFTRAN	PASCAL	DASMR
Programmieraufwand nach SADT und Detail-Entwurf	5 Std	8 Std	22 Std	27 Std
Programmumfang in Lochkarten	1424	1485 (2Ø97)	2Ø91	3168
davon sind Kommentare	446	415 (635)	541	4Ø7
"reiner" Programmcode	978	1Ø7Ø (1462)	155Ø	2761
Testaufwand: Gesamtzeit	64 Std	48 Std	32 Std	48 Std
Aufteilung (%) in "System"- und "Modul"-Fehler	7Ø/3Ø	8Ø/2Ø	5Ø/5Ø	3Ø/7Ø
entsprechende absolute Werte (Std)	44.8/19.2	38.4/9.6	16/16	14.4/33.6
min.Zykluszeit	1.5 sec	1.5 sec	1.5 sec	1.Ø sec
Reaktionszeit	Ø.18 sec	Ø.18 sec	Ø.18 sec	Ø.12 sec
Größe der Lademoduln (Worte):				
INPAS	4523	4548	4Ø64	132
INABST	4749	4749	4254	432
PAREIN	7993	8Ø45	6439	3Ø35
ALARM1	11217	1214Ø		4999
ALARM3	4951	5388		541
ALARM1+ALARM3	16168	17528	14132	554Ø
INITSS				393
ALARM2				1Ø13

Tabelle 2: Ergebnisse während der Implementierungs- und Testphase

6.4 Allgemeine Vorarbeiten und Zusatzprogramme

Die geforderte Art der Realisierung der rechnergeführten Überwachung und Steuerung des technischen Prozesses (4.1) erfordert einen gewissen Zeitaufwand für die Einarbeitung, was hier gesondert betrachtet werden soll.
Das Hauptproblem war gegeben durch die Forderung, ein ereignisgesteuertes System (4.2) mit Hilfe von mehreren Tasks zu realisieren. Die Synchronisation dieser Tasks und das Aufstellen eines passenden Zeitrasters zur Steuerung des Prozeßablaufs konnte innerhalb eines Mann-Monats abgeschlossen werden.
Diese Aufgabe war verknüpft mit der Implementierung der Prozeßablaufsteuerung in FORTRAN. Über die in 6.3 aufgezählten Testhilfsmittel hinaus waren besonders für die Tasksynchronisation weitere Kontrollausgaben erforderlich. Die Ausgabe dieser Kontrollwerte auf die Standard-E/A-Geräte brachte jedoch erhebliche zeitliche Verzerrungen des Programmablaufs mit sich, wodurch eine Kontrolle nicht mehr möglich war. Das Problem wurde gelöst, indem die Kontrollwerte in den von VORTEX bereitgestellten COMMON-Bereich geschrieben wurden und von dort zu einem späteren Zeitpunkt ausgewertet werden konnten.
Das Sequential PASCAL /16/ auf der V75 ist primär nicht für Realzeitaufgaben vorgesehen. Es mußte deshalb eine Erweiterung des PASCAL-Laufzeitsystems vorgenommen werden, um die VORTEX-Realzeit-Dienste zur Verfügung zu stellen. Weiterhin mußte das Ansprechen des Prozeßinterface MOPEK und der Prozeßwarte VFD ermöglicht werden sowie der Zugriff auf den COMMON-Bereich für die Inter-Task-Kommunikation. Die Anschlüsse wurden über speziell zu erstellende ASSEMBLER-Unterprogramme vorgenommen. Dies bedeutet eine Erweiterung der UP-Bibliothek für PASCAL (Aufwand insgesamt 3 Mann-Wochen).

7. VERGLEICH DER SPRACHQUALITÄTEN

7.1 Modularität

PASCAL erfüllt mit kleinen Einschränkungen bei Blöcken alle Anforderungen und unterstützt somit ein modulares Programmieren. IFTRAN kommt PASCAL am nächsten, da es mit Ausnahme von Blöcken alle für die Modularität typischen Sprachmittel zur Verfügung stellt.
FORTRAN bietet für modulares Programmieren nur die Unterprogrammtechnik an. Die vorhandene Schleife und die bedingte Anweisung unterstützen nicht die Modularisierung eines Programms. Bei BASIC und DASMR kann nur sehr eingeschränkt von modularen Programmen gesprochen werden. Während in DASMR die Möglichkeit besteht, Unterprogramme vom aufrufenden Programmteil zu trennen, ist dies in BASIC nicht möglich; BASIC besitzt kein eigentliches Konzept für Unterprogramme.
Mit Ausnahme von PASCAL kann in den übrigen Sprachen die Forderung nach genau definierten Ein- und Ausgängen von Moduln durchbrochen werden (durch Sprünge), was eine erhebliche Einschränkung im Hinblick auf die Definition der Modularität (siehe 2.4.1) bedeutet.

7.2 Erlernbarkeit

Der Begriff der Erlernbarkeit einer Sprache ist nur sehr schwer zu objektivieren. Eine Sprache wie BASIC ist sehr schnell zu erlernen, da sie nur wenige Sprachmittel besitzt und somit keine großen Anforderungen an den Lernenden stellt.
Den größten Umfang der hier untersuchten Sprachen hat zweifellos PASCAL, wobei unter anderem sehr komplexe Sprachmittel (z.B. Verbunde) angeboten werden. Ein Ingenieur, der nur wenige Kenntnisse über die Datenverarbeitung besitzt, muß hier einen erheblichen Mehraufwand leisten, um die Sprache im Sinne der

Definition von 2.4.2 zu beherrschen, kann allerdings dann "bessere" Programme erstellen, als wie dies in BASIC möglich wäre.
FORTRAN ist in seinen Sprachmitteln sehr einfach gehalten. Dies und die formelhafte Schreibweise der Befehle ermöglicht ein schnelles Beherrschen dieser Sprache.
Ein Ingenieur mit FORTRAN-Kenntnissen kann sich ohne großen Aufwand die Sprache IFTRAN aneignen. Dies gilt auch für FORTRAN-Unkundige, da auch bei IFTRAN das über FORTRAN gesagte gilt. Diese Variante beinhaltet einen interessanten Aspekt. Es ist möglich, IFTRAN anzuwenden, ohne alle FORTRAN-Mittel auszunutzen (z.B. keine Sprünge). Dies kann eine Verringerung des Lernaufwandes bedeuten, da durch die bessere Strukturierung der IFTRAN-Mittel das FORTRAN-Sprachkonzept leichter zu erfassen ist.
Das Programmieren mit ASSEMBLER erfordert für einen Anfänger in der DV ein gewisses Umdenken. Der ASSEMBLER ist eine maschinenorientierte Sprache. Jeder Ausdruck, wie er in einer problemorientierten Sprache möglich ist, muß in eine Befehlsfolge zerlegt werden. Weitere Probleme ergeben sich durch die Adreßrechnung. Das Beherrschen eines ASSEMBLERS setzt die Kenntnis über den zugrunde liegenden Rechner voraus. Unter dieser Voraussetzung allerdings ist trotz des meist hohen Befehlsumfangs ein schnelles Erlernen des ASSEMBLERS möglich.

7.3 Effizienz

Die Forderung nach Effizienz von Programmen hat vor allem durch die verbesserte Hardware sehr viel an Gewicht verloren. Auch bei kleineren Rechenanlagen sind die Speicherkapazitäten vergrößert und die Speicherzugriffszeiten verringert. Wichtiger als die Speichereffizienz ist bei den Prozeßrechnern die Zeit-

effizienz, speziell eine minimale Reaktionszeit auf Interrupts. Diese Zeiten sind jedoch stark betriebssystemabhängig und weniger sprachabhängig. Die hier erhaltenen Werte erlauben eine erste Aussage über die Reaktionszeiten (siehe auch 4.4).
Die Reaktionszeiten sowie die Programmgrößen sind in Tabelle 3, Kapitel 6.3 angegeben.
Die Größe der Lademoduln ist wie folgt zu interpretieren:
Die Werte sind korrigierte Werte. Standardmäßige Teile des Betriebssystems, die zu jeder Task dazugebunden werden, sind abgezogen. Verblieben sind die Werte, die durch das jeweilige Laufzeitsystem, Bibliotheksprogramme und durch den Interpreter (bei PASCAL) hinzukommen.
DASMR benötigt den geringsten Speicherplatz, die Werte von PASCAL, FORTRAN und IFTRAN sind in etwa gleich, wobei in der aufgeführten Reihenfolge die Speicherplatzbelegung zunimmt. Die Werte entsprechen der Programmgröße zur Ladezeit, keine Aussagen sind über die dynamische Speicherzuordnung in PASCAL zur Laufzeit zu erhalten. Durch den pulsierenden Speicher kann zur Laufzeit ein erheblicher Speicherzuwachs erfolgen.

7.4 Implementierungsaufwand

Eine eingehende Behandlung dieses Punktes erfolgte in Kapitel 6.

7.5 Testhilfen

In IFTRAN werden vom Sprachkonzept her Mittel zur Verfügung gestellt, mit denen gewünschte Programmoduln (Subroutinen) zu Testzwecken instrumentiert werden können. Als Ergebnis wird ein "Datenpfad" ausgegeben, der die Reihenfolge der abgearbeiteten Befehle, die Art der Entscheidung bei bedingten Anweisungen und

ähnliches beinhaltet. Dazu besteht die Möglichkeit des Editierens von Quellcode /12/. Allerdings ist dieses Testsystem aus Speichergründen auf der V75 nicht verfügbar.
In den übrigen Sprachen sind unmittelbar keine speziellen Testhilfen implementiert. Die Überwachung des Programmablaufs und/oder von Programmvariablen muß vom Programmierer durch explizites Einfügen geeigneter Programm-Kontroll-Ausgaben selbst definiert werden.
Eine Möglichkeit der Ablaufkontrolle ist in FORTRAN durch die "PAUSE"-Anweisung gegeben /6/, mit der vom Programmierer Haltepunkte gesetzt werden können.
Die Firma VARIAN bietet als unterstützende Testsoftware ein "DEBUG"-Programm an. Dieses Programm erlaubt es, auf Objektebene in dem zu testenden Programm Haltepunkte zu setzen, Register und Speicherzellen zu lesen und verändern. Aufgrund dieser Möglichkeiten ist das DEBUG-Programm nur für ASSEMBLER-Programme optimal einsetzbar. Bei höheren Sprachen ist dieses Testsystem kaum von Nutzen, da hier zu viele Kenntnisse über die Umsetzung eines "höheren" Befehls in den entsprechenden Maschinencode erforderlich sind. Ausnahme: Test von ASSEMBLER-Programmen, die an die höhere Sprache angeschlossen sind.

7.6 Zuverlässigkeit, Sicherheit

Möglichkeiten der Fehlererkennung zur Übersetzungszeit sind in PASCAL am meisten vorhanden. Die PASCAL-Fähigkeiten stimmen mit den aufgestellten Forderungen im wesentlichen überein. Hier erweist sich ein vordergründiger Nachteil von PASCAL unter dem Aspekt der Fehlersicherheit als Vorteil. In PASCAL können Prozeduren nicht vorübersetzt und abgelegt werden (Ausnahme: ASSEMBLER-Programme. Dies erlaubt eine umfassende Überprüfung der formalen und aktuellen Parameterlisten

und der globalen Variablen (vorübersetzte Prozeduren könnten keine globalen Variablen enthalten oder müßten sie als extern deklarieren, da diese Variablen im Hauptprogramm vereinbart werden).
In FORTRAN/IFTRAN werden keine Parameter-, Index- und Variablenüberprüfungen vorgenommen. Die Anzahl der Variablen im benannten COMMON-Bereich wird erst während des Bindens überprüft. Der COMMON-Bereich beinhaltet darüberhinaus eine weitere Unsicherheit: Durch den Zwang des Aufzählens sämtlicher Variablen in einer Programmeinheit, die vielleicht nur eine Variable benötigt, können Überschreibungen der anderen Werte erfolgen.
Der BASIC-Interpreter kann hauptsächlich nur Syntax-Fehler erkennen /5/, die durch Schreibfehler entstanden sind.
DASMR kann ebenfalls nur Schreibfehler erkennen, da mit einem ASSEMBLER eine große Klasse von Befehlen möglich ist, die nicht ohne weiteres als eventuelle Fehler erkannt werden können (z.B. Vermischung von Befehlen und Daten im Programm). Besonders gefährlich ist die Möglichkeit, daß Befehlscodes überschrieben werden können. Der Befehlsumfang und die Schreibweise der Befehle verringern beträchtlich die Fehlersicherheit. Die Fehlerrate bei ASSEMBLER wächst bei größeren Programmen im Verhältnis zu höheren Programmiersprachen erheblich an (siehe auch Tabelle 3, Kapitel 6.3).
Dies ist allerdings auch dadurch bedingt, daß bei höheren Sprachen weniger Befehle für ein Problem geschrieben werden müssen als bei einem ASSEMBLER.

7.7 Dokumentation, Lesbarkeit

Die einfachste Form einer Programmdokumentation, Kommentarzeilen, ist in jeder Sprache möglich. Darüberhinaus kann in PASCAL und DASMR ein Kommentar in die gleiche Zeile wie der entsprechende Befehl geschrieben werden.

Die weiteren Forderungen zur Lesbarkeit eines Programms (2.4.7) werden von den Sprachen wie folgt erfüllt:
In PASCAL existieren keine Beschränkungen hinsichtlich der Länge von Bezeichnern (signifikant sind die ersten 1Ø Characters). Durch eigene Typdefinitionen kann die Lesbarkeit erhöht werden.
Die im umfassenden Maße vorhandenen Mittel für einen modularen Programmaufbau (siehe 7.1) verbessern weiterhin die Lesbarkeit von PASCAL-Programmen.
Die FORTRAN-Schreibweise von Befehlen ist an mathematische Konventionen angelehnt (FORmula TRANslation). Die Länge von maximal 6 Charactern für einen Bezeichner langt im allgemeinen für dessen Verständlichkeit aus. Marken in FORTRAN bestehen aus Ziffern, was eine Verringerung der Lesbarkeit bedeutet, da hierdurch leicht die Übersicht über einzelne Programmteile verloren gehen kann. Eine Verbesserung kann durch konsequente Ausnutzung der Unterprogrammtechnik erzielt werden.
IFTRAN beinhaltet zunächst alle Aussagen in Bezug auf FORTRAN. Zusätzlich wird die Lesbarkeit von IFTRAN-Programmen neben den besseren Strukturen vor allen Dingen durch das automatische Einrücken des Quelllistings erhöht. Das Markieren von Programmteilen ist im eigentlichen IFTRAN-Programm nicht möglich (keine Sprünge).
BASIC erlaubt nur Bezeichner, die maximal aus einem Buchstaben und einer Ziffer bestehen. Dazu kommt ein nur geringes Maß an Strukturierungsmitteln und der sequentielle Programmaufbau. Umfangreichere BASIC-Programme sind trotz der geringen Anzahl von möglichen Anweisungen ohne explizite Kommentare nur schwer verständlich.
DASMR-Programme sind ohne ausreichende Kommentare wertlos, da bei größeren Programmsequenzen sehr schnell

der Überblick verloren geht. Eine beschränkte Möglichkeit zur Erhöhung der Lesbarkeit ist durch die Konstantendefinition mit Hilfe der EQU-Anweisung gegeben.

7.8 Wartung

Eine einfach zu handhabende Programmwartung ist besonders in der Prozeßprogrammierung eine unerläßliche Forderung. Bei der Wartungsfreundlichkeit eines Programms spielen Eigenschaften wie Programmzuverlässigkeit und Programmlesbarkeit eine große Rolle. Die Bewertung der Sprachen hinsichtlich der Wartungsfreundlichkeit entspricht der Zusammenfassung der Bewertungen für Modularität, Zuverlässigkeit und Lesbarkeit.

Die Zusammenfassung aller hier aufgeführten Punkte steht in Anhang C.

8. BEWERTUNG DER SPRACHEN

Die in den vorausgegangenen Kapiteln gemachten Aussagen über die Sprachen sollen einer Bewertung unterzogen werden. Dazu ist es notwendig, daß ein geeigneter Bewertungsmaßstab an die gewonnenen Ergebnisse angelegt wird. Dieser Maßstab beinhaltet zugleich eine Wichtung der verschiedenen Kriterien, die dem Vergleich zugrunde liegen.
In diesem Abschnitt wird die Art des Bewertungsmaßstabes beschrieben, eine Begründung der einzelnen Wichtungen der Kriterien gegeben und zum Schluß eine Kommentierung der Endergebnisse vorgenommen.

8.1 Der Bewertungsmaßstab

Der Kriterienkatalog für den hier durchgeführten Sprachvergleich ist in zwei Klassen unterteilbar:

- Anforderungen an die Sprachen hinsichtlich der vorhandenen bzw. benötigten Sprachmittel (siehe 2.2 und 2.3).
- Anforderungen hinsichtlich der Anwendungen dieser Sprachmittel (siehe 2.4).

Für die Anforderungen der ersten Klasse wird eine Tabelle erstellt, aus der zunächst zu entnehmen ist, welche geforderten Sprachmittel vorhanden sind und welche nicht. Eine solche einfache Tabelle ist für die andere Klasse nicht ohne weiteres zu erstellen, da hier keine quantitativ erfaßbaren Aussagen gemacht werden können.
Um diese unterschiedlichen Aussagen miteinander vergleichen zu können, wird ein System erstellt, das die einzelnen Kriterien mit Punkten bewertet. Die Verteilung der Punkte ermöglicht zugleich eine Wichtung der verschiedenen Kriterien.
Dieses Punktesystem wird auf die Tabelle der Sprachmittel ausgeweitet, um diese Ergebnisse ebenfalls in

eine Gesamtbewertung mitaufzunehmen. Die Verteilung der Punkte ist nach folgendem Schema durchgeführt:

Für die Sprachmittel: Sprachmittel, welche den Anforderungen entsprechen, werden mit zwei Punkten bewertet, teilweise abgedeckte Anforderungen mit einem Punkt. Sprachmittel, die über die Anforderungen hinausgehen, erhalten drei Punkte.

Für die Anwendung der Sprachmittel: Gemäß der Wichtigkeit eines Kriteriums wird eine Punktzahl als Anforderung vorgegeben, die eine Sprache bei vollständiger Erfüllung erreichen kann. Abweichungen werden mit entsprechendem Punktabzug festgehalten.

Die ausführliche Bewertungstabelle ist dem Anhang C zu entnehmen.

8.2 Begründung der Wichtungen

Wie oben erwähnt wird bei den Anwendungskriterien eine Wichtung der einzelnen Kriterien durchgeführt. Bedingt durch die Anforderungen an eine Sprache bei der Prozeßprogrammierung werden die Kriterien wie folgt eingestuft (in Klammern die jeweils maximale Punktzahl):

1. Zuverlässigkeit, Sicherheit (3Ø)
2. Modularität (27)
3. Wartung, Dokumentation, Lesbarkeit (24)
4. Effizienz (21)
5. Implementierungsaufwand, Testhilfen (18)
6. Erlernbarkeit (15)

Die Forderung nach der Programmzuverlässigkeit muß an erster Stelle stehen. Wenn ein Programm ein vorgegebenes Problem nicht richtig löst, verlieren die anderen Kriterien ihre Wichtigkeit.

An zweiter Stelle ist hier die Forderung nach der Modularität aufgeführt, welche eine wesentliche Voraussetzung für strukturiertes Programmieren darstellt.

Modularität ist zugleich die Grundlage für die Lesbarkeit und für die Wartungsfreundlichkeit von Programmen. Im Vergleich zu den übrigen Kriterien haben Forderungen nach geringem Implementierungsaufwand, umfangreichen Testhilfen und einfache Erlernbarkeit ein geringeres Gewicht. Dies ist unter anderem bedingt durch den speziellen Aspekt dieses Sprachvergleichs. Ein Programm zur rechnergeführten Prozeßüberwachung ist dadurch gekennzeichnet, daß die Zeit, in der das Programm eingesetzt wird, sehr groß ist im Verhältnis zu der Zeit, die für die Implementierung und Tests benötigt wird.
Die Wartungsfreundlichkeit ist höher einzuschätzen als der Aufwand während der Programmentwicklung.
Dies ist auch der Grund, warum immer mehr problemorientierte Sprachen zum Einsatz kommen; ein Programm in Maschinensprache, das im allgemeinen ein besseres Verhalten zur Laufzeit zeigt, ist im Gegensatz zu Programmen in einer höheren Sprache nur sehr schwer zu warten und den jeweiligen Gegebenheiten anzupassen.
Die Effizienz eines Programms ist hier weniger wichtig eingestuft worden. Ein streng modular aufgebautes Programm, das mit hoher Wahrscheinlichkeit uneffizient arbeitet, kann leicht an den kritischen Stellen optimiert werden. Um wieviel schwerer ist ein Programm zu warten, das nur nach dem Gesichtspunkt der Effizienz erstellt wurde! Darüberhinaus muß man sich unter Berücksichtigung der Ergebnisse von /1/ (Rechenzeit = 9Ø % Betriebssystemzeit + 1Ø % Programmrechenzeit) fragen, ob sich der Aufwand überhaupt lohnt, Programmrechenzeiten zu optimieren.
Im Gegensatz zu den Anwendungskriterien sind die Sprachmittel nicht gewichtet. Jede vorhandene Eigenschaft wird mit der gleichen Punktzahl bewertet.
Eine Verschiebung der Wichtungen ergibt sich allein dadurch, daß die Kriterien in unterschiedlich große

Untermengen aufgestellt sind. Durch diese Zerlegung werden die Kriterien ABLAUFSTEUERUNG und EIN-/AUSGABE stärker bewertet als die übrigen Sprachmittel. Dieser Umstand ist jedoch beabsichtigt, da diese beiden Kriterien einen großen Einfluß auf die Programmierung ausüben.

8.3 Ergebnis des Sprachvergleichs

Nach Addition der verteilten Punkte (siehe Anhang C) ergibt sich folgendes Ergebnis:

	PASCAL	IFTRAN	FORTRAN	DASMR	BASIC	Maximale Punktzahl
Sprachmittel	56	52	48	43	29	63
Anwendung	137	121	1Ø4	85	8Ø	177
Summe	193	173	152	128	1Ø9	24Ø

Zu diesem Ergebnis sind einige Anmerkungen zu machen:
Die augenblicklich verfügbare PASCAL-Version (Sequential PASCAL /16/) erlaubt kein Multitasking, das ein wesentliches Kriterium in der Prozeßprogrammierung darstellt. Aus diesem Grund muß im Augenblick vom Einsatz von PASCAL so lange abgeraten werden, bis eine Version zur Verfügung steht, die parallele Aktivitäten im vollen Umfang ermöglicht. Dann allerdings verspricht die Verwendung von PASCAL gegenüber den übrigen untersuchten Sprachen die besten Lösungsmöglichkeiten von Aufgaben in der Prozeßautomatisierung.
IFTRAN bietet sich als nächste Sprache an, solange PASCAL nicht im vollen gewünschten Umfang zur Verfügung steht.
FORTRAN rangiert in dieser Tabelle erst an dritter Stelle. Vor allem durch das bessere Abschneiden der FORTRAN-Erweiterung IFTRAN muß der Einsatz von FORTRAN über die Prozeßprogrammierung hinaus in Frage gestellt werden.

Durch die große Verbreitung von FORTRAN auf kleineren Rechnern kann diese Sprache vordergründig nicht übergangen werden. Jeder FORTRAN-Compiler ermöglicht jedoch den Einsatz eines in FORTRAN geschriebenen IFTRAN-Vorübersetzers.
Trotz der schlechten Plazierung von DASMR in diesem Vergleich hat der ASSEMBLER in der Prozeßprogrammierung eine große Bedeutung. In diesem Bereich der Datenverarbeitung treten immer wieder Probleme auf, die nur mit Hilfe des direkten Ansprechens der Hardware gelöst werden können. Allerdings sollte der Umfang der ASSEMBLER-Programmierung eben auf diesen speziellen Aufgabentyp beschränkt bleiben.
Das auf der V75 verfügbare BASIC ist in dieser Version nicht für die Prozeßprogrammierung geeignet. Dies kommt unter anderem dadurch zum Ausdruck, daß in BASIC nur eine exemplarische Realisierung der gestellten Aufgabe möglich war.

9. KRITISCHE ANMERKUNGEN ZU DEM SPRACHVERGLEICH

An dieser Stelle soll der in dieser Arbeit durchgeführte Sprachvergleich einer kritischen Betrachtung unterzogen werden.
Die meisten Ergebnisse sind durch den praktischen Teil des Vergleichs, der Mehrfachimplementierung eines Überwachungssystems, abgedeckt. Ein Vergleich mehrerer Sprachen kann nur dann einigermaßen objektiv sein, wenn die dazu benötigten Einzelschritte von verschiedenen Personen durchgeführt werden, die einen annähernd gleichen Wissensstand haben. Das bedeutet für diese Arbeit im einzelnen:
Der SADT-Entwurf, die Tasksynchronisation, das Erweitern der Laufzeitsysteme wird jeweils von verschiedenen Personen durchgeführt, ebenso wie das eigentliche Programmieren der gestellten Aufgabe.
Die Verteilung der einzelnen Teilaufgaben war nicht möglich. Das bedeutet, daß die verschiedenen Programmsysteme sequentiell entwickelt und implementiert wurden. Bei dieser Vorgehensweise ist es unmöglich zu verhindern, daß einmal gemachte Erfahrungen nicht auf die nächsten Sprachen angewendet werden. Deshalb sind vor allem die Werte aus der Implementierungsphase (Kapitel 6) dahingehend zu betrachten, daß bei fortschreitender Arbeit ein gewisser Lerneffekt eingetreten ist. Somit fallen die Zahlen bei den später untersuchten Sprachen im allgemeinen günstiger aus als bei den ersten Implementierungen.
Bei der Implementierung der Programme wurde darauf verzichtet, Programmteile zu optimieren (Grund siehe 8.2), um somit bessere Zeiten zu erzielen. Da dies allerdings bei allen Sprachen der Fall ist, tritt dadurch keine Verzerrung der Ergebnisse ein.
Ein weiterer Punkt, welcher die Ergebnisse beeinflußt, ist der Bewertungsmaßstab selbst, vor allem die Art

der Wichtung der Kriterien. Eine andere Prioritätenreihenfolge (siehe 8.2) bei den qualitativen Kriterien oder eine andere Bewertung der Sprachmittel kann eine ganz andere Aussage über die Sprachen liefern. Die hier gewonnenen Werte, zusammen mit den Aussagen von /1/, sind etwas unvollständig, da exakte Werte über die Reaktionszeiten auf Alarme nicht ermittelt werden konnten. Vielmehr war nur eine erste Näherung für diese Zeiten anzugeben (siehe 4.4).
Diese mehr subjektiven Aspekte können dennoch vernachlässigt werden. Durch die Berücksichtigung der verschiedenartigsten Kriterien, die eine im großen und ganzen komplette Anforderungsliste an die Prozeßprogrammierung darstellen, ist eine objektive Aussage über die Sprachen in weiten Bereichen möglich.
Somit kann diese Arbeit, zusammen mit /1/, als Grundlage dienen für weiterführende Diskussionen über die Auswahl von Sprachen für die Prozeßprogrammierung.

Literaturliste

/1/ Uhlig, C.
Untersuchungen zum Realzeitverhalten von Sprachen zur Prozeßprogrammierung mit Hilfe eines Laufzeitmeßsystems
KFK 2715 B

/2/ Varian Data Machines
VORTEX II Reference Manual
Bestell-Nr. 98 A 9952 244

/3/ Varian Data Machines
Varian 74 System Handbook
Bestell Nr. 98 A 99Ø6 21Ø

Varian 75 System Supplement
Bestell-Nr. 98 A 99Ø6 22Ø

/4/ Varian Data Machines
Varian Assembly Reference Manual
Bestell-Nr. 98 A 9952 45Ø

/5/ Varian Data Machines
V7Ø/62Ø BASIC Language
Bestell-Nr. 98 A 9952-Ø33

/6/ Varian Data Machines
FORTRAN IV Reference Manual
Bestell-Nr. 98 A 9952 Ø4Ø

/7/ Borrmann
Programmsystem zur zentralen Prozeßkommunikation über ein MOPEK-Interface im Rahmen der VORTEX-Organisation
KFK-EXT 13/77-1

/8/ Borrmann
Wirkungsweise und Ansprechmöglichkeiten des Prozeßinterface MOPEK/V73
(1977) unveröffentlicht

/9/ Lauber
Experimentelle Untersuchungen von Spracheigenschaften der Prozeßrechnersprache PEARL anhand von Modellprozessen
in: Krönig, D.
Praxis von Sprachen, Programmiersystemen und Programmgeneratoren
1. Treffen 76 des German Chapter of the ACM
Applied Computer Science 3
Carl Hanser Verlag 1976

/1Ø/ Haase
Vergleichende Übersicht über Anwendungen und
Implementationen von Real-Time-BASIC
in: Krönig, D.
siehe /9/

/11/ Geiger
Benutzerhandbuch für IFTRAN-1
(1976) unveröffentlicht

/12/ General Research Corporation
IFTRAN:
Structured Programming Preprocessors for FORTRAN
Santa Barbara, California

/13/ Raab, B.
IFCOM:
Eine auf der VARIAN V75 arbeitsfähige
Version von IFTRAN-3
(1977) unveröffentlicht

/14/ Kapp
Das PASCAL-System der VARIAN V75
(1977) unveröffentlicht

/15/ Varian Data Machines
PASCAL Reference Manual
Bestell-Nr. 81 A Ø411-ØØ1 A

/16/ Wirth, N.
An Assessment of the Programming Language PASCAL
IEEE-Transaction of the Software Engineering
Vol Se-1, Nr.2, p. 192-198

/17/ Bauer, Goos
Einführung in die Informatik
Heidelberger Taschenbücher, Bände 8Ø/91
Springer-Verlag, Berlin-Heidelberg-New York 1971

/18/ Vaeth, Kuhn, Gütle, Radek, Groll, Baumgärtel
Differenzdichte:
Ein In-Line-Meßgerät zur Bestimmung der
Urankonzentration in organischen Phasen
KFK 2323

/19/ Softech
An Introduction to SADT
Structured Analysis and Design Technique
Software Technology Company, Report 9Ø22/78R

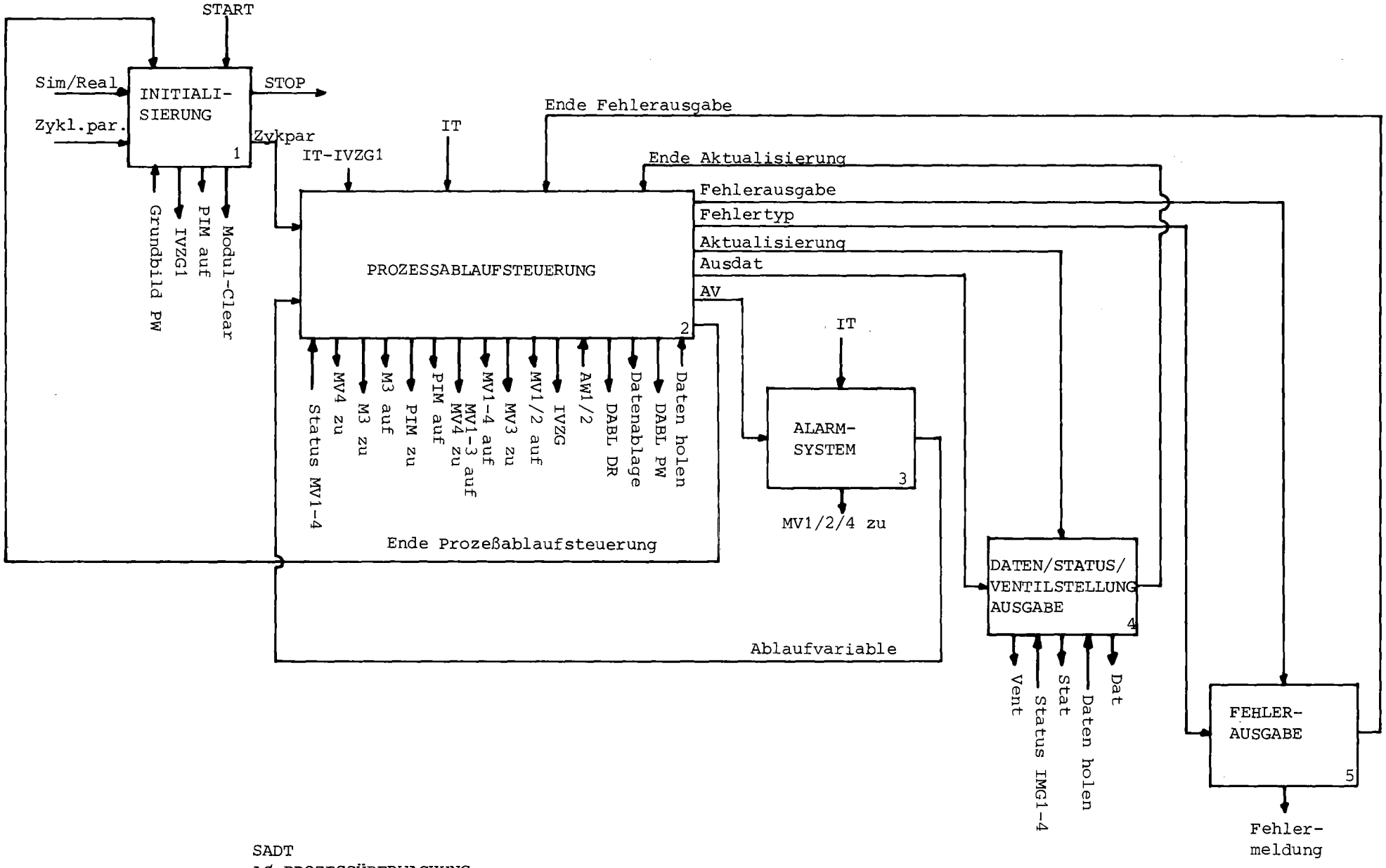
/2Ø/ Ross, D.T., Schoman, K.E., jr.
Structured Analysis for Requirements Definition
Softech, Inc.
Waltham, Massachusetts

A N H A N G A

SADT-Diagramme

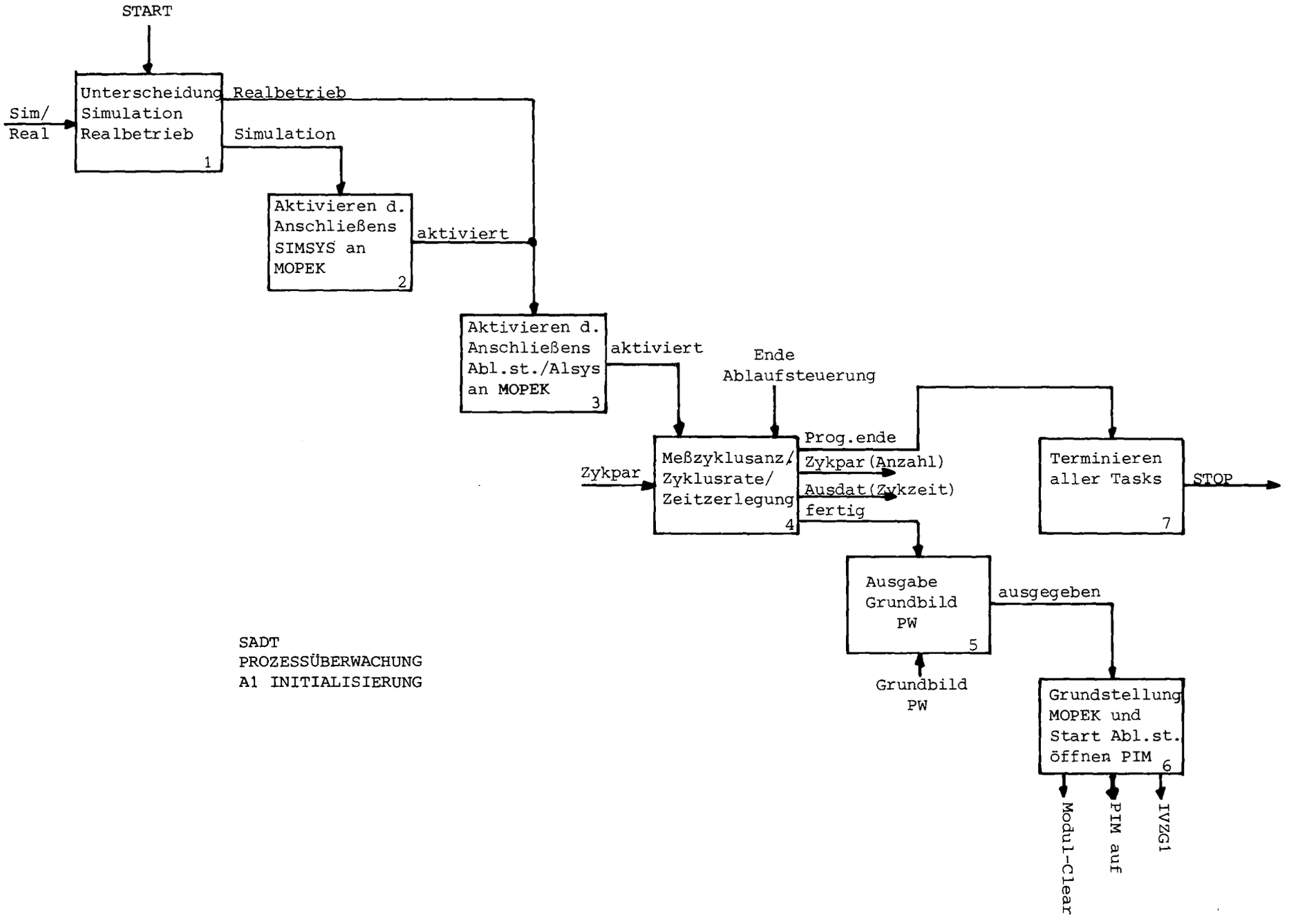
A.1 Prozeßüberwachung

A.2 Simulationssoftware

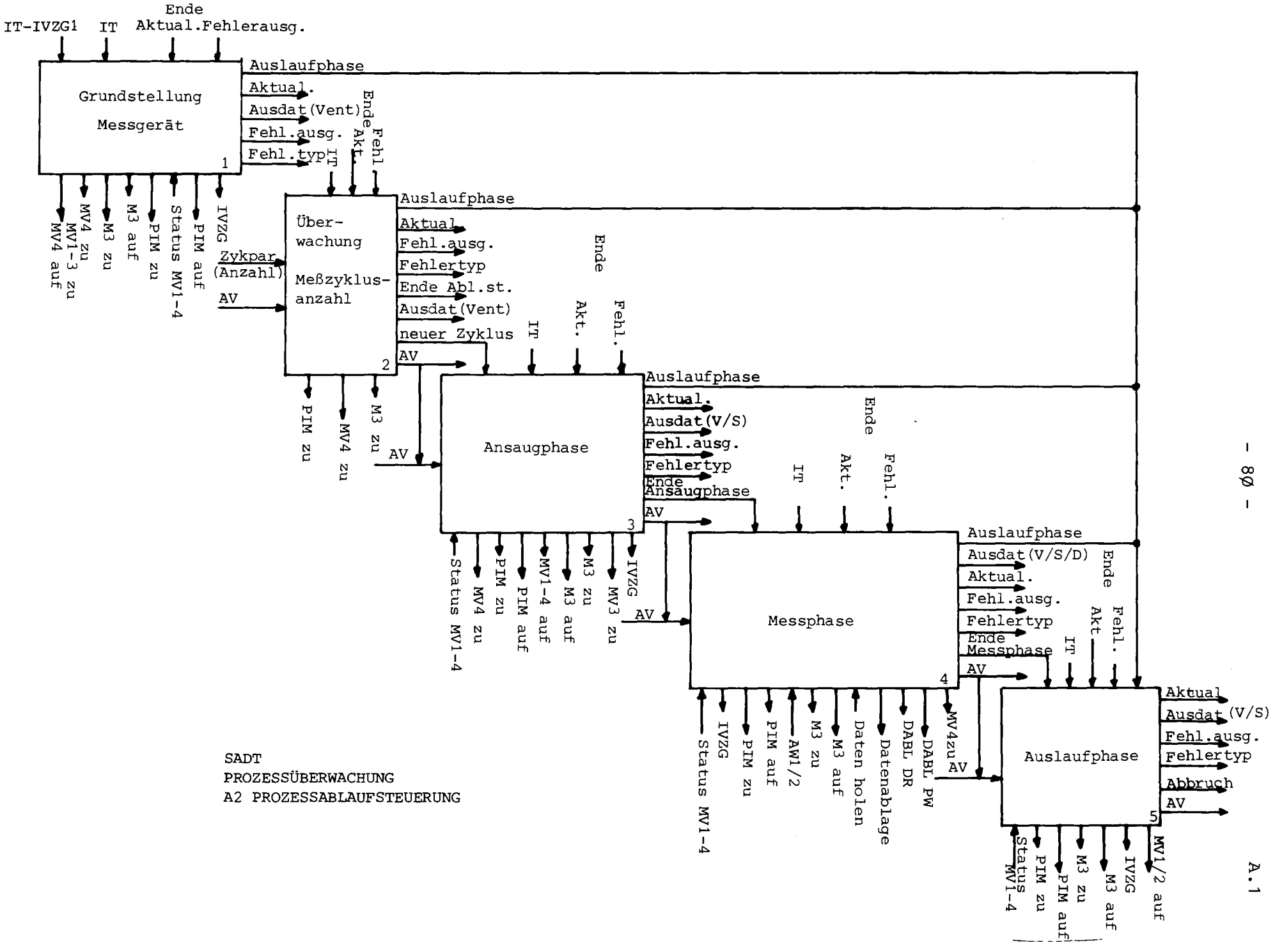


SADT
AØ PROZESSÜBERWACHUNG

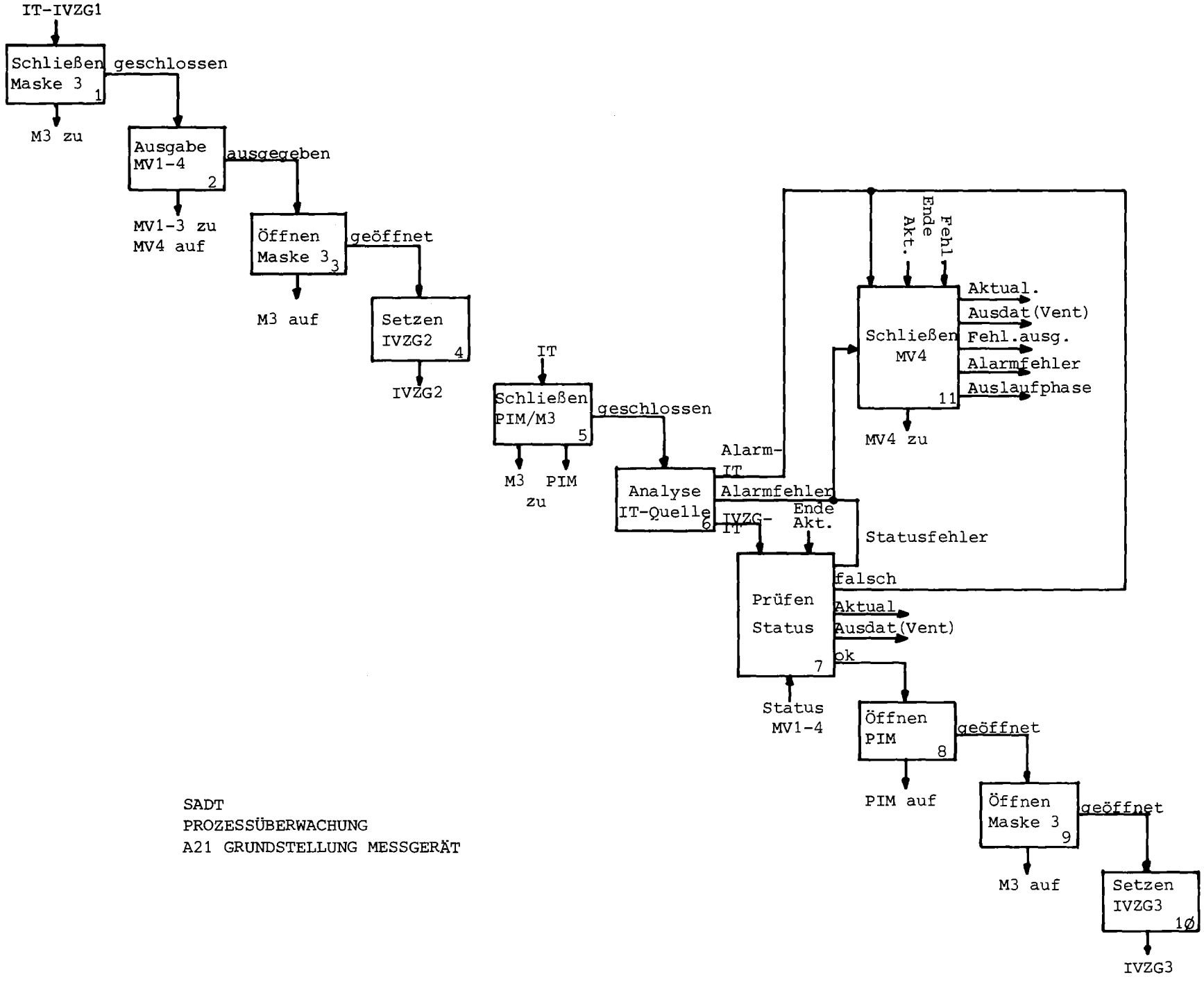
A.1



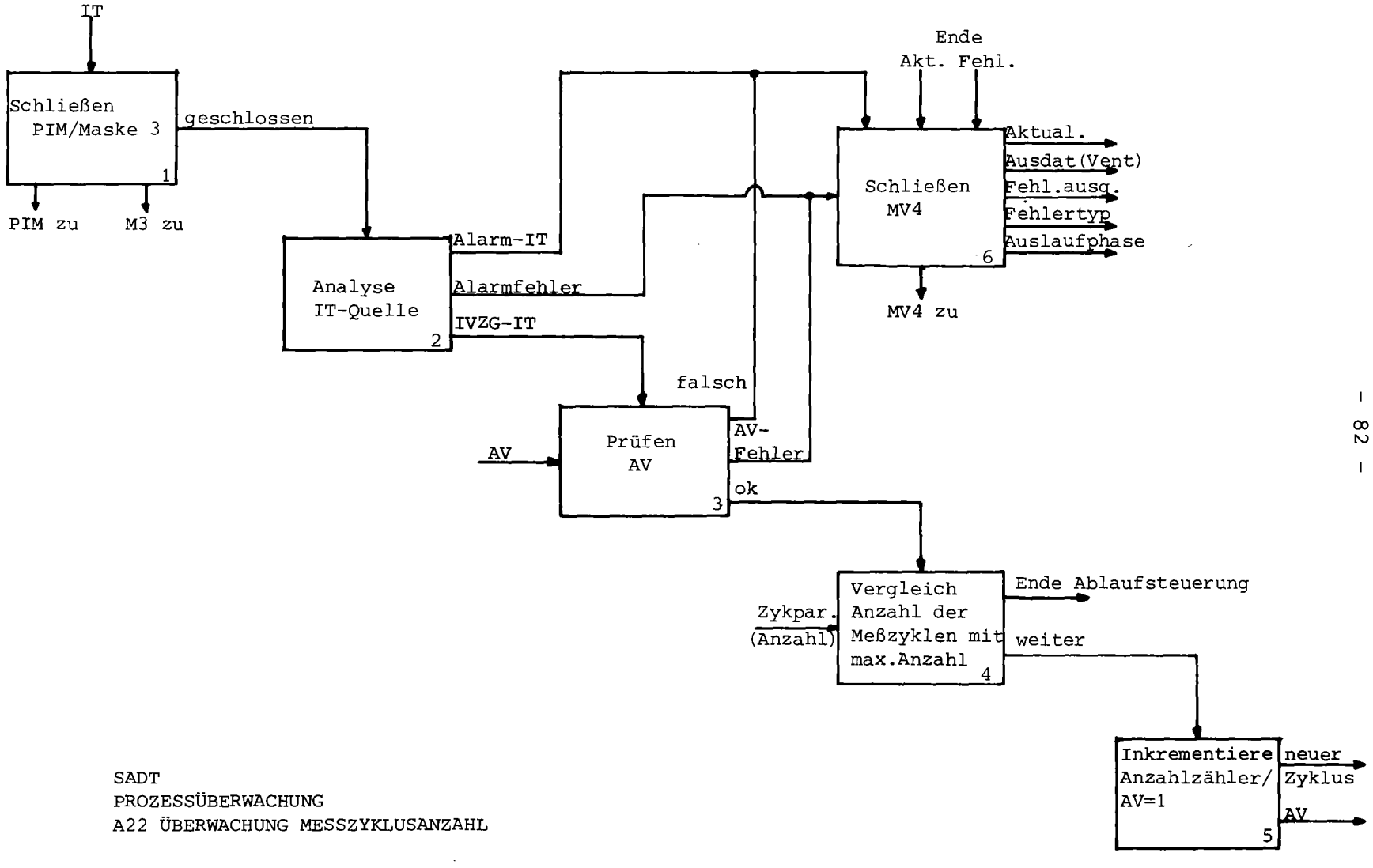
SADT
PROZESSÜBERWACHUNG
A1 INITIALISIERUNG



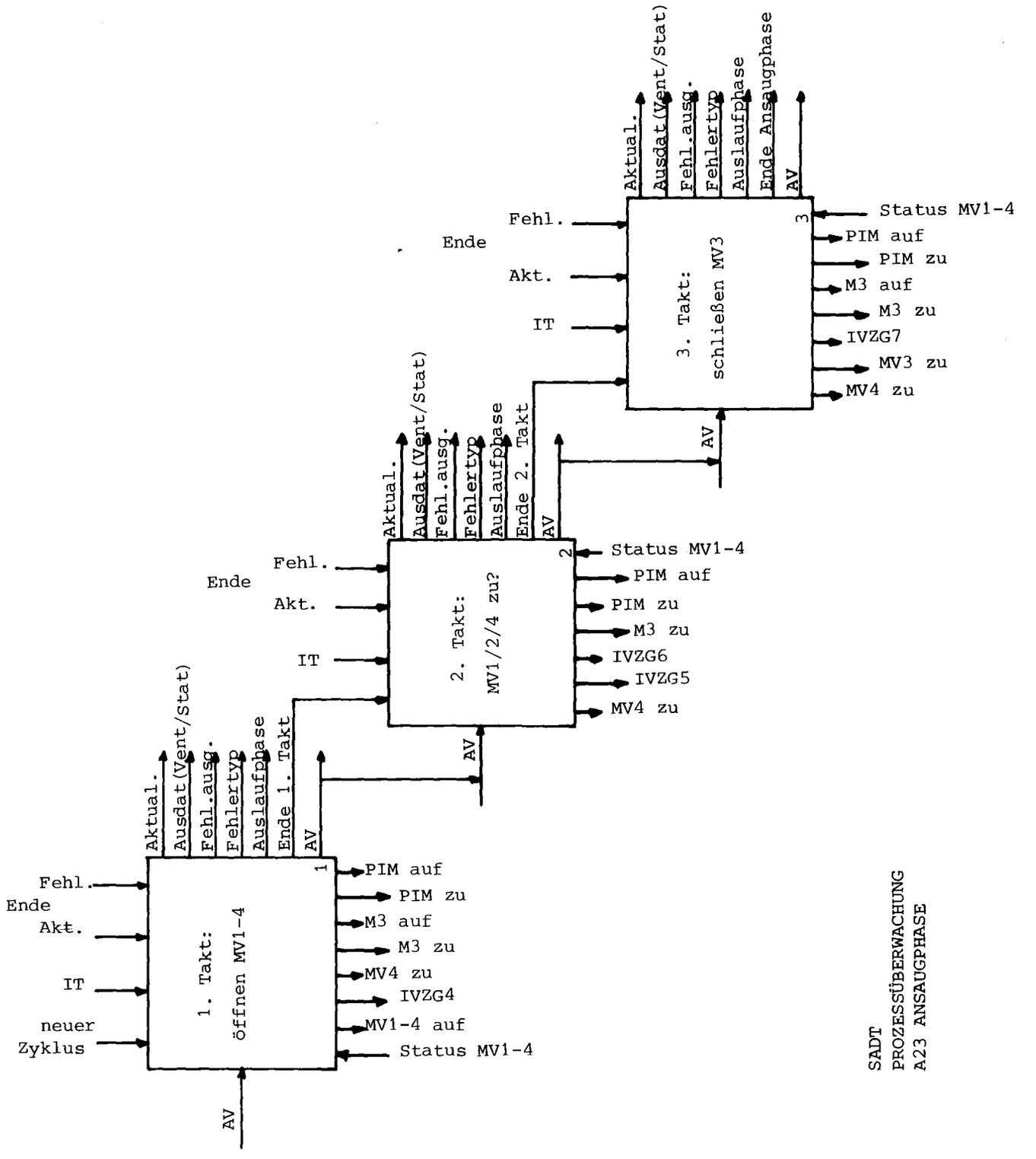
SADT
PROZESSÜBERWACHUNG
A2 PROZESSABLAUFSTEUERUNG



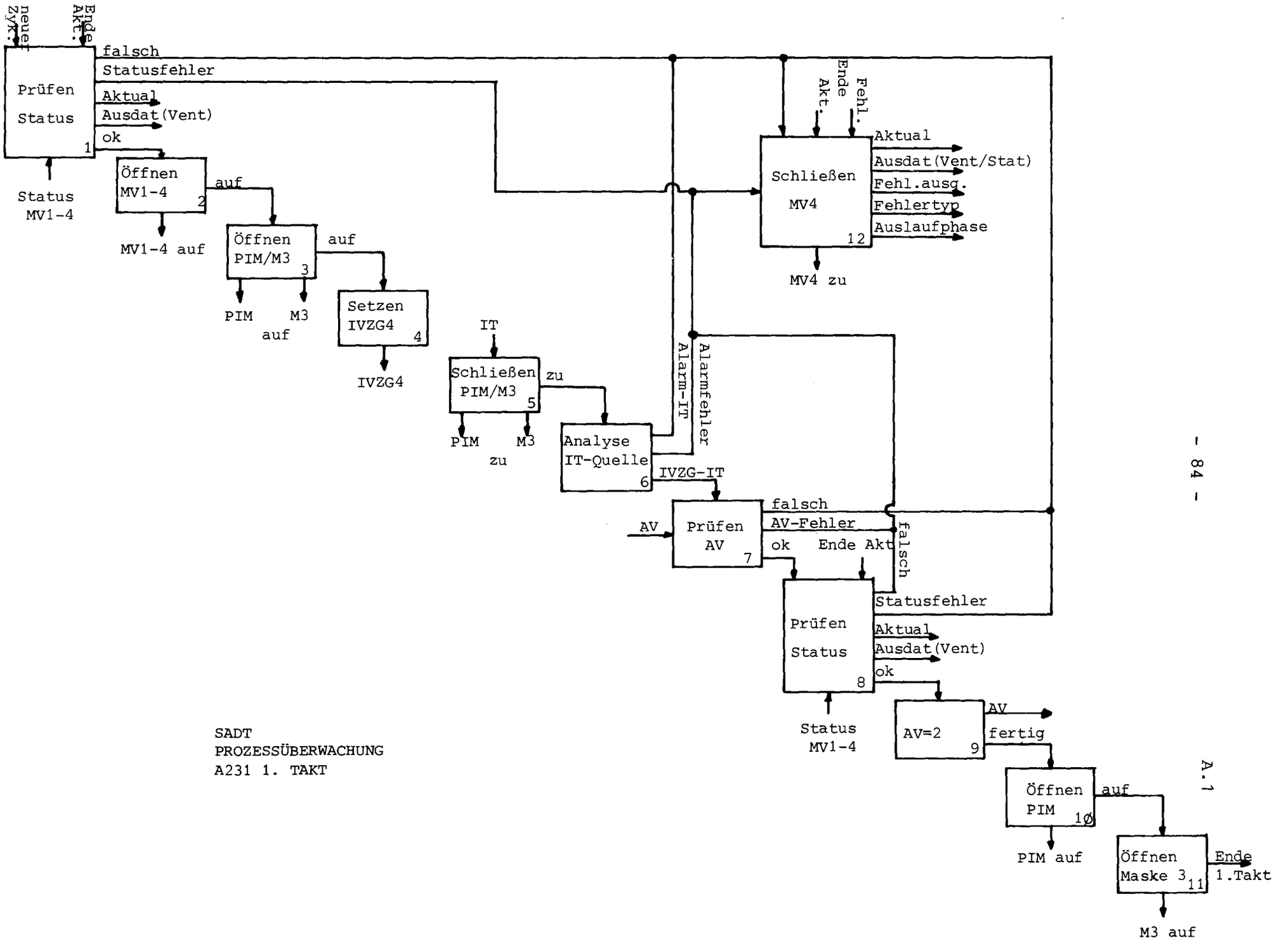
SADT
PROZESSÜBERWACHUNG
A21 GRUNDSTELLUNG MESSGERÄT



SADT
PROZESSÜBERWACHUNG
A22 ÜBERWACHUNG MESSZYKLUSANZAHL

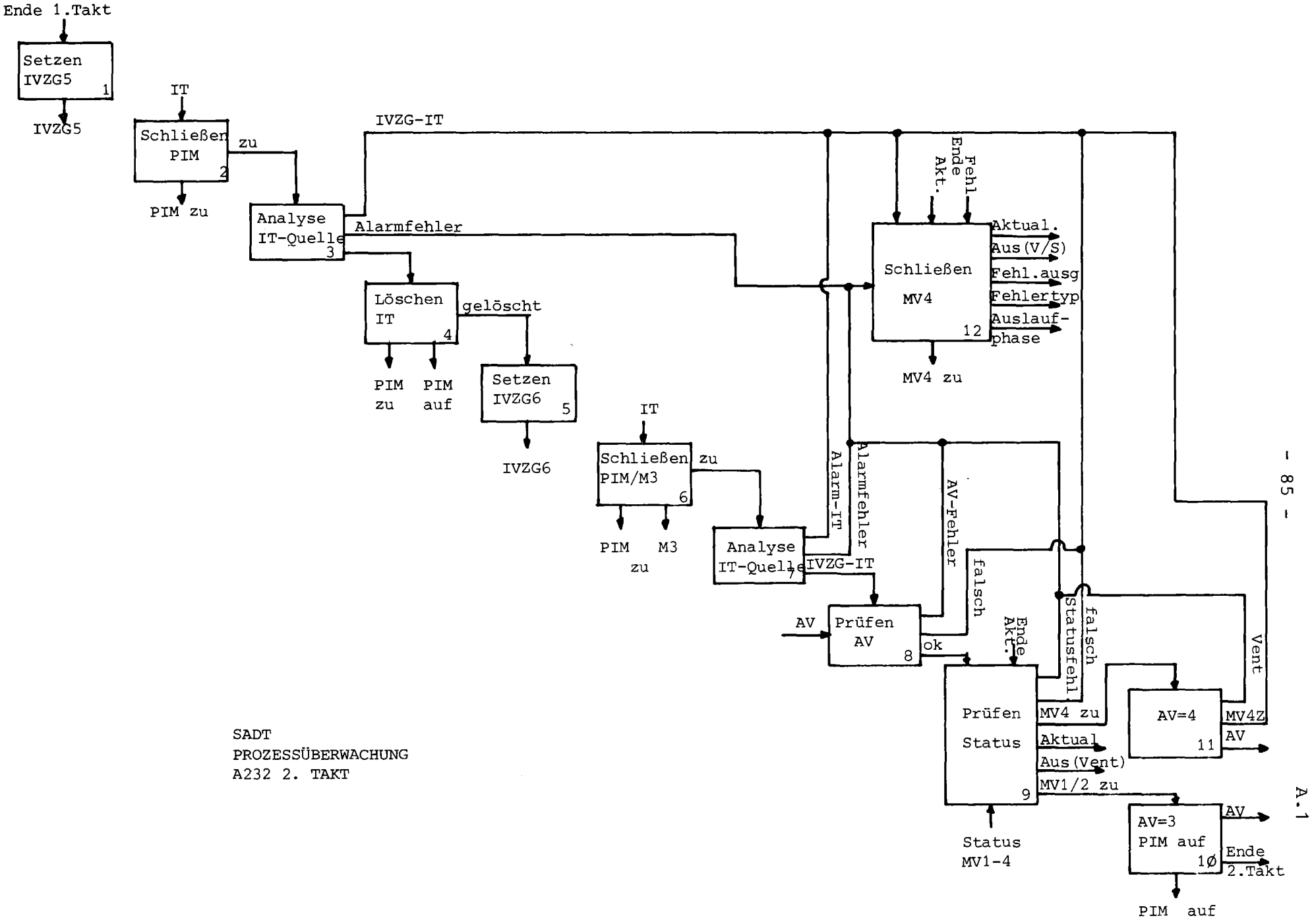


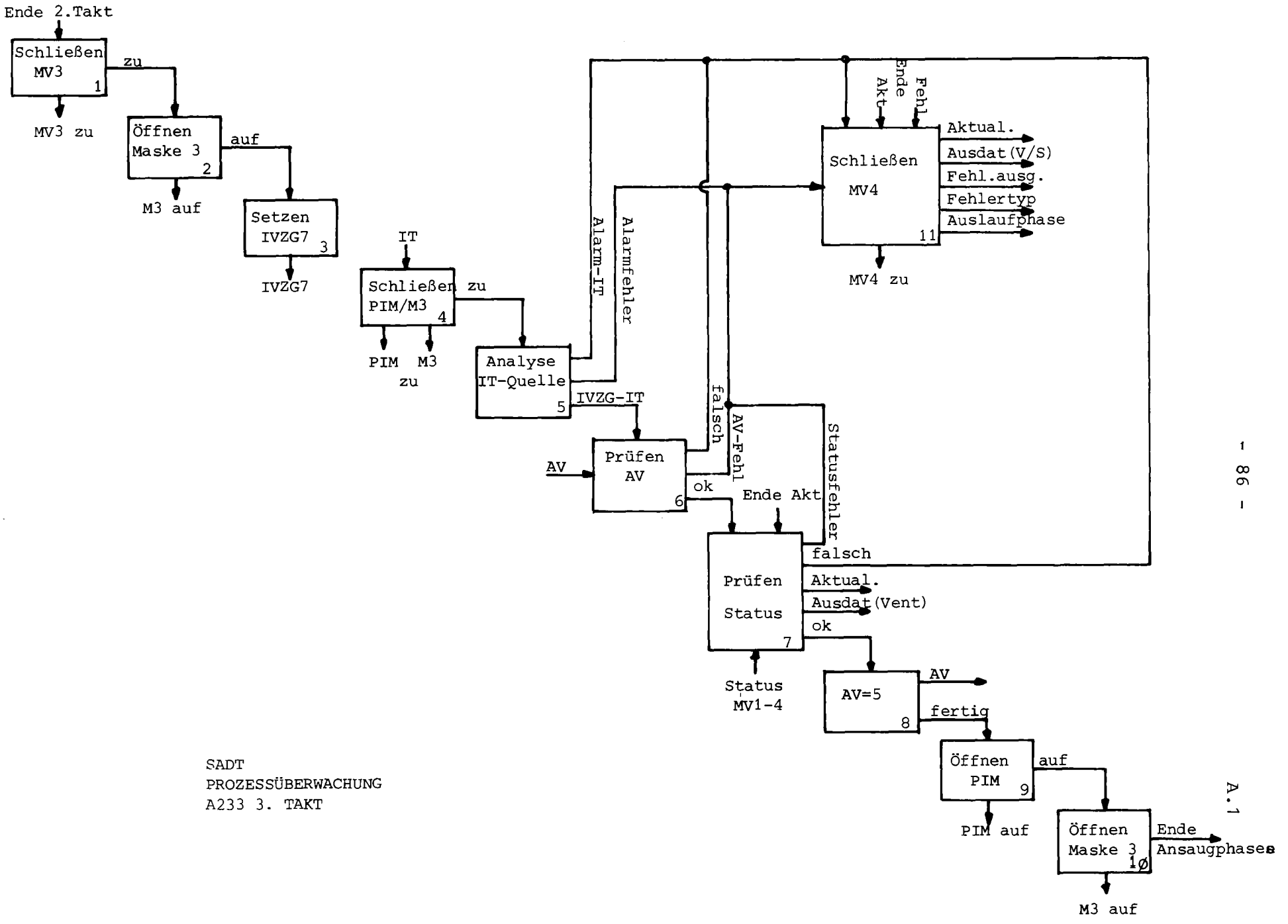
SADT
PROZESSÜBERWACHUNG
A23 ANSAUGPHASE



SADT
PROZESSÜBERWACHUNG
A231 1. TAKT

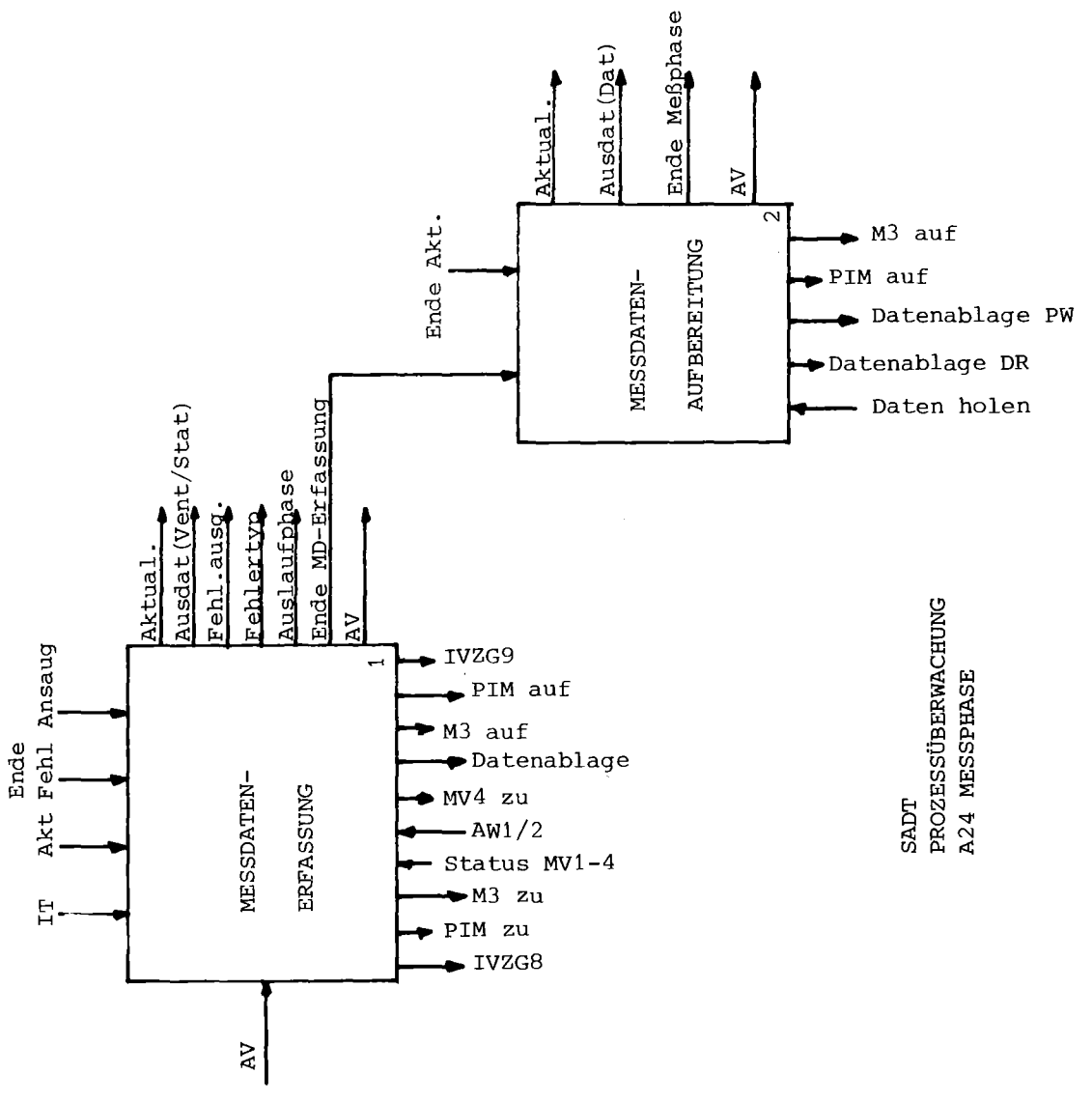
A.1


Ende 1.Takt
Setzen IVZG5 1
IVZG5
IT
Schließen PIM 2
zu
PIM zu
IVZG-IT
Analyse IT-Quelle 3
Alarmfehler
Löschen IT 4
gelöscht
PIM zu
PIM auf
Setzen IVZG6 5
IVZG6
IT
Schließen PIM/M3 6
zu
PIM zu
M3
Analyse IT-Quelle 7
Alarm-IT
Alarmfehler
IVZG-IT
AV
Prüfen AV 8
ok
AV-Fehler
falsch
Ende Akt.
Fehl.
Ende Akt.
Schließen MV4 12
Aktual.
Aus(V/S)
Fehl.ausg
Fehlertyp
Auslauf-phase
MV4 zu
Statusfehl
falsch
Vent
Prüfen Status 9
MV4 zu
Aktual
Aus(Vent)
MV1/2 zu
Status MV1-4
AV=4 11
MV4Z
AV
AV=3 PIM auf 1Ø
AV
Ende 2.Takt
PIM auf
SADT
PROZESSÜBERWACHUNG
A232 2. TAKT

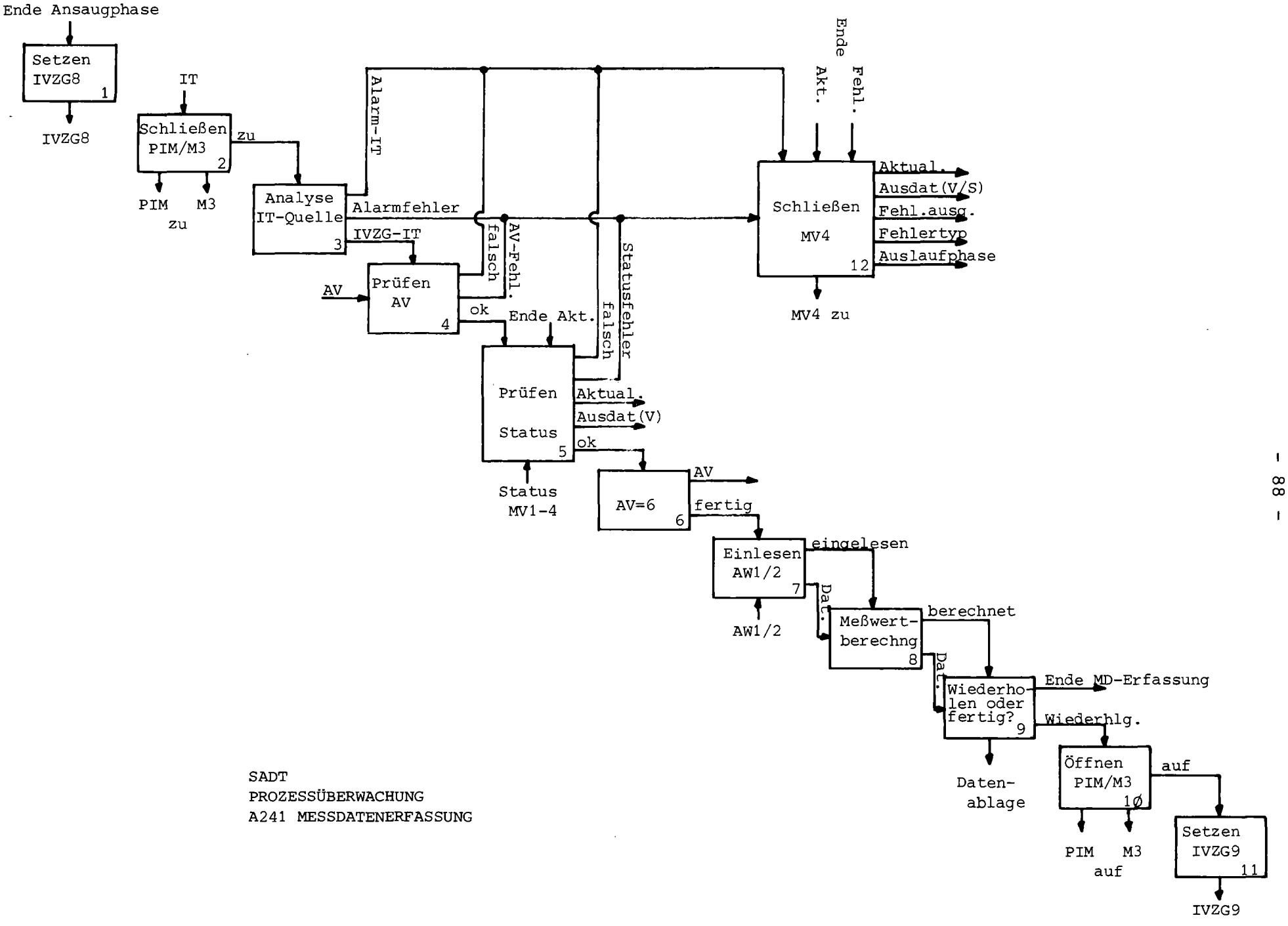


SADT
PROZESSÜBERWACHUNG
A233 3. TAKT

A.1

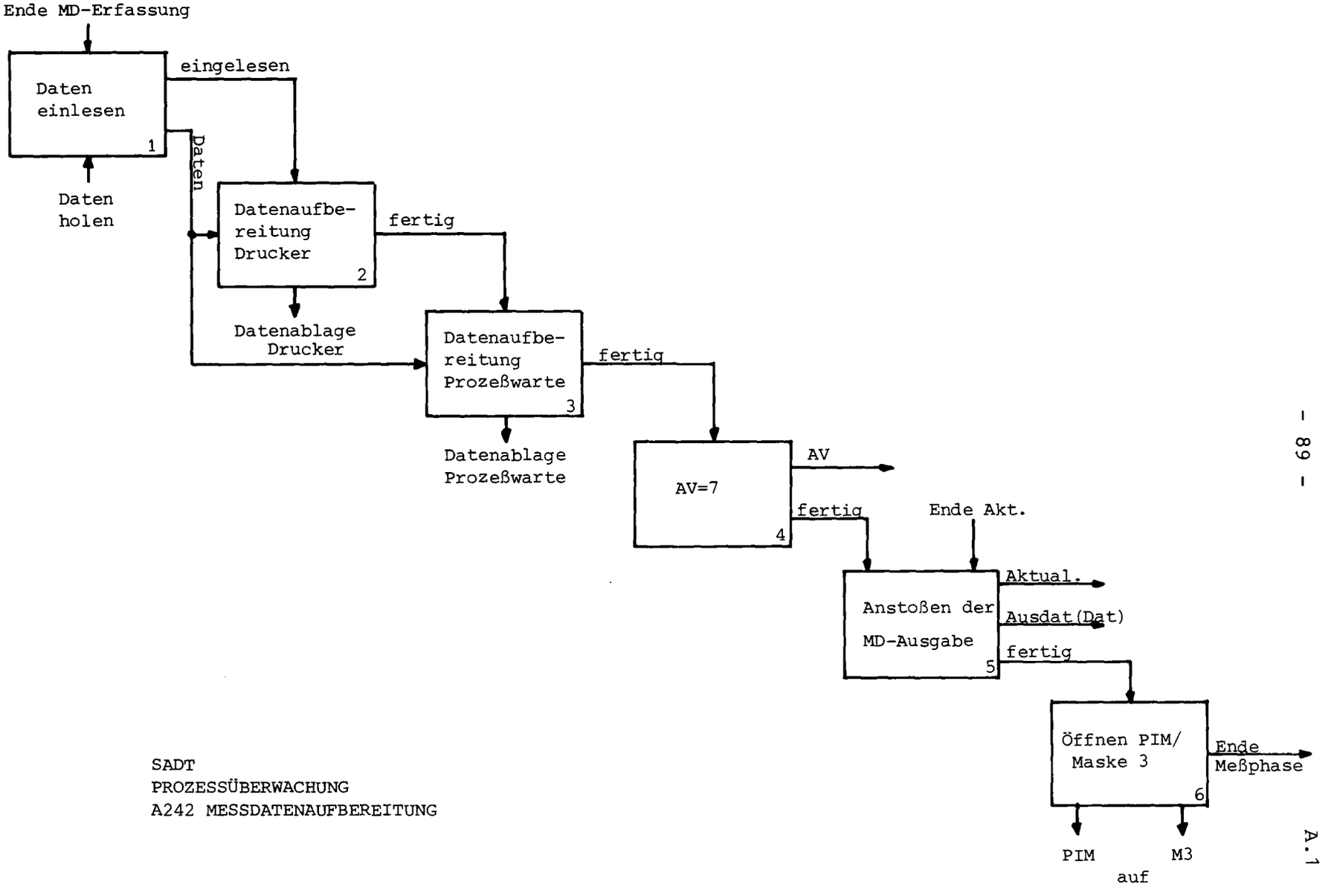


SADT
PROZESSÜBERWACHUNG
A24 MESSPHASE



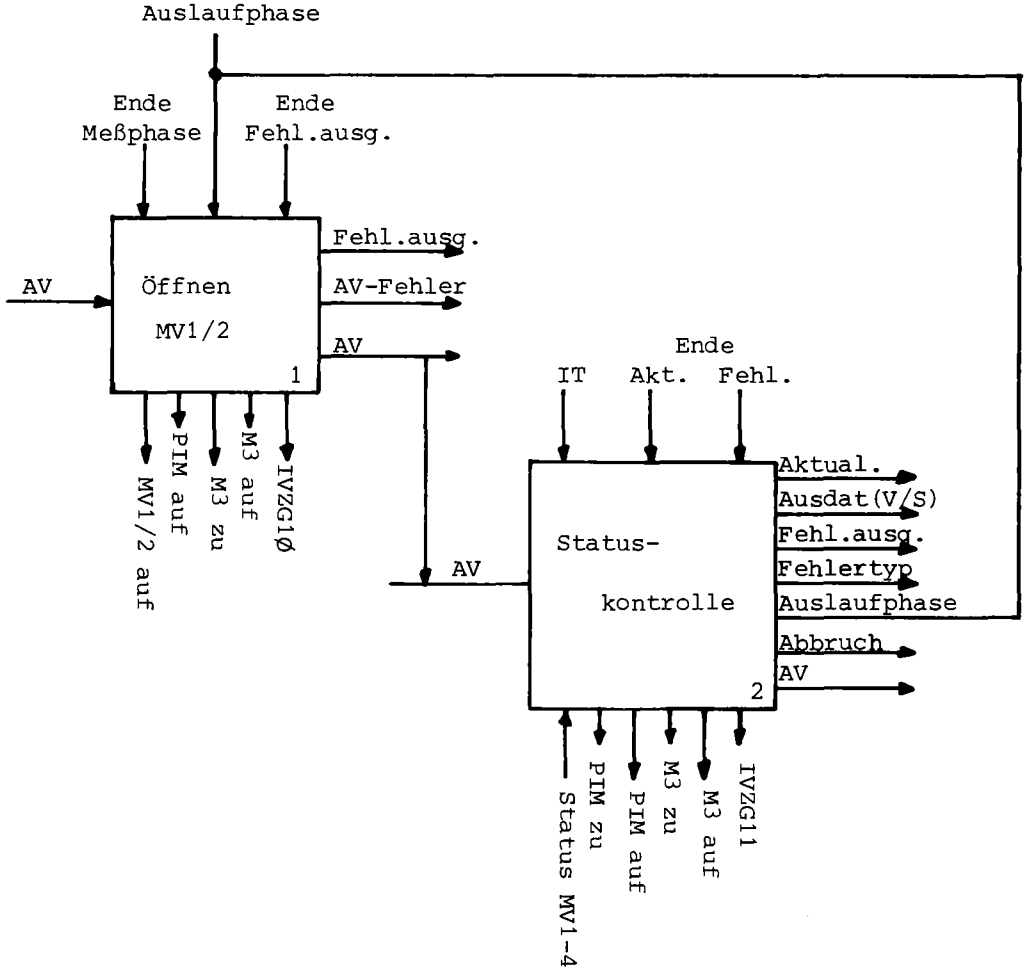
SADT
PROZESSÜBERWACHUNG
A241 MESSDATENERFASSUNG

A.1

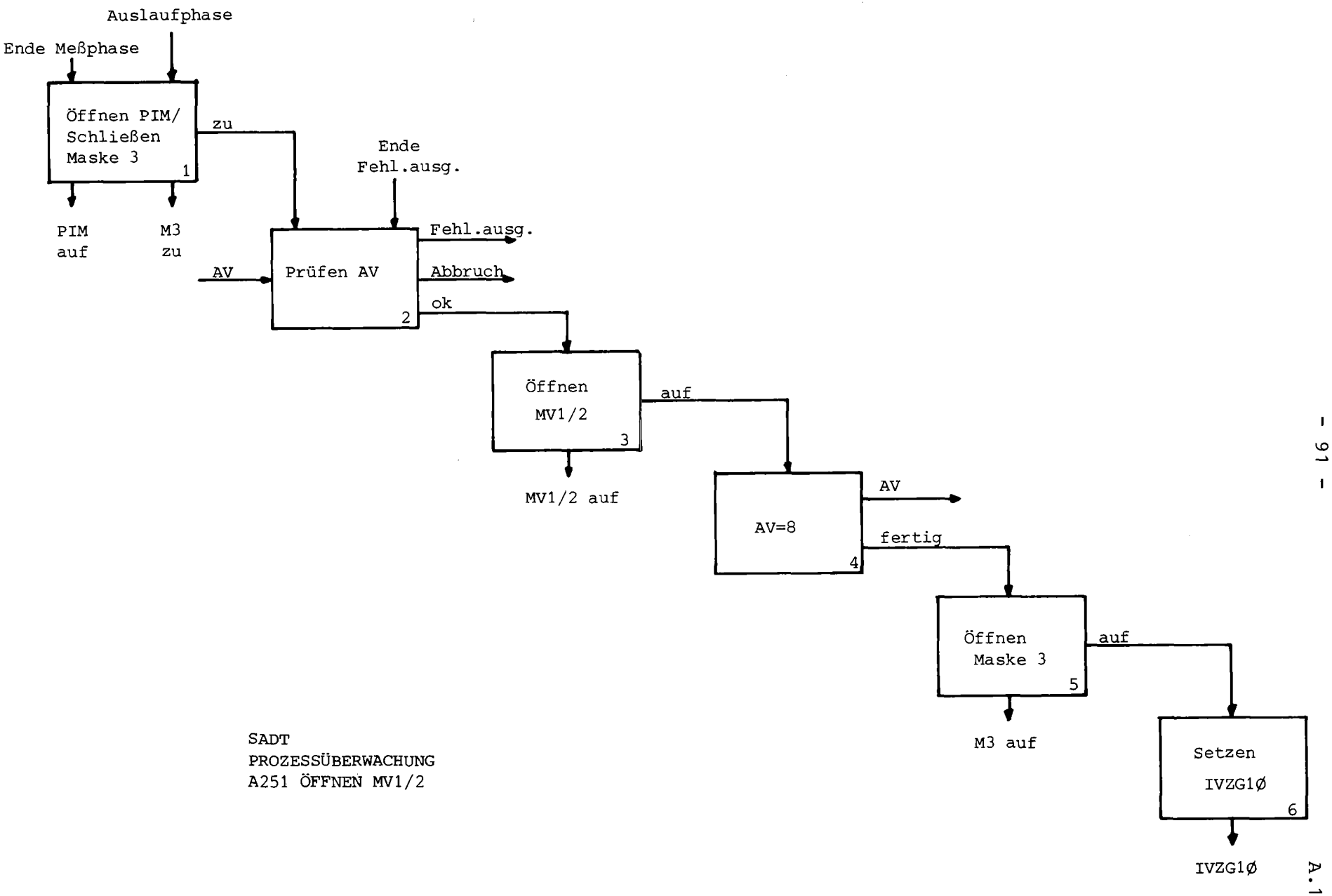


SADT
PROZESSÜBERWACHUNG
A242 MESSDATENAUFBEREITUNG

A.1

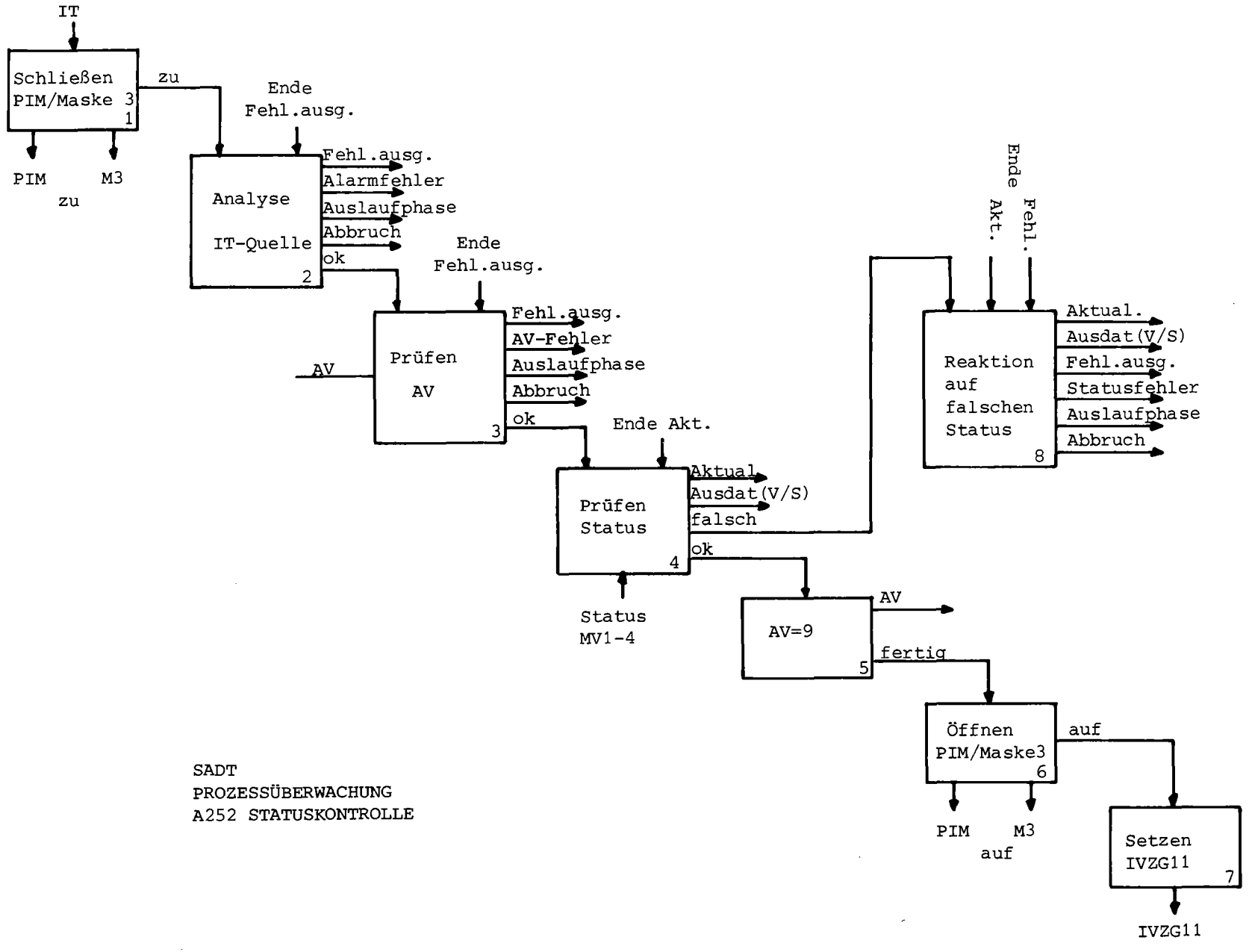


SADT
PROZESSÜBERWACHUNG
A25 AUSLAUFPHASE

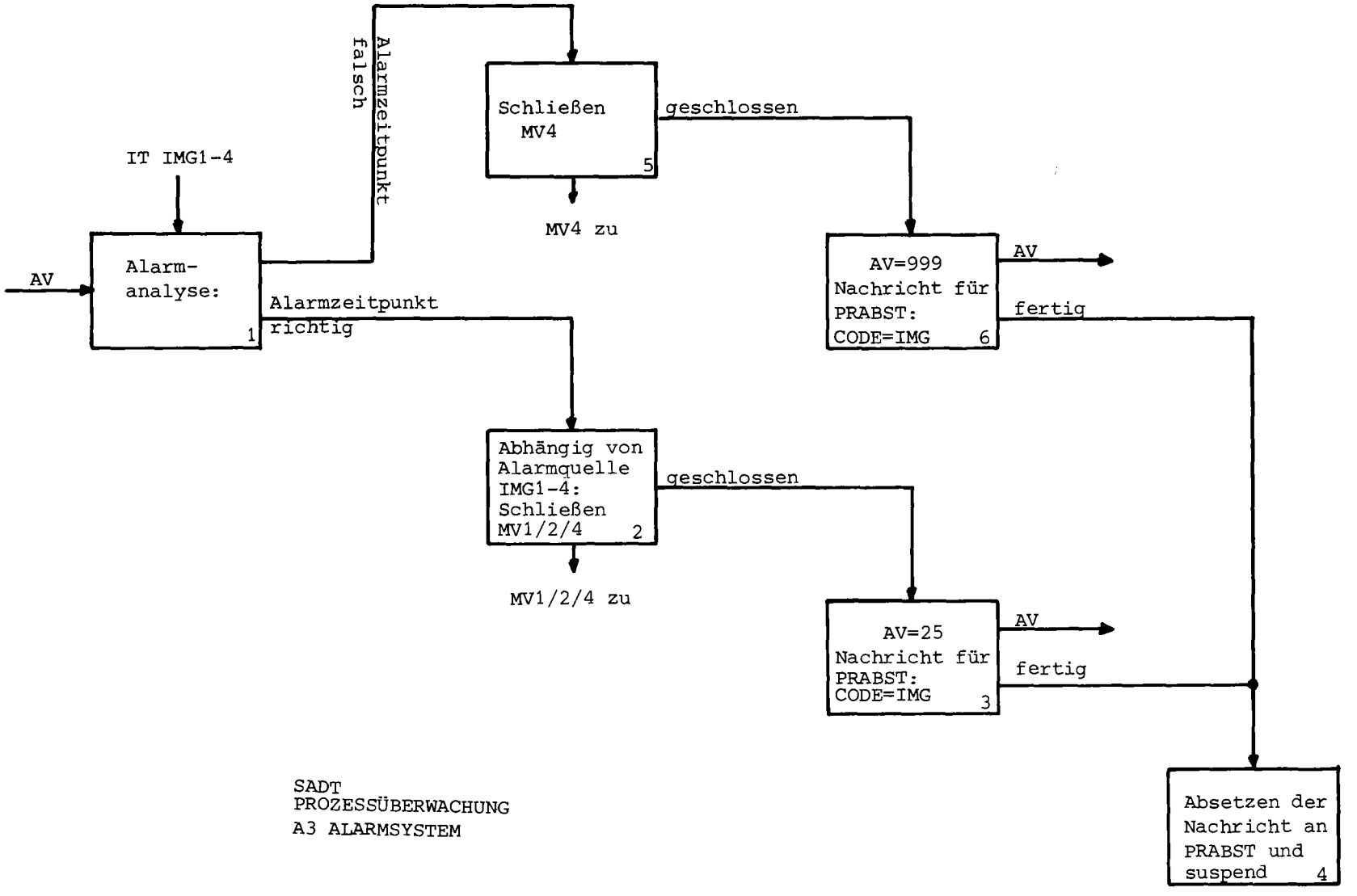


SADT
PROZESSÜBERWACHUNG
A251 ÖFFNEN MV1/2

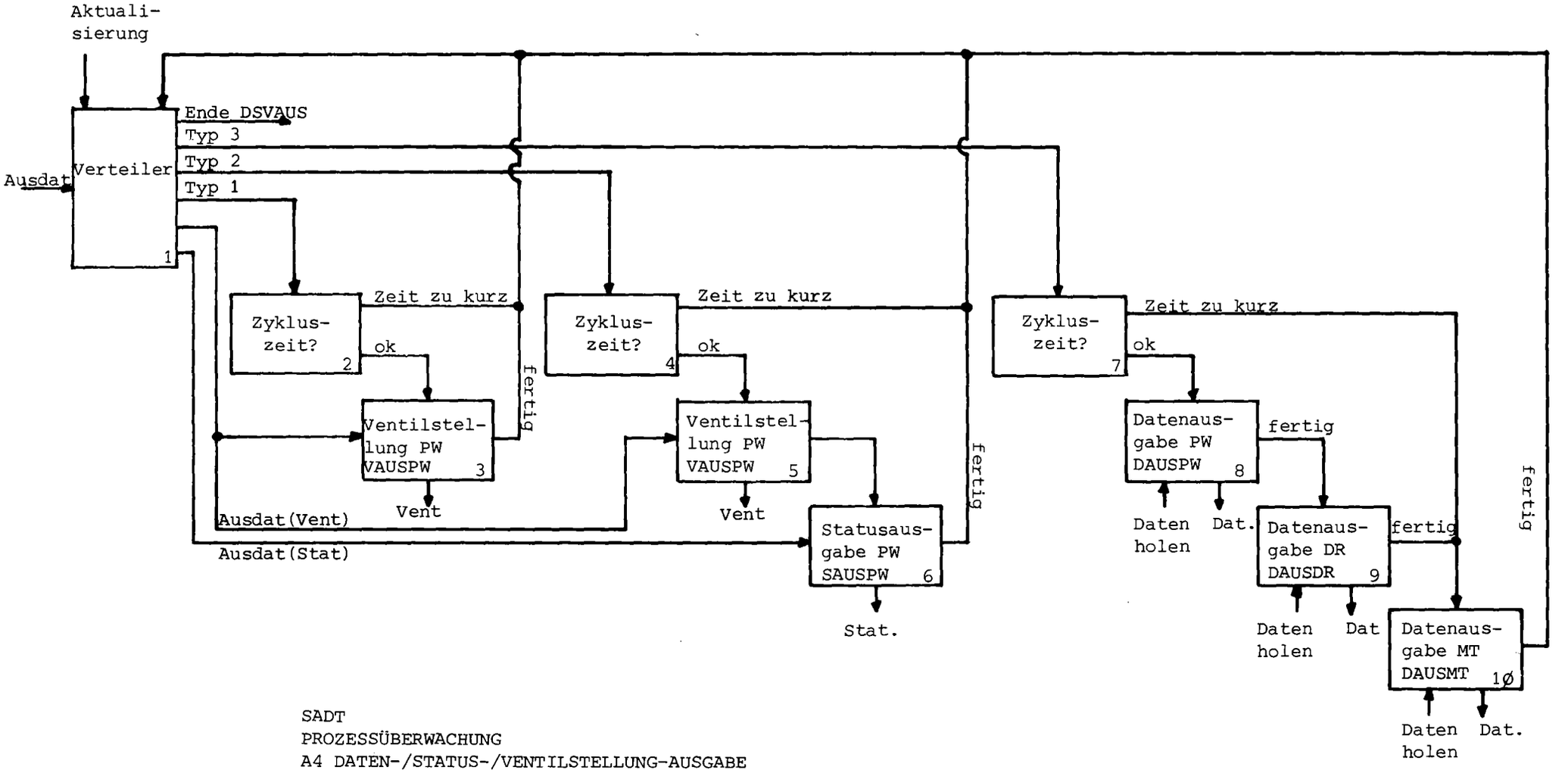
A.1



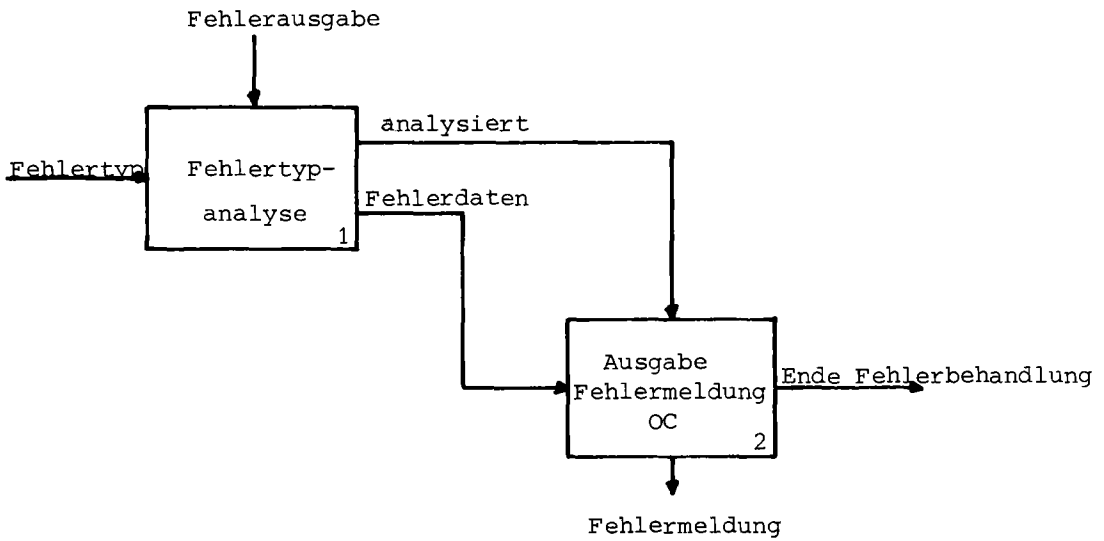
SADT
PROZESSÜBERWACHUNG
A252 STATUSKONTROLLE



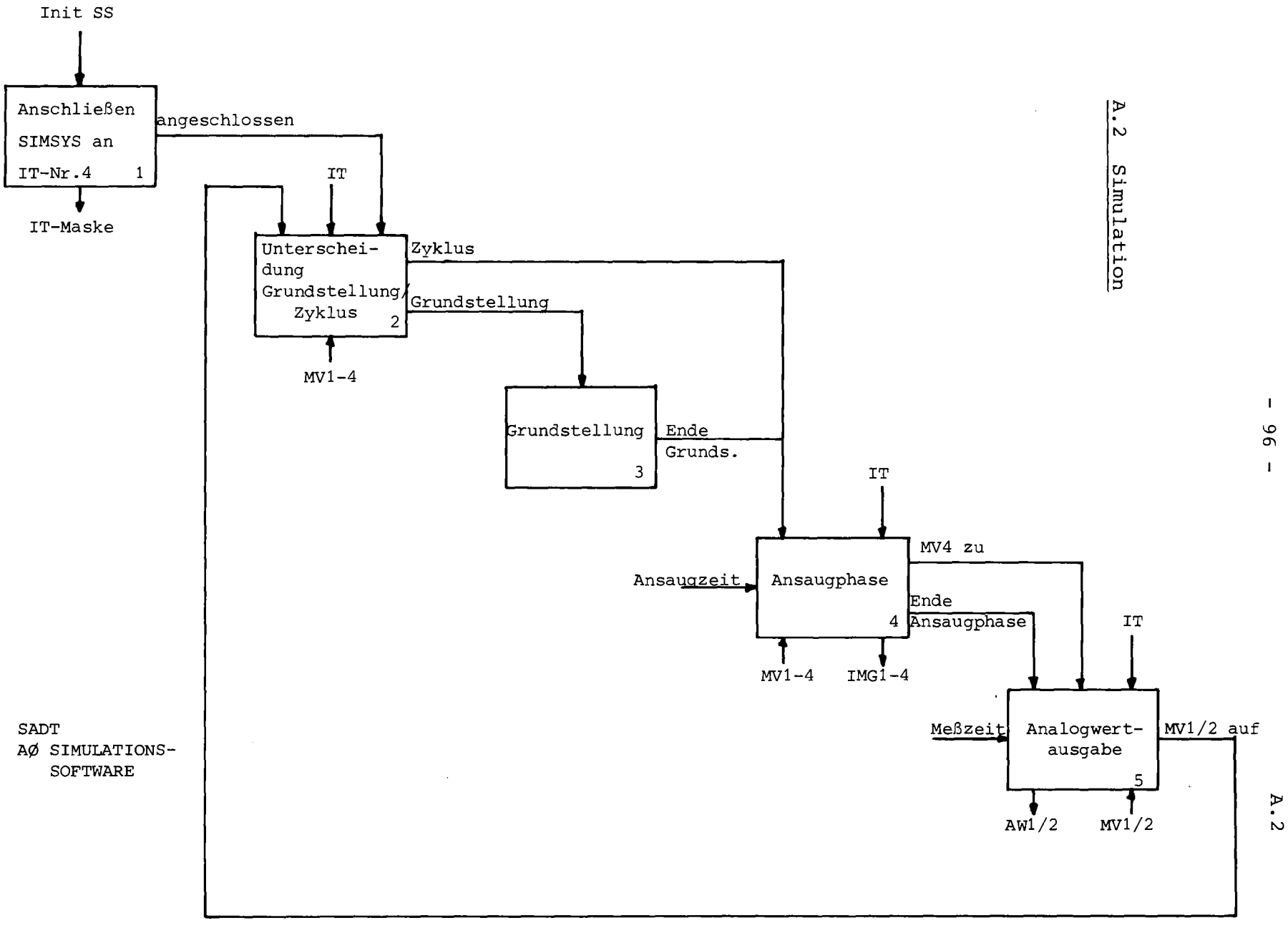
SADT
PROZESSÜBERWACHUNG
A3 ALARMSYSTEM

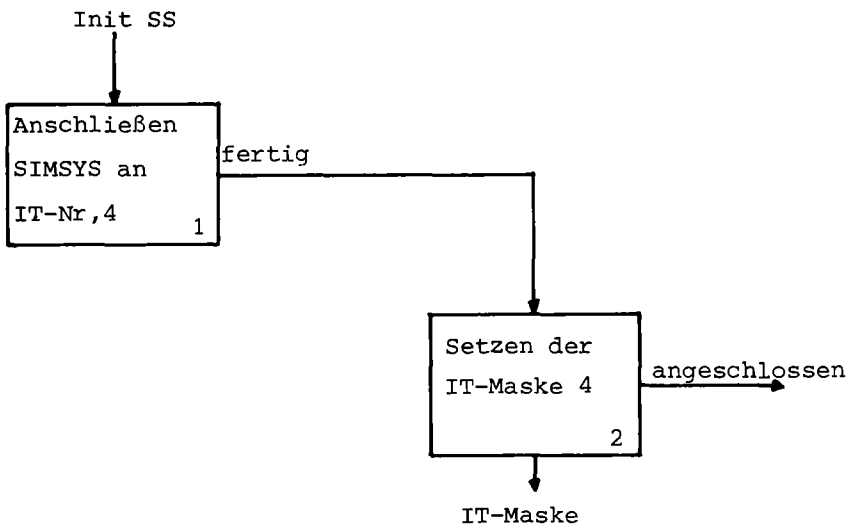


SADT
PROZESSÜBERWACHUNG
A4 DATEN-/STATUS-/VENTILSTELLUNG-AUSGABE

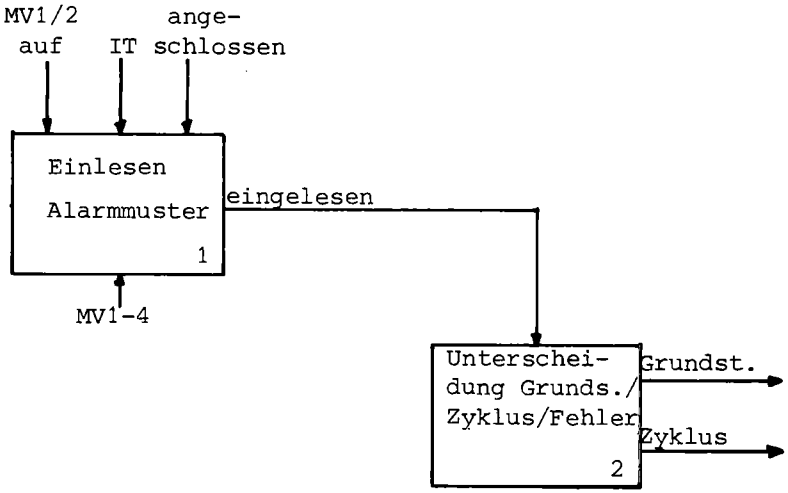


SADT
PROZESSÜBERWACHUNG
A5 FEHLERAUSGABE

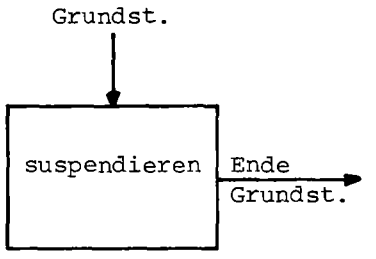

Init SS
Anschließen SIMSYS an IT-Nr.4 1
angeschlossen
IT-Maske
IT
Unterscheidung Grundstellung/ Zyklus 2
Zyklus
Grundstellung
MV1-4
Grundstellung 3
Ende Grunds.
IT
Ansaugzeit
Ansaugphase 4
MV4 zu
Ende Ansaugphase
MV1-4
IMG1-4
IT
Meßzeit
Analogwert-ausgabe 5
MV1/2 auf
AW1/2
MV1/2
SADT
AØ SIMULATIONS-SOFTWARE



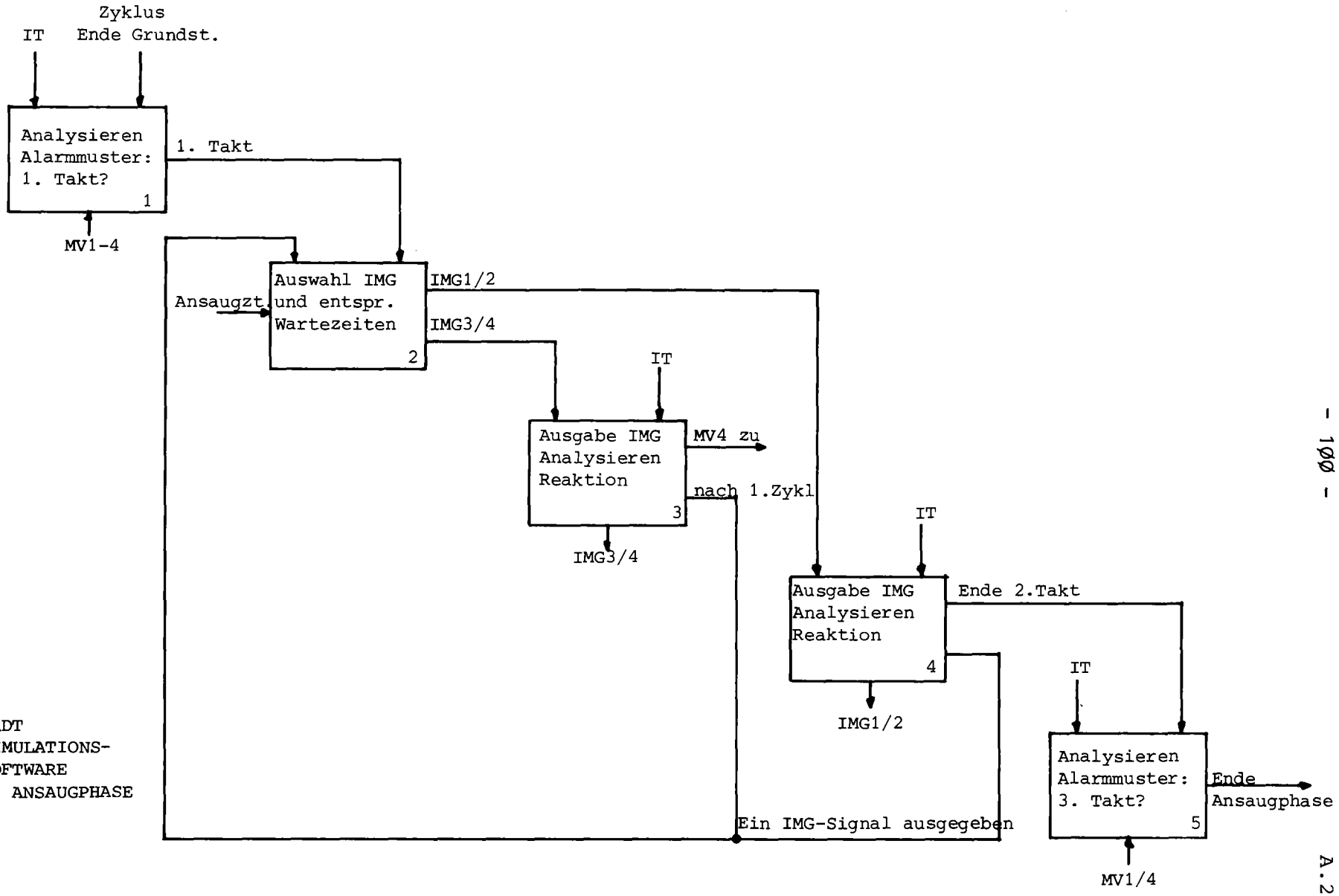
SADT
SIMULATIONSSOFTWARE
A1 ANSCHLIESSEN SIMSYS



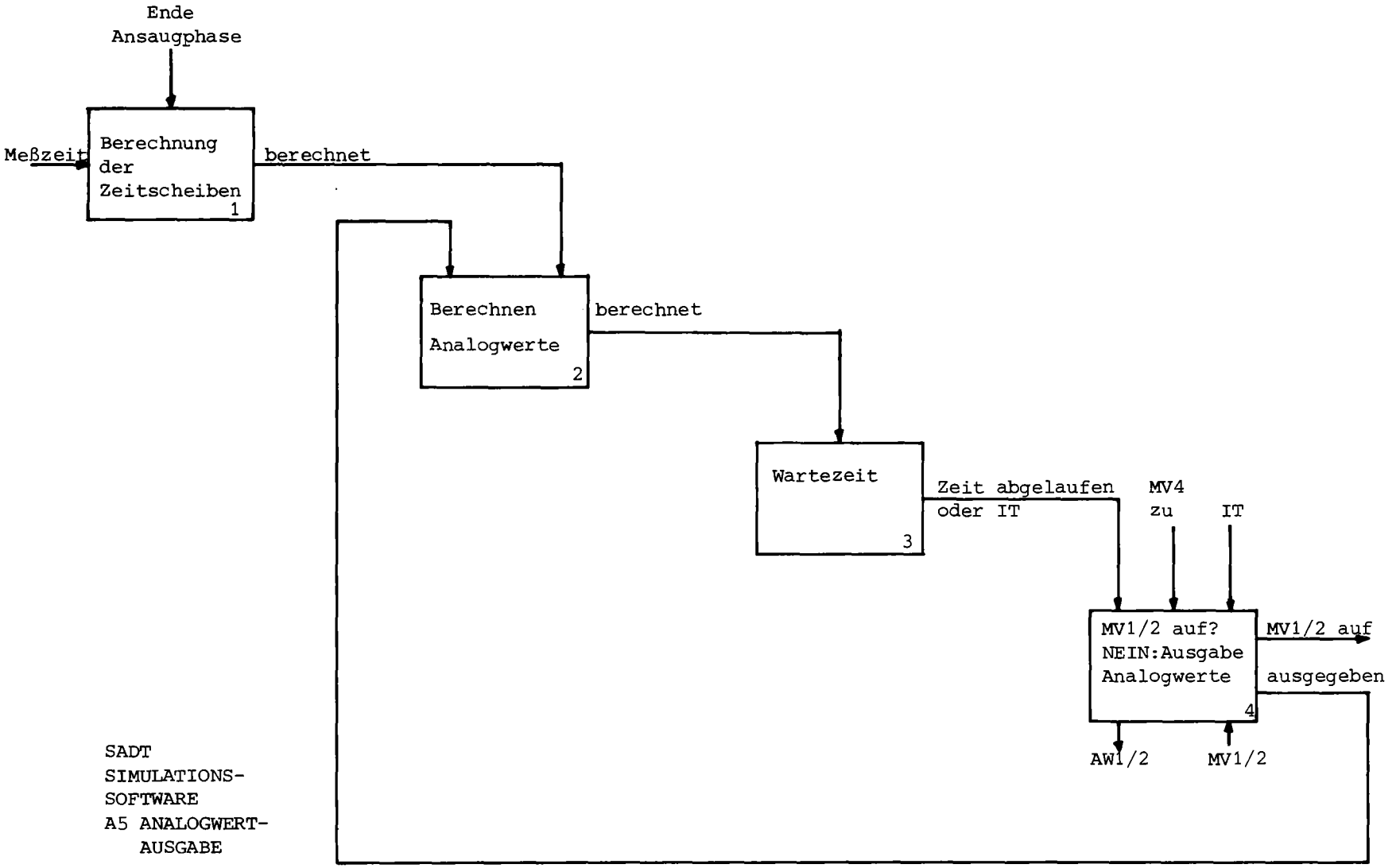
SADT
SIMULATIONSSOFTWARE
A2 UNTERSCHEIDUNG GRUNDSTELLUNG/ZYKLUS



SADT
SIMULATIONSSOFTWARE
A3 GRUNDSTELLUNG

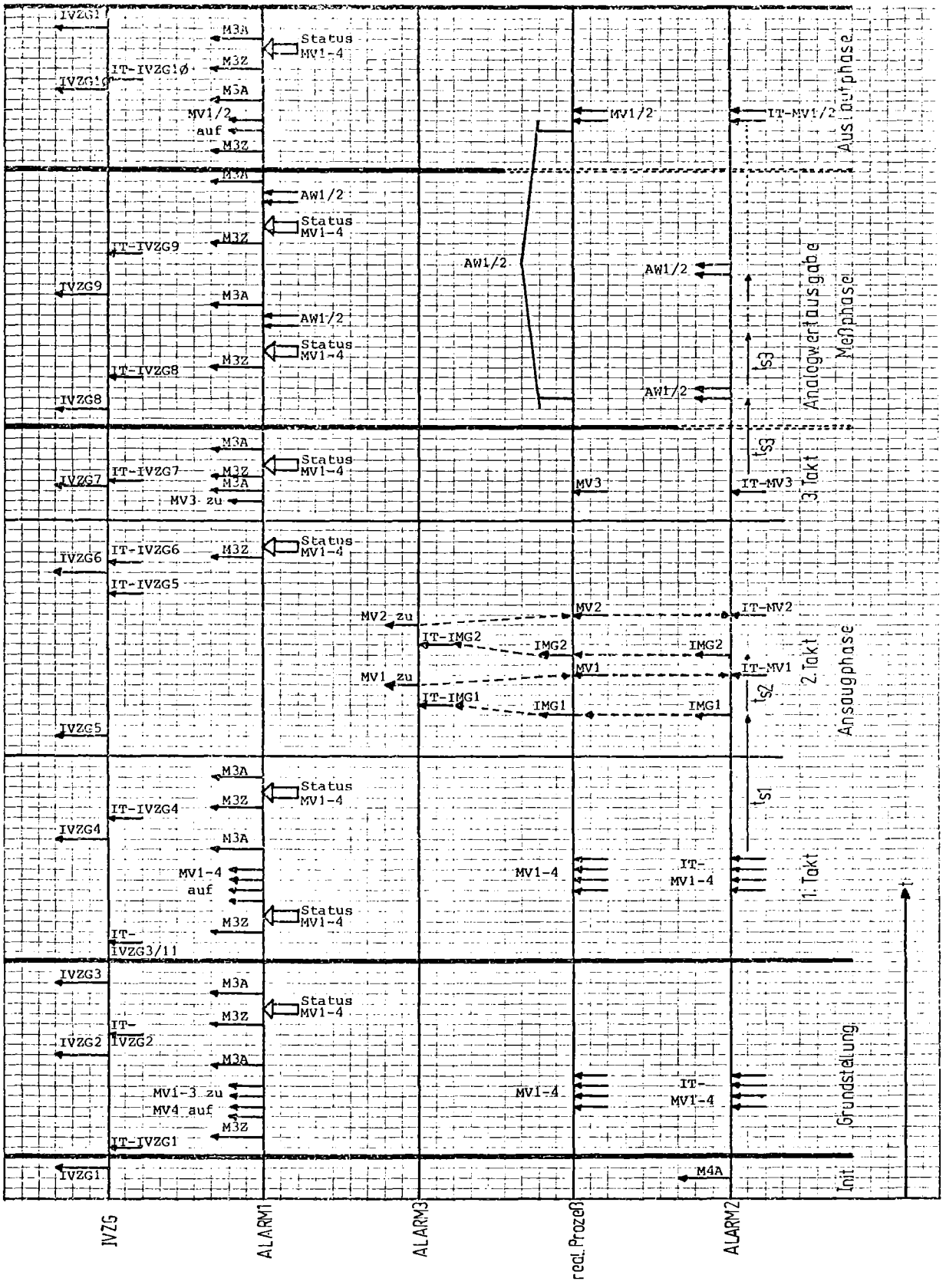


SADT
SIMULATIONS-
SOFTWARE
A4 ANSAUGPHASE


Ende
Ansaugphase
Meßzeit
Berechnung der Zeitscheiben
1
berechnet
Berechnen Analogwerte
2
berechnet
Wartezeit
3
Zeit abgelaufen oder IT
MV4 zu
IT
MV1/2 auf? NEIN:Ausgabe Analogwerte
4
MV1/2 auf
ausgegeben
AW1/2
MV1/2
SADT
SIMULATIONS-
SOFTWARE
A5 ANALOGWERT-
AUSGABE

A N H A N G B

Das dynamische Modell - Signaldiagramm



A N H A N G C

Bewertungstabellen

Sprachmittel	DASMR	BASIC	FORTRAN	IFTRAN	PASCAL	MINIMUM
DATENTYPEN						
INTEGER	2	1	2	2	2	J
REAL	2	2	2	2	2	J
CHAR	1	Ø	Ø	Ø	2	J
BIT	2	Ø	Ø	Ø	1	J
"Prozeßtypen"	1	Ø	Ø	Ø	1	J
DATENSTRUKTUREN						
Felder (max.Dim)	2	1	3	3	3	3
Verbunde	1	Ø	Ø	Ø	2	J
KOMMUNIKATION						
Inter-Task-Kommunikation	2	Ø	2	2	Ø	J
globale Variable	2	2	2	2	2	J
VEREINBARUNGEN						
implizite	Ø	1	1	1	Ø	(J)
explizite	2	Ø	2	2	2	J
Konstanten	2	Ø	Ø	Ø	2	J
UNTERPROGRAMME						
Funktionen	1	1	2	2	2	J
Prozeduren	2	2	2	2	2	J
GRUNDOPERATIONEN						
Arithmetische	2	2	2	2	2	J
Logische	2	Ø	2	2	2	J
Vergleiche	1	2	2	2	2	J
ABLAUFSTEUERUNG						
math.Operatorvorrang	Ø	2	2	2	2	J
Klammerung	Ø	2	2	2	2	J
Blockstruktur	Ø	Ø	Ø	Ø	1	J
Schleifen mit Anfang	Ø	Ø	Ø	2	2	J
Ausgang am Ende	Ø	2	2	2	2	J
Sprünge	1	1	1	1	Ø	(J)
Bed. Anweisungen: IF..THEN..ELSE	Ø	1	1	2	2	J
Fallunterscheidung	Ø	1	1	2	2	J

Fortsetzung Bewertungstabellen

Sprachmittel	DASMR	BASIC	FORTRAN	IFTRAN	PASCAL	MINIMUM
EIN-/AUSGABE						
gepuffert	2	2	2	2	2	J
ungepuffert	1	Ø	1	1	Ø	(J)
direkt	2	Ø	2	2	2	J
sequentiell	2	2	2	2	2	J
binär	2	Ø	2	2	2	J
alphanumerisch	2	2	2	2	2	J
logisch adressierend	2	Ø	2	2	2	J
Fehlerüberwachung	2	Ø	2	2	2	J
Summe	43	29	48	52	56	63

Erklärung der Bewertung:

Ø : nicht vorhanden
1 : teilweise vorhanden
2 : vorhanden
3 : mehr vorhanden als gefordertes Minimum

C

Fortsetzung Bewertungstabellen

Anwendung	DASMR	BASIC	FORTRAN	IFTRAN	PASCAL	maximale Bewertung
Zuverlässigkeit, Sicherheit	1Ø	12	2Ø	2Ø	29	3Ø
Modularität	7	4	13	2Ø	26	27
Dokumentation, Lesbarkeit	6	8	14	19	22	24
Wartung	7	7	14	18	23	24
Effizienz +) ++)	21	7	7	6	8	21
Implementierungs-aufwand ++)	7	12	9	11	11	18
Testhilfen	17	15	14	14	1Ø	18
Erlernbarkeit	1Ø	15	13	13	8	15
Summe	85	8Ø	1Ø4	121	137	177

+) Der Begriff Effizienz kann nicht absolut festgelegt werden. Deshalb wird hier die Relation der Programmgrößen untereinander verglichen. Ein Programm, das im Verhältnis zu einem anderen mehr Speicher belegt, wird dementsprechend eingestuft. Das "minimalste" Programm erhält die maximale Punktzahl.

++) BASIC ist nur exemplarisch realisiert. Das bedeutet, daß keine absoluten Aussagen hinsichtlich Effizienz und Implementierungsaufwand gemacht werden können. Die hier angegebenen Werte entstammen einer Schätzung anhand der BASIC-Aussagen, die man angeben konnte.

A N H A N G D

Das statische Programmprofil

Anzahl der Anweisungen		INPAS				INABST				PAREIN				ALARM1				ALARM3			
		FO	IF	PA	AS	FO	IF	PA	AS	FO	IF	PA	AS	FO	IF	PA	AS	FO	IF	PA+	AS
Add/Sub		Ø	Ø	Ø	2	Ø	Ø	2	1	18	18	22	22	41	39	31	168	Ø	1	1	9
Mult/Div		Ø	Ø	Ø	Ø	Ø	Ø	2	Ø	4Ø	4Ø	43	12	27	26	32	7	1	1	1	1
Vergleiche		2	2	4	2	4	4	4	5	11	15	24	9	1Ø8	142	128	134	7	22	12	11
logische Operationen		Ø	1	3	Ø	Ø	Ø	Ø	Ø	Ø	1	3	6	13	28	27	1	2	1Ø	2	Ø
UP-Aufrufe		3	3	16	Ø	4	4	18	4	17	17	53	19	153	157	336	176	8	7	13	4
Zuweisungen		Ø	Ø	6	-	4	4	6	-	56	55	87	-	219	245	4Ø5	-	15	17	26	-
Schleifen		Ø	1	3	1	Ø	Ø	1	Ø	2	7	16	3	13	18	49	29	Ø	2	1	2
Sprünge		2	Ø	Ø	2	Ø	Ø	Ø	7	1Ø	Ø	Ø	1Ø	97	Ø	Ø	199	13	Ø	Ø	19
bedingte Anweisungen		2	1	3	2	4	4	4	5	11	9	21	9	1ØØ	97	121	134	7	7	1	11
Ein-/Ausgabe:	OC	2	3	4	4	4	4	4	1	9	9	12	9	48	48	84	6	3	2	2	Ø
	LP	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	3	3	28	2	Ø	Ø	Ø	Ø
	DISC	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	1	1	1	7	7	7	7	Ø	Ø	Ø	Ø
	MT	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	3	3	28	2	Ø	Ø	Ø	Ø
	VFD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	2	2	2	2	8	8	5	8	Ø	Ø	Ø	Ø
	MOPEK	Ø	Ø	Ø	Ø	4	4	4	4	1	1	1	1	43	43	43	43	3	2	2	3

FO: FORTRAN
IF: IFTRAN
PA: PASCAL
AS: ASSEMBLER

OC : Operateur-Konsole
LP : Drucker
DISC : Platte
MT : Magnetband

+) In PASCAL gibt es keine Task "ALARM3". Die hier angegebenen Werte stammen aus dem Unterprogramm "ALARM" (siehe 6.2).