Diplomarbeit

Schulungskonzept für das Feldbussystem P-NET

ausgeführt am

Institut für Computertechnik
der Technischen Universität Wien

unter Anleitung von
O.Univ. Prof. Dr. Dietmar Dietrich

und den Betreuern
Dipl.-Ing. Mag. Martin Manninger
Univ. Ass. Dipl.-Ing. Thilo Sauter

durch

Franz J. GIRA
Dresdnerstraße 51/8
1200 WIEN

Wien, am 10.9.1996

(Franz J. GIRA)

Inhaltsverzeichnis

Kurzfassung

Diese Arbeit beschreibt den Aufbau einer Übungsanlage für das Feldbussystem P-NET im **Feldbus-Kompetenzzentrum** an der Technischen Universität Wien. Aufgrund der übersichtlichen Gestaltung des gesamten P-NET-Standards und der komfortablen Programmiermöglichkeit, ist P-NET besonders geeignet für den ersten Kontakt eines Studenten mit der Thematik der Feldbusse.

Um P-NET verstehen zu können wird zu Beginn dieser Arbeit anhand des im Standard umgesetzten ISO-OSI-Referenzmodells die Kommunikation der verschiedenen Knoten beschrieben. Ein weiterer wichtiger Teil der Arbeit war der elektrische und mechanische Aufbau der Anlage, dessen Ergebnisse ebenfalls dokumentiert werden. Darauf aufbauend wird als Kern dieser Arbeit ein Schulungskonzept vorgestellt, das bereits im Rahmen einer Laborübung im Sommersemester 1996 realisiert wurde.

Abstract

This work deals with designing a system for a training program on the fieldbus system P-NET. This system is installed at the **Center of Excellence for Fieldbus Systems** at Vienna´s **University of Technology.** Because of the clearly structured standardisation and the comfortable programming, P-NET is suitable for a students first steps into the topic of fieldbus systems.

To understand P-NET, at the beginning of this work the communication in the net is described by using the ISO OSI 7 layers reference model, which is implemented in the P-NET Standard. The next section presents the results of the mechanical assembly of the system. The main section of this work is the description of a training program, wich was completed in the summer term of 1996.

Abkürzungen

CENELEC	**C**omite **E**uropeen de **N**ormalisation **Elec**trotechnique
EEPROM	**E**lectrically **E**raseable **P**rogramable **R**ead **O**nly **M**emory
EIA	**E**lectronic **I**ndustries **A**ssociation
FIFO	**F**irst **I**n **F**irst **O**ut
GCS	**G**raphic **C**ontrol **S**tation
ISO	**I**nternational **S**tandardisation **O**rganisation
NRZ	**N**on **R**eturn to **Z**ero
OSI	**O**pen **S**ystem **I**nterconnection
PROM	**P**rogramable **R**ead **O**nly **M**emory
RAM	**R**andom **A**ccess **M**emory
ROM	**R**ead **O**nly **M**emory
RPW	**R**ead **P**rotected **W**rite
SDL	**S**pecification and **D**iscription **L**anguage
SWNo	**S**oft**w**ire **Number**
UPI	**U**niversal **P**rocess **I**nterface
VIGO	**V**irtual **I**nterface using **G**lobal **O**bjects
LAN	**L**ocal **A**rea **N**etwork
LON	**L**ocal **O**perating **N**etwork

1 Einleitung

Ein Konzept wie CIM (**C**omputer **I**ntegrated **M**anufacturing) setzt einen hohen Grad von Vernetzung eines Produktionsbetriebes voraus, wie es in Abb. 1 durch die sogenannte CIM-Pyramide angedeutet ist. Einer der gravierendsten Vorteile einer Vernetzung liegt auf der Hand: Zugriff der oberen Schichten auf alle Daten der Meßwerterfassung und auf den Status aller Stellglieder.

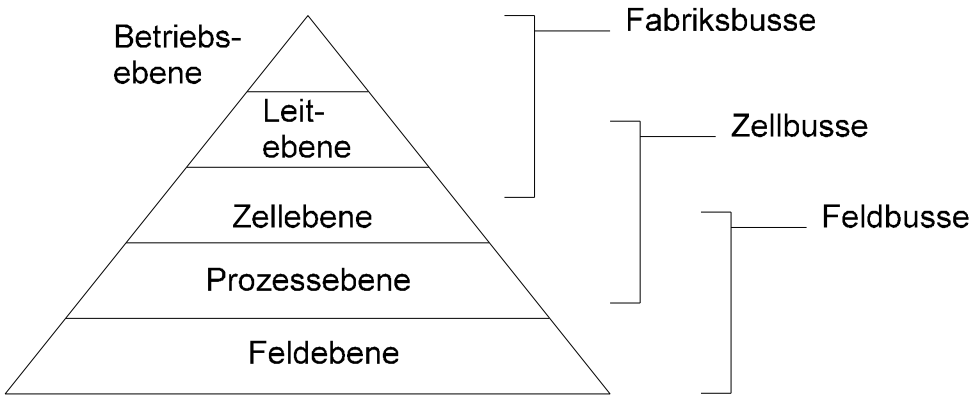


Abb. 1: Die CIM-Pyramide

Feldbusse werden heute in weiten Bereichen der Automatisierungstechnik eingesetzt. Diese Anwendungsgebiete können gemäß Abb. 1 in mehrere Ebenen unterteilt werden. Je nachdem welche Ebene der Automatisierungspyramide man betrachtet, kommen unterschiedliche Kommunikationssyteme zum Einsatz. Dabei überstreichen Feldbussysteme (von der untersten Ebene, der Feldebene kommend) mehrere Bereiche dieser Hierarchie.

Man kann Feldbussysteme danach einteilen, für welche Ebenen sie Funktionen bereitstellen. Ist das Einsatzgebiet auf eine Ebene, oder einen Teilbereich einer Ebene beschränkt, spricht man von Spezialbussen. Das sind zum Beispiel Feldbusse die auf die Verbindung von Sensoren und Aktoren mit der ersten Steuerungsebene spezialisiert sind und daher nur die unterste Schicht, die Feldebene abdecken. Beispiele für solche Bussysteme sind ASI, Interbus-S und das SERCOS-Interface.

Andere Systeme wiederum sind sehr komplex aufgebaut und unterstützten umfangreiche Protokolle, die eine Anwendung bis in die Zellenebene erlauben. Beispiele hierfür sind P-NET und PROFIBUS-FMS. Solche Feldbusse bezeichnet man deshalb auch als Universalbusse. Sie sind zum Aufbau größerer Strukturen fähig, ihr Einsatzgebiet überstreicht dabei mehrere Ebenen.

1.1 Entwicklung

Die ersten Prozeßrechner wurden Anfang der 60er Jahre entwickelt. Diese Rechner waren aber vorwiegend für die Prozeßoptimierung gedacht, die sich jedoch wesentlich ökonomischer auf einem zentralen Großrechner durchführen ließ. Die Hardwarekosten für eine hohe Anzahl an kleineren Rechnereinheiten waren zu dieser Zeit noch untragbar.

Mit der Erfindung des Mikroprozessors wurden in den 70er Jahren die ersten dezentralen Rechensysteme und Speicherprogrammierbaren Steuerungen gebaut. Durch diese neue Technik konnte die Qualität der gefertigten Produkte erhöht werden. Darüber hinaus ist die Flexibilität einer digital gesteuerten Anlage ein Vielfaches im Vergleich zu einer zu dieser Zeit herkömmlichen Anlage. Die in dieser Phase der Entwicklung eingesetzten Rechner arbeiteten jedoch noch weitestgehend im Inselbetrieb, boten also noch keine einheitliche Schnittstelle nach außen.

In den 80er Jahren wurden diese Einheiten immer komplexer. Sie wurden mit LC-Displays, Diagnosefunktionen und verschiedenen Sensor- bzw. Aktorverbindungen ausgestattet. Nun tauchte das Problem auf, daß eine große Menge an Information auf dem am Feldgerät vorhandenen Display zur Verfügung stand, das Prozeßleitsystem jedoch keinen Zugriff auf diese Daten hatte.

Neben dieser Schnittstelle zum Prozeßleitsystem bietet das Konzept des Feldbusses noch den Vorteil großer Flexibilität, da über eben diese Schnittstelle beliebige Steuer-, Meß- und Regelgeräte kommunizieren können [FBu92].

In der industriellen Praxis haben sich aufgrund unterschiedlicher Anforderungen, aber auch aus firmenpolitischen Gründen verschiedene Standards durchgesetzt. Einer dieser Standards ist das in dieser Diplomarbeit behandelte P-NET.

1984 von der dänischen Firma Process-Data speziell für die Prozeßautomatisierung entwickelt, wird P-NET heute firmenübergreifend von der *International P-NET User Organisation* mit Sitz in Silkeborg/DK vertreten.

1.2 Anforderungsprofile für ein Felbussystem

Die Anforderungen an ein Feldbussystem können je nach Einsatzgebiet in weiten Bereichen variieren. Wo in der Fertigungstechnik mit Reaktionszeiten von ms oder gar einigen µs gerechnet werden muß, findet man in der Verfahrenstechnik mit Zeiten im Sekundenbereich das Auslangen. Ebenso wie in der Fertigungstechnik selten Distanzen über 100m zu überwinden sind, hingegen bei verfahrenstechnischen Prozessen Leitungslängen von einigen hundert Metern anfallen. Unter solchen Gesichtspunkten erfüllen verschiedene Feldbussysteme jeweils spezielle Anforderungsprofile.

INTERBUS-S wurde z. B. speziell für die Vernetzung von Frequenzumrichtern, bei der eine kurze Reaktionszeit und hohe Datenmengen anfallen, entwickelt. Im Gegensatz dazu

lag bei PROFIBUS ursprünglich der Gedanke einer Vernetzung von SPS-Einheiten zugrunde.

P-NET hat seinen Platz in der Verfahrenstechnik, wurde ursprünglich zur Vernetzung von Durchflußtransmittern konzipiert und mittlerweile natürlich erweitert auf ein universelles Feldbussystem für verschiedenste Anwendungsgebiete [FBu92].

1.3 Ziele dieser Diplomarbeit

An Hand dieses Schrifttums soll gezeigt werden, wie mit der dazugehörigen Laboranordnung, Studenten oder anderen elektrotechnisch vorgebildeten Personen die Technologie und die Handhabung eines Feldbussystemes beigebracht werden kann. Aufgrund des übersichtlichen Aufbaus und der einfachen Programmierung ist P-NET für diesen Zweck besonders geeignet.

Von den theoretischen Grundlagen, die speziell für das Verständnis von P-NET nötig sind, werden folgende Punkte besonders herausgearbeitet:

- **PROCESS-PASCAL:** Dies ist eine von ISO-PASCAL abgeleitete Programmiersprache, die sich vor allem durch Multitaskingfähigkeit, einige spezielle Datentypen und der Möglichkeit, direkt auf P-NET-Variablen zuzugreifen, auszeichnet.
- Datentransport anhand des **ISO-OSI-Referenzmodells**, von dem im P-NET-Standard die Schichten 1-4 und 7 implementiert sind.
- Die **Channel-Struktur** von P-NET. Ein Channel ist die Zusammenfassung mehrerer Variablen, die einer Funktion dienen. Es gehören z. B. zu einem digitalen I/O neben dem Status, auch Daten wie der Ausgangsstrom, ein Grenzwert oder Errorflags zum jeweiligen Channel.

Mit dem in dieser Arbeit entwickelten Schulungskonzept soll gezeigt werden wie eine verbal formulierte Aufgabe (z. B. Temperatur-, Druck-, und/oder Durchflußmengenregelung) durch SDL-Diagramme erfaßt, in verschiedene *Tasks* aufgeteilt und anschließend auf einer bereits vorhandenen P-NET-Anlage realisiert wird.

Diese Aufgabenstellung hat einen durchaus praxisorientierten Aspekt: Ein großer Vorteil der Problemlösung mittels Feldbus liegt in seiner großen Flexibilität. Es ist z. B. sehr wahrscheinlich, daß sich die Anforderungen an eine chemische Anlage im Laufe ihres Bestehens mehrmals verändern oder Teile hinzugefügt werden, oder, wie in diesem Fall, das ganze System neu konfiguriert werden soll.

Nach erfolgreicher Inbetriebnahme der Versuchsanlage werden noch die Variation einzelner Systemparameter und das Auftreten verschiedener Störungen behandelt. Für diesen Fall sind Möglichkeiten vorgesehen, die Versorgungsspannung und die Busleitungen an verschiedenen Stellen zu unterbrechen. Speziell zur Behandlung von Fällen wie Über-

schreiten eines maximalen Ausgangsstromes oder Ausbleiben einer korrekten Reaktion, sind im P-NET-Standard einige Features implementiert.

2 Der P-NET-Standard

In diesem Abschnitt wird anhand des ISO-OSI-Referenzmodells der Aufbau von P-NET erläutert und zwar durch eine Gegenüberstellung der Formulierung laut ISO und der im P-NET-Standard festgelegten Implementierung. Des weiteren werden einige wichtige Begriffe wie *Channel* und *Softwiring* definiert. Zum Abschluß erfolgt noch eine kurze Einführung in die Programmiersprache PROCESS-PASCAL.

2.1 Implementierung des ISO-OSI-Referenzmodells

Die Idee hinter dem OSI Referenzmodell (Tab. 1) ist folgende:
Es gibt keine Vorschriften für die Gestaltung der Hard- und Software eines Systems, sondern lediglich eine Norm für die Protokolle mit denen es nach außen kommuniziert. Man spricht dann von einem **offenen System** [DD93].

7	**Anwendungsschicht** (Aplication Layer)
6	**Darstellungsschicht** (Presentation Layer)
5	**Kummunikationssteuerschicht** (Session Layer)
4	**Transportschicht** (Transportation Layer)
3	**Vermittlungsschicht** (Network Layer)
2	**Sicherungsschicht** (Link Layer)
1	**Bitübertragungsschicht** (Physical Layer)

Tab. 1: OSI-Referenzmodell

Im P-NET-Standard sind die Schichten 1 bis 4 und die Schicht 7 implementiert. Anhand dieser Umsetzung soll auf den folgenden Seiten dokumentiert werden, wie die einzelnen Knoten in einem P-NET-System kommunizieren.

Zuvor noch einige wichtige Begriffe:

Protokoll
Ein Protokoll ist die Art der Kommunikation zweier Instanzen, die der selben Schicht angehören z. B. Schicht 2 in Knoten 12 und Schicht 2 in Knoten 17.

Schnittstelle
Die Verbindung zwischen zwei benachbarten Schichten heißt Schnittstelle z. B. zwischen Schicht 2 und Schicht 3 im selben Knoten.

Dienst

Die Leistung, die eine Schicht der ihr direkt übergeordneten Schicht anbietet, ist ein Dienst.

2.1.1 Bitübertragungsschicht

Aufgabe laut ISO:
Steuerung des physikalischen Übertragungsmediums innerhalb des Kommunikationssystems.

Bei P-NET wurde auf den allgemein üblichen EIA RS485 Standard zurückgegriffen. Um die in diesem Standard vorgeschriebenen Abschlußwiderstände zu vermeiden, wird jedoch das Kabel zu einem physikalischen Ring geschlossen. Dadurch kann eine höhere Anzahl von Knoten, als die 32 im RS485 Standard erlaubten, angeschlossen werden [prEN170, S. 75]. Nach oben begrenzt ist die Knotenanzahl pro Bus durch die Länge eines Adressbytes: Für die Adressierung stehen 7 Bit zur Verfügung (siehe auch *Erstellen und Erkennen von Datenpaketen*), was Adressen von 0 bis 127 zuläßt, von denen jedoch die Werte 126 und 127 für Sonderfunktionen reserviert sind.

Daraus ergeben sich für **einen** P-NET-Bus folgende Daten:

Leitung:	Twisted Pair + Shield
max. Buslänge:	1200m
max. Knotenanzahl:	125

P-NET codiert nach dem NRZ-Verfahren (Non Return to Zero), bei dem das Signal während einer Bitperiode einen konstanten Wert hält.
Die einzige verwendete Übertragungsrate ist 76800 Bit/s (+/-0.2%).

2.1.2 Sicherungsschicht

Aufgabe laut ISO:
Sichern der Übertragung auf den einzelnen Teilstrecken des gesamten Übertragungsweges.

Bei P-NET hat die Sicherungsschicht im wesentlichen folgende Aufgaben [prEN170, S. 86]:

- Kontrolle des Buszugriffes
- Erkennung bzw. Erstellung der Knotenadresse und der Rahmenlänge eines Datenpakets
- Behandlung von Übertragungsfehlern

2.1.2.1 Multi Master Bus Access

Master sind die einzigen Instanzen im P-NET, die, sofern in Besitz des *Virtual Tokens*, von sich aus auf den Bus zugreifen können, **Slaves** hingegen sind nur nach einer an sie ergangenen Anfrage berechtigt, den Bus zu benutzen. Da P-NET ein Multimasterfähiges System ist, müssen die Zugriffsrechte auf den Bus verwaltet werden. Auch diese Aufgabe wird in Schicht 2 realisiert und zwar nach dem *Virtual Token Passing* Verfahren.

Jeder der maximal 32 in einem P-NET als Master definierten Knoten hat eine Knotenadresse (*Node Adress*) zwischen 1 und 32 und einen Leerlaufzähler (*Idle Bus Bit Period Counter*) der die Bitperioden zählt in denen sich der Bus im Leerlauf befindet. Weiters hat jeder Master einen Zugriffszähler (*Access Counter*) der inkrementiert wird, wenn der Leerlaufzähler 40, 50, 60, ...etc. erreicht.

Ist nun der Wert des Zugriffszählers gleich der Knotenadresse, hat dieser Master den Token und ist somit für eine Aktivität auf den Bus zugriffsberechtigt. Übersteigt der Zugriffszähler die Gesamtanzahl der Master wird er wieder auf 1 zurückgesetzt, dies ist der Grund, warum in jedem Master die **gleiche** Gesamtanzahl gespeichert sein muß.

Um die Konventionen und Zusammenhänge besser zu verdeutlichen, gibt es dazu in Abb. 2 ein einfaches Beispiel eines Systems mit 3 Master.

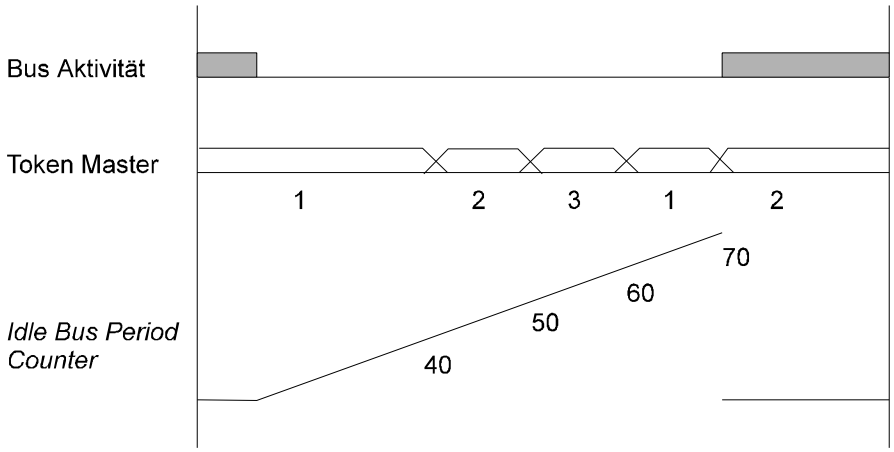


Abb. 2: Virtual Token Passing mit 3 Master

In diesem Fall hat zu Beginn Master 1 den Token und benutzt den Bus. Nach dem diese Aktivität beendet ist, geht der Bus in den Leerlauf über und der Leerlaufzähler wird laufend inkrementiert. Ist dieser bei 40 angelangt, wird der Token an Master 2 übergeben, nachdem dieser jedoch in der Zeitspanne von 10 Bitperioden nicht aktiv wird, geht der Token über zu Master 3, nach 10 Bitperioden weiter zu Master 1 und nach 10 weiteren Bitperioden wieder zu Master 2. Dieser hat in der Zwischenzeit Verwendung für den Bus und kann ihn für genau **eine** Anfrage benutzen.

2.1.2.2 Slave Bus Access

Slaves dürfen auf den Bus nur in einem Zeitfenster zwischen 11 und 30 Bitperioden nach einer Anfrage von einem Master zugreifen. Daher sollten Slaves so programmiert werden, das eine minimale Verzögerung zwischen Anfrage und Antwort entsteht, um das Gesamtsystem nicht zu blockieren. Die maximal erlaubte Verzögerung von 390 µs läßt sich aus folgenden Überlegungen einfach berechnen:

Übertragungsrate:	76800 Bit/s $\Rightarrow$
Dauer einer Bitperiode:	$T_{BIT} = 1/76800 = 13\mu s$
Die Antwort muß nach maximal 30 Bitperioden einlangen:	$30*T_{BIT} = 390\mu s$

Die verwendeten und berechneten Werte sind mit einer Toleranz von +/-0.2% zu verstehen.

2.1.2.3 Synchronisierung des Zugriffszählers

Jeweils die erste Adresse eines Datenpakets mit Bit7 = „1“ ist die Adresse des Token Masters, diese wird von jedem Master empfangen und mit dem internen Zugriffszähler verglichen. Sind diese beiden Werte verschieden geht dieser Master in den Zustand *out of synchronisation* über und darf den Bus nicht benutzen. Stimmen beim nächsten Vergleich die beiden Werte überein, gilt dieser Master wieder als *in synchronisation*. Ist der Offset gleich dem des ersten Vergleichs, wird der Zugriffszäler der empfangenen Token Master Adresse gleichgesetzt und der Master ist ebenfalls wieder *in synchronisation* [prEN170, S. 89].

2.1.2.4 Erstellen und Erkennen von Datenpaketen

Die Datenübertragung am Bus erfolgt in Datenpaketen *(Frames)*. Diese beinhalten Ziel- und Quelladressen, Kontroll- und Errorbytes und Nutzdaten, in der in Abb. 3 dargestellten Form.

Adreßfeld	Kontroll/Status	Nutzdatenlänge	Nutzdaten	Error
2-24 Byte	1 Byte	1 Byte	0-63 Byte	1-2 Bytes

Abb. 3: Format eines Datenpakets

Das **Adreßfeld** eines Datenpakets beinhaltet 2 bis 24 Adreßbytes, wovon jeweils Bit 0 bis Bit 6 die Eigentliche Adresse (1 bis 127) angibt und Bit 7 dazu dient verschiedene Adreßtypen bzw. Quell- und Zieladressen zu unterscheiden.

0	**Zieladresse**
1	**Quelladresse**

Abb. 4: Einfacher Adreßtyp

Dieser sogenannte **einfache Adreßtyp** beinhaltet nur eine Quelladresse und eine Zieladresse von jeweils einem Byte, kann daher nur zur Adreßierung innerhalb **eines** P-NET-Busses verwendet werden. Die Kennung zusammengesetzt jeweils aus Bit 7 ist „01“ (Abb. 4).

0	**Zieladresse**
0	**Zieladresse**
0	**Extraadresse**
1	**Quelladresse**
1	**Quelladresse**

Abb. 5: Komplexer Adreßtyp

Der **komplexe Adreßtyp** hingegen enthält zwei oder mehrere Zieladreßbytes und im dritten Byte die Anzahl der nachfolgend übertragenen Quelladreßbytes. Damit ist eine Adreßierung über mehrere P-NET-Ports hinweg möglich. Es ergibt sich eine Kennung von 3 oder mehr „0“ gefolgt von einer entsprechenden Anzahl von „1“(Abb. 5).

Das **Kontroll/Status Byte** hat beim senden eines Datenpakets von Master zu Slave die Funktion eines Kontrollbytes und beinhaltet ein Kommando, das im Slave ausgeführt wird. Es gibt u.a. Kommandos zum laden, speichern und für logische Verknüpfungen.
Im Antwort-Datenpaket des Slaves beinhaltet dieses Byte die Information über aufgetretene Fehler bei der Übertragung oder Fehler im Knoten.

2.1.3 Vermittlungsschicht

Aufgabe laut ISO:
Vermittlung und Aufbau des gesamten physikalischen Übertragungsweges.

2.1.3.1 Dualport Master

Bei P-NET gibt es Master mit zwei Ports, sogenannte *Dualport*- oder *Multiport-Master* , die jeweils zwei P-NET-Busse verbinden Die Vermittlungsschicht hat also in diesen Knoten die Funktion eines Routers zwischen zwei P-NET-Bussen und ist somit für das Routing des Übertragungsweges verantwortlich.

In Abb. 6 ist das Routing in einem P-NET-System mit 3 P-NET-Bussen dargestellt. Die dazugehörige Umwandlung des Adreßfeldes ist aus Abb. 7 ersichtlich: Master 1 hat eine Anfrage an Slave 3, also stehen zu Beginn eine Reihe von Zieladreßbytes und die Quelladresse 47 im Adreßfeld (Schritt 1). Da jedem der beiden Ports eines Multiport-Masters eine externe Adresse zugeordnet ist, wird bei erreichen von Port 1 des Multiport-Masters 1, P1 als Quelladresse eingeschrieben (Schritt 2). Umgekehrt wird bei Verlassen von Multiport-Master 1 über Port 2 dessen externe Adresse also 41 als Quelladreßbyte eingetragen (Schritt 3). Analog dazu geschieht die Umwandlung in Multiport-Master 2 (Schritte 4 und 5). Im P-NET-Bus 3 angelangt ist die Adreßierung von Slave 3 mit einem Byte eindeutig.

Für die Sendung der Antwort, kann nun dieses Adreßfeld, das am Ende nur noch aus Quelladreßbytes besteht, ohne weiteren Aufwand als Zieladreßfeld verwendet werden.

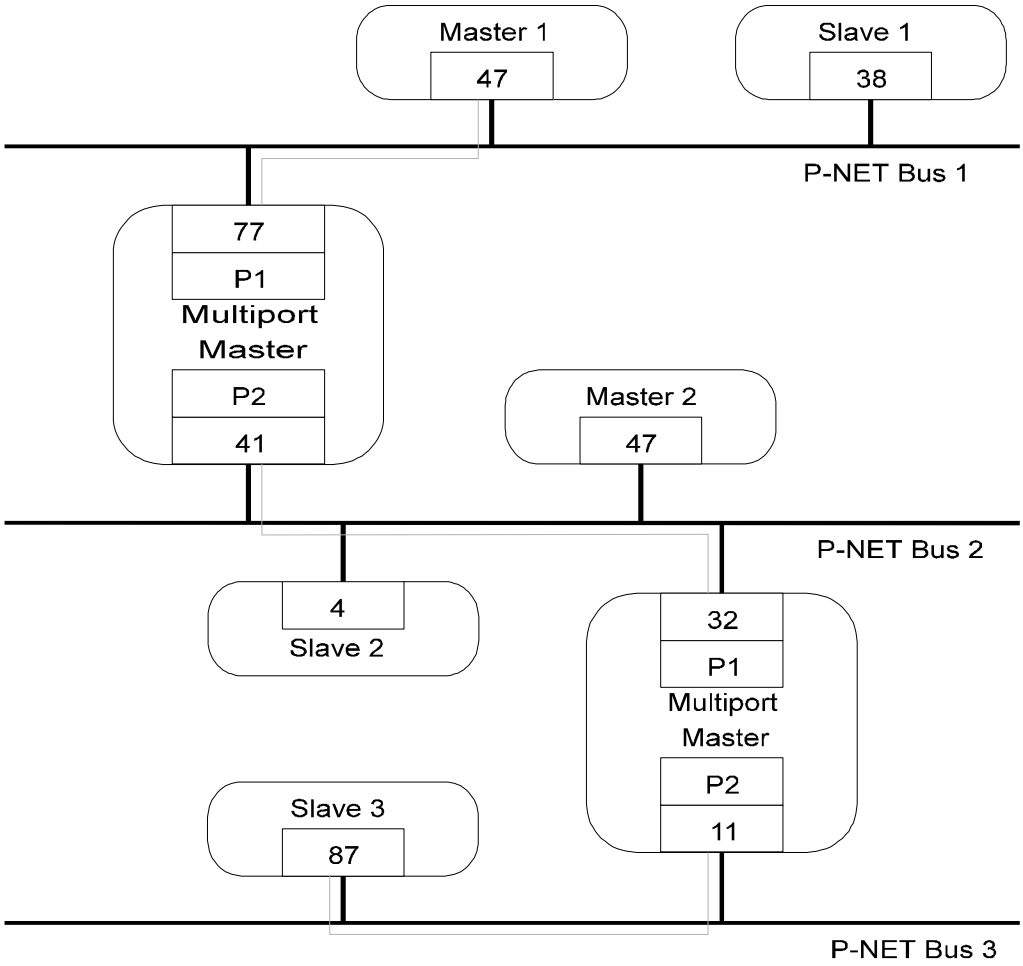


Abb. 6: Routing im P-NET

Schritt 1	77	P2	32	P2	87	47
Schritt 2	P2	32	P2	87	P1	47
Schritt 3	32	P2	87	41	P1	47
Schritt 4	P2	87	P1	41	P1	47
Schritt 5	87	11	P1	41	P1	47

Zieladreßbyte

Quelladreßbyte

Abb. 7: Umwandlung des Adreßfeldes bei einer Anfrage

2.1.3.2 Einfache Knoten

Die simpelsten P-NET-Knoten besitzen lediglich **einen** P-NET-Port. Das Adreßfeld eines von einem einfachen Knoten gesendeten Datenpakets ist daher stets ein einfacher Adreßtyp mit einem Zieladreßbyte und einem Quelladreßbyte. Routingfunktionen sind

daher nicht nötig, d. h. die Vermittlungsschicht hat lediglich für die Übergabe der Datenpakete zwischen der Sicherungsschicht und der Transportschicht zu sorgen.

2.1.4 Transportschicht

Aufgabe laut ISO:
Festlegen der Transportverbindung (= logische Verbindung) von der Nachrichtenquelle zur Nachrichtensenke.

Die wohl wichtigste Aufgabe der Transportschicht ist die Verwaltung der *Softwirelist*. Dies ist eine Tabelle von globalen Variablen wie z. B. Knotenadressen, Variablentypen, Meßwerte, SI Einheiten, etc. , auf die physikalisch dem eigenen oder einem externen Knoten zugeordnet sein können. Durch Kenntnis der logischen Adresse, der sogenannten *Softwire Number (SWNo)*, kann auf die Variablen der *Softwirelist* zugegriffen werden [prEN170, S. 98].

2.1.5 Anwendungsschicht

Aufgabe laut ISO:
Konkretisierung und Ausführung der Anwendung.

Diese oberste Schicht kann nun einerseits auf die von der Transportschicht angebotenen globalen Variablen zugreifen und bietet andererseits eine Schnittstelle zur Anwendung. An dieser Stelle sei bemerkt, daß die Anwendung selbst **nicht** Teil der Anwendungsschicht ist, sondern lediglich auf diese aufbaut.

Läuft also ein Anwenderprogramm, kann dieses über die *SWNo.* auf eine Variable im eigenen Knoten aber auch auf andere Knoten zugreifen. Wenn eine Eintragung in die *Softwirelist* eine externe Variable bezeichnet, d. h. sie befindet sich physikalisch außerhalb des Knotens in dem das Anwenderprogramm läuft, steht in der *Softwirelist* lediglich die Adresse des Knotens in dem die Variable lokalisiert ist. Die eigentliche Adresse dieser Variable ist in der *Softwirelist* des betreffenden Knotens eingetragen [prEN170, S. 109].

2.2 Die Channel Struktur

Ein einfacher P-NET-Knoten, z. B. ein analoger Eingang, hat die Aufgabe Meßdaten zu erfassen und weiterzusenden. Da die Meßdaten jedoch mehr Information als nur einen Zahlenwert benötigen um sinnvoll weiterverarbeite werden zu können, sind noch weitere Daten zur Skalierung, Umwandlung, Filterung, etc. nötig. Diese Zusammenfassung von

Variablen und Funktionen in Form eines Objektes, werden *Channel* bezeichnet. Es ist daher empfehlenswert ein objektorientiertes *Interface Modul* zu deklarieren, in dem nötigenfalls mehrere Channels zusammengefaßt werden können. Folglich können vom einem Programm sogar unbekannte Feldbusknoten angesprochen werden, solange sie bekannte Channeltypen benutzen.

Ein Channel ist in 16 Register unterteilt, von welchen jedes seine eigene logische Adresse, die bereits erwähnte *Softwire Number (SWNo)*, hat. Diese 16 Variablen oder Konstanten können von verschiedenen Typen oder ein Record verschiedener Typen sein, je nach Anforderung an den Channel. Weiters werden verschiedene Speicherformen (ROM, RAM, EEPROM) benutzt.

Im allgemeinen können Channels nicht nur für Meß- und Regelaufgaben definiert werden, sondern auch für Aufgaben der Datenübertragung wie z. B. Drucker Channels Barcodeleser Channels, etc. Nach einer kurzen Beschreibung der unterschiedlichen Speicherformen werden jedoch, zur Veranschaulichung des Aufbaus, einige standardisierte Channels beschrieben [STA92].

2.2.1 Speicherformen

Ein *Interface Modul* kann Daten auf verschiedene Arten speichern, abhängig von der notwendigen Verfügbarkeit nach einem gewünschtem RESET oder einem Ausfall der Versorgungsspannung [STA92].

Read Only

RAM Read Only: Die Variable wird im RAM abgelegt, kann aber dennoch nur gelesen werden.

PROM Read Only kann hingegen prinzipiell nur gelesen werden.

Read Protected Write (RPW)

RAM RPW: Die Variable ist nach einem RESET immer geschützt, kann aber nach setzen eines entsprechenden Flags verändert werden.

EEPROM RPW: Die Variable ist nach einem RESET immer geschützt, bis *WriteEnable* auf TRUE gesetzt wird. Danach kann der Wert verändert werden und behält ihren Wert auch während und nach einem RESET

Read Write

RAM ReadWrite: Die Variable kann immer verändert werden. Nach einem RESET ist ihr Wert Null.

Read Write, Protected BackUp Write

RAM InitEEPROM: Die Variable ist im EEPROM und im RAM abgelegt. Nach einem RESET wird der Wert vom EEPROM in den RAM kopiert. Bei Änderung des Wertes via P-NET wird nur der RAM verändert. Falls WriteEnable jedoch TRUE ist wird auch der EEPROM beschrieben.

RAM AutoSave: Hat die gleichen Eigenschaften wie RAM InitEEPROM. Darüber hinaus wird der Inhalt des RAMs automatisch in den EEPROM zurückkopiert.

2.2.2 Der Service Channel

Dieser Channel beinhaltet die ersten 16 festgelegten Softwire Numbers und hat somit die *Channel Nr.0*. Da jedes Gerät diesen Kanal besitzen muß, ist es über ihn möglich, sogar ein unbekanntes und nicht näher spezifiziertes Gerät anzusprechen. Er muß daher in jedem *Interface Modul* vorhanden sein und hat die in
Tab. 2 dargestellte Struktur [STA92].

SWNo	Identifier	Art des Speichers	Format	Type	Einh.
0	NumberOfSWNo	PROM Read Only		Integer	
1	DeviceID	PROM Read Only		Record	
2					
3	Reset	RAM Read Write	hex	Byte	
4	PNETSerialNo	Special Function		Record	
5					
6	TimeDate	Special Function		Record	
7	FreeRunTimer	RAM Read Only	dezimal	LongInteger	
8	WDTimer	RAM Read Write	dezimal	Real	s
9	ModuleConfig	EEPROM Read Only		Record	
A	WDPreset	EEPROM Read Only	dezimal	Real	s
B	MailFilter	RAM InitEEPROM		String	
C	MailBox	RAM Read Write		Buffer	
D	WriteEnable	RAM Read Write	binär	Boolean	
E	ChType	PROM Read Only		Record	
F	CommonError	RAM Read Write		Record	

Tab. 2: Struktur des Service Channels

Dazu eine kurze Erläuterung der einzelnen Softwire Numbers (SWNo.) :

SWNo. 0: *NumberOfSWNo* gibt die höchste SWNo. des Gerätes an

SWNo. 1: *DeviceID* gibt Type, Programmversion und Hersteller an.

SWNo. 3: *Reset*: Nach einschreiben von $FF führt das Gerät nach senden einer Bestätigung einen RESET durch.

SWNo. 4: *PNETSerialNo* hat die Form:

```
Record
   PNETNo: BYTE;
   SerialNo: String[20];
end;
```

Dieser Record beinhaltet die P-NET-Adresse des Geräts und dessen Seriennummer, die benötigt wird um die P-NET-Adresse via P-NET zu verändern.

SWNo. 6: *TimeDate*: Record für Zeit und Datum.

SWNo. 7: *FreeRunTimer*: Timer nach dem interne Abläufe synchronisiert werden.

P-NET Watch Dog Funktion
Jeder P-NET-Knoten der eine Output Funktion zu erfüllen hat, muß im Falle der Trennung vom Netz, d. h. es treffen keine neuen Anweisungen ein, einen für die Gesamtanlage sicheren Zustand einnehmen. Für diese sogenannte *Watch Dog Funktion* werden SWNo. 8,9 und A benötigt:

SWNo. 8: *WDTimer* wird nach einem RESET und anschließend bei jedem Aufruf des Knotens via P-NET mit dem Wert *WDPreset* (SWNo. A) geladen. Erreicht der Timer Null, wird das Flag *PnetWDRunOut* gesetzt und alle Outputs logisch 0 geschaltet.

SWNo. 9: *ModuleConfig* dient in erster Linie zur Aktivierung bzw. Deaktivierung der Watch Dog Funktion.

SWNo. A: *WDPreset* gibt die Maximale Zeit an, die zwischen zwei Aufrufen vergehen darf, ohne, daß die Watch Dog Funktion aktiv wird.

SWNo. B: *MailFilter* hat eine Filterfunktion für die *MailBox*.

SWNo. C: *MailBox* ist ein Speicher für Nachrichten (z. B. Alarmmeldung) mit einer Länge von maximal 89 Zeichen.

SWNo. D: *WriteEnable* ist nach einem RESET auf FALSE. Um geschützte Speicher zu verändern, muß WriteEnable auf TRUE gesetzt werden.

SWNo. E: *ChType* ist die Channelbeschreibung innerhalb eines Interface Moduls. In dieser ist abgelegt, welche Register und Funktionen (z. B. Watch Dog) in diesem Channel verwendet werden.

SWNo. F: *CommonError* beinhaltet die Information über alle Errors innerhalb eines Interface Moduls.

2.2.3 Der digitale I/O Channel

Dieser Channel kann als Schaltung betrachtet werden, die als Ausgang oder als Eingang konfiguriert werden kann (Abb. 8). Der Ausgang kann kontinuierlich geschaltet werden, als einzelner Impuls oder als periodischer Puls. Weitere Möglichkeiten sind eine Messung des Ausgangsstromes oder eine sogenannte *Feedback Control*, bei der über zwei zusätzliche Leitungen die korrekten Reaktionen auf einen Output festgestellt werden können [STA92], [PD21], [PD30].

Dazu sind folgende Variablen definiert:

SWNo	Identifier	Art des Speichers	Format	Type	Einh.
x0	FlagReg	RAM Read Write	binär	Array	
x1	OutTimer	RAM Read Write	dezimal	Real	s
x2	Counter	RAM Auto Save	dezimal	LongInteger	
x3	OutCurrent	RAM Read Write	dezimal	Real	A
x4	Operatingtime	RAM Auto Save	dezimal	Real	s
x5					
x6	FPTimer	RAM Read Write	dezimal	Real	s
x7	FBPreset	EEPROM RPW	dezimal	Real	s
x8	OutPreset	EEPROM RPW	dezimal	Real	s
x9	ChConfig	EEPROM RPW		Record	
xA	MinCurrent	EEPROM RPW	dezimal	Real	A
xB	MaxCurrent	EEPROM RPW	dezimal	Real	A
xC					
xD	Maintenance	EEPROM RPW		Record	
xE	ChType	PROM Read Only		Record	
xF	CHError	RAM Read Only	binär	Record	

Tab. 3: Struktur des digitalen I/O Channels

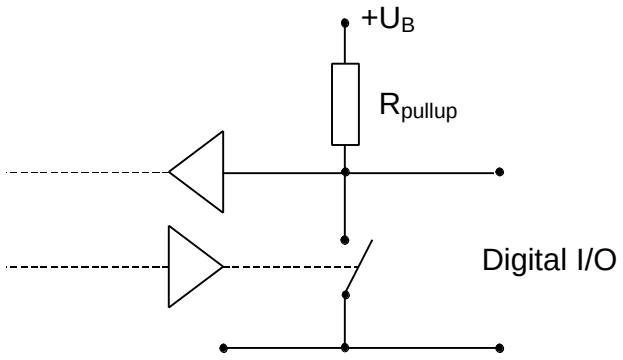


Abb. 8: Prinzipschaltbild des digitalen I/O Ports

Dazu wieder eine kurze Erläuterung der einzelnen Softwire Numbers:

SWNo. x0: *FlagReg*: Beinhaltet den Status des Aus- bzw. Einganges, ein Error Bit und weitere Kontrollfunktionen (Feedback Control, Error,...).

SWNo. x1: *OutTimer*: Timer für spezielle Output Funktionen z. B. One Shot Betrieb.

SWNo. x2 *Counter* zählt die Impulse am Eingang d. h. er inkrementiert seinen Wert wenn das *InFlag* aus SWNo. X0 von 0 auf 1 springt.

SWNo. x3: *OutCurrent* gibt den Laststrom am Ausgang in Ampere an.

SWNo. x4: *Operatingtime* gibt an, über welchen Zeitraum *InFlag* TRUE ist.

SWNo. x6: *FBTimer*: Der Feedback Timer wird benutzt, um festzustellen, ob eine Rückmeldung innerhalb der Zeit *FBPreset* einlangt und wird dazu bei jeder Zustandsänderung des Ausganges mit *FBPreset* geladen. Läuft der Timer ab, so wird ein entsprechendes Flag in SWNo. XF gesetzt.

SWNo. x7: *FBPreset* ist die maximale Zeit bis zum Eintreffen der Rückmeldung des Prozesses.

SWNo. x8: *OutPreset* speichert den Preset Wert für *OutTimer* und wird durch spezielle Output Funktionen in diesen geladen.

SWNo. x9: *ChConfig* legt den I/O Typ, die Art der Feedback Control (mit einer oder zwei Leitungen) und spezielle Output Funktionen fest.

SWNo. xA: *MinCurrent*: Wird dieser Wert unterschritten und ist der *FBTimer* abgelaufen, kann in SWNo. xF ein sogenannter Underload Alarm generiert werden.

SWNo. xB: *MaxCurrent* wirkt analog *MinCurrent* für überschreiten der Wertes.

SWNo. xD: *Maintenance* beinhaltet Datum und Code der letzten Wartung.

SWNo. xE: *ChType* besteht aus einer Reihe von Flags, die die benutzten Funktionen angeben.

SWNo. xF: *ChError* beinhaltet Errorbits wie: *Overload, Underload, FeedBackError, ...*

2.2.4 Der Analog Input Channel

Dieser Channel kann für folgende vier Meßbereiche konfiguriert werden:
- Spannung 0-100mV
- Strom 0-20mA
- Strom 4-20mA
- Temperaturmessung mittels Pt-100 Element (-100 bis +200C)

Weiters bietet dieser Channel unter anderem Errorfunktionen, Grenzwertüberwachung, eine Skalierung und einen Eingangsignalfilter [PD21]. Für diesen Channel sind folgende Variablen definiert:

SWNo	Identifier	Art des Speichers	Format	Type	Einh.
x0	AnalogIn	RAM Read Write	Decimal	Real	
x1					
x2					
x3					
x4					
x5					
x6					
x7	HighLevel	RAM init EEPROM	Decimal	Real	
x8	LowLevel	RAM init EEPROM	Decimal	Real	
x9	ChConfig	EEPROM RPW	Hex	Record	
xA					
xB	FullScale	EEPROM RPW	Decimal	Real	
xC	ZeroPoint	EEPROM RPW	Decimal	Real	
xD	Maintenance	EEPROM RPW		Record	
xE	ChType	PROM Read Only		Record	
xF	ChError	RAM Read Only	Binary	Record	

Tab. 4: Struktur des Analog Input Channels

SWNo. x0: *AnalogIn* beinhaltet den Meßwert, je nach Funktion des Channels in mA, mV oder C.

SWNo. x7: *HighLevel*: Bei Überschreiten dieses Wertes, wird das Flag *HighAlarm* gesetzt.

SWNo. x8: *LowLevel*: Bei Unterschreiten dieses Wertes, wird das Flag *LowAlarm* gesetzt.

SWNo. x9: *ChConfig* ist ein Record in dem alle möglichen Konfigurationen des Channelsvorgenommen werden. Die erste Teilvariable dieses Records ist *Enablebit*, in dem die Bit 0-7 folgende Bedeutung haben:

0 Inputsimulation
1 nicht benutzt
2 Alarm wenn der Laststrom unter 3 mA sinkt
3 Alarm bei Unterschreiten eines Benutzerdefinierten Grenzwertes
4 Alarm bei Überschreiten eines Benutzerdefinierten Grenzwertes
5 Alarm bei Unterschreiten des absoluten Grenzwertes
6 Alarm bei Überschreiten des absoluten Grenzwertes
7 nicht benutzt

An zweiter Stelle in diesem Record steht die Teilvariable *Functions*, durch die die eigentliche Funktion des Channels und die Zeitkonstante des Eingangsfilters festgelegt wird:

$0x Channel nicht benutzt
$1x Temperaturmessung mit Pt-100-Element
$2x Strommeßbereich 0-20 mA
$3x Strommeßbereich 4-20 mA
$4x Spannungsmessbereich 0-100 mV

$x0 kein Filter
$x1 Zeitkonstante 1 s
$x2 Zeitkonstante 2 s
$x3 Zeitkonstante 5 s
$x4 Zeitkonstante 10 s

SWNo. xB: *FullScale* ist bei Strom- bzw. Spannungsmessung der Maximalwert des Meßbereichs (z. B. 100mV)

SWNo. xC: ZeroPoint ist bei Strom- bzw. Spannungsmessung der Minimalwert des Meßbereichs (z. B. 0mV). Bei Temperaturmessung, beinhaltet dieser Wert einen Offset für das Pt-100 Element.

SWNo. xD: *Maintenance* beinhaltet Datum und Code der letzten Wartung.

SWNo. xE: *ChType* besteht im wesentlichen aus einer Reihe von Flags,

die die vom Channel benutzten Funktionen angeben.

SWNo. xF: *ChError* beinhaltet Errorbits (*SignalHigh, LowAlarm, ModuleError,...*)

2.2.5 Der Weight Channel

Dieser Channel bietet die Möglichkeit einer hochauflösenden Wiegung mit direkter Anzeige in den SI-Einheiten kg bzw. g. Weiters können 2 verschiedene Tara-Gewichte Angegeben werden um z. B. bei Abfüllvorrichtungen verschiedene Leergebinde benutzen zu können.

Im sogenannten *Belt Weight Mode* werden verschiedene Funktionen für das Messen eines Gewichts über ein Förderband angeboten. In diesem Modus berechnet sich der ausgegebene Wert *Weight0* als:

Weight0 := Weight0 + Flow * Sample Time

Es wird also die geförderte Menge aufintegriert, wobei *Flow* die Gewichtsänderung pro Zeit in kg/s darstellt und *Sample Time* die Zeit zwischen zwei Abtastungen ist. Diese Abtastzeit kann vom Benutzer zwischen 0.1 s und 5 s gewählt werden.

Um im *Belt Weight Mode* die Bandgeschwindigkeit zu verarbeiten, besteht die Möglichkeit, bei konstanter Geschwindigkeit, mittels eines Signals über einen digitalen Eingang, TRUE für das laufende Band und FALSE für das stehende Band zu verwenden. Ist die Bandgeschwindigkeit jedoch nicht konstant, muß am digitalen Eingang ein gepulstes Signal anliegen, dessen Frequenz der Geschwindigkeit proportional ist.

Für das Auslesen der Meßwerte, werden sowohl im *Weight Mode* als auch im *Belt Weight Mode* verschidene Funktionen, wie Mittelwertbildung über mehrere Samples und verschiedene Rundungen angeboten

Die folgenden Erläuterungen beziehen sich auf den für die Übung relevanten *Weight Mode* [PD30].

SWNo	Identifier	Art des Speichers	Format	Type	Einh.
x0	Weight0	RAM Read Write	Decimal	Real	kg
x1	Weight1	RAM Read Write	Decimal	Real	kg
x2	NetWeight	RAM Read Write	Decimal	Real	kg
x3	Flow	RAM Read Write	Decimal	Real	kg/s
x4	Frequency	RAM Read Write	Decimal	Real	
x5	FlagReg	RAM Read Write	Binary	BIT8	
x6	Tare	RAM Read Write	Decimal	Record	kg
x7	HighLevel	RAM init EEPROM	Decimal	Real	kg
x8	LowLevel	RAM init EEPROM	Decimal	Real	kg
x9	ChConfig	EEPROM RPW		Record	
xA	Factors	EEPROM RPW	Decimal	Record	
xB	FullScale	EEPROM RPW	Decimal	Real	kg
xC	Zeropoint	EEPROM RPW	Decimal	Real	kg
xD	Maintenance	EEPROM RPW		Record	
xE	ChType	PROM Read Only		Record	
xF	ChError	RAM Read Only	Binary	Record	

Tab. 5: Struktur des Weight Channels

SWNo. x0: *Weight0* ist das Nettogewicht vermindert um einer Benutzerdefinierte Konstante *Tare0*,
also: *Weight0 = NetWeight - Tare0*

SWNo. x1: *Weight1* ist das Nettogewicht vermindert um einer Benutzerdefinierte Konstante *Tare1*,
also: *Weight1 = NetWeight - Tare1*

SWNo. x2 *NetWeight*: Das Nettogewicht berechnet sich wie folgt:
*NetWeight = Input*FullScale - ZeroPoint*
Bei einschreiben von 0 wird automatisch der Wert für *ZeroPoint* und bei einschreiben einer bekannten Masse auf der Wiegeeinheit, der Wert *Full-Scale* berechnet.

SWNo. x3: *Flow* ist der Massefluß in kg/s und wird wie folgt berechnet:
Flow = (NetWeight - letzes *NetWeight) / SampleTime*

SWNo. x4: *Frequency* wird nur im Belt Weight Mode verwendet.

SWNo. x5: *FlagReg* beinhaltet unter anderem ein Vorzeichenbit und ein aus schließlich für den Belt Weight Mode verwendetes Flag.

SWNo. x6: *Tare* ist ein Record aus *Tare0, Tare1* und *GrossWeight*, dem Eigengewicht der Wiegeeinheit.

SWNo. x7: *HighLevel*: Bei Überschreiten dieses Wertes, wird das Flag

HighAlarm gesetzt.

SWNo. x8: *LowLevel*: Bei Unterschreiten dieses Wertes, wird das Flag *LowAlarm* gesetzt.

SWNo. x9: *ChConfig* ist ein Record in dem alle möglichen Konfigurationen des Channelsvorgenommen werden. Die erste Teilvariable dieses Records ist *Enablebit*, in dem die Bit 0-7 ähnliche Bedeutung wie beim Analog Input Channel haben.

An zweiter Stelle in diesem Record steht die Teilvariable *Functions*, durch die die eigentliche Funktion des Channels und die Abtastperiode festgelegt wird:

$0x	Channel nicht benutzt
$1x	Genauigkeit: hochpräzise
$2x	Genauigkeit: Industriestandard
$3x	Belt-Weight-Modus aktiv
$4x	Puls Input für Belt-Weight-Modus aktiv
$5x	Belt-Weight-Modus mit konstanter Pulsfrequenz

Die Abtastperiode kann zwischen 0.1 und 5 s eingestellt werden.

SWNo. xA: *Factors* beinhaltet die Auflösung im Wiegemodus, für die Durchflußmenge, und einen speziellen Skalierungsfaktor für den Belt Weight Mode.

SWNo. xB: *FullScale* ist der Wert der maximal wägbaren Masse.

SWNo. xC: *ZeroPoint* ist der Nullpunkt für das Nettogewicht *NetWeight*.

SWNo. xD: *Maintenance* beinhaltet Datum und Code der letzten Wartung.

SWNo. xE: *ChType* besteht im wesentlichen aus einer Reihe von Flags, die die benutzten Funktionen angeben.

SWNo. xF: *ChError* beinhaltet Errorbits wie: *SignalHigh, LowAlarm, ModuleError,...*

Bevor in späteren Abschnitten Channels von einer praxisorientierten Seite betrachtet werden, erfolgt im folgenden eine kurze Einführung in die Programmiersprache PROCESS-PASCAL und spezielle Erläuterungen ihrer Unterschiede zu ISO-PASCAL.

2.3 PROCESS-PASCAL

Die Programmierung der einzelnen P-NET-Knoten erfolgt in der Hochsprache PROCESS-PASCAL, das im wesentlichen vom weit verbreiteten ISO 7185 STANDARD PASCAL abgeleitet ist. Die wichtigsten Erweiterungen dazu sind:

- Multitaskingfähigkeit
- Erweiterte Standarddatentypen (Interface, Channel und Buffer)
- Prozeduren zur Interaktion an einem LC-Display (Update,...)

2.3.1 Multitasking

Als *Multitasking* bezeichnet man die Fähigkeit, mehrere Unterprogramme zur gleichen Zeit auf derselben CPU laufen zu lassen. Diese Unterprogramme, die dazu dienen können eine Tastatur abzufragen, einen Meßwert zu überwachen o.ä., nennt man *Task*. Da diese jedoch nicht wirklich parallel bearbeitet werden können, wird zwischen den einzelnen Tasks umgeschaltet.

Am besten erfolgt dies an einem Punkt des Programmablaufs, an dem eine Eingabe, das Ablaufen eines Timers o.ä. gewartet wird, so können unnötige Leerläufe vermieden werden [PP91]. Das Wechseln des Tasks erfolgt mit der Standardprozedur CHANGETASK. Das Anhalten des Programmablaufes und der spätere Wiedereinstieg ist aufgrund folgender Eigenschaften gewährleistet:

- ein Task hat seinen eigenen Speicherbereich
- ein Task hat seine eigenen *Program Counter* und *Stack Pointer*
- ein Task kann durch **keinen** Befehl aufgerufen werden

2.3.1.1 Cyclic-Task

Abb. 9 zeigt die Bearbeitung von vier Cyclic-Tasks, also Tasks die zyklisch bearbeitet werden. Die Abfolge der Bearbeitung ist durch die Reihung der Tasks im Quelltext des Programmes bestimmt. Wie zu sehen ist sind die Teilbearbeitungszeiten der verschiedenen Tasks von Fall zu Fall verschieden, da der CHANGETASK-Befehl an verschiedenen Stellen des Quelltextes eingefügt werden kann, um die CPU optimal auszunutzen.

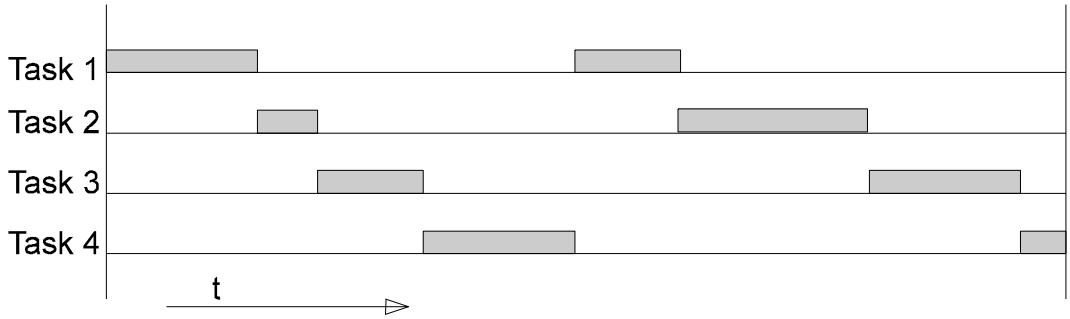


Abb. 9: Cyclic Tasks

Außer dieser einfachsten Form, des Cyclic-Tasks, besteht noch die Möglichkeit, einen Task als Timedinterrupt-Task oder Softwireinterrupt-Tasks aufzurufen.

2.3.1.2 Timedinterrupt-Task

Wie der Name schon andeutet wird ein Timedinterrupt-Task immer in vom Programmierer festgelegten Zeitintervallen ausgeführt. Die Angabe dieser Zeitspanne erfolgt in Sekunden mit einer Auflösung von 1/128 s.

2.3.1.3 Softwireinterrupt-Task

Der Softwireinterrupt-Task wird bei Auftreten eines bestimmten Ereignisses aufgerufen. Um in diesem Fall den richtigen Task zum Auslesen des Tastaturcodes zu starten, werden eine entsprechende SWNo. und der gewünschte Softwireinterrupt-Task mit derselben Interruptnummer versehen. Erfolgt nun ein Zugriff auf diese SWNo. wird der Task aufgerufen. Ob dies durch einen Schreib- oder einen Lesevorgang erfolgt muß vom Programmierer festgelegt werden.

Ein klassisches Beispiel dafür ist ein Task zum Abfragen einer Tastatur. In diesem Fall wird, falls zur Zeit ein CYCLIC TASK bearbeitet wird, dieser unterbrochen und nach Übergabe des Tastaturcodes eine entsprechende Aktion ausgelöst.

2.3.1.4 Prioritäten im Multitasking

Softwireinterrupt- und Timedinterrupt-Tasks sind prizipiell mit einer höheren Priorität als Cyclic-Tasks versehen.

Meldet sich während der Bearbeitung eines Cyclic-Tasks ein Softwireinterrupt- oder ein Timedinterrupt-Task an, so wird der Cyclic-Task mittels eines erzwungenen CHANGETASK unterbrochen und der neue Task bearbeitet. Ist dieser beendet (z. B. durch ein CHANGETASK im Quellcode des Programms) wird der ursprüngliche Task am Punkt der Unterbrechung wieder fortgesetzt.

In Abb. 10 ist solch ein Fall dargestellt. Zu Beginn wird Task 1 bearbeitet. Bei erreichen des CHANGETASK-Befehles wird übergewechselt auf Task 2 und auf gleiche Weise auf Task 3 und wieder auf Task 1. Während der neuerlichen Bearbeitung von Task 1 tritt nun ein Interrupt auf. D. h. Task 1 wird unterbrochen und der Interrupt wird behandelt. Anschließend wird Task 1 bis zum erreichen des CHANGETASK-Befehles weiterbearbeitet.

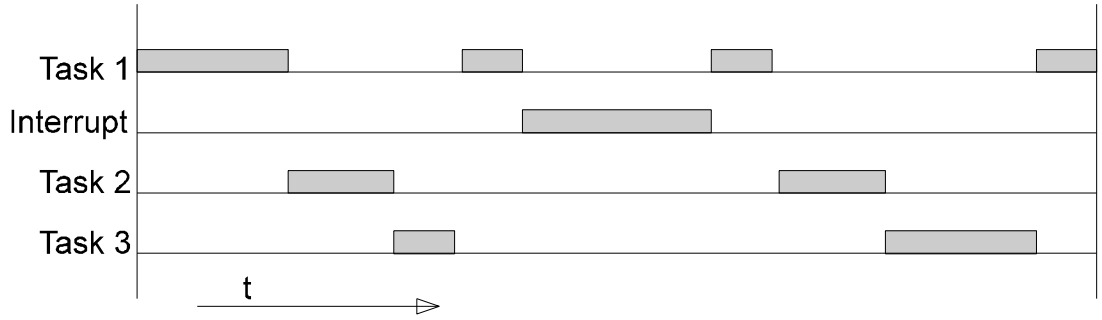


Abb. 10: Cyclic-Task unterbrochen von Softwire- bzw. Timedinterrupt-Tasks

Softwireinterrupt- und Timedinterrupt-Tasks haben die gleiche Prioritätsstufe und können sich daher nicht gegenseitig unterbrechen.

Bei der Aufspaltung eines Gesamtproblems in einzelne Tasks, ist zu beachten, daß diese (quasi)parallel ablaufen. Es ist daher sinnvoll Aufgaben die andauernd durchgeführt werden müssen als Task zu formulieren z. B. Abfrage der Tastatur, Anzeige auf einem Monitor oder LC-Display oder die Überwachung eines Grenzwertes der Produktionsanlage. Tasks die sequentiell arbeiten sollen, müssen mittels Flags miteinander kummunizieren (siehe auch *ANHANG B: HAUPTPROGRAMM*).

2.3.2 Programmaufbau

Dies ist der typische Aufbau eines PROCESS-PASCAL-Programmms mit 2 Tasks (Abb. 11).
Als erste Zeile jedes Programmes ist der Programmname anzugeben. Darauf folgt die Deklaration von Konstanten, Typen und Variablen. Auf globale Variablen und Konstanten kann von jedem Task zugegriffen werden, dies ermöglicht einen Datenaustausch der Tasks mit der „Außenwelt“.

Als nächstes folgen im Quellcode globale Prozeduren, die von jedem Task aufgerufen werden können. Da alle Tasks parallel bearbeitet werden, ist es natürlich auch möglich, daß eine globale Prozedur mehrmals simultan mit jeweils verschiedenen Parametern aufgerufen wird [PP91].

Variablen und Prozeduren können auch lokale Gültigkeit haben, also nur innerhalb eines Tasks programmiert und angesprochen werden.

```
Program name;

var
    globale Variablen

Procedure
    globale Proceduren

    Task Name_1;
    var
        lokale Variablen

    begin

    end;

    Task Name_2;
    var
        lokale Variablen

    begin

    end;

end.
```

Abb. 11: Aufbau eines PROCESS-PASCAL-Programmmes

Natürlich besteht auch die Möglichkeit Programmfragmente erst beim compilieren einzubinden, d.h. über eine Compileranweisung einen in einem anderen Quellcodefile abgelegten Programmteil zu inkludieren. Besonders nützlich ist diese Möglichkeit für das Einbinden bereits vorhandener Interface- und Channel-Deklarationen, wie in *2.3.4 Variablen im P-NET* gezeigt wird.

2.3.3 Datentypen

Neben den aus ISO-PASCAL bekannten Standarddatentypen wie INTEGER, REAL, BOOLEAN, STRING, usw... gibt es in PROCESS-PASCAL im wesentlichen 3 weitere Datentypen:

- BUFFER
- CHANNEL
- INTERFACE

Wie der Name schon besagt hat Eine Variable vom Typ BUFFER die Funktion eines Zwischenspeichers. In dem nach dem FIFO Prinzip jeweils ein Element pro Operation gelesen bzw. geschrieben werden kann. Das auslesen eines leeren BUFFERs und das schreiben in eine vollen BUFFER hat eine Errormeldung zufolge.

Der Datentyp CHANNEL wurde in Kapitel 2.2. bereits ausführlich beschrieben.

INTERFACE ist eine allen anderen Typen übergeordnete Struktur, d. h. sie kann, ähnlich einem Record, aus allen restlichen Typen, auch CHANNEL, zusammengesetzt sein. Wobei jeder Einfache Datentyp mit einer *SWNo.* assoziiert wird und einem Feld vom Typ CHANNEL 16 *SWNo.´*s zugeordnet sind [PP92].

2.3.4 Variablen im P-NET

Um auf Variablen, in einem beliebigen P-NET-Knoten zugreifen zu können, muß deren physikalischer Ort deklariert werden. Die dazu nötigen Bestimmungsstücke sind ein zuvor definiertes Interface Modul, die P-NET-Adresse und Nummer des P-NET-Ports, hier dargestellt an hand eines Beispiels in Abb. 12.

Das Include File PDMODULE.DEF sollte natürlich schon vom Hersteller des benutzten P-NET Knotens mitgeliefert werden und braucht dann lediglich auf die in Abb. 12 dargestellte Weise eingebunden werden.

P-NET-PortNr. gibt im Falle eines Multiport-Masters an, an welchem P-NET-Port der Knoten mit der entsprechenden Funktion lokalisiert ist. Das darauffolgende Adressfeld hat den Aufbau wie er in *2.1.3.1 Dualport Master* beschrieben ist, wobei das $-Symbol eine Zahl in hexadezimaler Darstellung kennzeichnet.

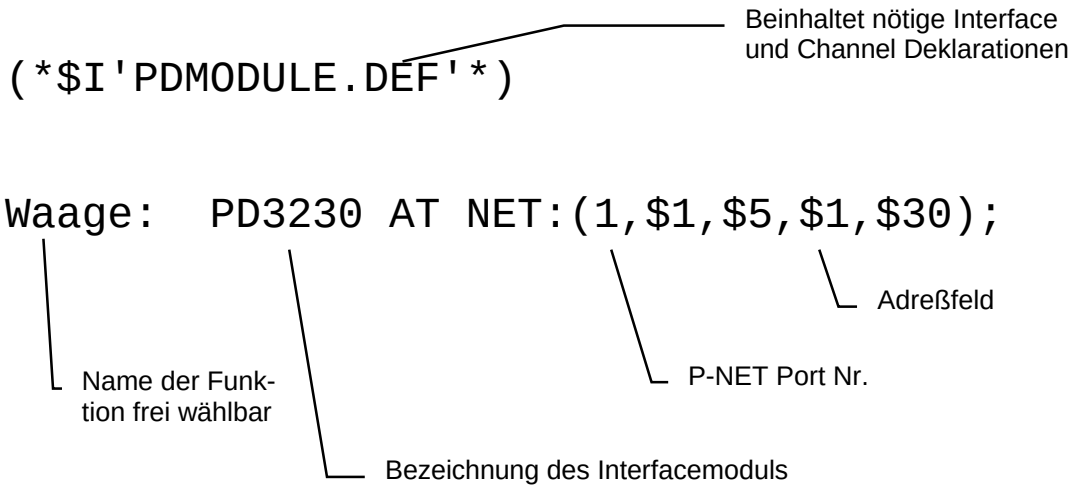


Abb. 12: Lokalisierung einer Funktion

Weiters besteht die Möglichkeit, eine Variable indirekt zu deklarieren. Das bedeutet, daß einer Variablen ein anderer (verständlicherer und kürzerer) Name zugeordnet werden kann, ohne ihr einen anderen Speicherplatz zuzuweisen. Durch diese Zuordnung nimmt die neue Variable automatisch den Typ der ursprünglichen Variable an. Ein Beispiel dazu wird in Abb. 13 erläutert.

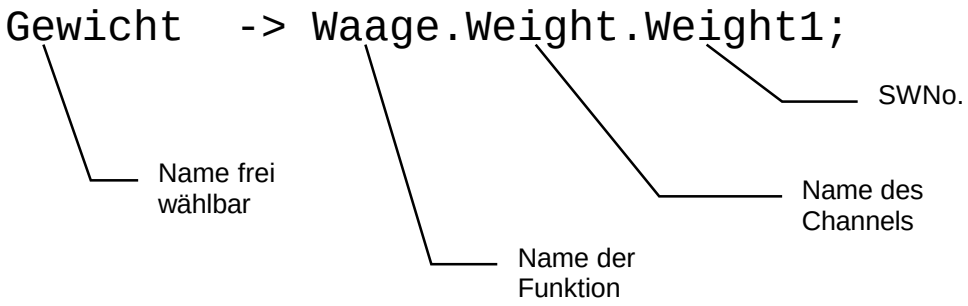


Abb. 13: Indirekte Deklaration

In der weiteren Programmierung kann nun die Meßgröße **Gewicht**, wie jede normal deklarierte Variable behandelt werden, ohne sie bei jeder Behandlung mit **Waage.Weight.Weight1** benennen zu müssen [PP92].

2.4 Vergleich mit anderen Feldbussystemen

Zum Abschluß des theoretischen Teiles, werden nun noch einige weitere Feldbussysteme kurz vorgestellt und anschließend nach verschiedenen Kriterien mit P-NET verglichen.

2.4.1 Grundkonzepte verschiedener Feldbussysteme

2.4.1.1 INTERBUS-S

INTERBUS-S wurde 1985 von der Firma Phoenix Contact als Single-Master-System entwickelt in weiterer Folge für die Normung als Sensor/Aktorbus spezifiziert. Definierte Reaktionszeiten sind bei diesem System durch Ring-Topologoie und sogenannte Summenrahmen garantiert. Beim Summenrahmenverfahren gibt der Master eine Zeitschlitzstuktur vor, in der jeder Slave eine bestimmte Menge von Daten schreiben kann [BON92, FBu92].

2.4.1.2 BITBUS

Der BITBUS wurde im Jahr 1983 von der Firma Intel für die Kommunikation zwischen Mikrokontrollern bzw. speicherprogrammierbaren Steuerungen (SPS) und Robotern konzipiert. Er gewährleistet den Informationsfluß zwischen einem Leitrechner und unterschiedlichen Produktionssegmenten und ermöglicht damit die Koordination von Fertigungszellen. Daraus folgt die Anforderung große Datenmengen bei geringer Echtzeitanforderung zu bewältigen [BON92, FBu92].

2.4.1.3 CAN

CAN (Controller Area Network) wurde von den Firmen Intel und Bosch ursprünglich für den Einsatz im Automobil entwickelt, um den Aufwand an Sensor- und Aktorleitungen zu verringern. Mittlerweile hat sich CAN auch im Maschinenbau etabliert, was auch dafür zurückzuführen ist, daß seit längerem die nötigen Controller Chips für die unteren Protokollschichten (OSI Modell Schicht 1 und 2) von verschiedenen Herstellern angeboten werden [BON92, FBu92].

2.4.1.4 PROFIBUS

Grundsätzlich wird bei diesem System unterschieden zwischen aktiven und passiven Teilnehmern. Aktive Teilnehmer (Master) dürfen, wenn sie im Besitz der Buszugriffsberechtigung sind, über den Bus verfügen, d.h. sie können Nachrichten ohne externe Aufforderung aussenden. Passive Teilnehmer (Slaves) dürfen lediglich empfangene Signale quittieren, oder auf Anfrage eines Masters eine Nachricht an diesen übermitteln.

Das Grundkonzept, dem von P-NET ähnlich, wird hier jedoch zwecks Vernetzung von speicherprogrammierbaren Steuerungen, Automatisierungssystemen, Sensoren und Aktoren, in einer wesentlich komplexeren Form als im P-NET umgesetzt [BON92, FBu92].

2.4.2 Vergleich

2.4.2.1 Adressierungsarten

Die vorgestellten Bussysteme verwenden zum Teil sehr unterschiedliche Methoden, um die Busteilnehmer beziehungsweise Datenpakete zu adressieren.

Wie unter *2.1.2.4 Erstellen und Erkennen von Datenpaketen* beschrieben, verwendet P-NET verschiedene Adreßtypen. Je nachdem, ob sich der Empfänger am selben P-NET-Bus befindet oder nicht, wird die entsprechende Adressierungsart gewählt. Bei einer Übertragung über einen oder mehrere P-NET-Busse enthält die Empfänger/Sender-Identifikation mehrere Bytes.

Beim BITBUS steht sowie beim PROFIBUS-FMS jeweils ein Byte zur Teilnehmeradressierung zur Verfügung, um 127 (PROFIBUS-FMS) bzw. 250 (BITBUS) Slaves zu adressieren [GRU95].

Der ASI adressiert seine Slaves auf zwei Arten. Erstens durch Masteraufruf, wobei hier wieder zwischen einem Adressieraufruf und der automatischen Adressierung unterschieden werden muß. Beim Adressieraufruf muß ein Slave nach dem anderen angeschlossen und mit einer neuen Adresse versehen werden, und bei der automatischen Adressierung können bei laufendem Betrieb Komponenten ausgetauscht werden, wobei danach jeder Teilnehmer eine neue Adresse zugeteilt bekommt. Zweitens gibt es ein eigens für den ASI entwickeltes Adressiergerät.

Im INTERBUS-S System hat jede Station eine Identifikationsnummer, die sich aus der physikalischen Position im Ring ergibt, mittels welcher der Master seine Slaves anspricht.

Der CAN dagegen adressiert objektorientiert, wobei maximal 2032 Objekte, also Systemteilnehmer adressierbar sind [GRU95].

2.4.2.2 Übertragungsraten

Für die Höhe der Datenübertragungsraten sind bei den meisten Bussystemen Leitungslänge und Zuverlässigkeitsüberlegungen die entscheidenden Kriterien. Je länger also die Leitung, desto geringer wird bei vielen Feldbussystemen die übertragbare Datenmenge. Entsprechend stark varieren daher auch die in der Literatur angegebenen Bereiche für die maximal möglichen Datenübertragungsraten.

Besonders ausgeprägt ist dies zum Beispiel beim CAN, wo Transferraten von 1 Mbit/s bei einer Netzausdehnung von maximal 40 m bis zu 10 kbit/s bei einer Leitungslänge von 1000 m möglich sind.

Bei PROFIBUS-FMS wird bei einer Netzausdehnung von maximal 200 m (ohne Repeater) eine Transferrate von bis zu 500 kbit/s verwendet. Bei einer reduzierten Übertragungsrate von 93,75 kbit/s sind 1200 m Ausdehnung möglich.

Anders die Situation bei P-NET, wo wie bei ASI und INTERBUS-S die Übertragungsraten unveränderlich sind (P-NET: 76,8 kbit/s, ASI: 150 kbit/s, INTERBUS-S: 300 kbit/s Peripheriebus, 500 kbit/s Fernbus). Eine solch strenge Spezifizierung erspart eine aufwendige und fehleranfällige Konfiguration des Systems vor Ort.

2.4.2.3 Maximaler Knotenabstand

Beim folgenden Vergleich des maximalen Teilnehmerabstandes der behandelten Feldbusse wurde für den Fall, daß mehrere Übertragungsmedien möglich sind, die RS-485-Norm als Vergleichsmedium gewählt.

Hinsichtlich des maximalen Abstandes zweier Netzkomponenten liegt P-NET mit seinen maximal 1,2 km Abstand gemeinsam mit dem PROFIBUS-FMS und dem BITBUS an der Spitze. Die eindeutig geringste Entfernung zwischen zwei Komponenten erlaubt der ASI, bei dem ein Maximum von 100 m vorgesehen ist. Beim INTERBUS-S sind 400 Meter erlaubt, beim DIN-Meßbus 500 m.

2.4.2.4 Buszugriff

Die verschiedenen Feldbussysteme benutzen unter anderem aufgrund ihrer Topologie unterschiedliche Busverwaltungs-Algorithmen. So verwenden z. B. P-NET und PROFIBUS-FMS Tokenpassing, bei CAN findet eine Priorisierung statt. Im folgenden sollen die Unterschiede kurz behandelt werden.

Eine Spezialität von P-NET ist, wie bereits erwähnt, das *virtual Tokenpassing*, eine selbständige Optimierung des Buszugriffes, falls mehrere Master an einen P-NET-Bus geschaltet sind.

Im Gegensatz dazu wird beim PROFIBUS-FMS der Token in definierten Zeitabständen von einem Master zum nächsten weitergereicht, ohne Rücksicht darauf, ob dieser auf den Bus zugreifen möchte, oder nicht.

CAN benutzt eine andere Methode der Buszuteilung, nämlich ein Arbitrierungskonzept, in dem jedem Master eine Objekt-Priorität zugeordnet ist, die über den Buszugriff bestimmt. Das heißt jener Master, der die höchste Identifier-Priorität hat, gewinnt die Arbitrierung und den Buszugriff, sodaß seine Nachricht ohne Zeitverlust gesendet wird. Diese Prioritätsvergabe stellt dementsprechend hohe Anforderungen an die Projektierung, um Echtzeitfähigkeit zu gewährleisten.

2.4.2.5 Fehlererkennung

Tritt bei der Datenübertragung innerhalb eines Bussystems ein Fehler auf, so verwenden die einzelnen Bussysteme zum Teil unterschiedliche Mechanismen zur Erkennung desselben. Eine allgemein übliche Methode, der *Cyclic Roundary Check* (CRC) findet sowohl beim BITBUS und PROFIBUS-FMS Anwendung. Auch CAN und der INTERBUS-S verwenden diesen Algorithmus, doch verfügt der INTERBUS-S noch zusätzlich eine Längenverivizierung und CAN über weitere vier Fehlererkennungs-Mechanismen. Der DIN-Meßbus und P-NET benutzen eine *Checksum* (P-NET 2 Byte, DIN-Meßbus 1 Byte). ASI verwendet eine komplizierte Kodierung, die ein ganzes Set von Regeln zur Fehlererkennung implementiert.

Die **Hammingdistanz** ist eine Maßzahl für die Güte der einzelnen Fehlererkennungsmechanismen. Sie beschreibt, wieviele Bit einer Nachricht falsch sein können, um vom System als Fehler erkannt zu werden. Eine hohe Hammingdistanz ist also ein Indiz für ein gutes Datensicherungsverfahren.

P-NET hat wie auch der PROFIBUS-FMS, der BITBUS und der INTERBUS-S zur Hammingdistanz 4 und kann bis zu 3 gleichzeitige Bitfehler erkennen. Der DIN-Meßbus hingegen kann lediglich 1-Bit Fehler erkennen und hat die Hammingdistanz 2. CAN hingegen verwendet wie oben erwähnt ein sehr aufwendiges Fehlererkennungsvervahren und wird allgemein als sehr zuverlässig betrachtet. Seine Hammingdistanz liegt bei 6. Für den ASI wird in der Literatur meist ein Wert von 2 angegeben, wobei aufgrund seines unüblichen Datensicherungsmechanismus dies aber nicht wirklich aussagekräftig ist.

2.4.2.6 Ausfall eines Masters

In Systemen, die wie P-NET multimasterfähig sind (PROFIBUS-FMS, CAN), kann es zum Ausfall eines oder mehrerer Master kommen, wobei die einzelnen Feldbussysteme unterschiedlich darauf reagieren. Der PROFIBUS-FMS erkennt den Ausfall eines oder mehrerer Master und optimiert daraufhin die Busvergabe, indem der Token an diesen

nicht mehr weitergereicht wird. Bei P-NET hingegen wird ein Master-Ausfall nicht direkt wahrgenommen, sondern das System reagiert, als ob der Teilnehmer sein Senderecht nicht in Anspruch nimmt und reicht den Token nach den üblichen 10 Takten Wartezeit weiter.

2.4.2.7 Tabellarischer Vergleich

In Tab. 6 erfolgt nun zum Abschluß eine überblicksweise Gegenüberstellung der wichtigsten Daten der eben behandelten Bussysteme.

	P-NET	PROFIBUS	BITBUS	INETRBUS-S	CAN
Medium	2-adrig geschirmt	2-adrig geschirmt	2-adrig geschirmt	2-adrig geschirmt	frei wählbar
Topologie	passiver RIng	Linie	Linie	Ring	frei wählbar
Teilnehmerzahl	125	32	28	32	2032
Übertragungsrate	76,8 kBit/s	9,6 ... 500 kBit/s	62,5 ... 375 kBit/s	300 kBit/s	10...1000 kBit/s
Segmentlänge	1200 m	1200 m, 3000 m	300 m, 1200 m	400 m	40...1000 m
Busverwaltung	Virtual Token Passing	Token Passing	Polling	festes Zeitfenster	Bitweise Arbitration

Tab. 6: Vergleich verschiedener Systeme

3 Die Übungsanlage

In diesem Abschnitt werden zuerst grundlegende Überlegungen zum Thema Laborübung angestellt. Daraufhin folgt eine detaillierte Beschreibung der Anlage und deren Betriebssicherheit.

3.1 Anforderungen an die Laborübung

Das Ziel dieser Laborübung ist es, dem Studierenden einen Einstieg in die Technologie und die Handhabung von Feldbussystemen zu vermitteln. Aufgrund des übersichtlichen Aufbaus und der einfachen Programmierung erscheint P-NET für diesen Zweck besonders geeignet.

Aus persönlicher Erfahrung und durch Diskussionen mit Studienkollegen und Assistenten wurde versucht, einige für den Erfolg einer Laborübung wichtige Begriffe in Abb. 14 gegenüberzustellen.

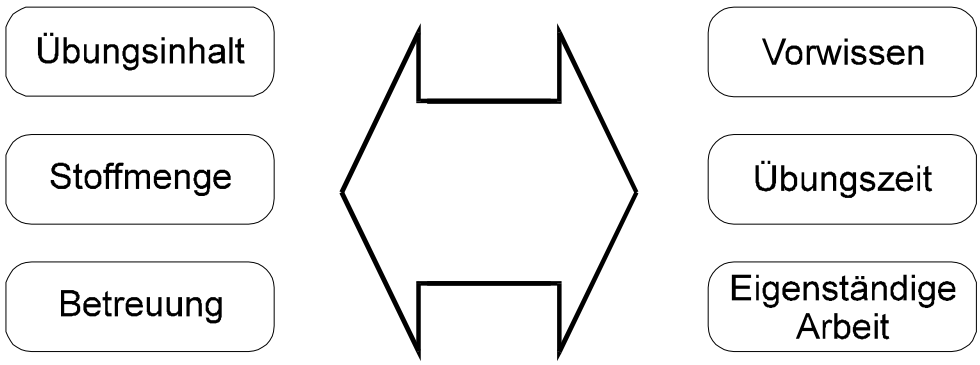


Abb. 14: Anforderungen an eine Laborübung

- Der **Übungsinhalt** sollte dem Wissensstand des Studierenden angepaßt sein, oder umgekehrt gesagt muß ein bestimmtes Grundwissen bereits **vor** der Lektüre des Übungsskriptums vorauszusetzen sein.
- Das Verhältnis zwischen der **Stoffmenge** und zur Verfügung stehender **Zeit**, muß richtig gewählt sein. Ein hektisches Übungsklima, bei dem keine Zeit für eigene Überlegungen bleibt, führt eher zu Frust als zu befriedigendem Lernerfolg.

- Der Student sollte nicht ohne **Betreuung** einfach auf die Übungsanlage losgelassen werden. Andererseits sollte nicht jeder Handgriff vorgegeben sein, da es sonst zu keinem **eigenständigen Arbeiten** kommt.

3.2 Anforderungen an die Anlage

Folgende Forderungen lagen der Planung dieser Anlage zugrunde:

- Anschaulichkeit für Studenten
- Fexibilität in der Anwendung
- Erweiterbarkeit

Der Wunsch nach **Anschaulichkeit** resultiert aus der Erfahrung, daß ein beweglicher Teil, oder wie in diesem Fall eine Pumpe und verschiedene Ventile, auf den Studierenden wesentlich motivierender wirken, als z. B. eine LED oder eine Voltmeteranzeige.

Fexibilität in der Anwendung bedeutet, daß auf ein und derselben Anlage ohne Veränderung der Hardware verschiedene Versuche und Übungen durchgeführt werden können.

Aus der erwünschten Zusammenarbeit mit weiteren P-NET-Anbietern resultiert schließlich die Forderung nach **Erweiterbarkeit** der bestehenden Anlage zu einer sogenannten Multivendoranlage, also der Verbindung von Modulen verschiedener Hersteller

3.3 Aufbau der Anlage

Auf der folgenden Seite sind in Abb. 15 der schematische Aufbau der Anlage und dazu in Tab. 7 die verwendeten Abkürzungen dargestellt.

V1-V5	Magnetventile
P	Förderpumpe
M	Motor für Rührwerk
H	Heizstab 1000W
T1	Temperaturfühler im oberen Becken (Becken 1)
T2	Temperaturfühler im unteren Becken (Becken 2)
L1	Levelschalter im oberen Becken
L2	Levelschalter im unteren Becken

Tab. 7: Abkürzungen für Sensoren und Aktoren

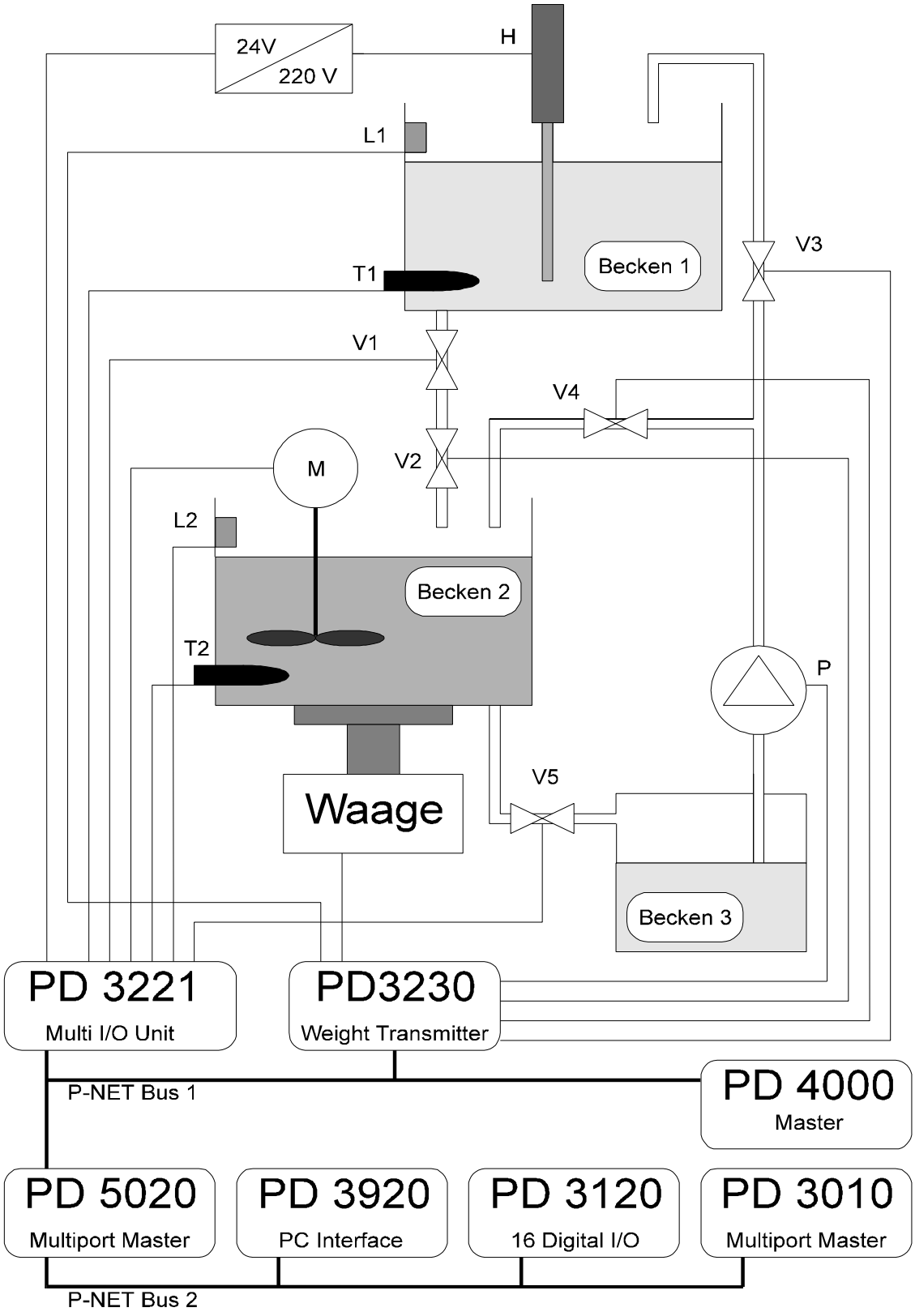


Abb. 15: Schematischer Aufbau der Anlage

3.3.1 P-NET-Bus 1

Durch P-NET-Bus 1 sind die folgenden 4 Module verbunden:

- PD 3221 Multi I/O
- PD 3230 Weight Transmitter
- PD 4000 Master
- PD 5020 Multiport-Master

Das **Becken 1** wird durch **Pumpe P** und **Ventil V3** aus dem Vorratsbehälter **Becken 3** bis zum **Level L1** befüllt und durch den im PD 3221 Modul eingebauten PID-Regler, mittels eines 1000W **Heizstabes H** auf einer bestimmten Temperatur gehalten. Wegen der hohen Leistung kann der Heizstab natürlich nicht direkt an das P-NET-Modul angeschlossen werden, sondern muß über einen elektronischen Schalter an das 220V Netz gekoppelt werden.

Becken 2 simuliert einen chemischen Prozeß. Zu der (kalten) Flüssigkeit werden über die **Ventile V1** und **V2**, in mehreren Schritten bestimmte Mengen aus **Becken 1** hinzugefügt und mit **Motor M** gerührt. **V1** und **V2** stellen durch ihre Serienschaltung ein redundantes System dar.

Weiters bestehen die Möglichkeiten, einen Teil der Flüssigkeit über das **Ventil V5** in das **Becken 3**, das in dieser Anordnung Quelle und Senke in einem darstellt, abzulassen bzw. über das **Ventil V4** kaltes Wasser aus **Becken 3** in **Becken 2** zu Pumpen.

In Tab. 8 ist die Zuweisung der Knoten bzw. der Channels dargestellt.

P-NET-Knoten	Channel Name	Funktion
PD 3221:	Digital_IO_1	Ventil V1
	Digital_IO_2	Ventil V5
	Digital_IO_3	Heizung
	Digital_IO_4	Mixer
	Digital_IN_1	Level Becken 2
	Analog_in_1	Temperatur Becken 1
	Analog_in_2	Temperatur Becken 2
PD 3230:	Digital_IO_1	Ventil V2
	Digital_IO_2	Ventil V3
	Digital_IO_3	Ventil V4
	Digital_IO_4	Pumpe
	Digital_IN_1	Level Becken 1
	Weight	Gewicht Becken 2

Tab. 8: Aufteilung der Channels

3.3.2 P-NET-Bus 2

Es wäre ohne weiteres möglich gewesen alle zur Verfügung stehenden P-NET-Module über einen P-NET-Bus zu verbinden, um jedoch die speziellen Möglichkeiten eines Multiport-Masters kennenzulernen, wurde ein zweiter P-NET-Bus realisiert. Dieser verbindet folgende Knoten:

- PD 5020 Multiport-Master
- PD 3920 PC-Interface
- PD 3120 Digital I/O
- PD 3010 Multiport-Master

Das PD 3120 Modul mit 16 digitalen Ein- und Ausgängen ist in Verbindung mit dem PD 3010 Master besonders für erste Versuche bei einer Laborübung gut geeignet, da hier getrennt vom relativ komplexen Flüssigkeitskreislauf der Anlage verschiedene Funktionen ausprobiert werden können.

Der PD 5020 Controller hat einerseits die Aufgabe eines Multiport-Masters, zum anderen läuft auf ihm eine Visualisierungssoftware, die Daten aus allen P-NET-Bussen graphisch darstellen kann.

Für einen weiteren Ausbau der Übungsanlage ist es natürlich möglich, einen dritten P-NET-Bus über den Multiport-Master PD 3010 an das System anzukoppeln.

3.4 Betriebssicherheit und Grenzen der Anlage

Im folgenden werden verschiedene Aspekte der Grenzwertüberschreitung und Sicherheit für Benutzer und Anlage behandelt.

3.4.1 Elektrische Spannungen

Die gesamte Anlage, inklusive Ventile und Pumpe, wird mit den für P-NET typischen 24 V Gleichspannung betrieben. Die einzige Ausnahme stellt der Heizstab dar, der über ein Relais im oberen Teil der Anlage an das 220 V Netz gekoppelt ist. Dieser Bereich ist jedoch gut isoliert und wäre vom eventuellen Überlaufen eines Beckens nicht betroffen. Es besteht somit keine Gefahr, daß der Benutzer mit 220 V in Berührung kommt.

3.4.2 Elektrische Ströme

Ein P-NET-Digital-IO wie in Abb. 16 dargestellt, liefert laut [PD30] am Punkt IN/OUT einen maximalen Ausgangsstrom von 1 A. Rührwerk und Motor überschreiten diese Grenze nicht, auch die Pumpe nimmt im Dauerbetrieb bei der vorliegenden Förderhöhe von ca. 75 cm einen Strom von 0.6 A auf.

Die Grenze von 1 A wird jedoch vom Einschaltstrom der Pumpe überschritten, was aufgrund der im PD 3230 implizierten Schutzmaßnahmen das Einschalten der Pumpe unmöglich machen würde. Aus diesem Grund wurde zur Strombegrenzung ein 10 Ohm / 5 W Leistungswiderstand in Serie zur Pumpe geschaltet, was als unerwünschten Nebeneffekt eine geringere Förderleistung der Pumpe zufolge hat.

Eine Möglichkeit, die Pumpe dennoch mit maximaler Leistung zu betreiben, wäre es, nach dem Anlaufen des Motors einen zweiten digitalen Ausgang ohne Vorwiderstand an den Motor zu schalten. Da jedoch kein weiterer Ausgang zur Verfügung steht, muß auf diese Option verzichtet werden.

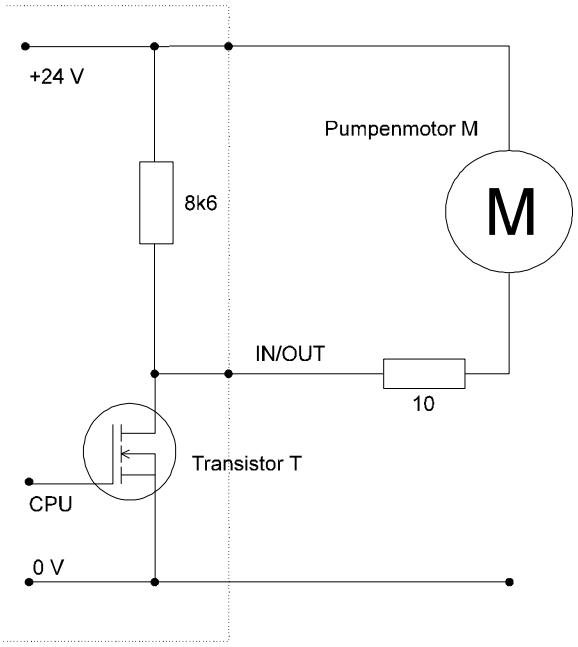


Abb. 16: Beschaltung des Digital-IO mit Pumpenmotor

Abb. 16 zeigt eine für den Digital-IO typische Beschaltung, bei der als Ausgang die Last von IN/OUT gegen +24V geklemmt ist.

Wird der Channel als Eingang genutzt ist der Punkt IN/OUT für low mit 0V zu verbinden [PD30, S. 4]. Auf diese Weise wurde bei der Übungsanlage mit den Schwimmerschaltern L1 und L2 die Levelüberwachung relisiert.

3.4.3 Thermische Belastung

In Hinsicht auf die thermische Belastung der Anlage sind folgende Punkte zu beachten:

- Pumpe und Ventile sind laut Lieferfirma für Temperaturen bis zum Siedepunkt des Wassers zu betreiben.
- Der Heizstab darf nur unter Wasser eingeschaltet werden, Betrieb an Luft führt in kürzester Zeit zur thermischen Zerstörung.
- Für das Behältermaterial (Plexiglas) und den dafür verwendeten Klebstoff werden vom Hersteller T_{max}=80 °C garantiert.
- Durch die schlechte Durchmischung im oberen Becken, treten bei 60 °C am Sensor in der Beckenmitte bereits Temperaturen um den Siedepunkt auf.

3.4.4 Überlauf der Becken

Während der bereits abgehaltenen Laborübungen, hat sich der Überlauf der Becken als größte Gefahr herausgestellt und ist unerwartet oft aufgetreten.

Obwohl durch die Progammierung der P-NET-Module dafür gesorgt war, daß die Bekken vor Überlauf geschützt sind, wurden von den Studenten der unter Punkt *4.4.5.2 Task 2: Levelüberwachung* programmierte Safety-Task verbotenerweise mit verschiedenen Begründungen deaktiviert. Dies führte in zwei Fällen zur totalen Überschwemmung des Arbeitsplatzes.

Ein anderes mögliches Konzept anstatt eines Überwachungstasks im PROCESS-PASCAL-Quellcode ist es, einen Master für die Übung zu verwenden und einen zweiten so zu programmieren, daß dieser im Notfall die richtigen Aktionen setzen kann. Bei dieser Variante besteht jedoch die Gefahr, daß zwei Master sich widersprechen und es infolgedessen zu rasch aufeinanderfolgenden Ein- und Ausschaltvorgängen kommen kann. Als quasi letzter Ausweg besteht noch die Möglichkeit, im Notfall manuell die Spannungshauptversorgung abzuschalten, was die Anlage wiederum in einen stabilen Zustand bringt.

Als beste Sicherung hat sich jedoch die Assembler-Programmierung der Slave-Module PD 3221 und PD 3230 erwiesen, wobei dafür zu sorgen ist, daß diese Programme immer aktiv sind. Mehr zum Thema Assembler-Programmierung wird in *3.5 PD Calculator Assembler* gebracht.
Sollte eines der P-NET-Module (speziell die Steckverbindungen) dennoch naß werden, ist die Anlage sofort abzuschalten, die Steckverbindungen sind zu trennen und zu trocknen. Diese Prozedur wurde bei einem, während des Aufbaus der Anlage, unbeabsichtigtem Überlauf von Becken 2 erprobt.

3.4.5 Wechseln der Flüssigkeit

Die einfachste Art der Befüllung ist, Leitungswasser über Becken 2 und geöffnetes Ventil 5 in den Kreislauf einzubringen. Je nach Verschmutzung ist das Wasser nach einigen Monaten zu wechseln. Die Idee ein Konservierungsmittel zu verwenden wurde aus Gründen des Aufwandes und eventuell auftretender Geruchsbelästigung wieder verworfen.

Für das Entleeren der Anlage ist ein manuell bedientes Kugelventil vorgesehen, das über ein Stück Schlauch das unter dem Wandverbau endet, direktes Entleeren von Becken 3 ermöglicht.

3.4.6 Ausfall der Versorgungsspannung

Bei einem Totalausfall der Versorgungsspannung werden alle Ventile geschlossen und Heizstab und Pumpe abgeschaltet. Die Anlage geht in einen stabilen Zustand über.

Fällt nur die Versorgungsspannung **eines** der beiden Slaves an P-NET-Bus 1 aus, so ist bis zu einem gewissen Grad durch die Verbindung der Meßleitungen gewährleistet, daß einer der Master über den verbleibenden Slave die Möglichkeit hat, die Anlage zu stabilisieren.

Wie aus Abb. 15 ersichtlich, sind im wesentlichen zwei redundante Systeme impliziert worden:

- V1 und V2 sind in Serie angeordnet, so daß eines der beiden Ventile im Störfall immer schließen kann.
- Der Füllstand bzw. das Gewicht in Becken 2 werden einerseits vom PD 3230 Weight Module andererseits vom PD 3221 überwacht.

Um die Reaktion des Systems in diesen Fällen erproben zu können, ist es möglich über 3 Schalter die Versorgungsspannung für den PD 3230, den PD 3221 und den Master PD 4000 getrennt zu schalten.

3.4.7 Unterbrechung der Busleitung

Auch um diesen Fall testen zu können, wurde in der Anlage wie in Abb. 17 dargestellt ein Schalter angebracht, mit dem P-NET-Bus1 auf Wunsch unterbrochen werden kann.

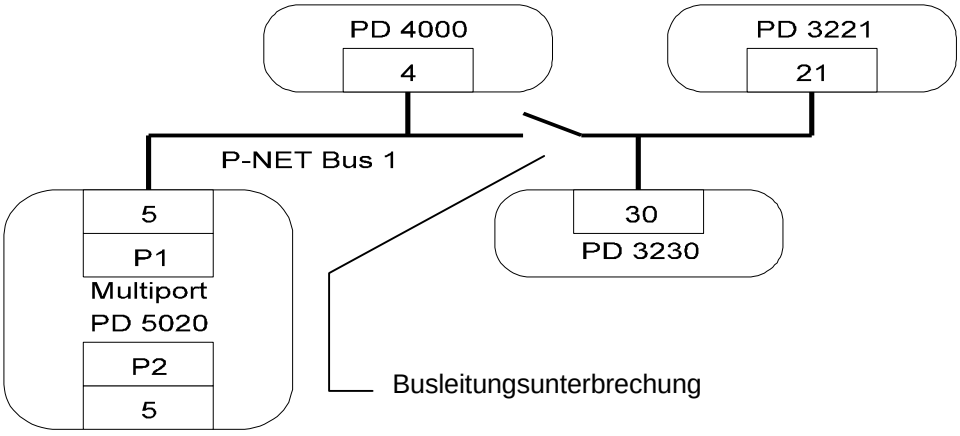


Abb. 17: Unterbrechung von P-NET-Bus 1

Wie bereits in 2.2.2 erläutert hat jedes P-NET-Slave-Modul eine sogenannte *Watchdog* Funktion [PD30, PD21, STA92]. Diese bewirkt bei Abtrennung vom P-NET-Bus, genauer gesagt bei längerem Ausbleiben des Requests eines Masters (siehe Abb. 17), den Übergang des Moduls in einen definierten Zustand.

Wird z. B. die vom Werk vorgegebene Programmierung nicht geändert, so führt das im Falle einer Abtrennung, nach 10 Sekunden zu einer Abschaltung aller Ausgänge. In manchen Fällen ist diese Funktion jedoch so zu programmieren, daß ein offenes Ventil, also ein eingeschalteter Ausgang den stabilen Zustand darstellt.

Wie bereits erwähnt besteht noch die Möglichkeit, für den Fall einer Busleitungsunterbrechung Slaves zu programmieren. Zu diesem Zweck muß dieser Slave über einen sogenannten *Calculator Channel* verfügen, dieser wird nicht in der Hochsprache PROCESS-PASCAL, sondern in einer Assemblersprache, dem sogenannten *PD Calculator Assembler* programmiert, der im nächsten Abschnitt behandelt wird.

3.5 PD Calculator Assembler

Diese integrierte Entwicklungsumgebung (Assembler, Debugger und Loader) bietet die Möglichkeit, Slaves mit Programmen zu laden, die sowohl im Normalbetrieb, als auch bei unterbrochener Busverbindung zum Master ihre Aufgabe erfüllen. Wobei die Ausführung eines Programmes nur bei gesetztem Flag *RunEnable* gewährleistet ist.

3.5.1 Möglichkeiten des Assemblers

Wie allgemein üblich, werden die Quellcode-Instruktionen in einem Textfile bearbeitet und abgelegt. Mit dem Assembler kann dieser Quelltext nun assembliert werden, d. h. es

wird der passende Maschinencode generiert. Dieser wird anschließend in dem für die Assemblerprogrammierung vorgesehenen *Calculator Channel* gespeichert.

Für die Funktionskontrolle und Fehlersuche besteht anschließend die Möglichkeit, die im *Calculator Channel* abgelegten Programme mit Hilfe von *Single Steps* und *Break Points* zu debuggen.

In Abb. 18 ist nun die Typische Benutzeroberfläche des Calculator Assemblers dargestellt, bei der sich die verfügbaren Funktionen großteils selbst erklären.

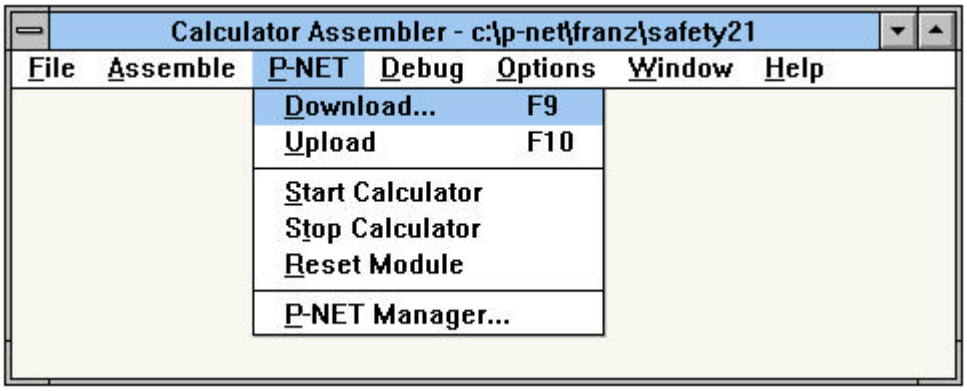
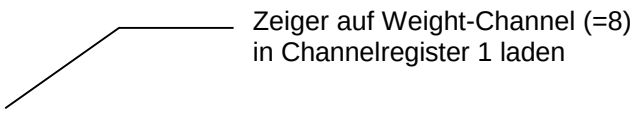


Abb. 18: Funktionen des Calculator Assemblers

Obwohl die Assemblerprogrammierung bereits etwas mehr Erfahrung im Umgang mit P-NET voraussetzt, sollen im folgenden dennoch anhand eines kurzen Programmes die prinzipiellen Möglichkeiten dieser Art der Programmierung aufgezeigt werden.

3.5.2 Programmierung

Aufgabenstellung: Der Weight Transmitter PD 3230 soll bei Überschreiten eines fix vorgegebenen Grenzwertes die Pumpe abschalten, und zwar sowohl im Normalbetrieb als auch im Falle einer Busunterbrechung.
Wie in Abb. 19 zu sehen, ist eine Besonderheit das sogenannte *Channel Register*, über das jeder Zugriff auf einen *Channel* bzw. auf eine spezielle *SWNo.* abgewickelt wird. Dazu muß im ersten Schritt ein Zeiger auf den Inhalt eines *Channels* in ein *Channel Register* (hier CR1) geladen werden. Um nun mit einer SWNo. z. B. einen Vergleich anzustellen, muß diese in den Akku geladen werden und kann dann verglichen werden (Abb. 19).



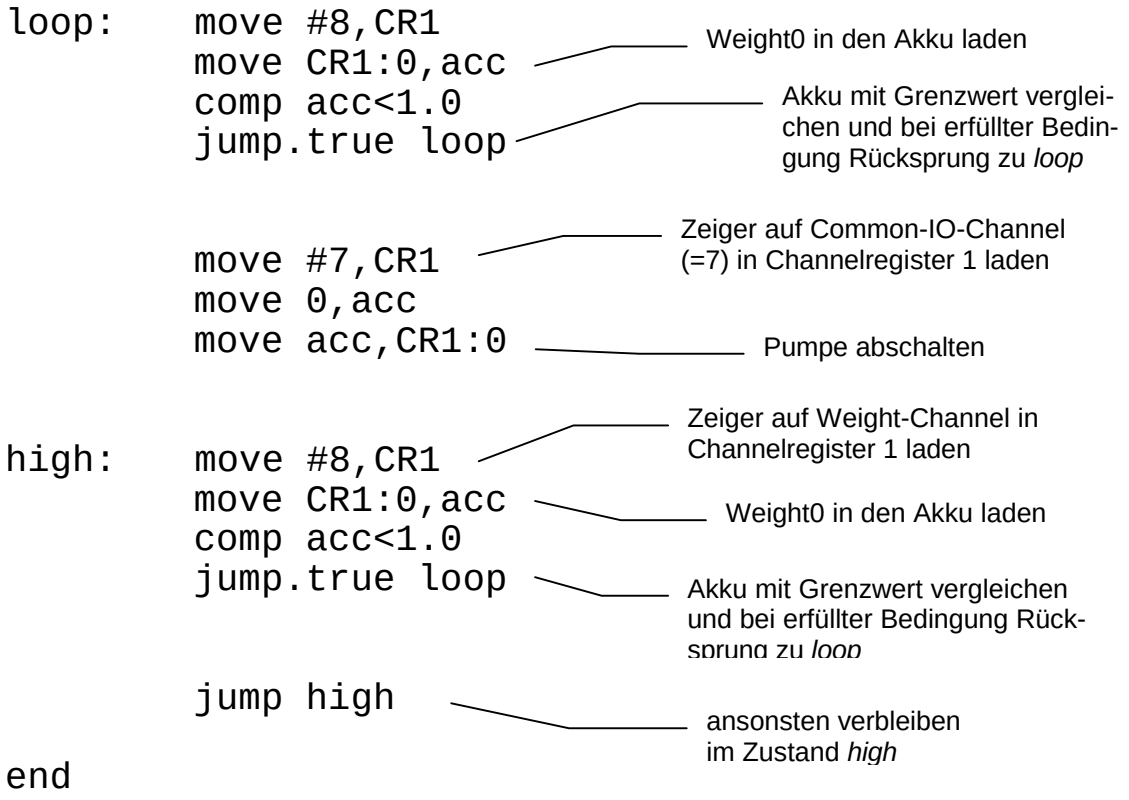


Abb. 19: Programmierung in PD Calculator Assembler

Wie bereits erwähnt muß, um ein Calculator-Assembler-Programm laufen zu lassen, im *Calculator Channel* des betreffenden Moduls das Flag *RunEnable* gesetzt sein. Dafür ist es jedoch nötig *WriteEnable* auf TRUE zu setzen, da sonst *RunEnable* nur im RAM gesetzt wird und nach einem Ausfall der Versorgungsspannung zwar das Programm noch im EEPROM gesichert ist, das *RunEnable* Flag jedoch defaultmäßig auf 0 gesetzt wird.

Mehr zum Thema Calculator Assembler ist im Manual [PD 93] nachzulesen.

4 Laborbetrieb

Dieser Abschnitt setzt sich nun mit dem eigentlichen Ablauf einer Laborübung auseinander, beginnend mit Aufgabenstellung und Initialisierung bis hin zur Programmierung und der Gesamtlösung des Problems.

4.1 Aufgabenstellung

Die folgende Aufgabenstellung soll einen Rahmen darstellen, in dem Freiheiten durchaus erwünscht sind, und somit der Übungsablauf zum Teil durch den Übungsteilnehmer selbst gestaltet werden kann. In diesem Fall sollte jedoch jede Überlegung begründet und dokumentiert werden.

Die Programmierung des Masters muß folgende Punkte beinhalten:

- Aus Becken 2 soll in bestimmten Zeitabständen eine definierte Flüssigkeitsmenge mit definierter Temperatur entnommen werden (= Abfluß in Becken 3).
- Die Ventile sollen für das Befüllen und Entleeren der Anlage auf eine sinnvolle Weise manuell zu bedienen sein.
- Parameter wie Solltemperaturwerte, Abgabemenge müssen angezeigt werden und nötigenfalls zu verändern sein.
- Da die Pumpe nur duch die Flüssigkeit geschmiert wird, würde ein Trockenlaufen zur Zerstörung der Dichtung führen.
- Sicherung gegen Grenzwertüberschreitungen laut *3.4 Betriebssicherheit und Grenzen der Anlage.*

Um die Übungsanlage und die auf ihr realisierte Aufgabe zu beschreiben, wird eine Sprache benötigt, die es ermöglicht, multitaskingfähige Systeme darzustellen. Im folgenden wird die in diesem Fall verwendete Sprache SDL/GR kurz vorgestellt.

4.2 SDL

SDL (Specification and Discription Language) verfolgt die Grundidee von parallelen Prozessen, die miteinander kommunizieren, ist also für die Beschreibung einer PROCESS-PASCAL-Programmierung sehr gut geeignet. Es folgt nun eine kurze Einführung in SDL und die Vorstellung einiger beispielhafter Elemente der grafischen Darstellungsform SDL/GR. Die Darstellung der Musterlösung in SDL-Diagrammen erfolgt in *ANHANG C: SDL-Diagramme.*

4.2.1 SDL-Grundlagen

SDL ist keine Programmiersprache, sondern eine Sprache zur Spezifikation und Beschreibung von Systemen [DD93]. Ein Gesamtsystem wird in 3 Ebenen aufgegliedert. Die oberste Ebene ist die sogenannte *Systemebene*, diese gliedert sich in mehrere *Blöcke* die durch sogenannte *Kanäle* miteinander kommunizieren. Die Blockebene wiederum ist unterteilt in einzelne *Prozesse*, die mittels *Signalen* kommunizieren.

Eine Darstellung in SDL kann sowohl als Software als auch als Hardware implementiert werden. Um ein System zu beschreiben ist folgendes Grundkonzept zu beachten:

- Die Darstellung erfolgt durch erweiterte Automaten in Form von Prozessen.
- Die Kommunikation der Prozesse untereinander erfolgt asynchron über Datenwege.
- Es besteht die Möglichkeit Daten zu speichern und zu manipulieren.
- Ein Prozeß ist entweder in einem Ruhezustand oder in einer Transition in einen Folgezustand.
- Prozesse sind prinzipiell gleichberechtigt. SDL ist kein hierarchisches Modell.
- Ein Prozeß kann nur durch sich selbst beendet werden.
- Für das Einlesen von Daten ist eine Input-Queue nach dem FIFO-Prinzip vorgesehen.

Während einer Transition, also dem Übergang von einem Zustand auf einen Folgezustand, werden Aktionen ausgeführt. Aktionen sind einerseits Empfang und Sendung von Information und andererseits die Bearbeitung von Information.

Es gibt verschieden Darstellungsformen von SDL. SDL/PR ist eine textorientierte Darstellungsform, SDL/GR hingegen bietet die Möglichkeit, Zusammenhänge grafisch darzustellen. Für die Behandlung dieser Aufgabenstellung wurde SDL/GR gewählt.

4.2.2 Grafische Darstellung von SDL

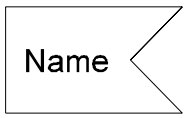
In der grafischen Darstellung der System- und Blockebene, werden die Kanäle mit Pfeilen versehen, die die Richtung der Kommunikation angeben und in eckigen Klammern die im jeweiligen Kanal versendeten Signale eingetragen. Ein Beispiel dazu ist die in *ANHANG C: SDL-Diagramme* dargestellte Musterlösung.

Die grafischen Elemente der Prozessebene, die für die Musterlösung benötigt wurden sind im folgenden dargestellt und kurz erläutert.

STATE-Element:

Das STATE-Element stellt einen stabilen Zustand dar. Es kann in einem Prozeß mehrmals auftreten und übernimmt somit auch die Konnektorfunktion. Es wird prinzipiell gefolgt von einem INPUT- oder OUTPUT-Element.

INPUT-Element:



Das INPUT-Element leitet eine Transition ein und wird daher immer von einem STATE-Element gefolgt. Die ankommenden Inputsignale werden in einer Queue nach dem FIFO-Prinzip verwaltet.

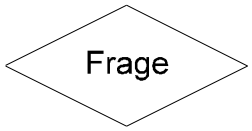
OUTPUT-Element:

Das OUTPUT-Element ist Teil einer Transition. Es ermöglicht asynchrone Kommunikation mit einem oder mehreren INPUT-Elementen. Es besteht weiters die Möglichkeit, gezielt einen bestimmten Prozeß (auch den eigenen) zu adressieren.

TASK-Element:

Das TASK-Element ist ein Teile einer Transition. Es dient dazu Soft- und Hardware-Aktionen zu leisten. Eine Software-Aktion wäre z. B. einer Variablen einen Wert zuzuweisen. Harware-Aktionen sind z. B. das Ansteuern von Pumpen und Ventilen.

DECISION-Element:



Das DECISION-Element dient der Abfrage von Daten und führt entsprechend dem Ergebnis zu einer Transitionsverzweigung, ist also Teil einer Transition. Die Frage muß derart formuliert sein, daß die Antwort keinen Variablenvergleich enthält.

Näheres zu den Grundlagen von SDL ist in [DD93] nachzulesen. Für eine intensivere Auseinandersetzung mit dem Thema ist [SDL92] zu empfehlen.

4.3 Initialisierung des P-NET-Systems

Da von einem P-NET-Neuling nicht erwartet werden kann, ein ganzes System zu initialisieren, also die Adressen, die Anzahl der Master, usw., festzulegen, wurde diese Aufgabe bereits vorweggenommen. Es liegt also ein vollständig initialisiertes System (Abb. 20) vor.

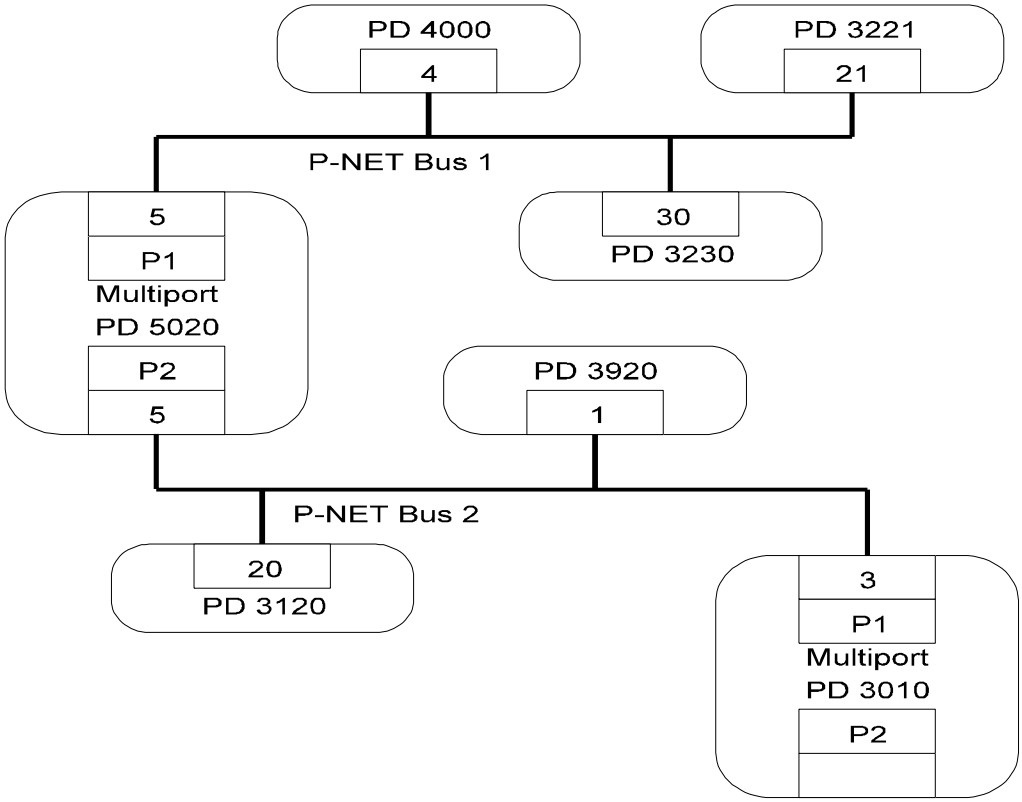


Abb. 20: Topologie der Übungsanlage

Da alle Slaves ab Werk die Adresse 0 zugewiesen haben, bzw. bei bereits anderweitig verwendeten Knoten die Adresse nicht bekannt ist, erfolgte die Adressvergabe an einen P-NET-Knoten in folgender Weise:

Mit dem PC wird **nur** das zu adressierende Modul verbunden, welches dann mit der *Broadcastadresse* 127 angesprochen werden kann. Im *Service Channel SWNo.4* kann dann die gewünschte Adresse eingetragen werden. Hätte man zu dieser Zeit mehrere Knoten mit dem PC verbunden, würde man mit der *Broadcastadresse* alle Knoten ansprechen, was natürlich nicht im Sinne der Sache ist.

Die *Broadcastadresse* in einem bereits fest verbundenen Netz sinnvoll einzusetzen, ist nur in Verbindung mit der, jedem P-NET-Modul eigenen Seriennummer möglich [PD30, PD21].

Zu Abb. 20 sei noch bemerkt, daß der Multiport-Master PD 5020 nur aus Gründen der Einfachheit an beiden Ports dieselbe Adresse, also 5, hat. Natürlich kann in jedem P-NET-Bus die Adresse frei gewählt werden.

4.4 Programmierung in PROCESS-PASCAL

Im folgenden soll die Aufgabe aus *Aufgabenstellung* schrittweise, durch die Aufgliederung in einzelne Teilaufgaben gelöst werden. Aufgrund der Multitaskingfähigkeit von PROCESS-PASCAL liegt es nahe, diese Aufteilung mit der Gliederung in *Tasks* gleichzusetzen. Diese Tasks sind so gestaltet, daß zu Beginn einfache Verknüpfungen von Funktionstasten mit Ventilen vorgenommen werden und erst in weiterer Folge komplexere Aufgaben zu lösen sind.

Dieses Konzept ermöglicht ein schrittweises Verständnis der Möglichkeiten von PROCESS-PASCAL und stellt den Begriff Multitasking besonders plastisch dar.

Weiters wird im folgenden zur anschaulichen Erklärung auf die Musterlösung der Aufgabenstellung verwiesen, deren Quellcode in *ANHANG A: FKEY4000.INC* bzw. in *ANHANG B: Hauptprogramm* dargestellt ist.

4.4.1 Der PROCESS-PASCAL-Compiler

Alle für diese Laborübung notwendigen Files sind auf dem dafür vorbereiteten Rechner unter `c:\p-net\labor` zu finden.

Da zur Zeit noch keine unter Windows lauffähige Version eines PROCESS-PASCAL-Compilers zu Verfügung steht, muß die Kompilierung noch auf der DOS-Ebene vorgenommen werden. Dazu wird ein PROCESS-PASCAL File (gekennzeichnet mit der Ex-

tention .PP) mit einem beliebigen Texteditor bearbeitet, abgespeichert und anschließend mit der Befehlszeile `ppcompil filename.pp` compiliert.

4.4.2 Initialisierung der Channels

Wie bereits in *2.2 Die Channel Struktur* ausführlich beschrieben, hat jeder *Channel* neben den eigentlichen Meß- oder Ausgabewerten eine Reihe von Einstellparamatern.

Das Festlegen der Funktionen eines Channels (In oder Out, Pt100 Input, Waage kalibrieren,...) kann nun wahlweise durch ein eigenes Initialisierungsprogramm, oder mittels des P-NET-Monitors, also direktes Einschreiben der entsprechenden Werte geschehen, wie in diesem Fall. Die eingeschriebenen Werte werden bis zur nächsten Änderung im EEPROM des betreffenden P-NET-Knotens gespeichert (siehe auch *2.2.1 Speicherformen)*. Besonders zu beachten ist, daß für das Ändern mancher Variablen das Flag *WriteEnable* auf TRUE gesetzt werden muß.

Es soll nun gezeigt werden, wie es mit Hilfe des Monitorprogramms möglich ist die wichtigsten Einstellungen zu machen, den Zustand des Systems zu überwachen (Error-Flags) und darüber hinaus Meßergebnisse wie Gewicht und Temperatur darzustellen.

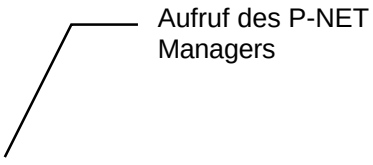
4.4.3 Das Monitorprogramm

Zur Zeit stehen eine DOS- und eine Windows-Version des Monitorprogramms zur Verfügung. Sie unterscheiden sich im wesentlichen nur durch die Bedienoberfläche, haben aber dieselben Möglichkeiten. Daher werden sich die folgenden Erklärungen auf die Windows-Version beziehen und sind sinngemäß auch auf die DOS-Version anzuwenden.

Die wesentlichen Aufgaben des Monitorprogramms sind die Darstellung und Manipulation einer SWNo. oder eines Bits daraus. Die wichtigsten Details dazu werden im folgenden aufgezeigt und kurz erläutert.

4.4.3.1 Einrichten des Programms

In Abb. 21 ist die Bedienoberfläche der Monitors unter Windows dargestellt. Da sie anderen Windows-Oberflächen sehr ähnlich ist, ist mit diesem Programm schnell ein effizientes Arbeiten möglich.



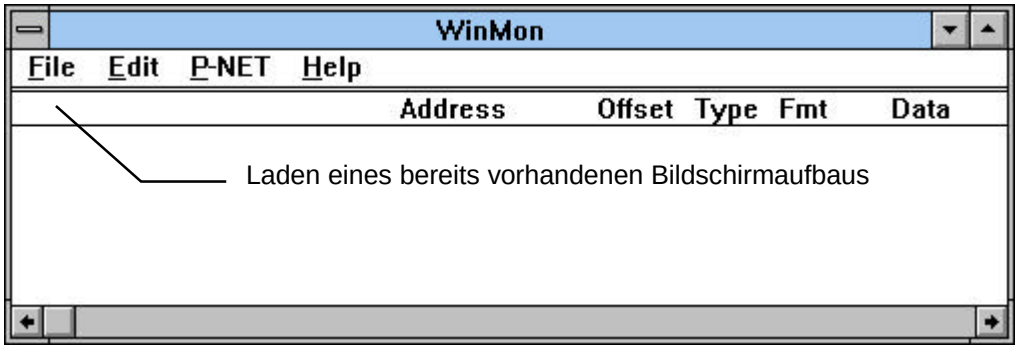


Abb. 21: Monitor

Nach Aufruf des Monitors muß für den späteren Zugriff auf die Variablen jedem P-NET-Knoten mittels des **P-NET-Managers** ein benutzerdefinierter Name zugewiesen werden.

In Abb. 22 sind die wichtigsten Details des P-NET-Managers dargestellt. Der Begriff *Projekt* beinhaltet die gesamte Konfiguration eines Netzes d.h. bei Änderungen in der Adressvergabe oder in der Topologie des Netzes, muß ein neues Projekt-File angelegt, oder das vorhandene abgeändert werden.

Über die Schaltflächen **New, Delete** und **Update** können Knoten in das Netz eingefügt, vom Netz entfernt oder geändert werden. Dazu gibt man unter **Node** einen benutzerdefinierten Namen ein und wählt aus einer vorhandenen Bibliothek ein **Type** aus.

Unter **Filename** ist der Pfad bzw. der Filename des sogenannten *Mapfiles* anzugeben, in welchem einer SWNo. ein Name zugeordnet ist. Ein Mapfile wird automatisch bei der Compilierung eines PROCESS-PASCAL-Programmes vom Compiler generiert.

Zuletzt gibt es im rechten unteren Bereich des P-NET-Managers noch eine Schaltfläche **Driver Setup**. Nach öffnen dieses Fensters, besteht die Möglichkeit, die Gesamtanzahl der Master und die Master-Adresse der PD 3920 PC-Interface-Karte festzulegen.

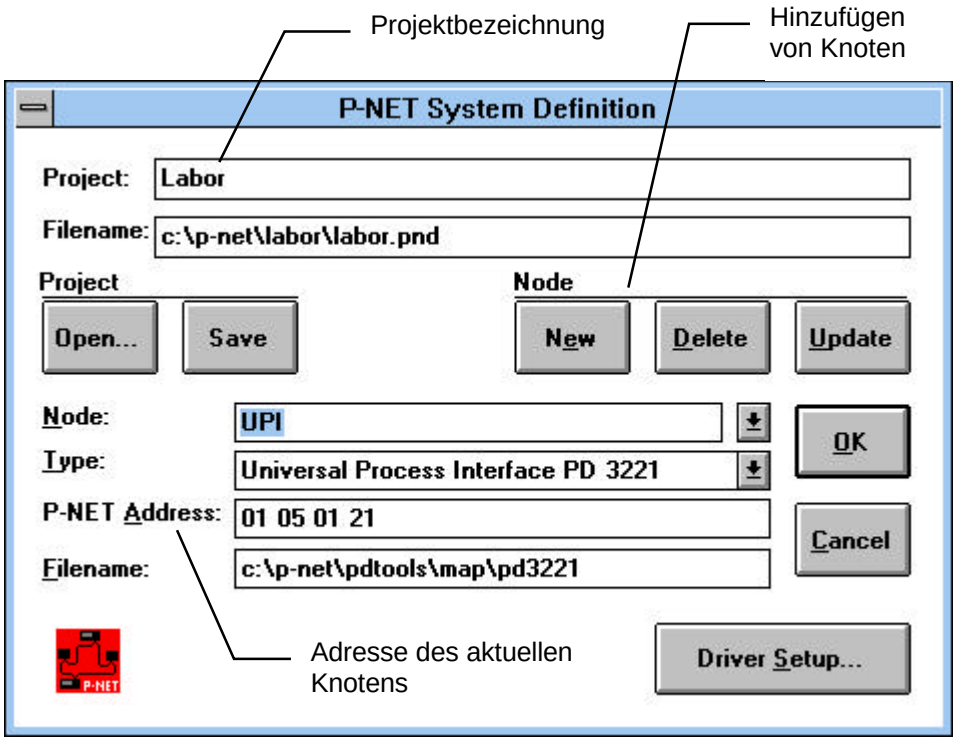


Abb. 22: P-NET-Manager

4.4.3.2 Darstellung und Manipulation von Daten

Der eigentliche Sinn des Monitorpogramms ist die Darstellung und Manipulation von Daten in einem P-NET-System. In Abb. 23 ist der Inhalt der einzelnen Spalten erläutert.

Um einen Wert anzuzeigen, muß in der linken Spalte nach Doppelklick mit der linken Maustaste, der exakte Name der Variable eingetragen werden. Im Falle des Digital IO ist für das Setzen des *OutFlag*, die Angabe der Bitnummer in eckigen Klammern vonnöten (siehe auch *2.2.3 Der digitale I/O Channel*). Durch einen Mausklick in die Checkbox kann dann die gewünschte Zeile aktiviert werden. Falls es sich bei der betreffenden SWNo. um einen Record handelt, besteht noch die Möglichkeit einen Offset, der die Position innerhalb des Records bezeichnet, anzugeben.

Unter dem Menüpunkt **Edit** können auf die von anderen WINDOWS-Anwendungen bekannte Art mittels *Append* Zeilen hinzugefügt und mit *Clear* gelöscht werden. Die Funktion *Cut* schneidet eine Zeile aus und überträgt sie in die Zwischenablage, *Copy* kopiert eine Zeile in die Zwischenablage und *Paste* fügt den Inhalt der Zwischenablage wieder ein.

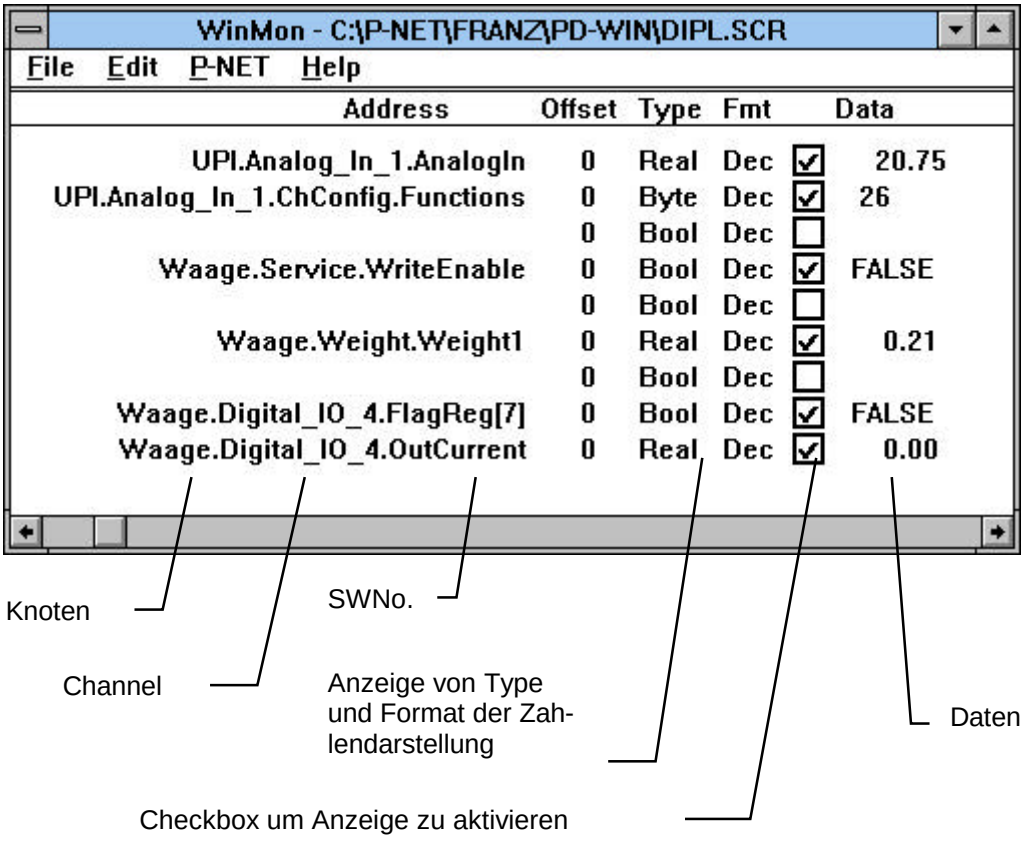


Abb. 23: Darstellung von Daten

4.4.4 Deklarationen

Der nächste Schritt ist nun die passende Deklaration von Variablen.

Wie in *Variablen im P-NET* bereits beschrieben ist es sinnvoll, indirekte Deklaration zu benutzen, um anschauliche und kurze Variablennamen zu erhalten. Eine für diese Aufgabenstellung zweckmäßige Deklaration wird in *ANHANG B: Hauptprogramm* vorgestellt.

4.4.5 Festlegen der Tasks

Natürlich existieren verschiedene Möglichkeiten die Aufgabenstellung zu strukturieren. Für die Musterlösung wurde jedoch folgende Aufteilung in einzelne Tasks gewählt:

- Keyboardtask
- Levelüberwachung
- Display und Updating von Parametern
- Heizen von Becken 1
- Regeln der Temperatur in Becken 2
- Ablassen der dosierten Menge

Als Rahmen für die Programmierung ist das Quellcodefile `PD4000.PP` vorgesehen, in dem bereits alle wichtigen *Include Files* vorgegeben sind. Der Sinn dieses Programmrahmens ist es, dem Programmierer den unbedingt nötigen Teil des Quellcodes bereits vorzugeben und somit eventuelle Fehlerquellen durch falsche Initialisierung auszuschalten.

4.4.5.1 Task 1: Keyboardtask

AUFGABE: Für das Hochfahren und das Abschalten der Anlage, also das Entleeren und Befüllen der Becken, muß die Möglichkeit bestehen, einzelne Ventile und die Pumpe manuell zu schalten.

Natürlich kann für diesen Fall eine eigene Routine geschrieben werden, die auf Tastendruck abrufbar ist, diese sollte aber nicht Inhalt dieser ersten, einfachsten Teilübung sein. Vielmehr soll in dieser ersten Teilaufgabe gezeigt werden wie einfach und übersichtlich Verknüpfungen zwischen Ein- und Ausgängen realisiert werden können. Um die Übersichtlichkeit des Quellcodes zu steigern, werden wie auch in *Anhang B* zu sehen ist, verschiedene, meist grundlegende Teile des Quellcodes als *Include-Files* eingebunden. Das betreffende Include-File für die Belegung von Funktionstasten ist `FKEY4000.INC.`

In Abb. 24 ist die Codierung der Tastatur des PD 4000 Controllers dargestellt. Im rechten Bereich ist ein Standard-Zehnerblock mit einigen zusätzlichen Tasten realisiert. Eine Besonderheit ist, daß bei Eingaben die "="-Taste die Funktion einer *Enter*-Taste übernimmt.

Im linken Bereich sind benutzerdefinierbare Funktionstasten vorgesehen. Die Numerierung der Funktionstasten kann direkt für die Abfrage und weitere Verknüpfungen verwendet werden (siehe *ANHANG A: FKEY4000.INC*). Zu der in Abb. 24 dargestellten Tastaturbelegung noch einige erläuterungen:

- Die Taste *Becken 1 entleeren* öffnet gleichzeitig Ventil 1 und Ventil 2. Um die Ventile wieder zu schließen ist diese Taste ein weiteres mal zu drücken. Diese Variante ist insofern zu bevorzugen, daß bei Verwendung einer separaten *Öffnen*- und *Schließen*-Taste die Anzahl der vorhanden Tasten nicht effizient ausgenutzt wäre.

- Den Zusammenhang zwischen der nächsten Taste *Becken 2 entleeren* und der Dosierung einer Wassermenge wird in *4.4.5.6 Task 6: Ablassen der dosierten* Menge beschrieben.
- Zum Abschluß wurde noch eine Taste die *Becken 1 befüllen* belegt, die analog der Taste *Becken 1 entleeren* das Ventil 3 und die Pumpe schaltet.

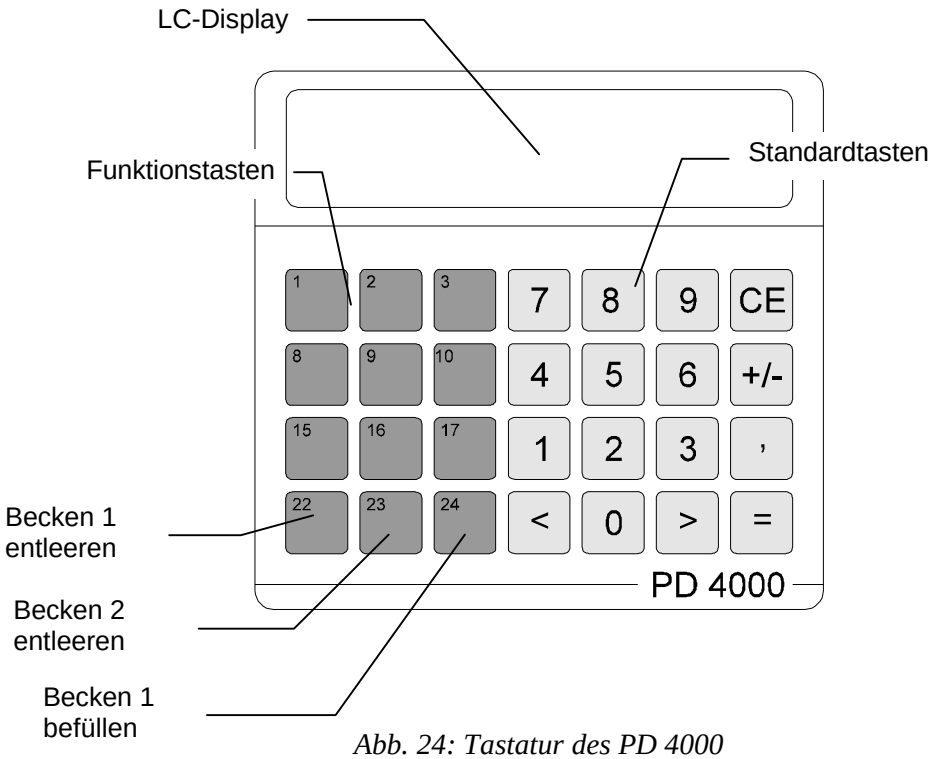


Abb. 24: Tastatur des PD 4000

Die gewünschten Verknüpfungen der Funktionstasten werden nun in dem bereits vorhandenen File `FKEY4000.INC` realisiert. Dieses Include-File enthält eine Prozedur, die durch eine Fallunterscheidung bei drücken einer bestimmten Funktionstaste die adäquate Aktion auslöst. Diese Prozedur wird vom eigentlichen Keyboardtask, der in `KEY4000.INC` enthalten ist, aufgerufen. Für die Musterlösung ist der Inhalt von `FKEY4000.INC` in *ANHANG A: FKEY4000.INC* dargestellt.

4.4.5.2 Task 2: Levelüberwachung

AUFGABE: Überwachung des Füllstandes beider Becken und bei Erreichen des Grenzwertes schalten der passenden Ventile und der Pumpe.

Für diese Teilaufgabe wird nun zum ersten mal ein Task eigenständig programmiert. Zur Überwachung des Füllstandes müssen die beiden Levelschalter L1 und L2 abgefragt werden und je nach Ergebnis die Ventile bzw. die Pumpe geschaltet werden. Diese Task trägt nicht zu unrecht den Namen *Safety*. Er ist zu Beginn der Arbeit an der Anlage zu programmieren (siehe Musterlösung *ANHANG B: Hauptprogramm*) und darf auf keinen Fall außer Betrieb gesetzt werden. Denn auch bei größter Vorsicht kann eine Situation auftreten, in der andernfalls eines der Becken überlaufen würde.

In Abb. 25 ist die Grundstruktur eines Cyclic-Tasks dargestellt und erläutert. Der Beginn der zyklisch zu bearbeitenden Schleife ist mit dem Befehl *loop* gekennzeichnet. Im Bereich *Command 1* bis *Command n* sind in diesem Fall die Abfrage und die Verknüpfungen der Levelschalter mit den Ventilen zu realisieren. Da es in diesem Fall keinen günstigereren Zeitpunkt für einen Taskwechsel gibt (z. B. warten auf das Ablaufen eines Timers), wird am Ende von jedem Schleifendurchlauf mit dem Befehl *Changetask* ein Wechsel zum nächsten Task erzwungen.

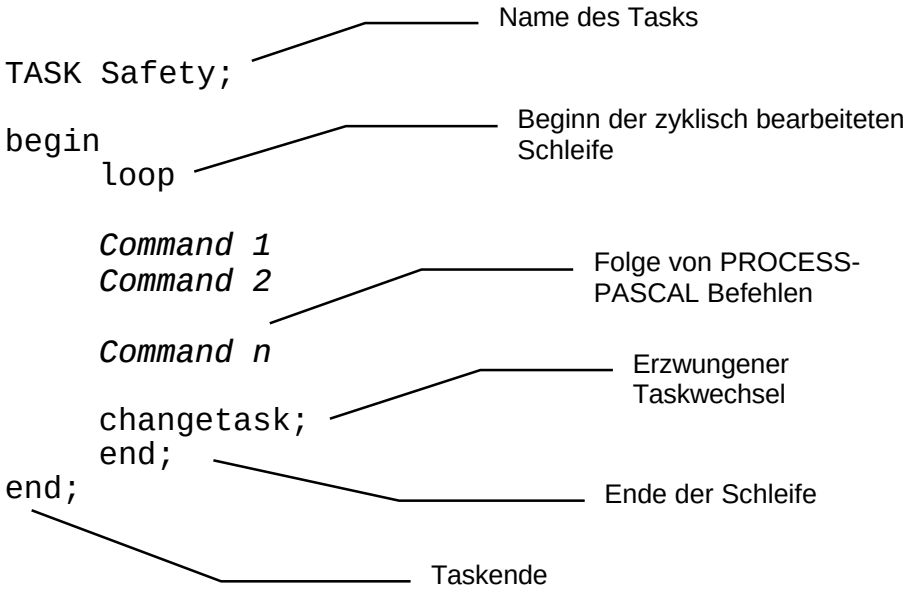


Abb. 25: Grundstruktur eines Tasks

4.4.5.3 Task 3: Display und updating von Parametern

AUFGABEN:

- Sinnvolle Darstellung von Parametern auf dem LC-Display
- Soll- und Grenzwerteingabe mittels UPDATE Funktion
- Eingabe der gewünschten Abflußmenge aus Becken 2

Dieser Task soll die in PROCESS-PASCAL implizierten Möglichkeiten, Programme für interaktives Arbeiten zu schreiben, demonstrieren. Das interessanteste Detail ist in diesem Zusammenhang wohl die Funktion UPDATE, die nun kurz behandelt wird.

UPDATE ist zugleich Ausgabe- und Eingabefunktion speziell für Master mit einer Displayeinheit. Zur Änderung des Wertes muß der Cursor mittels der Pfeiltasten auf die Position der Zahl bewegt werden, der neue Wert eingegeben und mit "=" bestätigt werden. Ein typisches Programmierbeispiel dazu ist in Abb. 26 dargestellt.

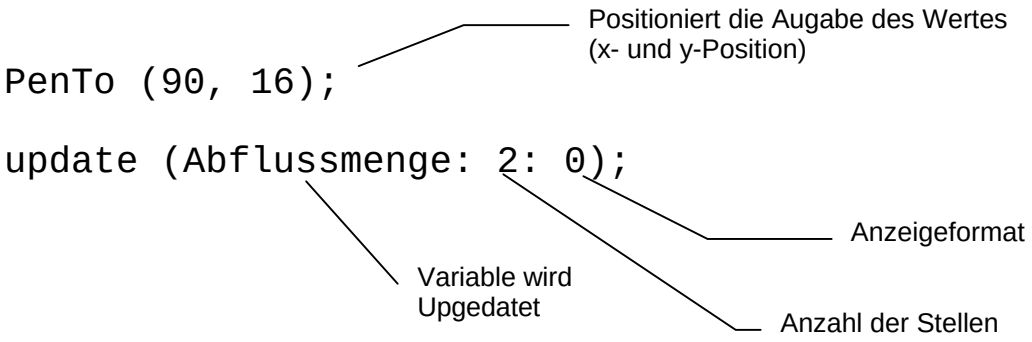


Abb. 26: Syntax der UPDATE-Funktion

Die *Anzahl der Stellen* ist inklusive Komma zu verstehen und an der Position *Anzeigeformat* kann festgelegt werden, wie der Wert der Variable angezeigt werden soll. Die vorgegeben Möglichkeiten sind dezimal, hexadezimal, binär oder dezimal mit Nullen aufgefüllt.

Die letzte Möglichkeit ist besonders für den Betrieb einer mehrstelligen Sieben-Segment-Anzeige von nutzen, da es für den Benutzer in manchen Bereichen nützlich ist, den Umfang der Anzeige von vornherein abschätzen zu können.

In Abb. 27 ist neben der Funktionstastenbelegung zum Wechseln der Ansicht, dargestellt wie z. B. die im Master PD 4000 gespeicherte Abflußmenge aus Becken 2 Verändert werden kann.

Da auf dem LC-Display nur eine begrenzte Anzahl von Parametern dargestellt werden kann, wurde in der Musterlösung die Möglichkeit realisiert, zwischen 2 Bilschirmansichten hin und her zu schalten und zwar mit den tasten *Bildschirm 1* und *Bildschirm 2*.

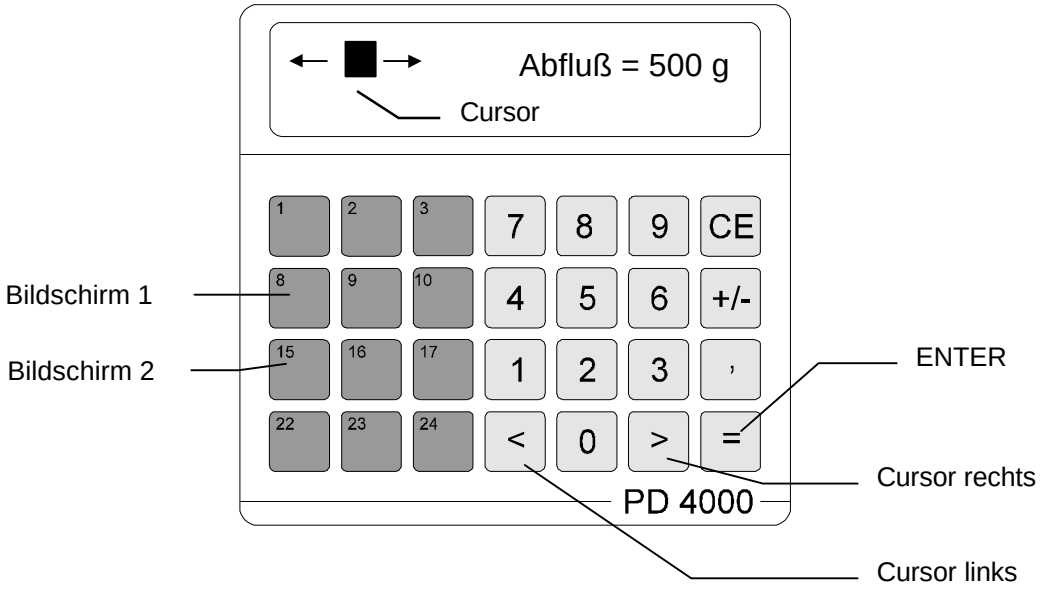


Abb. 27: Updating und Wechseln der Ansicht

Es ist auf jeden Fall darauf zu achten, daß auf jeweils einem Bildschirm sinnvolle Zusammenfassungen von Parametern dargestellt werden. Im Falle der Musterlösung sind z. B. auf Bildschirm 1 alle Daten, die den Heizvorgang betreffen dargestellt (Solltemperaturen, aktuelle Temperaturen, Heizzeit). Auf Bilschirm 2 sind Daten für den Abfluß dargestellt (Gewicht in Becken 1, Abflußmenge).

4.4.5.4 Task 4: Heizen von Becken 1

AUFGABE: Regelung der Wassertemperatur in Becken 1 Mittels des Heizstabes und des Temperaturmeßwertes.

Um diesen Task realisieren zu können, müssen im folgenden einige grundlegende physikalische und regelungstechnische Überlegungen angestellt werden.

Da ein klassischer Reglerentwurf in Ermangelung einer negativen Stellgröße entfällt (der Heizstab kann nicht kühlen), bieten sich zwei Möglichkeiten an:

- Zweipunktregelung

- Steuerung mit Überwachung der Temperatur

Die Zweipunktregelung entfällt aufgrund der schlechten Durchmischung im Becken 1, da im Endeffekt nur die Konvektion als Mischungsinitiator auftritt, was zu heftigem Überschwingen der Temperatur führen würde. Daher ist das Konzept der Steuerung der Heiztemperatur mit Überwachung zu bevorzugen.

Als Regelbasis wird die Temperatur vor Einschalten des Heizstabes erfaßt und daraus über den Energieinhalt die benötigte Heizdauer bestimmt. Das Problem vereinfacht sich bei diesem Ansatz insofern, als davon ausgegangen wird, daß bei einer Wiederbefüllung des Beckens 1 es zu einer günstigen Durchmischung des Wasserinhaltes und somit zu einer homogenen Temperaturverteilung über das gesamte Volumen kommt.

Man kann also von einem Kontinuumsmodell ausgehen und mittels folgender Gelichung die benötigte Wärmemenge Q und damit die Dauer, in welcher der Heizstab einzuschalten ist, bestimmen.

$$Q = m \cdot c_{H2O} \cdot \Delta J$$

Die Temperaturdifferenz $\Delta\vartheta$ ergibt sich aus Soll- minus Isttemperatur. Die Isttemperatur kann ohne weiteres über den Sensor gemessen werden, da im ungeheizten Zustand eine Homogene Temperaturverteilung gegeben ist. Die Masse m ist durch die Geometrie des Beckens 1 gegeben und die spezifische Wärme von Wasser ist aus der Literatur mit c_{H2O} = 4,187 kJ/kg.K bekannt.

Für konkrete Zahlenwerte kommt man zu folgendem Resultat:
Wassermasse m=5 kg
Isttemperatur ϑ_0=20°C
Solltemperatur ϑ_1=50°C

$$\Delta\vartheta = \vartheta_1 - \vartheta_0 = 30°C$$

$$Q = m \cdot c_{H2O} \cdot \Delta J = 5 * 4187 * 40 = \mathbf{628{,}05\ kJ}$$

Damit läßt sich die Einschaltdauer des Heizstabes (P_{Heizer} = 1000 W) als

$$\Delta t = \frac{Q}{P_{Heizer}}$$

angeben. Für oben begonnene Beispiel ergibt sich eine Einschaltdauer von

$$\Delta t = \frac{Q}{P_{Heizer}} = \frac{628050}{1000} = 10{,}5\,\text{min}\,.$$

Eine zusätzliche Abfrage des Temperatursensors T1 überwacht, daß am Beckenrand keine zu hohen Temperaturen auftreten, falls es zu einer ungünstigen Durchmischung des Beckens kommen sollte.

Da die Lösung dieser Teilaufgabe also eine reine Steuerungsaufgabe ist, wird der im PD 3221-Modul integrierte PID-Regler nicht verwendet, sondern lediglich der betreffende Digital-IO geschaltet. Dies geschieht indem zu Beginn der Heizzeit ein Timer gesetzt wird und im Task dessen Ablaufen überwacht wird.

Um nun einen korrekten Ablauf dieser und auch der folgenden Aufgaben zu gewährleisten, muß die Ausfürung dieses Tasks mit mehreren Flags verknüpft werden. Die in der Musterlösung angebotenen Verknüpfungen sind in *ANHANG B: Hauptprogramm* nachzulesen.

4.4.5.5 Task 5: Regelung von Becken 2

AUFGABE: Bereitstellung von Wasser mit in bestimmten Grenzen wählbarer Solltemperatur in Becken 2.

Um Becken 2 auf Solltemperatur zu regeln, stehen 2 Mittel zur Verfügung:
- Heißwasser aus Becken 1 über V1 und V2
- Kaltwasser aus Becken 3 über V4

Daraus wurde für die Musterlösung folgendes relativ einfaches Konzept entwickelt:
Die Temperatur des Kaltwassers wird nach Befüllen des Beckens 1 ermittelt, da zu diesem Zeitpunkt noch gute Durchmischung herrscht, und das Wasser direkt aus Becken 3 kommt. Die Temperatur in Becken 1 ist aus *4.4.5.4 Task 4: Heizen von Becken 1* bekannt. Weiters ist eine gewünschte Warmwassermenge vorgegeben.

Damit läßt sich nun folgende Schlußrechnung anstellen:

$$Menge_{kalt} = Menge_{warm} \cdot \frac{T_{2,soll} - T_{1,soll}}{T_{kalt} - T_{1,soll}}$$

$Menge_{kalt}$	Notwendige Kaltwassermenge
$Menge_{warm}$	Erzeugte Warmwassermenge
$T_{1,soll}$	Solltemperatur in Becken 1
$T_{2,soll}$	Solltemperatur in Becken 2
T_{kalt}	Temperatur des Kaltwassers aus Becken 3

Mit dieser Formel, läßt sich für eine gewünschte Warmwassermenge $Menge_{warm}$ bei bekannten Temperaturen, die nötige Kaltwassermenge $Menge_{kalt}$ berechnen. Das ist natürlich nur möglich, wenn $T_{2,soll}$ zwischen T_{kalt} und $T_{1,soll}$ liegt. Diese Grenzen können in der Programmierung im Rahmen des Anzeige- und Update-Tasks berücksichtigt werden.

Das Einlassen des Kaltwassers muß um einen bestimmten Betrag früher gestoppt w erden, da durch die Trägheit des Systems eine gewisse Wassermenge nachläuft. Diese Wassermenge wurde experimentell mit 47 g ermittelt (siehe auch *4.4.5.6 Task 6: Ablassen der dosierten* Menge). Anschließend kann begonnen werden, Heißwasser aus Bekken 1 beizumengen. Dies geschieht nicht nach der oben berechneten Menge, sondern über den Temperaturmeßwert in Becken 2, da dies wegen der schlechten Durchmischung in Becken 1, zu exakteren Ergebnissen führt.

Dabei stellt jedoch die relativ hohe Zeitkonstante des Pt100-Sensors ein Problem dar. Würde man den Meßwert des Pt100-Sensors als realen Augenblickswert annehmen und bei Nenntemperatur die Ventile 1 und 2 schließen, ist in Wirklichkeit bereits zu viel Heißwasser in Becken 2 gelangt. Daher wird zuerst *Sollwert-1°C* als Regelziel angenommen und eine bestimmte Zeit gewartet, bis der Pt100-Sensor den realen Wert anzeigt. Dann erst wird auf den Sollwert geregelt. Um die Genauigkeit weiter zu erhöhen, kann dieser Vorgang in mehreren, immer kleiner werdenden Schritten wiederholt werden.

Weiters ist zu bemerken, daß die vorgegebene Warmwassermenge $Menge_{warm}$ lediglich zur annähernden Berechnung des Mischungsverhältnisses dient und nicht Ziel der Regelung ist. Bei diesem Konzept ist das einzige Regelziel die Temperatur in Becken 2.

4.4.5.6 Task 6: Ablassen der dosierten Menge

AUFGABE: Es muß die durch eine UPDATE-Funktion festgelegte Flüssigkeitsmenge per Tastendruck aus Becken 2 in Becken 3 über Ventil 5 abgelassen werden.

Das Auslösen dieser Aktion wird wieder dem Keyboardtask zugeordnet, lediglich das Zeitgerechte Abschalten wird von diesem Task übernommen. Es kann daher das Öffnen von Ventil 5 ebensogut von eine anderen Instanz (z. B. Monitorprogramm) ausgelöst werden.

Da die Reaktionszeit des Systems nicht gleich null ist, also nach dem Schließen der Ventile sich noch etwas Wasser in der Leitung bzw. in den Ventilen selbst befindet, sind für den richtigen Schaltzeitpunkt der Ventile einige Überlegungen vonnöten:

Die Abhängigkeit der Ausflußgeschwindigkeit v von der Füllhöhe h ergibt sich aus dem Gesetz von Bernoulli:

$$p + \frac{r}{2} \cdot v^2 = \text{const.} \mapsto v = \sqrt{2 \cdot g \cdot h}$$

Bei Befüllung aus Becken 1 ist also eine Mehrbefüllung des Beckens 2, in Abhängigkeit der jeweiligen momentanen Füllhohe von Becken 1 zu erwarten. Ähnliche Überlegungen gelten sinngemäß beim Entleeren.

Damit verbunden ist ein Fehler, der sich nur durch eine Kalibrierung bzw. Linearisierung der Anlage kompensieren läßt. Ein theoretischer Modellansatz zur Berechnung der Korrekturfaktoren wurde aufgrund der Kleinheit der Gefäße und der damit verbundenen Abweichungen vom noch im vernünftigen Ausmaß mathematisch handhabbaren Modellansatz, verworfen. So wird z. B. in obiger Gleichung die Reibung vernachlässigt.

Daher ist eine experimentelle Erfassung der Korrekturfaktoren zu bevorzugen, wobei als Korrekturfaktoren jene Gewichtsdifferenzen zu verstehen sind, die sich ergeben, wenn das Programm bei einem Gewicht m_1 die Ventile schließt, sich jedoch schlußendlich ein Gewicht $m_1 + \Delta m$ einstellt.

Nach verschiedenen Messungen wurden folgende Sachverhalte festgestellt:

- Eine nicht unerwartete Abhängigkeit der Ausflußgeschwindigkeit aus dem Becken 1 - und damit ein notwendiges Korrigieren des Ausschaltgewichtes - in Abhängigkeit der Füllhöhe des Beckens 1. Da diese Höhe nicht direkt über eine Gewichtsmessung analog zum Becken 2 zugänglich ist, muß über die abgelassene Menge das Gewicht in Becken 1 berechnet werden.
- Die Systemträgheit bewirkt ein relativ konstantes Mehrbefüllen des Beckens 2, wenn dieses mittels der pumpe aus Becken 3 befüllt wird.
- Die Ausflußkennlinie in Abb. 28 zeigt wieder eine Nichlinearität, jedoch nicht die theoretische (Ausfluß ist proportional zu $\sqrt{h}$) sondern eine lineare Korrelation mit dem Gewicht und damit mit der Höhe.

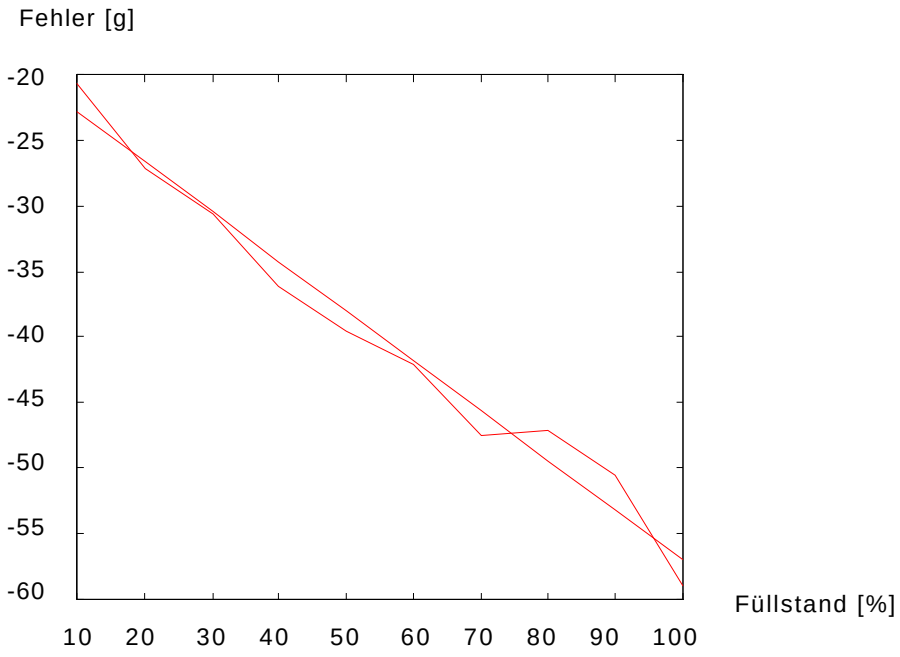


Abb. 28: Ausflußkennlinie

Eine Variante zur Berechnung der Eventgewichte (jenes gemessene Gewicht, ab welchem ein Zufluß zu stoppen ist) beruhte auf linearer Interpolation zwischen zwei gemessenen Korrekturwerten. Da in diesem Fall die Werte sehr stark streuen, wurde folgender, stark vereinfachter Modellanstz als Grundlage für die Programmierung gewählt.

- Bei den Ausflüssen reicht eine Linearisierung über die gesamte Füllhöhe. In Abb. 28 ist diese Gerade stichpunktiert eingetragen. Daraus folgt eine Geradengleichung folgender Form:

$$\Delta m = 0{,}0192 + 0{,}0188 \cdot Gewicht$$

Δm	überschüssige Masse, bzw. jener Wert, um den die Ventile früher geschlossen werden müssen
Gewicht	Momentane Wassermasse in Becken 2

- Wird Becken 2 aus Becken 3 über die Pumpe befüllt, ist für diesen konstanten Fehler eine Mittelwertbildung über einige Versuche hinreichend. Dieser konstante Wert wurde experimentell zu 47g ermittelt (siehe auch *4.4.5.5 Task 5: Regelung von Becken 2*).

4.4.6 Vereinfachung der Aufgabenstellung

Angesichts dieser relativ aufwendigen Modellierung des Gesamtsystems, könnte man bei den letzten beiden Punkten (*4.4.5.4 Task 4: Heizen von Becken 1* und *4.4.5.5 Task 5: Regelung von Becken 2*) speziell für regelungstechnisch weniger versierte Studenten gewisse Vereinfachungen treffen, also z. B. eine Ungenauigkeit von 60 g Wasser beim Ablassen noch tolerieren.

Andererseits sollte diese Aufgabenstellung jedoch demonstrieren, daß auch Aufgaben, die über bloßes Vergleichen von Grenzwerten hinausgehen, in PROCESS-PASCAL programmiert werden können.

5 Zusammenfassung

In diesen abschließenden Betrachtungen wird kurz auf den Ablauf der abgehaltenen Laborübungen eingegangen Daraus resultierend werden Punkte herausgearbeitet, die die Anlage erfüllt hat. Es werden aber auch Vorschläge für einen Ausbau der Anlage gemacht, um den Lernerfolg zu steigern und weitere Möglichkeiten von P-NET zu demonstrieren.

5.1 Die Laborübung

Wie bereits erwähnt wurden im Sommersemester 1996 die ersten Übungen an der Anlage abgehalten. Für diese Übungen wurde das Kapitel *2 Der P-NET-Standard* in geringfügig abgeänderter Form als Theorie-Skriptum ausgegeben. Der darauffolgende Eingangstest hat gezeigt, daß die meisten der angemeldeten Studenten das Stoffgebiet ausreichend gelernt haben.

Zum nachfolgenden praktischen Teil der Laborübung sei zu allererst bemerkt, daß eine Anlage, bei der mit Flüssigkeiten hantiert wird, scheinbar nie genügend gegen Überlauf abgesichert werden kann. Auf jeden Fall muß der Student auf verschiedene Sicherheitsmechanismen hingewiesen werden (Safety-Task, Notabschaltung,...).

Ansonsten ist es stark unterschiedlich, welcher Zeitraum für verschiedene Studenten ausreicht, um die Aufgabenstellung zu lösen, wobei ein bestimmter Zeitrahmen auf jeden Fall vorzugeben ist, da es ansonsten zu Terminkollisionen der einzelnen Gruppen kommt.

Ein grundlegendes Problem einer Gruppenübung ist Folgendes: Der ambitioniertere und besser vorbereitete Student übernimmt einen großen Teil der Programmierung, der Rest sitzt daneben und wird wenig Erfahrung mit dem Objekt der Übung, in diesem Fall P-NET, machen. Die Lösung dieses Problems wären entweder Einzelübungen oder Gruppen mit exakt gleichem Wissensstand der einzelnen Mitglieder. Letzteres ist natürlich in der Praxis unmöglich weil unüberprüfbar.

Einzelübungen sind für den Betreuer mit einem enormen Mehraufwand verbunden, was auch teilweise die Begründung für die Abhaltung von Gruppenübungen ist. Andererseits sollte eben gerade durch Gruppierung, die Zusammenarbeit mit ähnlich ausgebildeten Mitarbeitern erlernt werden - auch wenn innerhalb einer Gruppe gewisse Spannungen auftreten.

5.2 Erweiterungen

In den meisten Belangen ist die Übungsanlage den an sie gestellten Erwartungen gerecht geworden. So war z. B. das Feedback Puncto Anschaulichkeit und Wirklichkeitsnähe ein sehr gutes. Auch das für die Übung ausgegebene Laborskriptum hat seinen Zweck, nämlich theoretisches Grundwissen zu vermitteln, sehr gut erfüllt. Es gibt jedoch vermutlich kein System, das man als **perfekt** bezeichnen könnte - so auch dieses. Daher folgen nun einige Erweiterungsvorschläge, die das Arbeiten mit der Übungsanlage noch effizienter und komfortabler gestalten würden.

Um die Temperaturmessung in Becken 1 zu verbessern, wäre es am einfachsten, auch in diesem Becken ein Rührwerk zu installieren. Zur Zeit ist es aufgrund der schlechten Durchmischung von Heiß- und Kaltwasser nur bedingt möglich, ein repräsentatives Meßergebnis zu erhalten, was natürlich die Programmierung erschwert.

Eine weitere, relativ aufwendige Erweiterung wäre es, die Anlage an fließendes Wasser anzuschließen, d. h. auf die Pumpe und Becken 3 völlig zu verzichten. Der Zufluß würde dann vom Wasserleitungssystem der TU-Wien gespeist werden und der Abfluß ginge in die Kanalisation. Der Vorteil dieses Konzepts wäre, daß das zufließende Wasser immer annähernd gleiche Temperatur hätte. Bei der derzeitigen Konfiguration der Anlage bildet die Flüssigkeit einen geschlossenen Kreislauf. Dies führt dazu, daß sich mangels Abkühlung das Wasser in Becken 3 bei mehreren Umläufen kontinuierlich erwärmt. Um also sinnvoll arbeiten zu können, muß die Solltemperatur für Becken 1 bzw. Becken 2 mitangehoben werden, da der Heizstab nur heizen und nicht kühlen kann.

Wie bereits erwähnt ist auch aus Sicht der Busstruktur eine Erweiterung geplant. Am Port 2 des PD 3010 Multiport-Masters soll ein dritter P-NET-Bus installiert werden, über den Knoten verschiedener Hersteller miteinander verbunden werden. Auf diese Weise würde die Idee einer Multivendoranlage verwirklicht werden.

Die Zukunftsperspekive für einen Industriebetrieb ist es, bei der Installation oder der Erweiterung einer Anlage nicht auf einen Anbieter angewiesen zu sein, sondern verschiedene Knoten von verschiedenen Herstellern beliebig kombinieren zu können. Diese Möglichkeit wird sicher zu einer noch weiteren Verbreitung des Konzepts von P-NET führen.

ANHANG A: FKEY4000.INC

```
PROCEDURE FuncKeys(NewKey: BYTE);

BEGIN

(* CASE-Anweisung zur unterscheidung der einzelnen Tastaturcodes ( siehe auch Task 3: Display und updating von Parametern ) *)
  CASE NewKey of


(* Taste 1: Aus- und Einschalten des Mixers:
Hier wurde, wie für alle folgenden Schaltvorgänge anstatt jeweils einer Taste für AUS und einer Taste für EIN, lediglich eine einzige Taste pro geschalteter Einheit verwendet. Der Vorteil ist eine effizientere und Übersichtlichere Ausnutzung der Tastatur. *)

(* Ist der Mixer ausgeschaltet, wird er eingeschaltet *)
    01: if mixer_ein=AUS then mixer:=EIN else
(* Ist der Mixer eigeschaltet wird er ausgeschaltet *)
        if mixer_ein=EIN then mixer:=AUS;


(* Taste 2: RESET *)
(* Um an einem beliebigen Punkt des Prozesses, oder nach einer manuellen Entleerung der Becken einen bestimmten Zustand zu erzwingen, wurde diese Taste belegt.*)
    02: Begin
(* Hetztimer auf null setzen *)
        Delaytimer:=0;

(* Becken 1 „nicht geheizt“ setzen *)
        warm_ok:=FALSE;

(* Becken 2 „nicht voll“ setzen *)
        voll:=FALSE;
        end;
    03: ;


(* Taste 8: Bildschirm 1 aktivieren *)
    08: begin
        bild:=1;
(* Flag für einmaliges Löschen des Bildschirms setzen *)
        cl:=TRUE;
```

```
        end;
    09: ;
    10: ;


(* Taste 15: Bildschirm 2 aktivieren *)
    15: begin
        bild:=2;
(* Flag für einmaliges Löschen des Bildschirms setzen *)
        cl:=TRUE;
        end;
    16: ;
    17: ;


(* Taste 22: Entleeren von Becken 1 *)
(* Funktion analog Taste 1 *)
    22: if ventil2_auf=AUS then
         begin
          ventil2:=EIN;
          ventil1:=EIN;
         end
        else
        if ventil2_auf=EIN then
         begin
          ventil2:=AUS;
          ventil1:=AUS;
         end;



(* Taste 23: Entlehren von Becken 2 *)
(* Funktion analog Taste 1 *)

    23: if ventil5_auf=AUS then ventil5:=EIN else
        if ventil5_auf=EIN then ventil5:=AUS;


(* Taste 24: Befüllen von Becken 1 *)
(* Funktion analog Taste 1 *)

    24: Begin

        if ventil3_auf=AUS then
         begin
          ventil3:=EIN;
          pumpe:=EIN;
         end
        else
        if ventil3_auf=EIN then
```

```
          begin
           ventil3:=AUS;
           pumpe:=AUS;
          end;
         end;
  END;
END;
```

ANHANG B: Hauptprogramm

```
PROGRAM TestPD4000;

(*$I'PD4000.SYS'*)

(* Um auf dem LC-Display des PD 4000 Zeichen darstellen zu
können, müssen verschiedene Zeichensätze geladen werden. *)
(* Zeichensatz 6x7 *)
IMPORT CH6x7.COD;

(* Zeichensatz 8x10 *)
IMPORT CH8x10.COD;

(* Interfacedeklarationen *
(*$I'PDMODULE.DEF'*);


CONST

(* Deklaration von Konstanten*)
(* Um ein möglichst anschauliches Programmieren zu ermögli-
chen, werden die Boolean-Ausdrücke TRUE und FALSE für den
Motor mit EIN bzw. AUS gleichgesetzt. *)

EIN=TRUE;
AUS=FALSE;


(* Analog dazu werden die Boolean-Ausdrücke TRUE und FALSE
für die Ventile mit AUF bzw. ZU gleichgesetzt.*)

AUF=TRUE;
ZU=FALSE;

(* Weiters sind noch folgende Konstanten sinnvoll: *)
(* Heizleistung des Heizstabes *)
Heizleist=1000.0;

(* Wärmekapazität von Wasser *)
c_H2O=4187.0;

(* Wassermasse in Becken 1 *)
m=5.13;


VAR
```

```
(* Es wurden folgende Timer-Variablen benutzt:
     pt1:          Wartezeit für das PT1-Verhalten des
                   Pt100-Sensors
     nachlauf:     Nachlaufzeit für den Mixer
     DelayTimer:   Heizzeit für Becken 1 *)
pt1, nachlauf, DelayTimer : TIMER;

(* Verwendete Falgs:
     cl:       Wird beim Wechsel zwischen zwei Bildschirmen
               gesetzt und nach löschen des Bildschirms
               rückgesetzt
     voll:     wird gesetzt wenn Becken 2 mit Warmwasser
               gefüllt ist
     warm_ok:  bestätigt das Erreichen der Solltemperatur
               in Becken 1 *)
cl, voll, warm_ok : BOOLEAN;

(* Eswurden folgende Variablen vom Typ REAL verwendet:
     abfluss:      Benutzerdefinierte Abflußmenge aus
                   Becken 2
     anfang:       Gewichtswert bei dem Ventil 5 (Abfluß
                   aus Becken 2) geöffnet wird
     t_kalt:       Temperatur des Wassers aus Becken 3
     dTemp:        zu heizende Temperaturdifferenz in Bec
                   ken 1
     t1_soll:      Solltemperatur in Becken 1
     t2_soll:      Solltemperatur in Becken 2
     kaltwasser:   Berechnete Kaltwassermenge *)
abfluss, anfang, t_kalt,
 dTemp, t1_soll, t2_soll, kaltwasser : REAL;

(* Die Variable bild vom Typ INTEGER ist der Zähler für den
angezeigten Bildschirm *)
bild : INTEGER;


(* Die verschiedenen Funktionsgruppen werden im Netz loka-
lisiert, d.h. ein frei wählbarer Name z. B. temperaturmes-
sung wird dem P-NET-Modul PD3221 auf der Netzadresse 1,$21
zugewiesen. Analog dazu die Funktion wiegung dem Modul
PD3230 auf der Netzadresse 1,$30. IOs1 und IOs2 sind je-
weils die Gruppe von 4 Input/Output-Channels und 2 Input-
Channels am Modul PD3221 bzw. PD 3230 *)

temperaturmessung : PD3221 AT NET:(1,$21);
wiegung :           PD3230 AT NET:(1,$30);
IOs1 :              PD3221 AT NET:(1,$21);
IOs2 :              PD3230 AT NET:(1,$30);
```

```
(* Indirekte Deklaration der OUTPUTS *)
(* Wie schon in Variablen im P-NET ausführlich beschrieben
erfolgt hier die Zuweisung der tatsächlichen Variablennamen
den für den Programmierer einfacher zu handhabenden Be-
zeichnungen *)

ventil1          ->  IOs1.Digital_io_1.FlagReg[7];
ventil5          ->  IOs1.Digital_io_2.FlagReg[7];
heizung          ->  IOs1.Digital_io_3.FlagReg[7];
mixer            ->  IOs1.Digital_io_4.FlagReg[7];

ventil2          ->  IOs2.Digital_io_1.FlagReg[7];
ventil3          ->  IOs2.Digital_io_2.FlagReg[7];
ventil4          ->  IOs2.Digital_io_3.FlagReg[7];
pumpe            ->  IOs2.Digital_io_4.FlagReg[7];



(* Indirekte Deklaration der INPUTS *)
(* Analog zu den Outputs werden nun auch die Inputs dekla-
riert. In diesem Fall wird jedoch nicht Bit 7 des FlagReg
gesetzt sondern Bit 6 gelesen. *)

level_1          ->  IOs2.Digital_in_1.FlagReg[6];
level_2          ->  IOs1.Digital_in_1.FlagReg[6];


temperatur_1    ->  temperaturmessung.analog_in_1.analogin;
temperatur_2    ->  temperaturmessung.analog_in_2.analogin;


gewicht          ->  wiegung.Weight.Weight1;



(* Der folgende Block hat eine Feedback-Funktion. Ven-
til1_auf liest das InFlag, also FlagReg[6] des Channels aus
und bietet somit die Möglichkeit, festzustellen, ob Ventil
1 wirklich geöffnet ist, oder ein Fehler vorliegt. *)

ventil1_auf      ->  IOs1.Digital_io_1.FlagReg[6];
ventil5_auf      ->  IOs1.Digital_io_2.FlagReg[6];
heizung_ein      ->  IOs1.Digital_io_3.FlagReg[6];
mixer_ein        ->  IOs1.Digital_io_4.FlagReg[6];


ventil2_auf      ->  IOs2.Digital_io_1.FlagReg[6];
ventil3_auf      ->  IOs2.Digital_io_2.FlagReg[6];
ventil4_auf      ->  IOs2.Digital_io_3.FlagReg[6];
```

```
pumpe_ein        ->  IOs2.Digital_io_4.FlagReg[6];


IMPORT PD4000SP.COD;


(* Prozedur für die Abfrage der Funktionstasten. Sie hat
ursprünglich keine Funktion und wird vom Programmierer der
Anwendung entsprechend angepaßt. *)

(*$I'FKEY4000.INC'*)


(* Function zur Berechnung der Eischaltdauer des Heizstabes
laut Task 4: Heizen von Becken 1 *)

FUNCTION Time_on:real;

VAR dummy:real;

BEGIN
  IF level_1 THEN
       BEGIN
     dTemp:=T1_soll-temperatur_1;
     dummy:=M*c_H2O*dTemp/Heizleist;
(* Um einen zu kurzen Eischaltimpuls zu vermeiden wird eine
Eischaltdauer unter 0.2 Sekunden unterdrückt. *)
     IF dummy < 0.2 THEN Time_on:=0 else Time_on:=dummy;
       END
  else Time_on:=0;
END;




(*******************************************************)
(* Hier beginnt der Quellcodebereich, in dem die Tasks pro-
grammiert werden *)

(* Dieser Task muß der erste eines Programmes sein, da er
die Initialisierung des PD4000 einleitet. *)

(*$I'Init4000.INC'*)


(* Task zur Abfrage der Tastatur *)

(*$I'Key4000.INC'*)


(* Task zur Behandlung von Übertragungsfehlern *)
```

```
(*$I'InterErr.INC'*)



(*********************************************************)
(*    Beschreibung: Task 2: Levelüberwachung    *)

TASK Safety;

begin
 loop

(* Wenn Becken 1 voll ist, und Ventil 4 geschlossen, also
auch Becken 2 nicht befüllt wird, ist die Pumpe auszuschal-
ten und Ventil 3 zu schließen *)
  if (level_1=ein) and (ventil4=zu) then
  begin
  pumpe:=aus;
  ventil3:=zu;
  end;

(* Wenn Becken 2 voll ist, sind Ventil1 und zwei zu schlie-
ßen. Weiters ist, falls Ventil3 geschlossen ist die Pumpe
auszuschalten *)
  if level_2=ein then
  begin
     ventil1:=zu;
     ventil2:=zu;
     if ventil3=zu then pumpe:=aus;
  end;

(* Ist Becken 1 nicht voll, darf der Heizstab nicht einge-
schaltet sein *)
  If level_1=AUS then heizung:=aus;

  changetask;
 end;
end;



(*********************************************************)
(* Beschreibung: Task 3: Display und updating von Parametern *)
TASK Anzeigen;
```

```
BEGIN
  LOOP

(* Auswahl des von Bildschirm 1. Die Variable bild wird
durch den Keyboardtask manipuliert *)
if bild=1 then
 begin


(* Da der Keybordtask und somit die umschaltung zwischen
den Bildschirmen an irgendeinem Punkt dieses Tasks auftre-
ten kann, ist das Löschen des Bildschirms auf folgende wei-
se gelöst: *)
if cl then
     begin
     SetVideo(DefaultPen,ScreenWidth,ScreenHeight);
(*Sie Variable cl wird im Keyboardtask auf TRUE gesetzt und
nach löschen des Bildschirms wieder auf FALSE gesetzt, da
der Bildschirm sonst bei jedem Taskdurchlauf gelöscht wer-
den würde *)
     cl:=FALSE;
     end;


(* Angabe der Cursorposition *)
      PenTo(1,1);
(* Ausgabe auf dem Bildschirm *)
      Display('T1(soll)=');
(* Updating der Variable *)
      Update(T1_soll:2:0);

      PenTo(1,7);
      Display('T1=');
      Display(temperatur_1:5:2,' C');


(* Die Anzeige der Heizzeit erfolgt nur beim tatsächlichen
Ablauf des Timers *)
      if DelayTimer>=0 then
      begin
       PenTo(1,14);
       Display('Zeit=');
       Display(DelayTimer:3:0);
      end;

      PenTo(81,1);
      Display('T2(soll)=');
      Update(T2_soll:2:0);
(* Begrenzung: Eine Solltemperatur in Becken 2 größer als
60 C ist unmöglich *)
```

```
      if t2_soll>60 then t2_soll:=60;
(* Begrenzung: Eine Solltemperatur in Becken 2 kleiner als
25 C ist sinnlos. *)
      if t2_soll<25 then t2_soll:=25;
(* Falls nicht anders vorgegeben, wird die Solltemperatur
in Becken 1 um 5 C höher als in Becken 2 festgelegt. *)
      t1_soll:=t2_soll+5;


      PenTo(81,7);
      Display('T2=');
      Display(temperatur_2:5:2,' C');


  end; (* bild 1 *)


(* Auswahl des von Bildschirm 2 *)

if bild=2 then
 begin

if cl then
     begin
     SetVideo(DefaultPen,ScreenWidth,ScreenHeight);
     cl:=FALSE;
     end;


      PenTo(81,1);
      Display('Masse=',gewicht:5:3,'kg');

      PenTo(1,1);
      Display('Abfluß:');
      Update(abfluss:3:0);
      Display('g');

(* Abflußmengen unter 100 g werden sehr ungenau. *)
      if abfluss<100 then abfluss:=100;

 end; (* bild 2 *)
      ChangeTask;
  END;
END;




(*******************************************************)
(*   Beschreibung: Task 4: Heizen von Becken 1  *)
```

```
TASK Heizer_ein;
BEGIN
  LOOP

(* Die Variable warm_ok indiziert ob Becken 1 mit Heißwas-
ser gefüllt ist. Das Becken wird also befüllt, wenn der
Füllstand noch nicht erreicht ist und es nicht mit Heißwas-
ser befüllt ist. *)
  if (level_1=aus) and (warm_ok=FALSE) then
  begin
     ventil3:=auf;
     pumpe:=ein;
  end;

(* Ist der Füllstand erreicht und das Wasser noch nicht
heiß, wird der Heizstab eingeschaltet. *)
  If (level_1=ein) and (warm_ok=FALSE) THEN
  BEGIN
     HEIZUNG:=ein;
     DelayTimer:=Time_on;

(* Um einen Meßwert für das Kaltwasser aus Becken 3 zu er-
halten, wird an dieser Stelle die Temperatur des frisch be-
füllten Becken 1 gemessen. *)
     t_kalt:=temperatur_1;

(* Während der Timer abläuft, wird mittels changetask der
Task gewechselt. *)
     while (delaytimer>=0) do changetask;

(* Ist der Timer abgelaufen, wird der Heizstab aubgeschal-
tet und das Flag warm_ok auf TRUE gesetzt. *)
     HEIZUNG:=aus;
     warm_ok := TRUE;
  END; (*if*)

      ChangeTask;
  END; (*loop*)
END; (*task*)




(*********************************************************)
(*  Beschreibung: Task 5: Regelung von Becken 2 *)

TASK Becken_2_waermen;
BEGIN
  LOOP
```

```
(* Sind weniger als 300 g Wasser in Becken 2, wird dieses
als leer identifiziert. *)
  if gewicht<0.3 then voll:=FALSE;



(* Ist Becken 1 geheizt (warm_ok=TRUE) und Becken 2 leer
(voll=FALSE), beginnt der Mischvorgang. *)
  If (warm_ok=TRUE) and (voll=FALSE) THEN
     begin

(* Der Mixer läuft während der gesmten Abmischdauer. *)
     mixer:=on;

(* Berechnung der nötigen Kaltwassermenge (Schlußrechnung)
*)
      kaltwasser:=2*(T2_soll-t1_soll)/(t_kalt-T1_soll);

(* Begrenzung der Kaltwassermenge auf 2 kg *)
      if kaltwasser>2 then kaltwasser:=2;

          ventil4:=auf;
          pumpe:=ein;

(* Ventil 4 auf, Pumpe ein, bis die Kaltwassermenge abzüg-
lich des Korrekturwertes -0,047 erreicht ist. *)
          while (gewicht<kaltwasser-0,047) do changetask ;

          pumpe:=aus;
          ventil4:=zu;

          ventil1:=auf;
          ventil2:=auf;

(* Heißwasser nachfüllen, bis 1 C unter Sollwert von Becken
2. *)
          while (temperatur_2<t2_soll-1) and (gewicht<2) do
changetask;

          ventil1:=zu;
          ventil2:=zu;

(* Da der Pt100-Sensor eine relativ große Zeitkonstante
hat, wird zuerst bei 1 C unter dem Sollwert angehalten.
Nach Ablauf der Timers PT1 (8s) wird erst auf den Sollwert
hingeregelt. *)
          pt1:=8;
          while (pt1>0) do changetask;
```

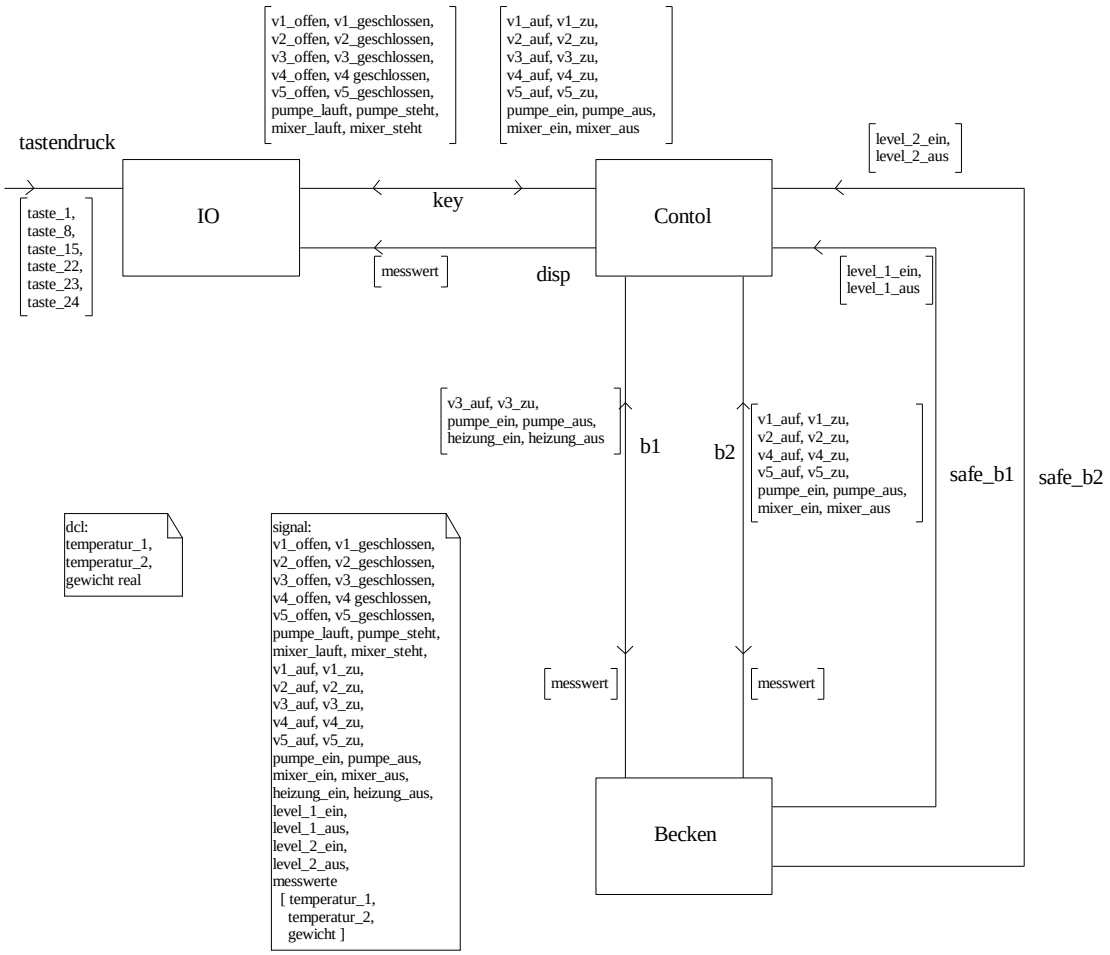
```
          ventil1:=auf;
          ventil2:=auf;

(* Heißwasser nachfüllen bis der Sollwert erreicht ist. *)
          while (temperatur_2<t2_soll) and (gewicht<2) do
changetask;

          ventil1:=zu;
          ventil2:=zu;

(* Becken 2 ist voll: voll=TRUE *)
          voll:=TRUE;

(* Becken 1 ist nicht mehr gefüllt mit Heißwasser:
warm_ok=FALSE *)
          warm_ok:=FALSE;

(* Der Mixer läft für optimale Durchmischung noch 5 s nach.
*)
          nachlauf:=5;
          while (nachlauf>0) do changetask;
          mixer:=aus;

      end; (*if*)

      ChangeTask;

  END; (*loop*)
END; (*task*)




(*******************************************************)
(* Beschreibung: Task 6: Ablassen der dosierten Menge *)

TASK Becken_2_entleeren;
BEGIN
  LOOP

(* Der Anfangswert des Gewichts wird gespeichert, wenn Ven-
til5 offen ist (ventil5=auf) und Becken 2 im voll-Zustand
ist ist (voll=TRUE)
      if (ventil5=auf) and (voll=TRUE)  then begin
       anfang:=gewicht;

(* Korrekturformel für Ausfluß *)
       while ((anfang-gewicht)<
       (abfluss/1000-0.0192-0.0188*gewicht)) do changetask;
```

```
(* nach Abfluß der gewünschten Menge: Ventil 5 schließen,
v5_offen wieder auf FALSE setzen *)
       ventil5:=zu;
      end;

      ChangeTask;

  END; (*loop*)
END; (*task*)

END.
```

ANHANG C: SDL-Diagramme

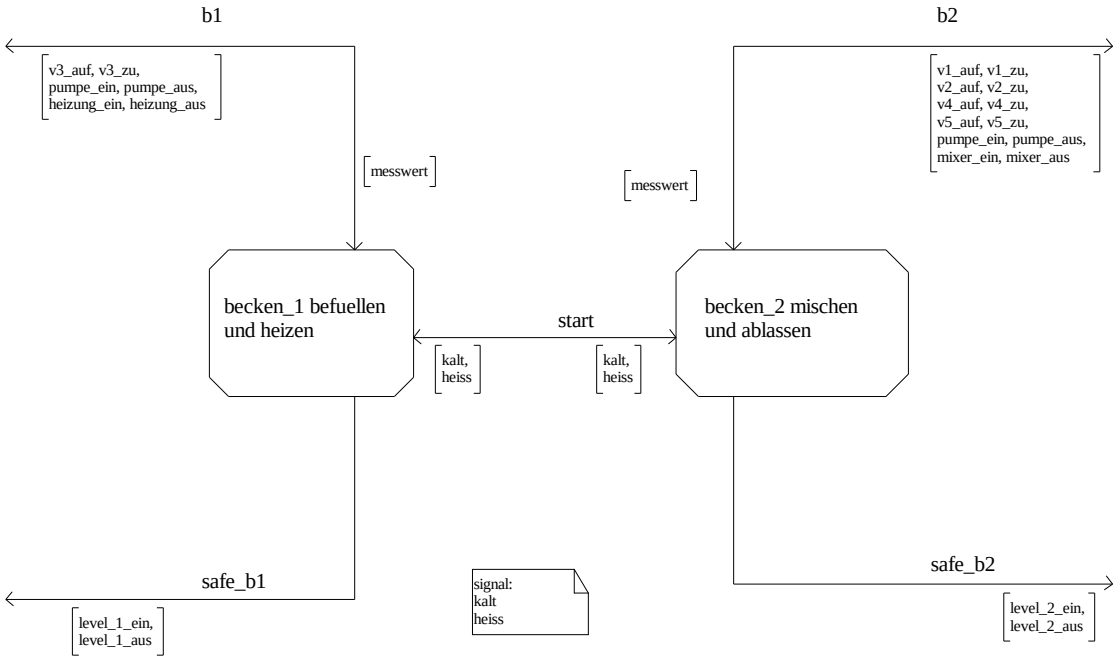
System DIPLOMARBEIT

Seite_1



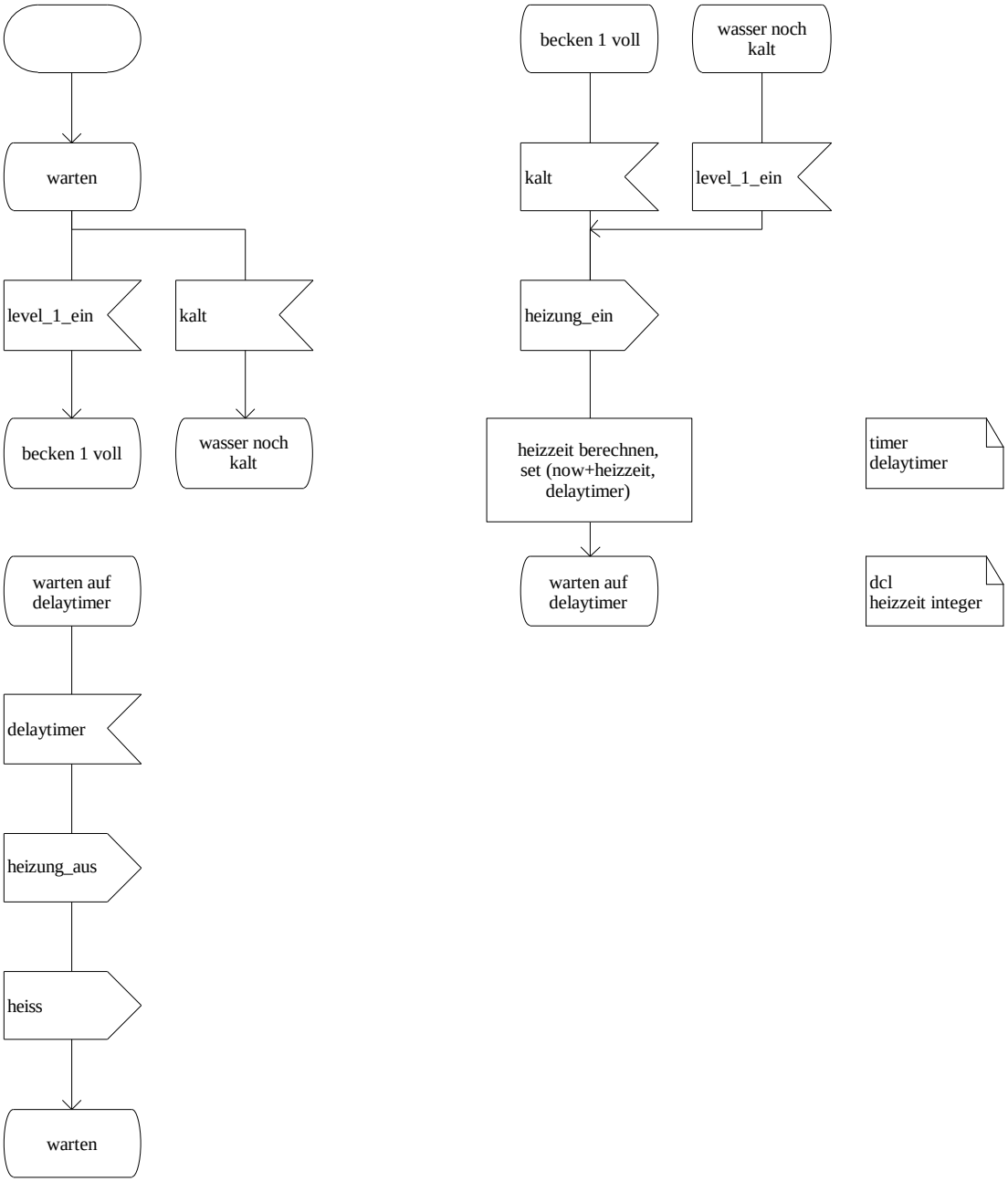
Block Becken

becken(1)



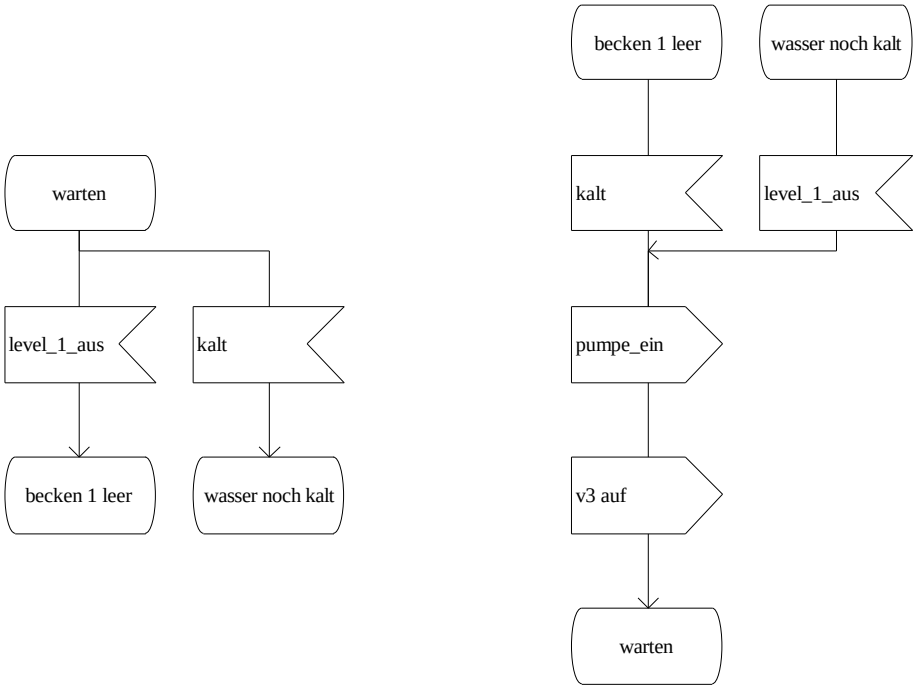
Process becken_1

heizen(3)



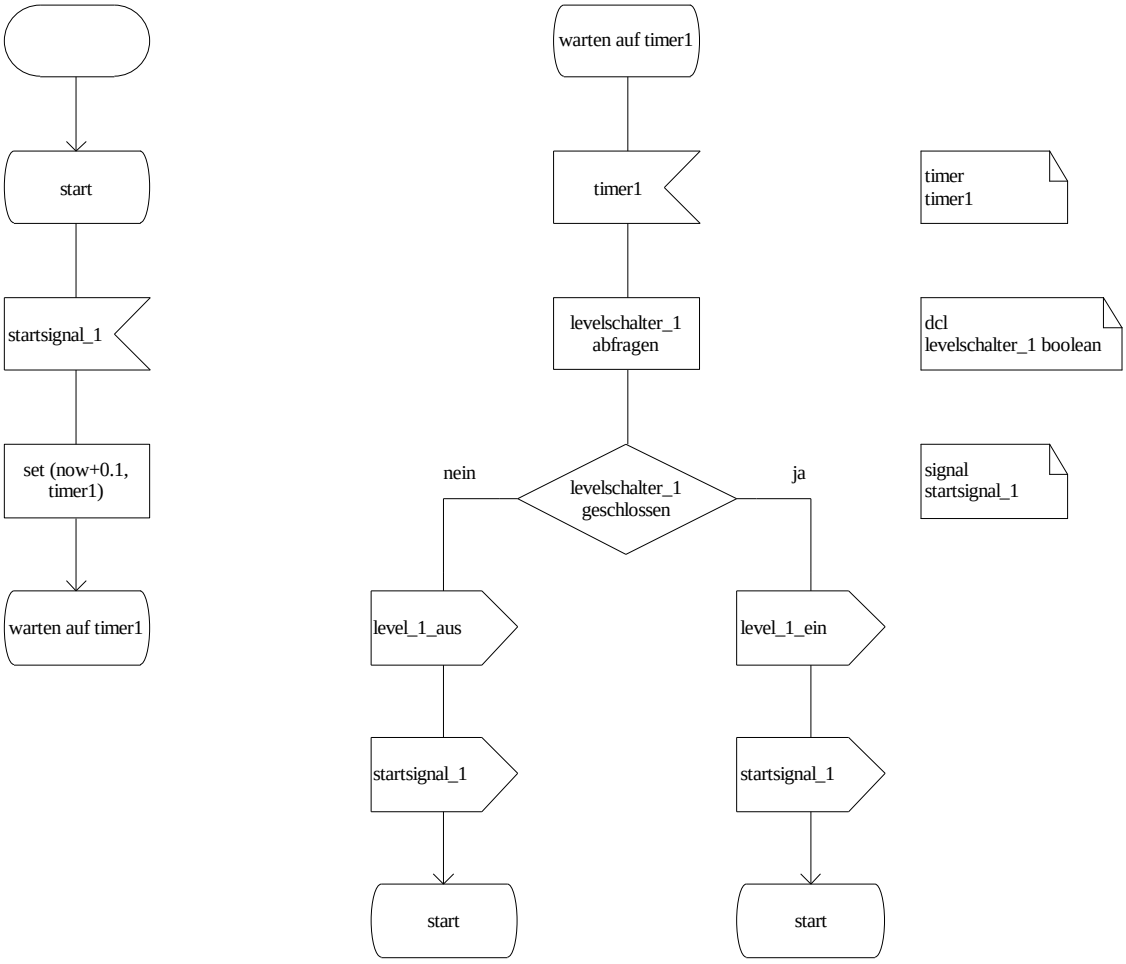
Process becken_1

befüllen(3)



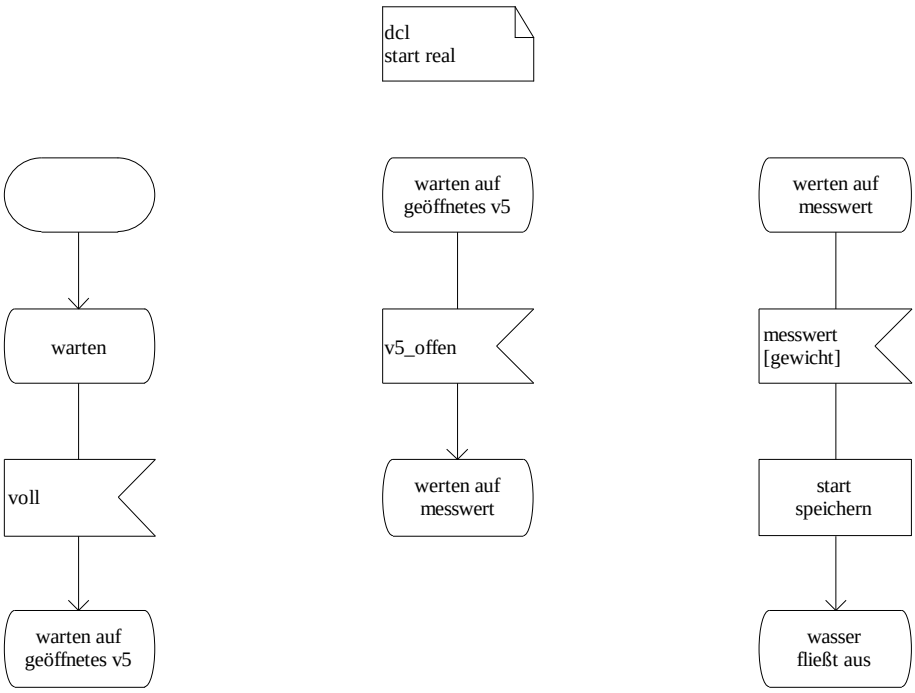
Process becken_1

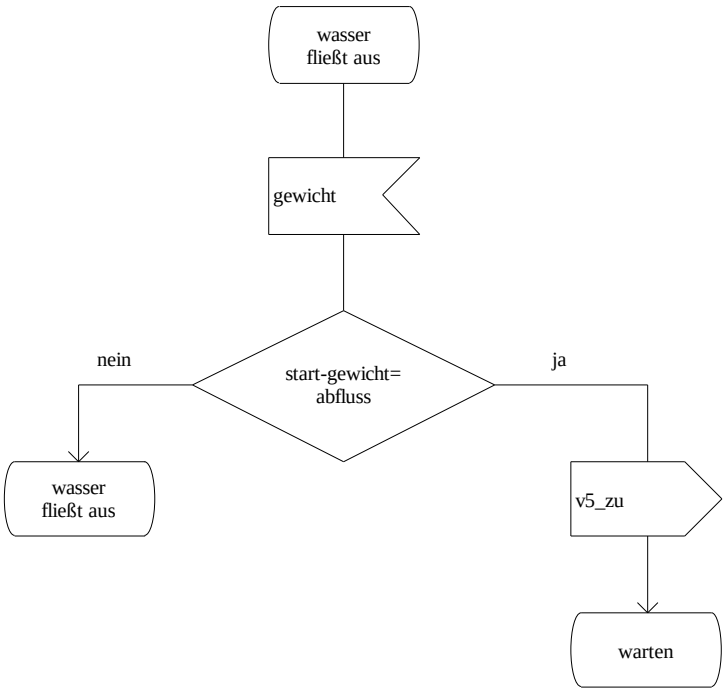
level_b1(3)



Process becken_2

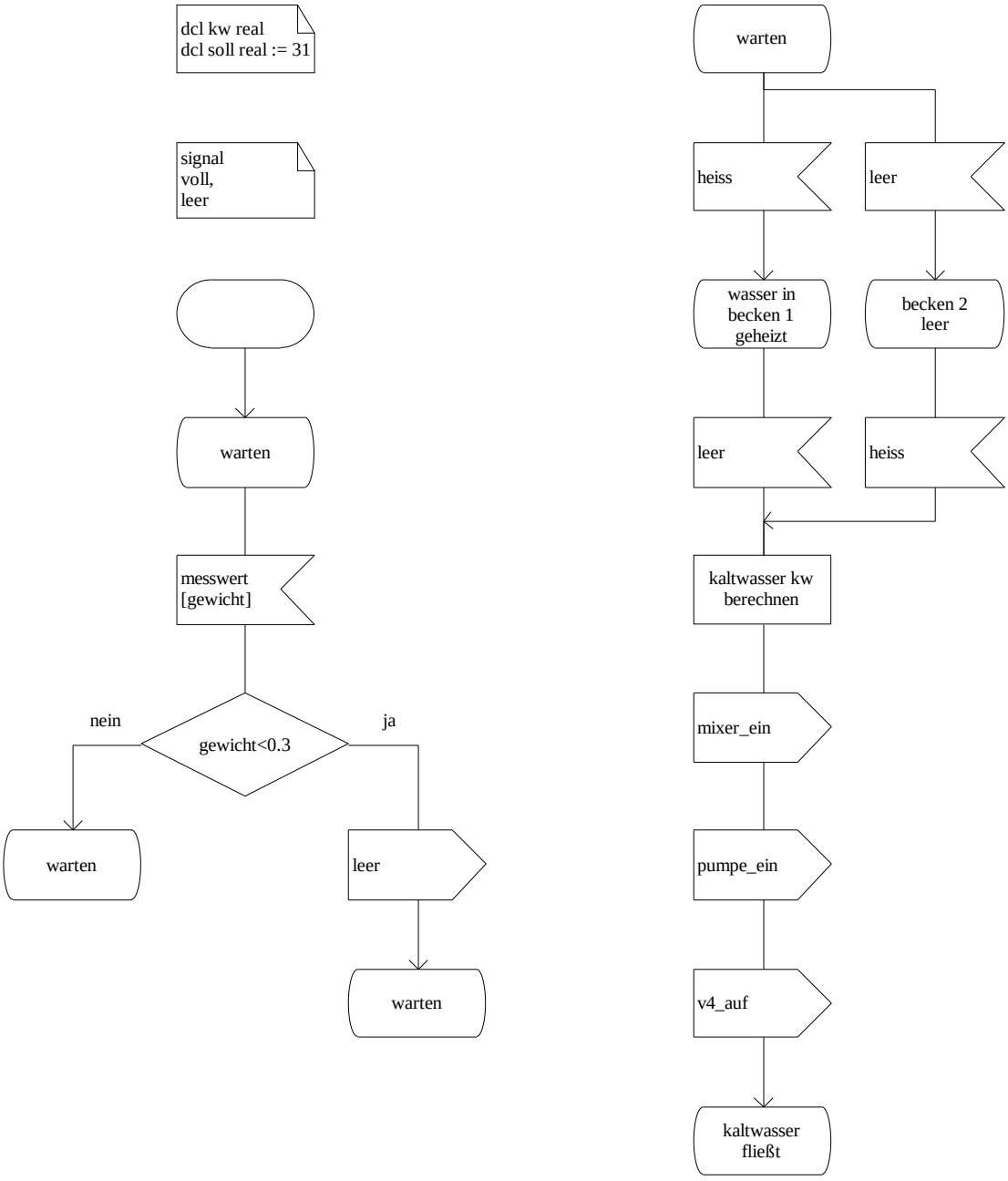
ablassen(4)





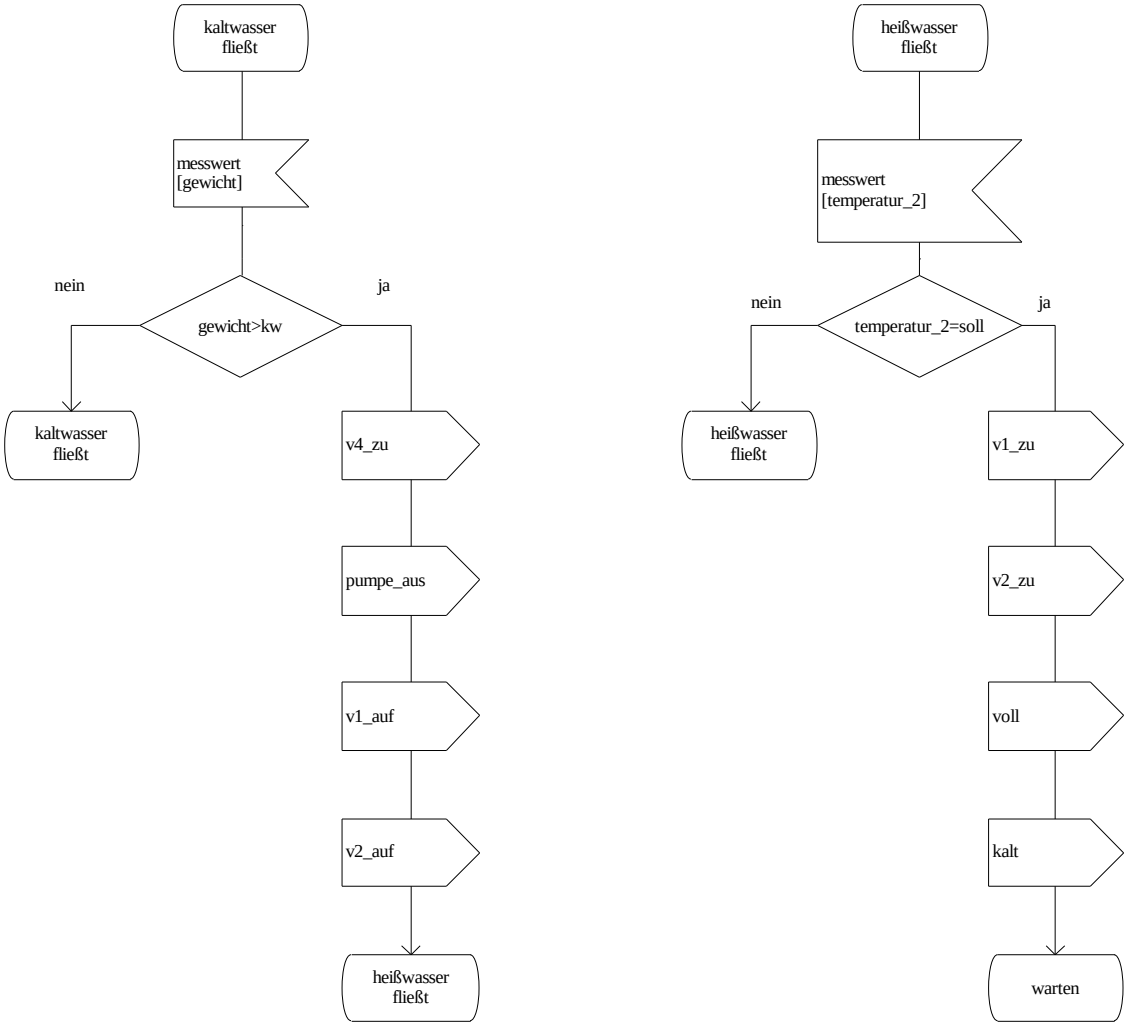
Process becken_2

mischen1(4)



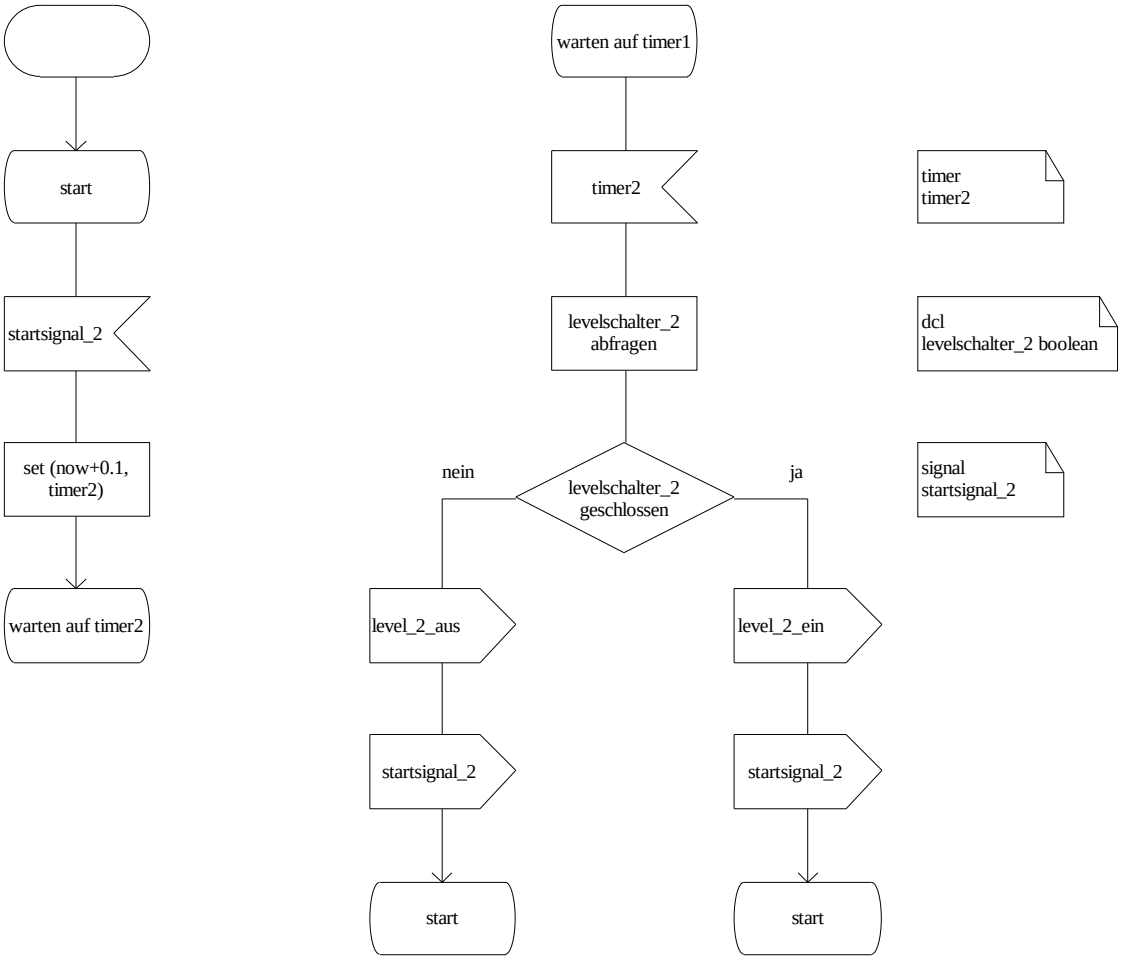
Process becken_2

mischen2(4)



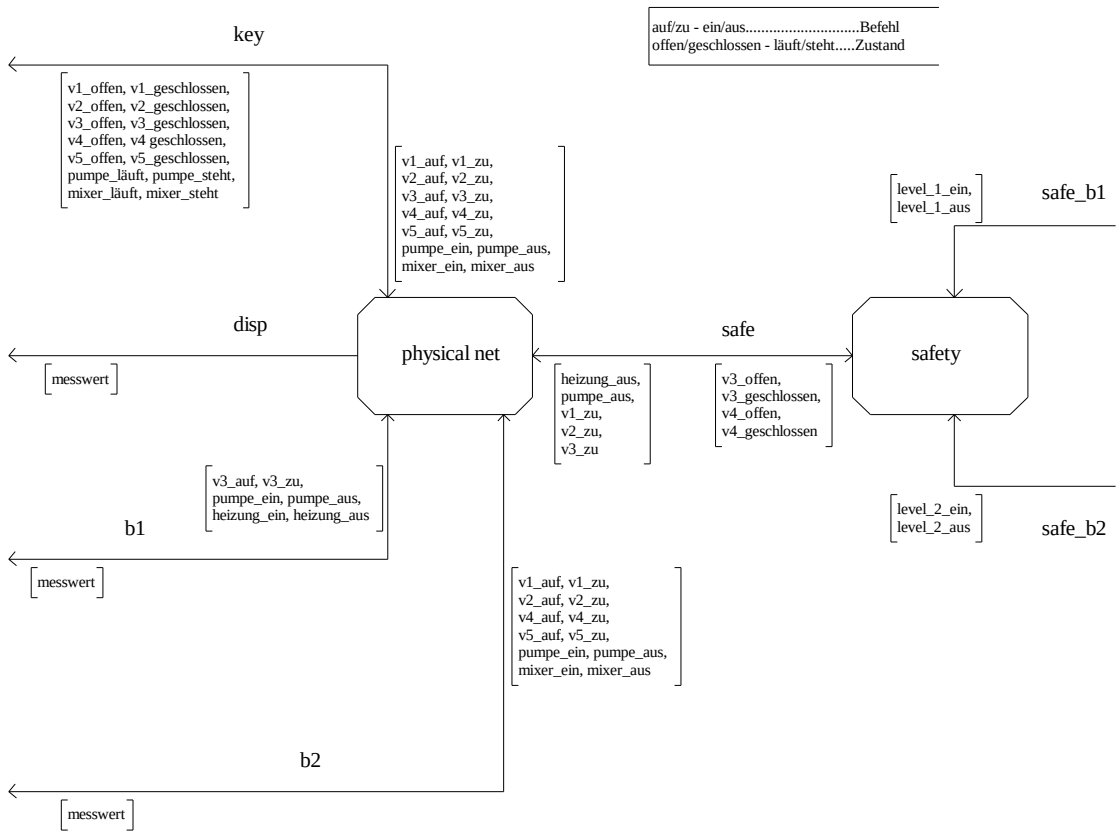
Process becken_2

level_b2(4)



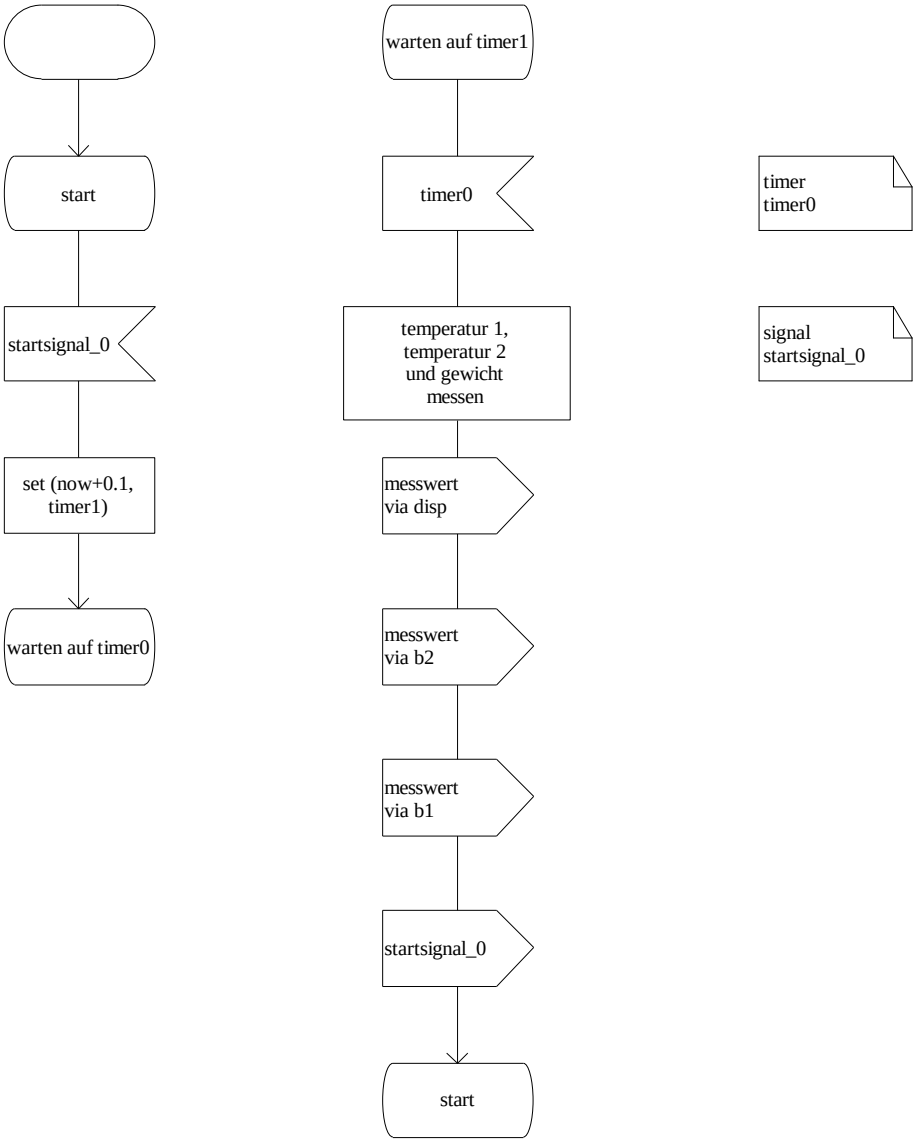
Block Control

control(1)



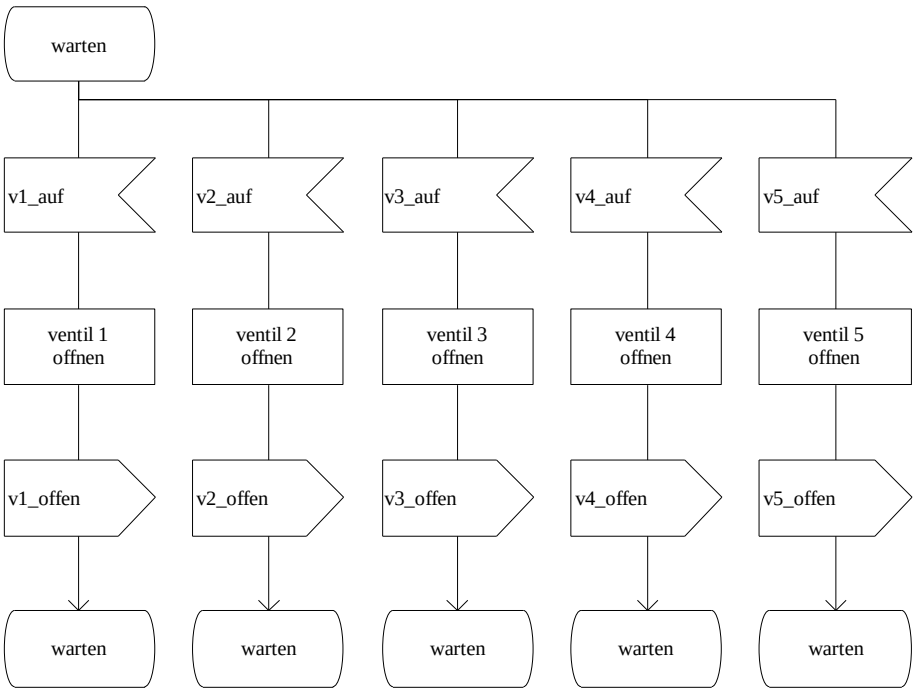
Process physical

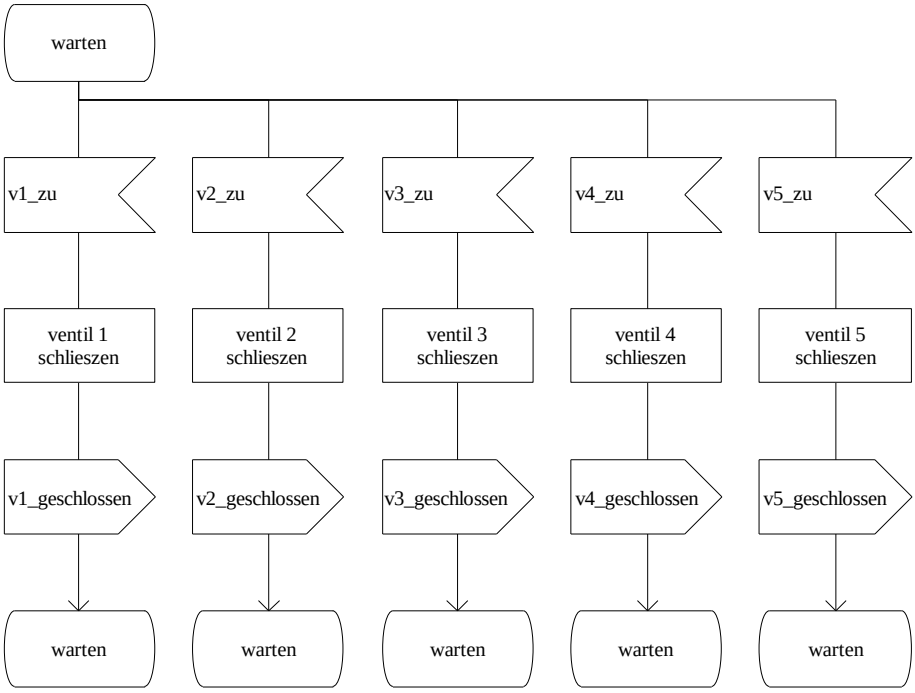
messung(3)



Process physical

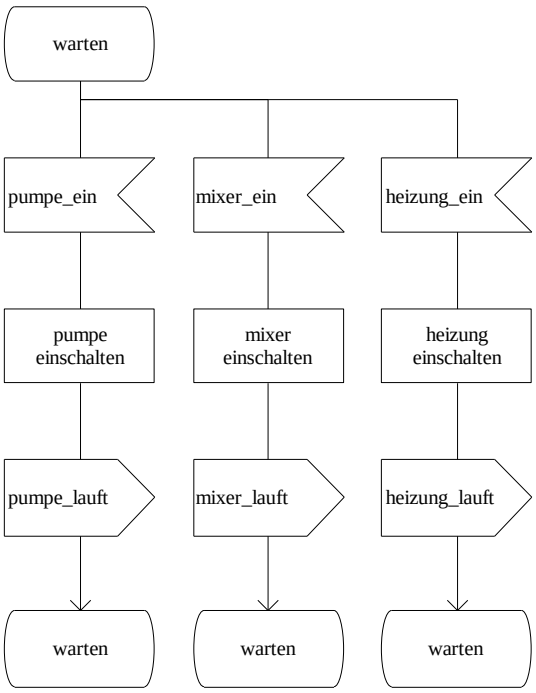
ventile(3)

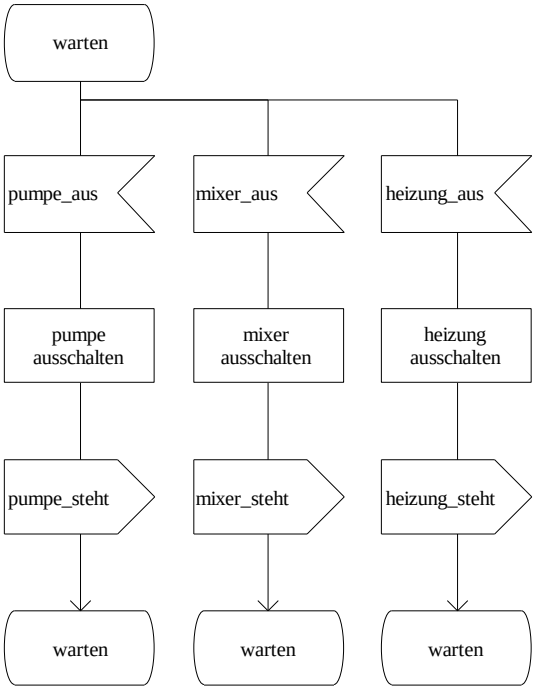




Process physical

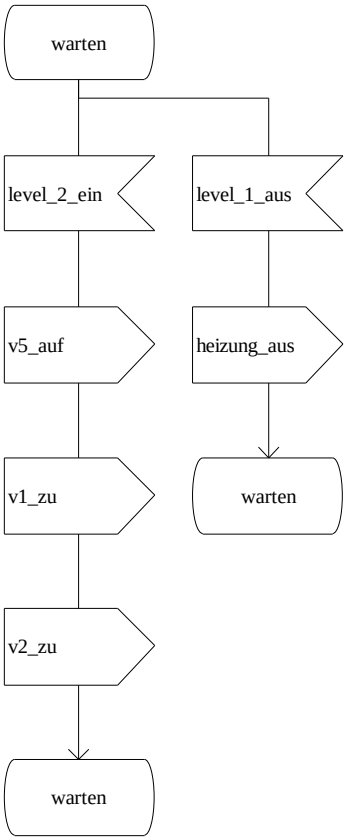
steuern(3)





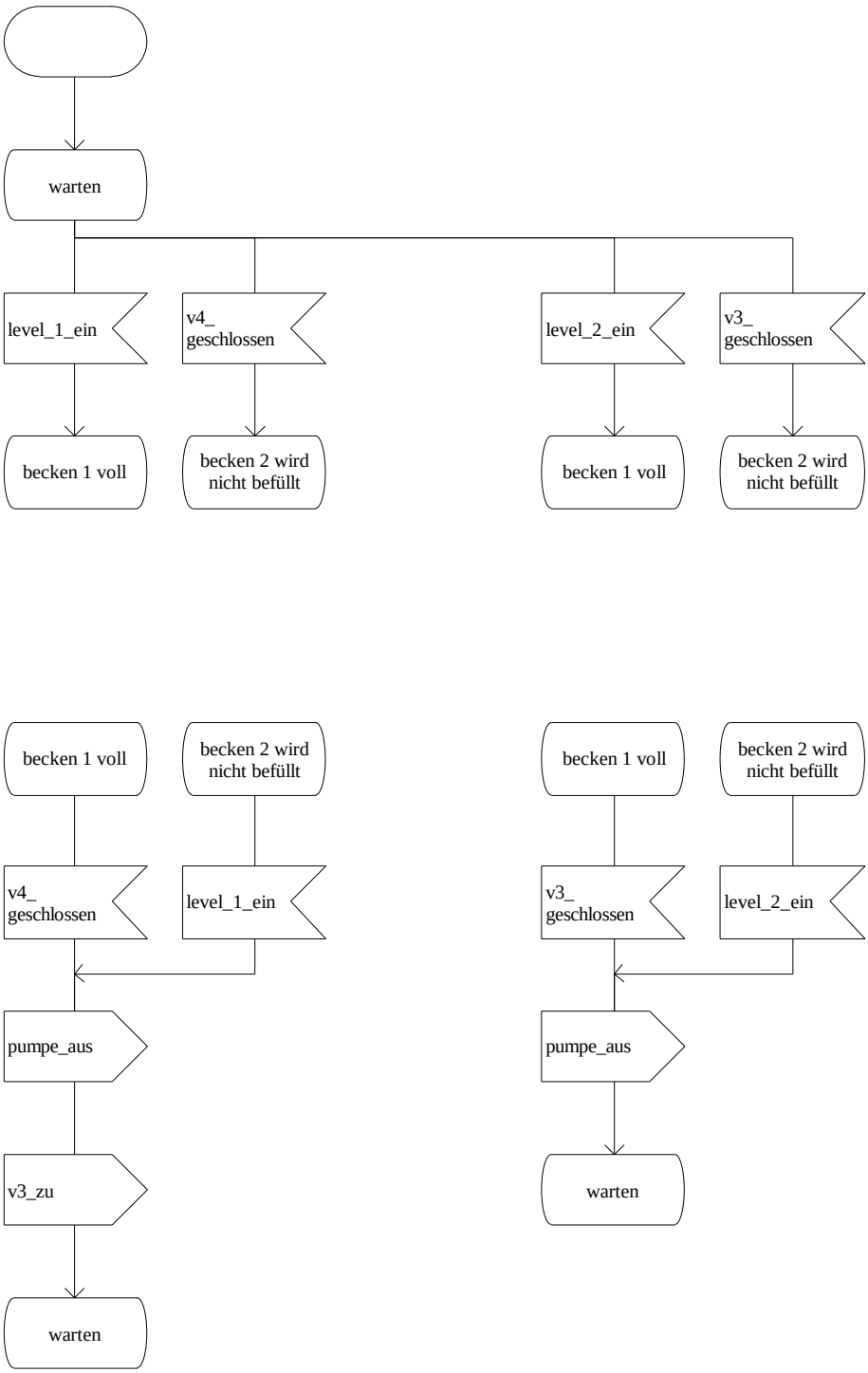
Process safety

safety2(2)



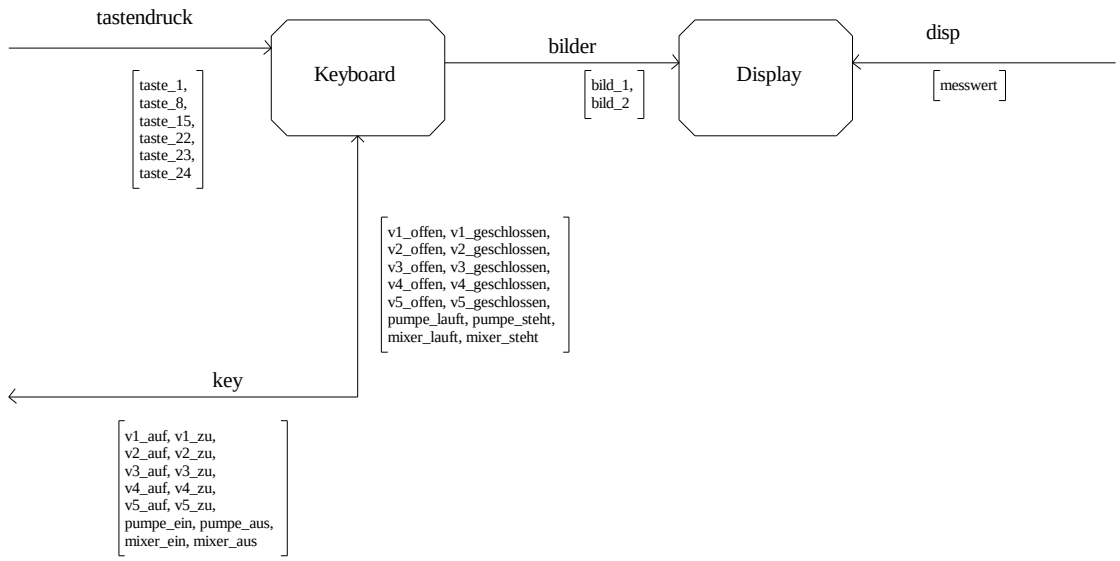
Process safety

safety1(2)



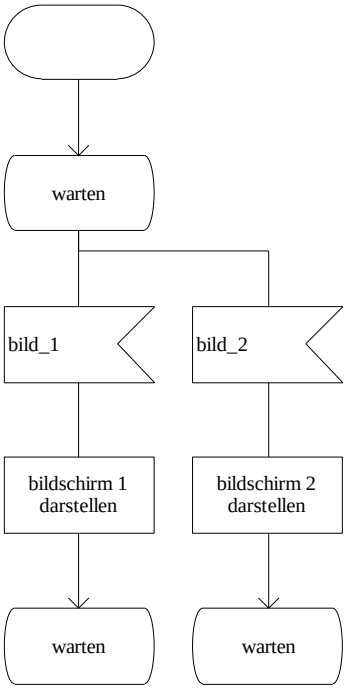
Block IO

IO(1)


tastendruck
taste_1,
taste_8,
taste_15,
taste_22,
taste_23,
taste_24
Keyboard
bilder
bild_1,
bild_2
Display
disp
messwert
v1_offen, v1_geschlossen,
v2_offen, v2_geschlossen,
v3_offen, v3_geschlossen,
v4_offen, v4_geschlossen,
v5_offen, v5_geschlossen,
pumpe_lauft, pumpe_steht,
mixer_lauft, mixer_steht
key
v1_auf, v1_zu,
v2_auf, v2_zu,
v3_auf, v3_zu,
v4_auf, v4_zu,
v5_auf, v5_zu,
pumpe_ein, pumpe_aus,
mixer_ein, mixer_aus

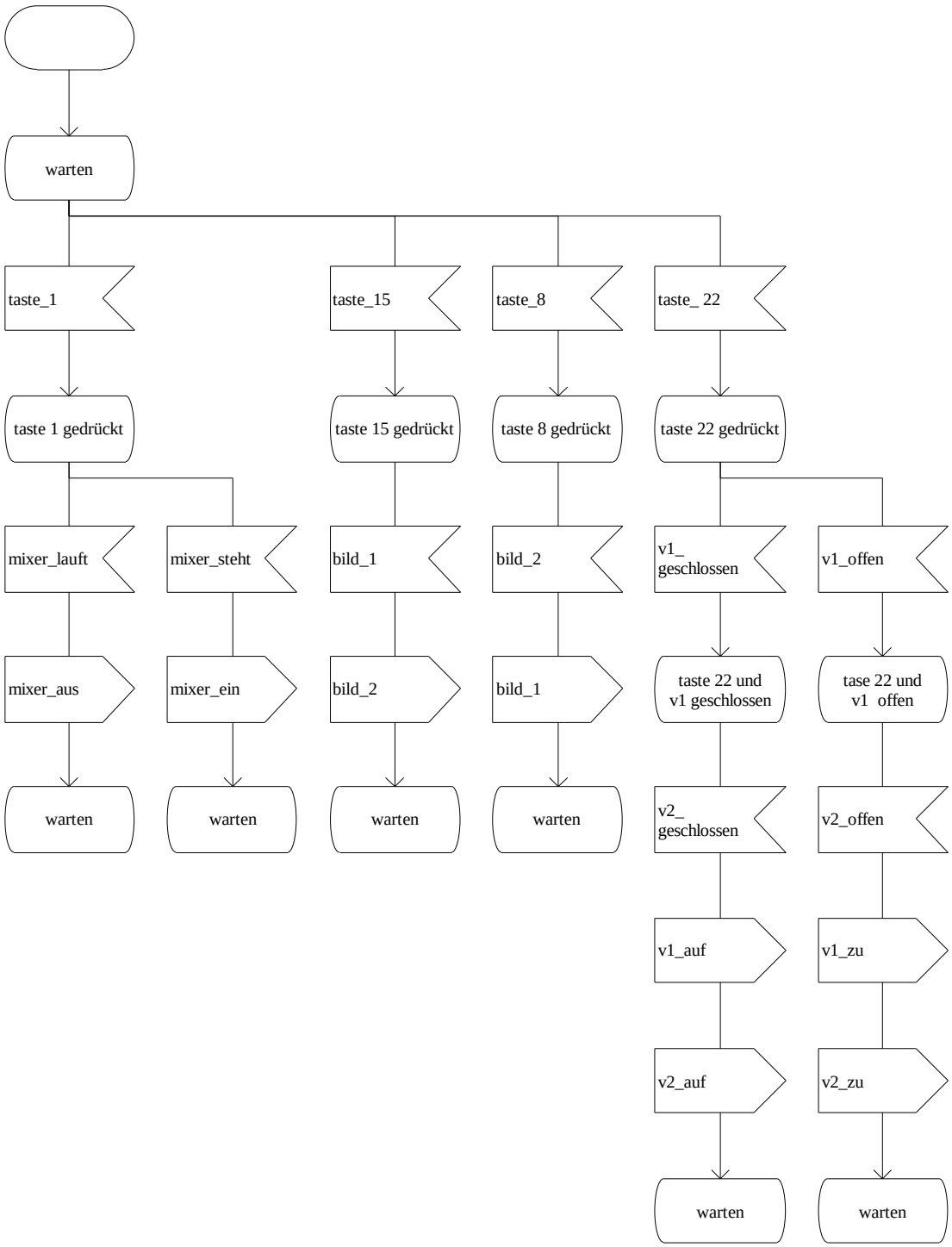
Process Display

screen(1)



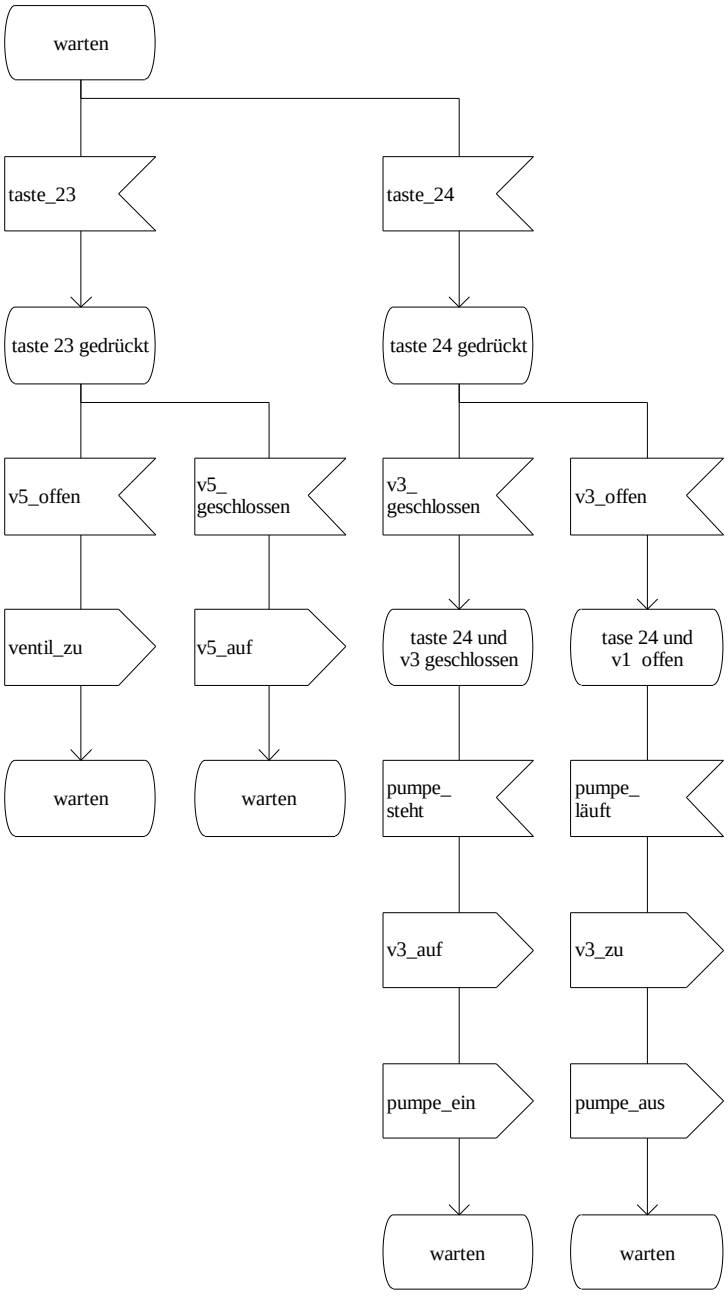
Process Keyboard

key1(2)



Process Keyboard

key2(2)



Literaturverzeichnis

Zeichen	Autor	Titel
[FBu92]	Borst, Walter	Der Feldbus Franzis Verlag, München 1992
[prEN170]	CENELEC	prEN 50170, European Standard Brussels, Belgium 1995
[PP91]	Process Data	PROCESS-PASCAL V2.0; Users Manual Silkeborg, Denmark 1991
[STA94]	Process Data	P-NET, Standard Silkeborg, Denmark 1994
[STA92]	Process Data	P-NET, Standardized general purpose channel Silkeborg, Denmark 1992
[PD93]	Process Data	PD Calculator Assembler Silkeborg, Denmark 1993
[PD30]	Process Data	Weight Transmitter, PD 3230, Manual Silkeborg, Denmark 1993
[PD21]	Process Data	Universal Process Interface, PD 3221, Manual Silkeborg, Denmark 1995
[DD93]	Dietrich, Dietmar	Bussysteme und Rechnerkommunikation Vorlesungsskriptum, Wien 1993
[BON92]	Bonfig, Walter	Feldbus-Systeme Expert Verlag, 7044 Ehningen 1992
[SDL91]	Belina, Ferenc	SDL Redwood Books, Trowbridge, Wiltshire
[GRU95]	G. Gruhler (Hrsg.)	Einführende Darstellung und detaillierter Vergleich von Feldbussystemen Alteburgstraße 150, D-72762 Reutlingen 1995

Abbildungsverzeichnis

Tabellenverzeichnis

Index

R

S

T

U

V

W