Skriptum zum
Hochschulkurs

Feldbussysteme

verfaßt von

Franz J. Gira

und

Dipl.-Ing. Mag. Martin Manninger

1996-04-24

I. Inhaltsverzeichnis

II. Abkürzungen

ASI	**A**ktuator-**S**ensor-**I**nterface
CENELEC	**C**omite **E**uropeen de **N**ormalisation **Elec**trotechnique
EEPROM	**E**lectrically **E**raseable **P**rogrammable **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
LAN	**L**ocal **A**rea **N**etwork
LON	**L**ocal **O**perating **N**etwork
NRZ	**N**on **R**eturn to **Z**ero
OSI	**O**pen **S**ystem **I**nterconnection
Profibus	**Pro**cess **Fi**eld**bus**
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
SPS	**S**peicher**p**rogrammierbare **S**teuerung
SWNo	**S**oft**w**ire **Number**
UPI	**U**niversal **P**rocess **I**nterface
VIGO	**V**irtual **I**nterface using **G**lobal **O**bjects

III. Grundbegriffe von P-NET

P-NET ist ein ***Feldbussystem***, das von der dänischen Firma *Proces-Data A/S* entwickelt wurde. Seit 1989 ist das Protokoll jedoch offengelegt und über die *International P-NET User Organization* frei erhältlich. P-NET ist mit der CENELEC-Norm EN50170 neben *Profibus* (deutsch) und *FIP* (französisch) als ***Europa-Standard*** festgelegt worden.

Das P im Namen **P-NET** steht für ***Prozeßautomatisierung***, was das primäre Anwendungsgebiet dieses Feldbussystems anzeigen soll. Nichtsdestotrotz ist P-NET jedoch ein universell einsetzbares System.

Die Grundstruktur von P-NET ist ein ***physikalischer Bus***, der zu einem ***Ring*** zusammengeschlossen ist. Mit Hilfe sogenannter *Multiport Master*, die in zwei Bussen hängen und die Funktion eines *Routers* erfüllen, können jedoch richtige ***Netzwerke*** und dadurch auch redundante Systeme aufgebaut werden.

Auf jedem Bus können bis zu 125 Knoten der Type Master oder Slave hängen. Ein ***Slave*** darf den Bus nur nach der Anfrage eines Masters zum Senden einer Antwort benutzen. Die Mediumzugriffssteuerung der ***Master*** erfolgt mittels eines Tokenverfahrens, womit ***Echtzeitfähigkeit*** garantiert ist. Master, die gerade nicht im Besitz des Tokens sind, können von demjenigen, der das Token hat, als Slave angesprochen werden

Controller sind die intelligenten Geräte am Bus. Sie können in der Hochsprache ***Process-Pascal*** programmiert werden, wobei die Softwareentwicklung selbstverständlich am PC erfolgt. Doch auch die „unintelligenten“ Knoten, wie etwa I/O-Einheiten, sind intelligenter als bei vergleichbaren anderen Feldbussystemen. Sie sind einerseits in einer vereinfachten Assemblersprache namens ***Calculator Assembler*** programmierbar und erledigen andererseits gewisse Aufgaben wie die Umrechnung von Meßwerten in SI-Einheiten oder eine PID-Regelung selbsttätig.

Jedes P-NET-Modul beinhaltet mehrere ***Channels***, die eine objektartige Gruppierung von mehreren Werten wie Ein- und Ausgangswerte, Grenzwerte, etc. sind. Die wichtigsten Channelarten sind standardisiert und finden sich daher in P-NET-Modulen verschiedener Hersteller in gleicher Weise.

Die für P-NET entwickelte Software ***VIGO*** bildet Objekte am P-NET unter Windows ab, sodaß über OLE jede Windows-Applikation (z.B. *Excel* oder ein kleines *Visual Basic*-Programm) ganz einfach auf das P-NET zugreifen kann. Eine weitere Möglichkeit zur Visualisierung ist der Controller PD5020, der über Anschlüsse für Tastatur, Maus und VGA-Monitor verfügt und mit der

Software ***GCS*** erlaubt, innerhalb weniger Minuten eine ansprechende Visualisierung eines Prozesses zu erzeugen.

IV. Allgemeine Bewertung und kurzer Vergleich

P-NET besticht durch viele intelligente Detaillösungen und Bedienerfreundlichkeit. Hier gibt es große Ähnlichkeiten mit dem amerikanischen System *LON*, das aber ein total dezentrales Konzept ohne Unterscheidung von Master oder Slave verfolgt. Andere Feldbussysteme setzen noch immer auf die SPS als zentrales Element, was jedoch schlechte Strukturierbarkeit der Programme und hohe Komplexität beim Aufbau von Master-zu-Master-Verbindungen bedingt.

Die Multi-Master-Fähigkeit und die universelle Anwendbarkeit hat P-NET mit *PROFIBUS-FMS* gemeinsam, wobei aber im zweiten Punkt der PROFIBUS noch besser dasteht. Der P-NET-Standard ist ja ähnlich wie bei *Interbus-S* sehr restriktiv, wodurch zwar einerseits Interoperabilitätsprobleme vermieden werden, aber andererseits gewisse Möglichkeiten verlorengehen. Über die Interoperabilität hinaus ist bei P-NET auch die Austauschbarkeit von Modulen sehr weit fortgeschritten. Durch die Festlegung der *Channels* kann beispielsweise ein Gerät der Firma X, das den Channel *Digital_IO_1*6 beinhaltet, einfach durch ein Gerät der Firma Y ausgetauscht werden, wenn dort der selbe Channel verwendet wird. Es sind keine Änderungen in den Programmen anderer Knoten nötig.

Von der Geschwindigkeit her kann P-NET mit einfachen Single-Master-Systemen wie *Interbus-S* oder *ASI* natürlich nicht mithalten, sehr wohl aber mit *PROFIBUS-FMS*. Darüber hinaus erlaubt P-NET im Gegensatz zu den meisten anderen Feldbussystemen den Aufbau von richtigen Netzwerken.

Die größte Schwäche von P-NET ist die derzeit noch relativ geringe Verbreitung. Dies sollte sich aber durch die Festschreibung als Europa-Norm in den nächsten Jahren ändern. Derzeit sind weltweit über 5000 P-NET-Installationen im Einsatz. Primäre Einsatzgebiete sind überall dort, wo analoge Prozeßdaten erfaßt und ausgewertet werden müssen, beispielsweise in der Verfahrenstechnik, in der Lebensmittelindustrie, in der Umwelttechnik und sogar in Tanklastwägen. P-NET-Module können zurzeit von über 60 Firmen aus 13 Ländern bezogen werden.

V. ISO-OSI-Grundlagen

Die Idee hinter dem OSI Referenzmodell (Bild 1) ist die 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** (Application Layer)
6	**Darstellungsschicht** (Presentation Layer)
5	**Kommunikationssteuerungsschicht** (Session Layer)
4	**Transportschicht** (Transport Layer)
3	**Vermittlungsschicht** (Network Layer)
2	**Sicherungsschicht** (Data Link Layer)
1	**Bitübertragungsschicht** (Physical Layer)

Abb. 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, wie 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.

VI. Bitübertragungsschicht

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

Bei P-NET wurde auf den allgemein üblichen EIA RS-485 Standard zurückgegriffen. Um die in diesem Standard vorgeschriebenen Abschlußwiderstände zu vermeiden, wird jedoch das Kabel zu einem physikalischen Ring geschlossen. Dadurch kann auch eine höhere Anzahl von Knoten als die 32 im RS-485 Standard erlaubten angeschlossen werden [EN50170, S. 75]. Nach oben begrenzt ist die Knotenanzahl pro Bus auf jeden Fall dadurch, daß nur 7 Bit für die Adressierung innerhalb eines Busses zur Verfügung stehen. Dies läßt also Adressen von 0 bis 127 zu. Die Werte 0[1], $7E[2] und $7F[3] sind jedoch für Sonderfunktionen reserviert.

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

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

P-NET codiert physikalisch 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 Baud (+/-0.2%).

[1] Fabriksneue Slaves sind immer auf Adresse 0 eingestellt und können so, ohne einen Konflikt zu erzeugen, an einen bestehenden P-NET-Bus angeschlossen werden.
[2] Das Setzen der Knotenadresse kann so erfolgen, daß die gewünschte Adresse zusammen mit der Seriennummer des Moduls auf die Adresse $7E geschickt wird. Alle Knoten am Bus erhalten diese Nachricht, aber nur derjenige, der seine Seriennummer erkennt, beachtet sie.
[3] Dies ist die Broadcast-Adresse.

VII. Sicherungsschicht

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

Da das ISO-OSI-Modell aber nur von Punkt-zu-Punkt-Verbindungen ausgeht und Mehrpunktverbindungen, wie sie Busse ja darstellen, außer acht läßt, ist die wichtige Frage der Mediumzugriffssteuerung in den OSI-Normen nicht behandelt. Bei LANs wird deshalb die Sicherungsschicht in zwei Teilschichten unterteilt, von denen die eine (LLC-Teilschicht, Logical Link Control) die Datensicherung übernimmt und die andere (MAC-Teilschicht, Medium Access Control) die Mediumzugriffssteuerung. Eine solche Teilung ist auch bei Feldbussystemen sinnvoll, wird aber im P-NET-Standard nicht beschrieben.

Bei P-NET hat die Sicherungsschicht im wesentlichen folgende Aufgaben [EN170, S. 86]:

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

VII. 1. Virtual Token Passing

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*) von 1 bis 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.

Abb. 2 zeigt dazu ein einfaches Beispiel eines Systems mit 3 Master-Knoten:

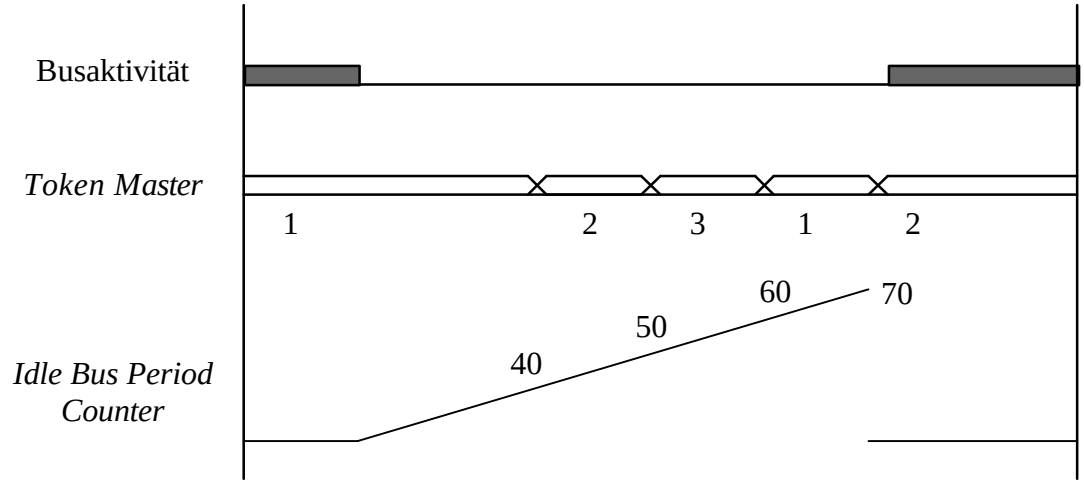


Abb. 2: Virtual Token Passing mit 3 Master

In diesem Fall hat zu Beginn Master 1 den Token und benutzt den Bus. Nachdem 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 virtuell an Master 2 übergeben. Nachdem dieser jedoch in der Zeitspanne von 10 Bitperioden nicht aktiv wird, geht der Token beim Stand von 50 über zu Master 3, nach weiteren 10 Bitperioden weiter zu Master 1 und dann wieder zu Master 2. Dieser hat in der Zwischenzeit Verwendung für den Bus und kann ihn für genau **eine** Anfrage benutzen.

VII. 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, daß 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 ⇒
Dauer einer Bitperiode:	T_{BIT} = 1/76800 = 13µs
Die Antwort muß nach maximal 30 Bitperioden einlangen:	30*T_{BIT} = 390µs

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

VII. 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 doch wieder überein, gilt dieser Master wieder als *in synchronisation*. Ist der Offset gleich dem des ersten Vergleichs, wird der Zugriffszähler der empfangenen Token Master Adresse gleichgesetzt und der Master ist ebenfalls wieder *in synchronisation* [EN50170, S. 89].

Um auch bei schwachem Busverkehr die Synchronisation zu wahren, muß, sobald der Leerlaufzähler den Wert 360 erreicht, der nächste *Token Master* ein Synchronisationspaket senden, wenn er keine anderweitige Verwendung für den Bus hat.

VII. 4. Erstellen und Erkennen von Datenpaketen

Die Datenübertragung am Bus erfolgt in Datenpaketen *(Frames)*. Diese beinhalten Ziel- und QuellAdressen, Kontroll- und Errorbytes und Nutzdaten.

Adreßfeld	Control/Status	Nutzdatenlänge	Nutzdaten	Checksumme
2-24 Byte	1 Byte	1 Byte	0-63 Byte	1-2 Byte

Abb. 3: Format eines Datenpaketes

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.

Einfacher Adreßtyp :

0	Zieladresse
1	Quelladresse

Abb. 4: Einfacher Adreßtyp

Dieser Adreßtyp für einfache Knoten 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).

Komplexer Adreßtyp:

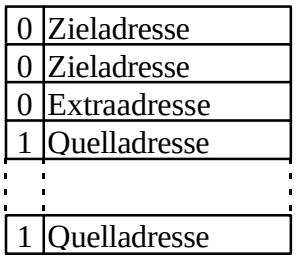


Abb. 5: Komplexer Adreßtyp

Diese Art von Adreßfeld 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 **Control/Status-Byte** hat beim Senden eines Datenpakets von Master zu Slave die Funktion eines Kommandos, das vom Slave ausgeführt wird. Es gibt u.a. Kommandos zum Laden, zum 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.

VIII.Vermittlungsschicht

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

VIII.1. Multiport-Master

In P-NET gibt es Master mit zwei Ports, sogenannte *Multiport Devices* oder auch *Dualport 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.

VIII.2. Routing

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 Masters 1 folglich P1 als Quelladresse eingeschrieben (Schritt 2). Umgekehrt wird bei Verlassen von 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 das Senden der Antwort kann nun dieses Adreßfeld, das am Ende nur noch aus Quelladreßbytes besteht, ohne weiteren Aufwand als Zieladreßfeld verwendet werden.

Der Nachteil dieses zugegeben sehr einfach handzuhabenden Routingverfahrens ist, daß bei einer Änderung im Aufbau des Netzes auch alle Programme geändert werden müssen, die auf entfernte Knoten zugreifen, zu denen der bisherige Weg nun nicht mehr existiert. Das Datenpaket kann nicht automatisch auf einen anderen Weg umgeleitet werden, auch wenn dies von der Struktur des Netzwerkes her möglich wäre. Die Programmänderungen beschränken sich jedoch durch die günstigen Eigenschaften von *Process-Pascal* auf den Deklarationsteil.

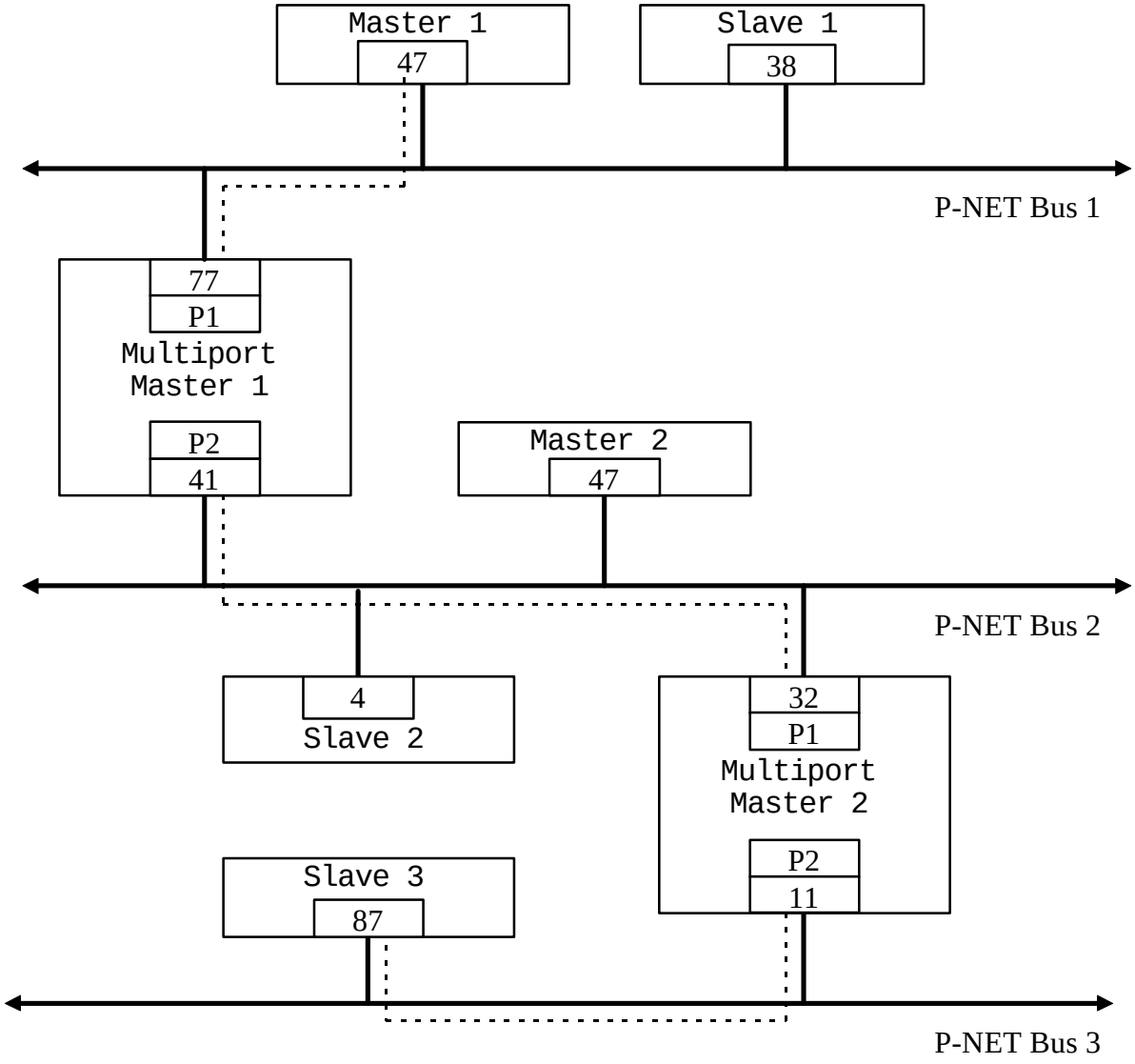


Abb. 6: Routing im P-NET

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

IX. 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, Meßwerten, SI-Einheiten, etc., 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 [EN170, S. 98].

X. Anwendungsschicht

Aufgabe laut ISO:
Konkretisierung, Ausführung der Anwendung.

Diese oberste Schicht kann nun auf die von der Transportschicht angebotenen globalen Variablen zugreifen.

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 [EN170, S. 109].

XI. 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 weiterverarbeitet 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 Cannels 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].

XI. 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 prinzipiell nicht beschrieben werden.

Read Protected Write (RPW)

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

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

XI. 2. Der Service-Channel (Channel 0)

Dieser Channel beinhaltet die ersten 16 festgelegten Softwire Numbers. Da jedes Gerät diesen Kanal besitzt, ist es über Channel 0 möglich, sogar ein unbekanntes und nicht näher spezifiziertes Gerät anzusprechen. Er muß daher in jedem *Interface Modul* vorhanden sein und weist folgende Struktur auf [STA92]:

SWNo	Identifier	Art des Speichers	Format	Type	Einheit
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	LongInt	
8	WDTimer	RAM Read Write	dezimal	Real	s
9	ModuleConfig	EEPROM Read		Record	
A	WDPreset	EEPROM Read	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. 1: Struktur des Service Channels

Dazu eine kurze Erläuterung der einzelnen Softwire Numbers:

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

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

SWNo. 3: *Reset*: Nach dem Einschreiben von $FF führt das Gerät (nach dem 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 *Watchdog-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 Watchdog Funktion.

SWNo. A: *WDPreset* gibt die maximale Zeit an, die zwischen zwei Aufrufen vergehen darf, ohne daß die Watchdog 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. Watchdog) in diesem Channel verwendet werden.

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

XI. 3. Der digitale I/O-Channel

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

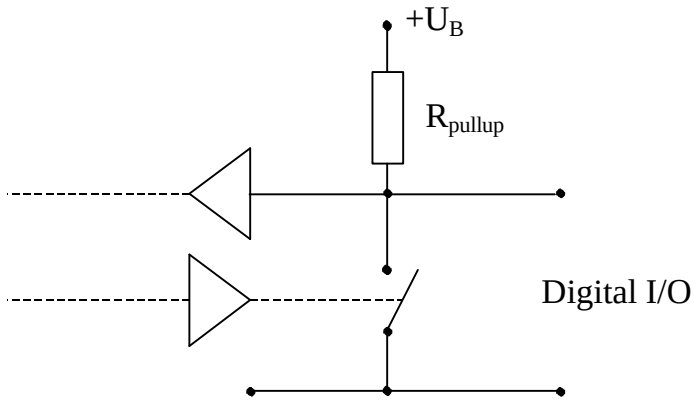


Abb. 8: Prinzipschaltbild des digitalen I/O Ports

Dazu sind folgende Variablen definiert:

SWNo	Identifier	Art des Speichers	Format	Type	Einheit
x0	FlagReg	RAM Read Write	binär	Array	
x1	OutTimer	RAM Read Write	dezimal	Real	s
x2	Counter	RAM Auto Save	dezimal	LongInt	
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. 2: Struktur des digitalen I/O Channels

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).

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 des 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, ...*

XI. 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].

Dazu sind folgende Variablen definiert:

SWNo	Identifier	Art des Speichers	Format	Type	Einheit
x0	AnalogIn	RAM Read Write	Decimal	Real	
x1					
x2					
x3					
x4					
x5					
x6					
x7	HighLevel	RAM initEEPROM	Decimal	Real	
x8	LowLevel	RAM initEEPROM	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. 3: 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* legt die Funktion des Channels, die möglichen Errors und die Eingangsfilterzeitkonstante fest.

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 Temperatur-messung, 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 wie: *SignalHigh, LowAlarm, ModuleError,...*

XI. 5. Der Weight-Channel

Dieser Channel bietet die Möglichkeit, zwei Größen zu wiegen:

- eine für einen bestimmten Meßzeitraum konstante Masse: *Weight Mode*
- einen Massefluß (z. B. über ein Förderband): *Belt Weight Mode*

Die folgenden Erläuterungen beziehen sich im wesentlichen auf den Weight Mode [PD30].

SWNo	Identifier	Art des Speichers	Format	Type	Einheit
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 initEEPROM	Decimal	Real	kg
x8	LowLevel	RAM initEEPROM	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. 4: Struktur des Weight Channels

SWNo. x0: *Weight0* ist das Nettogewicht, vermindert um eine 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*

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 ausschließlich 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* legt die Funktion des Channels (Precision, Industrial, Belt Weight...), die verwendeten Alarmflags und die Samplezeit fest und bietet weiters die Möglichkeit eine Mittelwertbildung.

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,...*

XII. PROCESS-PASCAL

Die Programmierung der einzelnen P-NET Knoten erfolgt in der Hochsprache PROCESS-PASCAL, das im wesentlichen vom weit verbreiteten ISO7185-STANDARD-PASCAL abgeleitet ist. Die wichtigsten Erweiterungen dazu sind:

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

XII. 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

Abb. 9 zeigt die Bearbeitung von vier *CYCLIC TASKS*, also Tasks, die zyklisch bearbeitet werden. Die Abfolge ist durch die Reihung der Tasks im Quelltext des Programmes bestimmt.

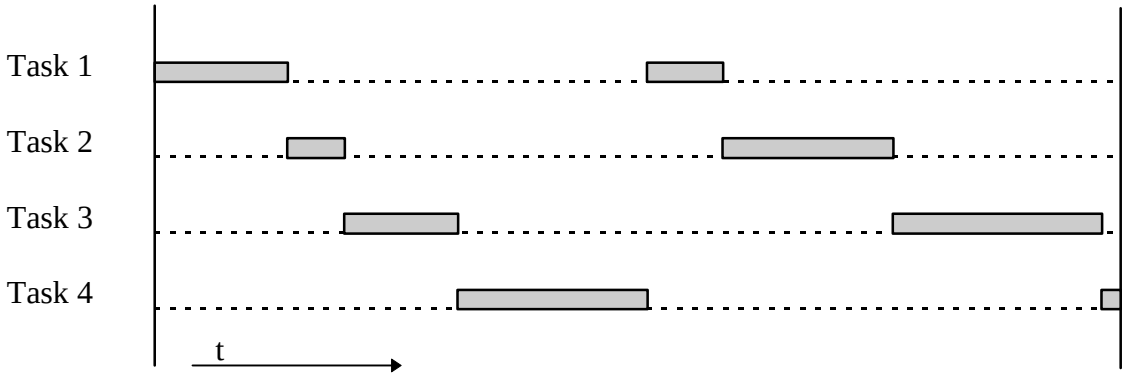


Abb. 9: CYCLIC TASKs

Außer *CYCLIC TASKs* existieren noch zwei weitere Arten: *TIMEDINTERRUPT TASKs* und *SOFTWIREINTERRUPT TASKs*.

Wie der Name schon sagt, wird ein *TIMEDINTERRUPT TASK* immer in festgelegten Zeitintervallen, und der *SOFTWIREINTERRUPT TASK* bei Auftreten eines bestimmten Zustandes (z. B. LevelMax überschritten) aufgerufen.

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 (Abb. 10).

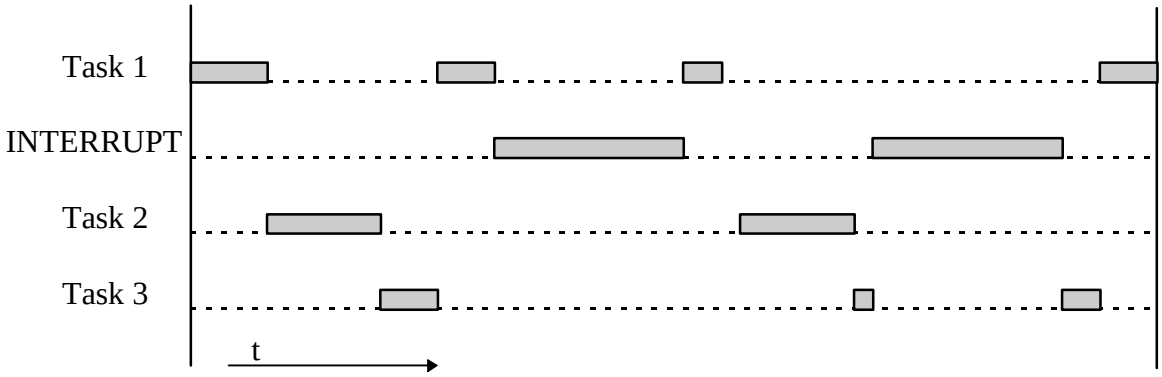


Abb. 10: CYCLIC TASK unterbrochen von SOFTWIRE- bzw. TIMEDINTERRUPT TASKs

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).

XII. 2. Programmaufbau

Dies ist z.B. der Aufbau eines PROCESS-PASCAL - Programmes mit 2 Tasks (Abb. 11).

Auf globale Variablen kann von jedem Task zugegriffen werden, dies ermöglicht einen Datenaustausch der Tasks mit der Außenwelt.

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

```
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 Programmes

XII. 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

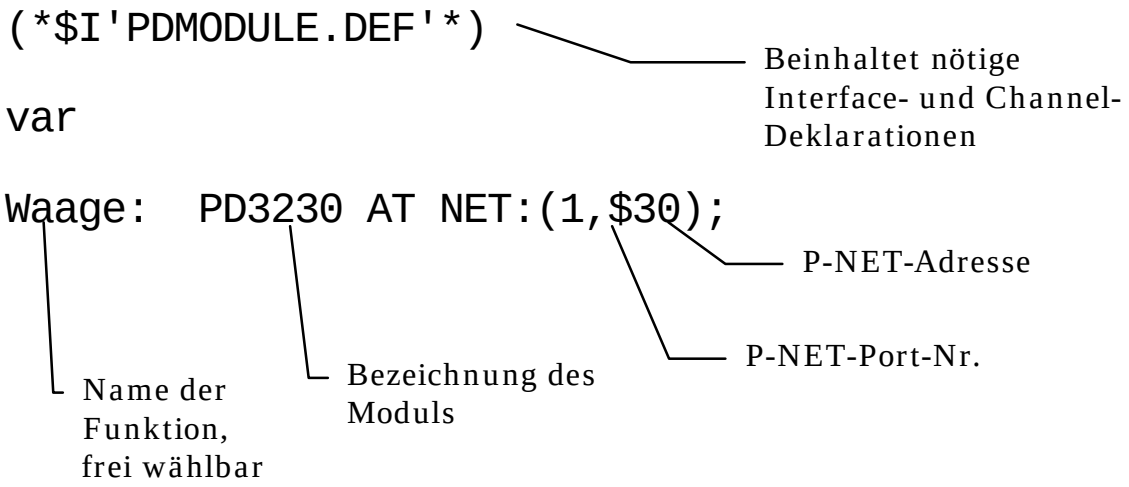
Wie der Name schon besagt, hat eine Variable vom Typ BUFFER die Funktion eines Zwischenspeichers, n den nach dem FIFO-Prinzip pro Operation jeweils ein Element gelesen bzw. geschrieben werden kann. Das Auslesen eines leeren BUFFERs und das schreiben in eine vollen BUFFER hat eine Errormeldung zufolge.

Der Datentyp CHANNEL wurde bereits ausführlichst 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 auch 16 *SWNo.*s zugeordnet sind [PP92].

XII. 4. Variablen im P-NET

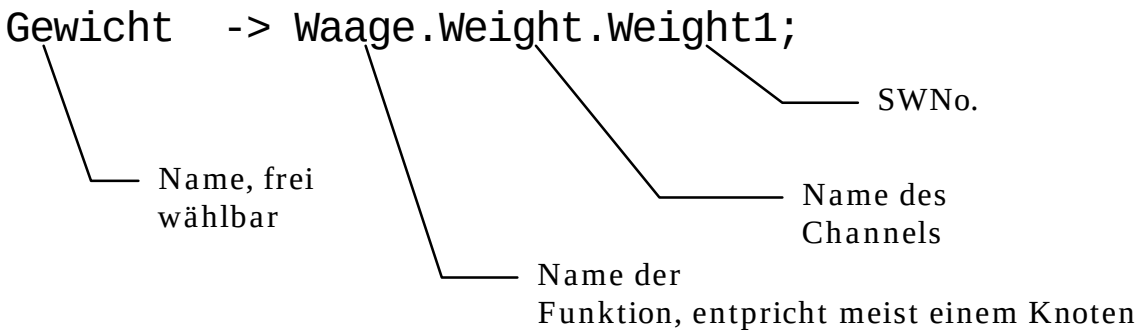
Um auf Variablen in einem beliebigen P-NET Knoten zugreifen zu können, muß deren physikalischer Ort deklariert werden. Die dazu nötigen Bestimmungsstücke sind ein zuvor definiertes *Interface Modul*, die P-NET-Adresse und die Nummer des P-NET-Ports:



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:



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]. **Gewicht** ist also nur ein Synonym für den Zugriff auf eine Variable in dem bereits zuvor deklarierten Knoten **Waage.**

XIII.VIGO

XIV. Literaturverzeichnis

Zeichen	Autor	Titel
[FBu92]	Borst, Walter	Der Feldbus Franzis Verlag, München 1992
[EN50170]	CENELEC	EN 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 types, Silkeborg, Denmark 1992
[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