POSITIONIER- UND BAHNSTEUERUNG MCU-3T
Programmier- und Referenz-Handbuch / PHB

Stand: 24.02.1997, Disk V2-00 97/09
Board-Revision: A

1 Einführung

Dieses Handbuch enthält alle notwendigen Angaben zur Programmierung der MCU-3T. Bevor die verschiedenen Programmiermethoden und Betriebsarten vorgestellt werden können, ist es notwendig verschiedene Funktionen der *rw_TOS*-Betriebssystem-Software zu beschreiben.
Anmerkung: Weitere Informationen zu *rw_TOS* sind im [BHB / Kapitel 4.1] enthalten.

2 Interne Details der *rw_TOS*-Betriebssystem-Software

Wie schon im Bedienungs-Handbuch erwähnt, beruht die Leistungsfähigkeit der MCU-3T unter anderem auf der Betriebssystemsoftware *rw_TOS*. In den nachfolgenden Kapiteln werden die in *rw_TOS* implementierten Funktionen wie z.B. die Profilgenerierung oder Endschalterbehandlung beschrieben.

2.1 Der MCU-3T Lageregler

Die Grundbetriebsart der MCU-3T ist die Lageregelung. In dieser Betriebsart versucht die MCU-3T die Motorposition auf der Sollposition festzuhalten. Der Regelkreis besteht gewöhnlich aus den Komponenten Digitaler Regler - Digital/Analog Wandlung - Leistungsteil - Motor - Enkoder - Impulserfassung. Der Enkoder ist in den meisten Fällen direkt am Motor angebaut, d.h. starr mit der Motorachse verbunden. Falls dies nicht der Fall ist, sind die Übertragungsglieder zwischen Motorachse und Enkoderachse zusätzlich im Regelkreis enthalten. An der Motorachse ist weiterhin die Last angeschlossen. Das Verhalten der Regelung wird durch alle im Regelkreis enthaltenen Glieder und die Last bestimmt. Bei einem gegebenen System läßt sich das Regelverhalten nur durch die des digitalen Filters beeinflussen. Hierbei müssen alle möglichen Betriebsfälle, z.B. Änderungen der Last berücksichtigt werden.

2.1.1 Regelkreis geöffnet / geschlossen

Nach dem Einschalten ist der Regelkreis zunächst geöffnet. Auf den Stellgrößenausgang (Motor-Command-Port) wird der Wert 0 ausgegeben. Durch Ausgabe eines Wertes kann die angeschlossene Achse ungeregelt verfahren werden. Mit dem PCAP-Befehl *cl()* (close loop) wird der Regelkreis geschlossen. Hierbei wird die aktuelle Position als Sollposition übernommen, um ein unbeabsichtigtes Verfahren der Motorachse zu verhindern. Nur nach Aktivierung der Lageregelung können Verfahrprofile durchgeführt werden. Dies gilt auch für Schritt-Motoren.

2.1.2 PIDF-Filter

Das digitale Filter hat die Struktur eines realen PIDF-Filters. Mit diesem Reglertyp lassen sich nahezu alle in der Praxis auftretenden Regelstrecken stabil einstellen.

2.1.2.1 Die Filterparameter K_D, K_I, K_P

Die Einstellung erfolgt über die Filterparameter K_D, K_I und K_P. Die Bedeutung dieser Parameter lassen sich sehr einfach auf die in der Literatur gängigen Parameter Proportionalverstärkung K_P, Vorhaltezeit T_V (Differenzierzeit) und Nachstellzeit T_N (Integrationszeitkonstante) zurückführen.

- K_P - Proportionalverstärkung
- K_I - Integrierbeiwert
- K_D - Differenzierbeiwert
- T_V - Vorhaltezeit
- T_N - Nachstellzeit

$K_I = K_P / T_N$
$K_D = K_P * T_V$

Falls ein Regler mit anderer Struktur realisiert werden soll, so lassen sich die einzelnen Anteile einfach durch Nullsetzen disaktivieren.

2.1.2.2 Zusätzliches Phasenglied

Das gegebene digitale PIDF-Filter ist standardmäßig in Kette geschaltet mit einem Verzögerungsglied erster Ordnung mit der Zeitkonstante $T_{A/2}$ (halbe Abtastzeit). Daher kommt die Bezeichnung reales PIDF-Filter. Mit dem Filterparameter K_{PL} kann nun diese Verzögerungszeit noch verkleinert werden. Dadurch ist eine härtere Einstellung des Regler möglich. Der Parameter K_{PL} darf prinzipiell Werte zwischen 0 und 1 annehmen. In der Praxis ist jedoch ein Wert größer als ca. 0.95 nicht mehr sinnvoll.
Der Zusammenhang zwischen K_{PL} und der Verzögerungszeit läßt sich einfach darstellen:

K_{PL} - Filterparameter
T_{VERZ} - reale Verzögerungszeit des PIDF-Filters
T_A - Abtastzeit

$$T_{VERZ} = (1 - K_{PL}) * T_A / 2$$

2.1.2.3 Abtastzeit

Im obigen Abschnitt wurde die Abtastzeit T_A verwendet. Diese ist eine charakteristische Größe für den digitalen Regler. Die Abtastzeit ist die Zeit, nach der jeweils Soll- und Istwert abgetastet werden und über den Regelalgorithmus die Stellgröße berechnet wird. Wenn die Abtastzeit klein ist gegenüber den vorkommenden Streckenzeitkonstanten kann die Dimensionierung des Reglers wie bei einem kontinuierlichen Regler erfolgen. Dadurch sind zur Einstellung keine speziellen Kenntnisse der digitalen Regelungstechnik erforderlich.
Anmerkung: In der MCU-3T-Standardversion ist die Abtastzeit auf **1,28ms** eingestellt.

2.2 Der MCU-3T Profilgenerator

Beim Verfahren mit den einzelnen Achsen werden die angegebenen Wege mit einem Trapez-Drehzahl-Profil angefahren. Für ein derartiges Trapez-Drehzahl-Profil sind die Größen Anfangsgeschwindigkeit, Anfangsposition, Beschleunigung, Maximalgeschwindigkeit, Zielposition und Zielgeschwindigkeit maßgebend. Die vorliegende Profilgenerierung erzeugt für den Lageregler [Kapitel 2.1] abtastsynchron die entsprechenden Sollwerte, damit von der augenblicklichen Position ausgehend von der Momentangeschwindigkeit zur Maximalgeschwindigkeit beschleunigt wird. Die Anfangsgeschwindigkeit und Anfangsposition sind Momentanwerte und werden nicht als Parameter für ein Bewegungsprofil angegeben. Bevor die Zielposition erreicht wird, bremst der Profilgenerator rechtzeitig mit der vorgegebenen Beschleunigung ab, damit im vorgegebenen Zielpunkt die Zielgeschwindigkeit erreicht wird.

2.2.1 Profilgenerierung für JOG-Befehle

Beim Abfahren eines Trapez-Drehzahlprofils mit JOG-Verfahrbefehlen (Einzelachsbewegungen) sind nun einige Sonderfälle möglich:

- Die Anfangsgeschwindigkeit ist negativ gegenüber der Verfahrrichtung. Die Achse verfährt also zunächst in die falsche Richtung, bremst jedoch ab, kehrt um und beschleunigt nun in die richtige Richtung.
- Die Endgeschwindigkeit ist negativ gegenüber der Verfahrrichtung. Die Achse fährt zunächst über den Zielpunkt hinaus, bremst ab, kehrt die Richtung um und hat beim nochmaligen Erreichen des Zielpunktes die Zielgeschwindigkeit.
- Die Anfangsgeschwindigkeit ist gleich der Maximalgeschwindigkeit.
- Die Anfangsgeschwindigkeit ist höher als die Maximalgeschwindigkeit. In diesem Fall wird automatisch auf die Maximalgeschwindigkeit abgebremst.
- Die Endgeschwindigkeit ist gleich der Maximalgeschwindigkeit.
- Die Maximalgeschwindigkeit wird nicht erreicht, da vorher abgebremst werden muß um die Zielgeschwindigkeit bis zur Zielposition zu erreichen. In diesem Fall wird ein Dreieck-Drehzahl-Profil abgefahren.

Alle diese Fälle werden richtig behandelt, falls der abzufahrende Weg jeweils ausreicht. Weiterhin muß stets eine positive Maximalgeschwindigkeit und Beschleunigung angegeben werden und die Endgeschwindigkeit muß kleiner oder gleich groß sein wie die Maximalgeschwindigkeit. Bei Angabe einer negativen Beschleunigung wird diese für die Bremsrampe des Profils herangezogen. Bei JOG-Verfahrbefehlen ist es also möglich Beschleunigungs- und Bremsrampen mit unterschiedlicher Steilheit zu programmieren.

Bei nachfolgend aufgeführten Fällen treten Geschwindigkeitssprünge, also unerwünscht hohe Beschleunigungen auf. Falls diese vom System nicht realisiert werden können, tritt ein Schleppfehler auf, der jedoch i.a. nach einer begrenzten Zeit wieder ausgeregelt wird. Beim Einsatz von Schrittmotoren können diese Fälle i.a. nicht zugelassen werden.

- Der angegebene Verfahrweg reicht nicht zum Abbremsen aus.
- Die Zielgeschwindigkeit ist höher als die Maximalgeschwindigkeit. In diesem Fall wird am Profilende die Verfahrgeschwindigkeit auf die Zielgeschwindigkeit gesetzt.

2.2.2 Profilgenerierung für MOVE-Befehle

Beim Abfahren eines Trapez-Drehzahlprofils mit MOVE-Verfahrbefehlen (Mehrachsbewegungen mit Interpolation) mit einer oder mehreren Achsen sind folgende Sonderfälle möglich:

- Die Endgeschwindigkeit ist negativ gegenüber der Verfahrrichtung. Das System fährt zunächst über den Zielpunkt hinaus, bremst ab, kehrt die Richtung um und hat beim nochmaligen Erreichen des Zielpunktes die Zielgeschwindigkeit.
- Die Anfangsgeschwindigkeit ist gleich der Maximalgeschwindigkeit.

- Die Anfangsgeschwindigkeit ist höher als die Maximalgeschwindigkeit. Bei direkten MOVE-Befehlen wird in diesem Fall automatisch auf die Maximalgeschwindigkeit abgebremst. Bei Spooler-Befehlen wird die Anfangsgeschwindigkeit auf die Maximalgeschwindigkeit gesetzt. Dies entspricht einem Geschwindigkeitssprung.
- Die Endgeschwindigkeit ist gleich der Maximalgeschwindigkeit.
- Die Maximalgeschwindigkeit wird nicht erreicht, da vorher abgebremst werden muß um die Zielgeschwindigkeit bis zur Zielposition zu erreichen. In diesem Fall wird ein Dreieck-Drehzahl-Profil abgefahren.

Alle diese Fälle werden richtig behandelt, falls der abzufahrende Weg jeweils ausreicht. Weiterhin muß stets eine positive Maximalgeschwindigkeit und Beschleunigung angegeben werden und die Endgeschwindigkeit muß kleiner oder gleich groß sein wie die Maximalgeschwindigkeit. Bei Angabe einer negativen Beschleunigung oder Maximalgeschwindigkeit wird das Profil verworfen. Bei MOVE-Verfahrbefehlen ist es nicht möglich Beschleunigungs- und Bremsrampen mit unterschiedlicher Steilheit in einem Verfahrbefehl zu programmieren. Falls dies erforderlich ist können jedoch mehrere MOVE-Verfahrbefehle hintereinander programmiert werden.

Bei nachfolgend aufgeführten Fällen treten Geschwindigkeitssprünge, also unerwünscht hohe Beschleunigungen auf. Falls diese vom System nicht realisiert werden können, tritt ein Schleppfehler auf, der jedoch i.a. nach einer begrenzten Zeit wieder ausgeregelt wird. Im Zusammenhang mit Schrittmotoren dürfen diese Fälle i.a. nicht zugelassen werden.

- Der angegebene Verfahrweg reicht nicht zum Beschleunigen auf die Zielgeschwindigkeit aus. In diesem Fall wird die Zielgeschwindigkeit auf einen Wert gesetzt der im angegebenen Profil erreicht werden kann. In diesem Fall tritt jedoch kein Geschwindigkeitssprung auf.
- Der angegebene Verfahrweg reicht nicht zum Abbremsen aus. In diesem Fall wird die Profil-Anfangsgeschwindigkeit auf einen Wert gesetzt der es erlaubt im angegebenen Profil auf die Endgeschwindigkeit abzubremsen.
- Die Zielgeschwindigkeit ist höher als die Maximalgeschwindigkeit. In diesem Fall wird am Profilende die Verfahrgeschwindigkeit auf die Zielgeschwindigkeit gesetzt.
- Die Richtung des Verfahrprofils wird geändert. In diesem Fall wird der Betrag des Geschwindigkeitsvektors aus der bisherigen Richtung in die nun abzufahrende Richtung gelegt. In diesem Fall treten bei den beteiligten Achsen mehr oder weniger große Geschwindigkeitssprünge auf. Besondere Vorsicht ist hier beim Einsatz von Schrittmotorsystemen geboten.

Diese Art der Profilgenerierung wird nicht nur beim Abfahren von linearen Bewegungskommandos ausgeführt. Auch beim Abfahren von Kreisbewegungen mit zwei Achsen wird die Bahngeschwindigkeit nach diesem Schema generiert.

2.2.3 Beschleunigung

Falls eine Beschleunigung kleiner als Null angegeben wird, wird bei MOVE-Befehlen der Datensatz verworfen. Bei JOG-Befehlen gibt eine negative Beschleunigung die Steilheit der Bremsrampe vor. Defaultmäßig hat die Bremsrampe die gleiche Steilheit wie die Beschleunigungsrampe. Die Einheiten für die Beschleunigung können im Hilfsprogramm *mcfg.exe* achsspezifisch festgelegt werden. Für die Interpolationsbefehle (MOVE-Befehle) gibt es verschiedene Möglichkeiten zur Auswahl der Einheiten. Die Wertangabe für die Beschleunigung erfolgt als Gleitpunktzahl, der Wertebereich ist also nahezu unbegrenzt. Bei Angabe einer höheren Beschleunigung als vom System realisiert werden kann, entsteht während der Beschleunigungsphase ein vergrößerter Schleppfehler.

2.2.4 Maximal-Geschwindigkeit

Die Maximalgeschwindigkeit muß immer größer als Null angegeben werden, andernfalls wird der Datensatz verworfen (MOVE) bzw. wird ein Endlosprofil in die falsche Richtung abgefahren (JOG). Die Einheiten für die Maximalgeschwindigkeit können im Hilfsprogramm *mcfg.exe* achsspezifisch festgelegt werden. Für die Interpolationsbefehle gibt es verschiedene Möglichkeiten zur Auswahl der Einheiten. Die Wertangabe für die

Maximalgeschwindigkeit erfolgt als Gleitpunktzahl, der Wertebereich ist also nahezu unbegrenzt. Bei Angabe einer höheren Geschwindigkeit als vom System realisiert werden kann, entsteht während des Verfahrens ein vergrößerter Schleppfehler. Falls die angegebene Maximalgeschwindigkeit kleiner ist als die Anfangsgeschwindigkeit, gelten die oben genannten Bedingungen abhängig von der Befehlsart.

2.2.5 Zielgeschwindigkeit

Die Zielgeschwindigkeit kann positiv, negativ oder natürlich mit 0 angegeben werden. Die Richtung der Zielgeschwindigkeit bezieht sich immer auf die Verfahrrichtung. Beim Verfahren in negativer Richtung und positiver Zielgeschwindigkeit wird also in negativer Richtung weitergefahren. Die Zielgeschwindigkeit hat dieselbe Einheit wie die Maximalgeschwindigkeit. Die Wertangabe erfolgt als Gleitpunktzahl, der Wertebereich ist also nahezu unbegrenzt. Bei Angabe einer höheren Geschwindigkeit als vom System realisiert werden kann, entsteht während des Verfahrens ein vergrößerter Schleppfehler. Falls die angegebene Zielgeschwindigkeit größer ist als die Maximalgeschwindigkeit, wird das Verfahrprofil mit einem Geschwindigkeitssprung beendet. Die Momentangeschwindigkeit wird in diesem Fall am Profilende auf die Zielgeschwindigkeit gesetzt.

2.2.6 Geschwindigkeitskorrektur

In bestimmten Fällen ist es erwünscht, die Achs- oder Bahngeschwindigkeit während der Ausführung eines Trapez- Drehzahlprofils zu verändern. Ein typisches Beispiel hierfür ist die manuelle Geschwindigkeitskorrektur (Override). Hierzu stehen dem Anwender verschiedene SAP- und PCAP-Befehle zur Verfügung.
Der Geschwindigkeitskorrekturfaktor, dessen Defaultwert = 1.0 ist, wirkt sich gleichermaßen auf Geschwindigkeiten und Beschleunigungen aus.

2.2.7 Zielposition / Verfahrweg

Die Angabe des Zieles kann als Relativ- oder Absolutwert erfolgen. Bei Angabe eines Relativwertes wird um den angegebenen Weg verfahren, d.h. es wird ein Verfahrweg programmiert. Bei Angabe eines Absolutwertes wird auf die angegebene Position verfahren, d.h. es wird eine Zielposition programmiert. Der Bezugspunkt für absolute Zielpositionen ist der Maschinennullpunkt.

2.2.8 Betriebsarten zur Befehlsabarbeitung

Verfahrbefehle und sonstige Befehle können in zwei unterschiedlichen Betriebsarten, dem sogenannten Direkten-Modus und dem Spool-Modus ausgeführt werden. Die zur Ausführung kommende Betriebsart wird durch die Syntax des jeweiligen Befehls automatisch festgelegt.

Anmerkung: Die Kommandokürzel der *spool*-Befehle sind im Gegensatz zu den Direkt-Befehlen durch das Zeichen 's' als ersten Buchstaben im Befehlswort gekennzeichnet. Es stehen für beide Programmiermethoden also SAP- und PCAP-Programmierung identische *spool*-Befehle zur Verfügung.

2.2.8.1 Direkter Modus

Der direkte Modus wird durch den Aufruf von speziellen *move*- und *jog*-Befehlen automatisch aktiviert.
Beim Programmieren eines Verfahrbefehls im direkten Modus, wird mit der Ausführung des angegebenen Befehls nach einer systembedingten Verzögerungszeit (ca. 2 - 3 Abtastintervalle) begonnen. Ein bereits ablaufendes Profil wird nicht bis zum Ende abgefahren, die Momentanwerte von Geschwindigkeit und Position werden als Anfangswerte für den aktuellen Verfahrbefehl übernommen. Falls die Profildaten und die Anfangswerte konsistent sind, d.h. den obigen Anforderungen entsprechen, wird ein gerade ablaufendes Profil nahtlos fortgesetzt. So ist es z.B. möglich den Zielpunkt eines ablaufenden Profils zu verändern, die Geschwindigkeit nochmals zu erhöhen, oder die Beschleunigung der Bremsrampe nachträglich zu verändern, ja sogar die Beschleunigung während einer Beschleunigungsrampe zu verändern. Falls verschiedene Profile nacheinander abgefahren werden sollen, muß jeweils das Profilende abgewartet werden.

Anmerkung: Eventuell im Spooler vorliegende Daten werden bei der Ausführung von Befehlen im direkten Modus verworfen.

2.2.8.2 Spool-Modus

Im Spool-Modus kann eine große Anzahl von Verfahr- oder sonstigen Befehlen in eine Warteschlange (Spooler) eingetragen werden. Die Abarbeitung der Befehle, welche im Spooler eingetragen sind, wird beispielsweise durch den PCAP-Befehl *ssms()* gestartet. Während der Abarbeitung ist es möglich, den Spooler mit weiteren Befehlen zu beschreiben. Die Abarbeitung der Befehle aus dem Spooler erfolgt nacheinander ohne Verzögerung. Der freie Spoolerbereich wird mit jedem Befehlseintrag kleiner, beim Beginn jeder Befehlsausführung wieder größer. Wenn alle Befehle im Spooler abgearbeitet sind, wird automatisch wieder in den Direkt-Modus geschaltet, d.h. nach erneutem Eintragen von *spool*-Befehlen muß deren Abarbeitung erneut gestartet werden.

Anmerkung: Damit die Spoolereinträge richtig abgearbeitet werden können, müssen folgende Voraussetzungen gelten:

- Alle Achsen, für die Kommandos gespoolt werden sollen, müssen beim ersten Spoolereintrag in der Lageregelung sein.
- Die Geschwindigkeit dieser Achsen muß vor der Ausführung des ersten *spool*-Befehls Null sein, deshalb darf der Befehl Start Spooled Motions Synchronised *ssms()* nur dann ausgeführt werden, wenn alle beteiligten Achsen stehen.

2.3 Interpolation mit der MCU-3T

Das Verfahren einzelner Achsen mit der MCU-3T erfolgt mit den *jog*-Befehlen. Um mehrere Achsen interpoliert zu verfahren, stehen die *move*-Befehle zur Verfügung. Mit der MCU-3T ist es möglich Kreis-, Linear- und Helixinterpolationen durchzuführen. Mehrere Interpolationsprofile können gleichzeitig, mit beliebiger Anfangs- und Endgeschwindigkeit abgearbeitet werden. Alle Interpolationsberechnungen erfolgen abtastsynchron (1.28 ms).

2.3.1 Linearinterpolation

Bei der Linearinterpolation wird eine beliebige Anzahl von Achsen auf einer Raumgeraden (n-dimensional) vom Startpunkt zum Zielpunkt (Absolutpositionierung) oder um einen Raumvektor (Relativpositionierung) verfahren. Parameter bei der Linearinterpolation sind die teilnehmenden Achsen, der Verfahrweg bzw. die Zielposition, die Bahnbeschleunigung, die maximale Bahngeschwindigkeit und die Bahn-Zielgeschwindigkeit. Beim Interpolieren mit einer Anfangsgeschwindigkeit sollte gewährleistet sein, daß die Richtungsvektoren der Anfangsgeschwindigkeit und des Interpolationsprofiles übereinstimmen. Ansonsten wird die Richtung des Geschwindigkeitsvektors geändert. Dies kann zu Geschwindigkeitssprüngen bei den teilnehmenden Achsen führen. Falls die Interpolationsrichtung von einem Profil zum nächsten geändert werden muß, sollte ein Zwischenstopp erfolgen. Bei Richtungsumkehr ist die Beendigung des ersten Profils mit negativer Zielgeschwindigkeit möglich.

2.3.1.1 Formale Linearinterpolation

Beim Abfahren von Konturen besteht die Möglichkeit, daß eine Achse die momentane Motorposition beibehalten muß, während die anderen Achsen interpoliert verfahren werden. Diese stillstehende Achse kann jedoch formal an dieser Interpolation der anderen Achsen teilnehmen und bleibt somit auf diese synchronisiert. Diese formale Interpolation ist besonders wichtig in der Spool-Betriebsart und wird automatisch für alle Achsen selektiert, bei denen ein Verfahrweg von 0 programmiert wird.

2.3.2 Kreisinterpolation

Die Kreisinterpolation wird mit zwei beliebigen Achsen durchgeführt. Parameter bei der Kreisinterpolation sind die teilnehmenden Achsen, die Koordinaten des Kreismittelpunktes, der Verfahrwinkel (positiv oder negativ), die Bahnbeschleunigung, die Bahn-Maximalgeschwindigkeit und die Bahn-Zielgeschwindigkeit. Die Koordinaten des Kreismittelpunktes können in Absolutkoordinaten oder in Relativkoordinaten angegeben werden.
Beim Interpolieren eines Kreises mit einer Anfangsgeschwindigkeit muß darauf geachtet werden, daß die Anfangsgeschwindigkeit die gewünschte Richtung hat, d.h. die Richtung der Tangente im Kreisstartpunkt. Ansonsten wird die Richtung des Geschwindigkeitsvektors geändert. Dies kann zu Geschwindigkeitssprüngen bei den teilnehmenden Achsen führen. Falls die Interpolationsrichtung von einem Profil zum nächsten geändert werden muß, sollte ein Zwischenstopp erfolgen. Bei Richtungsumkehr ist die Beendigung des ersten Profils mit negativer Zielgeschwindigkeit möglich.

2.3.3 Helixinterpolation

Die Helix-Interpolation wird für zwei beliebige Achsen als Zirkular-Interpolation und mit einer dritten beliebigen Achse als Linear-Interpolation ausgeführt.

2.3.4 Synchrone und asynchrone Interpolationen

Die MCU-3T gestattet unter anderem auch das Abarbeiten mehrerer verschiedener Interpolationen zum gleichen Zeitpunkt. So ist es beispielsweise möglich, zwei Zirkular-Interpolationen mit jeweils zwei verschiedenen Achskanälen auszuführen. Die jeweiligen Interpolationen können dabei synchron als auch asynchron zueinander ausgeführt werden. Die synchrone Betriebsart wird dabei besonders gut durch den

Spoolermechanismus unterstützt. Selbstverständlich kann auch neben einer Interpolation jede beliebige andere Achse, die nicht im Interpolationszusammenhang verwendet wird, unabhängig verfahren werden.

2.4 Die MCU-3T Endschalterbehandlung

Die MCU-3T bietet eine große Palette von Möglichkeiten der Endschalterbehandlung bzw. Verfahrbereichsbegrenzung. So ist es möglich einen oder mehrere beliebige Digitaleingänge als Hardwareendschalter Links oder Rechts zu konfigurieren. Bei der Konfiguration wird dem Endschaltereingang zusätzlich eine Funktion TOM, SMA oder SMD zugeordnet. Weiterhin kann für jeden Achskanal zusätzlich ein Softwareendschalter links und rechts definiert werden. Die Endschalterpositionen sind frei wählbar. Auch hier kann zwischen den Funktionen TOM, SMA und SMD ausgewählt werden. Der Zustand der Endschalter kann dem Statusflag *axst* entnommen werden.
Ein bestimmter Endschalterzustand wird beim Unterschreiten der Sollposition unter die Endschalterposition gelöscht.
Anmerkung: Alle Endschalterzustände werden beim Schließen des Regelkreises [Kapitel 4.4.4 - *cl()*] gelöscht.

2.4.1 Endschalterfunktion TOM (Turn-Off-Motor)

Bei dieser Endschalterfunktion wird der Motor in die Endschalterrichtung stromlos geschaltet, d.h. die Achse läuft beim Ansprechen des Endschalters ungeregelt aus und kann nicht weiter in den Endschalterbereich, lediglich gegen die Endschalterrichtung verfahren werden. Die Sollposition kann jedoch, z.B. durch ein gerade ablaufendes Profil weiterhin in den Endschalterbereich laufen. Beim Herausfahren aus dem Endschalterbereich sind unkontrollierte Geschwindigkeitssprünge möglich.

2.4.2 Endschalterfunktion SMA (Stop-Motor-Abruptly)

Bei dieser Endschalterfunktion wird die Sollposition beim Überschreiten der Endschalterposition festgehalten. Der Lageregler hält die Achse in dieser Position fest. Die vom Profilgenerator berechnete Sollposition wird jedoch intern richtig weitergeführt. Beim Ein- und Austritt der Sollposition aus dem Endschalterbereich sind unkontrollierte Geschwindigkeitssprünge möglich.

2.4.3 Endschalterfunktion SMD (Stop-Motor-Decelerate)

Bei dieser Endschalterfunktion wird die betreffende Achse mit der spezifizierten Stop Deceleration {*sdec*} auf Geschwindigkeit 0 abgebremst. Die Achse wird in den direkten Modus geschaltet und etwaige Spoolereinträge werden verworfen. Ein weiteres geregeltes Verfahren in den Endschalterbereich ist nicht mehr möglich. Die Achse kann mit allen Verfahrbefehlen aus dem Endschalterbereich herausgefahren werden. Dies ist die Universal-Endschalterfunktion.

3 Die MCU-3T Programmiermethoden

Ein wichtiger Bestandteil der MCU-3T-Positionier- und Bahn-Steuerung ist das Echtzeit-Multi-Task-Betriebssytem *rw_TOS* (*T*ransputer *O*perating *S*ystem). Dieses ist in der Datei *rwtos.btl* abgelegt und wird einmalig pro PC-Systemstart mit dem Bootprogramm *mcbt.exe* innerhalb weniger Sekunden in den Arbeitsspeicher der TPU-6002-Transputer-Karte geladen. *rw_TOS* nutzt die im Transputer implementierten Hardware-Eigenschaften wie Task-Scheduler und Multiprozessor- Parallelverarbeitung mit anderen Transputer-Systemen über die sogenannten Transputer-Link-Kanäle.
Die Betriebssystemsoftware *rw_TOS* ist in verschiedene Tasks aufgeteilt, welche im wesentlichen zwei unterschiedliche Arten der Anwenderprogrammierung ermöglicht.

Anmerkung: *rwtos.btl* und *mcbt.exe* sind Bestandteil der MCU-3T TOOLSET Software. Weitere Informationen dazu sind im Bedienungshandbuch enthalten.

3.1 PC-Applikations-Programmierung (PCAP-Programming, auch Direkt-Programmierung)

Die MCU-3T PC-Applikations-Programmierung (PCAP) wird durch ein auf dem PC ablaufendes Anwenderprogramm erledigt. Die Programmerstellung erfolgt mit Hilfe einer höheren Programmiersprache wie *Turbo C*, *Microsoft C* oder *Turbo Pascal*. Mit Hilfe der im Lieferumfang enthaltenen Funktionenbibliotheken für diese Programmiersprachen kann der Anwender auf einen leistungsfähigen Befehlsvorrat zurückgreifen, welcher eine schnelle und effektive Programmerstellung gestattet. Zum Befehlsumfang gehören beispielsweise Verfahrbefehle mit und ohne Interpolation, Ein- Ausgabe-Befehle, Abfrage-Befehle, *spool*-Befehle usw.
Das typische Anwenderprogramm sendet einen oder mehrere dieser Befehle an die MCU-3T und wartet im Anschluß auf die Abarbeitung dieser Aufträge. Nachdem die entsprechenden Befehle durch die *PC-Task* im *rw_TOS*-Betriebssystem selbsttätig ausgeführt wurden, können neue Befehlsaufträge an die *PC-Task* übermittelt werden. Die Zeit zwischen Befehlsauftrag und Befehlsabarbeitung kann das Anwenderprogramm nutzen, um weitere applikationsspezifische Aufgaben zu erledigen.
Da die Programmierung durch den direkten Zugriff eines PC-Anwenderprogramms erfolgt, wird diese Programmiermethode auch PC-Direkt-Programmierung genannt.
Anmerkung: In den nachfolgenden Kapiteln wird gelegentlich der Begriff PCAP-Befehl verwendet. Dieser Befehlstyp hat die soeben beschriebene Programmiermethode zur Grundlage.

3.2 Standalone-Applikations-Programmierung (SAP-Programming)

Im Gegensatz zur PC-Applikations-Programmierung gestattet die Stand-Alone-Applikations-Programmiermethode eine Programmabarbeitung gänzlich ohne Hilfe eines PC-Anwenderprogrammes. Ein in der Programmiersprache *rw_SymPas* erstelltes Anwenderprogramm wird mit Hilfe des in der Entwicklungsumgebung *mcfg.exe* integrierten Compilers *NCC* übersetzt und erzeugt ein für die MCU-3T verständliches Betriebsprogramm. Dieses Betriebsprogramm kann auf die MCU-3T geladen werden und wird mit Hilfe der *CNC-Task* (CNC = Computerized Numerical Control) in *rw_TOS* selbsttätig ausgeführt. Sofern eine Synchronisation zwischen einem PC-Anwenderprogramm und dem MCU-3T-Standalone-Programm notwendig ist, kann diese mit Hilfe vordefinierter System-Variablen, auf welche beide System-Partner (PC und MCU-3T) zugreifen können, durchgeführt werden.
Anmerkung: In den nachfolgenden Kapiteln wird des öfteren der Begriff SAP-Befehl verwendet. Dieser Befehlstyp hat die soeben beschriebene Programmiermethode zur Grundlage.

3.2.1 SAP-Multitasking

Die Betriebssystemsoftware *rw_TOS* kann bis zu 4 SAP-Programme gleichzeitig abarbeiten. Alle gleichzeitig ausgeführten Tasks haben die gleiche Priorität. Die verschiedenen Task werden über Nummern angesprochen. Die kleinste Tasknummer hat den Wert 0 und die größte somit den Wert 3.
Die Multitasking-Programmierung ermöglicht es, eine komplexe Aufgabe in kleine überschaubare Teilaufgaben zu zerlegen. Beispielsweise könnte eine Task zur Referenzfahrt, eine andere zur Überwachung des Antriebes mit den entsprechenden EVENT-Handlern und wieder eine andere zur reinen SPS-Steuerung mit entsprechenden Zugriffen auf Digital-I/O bzw. PC-Kommunikation mit vordefinierten Registern verwendet werden.
Die verschiedenen SAP-Programme können sich mit Hilfe verschiedener Task-Steuerbefehle selbsttätig stoppen, starten oder auch fortsetzen.
Die Synchronisation der CNC-Tasks untereinander und die Synchronisation auf ein evt. parallel ablaufendes PCAP-Anwenderprogramms bzw. der Datenaustausch zwischen diesen kann durch vordefinierte Register, den sogenannten COMMON-Variablen, ausgeführt werden. Hierzu stehen für alle CNC-Tasks 100 gemeinsame Ganzzahl- und 100 gemeinsame Gleitpunkt-Register zur Verfügung.
Jeder CNC-Task steht zusätzlich ein lokaler Speicherbereich mit einer Größe von 1000 Bytes (COMMON BUFFER) zur Verfügung, auf den sowohl der PC als auch die entsprechende CNC-Task sowohl lesend als auch schreibend zugreifen kann. Dieser kann z.B. zum Aufbau eines benutzerspezifischen Befehlssatzes verwendet werden.

4 PC-Applikations-Programmierung

4.1 Einführung

In der MCU-3T TOOLSET Software sind Bibliotheksfunktionen für die Programmiersprachen *Turbo Pascal* (ab Version 5.0), *Turbo C* und *Microsoft C* enthalten. Die einzelnen Funktionen werden über den TSR-Treiber *mcutsr.exe* ausgeführt. Die Bedeutung der einzelnen Funktionsparameter und deren Datentypen ist für beide Programmiersprachen identisch.
Das Einbinden der Funktionenbibliotheken in die jeweilige Programmiersprache wird nachfolgend erläutert:

Turbo Pascal: Der Name der Funktionenbibliothek ist *mcutsr.pas*. Mit Hilfe dieser Funktionen wird die Verbindung zwischen PC-Applikations-Programm und dem TSR-Treiber *mcutsr.exe* hergestellt. Diese Datei ist als *unit* deklariert und wird mit Hilfe der *uses*-Anweisung zum Anwenderprogramm gebunden.
Achtung: Verschiedene Systemparameter besitzen den Datentyp *doubl*e. Dies bedeutet, daß das Anwenderprogramm mit der Option *{$N+}* compiliert werden muß!

C (*Turbo C, Microsoft C):* Der Name der Funktionenbibliothek ist *mcutsr*.c. Mit Hilfe dieser Funktionen wird die Verbindung zwischen PC-Applikations-Programm und dem TSR-Treiber *mcutsr.exe* hergestellt. Diese Datei kann mit Hilfe der #*include*-Anweisung in das Anwenderprogramm miteingebunden werden. Ebenso könnte auch eine Objektdatei aus diesem File generiert werden, welche zum Anwenderprogramm dazugebunden wird. In diesem Fall sollte im Anwenderprogramm die Header-Datei *mcutsr*.*h*, welche die Prototypdefinition der einzelnen Funktionen enthält, eingebunden werden.

4.2 Beispielprogramme zur Anwendung der Funktionenbibliotheken

Die in der MCU-3T TOOLSET Software enthaltenen Beispielprogramme zeigen die einfache Anwendung nachfolgend beschriebener Funktionen. Die Quelltexte der Beispielprogramme sind durch Kommentare selbsterklärend. Deshalb wird an dieser Stelle auf eine detaillierte Programmbeschreibung dieser Beispiele verzichtet. Die einzelnen Beispielprogramme für die beiden Programmiersprachen sind in den angegebenen Unterverzeichnissen zu finden und haben folgende Namen:

Turbo Pascal:	*ld.pas, mcutsr.pas*, *move.pas* usw. Verzeichnis: TP
C:	*ld.c, mcutsr.c*, *mcutsr.h*, *move.c* usw. Verzeichnis: C

4.3 Definitionen, Strukturen und Records

Bevor die einzelnen Funktionen erklärt werden, erfolgt die Beschreibung einiger Definitionen, Strukturen bzw. Records die zum Teil als Parameter für diese Funktionen benötigt werden. Die Deklaration der benötigten Struktur- bzw. Record-Datenfelder erfolgt immer im PC-Anwenderprogramm. Dies hat den Vorteil, daß der TSR-Treiber nur wenig PC-Arbeitsspeicher belegt.
Alle nachfolgend abgedruckten Struktur- bzw. Record-Typen und Systemkonstanten sind in den oben genannten Programmiersprachen in den Dateien *mcutsr.h* bzw. *mcutsr.pas* definiert.

Anmerkung: Die Systemkonstante *REALAXIS* (*mcutsr.h*) muß auf die tatsächlich im System vorhandene Achsanzahl gesetzt werden.

4.3.1 Definitionen

Tabelle 1: Systemkonstanten

Name	Typ	Funktion
Softint	integer	Dieser Parameter ist bei allen Funktionsaufrufen notwendig. Er gibt an, unter welcher Software-Interrupt-Nummer der TSR-Treiber mcutsr.exe aufgerufen wird. Der Defaultwert für softint ist 60 hex. Dieser Wert kann mit dem Konfigurationsprogramm mcfg.exe verändert werden.
MAXAXIS	integer	Maximale Anzahl der möglichen Achsen. Derzeit unterstützt die TOOLSET Software bis zu 18 Achsen.
LONGINT	long/longint	Datentyp long in der Programmiersprache C bzw. longint in der Programmiersprache Turbo-Pascal.

4.3.2 Strukturen und Records

Je nach Programmiersprache spricht man entweder von Strukturen *(C)* bzw. Records *(Pascal)*. Der Aufbau und die Funktionsweise dieser Datentypen ist für beide Programmiersprachen identisch. Zur Steigerung der Übersichtlichkeit sind alle Struktur- bzw. Record-Typen groß und deren Komponenten klein geschrieben.

4.3.2.1 Struktur- und Record-Typ AS

Tabelle 2: Struktur- und Record-Typ AS

Element	Typ	(Kurzwortbedeutung), Funktion
unoa	LONGINT	(used number of axis) Anzahl der anzuwählenden Achsen bei verschiedenen Funktionsaufrufen.
san	Feld mit MAXAXIS LONGINT	(selected axis number) Feld der anzuwählenden Achsen. Dieses Feld ist mit Index 0 beginnend je nach Anzahl der verwendeten Achsen zu initialisieren.

Anmerkung: Die Zählweise der Achskanäle beginnt bei dem Wert 0.

```
Beispiel: Anwahl der ersten und dritten Achse

as.unoa = 2;     // Anzahl der Achsen
as.san[0] = 0;   // erste Achse
as.san[1] = 2;   // dritte Achse
```

4.3.2.2 Struktur- bzw. Record-Typ TSRP

Um mit den einzelnen Achssystemen arbeiten zu können, muß für alle Achsen je ein Struktur- bzw. Record-Typ *TSRP* deklariert werden. Mit Hilfe der in TSRP enthaltenen Struktur- bzw. Record-Elemente erfolgt bei verschiedenen PCAP-Befehlen der Datenaustausch mit der MCU-3T. So können achsspezifische Systemgrößen wie Beschleunigungen, Geschwindigkeiten und Positionen mit Hilfe spezieller Lese- und Schreibbefehle abgefragt bzw. gesetzt werden.
Achtung: Die einzelnen Elemente der Struktur TSRP werden nicht automatisch initialisiert, d.h. der Anwender muß diese durch direktes Setzen bzw. vorheriges Lesen aktualisieren.

Anmerkung: Es muß darauf geachtet werden, daß bei der Verwendung von mehr als einem Achskanal die Strukturen bzw. Records *TSRP* im Speicher direkt hintereinander angeordnet werden, da der TSR-Treiber *mcutsr.exe* zum Teil mit Hilfe von Adreßberechnungen auf die verschiedenen Achsparameter zugreift. Die korrekte Anordnung im PC-Arbeitsspeicher wird erzwungen, indem TSRP als Feld-Variable deklariert wird. Die Größe des Feldes richtet sich dabei nach der Anzahl der im System vorhandenen Achsen.

Tabelle 3: Struktur- und Record-Typ TSRP (achsspezifische Parameter)

Element	Typ	(Kurzwortbedeutung), Funktion
an	LONGINT	(axis number)
kp	double	(PIDF filter parameter kp)
ki	double	(PIDF filter parameter ki)
kd	double	(PIDF filter parameter kd)
kpl	double	(PIDF filter parameter kpl)
kfca	double	(PIDF forward compensation acceleration)
kfcv	double	(PIDF forward compensation velocity)
jac	double	(jog acceleration)
jvl	double	(jog velocity)
jtvl	double	(jog target velocity)
jovr	double	(jog override)
hac	double	(home acceleration)
hvl	double	(home velocity)
rp	double	(real position)
dp	double	(desired position)
tp	double	(target position)
sll	double	(software limit left)
slr	double	(software limit right)
ipw	double	(in position window)
mpe	double	(maximum position error)
gf	double	(gear factor)
mcp	LONGINT	(motor command port)
axst	LONGINT	(axis status)
lsm	LONGINT	(left spool memory)
epc	LONGINT	(eeprom programming cycle)
digi	LONGINT	(digital inputs)
digo	LONGINT	(digital outputs)
ifs	LONGINT	(interface status)
scratch	Feld mit 4 mal LONGINT	(scratch field) Platzhalter zum nächsten TSRP-Satz

4.3.2.3 Struktur- und Record-Typ TRU (Trajectory Units)

Dieser Struktur- bzw. Recordtyp ist Parameter für den PCAP-Befehl *ctru()*.

Tabelle 4: Struktur- und Record-Typ TRU

Element	Typ	(Kurzwortbedeutung), Funktion
pu	LONGINT	(position unit) Positions-Einheit
tu	LONGINT	(time unit) Zeit-Einheit

4.3.2.4 Struktur- und Record-Typ LMP (Linear Motion Parameters)

Dieser Struktur- bzw. Recordtyp ist Parameter bei allen linearen Interpolationsbefehlen.

Tabelle 5: Struktur- und Record-Typ LMP

Element	Typ	(Kurzwortbedeutung), Funktion
ac	double	(acceleration) Bahnbeschleunigung
vl	double	(velocity) Bahngeschwindigkeit
tvl	double	(target velocity) Bahnzielgeschwindigkeit
dtm	Feld mit MAXAXIS double	(distance to move) Dieses Feld ist je nach Index der verwendeten Achsen zu initialisieren. Die Zählweise des Index beginnt bei 0. In die einzelnen Elemente werden die gewünschten Verfahrwege je nach Positionierart (absolut bzw. relativ) eingetragen. Die Zuordnung dieser spezifizierten Verfahrwege zu den gewünschten Achskanälen muß mit dem Struktur- bzw. Record-Typ AS übereinstimmen.

4.3.2.5 Struktur- und Record-Typ CMP (Circular Motion Parameters)

Dieser Struktur- bzw. Recordtyp ist Parameter bei allen zirkularen Interpolationsbefehlen.

Tabelle 6: Struktur- und Record-Typ CMP

Element	Typ	(Kurzwortbedeutung), Funktion
ac	double	(acceleration) Bahnbeschleunigung
vl	double	(velocity) Bahngeschwindigkeit
tvl	double	(target velocity) Bahnzielgeschwindigkeit
phi	double	Verfahrwinkel in Grad
dtca1	double	(distance to center x-Achse)
dtca2	double	(distance to center y-Achse) Die Zuordnung von dtca1 und dtca2 an die gewünschten Achskanäle wird mit dem Struktur- bzw. Record-Typ AS hergestellt. Der dort im Feld 0 eingetragene Achskanal ist die x-Achse. Im Feld 1 wird entsprechend die y-Achse eingetragen.

4.3.2.6 Struktur- und Record-Typ HMP (Helical Motion Parameters)

Dieser Struktur- bzw. Recordtyp ist Parameter bei allen Schraubenlinien-Interpolationsbefehlen.

Tabelle 7: Struktur- und Record-Typ HMP

Element	Typ	(Kurzwortbedeutung), Funktion
ac	double	(acceleration) Bahnbeschleunigung
vl	double	(velocity) Bahngeschwindigkeit
tvl	double	(target velocity) Bahnzielgeschwindigkeit
phi	double	Verfahrwinkel in Grad
dtca1	double	(distance to center x-Achse)
dtca2	double	(distance to center y-Achse)
dtma3	double	(distance to move z-Achse) Die Zuordnung von dtca1, dtca2 und dtma3 an die gewünschten Achskanäle wird mit dem Struktur- bzw. Record-Typ AS hergestellt. Der dort im Feld 0 eingetragene Achskanal ist die x-Achse. Im Feld 1 wird entsprechend die y-Achse und Feld 2 die z-Achse eingetragen.

4.3.2.7 Struktur- und Record-Typ TOSI (Transputer Operating System Informations)

Dieser Struktur- bzw. Recordtyp ist Parameter für den PCAP-Initialisierungsbefehl *mcuinit()*. Nach erfolgreicher Initialisierung der MCU-3T werden folgende *rw_TOS*-Informationen (*rwtos.btl*) in die Struktur TOSI eingetragen:

Tabelle 8: Struktur- und Record-Typ TOSI

Element	Typ	(Kurzwortbedeutung), Funktion
revision	Feld mit SIZE_STRREV characters	Momentane Software-Revision der rw_TOS-Betriebssystemsoftware. Diese muß mit der Systemkonstanten REVISION übereinstimmen.
number_axis	LONGINT	Anzahl der vorhandenen Achskanäle
sysfile_loaded	LONGINT	Diese Status-Variable zeigt mit dem Wert 1 an, ob die Systemdatei bereits auf die MCU-3T übertragen wurde.

Anmerkung: Mit Hilfe des PCAP-Ladebefehls *txbf()* wird die Systemdatei system.dat, welche hauptsächlich mit Hilfe des TOOLSET-Programms mcfg.exe verändert wird, auf die MCU-3T übertragen und bewirkt dort die Initialisierung systeminterner Parameter wie z.B. Beschleunigungen, Geschwindigkeiten, Filterkoeffizienten, Grenzwerte usw. Dieser Ladevorgang muß einmalig pro Systemstart erfolgen.

4.3.2.8 Struktur- und Record-Typ CBCNT (Common Buffer CNC-Task)

Jeder CNC-Task steht ein lokaler Speicherbereich mit einer Größe von 1000 Bytes (COMMON BUFFER) zur Verfügung, auf den sowohl der PC als auch die entsprechende CNC-Task sowohl lesend als auch schreibend zugreifen kann. Dieser kann z.B. zum Aufbau eines benutzerspezifischen Befehlssatzes verwendet werden.

Der Struktur- bzw. Recordtyp *CBCNCT* ist Parameter für die PCAP-Befehle *rdcbcnct()* und *wrcbcnct()*, mit deren Hilfe die COMMON BUFFER gelesen bzw. beschrieben werden können.

Tabelle 10: Struktur- und Record-Typ CBCNCT

Element	Typ	(Kurzwortbedeutung), Funktion
TaskNr	LONGINT	Task-Nummer (0..3)
size	LONGINT	Größe des Buffers [Bytes]
Buffer	Pointer	Zeiger auf einen Puffer, welcher zur MCU-3T übertragen werden soll, bzw. von der MCU-3T gelesen werden soll. Der Puffer muß mindestens size Bytes groß sein!

4.3.2.9 Struktur- und Record-Typ CNCTS (Computerized Numerical Control Task Status)

Dieser Struktur- bzw. Recordtyp ist Parameter für den PCAP-Statusabfragebefehl *rdcncts()*.

Tabelle 11: Struktur- und Record-Typ CNCTS

Element	Typ	(Kurzwortbedeutung), Funktion
errnum	LONGINT	Interne CNC-Task Fehlernummer. Sofern kein Fehler aufgetreten ist, hat errnum den Wert 0.
errline	LONGINT	Im Zusammenhang mit errnum wird mit diesem Element die fehlerverursachende Quelltextzeile des CNC Stand- Alone-Applikations-Programmes angezeigt.
stackfree	LONGINT	Momentan freier Stackbereich [Bytes] für die CNC- Task.
running	LONGINT	Dieses Status-Flag zeigt mit dem Wert 1 an, ob die CNC-Task gerade ein Programm abarbeitet.

4.4 PCAP-Hochsprachen-Funktionenreferenzliste

4.4.1 Aufbau der Referenzliste

Die Funktionen- und Befehls-Referenzliste ist alphabetisch sortiert. Die Beschreibung der einzelnen Befehle und Funktionen hat dabei folgenden Aufbau:

FUNKTIONSNAME: Dies ist der Name, mit dessen Hilfe die nachfolgend beschriebene Funktion aufgerufen wird.

KURZWORTBEDEUTUNG: Hier steht die ausführliche Beschreibung des entsprechenden Funktionsnamens.

TURBO PASCAL: Hier stehen die Prototypdefinitionen für die Programmiersprache *Turbo Pascal*. Es wird ersichtlich, welche Parameter für den entsprechenden Funktionsaufruf benötigt werden.

C: Prototypdefinition für die Programmiersprache *C*, sonst wie *Turbo Pascal*.

TSRP-KOMPONENTEN: Verschiedene Funktionen benötigen als Parameter Komponenten der Struktur bzw. des Records TSRP. Diese werden hier aufgeführt.

BESCHREIBUNG: Klartextbeschreibung des Befehls.

RÜCKGABEWERT: Sofern die Funktion einen Rückgabewert zurückliefert, wird dieser hier beschrieben.

ANMERKUNG: Bei immer wiederkehrenden Anmerkungen und Erläuterungen wird hier auf die entsprechenden Kapitel verwiesen.

BEISPIEL: Gelegentlich werden Beispiele für die entsprechenden Funktionsaufrufe gegeben.

4.4.2 Generelle Informationen

Alle Befehle und Funktionen, außer den *spool*-Befehlen, werden unmittelbar nach dem Aufruf ausgeführt. Bei allen *move*- und *jog*-Befehlen muß vor ihrer Ausführung sichergestellt werden, daß die beteiligten Achsen zuvor in Lageregelung geschaltet wurden (PCAP-Befehl *cl()*). Weiterhin wird bei den Bewegungsfunktionen z.T. zwischen Absolut- und Relativ-Verfahrbefehlen unterschieden. Die absoluten Verfahrbefehle werden im Absolutmaßsystem, also auf den Maschinennullpunkt bezogen, ausgeführt. Die relativen Verfahrbefehle werden inkremental, also von der momentanen Motorposition ausgehend, ausgeführt.
Das Ende der Profilabarbeitung wird sowohl im Direkt-Modus als auch im Spool-Modus durch das *pe*-Flag im *axst*-Register der Struktur bzw. dem Record TSRP angezeigt [Kapitel 4.4.27 - *rdaxst()*].
Bei den achsspezifischen Bewegungskommandos (*jog*-Befehle) werden alle Systemparameter wie Positionen, Verfahrwege, Beschleunigungen und Geschwindigkeiten in den im TOOLSET-Programm *mcfg.exe* festgelegten achsspezifischen Einheiten angegeben. Bei den Interpolationsbefehlen (*move*-Befehle) werden die in der Struktur (Record) TRU angewählten Einheiten herangezogen.
Die Umrechnung zwischen anwendungsspezifischen und systeminternen Einheiten erfolgt automatisch mit Hilfe der in *mcfg.exe* festgelegten Faktoren. Maßgeblich für die Umrechnung sind die Enkoderauflösung bzw. Schrittzahl, der Getriebefaktor und die angewählten Weg- und Zeiteinheiten.

4.4.3 azo, activate zero offsets

TURBO PASCAL: procedure azo(set_:integer; softint:integer);

C: void azo(int set, int softint);

BESCHREIBUNG: Jedem Achskanal können fünf unterschiedliche Nullpunktverschiebungen *(zero offsets)* zugeordnet werden. Mit Hilfe des Befehls *azo()* können die gewünschten achsspezifischen Verschiebungsparameter aktiviert werden. Im Parameter *set* (bzw. *set_*) wird spezifiziert, welcher Satz von Nullpunktverschiebungen aktiviert werden soll. Diese Variable wählt mit dem Wert 0..4 den gewünschten Satz von Nullpunktverschiebungen an. Sofern die Variable jedoch einen Wert größer als 4 hat, werden keine Nullpunktverschiebungen mehr berücksichtigt.

ANMERKUNG: Nullpunktverschiebungen dienen zur Festlegung eines neuen Koordinatensystems, ohne dabei den tatsächlichen Maschinennullpunkt beeinflussen (neu setzen) zu müssen.

4.4.4 cl, close loop

TURBO PASCAL: procedure cl(var as:AS; softint:integer);

C: void cl(struct AS far *as, int softint);

BESCHREIBUNG: Alle in AS spezifizierten Achskanäle werden mit diesem Befehl in die Lageregelung gebracht. Dabei werden die Istpositionen der beteiligten Achsen als Sollpositionen übernommen, um große Regelabweichungen zu vermeiden. Zusätzlich werden alle mit PAE-projektierten Digital-Ausgänge gesetzt. Diese Ausgänge können beispielsweise zur Ansteuerung von Relais verwendet werden, mit welchen wiederum die Freigabe der Leistungsverstärkereinheiten erfolgt.
Je nach selektiertem Achskanal werden die Relais K2 (Achskanal 1), K3 (Achskanal 2) und K4 (Achskanal 3) eingeschaltet [BHB / Kapitel 4.1.2.8].

ANMERKUNG: Die Lageregelung bewirkt das Abarbeiten des PIDF-Filters mit den entsprechend eingestellten Filterkoeffizienten.
Beim Schließen des Lageregelkreises werden alle Spoolerdaten der spezifizierten Achskanäle verworfen!

4.4.5 contcnct, continue numeric controller task

TURBO PASCAL: procedure contcnct(TaskNr:integer; softint:integer);

C: void contcnct(int TaskNr, int softint);

BESCHREIBUNG: Mit diesem Befehl kann ein SAP-Programm, welches zuvor mit dem SAP-Befehl *STOP*, *STOPCNCT()* oder mit dem PCAP-Befehl *stopcnct()* angehalten wurde, wieder fortgesetzt werden. Die in *TaskNr* (Werte 0..3) angewählte Task wird fortgesetzt.

ANMERKUNG: Ein SAP-Programm, welches mit dem SAP-Befehl *ABORT* angehalten wurde, kann nur mit dem SAP-Befehl *STARTCNCT()* oder dem PCAP-Befehl *startcnct()* neu gestartet, also nicht fortgesetzt, werden.

4.4.6 ctru, change trajectory units

TURBO PASCAL: procedure ctru(var tru:TRU; softint:integer);

C: void ctru(struct TRU far *tru, int softint);

BESCHREIBUNG: Mit diesem Befehl können die Einheiten für die Geschwindigkeits-, Beschleunigungs- und Positionsparameter aller Interpolationsbefehle (*move*-Befehle) umgeschaltet werden. Die Parameter werden in den gewählten Einheiten angegeben.

Für die TRU-Strukturkomponente *pu* (position unit) sind folgende Werte erlaubt:

Index	Einheit	Beschreibung
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radiant
6	counts	Counts
7	steps	Steps

Für die TRU-Strukturkomponente *tu* (time unit) sind folgende Werte erlaubt:

Index	Einheit	Beschreibung
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling Time

ANMERKUNG: Der Defaultwert für *pu* und *tu* ist 0. Somit wird für alle Wegangaben die Einheit [mm], für Geschwindigkeiten die Einheit [mm/s] und Beschleunigungen [mm/s^2] angenommen. Die gewählten Einheiten werden nur für Interpolationsbefehle (alle *move*-Befehle) herangezogen! Sofern es sich um achsspezifische Bewegungskommandos handelt (alle *jog*-Befehle), werden die in *mcfg.exe*

spezifizierten Achs-Einheiten berücksichtigt.
Die angewählten Einheiten sind auch maßgebend für ein evtl. parallel ablaufendes SAP-Programm.

4.4.7 dummy, dummy function call

TURBO PASCAL: function dummy(softint:integer):integer;

C: int dummy(int softint);

BESCHREIBUNG: Dieser Befehl hat keine Wirkung auf die MCU-3T, liefert jedoch den Wert -1 zurück. Sofern dieser Wert zurückgeliefert wird, kann davon ausgegangen werden, daß der richtige TSR-Treiber *(mcutsr.exe)* geladen wurde.

4.4.8 InitMcuSystem, initialise mcu system

TURBO PASCAL: function InitMcuSystem(var tsrp:TSRP; softint:integer):integer;

C: int InitMcuSystem(var TSRP far *tsrp, int softint);

BESCHREIBUNG: Diese Funktion führt die komplette Software-Initialisierung des Antriebssystemes durch. Der Funktionsaufruf sollte zu Beginn jedes PCAP-Anwenderprogramms ausgeführt werden. Innerhalb dieser Funktion werden verschiedene PCAP-Basis-Funktionen aufgerufen. Unter anderem werden die Achsennummern {an} in der Struktur *tsrp* initialisiert. Der Anwender muß dafür sorgen, daß die tatsächlich vorhandene Achsanzahl in der Systemkonstanten *REALAXIS* entsprechend gesetzt ist. Die Funktion überprüft weiterhin, ob der TSR-Treiber *mcutsr.exe* resident in den PC-Arbeitsspeicher geladen wurde, und ob es sich bei dem geladenen Treiber um den korrekten Treiber handelt. Sofern die Systemdatei *system.dat* noch nicht auf die MCU-3T übertragen wurde, wird dies hier getan. Am Ende der Funktion werden die Achsparameter von allen Achsen in die Struktur *tsrp* eingelesen.

ANMERKUNG: PCAP-Befehle *txbf()*, *mcuinit()*, Struktur bzw. Record-Typ TOSI

RÜCKGABEWERT: Die Funktion kann folgende Werte zurückliefern:

Rückgabewert	Fehler-Beschreibung
0	kein Fehler
30	TSR-Treiber mcutsr.exe ist nicht resident geladen an softint
31	falscher TSR-Treiber
32	der Zugriff auf die MCU-3T ist nicht möglich dieser Fehler kann folgende Ursachen haben: - die rw_TOS-Betriebssystemsoftware ist nicht geladen (-> mcbt.exe) - die MCU-3T Basis-Adresse [IHB / Kapitel 4.2] ist falsch eingestellt - die MCU-3T ist nicht im PC installiert
33	falsche rw_TOS-Betriebssystemsoftware
lderr	Fehler-Rückgabewert von PCAP-Befehl txbf()

4.4.9 ja, jog absolute

TURBO PASCAL: procedure ja(var as:AS; var tsrp:TSRP; softint:integer);

C: void ja(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].tp
n = 0 .. Anzahl der vorhandenen Achsen-1

BESCHREIBUNG: Die in *AS* gewählten Achskanäle werden auf die in *TSRP[n].tp* angegebenen Zielpositionen mit Hilfe eines Trapez-Drehzahl-Profils absolut verfahren. Zur Profilgenerierung werden die achsspezifischen Systemparameter *jac* (*jog*-Beschleunigung), *jvl* (*jog*-Geschwindigkeit) und *jtvl* (*jog*-Zielgeschwindigkeit) herangezogen. Diese Parameter können mit Hilfe von Schreib- und Lese-Befehlen jederzeit gesetzt und abgefragt werden. Die Defaultwerte werden im Hilfsprogramm *mcfg.exe* spezifiziert. Die Bahnparameter werden in den in *mcfg.exe* festgelegten achsspezifischen Einheiten (Weg, Zeit) angegeben.

ANMERKUNG: Sofern dieser Befehl gleichzeitig für mehrere Achsen ausgeführt wird, können diese aufgrund der achsspezifischen Systemparameter zu unterschiedlichen Zeitpunkten die Zielpositionen erreichen [Kapitel 2.2.8.1].
Die achsspezifischen Parameter wie z.B. Beschleunigungen und Geschwindigkeiten können jederzeit mit Hilfe von Lese- und Schreibbefehlen abgefragt bzw. gesetzt werden.

4.4.10 jhi, jog home index

TURBO PASCAL: procedure jhi(var as:AS; var tsrp:TSRP; softint:integer);

C: void jhi(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].tp
n = 0 .. Anzahl der vorhandenen Achsen-1

BESCHREIBUNG: Mit Hilfe dieses Befehls wird der Indexsuchlauf aller in *AS* angewählten Achskanäle gestartet. Der Suchlauf wird entweder beim Aktivieren des Index- (Nullspur) Signals vom Inkrementalenkoder oder nach Überschreiten der in *tp* spezifizierten Weg- bzw. Winkelangabe beendet. Der Suchlauf wird mit Hilfe eines Trapez-Drehzahl-Profils durchgeführt. Die Parameter für den Profilgenerator sind dabei die Systemdaten *hac* und *hvl*, welche mit Hilfe von *mcfg.exe* bzw. den entsprechenden Schreibbefehlen gesetzt werden können. Beim Erkennen des Indexsignals (Nullspur), wird der Motor mit der Beschleunigung *hac* auf Geschwindigkeit 0 abgebremst. Der Parameter *tp* wird als relativer Verfahrweg in der achsspezifischen Positionseinheit angegeben. Die Suchrichtung wird durch das Vorzeichen von *tp* bestimmt. Im allgemeinen wird das Achssystem zuerst auf einen Referenzschalter (Nocken) gefahren. Um die mechanische Ungenauigkeit dieses Nockens zu eliminieren, bietet es sich an, im Anschluß den Index-Suchlauf durchzuführen. Die Befehlsausführung kann mit Hilfe des Profil-Flags im *axst*-Register und der Zustand des Index-Signals mit dem *digi*-Register [Kapitel 4.4.34] abgefragt werden. Das Profilflag bleibt bis zum Ende des Suchlaufs auf 1 gesetzt.

ANMERKUNG: Um eine möglichst genaue Index-Positionierung zu realisieren, sollte der Suchlauf mit möglichst kleiner Verfahrgeschwindigkeit ausgeführt werden. Es gibt jedoch auch die Möglichkeit, den Suchlauf in zwei Schritten durchzuführen. Im ersten Schritt kann der Suchlauf z.B. in positive Verfahrrichtung mit relativ großer Suchgeschwindigkeit gestartet werden. Im zweiten Schritt wird der Suchlauf dann in die negative Richtung mit kleiner Suchgeschwindigkeit abgeschlossen. Die Suchgeschwindigkeit kann mit den PCAP-Befehlen *rdhvl()* und *wrhvl()* gelesen und geschrieben werden.

4.4.11 jhl, jog home left

TURBO PASCAL: procedure jhl(var as:AS; softint:integer);

C: void jhl(struct AS far *as, int softint);

BESCHREIBUNG: Dieser Befehl startet den Referenzsuchlauf aller in *AS* spezifizierten Achskanäle in negative Verfahrrichtung. Der Suchlauf wird mit Hilfe eines Endlos-Trapez-Drehzahlprofils ausgeführt. Die achsspezifischen Systemdaten *hac* und *hvl* dienen dabei als Parameter zur Profilgenerierung. Sofern ein mit REF-Funktion projektierter Digital-Eingang der MCU-3T bei dem gewählten Achskanal aktiviert wird, wird der Suchlauf durch Abbremsen (mit *hac*) der Achse auf Geschwindigkeit 0 beendet. Dieser Zustand kann mit Hilfe des *pe*-Profil-Flags im *axst*-Register abgefragt werden. Das Profilflag bleibt bis zum Ende des Suchlaufes auf 0 gesetzt.

4.4.12 jhr, jog home right

TURBO PASCAL: procedure jhr(var as:AS; softint:integer);

C: void jhr(struct AS far *as, int softint);

BESCHREIBUNG: Die Funktionsweise dieses Befehls ist identisch mit dem PCAP-Befehl *jhl()*, jedoch wird der Suchlauf in die positive Verfahrrichtung gestartet.

4.4.13 jr, jog relative

TURBO PASCAL: procedure jr(var as:AS; var tsrp:TSRP; softint:integer);

C: void jr(struct AS far *as, struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].tp
n = 0 .. Anzahl der vorhandenen Achsen-1

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *ja()*, mit dem Unterschied, daß es sich bei der Wegangabe *tp* um einen relative (inkrementale) Verfahrstrecke handelt. Ausgehend von der momentanen Position wird der Motor um die angegebene Strecke (bzw. Winkel) nach links (negative Werte) bzw. rechts (positive Werte) verfahren.

4.4.14 js, jog stop

TURBO PASCAL: procedure js(var as:AS; softint:integer);

C: void js(struct AS far *as, int softint);

BESCHREIBUNG: Die in AS gewählten Achskanäle werden mit der achsspezifischen Verzögerung *sdec* auf Geschwindigkeit 0 abgebremst und in Lageregelung gehalten. Bis zum Ende des Abbremsvorgangs ist das *pe*-Flag im *axst*-Register rückgesetzt. Die Verzögerung *sdec* kann mit Hilfe von Schreib- und Lese-Befehlen jederzeit gesetzt und abgefragt werden. Der Defaultwert wird im Hilfsprogramm *mcfg.exe* spezifiziert.

ANMERKUNG: Sofern dieser Befehl gleichzeitig für mehrere Achsen ausgeführt wird, können diese aufgrund der achsspezifischen Systemparameter zu unterschiedlichen Zeitpunkten die Zielpositionen erreichen [Kapitel 2.2.8.1].

4.4.15 lps, latch position synchronous

TURBO PASCAL: procedure lps(an:integer; mst:integer; softint:integer);

C: void lps(int an, int mst, int softint);

BESCHREIBUNG: Mit diesem Befehl kann ein Latch-Vorgang synchron zum Abtastzyklus des in *an* angewählten Achskanals ausgelöst werden. Nach dem Aufruf wird die Ist-Position *{rp}* nach jeweils *mst* Abtastintervallen zwischengespeichert. Sofern ein Latch-Vorgang stattgefunden hat, wird dies im *axst*-Register im Flag *lpsf* (Bit Nr. 16) angezeigt. Mit dem PCAP-Lesebefehl *rdlp()* oder dem SAP-Achsenqualifizierer *lp* kann die zwischengespeicherte Position ausgelesen werden. Das Auslesen löscht auch das *lpsf*-Flag im *axst*-Register.

ANMERKUNG: Der Befehl wird hauptsächlich beim Aufzeichnen von Konturen und Teach-In-Anwendungen verwendet, da er das Aufzeichnen von Positionsdaten in Echtzeit von einer oder mehreren Achsen ermöglicht. Typische Werte für *mst* sind 10..100 Abtastintervalle (-> 12.8ms..128.0ms). Der genaue Wert hängt jedoch von der Verarbeitungsgeschwindigkeit der jeweiligen Applikation ab.

4.4.16 mca, move circular absolute
smca, spool motion circular absolute

TURBO PASCAL: procedure mca(var as:AS; var cmp:CMP; softint:integer);
procedure smca(var as:AS; var cmp:CMP; softint:integer);

C: void mca(struct AS far *as, struct CMP far *cmp, int softint);
void smca(struct AS far *as, struct CMP far *cmp, int softint);

BESCHREIBUNG: Dieser Befehl bewirkt die zirkulare Interpolation der ersten beiden in AS spezifizierten Achskanäle. Bezüglich der Achsenauswahl gibt es keine Einschränkungen. Die Kreisinterpolation wird auf Basis eines Trapez-Drehzahl-Profiles, d.h. unter Berücksichtigung von Maximalbeschleunigung und Maximalgeschwindigkeit, durchgeführt. Als Interpolationsparameter werden die in CMP spezifizierten Struktur- bzw. Recordkomponenten herangezogen. Dies sind die Bahnbeschleunigung *ac*, die Bahngeschwindigkeit *vl* und die Bahnzielgeschwindigkeit *tvl*. Die in *dtca1* und *dtca2* eingetragenen Koordinaten spezifizieren den Kreismittelpunkt im Absolutmaßsystem. Dabei wird *dtca1* der ersten in AS programmierten Achse und *dtca2* der zweiten in AS spezifizierten Achse zugeordnet. Die Einheiten der Bahnparameter werden mit dem PCAP-Befehl *ctru()* gewählt.
Der Winkel *phi* spezifiziert den abzufahrenden Verfahrwinkel mit der Einheit Grad. Die Drehrichtung wird durch das Vorzeichen der Winkelgröße festgelegt. Positive Werte bedeuten, Drehrichtung im Gegenuhrzeigersinn und negative Werte Drehrichtung im Uhrzeigersinn. Der Verfahrwinkelbereich ist nicht auf bestimmte Grenzen fixiert, d.h. es können auch Teil- oder Vielfach-Kreise abgefahren werden.

ANMERKUNG: Kapitel 2.3 Interpolation mit der MCU-3T.

4.4.17 mcr, move circular relative
smcr, spool motion circular relative

TURBO PASCAL: procedure mcr(var as:AS; var cmp:CMP; softint:integer);
procedure smcr(var as:AS; var cmp:CMP; softint:integer);

C: void mcr(struct AS far *as, struct CMP far *cmp, int softint);
void smcr(struct AS far *as, struct CMP far *cmp, int softint);

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *mca()* bis auf den Unterschied, daß die in *dtca1* und *dtca2* spezifizierten Koordinaten inkremental, bzw. relativ, auf die aktuelle Motorpositionen bezogen werden.

ANMERKUNG: Kapitel 2.3 Interpolation mit der MCU-3T.

4.4.18 mcuinit, motion control unit initialisation

TURBO PASCAL: procedure mcuinit(var tosi:TOSI; softint:integer);

C: void mcuinit(struct TOSI far *tosi, int softint);

BESCHREIBUNG: Mit dieser Funktion werden verschiedene Initialisierungen innerhalb des TSR-Treibers *mcutsr.exe* durchgeführt. Es wird geprüft, ob eine Kommunikation zwischen PC und MCU-3T möglich ist. Sofern dies der Fall ist, werden die von der MCU-3T zurückgelieferten *rw_TOS*-spezifischen Systemdaten in die Struktur bzw. im Record TOSI eingetragen. Anhand von TOSI können die *rw_TOS*-spezifischen Systeminformationen auf Gültigkeit abgeprüft werden.
Sofern der Kommunikationsaufbau zur MCU-3T nicht möglich war, enthält die gesamte Struktur TOSI den Wert 0.

ANMERKUNG: Dieser Befehl löst keinen Reset auf der MCU-3T aus. Dies ist mit den PCAP-Befehlen *ra()* oder *rs()* durchzuführen.
Mit dem Rückgabewert TOSI.sysfile_loaded kann festgestellt werden, ob die Systemdatei *system.dat* bereits mit Hilfe des PCAP-Ladebefehls *txbf()* auf die MCU-3T übertragen wurde. Ist dieser Wert 0, so muß nach erfolgreichem *mcuinit()*-PCAP-Befehl der PCAP-Befehl *txbf()* ausgeführt werden, damit ein Arbeiten mit der MCU-3T möglich ist.
Bei den mitgelieferten PCAP-Beispielprogrammen ist dieser Befehl in der Funktion *InitMcuSystem()* enthalten. Dort wird der Kontrollmechanismus der Systeminitialisierung nochmals verdeutlicht.

4.4.19 mha, move helical absolute
smha, spool motion helical absolute

TURBO PASCAL: procedure mha(var as:AS; var hmp:HMP; softint:integer);
procedure smha(var as:AS; var hmp:HMP; softint:integer);

C: void mha(struct AS far *as, struct HMP far *hmp, int softint);
void smha(struct AS far *as, struct HMP far *hmp, int softint);

BESCHREIBUNG: Mit diesem Kommando wird eine Helix- oder Schraubenlinieninterpolation durchgeführt. Dieser Befehl ist eine Erweiterung der Zirkularinterpolation. Deshalb treffen die beim PCAP-Befehl *mca()* genannten Aussagen für dieses Kommando ebenfalls zu, mit dem Unterschied daß die Bahnparameter in der Struktur bzw. im Record HMP eingetragen werden. Für die dritte in AS spezifizierte Achse kann zusätzlich der Parameter *dtma3* programmiert werden. Dies ist der absolute Verfahrweg für die dritte Achse. Während die ersten beiden Achsen eine Zirkularinterpolation durchführen, wird die dritte Achse linear verfahren. Alle drei Achsen erreichen zum gleichen Zeitpunkt ihre Zielpositionen.

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert!

4.4.20 mhr, move helical relative
smhr, spool motion helical relative

TURBO PASCAL: procedure mhr(var as:AS; var hmp:HMP; softint:integer);
procedure smhr(var as:AS; var hmp:HMP; softint:integer);

C: void mhr(struct AS far *as, struct HMP far *hmp, int softint);
void smhr(struct AS far *as, struct HMP far *hmp, int softint);

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *mha()* bis auf den Unterschied, daß die in *dtca1*, *dtca2* und *dtma3* programmierten Wegangaben inkremental, bzw. relativ, auf die momentanen Motorpositionen bezogen werden.

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert!

4.4.21 mla, move linear absolute
smla, spool motion linear absolute

TURBO PASCAL: procedure mla(var as:AS; var lmp:LMP; softint:integer);
procedure smla(var as:AS; var lmp:LMP; softint:integer);

C: void mla(struct AS far *as, struct LMP far *lmp, int softint);
void smla(struct AS far *as, struct LMP far *lmp, int softint);

BESCHREIBUNG: Mit diesem Befehl wird eine Linear- oder Geradeninterpolation mit absoluten Zielangaben durchgeführt. Zur Interpolation sind alle Achsen im n-dimensionalen Raum zulässig. Welche Achsen an der Interpolation teilnehmen sollen, wird in AS spezifiziert. Mit Hilfe von LMP werden die Bahnbeschleunigung *ac*, Bahngeschwindigkeit *vl* und Bahnzielgeschwindigkeit *tvl* für die Linearinterpolation festgelegt. Die Einheiten der Bahnparameter werden mit dem Befehl *ctru()* gewählt.
Je nach Anzahl der Achsen *unoa* werden die gewünschten Achsen im Feld *san* und die entsprechenden Verfahrwege im Feld *dtm* eingetragen. Dabei wird der Verfahrweg im Feld *dtm[n]* der Achsennumer *n+1* zugeordnet. Die Interpolation bezieht sich auf die in AS eingetragenen Achsen. Die Verfahrwege werden als absolute, also auf den Maschinennullpunkt bezogene, Weg- bzw. Winkelinformationen interpretiert.

ANMERKUNG: Kapitel 2.3 Interpolation mit der MCU-3T.

4.4.22 mlr, move linear relative
smlr, spool motion linear relative

TURBO PASCAL: procedure mlr(var as:AS; var lmp:LMP; softint:integer);
procedure smlr(var as:AS; var lmp:LMP; softint:integer);

C: void mlr(struct AS far *as, struct LMP far *lmp, int softint);
void smlr(struct AS far *as, struct LMP far *lmp, int softint);

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *mla()*, jedoch werden die im Feld *dtm* spezifizierten Verfahrwege inkremental, bzw. relativ zur momentanen Motorposition, interpretiert.

ANMERKUNG: Kapitel 2.3 Interpolation mit der MCU-3T.

4.4.23 ms, motion stop

TURBO PASCAL: procedure ms(var as:AS; softint:integer);

C: void ms(struct AS far *as, int softint);

BESCHREIBUNG: Die in *AS* gewählten Achskanäle werden mit der zur Zeit gültigen Bahnbeschleunigung bzw. Achsenverzögerung auf Geschwindigkeit 0 abgebremst und in Lageregelung gehalten. Bis zum Ende des Abbremsvorgangs ist das *pe*-Flag im *axst*-Register rückgesetzt. Der Richtungsvektor einer evtl. gerade ablaufenden Interpolation wird durch diesen Befehl nicht verändert. Falls die angewählten Achsen gerade einen Kreis abfahren, erfolgt die Abbremsung auf der Kreisbahn mit der angegebenen Bahnbeschleunigung.
Achsen, die mit einer Endgeschwindigkeit verfahren, werden mit der achsspezifischen Verzögerung *sdec* auf Geschwindigkeit 0 abgebremst

ANMERKUNG: Nicht gemeinsam interpolierende Achsen können den Zielpunkt zu unterschiedlichen Zeitpunkten erreichen.

4.4.24 ol, open loop

TURBO PASCAL: procedure ol(var as:AS; softint:integer);

C: void ol(struct AS far *as, int softint);

BESCHREIBUNG: Dieser Befehl öffnet den Lageregelkreis aller in *AS* angewählten Achsen. Auf den Motor-Command-Ports wird je bei Servo-Achsen 0V Ausgangsspannung und bei Schrittmotorachsen 0Hz Schrittfrequenz ausgegeben. Alle mit PAE-Funktion projektierten MCU-3T Digitalausgänge werden für die programmierten Achskanäle inaktiv gesetzt. Je nach selektiertem Achskanal werden die Relais K2 (Achskanal 1), K3 (Achskanal 2) und K4 (Achskanal 3) abgeschaltet [BHB / Kapitel 4.1.2.8].

ANMERKUNG: Dieser Befehl wird hauptsächlich in Ausnahmesituationen wie Endschalterbegrenzung, Schleppfehlerüberschreitung usw. verwendet.

4.4.25 ra, reset axis

TURBO PASCAL: procedure ra(var as:AS; softint:integer);

C: void ra(struct AS far *as, int softint);

BESCHREIBUNG: Mit diesem Befehl kann ein achsspezifischer Rücksetzvorgang durchgeführt werden. Dieser bewirkt den Abbruch eines evtl. ablaufenden Profils, das Öffnen des Lageregelkreises, die Sollwertabschaltung, das Verwerfen von evtl. vorhanden Spoolerdaten und das Nullsetzen der Positionsregister. Die Ausgänge werden auf die projektierten Default-Werte gesetzt. Die achsspezifischen Overridefaktoren (PCAP-Befehl *wrjovr()* und *wrtrovr()*) werden auf den Wert 1.0 gesetzt. Die evt. projektierten Softwareendlagen werden für die in *ra()* angewählten Achskanäle nicht mehr überwacht.

ANMERKUNG: Alle Systemdaten wie Beschleunigungen, Geschwindigkeiten, Filterparameter usw. bleiben gespeichert und brauchen deshalb nicht neu geladen zu werden. Dieser Befehl wird hauptsächlich bei der Systeminitialisierung bzw. in Ausnahmesituationen verwendet.

4.4.26 rdap, read axis parameters

TURBO PASCAL: procedure rdap(var tsrp:TSRP; softint:integer);

C: void rdap(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: alle, d.h. TSRP[n].an ... TSRP[n].ifs

BESCHREIBUNG: Mit diesem Befehl können alle achsspezifischen Ein- und Ausgangsgrößen der Struktur bzw. des Records TSRP mit einem Lesebefehl eingelesen werden.

RÜCKGABEWERT: Nach Ausführung des Befehls stehen die Ein- und Ausgangsgrößen in den jeweiligen Struktur- bzw. Record-Komponenten der Struktur bzw. dem Record TSRP.

ANMERKUNG: Die einzelnen Struktur- bzw. Record-Komponenten können auch mit speziellen Lesebefehlen abgefragt werden. Im Normalfall werden diese Lesebefehle wegen der kürzeren Zugriffszeit bevorzugt.

4.4.27 rdaxst, read axis status

TURBO PASCAL: procedure rdaxst(var tsrp:TSRP; softint:integer);

C: void rdaxst(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].axst

BESCHREIBUNG: Mit diesem Befehl können verschiedene Status- und Errorflags der Rampen- und Interpolationstask achsspezifisch abgefragt werden. Normalerweise wird dieser Befehl im PCAP-Programm zyklisch wiederholt, um mit dem nachfolgend beschriebenen *pe*-Flag abzuprüfen, ob die Verfahrkommandos der beteiligten Achsen fertig abgearbeitet wurden. Zusätzlich werden mit diesem Befehl eine Reihe von Fehlerflags im *axst*-Register aktualisiert. Diese sollten ebenfalls zyklisch ausgewertet werden, um ein sicheres Betriebsverhalten durch das PCAP-Programm zu garantieren.

RÜCKGABEWERT: Der bitkodierte Rückgabewert befindet sich nach Ausführung dieses Befehls in der Struktur- bzw. Recordkomponente *axst* und hat den in nachfolgend abgedruckter Tabelle beschriebenen Aufbau.

ANMERKUNG: [BHB / Kapitel 4.4.8.1] und [BHB / Kapitel 4.4.8.3].

Tabelle 15: Bitkodierter Aufbau des axst-Wortes

Bit-Nr.	Name	Funktion
0		Nicht belegt, dieses Flag hat immer den Wert 0.
1	*eo*	Emergency-Out Error-Flag: Hat den Wert 1, wenn ein als EO-projektierter Digitaleingang aktiv ist.
2	*dnr*	Drive-Not-Ready Error-Flag: Hat den Wert 1, wenn ein als DR-projektierter Digitaleingang inaktiv ist.
3	*lslh*	Limit-Switch left Hardware Error-Flag: Hat den Wert 1, wenn ein als LSL_TOM oder LSL_SMA projektierter Digitaleingang aktiv ist.
4	*lsrh*	Limit-Switch right Hardware Error-Flag: Hat den Wert 1, wenn ein als LSR_TOM oder LSR_SMA projektierter Digitaleingang aktiv ist.
5	*lsls*	Limit-Switch left Software Error-Flag: Hat den Wert 1, wenn die linke Software-Endlage überschritten wird. Die linke Software-Endlage ist im achsspezifischen Systemparameter {sll} abgelegt. Damit dieses Flag aktiv wird müssen zusätzlich zwei Bedingungen gelten: Die Softwareendlage muß mit einer der Funktionen TOM oder SMA projektiert werden und zuvor muß das shp()-Kommando ausgeführt worden sein.
6	*lsrs*	Limit-Switch right Software Error-Flag: Hat den Wert 1, wenn die rechte Software-Endlage überschritten wird. Die rechte Software-Endlage ist im achsspezifischen Systemparameter {slr} abgelegt. Damit dieses Flag aktiv wird müssen zusätzlich zwei Bedingungen gelten: Die Softwareendlage muß mit einer der Funktionen TOM oder SMA projektiert werden und zuvor muß das shp()-Kommando ausgeführt worden sein.
7	*mpe*	Maximum Position Error-Flag: Hat den Wert 1, wenn der zulässige Schleppfehler überschritten wurde. Der maximal erlaubte Schleppfehler wird im Systemparameter {mpe} spezifiziert. Mit Hilfe der PCAP-Befehle wrmpe()und rdmpe() kann der Parameter auch während der Laufzeit verändert werden
8	*dhef*	Data handling error flag: Hat den Wert 1 wenn ein Datenfehler (z.B. Inkonsistente Profildaten) durch das rw_TOS-Betriebssystems festgestellt wird. Sofern dieser Fehler vorliegt, werden die Regelkreise der jeweiligen Achse geöffnet. Das Schließen des Regelkreises ist nur nach Systemneustart (*mcbt.exe*) oder nach Ausführung der Befehle *ra()* [Kapitel ra, reset axis4.4.25] oder *rs()* [Kapitel 4.4.67] möglich.
9	*cef*	Daten-Konfigurations-Fehler. Das cef-Flag wird gesetzt, wenn die Informationen für Betriebsarten, Signalverarbeitung oder CPU-Nummer auf der MCU-3T nicht mit den Systemdaten (system.dat) übereinstimmen Der Konfigurations-Check wird nach folgenden Ereignissen automatisch durchgeführt: • Nach jeder Rücksetzanweisung (z.B. PCAP-Befehl rs()), • Nach jedem Übertragen der Systemdatei system.dat mit dem PCAP-Befehl txbf(). Die Behebung der Fehlerursache kann durch Speichern der Systemdaten im Menü [Save Changes] erfolgen.
10..11		Nicht belegt, diese Flags haben immer den Wert 0.
12	*pe*	Profil-Ende Status-Flag: Hat den Wert 1, wenn das Profilende erreicht ist.
13	*cl*	Closed-Loop Status-Flag: Hat den Wert 1, wenn der Achskanal in Lageregelung ist.
14	*ip*	In-Position Status-Flag: Hat den Wert 1, wenn das Profilende erreicht wurde und zusätzlich die Differenz von Soll- und Istposition des Achskanals die im achsspezifischen Systemparameter {ipw} enthaltene Positionsdifferenz unterschreitet
15	*ui*	User Input Status-Flag: Hat den Wert 1, wenn ein als UI-projktierter Digital Eingang aktiv ist.
16	*lpsf*	Das Latch Position Synchronous Flag zeigt an, daß ein Latch-Vorgang synchron zum Abtastzyklus stattgefunden hat [Kapitel 4.4.15], oder ein mit LP-Funktion projektierter Digital-Eingang aktiviert wurde[BHB / Kapitel 4.4.3.1].
17..31		Nicht belegt, diese Flags haben immer den Wert 0.

4.4.28 rdaxstb, read axis status bit

TURBO PASCAL: function rdaxstb(an:integer; bitnr:integer; softint:integer):boolean;

C: int rdaxstb(int an, int bitnr, int softint);

BESCHREIBUNG: Mit dieser Funktion kann eine MCU-3T Achsen-Statusinformation abgefragt werden. Die Achsnummer muß im Parameter *an* (0, 1, ... *REALAXIS*) spezifiziert werden.

RÜCKGABEWERT: Die Funktion liefert den Wert 1 bzw. TRUE zurück, sofern der entsprechende Eingang von *bitnr* aktiv ist.
Die Zuordnung von *bitnr* zu den jeweiligen Achsen-Statusinformationen wird in Tabelle 15 beschrieben, jedoch erfolgt bei *bitnr* die Zählweise bei dem Wert 1, d.h. um beispielsweise *pe* abzufragen, muß *bitnr* den Wert 13 haben!

ANMERKUNG: [BHB / Kapitel 4.4.8.1], [BHB / Kapitel 4.4.8.3] und PCAP-Befehl *rdaxst()*

4.4.29 rdcbcnct, read common buffer CNC-Task

TURBO PASCAL: function rdcbcnct(var cbcnct:CBCNCT; softint:integer):integer;

C: int rdcbcnct(struct CBCNCT far *cbcnct, int softint);

BESCHREIBUNG: Jede CNC-Task hat einen lokalen Speicherbereich, den sogenannten Common-Buffer, der sowohl von der jeweiligen CNC-Task als auch durch ein PCAP-Programm gelesen und beschrieben werden kann.
Mit dieser Funktion kann der komplette CNC-Task-spezifische Buffer (oder nur ein Teil davon) eingelesen werden. Mit dem Funktionsparameter *cbcnct* erfolgt die Auswahl des CNC-Task-Buffers, die einzulesende Größe in Bytes und die Speicheradresse, wohin dieser Block eingelesen werden soll.

RÜCKGABEWERT: DIE FUNKTION RDCBCNCT() HAT FOLGENDEN BITKODIERTEN RÜCKGABEWERT:
Bit 0: 1 wenn ungültige Task-Nummer
Bit 1: 1 wenn maximal erlaubte Buffergröße überschritten
Dies bedeutet, daß die Funktion im Normalfall den Wert 0 zurückliefert.

ANMERKUNG: DIE CNC-TASK-SPEZIFISCHE BUFFERGRÖßE BETRÄGT 1000 BYTES.
Der Struktur- (Record) Aufbau von CBCNCT ist im Kapitel 4.3.2.8 abgedruckt.
PCAP-Befehl *wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

4.4.30 rdcd, read common double

TURBO PASCAL: procedure rdcd(ndx: integer; var cdbuf:CDBUF; softint:integer);

C: void rdcd(int ndx, struct CDBUF far *cdbuf, int softint);

BESCHREIBUNG: Mit dieser Funktion können vordefinierte Variablen der CNC-Task eingelesen werden. Dies sind die *rw_SymPas*-Variablen CD0 .. CD99. Der erste Parameter gibt dabei die Nummer -*index*- der gewünschten einzulesenden Variablen an. Der Wertebereich von *index* ist dabei 0 bis 99. Der zweite Parameter ist ein Zeiger auf ein Feld mit 100 double-Variablen.

RÜCKGABEWERT: Der Befehl *rdcd()* trägt im mit *index* spezifizierten Feld den aktuellen Wert der entsprechenden *CD*-Variable ein.

ANMERKUNG: Der Inhalt aller Common Variablen bleibt auch nach einem Systemrücksetzvorgang, welcher z.B. durch das *rs()*-Kommando ausgeführt wird, gespeichert. Wenn dies nicht erwünscht ist, sollten die betreffenden Variablen beim Programmstart auf den gewünschten Wert gesetzt werden.

4.4.31 rdci, read common integer

TURBO PASCAL: procedure rdci(ndx: integer; var cibuf:CIBUF; softint:integer);

C: void rdci(int ndx, struct CIBUF far *cibuf, int softint);

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *rdcd()*, jedoch werden hier nicht Werte vom Typ double eingelesen sondern vom Typ LONGINT. Es handelt sich dabei um die *rw_SymPas*-Variablen CI0 .. CI99.

4.4.32 rdcncts, read computerized numeric controller task status

TURBO PASCAL: procedure rdcncts(TaskNr:integer; var cncts:CNCTS; softint:integer):integer;

C: void rdcncts(int TaskNr, struct CNCTS far *cncts, int softint);

BESCHREIBUNG: Mit Hilfe dieses Befehls kann der aktuelle Zustand der in *TaskNr* (Werte 0..3) angewählten CNC-Task abgefragt werden. Die Ergebnisse befinden sich nach Ausführung dieses Befehls in der Struktur bzw. dem Record *CNCTS*.

RÜCKGABEWERT: Die Rückgabewerte, welche sich nach Ausführung von *rdcncts()* in CNCTS befinden, werden im Kapitel 4.3.2.9 beschrieben.

4.4.33 rdigi, reset digital inputs

TURBO PASCAL: procedure rdigi(var tsrp:TSRP; softint:integer);

C: void rdigi(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].digi
n = 0 .. Anzahl der Achsen -1

BESCHREIBUNG: Mit diesem Befehl können die in *digi* achsspezifisch gespeicherten Statusinformationen gelöscht werden.

ANMERKUNG: *rddigi()* [Kapitel 4.4.34]

4.4.34 rddigi, read digital inputs

TURBO PASCAL: procedure rddigi(var tsrp:TSRP; softint:integer);

C: void rddigi(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].digi
n = 0 .. Anzahl der Achsen -1

BESCHREIBUNG: Mit dieser Funktion können folgende Signalzustände abgefragt werden:

- Der aktuelle Zustand der 16 MCU-3T Digital-Eingänge
- Der aktuelle Zustand der Nullspur- (Index) Signals vom Inkrementalenkoder
- Ein zwischengespeicherter Fehler des Meßwerterfassungssystems
- Eine zwischengespeicherte Flanke des Nullspur- (Index) Signals vom Inkrementalgeber
- Eine zwischengespeicherte Flanke des Hardware-Latchsignals (Strobe)

Sofern ein Eingang aktiv ist, wird dies mit dem Wert 1 des jeweiligen Bit angezeigt. Optional können alle Digitaleingänge im TOOLSET Programm *mcfg.exe* mit Invertierung projektiert werden. Ebenso ist es möglich, bei Verwendung eines Inkrementalenkoders mit Index-Signal die gewünschte Polarität zu projektieren.

RÜCKGABEWERT: Der bitkodierte Rückgabewert befindet sich in der Struktur- bzw. Recordkomponente *digi* und hat den in nachfolgend abgedruckter Tabelle beschriebenen Aufbau.

ANMERKUNG: Es gibt keine festgelegte Achszuordnung der Digital-Eingänge.
Bit 16..19 können durch den *rdigi()*-Befehl [Kapitel 4.4.33] zurückgesetz werden. [BHB / Kapitel 4.4.3.1] und [BHB / Kapitel 4.4.3.2].

4.4.34.1 Achsenqualifizierer digi

Mit dem Register *digi* kann der Zustand der Digital-Eingänge der MCU-3T abgeprüft werden. Sofern die jeweiligen Eingänge aktiv sind, wird dies mit dem Wert 1 an der jeweiligen Bitposition angezeigt.

Tabelle 12: Bitkodierter Aufbau des digi-Wortes

Bit-Nr.	Funktion	X22/Pin
0	Eingang 1	9
1	Eingang 2	10
2	Eingang 3	11
3	Eingang 4	12
4	Eingang 5	13
5	Eingang 6	14
6	Eingang 7	15
7	Eingang 8	16
8	Eingang 9	42
9	Eingang 10	43
10	Eingang 11	44
11	Eingang 12	45
12	Eingang 13	46
13	Eingang 14 u. Hardware-Strobe-Signal zum Latchen der Istposition Achskanal 1	47
14	Eingang 15 u. Hardware-Strobe-Signal zum Latchen der Istposition Achskanal 2	48
15	Eingang 16 u. Hardware-Strobe-Signal zum Latchen der Istposition Achskanal 3	49
16	Null-Spur vom Inkrementalenkoder, achsspezifisch	--
17	Fehler des Meßwerterfassungssystems, achsspezifisch	--
18	Zwischengespeicherter Wert des Nullspursignals vom Inkrementalgeber, achsspezifisch	--
19	Zwischengespeicherter Wert des Latchsignals (Hardware-Strobe), achsspezifisch	--
20..31	Nicht belegt, diese Flags haben immer den Wert 0	--

4.4.35 rddigib, read digital input bit

TURBO PASCAL: function rddigib(an:integer; bitnr:integer; softint:integer):boolean;

C: int rddigib(int an, int bitnr, int softint);

BESCHREIBUNG: Mit dieser Funktion kann der aktuelle Zustand eines MCU-3T Digital-Eingangs und diverser anderer Logiksignale abgefragt werden. Die Achsnummer muß im Parameter *an* (0, 1, ... *REALAXIS*) spezifiziert werden.

RÜCKGABEWERT: Die Funktion liefert den Wert 1 bzw. TRUE zurück, sofern der entsprechende Eingang von *bitnr* aktiv ist.

ANMERKUNG: Bit-Nr. 17..20 können durch den *rdigi()*-Befehl [Kapitel 4.4.33] zurückgesetzt werden.
[BHB / Kapitel 4.4.3.1], [BHB / Kapitel 4.4.3.2] und PCAP-Befehl *rddigi()*

Tabelle 13: Zuordnung von *bitnr* zu den jeweiligen MCU-3T Digitaleingängen

'bitnr'	Funktion	X22/Pin
1	Eingang 1	9
2	Eingang 2	10
3	Eingang 3	11
4	Eingang 4	12
5	Eingang 5	13
6	Eingang 6	14
7	Eingang 7	15
8	Eingang 8	16
9	Eingang 9	42
10	Eingang 10	43
11	Eingang 11	44
12	Eingang 12	45
13	Eingang 13	46
14	Eingang 14	47
15	Eingang 15	48
16	Eingang 16	49
17	Null-Spur vom Inkrementalenkoder, achsspezifisch	--
18	Fehler des Meßwerterfassungssystems, achsspezifisch	--
19	Zwischengespeicherter Wert des Nullspursignals vom Inkrementalgeber, achsspezifisch	--
20	Zwischengespeicherter Wert des Latchsignals (Hardware-Strobe), achsspezifisch	--
21..32	Nicht belegt, diese Flags haben immer den Wert 0	--

4.4.36 rddigo, read digital outputs

TURBO PASCAL: procedure rddigo(var tsrp:TSRP; softint:integer);

C: void rddigo(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].digo

BESCHREIBUNG: Mit diesem Befehl wird der aktuelle Ausgabe-Status der MCU-3T-Digital-Ausgänge in die achsspezifische Struktur- bzw. Record-Komponente *digo* eingelesen. Die dort gesetzten Bits repräsentieren gesetzte Ausgänge.

RÜCKGABEWERT: Die bitkodierten Rückgabewerte befinden sich nach Ausführung dieses Befehls in der Struktur- bzw. Recordkomponente *digo*. Diese Komponente hat den beim PCAP-Befehl *wrdigo()* abgedruckten Aufbau.

4.4.37 rddigob, read digital output bit

TURBO PASCAL: function rddigob(an:integer; bitnr:integer; softint:integer):boolean;

C: int rddigob(int an, int bitnr, int softint);

BESCHREIBUNG: Mit dieser Funktion kann der aktuelle Zustand eines MCU-3T Digital-Ausgangs abgefragt werden. Die Achsnummer muß im Parameter *an* (0, 1, ... *REALAXIS*) spezifiziert werden.

RÜCKGABEWERT: Die Funktion liefert den Wert 1 bzw. TRUE zurück, sofern der entsprechende Ausgang von *bitnr* aktiv ist. Die Zuordnung von *bitnr* zu den jeweiligen Ausgängen ist beim PCAP-Befehl *wrdigob()* abgedruckt.

4.4.38 rddp, read desired position

TURBO PASCAL: procedure rddp(var tsrp:TSRP; softint:integer);

C: void rddp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].dp

BESCHREIBUNG: Der MCU-3T-Profilgenerator errechnet eine interne Führungsgröße, die sogenannte Sollposition (= gewünschte Position oder desired position). Diese kann mit diesem Befehl eingelesen werden. Im Normalfall muß in der Betriebsart Lageregelung die Ist-Position [Kapitel 4.4.60 - *rdrp()*] und diese Sollposition bis auf tolerierbare Abweichungen identisch sein.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die Sollposition im Feld *dp* zur Verfügung. Der Wert wird in der achsspezifischen Positionseinheit zurückgeliefert.

ANMERKUNG: Diese Sollposition wird unter anderem auch zur Soll-Istwert-Differenzbildung für die automatische Schleppfehlerüberwachung herangezogen.

4.4.39 rddv, read desired velocity

TURBO PASCAL: procedure rddv(var tsrp:TSRP; softint:integer);

C: void rddv(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].dv

BESCHREIBUNG: Diese Funktion liefert die achsspezifische Soll-Geschwindigkeit des MCU-3T-Profilgenerators zurück. Im Idealfall entspricht der eingelesene Wert der tatsächlichen Achsgeschwindigkeit (Ist-Geschwindigkeit).

RÜCKGABEWERT: Nach Ausführen der Funktion, steht die Sollgeschwindigkeit im Register *dv* in der achsspezifischen Geschwindigkeitseinheit zur Verfügung.

ANMERKUNG: Die Sollgeschwindigkeit kann nur durch entsprechende Verfahrbefehle beeinflußt werden.

4.4.40 rdepc, read EEPROM programming cycle

TURBO PASCAL: procedure rdepc(var tsrp:TSRP; softint:integer);

C: void rdepc(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].epc

BESCHREIBUNG: Mit dieser Funktion kann die momentane Anzahl der MCU-3T EEPROM Programmierzyklen gelesen werden. Die Zyklusnummer wird bei jedem Speichervorgrang im TOOLSET Programm *mcfg.exe* im EEPROM um eins erhöht. Das EEPROM läßt sich mindestens 10000 mal beschreiben.

RÜCKGABEWERT: Die momentane Programmier-Zyklusnummer befindet sich nach Ausführung dieses Befehls in der Struktur- bzw. Record-Komponente *epc*.

4.4.41 rdf, read filter

TURBO PASCAL: procedure rdf(var tsrp:TSRP; softint:integer);

C: void rdf(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv
n = 0 .. Anzahl vorhandener Achsen-1

BESCHREIBUNG: Mit Hilfe dieses Befehls können die aktuellen achsspezifischen PIDF-Filter-Koeffizienten der MCU-3T eingelesen werden. Die Defaultwerte dieser Koeffizienten werden mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls stehen die Rückgabewerte in den oben aufgeführten TSRP-Struktur- bzw. Record-Komponenten.

ANMERKUNG: Weitere Angaben zum PIDF-Filter sind im Kapitel 2.1.2, [BHB / Kapitel 4.1.1] und [IHB / Kapitel 5.2] enthalten.
PCAP-Befehl *uf()*

4.4.42 rdgf, read gear factor

TURBO PASCAL: procedure rdgf(var tsrp:TSRP; softint:integer);

C: void rdgf(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].gf

BESCHREIBUNG: Diese Funktion liefert den achsspezifischen Getriebe-Faktor {gf} zurück. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführen der Funktion, steht der Faktor im Register *gf* in der achsspezifischen Einheit zur Verfügung.

ANMERKUNG: Der Getriebefaktor kann mit dem PCAP-Befehl *wrgf()* jederzeit gesetzt werden.

4.4.43 rdhac, read home acceleration

TURBO PASCAL: procedure rdhac(var tsrp:TSRP; softint:integer);

C: void rdhac(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].hac

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische Referenzfahrtbeschleunigung *hac* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die Referenzfahrtbeschleunigung im Feld *hac* zur Verfügung. Der Wert wird in der achsspezifischen Beschleunigungseinheit zurückgeliefert.

ANMERKUNG: Die Referenzfahrtbeschleunigung kann unter anderem mit dem PCAP-Befehl *wrhac()* jederzeit gesetzt werden.

4.4.44 rdhvl, read home velocity

TURBO PASCAL: procedure rdhvl(var tsrp:TSRP; softint:integer);

C: void rdhvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].hvl

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische Referenzfahrtgeschwindigkeit *hvl* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die Referenzfahrtgeschwindigkeit im Feld *hvl* zur Verfügung. Der Wert wird in der achsspezifischen Geschwindigkeitseinheit zurückgeliefert.

ANMERKUNG: Die Referenzfahrtgeschwindigkeit kann unter anderem mit dem PCAP-Befehl *wrhvl()* jederzeit gesetzt werden.

4.4.45 rdifs, read interface status

TURBO PASCAL: procedure rdifs(var tsrp:TSRP; softint:integer);

C: void rdifs(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].ifs

BESCHREIBUNG: Mit diesem Register können Statusinformationen der MCU-3T eingelesen werden.

RÜCKGABEWERT: Der bitkodierte Rückgabewert befindet sich in der Struktur- bzw. Recordkomponente *ifs* und hat den in nachfolgend abgedruckter Tabelle beschriebenen Aufbau.

ANMERKUNG: [BHB / Kapitel 4.4.8.5]

4.4.45.1 Achsenqualifizierer ifs

Mit diesem Register können verschiedene Status-Informationen der MCU-3T abgefragt werden. Sofern die jeweilige Status-Information gültig ist, wird dies mit dem Wert 1 an der jeweiligen Bitposition angezeigt.

Tabelle 14: Bitkodierter Aufbau des ifs-Wortes

Bit-Nr.	Funktion
0	*edv*: Die im EEPROM abgelegten Systeminformationen und Daten sind gültig.
16	*pfe*: Das Power-Fail-Error-Flag wird immer dann auf „1“ gesetzt, wenn die Betriebsspannung auf der MCU-3T eine Schwellenspannung von 4.75V unterschreitet. Nach dem Einschalten der Baugruppe ist das Flag ebenfalls auf „1“ gesetzt.
17	*wdog*: Das Watchdog-Flag wird auf „1“ gesetzt, sofern die Watchdoglogik auf der MCU-3T angesprochen hat.
18	*iae*: Das Invalid-Access-Error-Flag wird auf „1“ gesetzt, sofern innerhalb der Betriessystemsoftware *rw_TOS* ein ungültiger Zugriff stattgefunden hat.
19..31	Nicht belegt, diese Flags haben immer den Wert 0

Anmerkungen: Die Fehlerflags 16..18 werden in einer Initialsierungsroutine der *rw_TOS*-Firmware aus einem internen Logikregister in das *ifs*-Register kopiert. Das Logikregister wird anschließend gelöscht, d.h. die Flags stehen nach einem zweiten Bootvorgang (*mcbt.exe*) nicht mehr zur Verfügung. Die Flags können auch durch den *rifs()*-Befehl [Kapitel 4.4.66] zurückgesetzt werden.

4.4.46 rdifsb, read interface status bit

TURBO PASCAL: function rdifsb(an:integer; bitnr:integer; softint:integer):boolean;

C: int rdifssb(int an, int bitnr, int softint);

BESCHREIBUNG: Mit dieser Funktion kann eine interface Statusinformation abgefragt werden. Die Achsnummer muß im Parameter *an* (0, 1, ... *REALAXIS*) spezifiziert werden.

RÜCKGABEWERT: Die Funktion liefert den Wert 1 bzw. TRUE zurück, sofern der entsprechende Eingang von *bitnr* aktiv ist.
Die Zuordnung von *bitnr* zu den jeweiligen Statusinformationen wird in Tabelle 14 beschrieben, jedoch erfolgt bei *bitnr* die Zählweise bei dem Wert 1, d.h. um beispielsweise *edv* abzufragen, muß *bitnr* den Wert 1 haben!

ANMERKUNG: [BHB / Kapitel 4.4.8.5] und PCAP-Befehl *rdifs()*

4.4.47 rdipw, read in position window

TURBO PASCAL: procedure rdipw(var tsrp:TSRP; softint:integer);

C: void rdipw(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].ipw

BESCHREIBUNG: Diese Funktion liefert das achsspezifische In-Positions-Fenster zurück.

ANMERKUNG: Nach Ausführen der Funktion steht das In-Positions-Fenster im Register *ipw* in der achsspezifischen Positionseinheit zur Verfügung.
PCAP-Befehl *wripw()*

4.4.48 rdirqpc, read interrupt request PC

TURBO PASCAL: function rdirqpc(softint:integer):integer;

C: int rdirqpc(int softint);

BESCHREIBUNG: Mit diesem Befehl kann der momentane Zustand der auf der MCU-3T generierten Interruptquelle abgefragt werden. Sofern der Interrupt aktiv ist, liefert die Funktion den Wert 1 zurück, ansonsten den Wert 0.

ANMERKUNG: Der Interrupt kann u.a. durch die Systemvariable *IRQPC* mit Hilfe eines SAP-Programms gesetzt bzw. rückgesetzt werden [Kapitel 6.3.1.1 - PC-Interrupt-Generierung].

4.4.49 rdjac, read jog accleration

TURBO PASCAL: procedure rdjac(var tsrp:TSRP; softint:integer);

C: void rdjac(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jac

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische *jog*- (auch Eilgang-) Beschleunigung *jac* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die *jog*-Beschleunigung im Feld *jac* zur Verfügung. Der Wert wird in der achsspezifischen Beschleunigungseinheit zurückgeliefert.

ANMERKUNG: Die *jog*-Beschleunigung kann mit dem PCAP-Befehl *wrjac()* jederzeit gesetzt werden.

4.4.50 rdjtvl, read jog target velocity

TURBO PASCAL: procedure rdjtvl(var tsrp:TSRP; softint:integer);

C: void rdjtvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jtvl

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische *jog*- (auch Eilgang-) Zielgeschwindigkeit *jtvl* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die *jog*-Zielgeschwindigkeit im Feld *jtvl* zur Verfügung. Der Wert wird in der achsspezifischen Geschwindigkeitseinheit zurückgeliefert.

ANMERKUNG: Die Zielgeschwindigkeit kann unter anderem mit dem PCAP-Befehl *wrjtvl()* jederzeit gesetzt werden.

4.4.51 rdjvl, read jog velocity

TURBO PASCAL: procedure rdjvl(var tsrp:TSRP; softint:integer);

C: void rdjvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jvl

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische *jog*- (auch Eilgang-) Geschwindigkeit *jvl* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die *jog*-Geschwindigkeit im Feld *jvl* zur Verfügung. Der Wert wird in der achsspezifischen Geschwindigkeitseinheit zurückgeliefert.

ANMERKUNG: Die Zielgeschwindigkeit kann unter anderem mit dem PCAP-Befehl *wrjvl()* jederzeit gesetzt werden.

4.4.52 rdledgn, read led green

TURBO PASCAL: function rdledgn(softint:integer):integer;

C: int rdledgn(int softint);

BESCHREIBUNG: Der aktuelle Zustand der LED D36 (Masterboard MCU-3T, grün) kann mit Hilfe dieser Funktion eingelesen werden.

RÜCKGABEWERT: Der Rückgabewert der Funktion ist 1, sofern die Leuchtdiode eingeschaltet ist, ansonsten 0.

ANMERKUNG: PCAP-Befehl *wrledgn()*, Systemvariable *LEDGN*

4.4.53 rdledrd, read led red

TURBO PASCAL: function rdledrd(softint:integer):integer;

C: int rdledrd(int softint);

BESCHREIBUNG: Der aktuelle Zustand der LED D34 (Masterboard MCU-3T, rot) kann mit Hilfe dieser Funktion eingelesen werden.

ANMERKUNG: PCAP-Befehl *wrledrd()*, Systemvariable *LEDRD*

4.4.54 rdledyl, read led yellow

TURBO PASCAL: function rdledyl(softint:integer):integer;

C: int rdledyl(int softint);

BESCHREIBUNG: Der aktuelle Zustand der LED D35 (Masterboard MCU-3T, gelb) kann mit Hilfe dieser Funktion eingelesen werden.

ANMERKUNG: PCAP-Befehl *wrledyl()*, Systemvariable *LEDYL*

4.4.55 rdlp, read latched position

TURBO PASCAL: procedure rdlp(var tsrp:TSRP; softint:integer);

C: void rdlp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].lp

BESCHREIBUNG: Diese Funktion liefert die achsspezifische Latch-Position zurück. Der Latch-Vorgang kann durch verschiedene Mechanismen ausgelöst werden:
1. Beim Aktivieren eines mit LP-Funktion projektierten Eingangs. Hierbei beträgt die maximale Verzögerungszeit zwei Abtastintervalle (2.56ms). Ein neuer Latch-Vorgang wird erst nach disaktivieren des Latcheingangs ermöglicht.
2. Sofern zuvor ein *lps()*-PCAP-Kommando [Kapitel 4.4.15] ausgeführt wurde und die dort im Parameter *mst* spezifizierte Verzögerung abgelaufen ist.
3. In Echtzeit (max. 1µs Verzögerung) durch vorbelegte Digitaleingänge der MCU-3T. Ein neuer Latch-Vorgang wird erst nach disaktivieren des Latcheingangs ermöglicht.
Bei allen Methoden wird die Ist-Position {*rp}* der Motorachse zwischengespeichert.

RÜCKGABEWERT: Nach Ausführen der Funktion, steht die Latchposition im Register *lp* in der achsspezifischen Positionseinheit zur Verfügung.
Die Priorität der drei Methoden ist gleichbedeutend mit der Reihenfolge der Auflistung, d.h. Echtzeit-Latchen hat die höchste Priorität.

ANMERKUNG: PCAP-Befehl *wrlp()*

4.4.56 rdlpndx, read latched position index

TURBO PASCAL: procedure rdlpndx(var tsrp:TSRP; softint:integer);

C: void rdlpndx(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].lp

BESCHREIBUNG: Diese Funktion liefert die achsspezifische Latch-Position des Indexsignals (Nullspur) zurück. Beim Akivieren der Nullspur des Inkremetalgebers wird die Ist-Position *{rp}* der Motorachse in Echtzeit zwischengespeichert.

RÜCKGABEWERT: Nach Ausführen der Funktion, steht die Latchposition im Register *lp* in der achsspezifischen Positionseinheit zur Verfügung.

ANMERKUNG: Das Latchen der Nullspur vom Inkrementalgeber ist hilfreich bei der Enkoderverifikation und bei der Referenzfahrt-Programmierung. PCAP-Befehl *wrlpndx()*

4.4.57 rdlsm, read left spool memory

TURBO PASCAL: procedure rdlsm(var tsrp:TSRP; softint:integer);

C: void rdlsm(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].lsm

BESCHREIBUNG: Dieser Befehl liefert den freien Spoolbereich in Bytes zurück. Mit Hilfe eines PCAP- oder SAP- Anwenderprogramms kann der frei verfügbare Spoolbereich jederzeit abgefragt und gegebenenfalls nachgeladen werden. Somit ist es möglich sehr große Verfahrprofile ohne Unterbrechung der Profilgenerierung nachzuladen. Das Laden des Spoolbereichs erfolgt mit *spool*-Befehlen und kann mit beiden Programmiermethoden (PCAP und SAP) durchgeführt werden. Alle *spool*-Befehle bewirken eine Abnahme des frei verfügbaren Spoolbereichs und alle aus dem Spoolbereich ausgeführten Befehle ein Anwachsen.

ANMERKUNG: Die Spoolergröße ist achsspezifisch, d.h. daß ggf. der freie Spoolbereich der einzelnen Achskanäle stark unterschiedlich sein kann. Es stehen ca. 145kByte Spoolbereich pro Achskanal zur Verfügung.

4.4.58 rdmcp, read motor command port

TURBO PASCAL: procedure rdmcp(var tsrp:TSRP; softint:integer);

C: void rdmcp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].mcp

BESCHREIBUNG: Mit diesem Befehl kann die aktuelle Stellgröße der Motor-Command-Ports eingelesen werden.

RÜCKGABEWERT: Der Rückgabewert steht nach Ausführung des Befehls im Feld *mcp* zur Verfügung.

Bei Servo-Achsen wird ein Wert im Bereich -32767 .. 32767 zurückgeliefert. Dies entspricht einer Sollwertausgangsspannung von ca. -10V .. +10V.

Bei Schrittmotorachsen handelt es sich bei diesem Wert um eine Verzögerungszeit, die für die ausgegebene Schrittfrequenz maßgebend ist. Die Verzögerungszeit kann wie folgt in die Einheit [s] umgerechnet werden:

```
tver = (mcp+1) * 2 / CLOCK;

Beispiel: mit mcp = 12499 und CLOCK = 25MHz
wird tver = 0.5ms und f = 1kHz
```

Nach jedem Ablauf der Verzögerungszeit *tver* wird das Puls-Signal umgeschaltet, d.h. nach *2*tver* wird ein Schrittsignal mit *f = 1 / (2*tver) [Hz*] ausgegeben. Der in *mcp* zurückgelieferte Wert liegt im Bereich von -1048575 .. +1048575. Das Vorzeichen bestimmt die aktuelle Drehrichtung, d.h für die Berechnung von *tver* ist nur der Betrag von *mcp* heranzuziehen. Sofern in *mcp* der Wert 0 zurückgeliefert wird, bedeutet dies, daß kein Schrittsignal ausgegeben wird, d.h. der Motor steht.

4.4.59 rdmpe, read maximum position error

TURBO PASCAL: procedure rdmpe(var tsrp:TSRP; softint:integer);

C: void rdmpe(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].mpe

BESCHREIBUNG: Diese Funktion liefert den achsspezifischen Schleppfehlergrenzwert zurück.

ANMERKUNG: Nach Ausführen der Funktion steht der maximal erlaubte Schleppfehler im Register *mpe* in der achsspezifischen Positionseinheit zur Verfügung.
PCAP-Befehl *wrmpe()*

4.4.60 rdrp, read real position

TURBO PASCAL: procedure rdrp(var tsrp:TSRP; softint:integer);

C: void rdrp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].rp

BESCHREIBUNG: Diese Funktion liefert die achsspezifische aktuelle Position (= tatsächliche Position oder real position) zurück. Das Auslesen der Position kann zu jedem beliebigen Zeitpunkt, also auch während dem Verfahren der Achse, durchgeführt werden. Pro Abtastzyklus (1.28ms) steht ein neuer Istwert zur Verfügung.

ANMERKUNG: Nach Ausführen der Funktion steht die aktuelle Position im Register *rp* in der achsspezifischen Positionseinheit zur Verfügung.

4.4.61 rdsdec, read stop deceleration

TURBO PASCAL: procedure rdsdec(var tsrp:TSRP; softint:integer);

C: void rdsdec(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].sdec

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische Stopverzögerung *sdec* eingelesen werden. Der Defaultwert wird mit Hilfe des TOOLSET Programms *mcfg.exe* festgelegt.

RÜCKGABEWERT: Nach Ausführung des Befehls steht die Stopverzögerung im Feld *sdec* zur Verfügung. Der Wert wird in der achsspezifischen Beschleunigungseinheit zurückgeliefert.

ANMERKUNG: Die Stopverzögerung kann unter anderem mit dem PCAP-Befehl *wrsdec()* jederzeit neu gesetzt werden.

4.4.62 rdsll, read software limit left

TURBO PASCAL: procedure rdsll(var tsrp:TSRP; softint:integer);

C: void rdsll(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].sll

BESCHREIBUNG: Diese Funktion liefert die achsspezifische linke Software-Endlagen-Position zurück.

ANMERKUNG: Nach Ausführen der Funktion steht die linke Software-Endlagen-Position im Register *sll* in der achsspezifischen Positionseinheit zur Verfügung.
PCAP-Befehl *wrsll()*

4.4.63 rdslr, read software limit right

TURBO PASCAL: procedure rdslr(var tsrp:TSRP; softint:integer);

C: void rdslr(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].slr

BESCHREIBUNG: Diese Funktion liefert die achsspezifische rechte Software-Endlagen-Position zurück.

ANMERKUNG: Nach Ausführen der Funktion steht die rechte Software-Endlagen-Position im Register *slr* in der achsspezifischen Positionseinheit zur Verfügung.
PCAP-Befehl *wrslr()*

4.4.64 rdtp, read target position

TURBO PASCAL: procedure rdtp(var tsrp:TSRP; softint:integer);

C: void rdtp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].tp

BESCHREIBUNG: Mit Hilfe dieser Funktion kann die Zielposition (target position) achsspezifisch abgefragt werden. Die Zielposition wird immer als absolute Weg- bzw. Winkelgröße zurückgeliefert.

ANMERKUNG: Nach Ausführen der Funktion steht die Zielposition des letzten Verfahrbefehls im Register *tp* in der achsspezifischen Positionseinheit zur Verfügung. Der Befehl dient nur zu Kontrollzwecken.

4.4.65 rdtrovr, read trajectory override

TURBO PASCAL: procedure rdtrovr(var value:double; softint:integer);

C: void rdtrovr(double *value, int softint);

BESCHREIBUNG: Dieser Befehl liest eine Zwischengröße des aktuell gesetzten Bahngeschwindigkeitskorrekturwertes, welcher bei allen Interpolationsbefehlen (*move*-Befehlen) und den entsprechend selektierten Achsen (PCAP-Befehl *utrovr()*) berücksichtigt wird.

RÜCKGABEWERT Nach Ausführung des Befehls steht der Bahngeschwindigkeitskorrekturwert in der Variablen *value*.

ANMERKUNG: PCAP-Befehle *utrvr()*, *wrtrovr()*, *wrjovr()*, *rdtrovr()* und *rdjovr()*

4.4.66 rifs, reset interface status register

TURBO PASCAL: procedure rifs(var tsrp:TSRP; softint:integer);

C: void rifs(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].ifs

BESCHREIBUNG: Mit diesem Befehl können verschiedene Fehlerflags im MCU-3T Interface-Status-Register *ifs* rückgesetzt werden. Im Detail sind dies die Fehlerbits 16, 17, 18 - *pfe*, *wdog*, und *iae*. Das Rücksetzen sollte nur in Ausnahmesituationen, z.B. in einer Fehlerüberwachungsroutine, ausgeführt werden.

ANMERKUNG: [Kapitel 4.4.45- *rdifs()*]

4.4.67 rs, reset system

TURBO PASCAL: procedure rs(softint:integer);

C: void rs(int softint);

BESCHREIBUNG: Dieser Befehl bewirkt das Rücksetzen des kompletten Achssystems. Die Digital-Ausgänge werden auf die, mit Hilfe des TOOLSET-Programms *mcfg.exe* projektierten Defaultwerte gesetzt. Auf den Sollwertkanälen werden bei Servo-Achsen 0V Ausgangsspannung und bei Schrittmotor-Achsen 0Hz Schrittfrequenz ausgegeben. Bei allen Achsen wird der Lageregelkreis geöffnet. Die Spooler-Daten werden komplett verworfen. Alle CNC-Tasks werden angehalten. Alle projektierten Softwareendlagen werden nicht mehr überwacht. Alle Overridefaktoren (PCAP-Befehl *wrjovr()* und *wrtrovr()*) werden auf den Wert 1.0 gesetzt.

ANMERKUNG: Alle Systemdaten wie Beschleunigungen, Geschwindigkeiten, Filterparameter usw. bleiben gespeichert und brauchen deshalb nicht neu geladen zu werden.
Die Statusflags des *ifs*-Registers werden durch diesen Befehl nicht beeinflußt.
Der Inhalt aller Common Integer und Double-Variablen bleibt erhalten.

4.4.68 sdels, spooler delete synchronous

TURBO PASCAL: procedure sdels(var as:AS; softint:integer);

C: void sdels(struct AS far *as, int softint);

BESCHREIBUNG: Alle im Spooler eingetragenen Kommandos werden verworfen. Der gesamte Spoolbereich steht wieder zur freien Verfügung. Das Verwerfen der Spoolerdaten erfolgt für die in *AS* spezifizierten Achsen.

ANMERKUNG: Die aktuelle Operation, wie z.B. ein Verfahrbefehl, wird fertig ausgeführt.

4.4.69 shp, set home position

TURBO PASCAL: procedure shp(var tsrp:TSRP; softint:integer);

C: void shp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].tp
n = 0 .. Anzahl der vorhandenen Achsen-1

BESCHREIBUNG: Mit Hilfe dieses Befehls kann der achsspezifische Nullpunkt (home position) gesetzt werden. Der Parameter *tp* wird in der achsspezifischen Positionseinheit angegeben. Der Befehl wird im allgemeinen nach einem Referenzsuchlauf zum Setzen des Maschinennullpunktes verwendet. Er kann in beiden Betriebsarten Regelkreis geöffnet und Regelkreis geschlossen ausgeführt werden. Um ruckartige Motorbewegungen zu verhindern, sollte er jedoch nicht während dem Verfahren des selektierten Achskanals verwendet werden.

ANMERKUNG: Bis zur ersten Ausführung dieses Befehls werden die projektierten Softwareendlagen nicht überwacht. Dies bedeutet, daß vor der Ausführung des *shp()*-Kommandos eine Referenzfahrt unter Verwendung aller *move*- und *jog*-Befehle durchgeführt werden kann. Die Software-Endlagen werden nach Ausführung des *shp()*-Kommandos bis zum nächsten *ra()* bzw. *RA()* oder *rs()* bzw. *RS*-Kommando überwacht.

4.4.70 ssms, start spooled motions synchronous

TURBO PASCAL: procedure ssms(var as:AS; softint:integer);

C: void ssms(struct AS far *as, int softint);

BESCHREIBUNG: Mit Hilfe von *spool*-Befehlen können Kommandos an die einzelnen Achskanäle der MCU-3T übertragen werden. Diese werden in einer Warteschlange eingetragen. Der PCAP-Befehl *ssms()* veranlaßt den Synchronstart für die Spoolerbefehlsabarbeitung aller in *AS* spezifizierten Achsen.

ANMERKUNG: Kapitel 2.2.8.2 - Spool-Modus

4.4.71 sstps, spooler stop synchronous

TURBO PASCAL: procedure sstps(var as:AS; softint:integer);

C: void sstps(struct AS far *as, int softint);

BESCHREIBUNG: Mit Hilfe dieses Befehls wird die Befehlsabarbeitung aus dem Spooler aller in *AS* angewählten Achskanäle unterbrochen.

ANMERKUNG: Der aktuelle Befehl wird komplett abgearbeitet.

4.4.72 startcnct, start numeric controller task

TURBO PASCAL: procedure startcnct(TaskNr:integer; softint:integer);

C: void startcnct(int TaskNr, int softint);

BESCHREIBUNG: Ein zuvor geladenes SAP-Programm kann mit diesem Befehl gestartet werden. Die in *TaskNr* (Werte 0..3) angewählte CNC-Task arbeitet das SAP-Programm vom Programmstart an ab. Das Laden kann unter anderem mit dem PCAP-Befehl *txbf()* erfolgen.

ANMERKUNG: Ein laufendes SAP-Programm wird vor Ausführung dieses Befehls automatisch gestoppt.
PCAP-Befehl *txbf()*

4.4.73 stepcnct, step numeric controller task

TURBO PASCAL: procedure stepcnct(TaskNr:integer; softint:integer);

C: void stepcnct(int TaskNr, int softint);

BESCHREIBUNG: Dieser Befehl dient zur zeilenweisen Ausführung eines SAP-Programms.

ANMERKUNG: Der PCAP-Befehl *stepcnct()* ist momentan noch nicht implementiert!

4.4.74 stopcnct, stop numeric controller task

TURBO PASCAL: procedure stopcnct(TaskNr:integer; softint:integer);

C: void stopcnct(int TaskNr, int softint);

BESCHREIBUNG: Dieser Befehl bewirkt den Programmstop des momentan ablaufenden SAP-Programms in der mit *TaskNr* (Werte 0..3) angewählten CNC-Task und versetzt diese CNC-Task in einen inaktiven Zustand. Das SAP-Programm kann unter anderem mit dem SAP-Befehl *CONTCNCT()* oder dem PCAP-Befehl *contcnct()* wieder fortgesetzt werden.

ANMERKUNG: Eventuell freigegebene EVENT-Handler im SAP-Programm werden nach Ausführen des *stopcnct()*-Befehls nicht mehr abgearbeitet. Der Antrieb sollte vor Ausführung dieses Befehls in einen sicheren Betriebszustand gebracht werden.

4.4.75 txbf, transmit binary file

TURBO PASCAL: function txbf(var filename:string; softint:integer):integer;

C: int txbf(char far *filename, int softint);

BESCHREIBUNG: Mit dieser Funktion wird die im String- bzw. Zeichen-Parameter spezifizierte Datei auf die MCU-3T übertragen. Dabei sind jedoch nur zwei spezielle Datei-Typen erlaubt. Dies sind zum einen die Systemdatei *system.dat* (bzw. Dateien mit kompatiblem Aufbau) und zum anderen die aus der IDE oder mit Hilfe des Kommandozeilen-Compilers *ncc.exe* generierten Autocode-Dateien (CNC-Files) mit den Dateierweiterungsnamen .CNC.
Das Übertragen der Systemdatei *system.dat* bewirkt folgendes:
Alle Achskanäle werden mit den achsspezifischen Systemdaten initialisiert. Die Filterkoeffizienten des PIDF-Filters werden, wie beim PCAP-Befehl *uf()*, neu berechnet. Diese Systemdaten können unter anderem im TOOLSET Programm *mcfg.exe* editiert werden. Evtl. zuvor veränderte Systemgrößen, z.B. achsspezifische Geschwindigkeiten, Beschleunigungen usw. werden durch diesen Befehl wieder überschrieben.
Achtung! Das Übertragen von CNC-Files bewirkt folgendes: Der momentane Programm-Arbeitsspeicher einer CNC-Task wird mit dem Inhalt der spezifizierten Autocode-Datei überschrieben. Deshalb wird die entsprechende Task vor dem Ladevorgang automatisch angehalten. Das CNC-File enthält unter anderem die Information, in welche Task es geladen werden muß (Task 0..3). Nachdem das CNC-File erfolgreich übertragen wurde, kann dieses mit dem PCAP-Befehl *startcnct()* oder PCAP-Befehl *STARTCNCT()* gestartet werden.

ANMERKUNG: Normalerweise ist das Laden der Systemdatei *system.dat* nur einmalig pro Systemstart notwendig. Hierzu sind auch die Angaben beim PCAP-Befehl *mcuinit()* zu beachten. Im Parameter *filename* können bei Bedarf Laufwerks- und Pfad-Namen spezifiziert werden.

RÜCKGABEWERT: Die Funktion kann folgende Werte zurückliefern:

Rückgabewert	Fehler-Beschreibung
0	kein Fehler
1	Ungültiger Funktions-Code
2	Datei nicht gefunden
3	Pfad nicht gefunden
4	Zu viele Dateien geöffnet
5	Zugriff verweigert
6	Ungültiges file handle
12	Ungültiger Zugriff
20	Datei zu groß für CNC-Task-Arbeitsspeicher
21	Ungültiger Datei-Typ! (kein SAP-File oder keine System-Datei)

4.4.76 uf, update filter

TURBO PASCAL: procedure uf(var tsrp:TSRP; softint:integer);

C: void uf(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].kp, TSRP[n].ki, TSRP[n].kd, TSRP[n].kpl, TSRP[n].kfca, TSRP[n].kfcv
n = 0 .. Anzahl vorhandener Achsen-1

BESCHREIBUNG: Mit Hilfe dieses Befehls kann das PIDF-Filter der MCU-3T achsspezifisch gesetzt werden. Bevor der Befehl ausgeführt wird, muß sichergestellt sein, daß alle oben aufgeführten Strukturkomponenten initialisiert sind. Die Ausführung dieses Befehls kann jederzeit, auch während der Profilgenerierung, ausgeführt werden. Diese Eigenschaft ermöglicht eine echtzeitgerechte Anpassung an unterschiedliche Lastverhältnisse.

ANMERKUNG: Weitere Angaben zum PIDF-Filter sind im Kapitel 2.1.2, [BHB / Kapitel 4.1.1] und [IHB / Kapitel 5.2] enthalten.
PCAP-Befehl *rdf()*

4.4.77 utrovr, update trajectroy override

TURBO PASCAL: procedure utrovr(var as:AS; softint:integer);

C: void utrovr(struct AS far *as, int softint);

BESCHREIBUNG: Für alle in *AS* angewählten Achskanäle wird der aktuell gesetzte Geschwindigkeitsoverride berücksichtigt.

ANMERKUNG: Weitere Informationen sind beim PCAP-Befehl wrtrovr() nachzulesen.

4.4.78 wrcbcnct, write common buffer CNC-Task

TURBO PASCAL: function wrcbcnct(var cbcnct:CBCNCT; softint:integer):integer;

C: int wrcbcnct(struct CBCNCT far *cbcnct, int softint);

BESCHREIBUNG: Jede CNC-Task hat einen lokalen Speicherbereich, den sogenannten Common-Buffer, der sowohl von der jeweiligen CNC-Task als auch durch ein PCAP-Programm gelesen und beschrieben werden kann.
Mit dieser Funktion kann der komplette CNC-Task-spezifische Buffer (oder nur ein Teil davon) beschrieben werden. Mit dem Funktionsparameter *cbcnct* erfolgt die Auswahl des CNC-Task-Buffers, die Anzahl zu schreibender Bytes und die Startadresse des Blocks, der an die MCU-3T übertragen werden soll.

RÜCKGABEWERT: Die Funktion *wrcbcnct()* hat folgenden bitkodierten Rückgabewert:
Bit 0: 1 wenn ungültige Task-Nummer
Bit 1: 1 wenn maximal erlaubte Buffergröße überschritten
Dies bedeutet, daß die Funktion im Normalfall den Wert 0 zurückliefert.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. Der Struktur- (Record) Aufbau von CBCNCT ist im Kapitel 4.3.2.8 abgedruckt. PCAP-Befehl *rdcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

4.4.79 wrcd, write common double

TURBO PASCAL: procedure wrcd(ndx: integer; var cdbuf:CDBUF; softint:integer);

C: void wrcd(int ndx, struct CDBUF far *cdbuf, int softint);

BESCHREIBUNG: Mit dieser Funktion können Schreibzugriffe auf die sogenannten Common-Variablen, dies sind vordefinierte System-Variablen der CNC-Task, erfolgen. Es handelt sich dabei um die *rw_SymPas*-Variablen CD0 .. CD99. Der erste Parameter gibt dabei die Nummer *ndx* der zu beschreibenden Double-Variablen an. Der Wertebereich von *ndx* ist dabei 0 bis 99. Der zweite Parameter ist ein Zeiger auf die Struktur CDBUF mit 100 double-Variablen. Vor Ausführung des Befehls muß die zu schreibende Variable mit dem entsprechend gewünschten Wert initialisiert werden.

ANMERKUNG: Der Inhalt aller Common Variablen bleibt auch nach einem Systemrücksetzvorgang, welcher z.B. durch das *rs()*-Kommando ausgeführt wird, gespeichert. Wenn dies nicht erwünscht ist, sollten die betreffenden Variablen beim Programmstart auf den gewünschten Wert gesetzt werden.

4.4.80 wrci, write common integer

TURBO PASCAL: procedure wrci(ndx: integer; var cibuf:CIBUF; softint:integer);

C: void wrci(int ndx, struct CIBUF far *cibuf, int softint);

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *wrcd()* bis auf den Unterschied daß es sich hier um die *rw_SymPas*-System-Variablen CI0 .. CI99 vom Typ LONGINT handelt.

ANMERKUNG: PCAP-Befehl *wrcd()*

4.4.81 wrdigo, write digital outputs

TURBO PASCAL: procedure wrdigo(var tsrp:TSRP; softint:integer);

C: void wrdigo(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].digo

BESCHREIBUNG: Mit diesem Register können die Digital-Ausgänge der MCU-3T gesetzt werden. Zu beachten ist, daß die Digitalausgänge auf der MCU-3T nicht achsspezifisch gruppiert sind. Sofern ein Ausgang gesetzt werden soll, wird dies durch Setzen des jeweiligen Bits erreicht. Der bitkodierte Aufbau des *digo*-Statuswortes kann folgender Tabelle entnommen werden:

Tabelle 16: Bitkodierter Aufbau des digo-Wortes

Bit-Nr.	Funktion	Stecker X22 / PIN
0	Ausgang 1	26
1	Ausgang 2	27
2	Ausgang 3	28
3	Ausgang 4	29
4	Ausgang 5	30
5	Ausgang 6	31
6	Ausgang 7	32
7	Ausgang 8	33
8..31	Nicht belegt.	--

4.4.82 wrdigob, write digital output bit

TURBO PASCAL: procedure wrdigob(an:integer; bitnr:integer; value: boolean; softint:integer);

C: wrdigob(int an, int bitnr, int value, int softint);

BESCHREIBUNG: Mit dieser Funktion kann ein MCU-3T Digital-Ausgang gesetzt bzw. rückgesetzt werden. Die Achsnummer muß im Parameter *an* (0, 1, ... *REALAXIS*) spezifiziert werden. Das Rücksetzen des Ausgangs erfolgt mit dem Wert 0 bzw. FALSE.

ANMERKUNG: PCAP-Befehl *wrdigo()*

Tabelle 17: Zuordnung von *bitnr* zu den jeweiligen MCU-3T Digitalausgängen

'bitnr'	Funktion	Stecker X22 / PIN
1	Ausgang 1	26
2	Ausgang 2	27
3	Ausgang 3	28
4	Ausgang 4	29
5	Ausgang 5	30
6	Ausgang 6	31
7	Ausgang 7	32
8	Ausgang 8	33
9..32	Nicht belegt.	--

4.4.83 wrdp, write desired position

TURBO PASCAL: procedure wrdp(var tsrp:TSRP; softint:integer);

C: void wrdp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].dp

BESCHREIBUNG: Mit diesem Befehl kann die achsspezifische Sollposition (*dp*) geschrieben werden. Dieser Befehl wird normalerweise nie benötigt und sollte nur in ganz besonderen Fällen wie z.B. beim Test oder bei der Inbetriebnahme verwendet werden. Das Verändern der Sollposition wirkt sich lediglich in der Betriebsart Lageregelung aus. Bei großen Differenzen zwischen dieser Sollposition (*dp*) und der aktuellen Position (*rp*) muß damit gerechnet werden, daß der Motor mit der maximalen Systembeschleunigung auf diese Position nachgeführt wird.

ANMERKUNG: Das Schreiben der Sollposition (*dp*) bei der Ausführung von Bewegungskommandos kann unter Umständen zu einem unkontrollierten Prozessverhalten führen und sollte deshalb vermieden werden.
PCAP-Befehl *rddp()*

4.4.84 wrgf, write gear factor

TURBO PASCAL: procedure wrgf(var tsrp:TSRP; softint:integer);

C: void wrgf(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].gf

BESCHREIBUNG: Mit diesem Befehl kann der achsspezifische Getriebe-Faktor in der entsprechenden Einheit neu gesetzt werden. Dies ist z.B. notwendig bei Schalt-Getrieben oder durch laufzeitbedingte Änderungen von Systemgrößen wie z.B. Werkstück- oder Werkzeug-Abmessungen oder anderen Korrekturfaktoren.

ANMERKUNG: Zu beachten ist, daß gerade bei großen Änderungen des Getriebe-Faktors die aktuellen achsspezifischen Beschleunigungs- und Geschwindigkeitsparameter an diesen neuen Faktor angepaßt werden müssen, da dieser zur Umrechnung dieser Systemparameter herangezogen wird.
Der aktuell gesetzte Wert von *gf* kann mit dem PCAP-Befehl *rdgf()* gelesen werden.

4.4.85 wrhac, write home acceleration

TURBO PASCAL: procedure wrhac(var tsrp:TSRP; softint:integer);

C: void wrhac(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].hac

BESCHREIBUNG: Mit diesem Befehl wird die achsspezifische Maximalbeschleunigung *hac* für alle Referenzfahrtbefehle (*home*-Befehle) gesetzt. Sofern dieser Befehl nicht zur Ausführung kommt, wird mit dem im TOOLSET-Programm *mcfg.exe* festgelegten Systemparameter gearbeitet. Der Systemparameter kann zu jedem beliebigen Zeitpunkt überschrieben werden.

ANMERKUNG: Der aktuell gesetzte Wert von *hac* kann mit dem PCAP-Befehl *rdhac()* gelesen werden.

4.4.86 wrhvl, write home velocity

TURBO PASCAL: procedure wrhvl(var tsrp:TSRP; softint:integer);

C: void wrhvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].hvl

BESCHREIBUNG: Mit diesem Befehl wird die achsspezifische Maximalgeschwindigkeit mit Hilfe der Variablen *hvl* für alle Referenzfahrtbefehle (*home*-Befehle) gesetzt. Sofern dieser Befehl nicht zur Ausführung kommt, wird mit dem im TOOLSET-Programm *mcfg.exe* festgelegten Systemparameter gearbeitet. Der Systemparameter kann zu jedem beliebigen Zeitpunkt überschrieben werden.

ANMERKUNG: Der aktuell gesetzte Wert von *hac* kann mit dem PCAP-Befehl *rdhvl()* gelesen werden.

4.4.87 wripw, write in position window

TURBO PASCAL: procedure wripw(var tsrp:TSRP; softint:integer);

C: void wripw(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].ipw

BESCHREIBUNG: Mit diesem Befehl kann das mit Hilfe des TSW-Programms *mcfg.exe* festgelegte In-Positions-Fenster {ipw} während der Laufzeit verändert werden. Das Fenster wird auf den in *ipw* gesetzten Wert neu festgelegt. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: Das In-Positions-Fenster wird nur dann überwacht, sofern ein Wert größer 0.0 spezifiziert wurde.
[BHB / Kapitel 4.4.2.12]
PCAP-Befehl *rdipw()*

4.4.88 wrjac, write jog acceleration

TURBO PASCAL: procedure wrjac(var tsrp:TSRP; softint:integer);

C: void wrjac(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jac

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *wrhac().* Jedoch wird hier die maximale Systembeschleunigung mit Hilfe der Variablen *jac* für alle *jog*-Befehle festgelegt.

ANMERKUNG: Der aktuell gesetzte Wert von *jac* kann mit dem PCAP-Befehl *rdjac()* gelesen werden.

4.4.89 wrjovr, write jog override

TURBO PASCAL: procedure wrjovr(var trsp:TSRP; softint:integer);

C: void wrjovr(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jovr

BESCHREIBUNG: Dieser Befehl setzt den achsspezifischen Geschwindigkeitskorrekturwert. Dieser Korrekturwert wird bei allen *Jog*-Befehlen berücksichtigt. Der Parameter *jovr* muß einen Wert größer 0.0 haben. Alle Werte kleiner 1.0 resultieren in einer Reduzierung der Achsgeschwindigkeit. Sofern *value* einen Wert größer 1.0 hat, äußert sich dies mit einer Erhöhung der Geschwindigkeit.

ANMERKUNG: Zu beachten ist, daß der spezifizierte Korrekturwert gleichermaßen auf die aktuelle Achsbeschleunigung wirkt. Ein zu rasches Anheben- bzw. Absenken des Korrekturwertes kann sich in einem Beschleunigungssprung (Ruck) der Achse äußern. Der Korrekturfaktor sollte deshalb über Verzögerungsschleifen linear bis zum gewünschten Endwert inkrementiert- bzw. dekrementiert werden. Bei der Ausführung der PCAP-Befehle *ra()*, *rs()* oder SAP-Befehle *RA(), RS*, wird der Override-Faktor auf den Defaultwert 1.0 initialisiert.
PCAP-Befehl *rdjovr()*

4.4.90 wrjtvl, write jog target velocity

TURBO PASCAL: procedure wrjtvl(var tsrp:TSRP; softint:integer);

C: void wrjtvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jtvl

BESCHREIBUNG: Mit diesem Befehl wird die achsspezifische jog-Zielgeschwindigkeit mit Hilfe der Variablen *jtvl* für die *jog*-Befehle *ja()* und *jr()* gesetzt. Sofern dieser Befehl nicht zur Ausführung kommt, wird mit dem im TOOLSET-Programm *mcfg.exe* festgelegten Systemparameter gearbeitet. Der Systemparameter kann zu jedem beliebigen Zeitpunkt überschrieben werden.

ANMERKUNG: Der aktuell gesetzte Wert von *jtvl* kann mit dem PCAP-Befehl *rdjtvl()* gelesen werden.

4.4.91 wrjvl, write jog velocity

TURBO PASCAL: procedure wrjvl(var tsrp:TSRP; softint:integer);

C: void wrjvl(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].jvl

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *wrhvl().* Jedoch wird hier die maximale Verfahrgeschwindigkeit mit Hilfe der Variablen *jvl* für alle *jog*-Befehle festgelegt.

ANMERKUNG: Der aktuell gesetzte Wert von *jvl* kann mit dem PCAP-Befehl *rdjvl()* gelesen werden.

4.4.92 wrledgn, write led green

TURBO PASCAL: procedure wrledgn(value:integer; softint:integer);

C: void wrledgn(int value, int softint);

BESCHREIBUNG: Mit diesem Befehl kann die grüne Leuchtdiode D36 (Masterboard MCU-3T) ein- bzw. ausgeschaltet werden. Das Einschalten erfolgt mit dem Wert 1, das Ausschalten mit dem Wert 0.

ANMERKUNG: Dieser Befehl dient vor allem als Test- und Diagnosehilfsmittel.

4.4.93 wrledrd, write led red

TURBO PASCAL: procedure wrledrd(value:integer; softint:integer);

C: void wrledrd(int value, int softint);

BESCHREIBUNG: wie PCAP-Befehl *wrledgn()*, jedoch rote LED D34

4.4.94 wrledyl, write led yellow

TURBO PASCAL: procedure wrledyl(value:integer; softint:integer);

C: void wrledyl(int value, int softint);

BESCHREIBUNG: wie PCAP-Befehl *wrledgn()*, jedoch gelbe LED D35

4.4.95 wrlp, write latched position

TURBO PASCAL: procedure wrlp(var tsrp:TSRP; softint:integer);

C: void wrlp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].lp

BESCHREIBUNG: Dieser Befehl setzt die achsspezifische Latchposition auf den in *lp* gesetzten Wert. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: PCAP-Befehl *rdlp()*

4.4.96 wrlpndx, write latched position index

TURBO PASCAL: procedure wrlpndx(var tsrp:TSRP; softint:integer);

C: void wrlpndx(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].lp

BESCHREIBUNG: Dieser Befehl setzt die achsspezifische Latchposition der Nullspur (Index) auf den in *lp* gesetzten Wert. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: PCAP-Befehl *rdlpndx()*

4.4.97 wrmcp, write motor command port

TURBO PASCAL: procedure wrmcp(var tsrp:TSRP; softint:integer);

C: void wrmcp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].mcp

BESCHREIBUNG: Dieser Befehl dient zum Beschreiben des Motor-Command-Ports auf den im Feld *mcp* gesetzten Wert. Dies ist vor allem bei der Inbetriebnahme hilfreich, wenn z.B. der Sollwertkanal des Antriebssystems überprüft werden soll. Im Idle-Mode (keine Lageregelung) kann die Motorachse mit diesem Befehl ungeregelt verfahren werden. Auf diese Art kann z.B. die Drehrichtung des Antriebes, die korrekte Arbeitsweise der Impulserfassung und Endschalter u.a. überprüft werden, bevor die Inbetriebnahme in der Betriebsart Lageregelung fortgesetzt wird.

Bei Servo-Achsen kann *mcp* auf einen Wert zwischen -32767 und +32767 gesetzt werden. Dieser Wertebereich entspricht dem Analogausgangsspannungsbereich von -10V bis +10V. Eventuell muß eine projektierte Invertierung des Analogausgangssignals berücksichtigt werden.

Bei Schrittmotorachsen kann mit *mcp* eine Zeitverzögerung spezifiziert werden, mit deren Hilfe ein Schrittsignal für Schrittmotor-Leistungsendstufen generiert wird. Die Frequenz dieses Schrittsignals kann wie folgt berechnet werden:

$$f_{Pulse} = CLOCK/2/(mcp+1)$$

Beispiel: mit mcp = 100 und CLOCK = 25MHz
wird f_{Pulse}*=12376[Hz]*

Der Wertebereich von *mcp* liegt zwischen -1048574 und +1048574. Das Vorzeichen selektiert die gewünschte Verfahrrichtung und beeinflußt das achsspezifische Richtungssignal. Für das Schrittsignal f_{Pulse} ist nur der Betrag von *mcp* maßgebend. Zu beachten ist, daß der Wert 0 von *mcp* ein Schrittsignal von 0Hz bewirkt, d.h. der Motor bleibt stehen.

ANMERKUNG: Sofern sich das Achssystem in Lageregelung befindet, wirkt sich dieser Befehl höchstens für den Zeitraum eines Abtastintervalles aus, da die Motor-Command-Ports nach der Abarbeitung des PIDF-Filters neu gesetzt werden.
PCAP-Befehl *rdmcp()*

4.4.98 wrmpe, write maximum position error

TURBO PASCAL: procedure wrmpe(var tsrp:TSRP; softint:integer);

C: void wrmpe(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].mpe

BESCHREIBUNG: Mit diesem Befehl kann die mit Hilfe des TSW-Programms *mcfg.exe* festgelegte Schleppfehlergrenze {mpe} während der Laufzeit verändert werden. Der achsspezifische maximal erlaubte Schleppfehler wird auf den in *mpe* gesetzten Wert neu festgelegt. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: Die Schleppfehlerüberwachung findet nur dann statt, wenn ein Wert größer 0.0 spezifiziert wurde und der Regelkreis geschlossen ist.
[BHB / Kapitel 4.4.2.10]
PCAP-Befehl *rdmpe()*

4.4.99 wrrp, write real position

TURBO PASCAL: procedure wrrp(var tsrp:TSRP; softint:integer);

C: void wrrp(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].rp

BESCHREIBUNG: Dieser Befehl setzt das achsspezifische aktuelle Positionsregister auf den in *rp* gesetzten Wert und ist nur im Open-Loop-Mode (keine Lageregelung) wirksam. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: Mit diesem Befehl verschiebt sich automatisch der Maschinen-Nullpunkt!

4.4.100 wrsdec, write stop deceleration

TURBO PASCAL: procedure wrsdec(var tsrp:TSRP; softint:integer);

C: void wrsdec(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].sdec

BESCHREIBUNG: Mit diesem Befehl wird die achsspezifische Stopverzögerung *sdec* für den PCAP-Befehl *js()* [Kapitel 4.4.14], SAP-Befehl *JS()* [Kapitel 6.6.25], die mit SMD-projektierten Software-Endlagen [BHB / Kapitel 4.4.2.11] und die mit LSL_SMD bzw. LSR_SMD projektierten Digital-Eingänge [BHB / Kapitel 4.4.3.1] gesetzt. Sofern *wrsdec()* nicht zur Ausführung kommt, wird mit dem im TOOLSET-Programm *mcfg.exe* festgelegten Systemparameter gearbeitet. Der Systemparameter kann zu jedem beliebigen Zeitpunkt überschrieben werden.

ANMERKUNG: Der aktuell gesetzte Wert von *sdec* kann mit dem PCAP-Befehl *rdsdec()* gelesen werden.
[BHB / Kapitel 4.4.2.11] und [BHB / Kapitel 4.4.3.1]

4.4.101 wrsll, write software limit left

TURBO PASCAL: procedure wrsll(var tsrp:TSRP; softint:integer);

C: void wrsll(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].sll

BESCHREIBUNG: Mit diesem Befehl kann die mit Hilfe des TSW-Programms *mcfg.exe* festgelegte achsspezifische linke Software-Endlagen-Position {sll} während der Laufzeit verändert werden. Die linke Software-Endlage wird auf den in *sll* gesetzten Wert neu festgelegt. Die Wertangabe erfolgt in der achsspezifischen Positionseinheit.

ANMERKUNG: Die gesetzte Software-Endlage wird nur dann berücksichtigt, wenn die Home-Position des entsprechenden Achskanals bereits definiert wurde oder nach Ausführung dieses Befehls gesetzt wird.
[BHB / Kapitel 4.4.2.11]
PCAP-Befehle *rdsll()*, *shp()*, SAP-Befehl *SHP()*

4.4.102 wrslr, write software limit right

TURBO PASCAL: procedure wrslr(var tsrp:TSRP; softint:integer);

C: void wrslr(struct TSRP far *tsrp, int softint);

TSRP-KOMPONENTEN: TSRP[n].slr

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *wrsll()*, jedoch wird die rechte Software-Endlage mit dem im Parameter *slr* gesetzten Wert neu definiert.

4.4.103 wrtrovr, write trajectory override

TURBO PASCAL: procedure wrtrovr(var value:double; softint:integer);

C: void wrtrovr(double *value, int softint);

BESCHREIBUNG: Dieser Befehl setzt den Bahngeschwindigkeitskorrekturwert für alle Interpolationsbefehle (*move*-Befehle). Der Parameter *value* muß einen Wert größer 0.0 haben. Alle Werte kleiner 1.0 resultieren in einer Reduzierung der Bahngeschwindigkeit. Sofern *value* einen Wert größer 1.0 hat, äußert sich dies mit einer Erhöhung der Bahngeschwindigkeit.
Der in *value* spezifizierte Korrekturwert wird auf der MCU-3T in einer System-Variablen zwischengespeichert und ist erst nach Ausführung des PCAP-Befehls *utrovr()*, bzw. SAP-Befehls *UTROVR()* wirksam. Die dort angewählten Achskanäle werden auch während der Bahnfahrt je nach Korrekturfaktor *value* abgebremst bzw. beschleunigt.

ANMERKUNG: Zu beachten ist, daß der spezifizierte Korrekturwert gleichermaßen auf die aktuelle Bahnbeschleunigung wirkt. Ein zu rasches Anheben- bzw. Absenken des Korrekturwertes kann sich in einem Beschleunigungssprung (Ruck) der Achsen äußern. Der Korrekturfaktor sollte deshalb über Verzögerungsschleifen linear bis zum gewünschten Endwert inkrementiert- bzw. dekrementiert werden. Bei der Ausführung des PCAP-Befehl *rs()* oder SAP-Befehl *RS*, wird der Override-Faktor auf den Defaultwert 1.0 initialisiert.
PCAP-Befehle *wrtrovr()*, *wrjovr()*, *rdtrovr()* und *rdjovr()*

4.5 Der Zugriff auf die MCU-3T über den I/O-Adreßbereich

In der MCU-3T TOOLSET Software sind Bibliotheksfunktionen für die Programmiersprache *Turbo C* und *Microsoft C* enthalten, mit denen auch ein direkter Zugriff auf die MCU-3T ermöglicht wird. In diesem Fall können Anwenderprogramme erstellt werden, welche ohne den TSR-Treiber *mcutsr.exe* auskommen. Der direkte Zugriff ist z.B. notwendig wenn ein anderes Betriebssystem als DOS, wie z.B. Unix, RMX, QNX, LINUX oder andere eingesetzt werden soll. Auch beim Einsatz von WINDOWS oder WINDOWS-NT können diese Programmdateien u.U. hilfreich sein.
Anmerkung: Die TOOLSET-Dienstprogramme wie z.B. *mcfg.exe* benötigen den Gerätetreiber *mcutsr.exe*.

4.5.1 Funktionenbibliotheken für die MCU-3T I/O-Programmierung

Auf der MCU-3T TSW-Diskette befinden sich im Verzeichnis *C\SRVR* die Funktionenbibliotheken *mcusrvr.c* und *tpulink.c* nebst den Header-Dateien *mcusrvr.h* und *tpulink.h*.

Die Funktionen und Definitionen der Dateien *mcusrvr.c* bzw. *mcusrvr.h* sind absolut identisch mit denen im Kapitel 4.1 beschriebenen Dateien *mcutsr.c* und *mcutsr.h* bis auf den Unterschied, daß hier bei allen Funktionen der Parameter *softint* entfällt.

Die Dateien *tpulink.c* und *tpulink.h* enthalten Bibliotheksfunktionen, mit denen verschiedene Datentypen und -Blöcke von und zur MCU-3T übertragen werden können. Der Quelltext ist gut dokumentiert, so daß eine individuelle Anpassung an die geforderte Hard- und Software-Umgebung keine Schwierigkeiten bereiten dürften.

Wichtig: Zu beachten sind die bei den Systemkonstanten *TPUBASEADRESS* (*tpulink.h*) *DOSCALLALLOWED* (*tpulink.h*) und *REALAXIS* (*mcusrvr.h*) angeführten Informationen!

Bei der Programmdatei move.c handelt es sich um das gleiche Beispielprogramm, wie in Kapitel 4.2 beschrieben.

4.5.2 DLL-Bibliothek für die MCU-3T I/O-Programmierung

Im Verzeichnis DLL der MCU-3T TSW-Diskette befindet sich die DLL-Bibliothek *MCUDLL.DLL*. Mit Hilfe dieser Zusatzbibliothek kann die MCU-3T-Programmierung mit den gängigen Hochsprachen zur WINDOWS-Programmierung erfolgen. Dies sind z.B. Microsoft Visual C++, Microsoft Visual Basic, Borland C++ oder Borland Delphi. Die DLL-Datei (16bit) ist einsatzfähig under den WINDOWS-Betriebssystemen 3.x und WINDOWS 95.

4.5.3 Schnittstellenbibliothek für die Programmiersprache Borland DELPHI

Im Verzeichnis DELPHI befindet sich die Unit-Datei mcusrvr.pas. Diese Funktionenreferenzbibliothek kann mit der oben beschriebenen DLL-Datei *mcudll.dll* zur Programmerstellung für die oben genannten Windows-Plattformen verwendet werden.

5 Die Programmiersprache rw_SymPas für die Standalone-Applikations-Programmierung

5.1 Einführung

rw_SymPas ist eine Programmiersprache zur Erzeugung von selbständig ablauffähigen CNC-Programmen (Stand-Alone-Applikations-Programmen) für die Positioniersteuerung MCU-3T. Die lexikalische und semantische Grammatik von *rw_SymPas* ist stark an die Programmiersprache *Pascal* angelehnt.

5.2 Lexikalische Grammatik

Dieses Kapitel enthält eine formale Definition der lexikalischen Grammatik von rw_SymPas. Diese befaßt sich mit den wortähnlichen Einheiten einer Sprache, sogenannten »Symbolen« oder »Tokens«. Die semantische Grammatik bestimmt die Regeln, nach denen Symbole zur Bildung von Ausdrücken, Anweisungen oder anderen Einheiten kombiniert werden können.
In *rw_SymPas* ergeben sich die Symbole als Folge der Operationen, die der *NCC*-Compiler mit dem Benutzerprogramm durchführt. Ein *rw_SymPas* Programm ist eine Abfolge von ASCII-Zeichen, die den Quellcode darstellen, der mit einem Texteditor (z.B. CNC-Edit) erstellt wurde. Die grundlegende Programmeinheit in *rw_SymPas* ist die Datei. Sie entspricht einer benannten DOS-Datei im Speicher oder auf der Platte und hat die Namenserweiterung .SRC.

5.2.1 Whitespace

In der lexikalischen Analysephase der Compilierung wird die Quellcodedatei in Symbole und »Whitespace« geparst (d.h. zerlegt). Whitespace ist der Sammelbegriff für Zeichen, die als Trenner gelten: Leerzeichen, Tabulatoren, Zeilenvorschübe und Kommentare. Whitespace dient zur Markierung von Beginn und Ende eines Symbols, aber abgesehen davon wird Whitespace ignoriert.

5.2.2 Kommentare

Kommentare sind Textzeilen, die Erklärungen zum Programm enthalten. Sie werden vor dem Parsen aus dem Quelltext entfernt.
Ein *rw_SymPas* Kommentar ist eine Zeichenfolge, die nach dem Zeichen { steht. Der Kommentar endet bei dem ersten Auffinden des Zeichens }, welches auf das Startsymbol { folgt. Die Verschachtelung von Kommentaren ist nicht zulässig.
Weiterhin ist es möglich einen einzeiligen Kommentar mit zwei Schrägstrichen // anzulegen. Der Kommentar kann an beliebiger Stelle beginnen und erstreckt sich bis zur nächsten Zeile.

5.2.3 Symbole

rw_SymPas kennt folgende Arten von Symbolen

Symbol:
Schlüsselwort
Bezeichner
Qualifizierte Bezeichner
Labels
Konstante
Operator
Interpunktionszeichen (auch Trennzeichen)

5.2.3.1 Schlüsselwörter

Schlüsselwörter sind für spezielle Zwecke reservierte Wörter, die nicht als normale Bezeichnernamen verwendet werden dürfen. In der folgenden Tabelle sind alle *rw_SymPas* Schlüsselwörter aufgelistet.

Tabelle 18: Alle *rw_SymPas* Schlüsselwörter

and	begin	boolean	const
do	double	downto	else
end	for	goto	if
integer	label	mod	module
not	or	procedure	repeat
shl	shr	single	then
timer	to	until	var
while	xor		

5.2.3.2 Bezeichner

Bezeichner können aus folgenden Elementen bestehen:

Bezeichner
Nicht-Ziffer
Bezeichner Nicht-Ziffer
Bezeichner Ziffer

Nicht-Ziffer: eines der folgenden Zeichen
a b c d e f g h i j k l m n o p q r s t u v w x y z _
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Ziffer: eines der folgenden Zeichen
0 1 2 3 4 5 6 7 8 9

Beispiele:

A, AA, AB, A1, A2, _A *// gültig*
1A, ?B *// ungültig*

5.2.3.2.1 Namen- und Längenbeschränkung

Bezeichner sind beliebige Namen von einer beliebigen Länge für Variablen, Prozeduren, Labelnamen, etc. Bezeichner können die Buchstaben A bis Z, a bis z, den Unterstrich und die Ziffern 0 bis 9 enthalten. Es gibt jedoch folgende Einschränkungen:

- Das erste Zeichen muß ein Buchstabe oder ein Unterstrich sein.
- Nur die ersten 32 Zeichen sind signifikant. Sofern der Bezeichner mehr als 32 Zeichen enthält, werden die restlichen Zeichen verworfen. Bei grossen *rw_SymPas* Programmen sollte man sich auf kurze Namen beschränken, um den Arbeitsspeicher des PC zu entlasten.

5.2.3.2.2 Bezeichner Groß- und Kleinschreibung

In *rw_SymPas* wird zwischen Groß- und Kleinschreibung unterschieden, so daß *Position*, *position* und *positioN* unterschiedliche Bezeichner sind.

5.2.3.2.3 Eindeutigkeit und Gültigkeit von Bezeichnern

Bezeichner können beliebige Namen sein, die den geltenden Regeln entsprechen. Es kann jedoch zu Fehlern kommen, wenn derselbe Name innerhalb desselben Geltungsbereichs für mehrere Bezeichner verwendet wird, die denselben Namensbereich haben. Gleiche Namen sind für verschiedene Namensbereiche zulässig, unabhängig vom Geltungsbereich. Die Definition des Geltungsbereichs von Bezeichnern wird im Kapitel 5.3.2.2 erläutert.

5.2.3.3 Standardbezeichner

rw_SymPas definiert bereits eine Reihe von Bezeichnern, für die deshalb der Name Standardbezeichner verwendet wird. In der folgenden Tabelle sind alle *rw_SymPas* Standardbezeichner aufgelistet.

Tabelle 19: Alle *rw_SymPas* vordefinierten Standardbezeichner

abort	cl	contcnct	disev
enev	ja	jaw	jhi
jhiw	jhl	jhlw	jhr
jhrw	jr	jrw	js
jsw	mca	mcaw	mcr
mcrw	mha	mhaw	mhr
mhrw	mla	mlaw	mlr
mlrw	ms	msw	ol
ra	rs	shp	smca
smcr	smha	smhr	smla
smlr	ssms	ssmsw	startcnct
stop	stopcnct	uf	wt
utrovr			

5.2.3.4 Achsenbezeichner

Jeder Achskanal wird mit Hilfe eines symbolischen Namens referenziert. Dieser Name kann mit bis zu 8 Zeichen vom Benutzer frei gewählt werden. Diese Achsenbezeichner werden von *rw_SymPas* ebenfalls in die Standardbezeichnerliste mitaufgenommen.
Anmerkung: Die automatische Deklaration der Achsenbezeichner weicht vom Standard-Pascal ab.

5.2.3.5 Qualifizierte Bezeichner

Der Bezug auf Bezeichner gleichen Namens, die für verschiedene Achssysteme (von *rw_SymPas*) deklariert sind, geschieht über eine Qualifizierung durch Voranstellen des Achsenbezeichners.
Beispiele:

```
A1.digo := 0;               // alle Ausgänge der MCU-3T rücksetzen
A2.digo := $FFFFFFFF;       // alle Ausgänge der MCU-3T setzen
```

Anmerkung: Der Variablenbezug auf qualifizierte Bezeichner weicht vom Standard-Pascal ab.

5.2.3.6 Labels

Für den Aufbau eines Labels gelten dieselben Regeln wie für die Bezeichner. Labels werden ausschließlich im Zusammenhang mit der Anweisung *goto* verwendet.

5.2.3.7 Konstanten

Konstanten sind Symbole, die für feste numerische Werte stehen. *rw_SymPas* kennt zwei Klassen von Konstanten: Gleitkomma und Integer. Der Datentyp einer Konstante wird vom *NCC*-Compiler auf Grund ihres numerischen Wertes und ihres Formats im Quelltext abgeleitet. Tabelle 20 zeigt die formale Definition einer Konstante.

Tabelle 20: Formale Definition einer Konstanten.

Konstante: Gleitpunktkonstante Integerkonstante
Gleitpunktkonstante: Fraktionale-Konstante<Exponent> Ziffernfolge Exponent
Fraktionale Konstante: <Ziffernfolge>.Ziffernfolge Ziffernfolge.
Exponent: e<Vorzeichen>Ziffernfolge E<Vorzeichen>Ziffernfolge
Vorzeichen: Eines der folgenden Zeichen + -
Ziffernfolge: Ziffer Ziffernfolge Ziffer
Integer-Konstante: <Vorzeichen>Dezimale Konstante Hexadezimale Konstante
Dezimale-Konstante: Ziffer Dezimale-Konstante Ziffer
Hexadezimale-Konstante: $ Hexziffer Hexadezimale-Konstante Hexziffer
Ziffer: 0 1 2 3 4 5 6 7 8 9
Hexziffer: 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

5.2.3.7.1 Integer-Konstanten

Integerkonstanten können dezimale (Basis 10) oder hexadezimale (Basis 16) Zahlen sein. Zu beachten ist, daß für dezimale und nichtdezimale Konstanten unterschiedliche Regeln gelten.

5.2.3.7.1.1 Dezimalkonstanten

Es sind Dezimalkonstanten von -2147483648 bis 2147483647 zugelassen. Konstanten ausserhalb des Bereichs werden automatisch auf den entsprechenden Minimal- bzw. Maximalwert begrenzt.

5.2.3.7.1.2 Hexadezimale Konstanten

Alle Konstanten, die mit dem Dollarzeichen ($) beginnen, werden als hexadezimale Konstante interpretiert. Hexadezimale Konstanten von $80000000 bis $7FFFFFFF sind zugelassen. Konstanten ausserhalb des Bereichs werden auf den entsprechenden Minimal- bzw. Maximalwert begrenzt.

5.2.3.7.2 Gleitpunkt-Konstanten

Eine Gleitpunktkonstante setzt sich aus 4 Bestandteilen zusammen:

- Vorkommastellen
- Dezimalpunkt
- Nachkommastellen
- e oder E und ein vorzeichenbehafteter Integerexponent (optional)

Es können entweder die Vorkomma- oder Nachkommastellen wegfallen (aber nicht beide). Der Dezimalpunkt oder der Buchstabe e (E) kann weggelassen werden (aber nicht beide). Diese Regeln ermöglichen sowohl die konventionelle als auch die wissenschaftliche Notation (mit Exponenten).

5.2.3.7.2.1 Der Typ der Gleitkommakonstanten

Gleitkommakonstanten werden grundsätzlich als Double-Werte behandelt. Sie werden in einem Doppelwort (8 Byte) nach IEE abgelegt. Der Bereich ist $1.7*10^{-308}$ bis $1.7*10^{308}$.

5.2.3.7.2.2 Deklaration von Konstanten

Eine Konstanten-Deklaration vereinbart einen Bezeichner, der innerhalb des entsprechenden Blocks für einen konstanten Wert steht. Beispiel Konstantendeklaration:

```
Const            one = 1;
```

Vorzeichenbehaftete Konstanten stehen für einen Ganzzahl- oder Gleitkommawert. Die Berechnung von Konstanten ist nicht möglich.

5.2.3.7.3 Interpunktionszeichen

Interpunktionszeichen (auch Trennzeichen genannt) sind in *rw_SymPas* wie folgt definiert:

Interpunktionszeichen: eines der folgenden Symbole
() , ; : =

5.2.3.7.3.1 Runde Klammern

() fassen Ausdrücke zusammen, isolieren konditionale Ausdrücke und repräsentieren Prozeduraufrufe und Prozedurparameter:

```
d := c * (a + b);           // Ändern der normalen Reihenfolge
if (d = z) then ...         // Erforderlich bei einer bedingten
                            // Anweisung
proc()                      // Prozeduraufruf ohne Argumente
```

5.2.3.7.3.2 Komma

Das Komma (,) trennt die Elemente in einer Prozedur-Argumentliste:

```
mlr (A1, A2);
```

5.2.3.7.3.3 Strichpunkt

Der Strichpunkt (;) dient als Endekriterium einer Anweisung. Jeder gültige *rw_SymPas* Ausdruck (auch der leere Ausdruck), an dessen Ende ein Strichpunkt steht, wird als Anweisung (Ausdruck-Anweisung) interpretiert.

5.2.3.7.3.4 Gleichheitszeichen

Das Gleichheitszeichen (=) trennt Konstantendeklarationen von den Initialisierungswerten:

```
Const one = 1.0;
```

5.3 Semantische Grammatik

In diesem Kapitel wird die formale Definition von der *rw_SymPas* Sprachstruktur erläutert. Diese semantische Grammatik bestimmt die Regeln, nach denen Symbole zur Bildung von Ausdrücken, Anweisungen oder anderen bedeutungstragenden Einheiten kombiniert werden können.

5.3.1 Deklarationen

Im folgenden Abschnitt befindet sich eine kurze Zusammenfassung über Themen, die mit Deklarationen zu tun haben: Objekte, Typen, Blöcke, Lokalität und Geltungsbereich. Lokalität und Geltungsbereich legen diejenigen Programmteile fest, von denen aus ein zulässiger Zugriff auf das mit dem Bezeichner verknüpfte Objekt möglich ist.

5.3.1.1 Objekte

Ein Objekt ist ein identifizierbarer Speicherbereich, in dem ein fester oder variabler Wert (oder eine Menge von Werten) steht. Jedes Objekt hat einen Namen und einen Typ (den sogenannten Datentyp). Der Zugriff auf das Objekt erfolgt über seinen Namen. Dieser Name kann ein einfacher Bezeichner oder ein komplexer Ausdruck sein, der eindeutig auf ein Objekt zeigt. Der Typ wird dazu verwendet um:

- die korrekte Speicherreservierung festzulegen, die zu Beginn erforderlich ist
- durch Überprüfung der Typen sicherzustellen, daß zulässige Zuweisungen erfolgen

Zu den vordefinierten Typen von *rw_SymPas* gehören der Datentyp Boolean, Integerzahlen mit Vorzeichen und Gleitkommazahlen mit unterschiedlicher Genauigkeit.
Deklarationen stellen die Verbindung zwischen Bezeichnern und Objekten her. Jede Deklaration verknüpft einen Bezeichner mit einem Datentyp. Darüber hinaus bestimmen die meisten Deklarationen, die sogenannten Definitionsdeklarationen, auch die Generierung des Objekts (wo und wann) und übernehmen die Zuweisung des Speicherplatzes.

5.3.1.2 Typen

Jede Deklaration einer Variablen muß den Typ dieser Variablen angeben. Der Typ legt den Wertebereich der Variablen fest und bestimmt die Operationen, die mit ihr ausgeführt werden können. Eine Typendefinition vereinbart also einen Bezeichner, der seinerseits für einen bestimmten Typ steht.

Typen-Deklaration:
    Bezeichner = Typ;

Typ:
    Boolean-Typ
    Ganzzahl-Typ
    Gleitpunkt-Typ

5.3.1.2.1 Boolean-Typ

Der Datentyp *Boolean* kann ausschließlich einen der vordefinierten Werte *FALSE* oder *TRUE* annehmen. Dabei gelten folgende Beziehungen:

- *FALSE* < *TRUE*
- Ordinalzahl von *FALSE* = 0
- Ordinalzahl von *TRUE* = 1

5.3.1.2.2 Ganzzahl-Typ

rw_SymPas stellt die Ganzzahl-Typen *Integer* und *Timer* zur Verfügung.

Tabelle 21: Der Ganzzahl-Typ und sein Wertebereich

Typ	Bereich	Format
Integer	-2147483648 .. 2147483647	32 Bit mit Vorzeichen
Timer	0 .. 4294967295	32 Bit ohne Vorzeichen

5.3.1.2.3 Gleitpunkt-Typen (Real-Typen)

rw_SymPas kennt zwei verschiedene Arten von Gleitpunkt-Typen: *Single* und *Double*. Die Typen unterscheiden sich voneinander sowohl durch ihren Wertebereich als auch in der Genauigkeit mit ihnen durchgeführter Operationen.
Anmerkung: Gelegentlich wird auch der Begriff Real-Typ für Gleitpunkt-Typ verwendet.

Tabelle 22: Die Gleitpunkt-Typen und ihre Genauigkeit

Typ	Bereich	Format
Single	$-1.2e^{-38}$.. $3.4e^{38}$	7 bis 8 Stellen
Double	$-2.2e^{-308}$ .. $1.8e^{308}$	15 bis 16 Stellen

5.3.1.2.4 Zuweisungskompatibilität von Typen

Die Zuweisungskompatibilität ist unabdingbar, wenn ein Wert zugewiesen werden soll. Der Wert eines Typs T_2 kann einem Wert T_1 zugewiesen werden (d.h. T_1:=T_2), wenn eine der folgenden Bedingungen erfüllt ist:

- T_1 und T_2 sind vom gleichen Typ.
- T_1 hat den Typ Double, T_2 den Wert Integer oder Single.
- T_1 hat den Typ Single, T_2 den Wert Integer.

Wenn keine dieser Bedingungen erfüllt, Zuweisungskompatibilität aber erforderlich ist, meldet der *NCC*-Compiler einen Fehler.

5.3.1.3 Variablen

5.3.1.3.1 Automatische Typ-Konvertierung

rw_SymPas führt eine automatische Typ-Konvertierung durch, sofern mit unterschiedlichen Typen in einem Ausdruck operiert wird. Die Konvertierung wird wie folgt ausgeführt: Integer nach Single oder Integer und Single nach Double. Beispiel:

```
...
Var
     i : Integer;
     s : Single;
     d : Double;
...

d := s * i;      // s und i werden automatisch nach Double konvertiert

s := i;          // i wird automatisch nach Single konvertiert
```

5.3.2 Blöcke, Lokalität und Geltungsbereich

Ein Block besteht aus beliebig angeordneten Deklarationen und Anweisungen. Jeder Block ist Teil einer Prozedur-Deklaration oder eines Programms. Alle Bezeichner und Labels im Deklarationsteil des Blocks sind in ihrer Wirkung auf diesen Block beschränkt - sie sind lokal zu diesem Block.

5.3.2.1 Syntax

Der syntaktische Aufbau jedes Blocks läßt sich wie folgt darstellen:

Block:
  Deklarationsteil
  Befehlsteil

5.3.2.1.1 Deklarationsteil

Deklarationsteil:
  Label-Deklarationsteil
  Konstanten-Deklarationsteil
  Variablen-Deklarationsteil
  Deklarationsteil Label-Deklarationsteil
  Deklarationsteil Konstanten-Deklarationsteil
  Deklarationsteil Variablen-Deklarationsteil

5.3.2.1.1.1 Label-Deklarationsteil

Im *Label-Deklarationsteil* werden alle Labels vereinbart, die *goto*-Sprungziele im Befehlsteil des betreffenden Blocks darstellen sollen. Jedes Label darf innerhalb des Befehlsteils nur einmal definiert werden (d.h. jedes *goto* muß ein eindeutiges Ziel haben).

Aufbau des Label-Deklarationsteils:

label Labels;

Labels:
LabelName
Labels, LabelName

5.3.2.1.1.2 Konstanten-Deklarationsteil

Der Deklarationsteil für Konstanten beinhaltet alle Vereinbarungen von Konstanten, die lokal zum entsprechenden Block sind.

Aufbau des Konstanten-Deklarationsteils:

const Konstanten-Deklarationen

Konstanten-Deklarationen:
Konstanten-Deklaration
Konstanten-Deklarationen Konstanten-Deklaration

5.3.2.1.1.3 Variablen-Deklarationsteil

Der Deklarationsteil für Variablen beinhaltet alle Variablen-Deklarationen, die lokal zum entsprechenden Block sind.

Aufbau des Variablen-Deklarationsteils:

var Variablen-Deklarationen

Variablen-Deklarationen:
Variablen-Deklaration
Variablen-Deklarationen Variablen-Deklaration

5.3.2.1.2 Befehlsteil

Im Befehlsteil werden alle Operationen definiert, die bei der Aktivierung des Blocks ausgeführt werden.

Befehlsteil:
Verbundanweisung

Zulässige Verbundanweisungen werden im Kapitel 5.3.5.5 erläutert.

Der Befehlsteil des Hauptprogrammblocks hat folgenden Aufbau:

```
begin
        Anweisungsfolge;
end.
```

5.3.2.2 Geltungsbereich

Jeder Bezeichner und jedes Label einer Deklaration vereinbart genau ein Objekt bzw. ein Sprungziel. Deshalb muß ein Bezeichner, genauso wie ein Label, immer im Geltungsbereich seiner Deklaration sein, wenn er im Programm erscheint. Der Geltungsbereich von Bezeichnern und Labels liegt zwischen der eigentlichen Deklaration und dem Ende des dazugehörigen Blocks, wobei alle Blöcke mit eingeschlossen sind, die dieser Block umfaßt. Allerdings gibt es hierzu einige Ausnahmen, die in den folgenden Absätzen erläutert werden.

5.3.2.2.1 Neudeklaration in einem untergeordneten Block

Mit der Annahme, daß ein Block »Aussen« einen Block »Innen« umfaßt, d.h. ihm übergeordnet ist, beschränkt jede Neudeklaration eines Bezeichners von »Aussen« im Block »Innen« den Geltungsbereich dieses Bezeichners auf den Block »Innen«. Anders formuliert: Wenn »Aussen« eine Variable *x* deklariert und »Innen« eine Variable gleichen Namens, dann können Anweisungen im Block »Innen« nicht auf die in »Aussen« deklarierte Variable *x* zugreifen.

5.3.2.2.2 Der Ort einer Deklaration im Block

Bezeichner und Labels müssen deklariert sein, bevor sie in einem Block benutzt werden können. Auf Zugriffsversuche vor ihrer eigentlichen Deklaration reagiert der *NCC*-Compiler mit der Fehlernummer 3.

5.3.2.2.3 Neudeklarationen innerhalb eines Blocks

Bezeichner und Labels können auf der obersten Ebene eines Blocks nur jeweils einmal deklariert werden, es sei denn, ihre Neudeklaration erfolgt innerhalb eines untergeordneten Blocks.

5.3.2.2.4 Standardbezeichner

rw_SymPas bietet eine ganze Reihe vordefinierter Konstanten, Typen und Prozeduren, die so arbeiten, als wären sie innerhalb eines Blocks deklariert worden, der das ganze Programm umschließt. Folglich umfaßt ihr Geltungsbereich das gesamte Programm.

5.3.3 Variablen

5.3.3.1 Die Deklaration von Variablen

Die Variablen-Deklaration enthält eine Liste von Bezeichnern, die ihrerseits für neue Variablen und ihren jeweiligen Typ stehen.

Variablendeklaration:
Bezeichnerliste: Typ;

Bezeichnerliste:
Variablenname
Variablennamen, Variablenname

Typ:
BOOLEAN
INTEGER
SINGLE
TIMER
DOUBLE

Beispiele für gültige Variablendeklarationen sind:

```
var
     on, off:    BOOLEAN;
     eins:       INTEGER;
     dvalue:     DOUBLE;
     ticks:      TIMER;
```

Wenn ein Bezeichner in der Bezeichnerliste eines Deklarationsteils steht, gilt er innerhalb des gesamten Blocks, für den er deklariert wurde. Es kann über den ganzen Block hinweg auf diese Variable Bezug genommen werden, solange nicht in einem untergeordneten Block derselbe Bezeichner für eine andere Variable verwendet wird (»Redeklaration«). Eine redeklarierte Variable benutzt den Namen eines bereits existierenden Bezeichners, stellt aber ansonsten eine eigenständige Einheit dar. Der Wert der ursprünglichen Variablen wird durch die Redeklaration nicht beeinflußt. Variablen die außerhalb vo» Prozeduren deklariert sind, werden als *global* bezeichnet. Innerhalb von Prozeduren deklarierte Variablen sind *lokal*.

5.3.3.1.1 Timer-Deklaration

Eine Eingabe mit Hilfe der vordefinierten System-Variablen *CLOCK* liefert einen Wert vom Typ *Timer*, welcher die Zeit darstellt. Dieser Wert wird von einer internen Uhr des Transputers geliefert, die in regelmäßigen Zeitabständen ihren Wert ändert. Dieser Wert läuft zyklisch weiter, d.h. nach dem größten positiven Wert wird als nächstes der kleinste negative Wert geliefert. Das Zeitintervall, in der diese interne Uhr hochgezählt wird, beträgt 64µs.
Mit Hilfe *von CLOCK* kann jederzeit der Zählerstand dieser Uhr einer *Integer* oder *Timer*-Variablen zugewiesen werden. Sofern verschiedene Zeiten miteinander zu vergleichen sind, sollte dieser Vergleich nur mit Hilfe von *Timer*-Variablen durchgeführt werden, da hier ein Timer-Überlauf beim Vergleichsoperator > automatisch berücksichtigt wird. Ebenso wird die Addition und Subtraktion mit *Timer*-Variablen in Modulo-Technik, also vorzeichenlos durchgeführt. Hingegen kann bei *Integer*-Variablen ein Über- bzw. Unterlauf stattfinden, welcher wiederum das Setzen des Transputer-Error-Flags nach sich zieht und in bestimmten Situationen zum Abbruch des *rw_TOS*-Betriebssystems führt.
Eine praktische Timer-Anwendung könnte etwa so aussehen:

```
Const
     s := 15625;            // 15625 ticks = 1s
Var
     t: timer

t := CLOCK + 5*s;           // Verzögerung von 5s ab jetzt berechnen
...
repeat
     ...
until CLOCK > t;            // warten bis 5s vorbei
...
```

In diesem Beispiel wird ersichtlich, daß die Addition von *CLOCK* und der Verzögerungszeit bei großen Werten von *CLOCK* zum Überlauf führt. Deshalb empfiehlt es sich für die Deklaration der Variable *t* den *Timer* anstatt den *Integer*-Typ zu verwenden. Ein weiterer Grund ist die Abfrage auf das Erreichen der berechneten Verzögerungszeit. Im Fall eines Überlaufs bei der Berechnung der Verzögerungszeit ist der Inhalt von *t* nämlich kleiner als *CLOCK*. Dieser Umstand wird durch die Deklaration als *Timer*-Variable ebenfalls richtig behandelt.
Der Wertebereich einer Timer-Variable liegt zwischen 0 und 4294967295. Es können Verzögerungszeiten bis zu 38h realisiert werden.

5.3.3.2 Umwandlung von Variablentypen

Der Bezug auf eine Variable eines bestimmten Typs kann in den Bezug auf eine Variable eines anderen Typs umgewandelt werden.

Typ Umwandlung:
    Typbezeichner (Variablenbezug)

Typbezeichner:
    BOOLEAN
    INTEGER
    SINGLE
    DOUBLE

Einige Beispiele für die Umwandlung von Variablen-Typen:

```
var
     B : BOOLEAN;
     I : INTEGER;
     D : DOUBLE;

     B := BOOLEAN (I);
     B := BOOLEAN (D);
     D := B;
     I := INTEGER (D);
     I := B;
```

5.3.4 Ausdrücke

Ausdrücke bestehen aus Operatoren und Operanden. Die meisten Operatoren von *rw_SymPas* verknüpfen zwei Operanden und werden deshalb als binär bezeichnet. Die restlichen Operatoren arbeiten mit nur einem Operanden, daher bezeichnet man sie als unär. Binäre Operatoren benutzen die herkömmliche algebraische Form wie z.B. *a+b*. Ein unärer Operator steht immer unmittelbar vor seinem Operanden, wie bei *-b*. Bei umfangreichen Ausdrücken regelt die in Tabelle 23 gezeigte *Rangfolge* der Operatoren die Reihenfolge der Berechnung. Es gelten drei grundlegende Regeln:

- Ein Operand zwischen zwei Operatoren von unterschiedlichem Rang ist immer an den höherrangigen Operator gebunden.
- Ein Operand zwischen gleichrangigen Operatoren ist immer an den Operator gebunden, der links von ihm steht.
- Ausdrücke in Klammern werden als einzelner Operand betrachtet und immer als erstes ausgewertet.

Tabelle 23: Rangfolge der Operatoren

Operatoren	Rangfolge	Kategorie
-, +, not	1 (am höchsten)	unär
*, /, mod, shl, shr, and	2	multiplizierend
+, -, or, xor	3	addierend
=, <>, <, >, <=, >=	4	relational

Operationen von gleichem Rang werden normalerweise von links nach rechts durchgeführt.

5.3.4.1 Syntax von Ausdrücken

Die Rangfolge der Operatoren folgt der Syntax von Ausdrücken, die sich aus Faktoren, Termen und einfachen Ausdrücken zusammensetzen. Faktoren lassen sich durch folgende Syntax darstellen:

Faktor:
- Variablenbezug
- vorzeichenlose Konstante
- (Ausdruck)
- *not* Faktor
- Typ-Umwandlung (Werte)

vorzeichenlose Konstante:
- vorzeichenloser numerischer Wert
- Zeichenfolge
- Konstanten-Bezeichner

Die folgenden Angaben stellen gültige Faktoren dar:

Dummy	Variablenbezug
15	vorzeichenlose Konstante

5.3.4.2 Operatoren

Operatoren werden in vier Gruppen unterschieden: Arithmetische, logische, boolsche und relationale Operatoren.

5.3.4.3 Arithmetische Operatoren

Folgende Tabellen zeigen die Operanden- und Ergebnistypen binärer bzw. unärer arithmetischer Operationen.

Tabelle 24: Binäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Addition	Integer, Real	Integer, Real
-	Subtraktion	Integer, Real	Integer, Real
*	Multiplikation	Integer, Real	Integer, Real
/	Division	Integer, Real	Integer, Real
mod	Modulo	Integer	Integer

Anmerkung: Sofern einer der Operanden den Typ *Timer* hat, werden Addition und Subtraktion mit Hilfe der Modulo-Technik durchgeführt. Es erfolgt keine Überlaufprüfung, da die *Timer*-Werte zyklisch sind. Weitere Angaben zum *Timer*-Typ sind im Kapitel 5.3.3.1.1 enthalten.

Tabelle 25: Unäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Identität	Integer, Real	Integer, Real
-	Negation	Integer, Real	Integer, Real

Wenn beide Operanden eines Operators +, -, *, /, oder *mod* einen Integer-Typ haben, hat das Ergebnis ebenfalls den Typ *Integer*. Ist einer der Operanden eines Operators +, -, * oder / vom Typ *Real*, dann hat das Ergebnis ebenfalls den Typ *Real*.
Der Operator *mod* liefert den Rest der Division seiner Operanden zurück, also:

```
i mod j = i - (i/j)*j;
```

5.3.4.4 Logische Operatoren

Tabelle 26 gibt die Typen der Operanden und Ergebnisse logischer Operationen wieder.

Tabelle 26: Logische Operationen

Operator	Operation	Operandentyp	Ergebnistyp
not	bitweise Negation	Integer	Integer
and	bitweises UND	Integer	Integer
or	bitweises ODER	Integer	Integer
xor	bitweise Antivalenz	Integer	Integer
shl	Linksschieben	Integer	Integer
shr	Rechtsschieben	Integer	Integer

Anmerkung: *not* ist ein unärer Operator.

Die Operationen i *shl* j und i *shr* j verschieben den Wert von *i* um *j* Bitpositionen nach links bzw. nach rechts, entsprechen also einer Multiplikation bzw. Division mit 2^j.

5.3.4.5 Boolsche Operatoren

Tabelle 27 gibt die Typen der Operanden und Ergebnisse boolscher Operationen wieder.

Tabelle 27: Boolsche Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
not	logische Negation	Boolean	Boolean
and	logisches UND	Boolean	Boolean
or	logisches ODER	Boolean	Boolean
xor	logische Antivalenz	Boolean	Boolean

Anmerkung: Der Operator **not** ist auch hier unär.

Bei Operanden des Typs *Boolean* bestimmt die normale boolsche Logik das Ergebnis dieser Operationen. Beispielsweise ergibt *a and b* nur dann *TRUE* (wahr), wenn *a* und *b* wahr sind.

5.3.4.6 Relationale Operatoren

Tabelle 28 gibt die Operandentypen und die Ergebnisse relationaler Operationen wieder.

Tabelle 28: Relationale Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
=	gleich	Integer, Real	Boolean
<>	ungleich	Integer, Real	Boolean
<	kleiner als	Integer, Real	Boolean
>	größer als	Integer, Real	Boolean
<=	kleiner gleich	Integer, Real	Boolean
>=	größer gleich	Integer, Real	Boolean

Anmerkung: Sofern einer der Operanden den Typ *Timer* hat wird die Operation *größer als* (>) mit Hilfe der Modulo-Technik durchgeführt. Es erfolgt keine Überlaufprüfung, da die *Timer*-Werte zyklisch sind. Weitere Angaben zum *Timer*-Typ sind im Kapitel 5.3.3.1.1 enthalten.

5.3.5 Anweisungen

Der engl. Begriff *statement* steht für alle Konstrukte, die eine durch die MCU-3T ausführbare Aktion vereinbaren. In diesem Hanbuch wird der Terminus »Anweisung« als Oberbegriff für Anweisungen (wie *begin, end* oder *for*) und *Befehle* (wie *goto*, Zuweisungen, Prozedur-Aufrufe usw.) verwendet.
Jeder Anweisung (d.h. jeder Vereinbarung einer ausführbaren Aktion) kann ein Label vorangestellt werden, auf das wiederum ein Bezug mit *goto* möglich ist: ein *goto* zu diesem Label bewirkt einen direkten Sprung zu dieser Anweisung und ihre Ausführung.

Aufbau einer Anweisung:

Label: Anweisung

Anweisung:
Zuweisung
Prozeduranweisung
goto-Anweisung

5.3.5.1 Zuweisungen

Zuweisungen ersetzen den momentanen Wert einer Variablen mit einem neuen Wert, der über einen Ausdruck angegeben wird.

Aufbau einer Zuweisung:

Variablenbezug **:=** Ausdruck

5.3.5.2 Prozeduraufrufe

Durch Angabe eines Prozedurbezeichners wird eine Prozedur aufgerufen, die mit diesem Bezeichner deklariert wurde. Die Übergabe von Parametern an die Prozedur wird nicht unterstützt.

5.3.5.3 Die goto-Anweisung

führt einen Sprung zum angegebenen Label aus: Das Programm wird an dem Punkt fortgesetzt, der unmittelbar auf das Label folgt. Die Syntax von *goto* ist:

goto Label

Bei Verwendung von *goto* müssen folgende Regeln beachtet werden:

- Das Label, auf das *goto* Bezug nimmt, muß im selben Block stehen wie die *goto*-Anweisung selber. Es ist nicht möglich, mit *goto* beliebig zwischen Prozeduren hin- und herzuspringen.
- Der Bezug auf einen strukturierten Anweisungsblock von einem Programmteil außerhalb dieses Blocks (d.h ein Sprung auf eine tiefere Verschachtelungsebene) kann unvorhersehbare Folgen haben. *rw_SymPas* kann derartige Fehler nicht erkennen.

5.3.5.4 Strukturierte Anweisungen

bestehen aus mehreren ineinander verschachtelten Ebenen, die ihrerseits Anweisungen enthalten. Sie werden entweder in der Reihenfolge ihres Erscheinens (Verbund-Anweisungen), bedingt (konditionale Anweisungen) oder wiederholt (Wiederholungsanweisungen bzw. Schleifen) ausgeführt.

Strukturierte Anweisung:
- Blockbefehl
- bedingte Anweisung
- Wiederholungsanweisung

5.3.5.5 Verbundanweisungen

Die Verbundanweisungen legen fest, daß die einzelnen darin enthaltenen Komponenten in der Reihenfolge ausgeführt werden, in der sie im Quelltext erscheinen. Alle im Verbund enthaltenen Anweisungen werden als einziger Block behandelt und genügen so den Forderungen an Stellen, wo die Syntax von *rw_SymPas* nur eine einzelne Anweisung zulässt. Anfang und Ende eines Verbundes werden durch *begin* und *end* gekennzeichnet, die einzelnen Komponenten sind durch Semikolons voneinander getrennt.

Die Verbundanweisung lässt sich wie folgt darstellen:

begin Anweisungsfolge **end;**

Anweisungsfolge:
Anweisung;
Anweisungsfolge Anweisung;

Beispiel:

```
// ...
var
        i: Integer;
        j: Integer;
        temp: Integer;
// ...
begin
        if (i > 0) then i := 0;
        else begin
                // j und i vertauschen
                temp := i;
                i := j;
                j := temp;
        end;
end.
```

5.3.5.6 Bedingte Anweisungen

Bedingte Anweisungen bieten eine oder mehrere Optionen und wählen eine ihrer Komponenten (ggf. auch keine) zur Anweisung aus.

5.3.5.6.1 Die if-Anweisung

lässt sich wie folgt darstellen:

if (Bedingter-Ausdruck) **then** w-Anweisung <**else** f-Anweisung>

Die Klammern um *Bedingter-Ausdruck* sind nicht unbedingt erforderlich. Das Ergebnis von *Bedingter-Ausdruck* muß den Standardtyp *Boolean* haben. Ist *Bedingter-Ausdruck* TRUE, so wird *w-Anweisung* ausgeführt; andernfalls wird *w-Anweisung* ignoriert.
Ist das optionale *else f-Anweisung* vorhanden und *Bedingter-Ausdruck* wahr, so wird *w-Anweisung* ausgeführt; andernfalls wird *w-Anweisung* ignoriert und *f-Anweisung* ausgeführt.
Die Anweisungen *f-Anweisung* und *w-Anweisung* können selbst *if-Anweisungen* sein, wodurch verschachtelte Bedingungstest in nahezu beliebiger Tiefe ermöglicht werden. Bei verschachtelten *if..else*-Konstrukten muß sehr sorgfältig vorgegangen werden - es ist darauf zu achten, daß die korrekten Anweisungen ausgewählt werden. *Else*-Mehrdeutigkeiten werden gelöst, indem ein *else* dem letzten auf derselben Schachtelungstiefe aufgetretenen *if*-ohne-*else* zugeordnet wird. Für *w-Anweisung* und *f-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7 Schleifen

Schleifen (oder Wiederholungsanweisungen) legen die wiederholte Ausführung bestimmter Programmteile fest.

Schleife:
- while-Anweisung
- repeat-Anweisung
- for-Anweisung

5.3.5.7.1 Die while-Anweisung

Das Format für eine **while**-Anweisung lautet:

while (Bedingungsausdruck) **do** w-Anweisung;

Die Klammern um *Bedingungsausdruck* sind nicht unbedingt erforderlich. Die Schleifenanweisung *w-Anweisung* wird solange ausgeführt, bis der *Bedingungsausdruck* den Wert FALSE ergibt. Der *Bedingungsausdruck* wird vorher ausgewertet und getestet. Ergibt sich ein Wert, der TRUE ist (wahr), wird *w-Anweisung* ausgeführt. Wenn das Programm auf keine Sprunganweisungen trifft, die das Verlassen der Schleife zur Folge haben, wird der *Bedingungsausdruck* erneut ausgewertet. Dieser Vorgang wiederholt sich solange, bis *Bedingungsausdruck* den Wert FALSE ergibt. Gibt es keine Sprunganweisungen, muß *w-Anweisung* den Wert von *Bedingungsausdruck* beeinflussen oder *Bedingungsausdruck* selbst muß sich während der Auswertung ändern, damit Endlosschleifen vermieden werden. Für *w-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7.2 Die repeat-Anweisung

Das Format für eine *repeat*-Anweisung lautet:

repeat r-Anweisung **until** (Bedingungsausdruck);

Die Klammern um *Bedingungsausdruck* sind nicht unbedingt erforderlich. Die Anweisung *r-Anweisung* wird ausgeführt, solange *Bedingungsausdruck* den Wert FALSE (falsch) hat. Im Gegensatz zur *while*-Anweisung wird *Bedingungsausdruck* nicht vor, sondern nach jeder Ausführung der Schleifenanweisung getestet. *r-Anweisung* wird daher mindestens einmal ausgeführt.
Für *r-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7.3 Die for-Anweisung

Das Format für eine *for*-Anweisung lautet:

for Laufvariable := Startwert **to/downto** Endwert **do** f-Anweisung;

Die Laufvariable muß der Bezeichner einer Variablen des Typs *integer* sein, die entweder innerhalb desselben Blocks wie die *for*-Anweisung lokal oder global zum gesamten Programm deklariert ist. Die Definition einer Schleife mit *for* schließt die Festlegung eines *Start*- und *Endwert*es mit ein. Beide Werte müssen ebenfals vom Typ *integer* sein, der zu dem der Laufvariablen zuweisungskompatibel ist.
Beim Start der Schleife wird die Laufvariable auf den *Startwert* gesetzt und für jeden Schleifendurchlauf um eins erhöht bzw. erniedrigt - solange, bis der *Endwert* erreicht ist. Bei jedem Durchlauf wird die im Rumpf der Schleife enthaltene *f-Anweisung* bzw. Verbundanweisung einmal ausgeführt. Wenn die Endbedingung der Schleife bereits vor dem ersten Durchlauf gegeben ist (d.h. *Endwert* < *Startwert* bzw. *Endwert* > *Startwert* bei Verwendung von *downto*), dann werden Schleife und dazugehörender Rumpf komplett übersprungen.

5.3.6 Prozeduren und Funktionen

Prozeduren und Funktionen stellen formal gesehen zusätzliche Ebenen innerhalb des Hauptprogramm-Blocks dar, also eine Verschachtelung. Eine Prozedur wird durch einen Prozeduraufruf (d.h. die Angabe eines Bezeichners) aktiviert und liefert keinen direkten Wert zurück. Eine Funktion wird bei der Berechnung eines Ausdrucks aktiviert, in der ihr Bezeichner erscheint, und hat normalerweise ein Ergebnis, das für diesen Aufruf mit dem Funktionsbezeichner gleichgesetzt werden kann.

5.3.6.1 Prozedurdeklarationen

Eine Deklaration, die mit dem reservierten Wort *procedure* eingeleitet wird, verbindet einen Bezeichner und einen Block von Anweisungen zu einer Prozedur. Derartig deklarierte Prozeduren können durch Angabe ihres Bezeichners aktiviert (d.h. aufgerufen) werden. Eine Prozedurdeklaration hat folgenden formalen Aufbau:

Prozedurkopf; Prozedurblock;

Der Prozedurkopf benennt die Prozedur (d.h. ordnet ihr einen Bezeichner zu). Nach *Pascal*-Standard wäre an dieser Stelle die Deklaration von formalen Parametern gestattet. Diese wird in *rw_SymPas* nicht unterstützt. Ein Datenaustausch zwischen dem Hauptprogramm und einer Prozedur kann jedoch über globale Variablen durchgeführt werden. Durch die Angabe ihres Bezeichners wird eine Prozedur aktiviert: die im Befehlsteil der entsprechenden Prozedurdeklaration definierten Aktionen werden ausgeführt.
Eine Prozedur, die ihre eigene Anweisung als Bestandteil ihres Befehlsteils enthält, wird rekursiv ausgeführt, d.h. sie ruft sich selber wiederholt auf. In diesem Zusammenhang muß ein geeignetes Kriterium für den Abbruch der Rekursion gefunden werden, bevor der interne CNC-Task-Stack überläuft.
Das Schachteln von Prozeduren in *rw_SymPas* ist nicht möglich.

5.3.6.2 Funktionsdeklarationen

Anmerkung: Die nach *Pascal*-Standard implementierte Funktionendeklaration ist in *rw_SymPas* nicht möglich, jedoch gibt es verschiedene vordefinierte System-Funktionen.

Diese Funktionen werden bei der Berechnung von Ausdrücken aktiviert, in denen ihr Bezeichenr erscheint, und stehen dort stellvertretend für den von ihnen zurückgelieferten Wert. Ein Funktionsbezeichner kann überall anstelle eines Operanden in einem Ausdruck eingesetzt werden, solange der Typ des Funktionsergebnisses mit dem des ersetzten Operanden kompatibel ist.
Zuweisungen an einen Funktionsbezeichner sind nicht erlaubt.
Der Aufruf einer Funktion geschieht über die Angabe ihres Bezeichners, gefolgt von einer Liste aktueller Parameter, die in Typ und Reihenfolge den formalen Parametern der enstprechend vordefinierten Funktion entsprechen müssen.

5.3.7 Die Syntax eines *rw_SymPas*-Programmes

Ein *rw_SymPas*-Programm hat eine ähnliche Form wie eine Prozedurdeklaration. Die Unterschiede liegen lediglich im Programmkopf.

rw_SymPas-Programm:

Programmkopf; Programmblock**.**

5.3.7.1 Der Programmkopf

Der Programmkopf legt den Namen eines Programms fest, hat aber keine besondere Bedeutung.

Programmkopf:
program Bezeichner

Beispiel:

```
program Test;
```

5.3.7.2 Der Programmblock

Programmblock:
Implementationsteil
Prozedur-Befehlsteil
Initialisierungsteil

Implementationsteil:
Konstanten-Deklaration
Variablen-Deklaration
Implementationsteil Konstanten-Deklaration
Implementationsteil Variablen-Deklaration

Initialisierungsteil:
begin
Befehlsteil
end

Der Initialisierungsteil ist der letzte Bestandteil eines *rw_SymPas*-Programms und stellt das Hauptprogramm dar. Er besteht aus einem mit **begin** eingeleiteten Block, der Anweisungen enthält und durch ein abschliessendes **end** beendet wird. Der gesamte Programmblock wird mit dem Zeichen **„.**“ abgeschlossen.

6 Standalone-Applikations-Programmierung

6.1 Einführung

Die Programmiersprache *rw_SymPas* ist mit einem umfangreichen Befehlssatz ausgestattet, mit dessen Hilfe eine flexible und effiziente Programmerstellung ermöglicht wird. Die Prozeduraufrufe werden bis auf wenige Ausnahmen nach *Pascal*-Konvention durchgeführt.
Da die Prozedurnamen und auch die Funktionsweise der einzelnen Prozeduren für die beiden Programmiermethoden Standalone-Applikations-Programmierung [SAP] und PC-Applikations-Programmierung [PCAP] identisch sind, erfolgt eine detaillierte Beschreibung nur bei den Befehlen der PCAP-Programmierung.
Die Auflistung der einzelnen Befehle erfolgt in alphabetischer Reihenfolge.

6.2 rw_SymPas-Beispielprogramme

Die in der MCU-3T TOOLSET Software enthaltenen *rw_SymPas* Beispielprogramme zeigen die einfache Anwendung nachfolgend beschriebener Funktionen. Die Quelltexte der Beispielprogramme sind durch Kommentare selbsterklärend. Deshalb wird an dieser Stelle auf eine detaillierte Programmbeschreibung dieser Beispiele verzichtet. Die Beispielprogramme haben alle den Dateierweiterungsnamen .SRC und sind im Unterverzeichnis SAP der MCU-3T TOOLSET Software Diskette abgelegt.

6.3 Abkürzungen, System-Parameter, Achsenspezifizierer und Achsenqualifizierer

Für die nachfolgend abgedruckte SAP-Funktionenreferenzliste werden hier zunächst die einzelnen Abkürzungen und Typen erklärt, welche zum Teil als Parameter für die verschiedenen Funktionen dienen.

6.3.1 System-Parameter

Die von der *rw_SymPas*-Programmiersprache vordefinierten Systemparameter werden tabellarisch aufgelistet und deren Funktionsweise erläutert. Zu beachten ist, daß der *NCC*-Compiler bei diesen Parametern zwischen Groß- und Kleinbuchstaben unterscheidet.

Tabelle 29: *rw_SymPas* vordefinierte System-Parameter

Name	Typ	Kurzwortbedeutung	Funktion
CI0..CI99	integer	Common Integer 0..99	100 vordefinierte Integer-Variablen zum Datenaustausch oder zur Synchronisation mit einem parallel ablaufendem PC-ApplikationsProgramm. Weitere Informationen bei den PCAP-Befehlen rdci() u. wrci().
CD0.. CD99	double	Common Double 0..99	100 vordefinierte Double-Variablen. Sonst wie CI0..CI99. Weitere Informationen bei den PCAP-Befehlen rdcd() und wrcd().
LEDGN	boolean	Led green	Grüne Leuchtdiode (D36) auf MCU-3T, eingeschaltet wenn TRUE
LEDRD	boolean	Led red	Rote LED (D34), sonst wie LEDGN
LEDYL	boolean	Led yellow	Gelbe LED (D35), sonst wie LEDGN
IRQPC	boolean	Interrupt Request PC	PC-Interrupt-Anforderung, aktiv wenn TRUE
PHI	double		Verfahrwinkel für Kreis- und Helix-Profile
PU	integer	Position Unit	Index für Positions-Einheit
TRAC	double	Trajectory Acceleration	Bahnbeschleunigung für Linear-, Kreis- und Helix-Profile
TROVR	double	Trajectory Override	Bahngeschwindigkeitskorrekturwert
TRTVL	double	Trajectory Target Velocity	Bahnzielgeschwindigkeit für Linear, Kreis- und Helix-Profile
TRVL	double	Trajectory Velocity	Bahngeschwindigkeit für Linear, Kreis- und Helix-Profile
TU	integer	Time Unit	Index für Zeit-Einheit

6.3.1.1 PC-Interrupt-Generierung

Mit Hilfe des System-Parameters *IRQPC* ist es möglich, auf dem PC einen Hardware-Interrupt auszulösen. Diese Möglichkeit gestattet eine effiziente Methode für den Einsatz der beiden Programmiermethoden PC-Applikations- und Standalone-Applikations-Programmierung. Mit Hilfe eines Stand-Alone-Programms kann ein weitgehend autarker Prozeßablauf erfolgen, der nur im Bedarfs- oder Fehlerfall das parallel ablaufende PC-Programm unterbrechen muß. Die Unterbrechung geschieht dann mit Hilfe dieser Interrupt-Generierung. Nach Erkennen des Hardware-Interupts durch das PC-Programm kann mit den oben aufgeführten Common Variablen ein Datenaustausch zwischen den beiden parallel ablaufenden Programmen erfolgen.
Anmerkung: Die Hardware-Konfiguration für die PC-Interruptgenerierung wird im Inbetriebnahme-Handbuch beschrieben. Weiterhin muß beachtet werden, daß im PC-Anwenderprogramm der selektierte Hardware-Interrupt im Interrupt-Mask-Register (I/O-Adresse 21 hex) freigegeben und ein Interrupt-Handler auf die entsprechende Unterbrechungs-Kennzahl eingerichtet wird.

6.3.1.2 Systemparameter für die Einheiten-Verarbeitung

Bei sämtlichen *move*-Befehlen der *rw_SymPas*-Programmiersprache erfolgt die Angabe der Beschleunigungs- *(TRAC)*, Geschwindigkeits- *(TRTVL, TRVL)* und Positionsparameter jeweils in angewählten Weg- und Zeiteinheiten. Mit den beiden nachfolgend aufgeführten Systemparametern ist es möglich, die Weg- (PU) und Zeit-Einheit (TU) jederzeit umzuschalten.

Tabelle 30: System-Parameter PU

Wert	Einheit	Kurzwortbedeutung
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radiant
6	counts	Counts
7	steps	Steps

Tabelle 31: System-Parameter TU

Wert	Einheit	Kurzwortbedeutung
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling Time

Anmerkung: Die Default-Werte für TU und PU werden in dem Menü [Setup][Set CNC-specific parameters] in der CNC-Editor-Umgebung festgelegt.
Die gewählten Einheiten werden nur für Interpolationsbefehle (alle *move*-Befehle) herangezogen! Sofern es sich um achsspezifische Bewegungskommandos (alle *jog*-Befehle) handelt, werden die in *mcfg.exe* spezifizierten Achs-Einheiten berücksichtigt. Hier ist keine Umschaltung während der Laufzeit möglich.

6.3.2 Achsen-Spezifizierer

Die verschiedenen Achskanäle werden mit einem symbolischen Namen referenziert. Diese Namen können im Programm *mcfg.exe* vom Anwender frei gewählt werden. In der Programmiersprache *rw_SymPas* werden diese Namen automatisch vordefiniert und dienen im Anwenderprogramm bei verschiedenen Befehlen als Parameter. Zu beachten ist, daß der *NCC*-Compiler bei den Achsen-Spezifizierern zwischen Groß- und Kleinschreibung unterscheidet.

6.3.3 Achsen-Qualifizierer

Die nachfolgend aufgeführten Systemparameter verstehen sich als Achsen-Qualifizierer und sind deshalb für alle im System vorhandenen Achskanäle und damit für alle Achsen-Spezifizierer verfügbar. Mit Hilfe dieser Parameter können verschiedene achsspezifische Daten abgefragt bzw. gesetzt werden. Zu beachten ist, daß der *NCC*-Compiler bei diesen Parametern zwischen Klein- und Großschreibung unterscheidet. Der Bezug auf einen Achsen-Qualifizierer wird durch Angabe eines Achsen-Spezifizieres, das Zeichen '.' und dem Achsen-Qualifizierer hergestellt. Dies wird mit nachfolgendem Beispiel ersichtlich:

```
...
var
     input: integer;
...
input := A2.digi;           // Digitaleingänge von Achskanal 2
                            // einlesen
...
```

Tabelle 32: Achsen-Qualifizierer

Name	Typ	Kurzwortbedeutung	Funktion
epc	integer	EEPROM programming cycles	Anzahl der Programmierzyklen
digi	integer	digital inputs	Digital-Eingänge der MCU-3T (wortweise) Verschiedene Flags dieses Registers können durch Zuweisung eines beliebigen Wertes an dieses Register gelöscht werden [Kapitel 4.4.33].
digo	integer	digital outputs	Digital-Ausgänge der MCU-3T (wortweise)
ifs	integer	interface status	Interface Status-Flags der MCU-3T (wortweise) Verschiedene Flags dieses Registers können durch Zuweisung eines beliebigen Wertes an dieses Register gelöscht werden [Kapitel 4.4.66].
axst	integer	axis status	Error-, Status- und Profil-Flags (wortweise)
dp	double	desired position	Soll-Position des Achskanals
dv	double	desired velocity	Soll-Geschwindigkeit des Achskanals
hac	double	home acceleration	Beschleunigung für home-Befehle
hvl	double	home velocity	Geschwindigkeit für home-Befehle
ipw	double	in position window	Positionsabhängiges Zielfenster
jac	double	jog acceleration	Beschleunigung für jog-Befehle
jovr	double	jog override	Geschwindigkeitsfaktor
jtvl	double	jog target velocity	Zielgeschwindigkeit für jog-Befehle
jvl	double	jog velocity	Geschwindigkeit für jog-Befehle
kd	double		PIDF-Filter-Koeffizient für Differentation
ki	double		PIDF-Filter-Koeffizient für Integration
kp	double		PIDF-Filter-Koeffizient für Verstärkung
kpl	double		PIDF-Filter-Koeffizient zur zus. Phasen-Voreilung
kfca	double		PIDF-Filter-Koeffizient zur Vorwärtskompensation für Beschleunigung
kfcv	double		PIDF-Filter-Koeffizient zur Vorwärtskompensation für Geschwindigkeit
lp	double	latched position	gelatchter Positionswert
lpndx	double	latched position index	gelatchter Positionswert mit Index-Signal (Nullspur)
lsm	integer	left spool memory	freier Spoolbereich [Bytes]
mcp	integer	Motor Command Port	Servo-Motoren: Sollwert für Analog-Port, Wertebereich -2047 .. +2047 (-10V .. +10V) Stepper-Motoren: Schrittsignal für Schrittmotor-Leistungsendstufen, Wertebereich -32767.. +32767
mpe	double	maximum position error	maximal erlaubter Schleppfehler
rp	double	real position	Ist-Position des Achskanals
sdec	double	stop deceleration	Stopverzögerung des Achskanals
sll	double	software limit left	linke Software-Endlage
slr	double	software limit right	rechte Software-Endlage
tp	double	desired position	Ziel-Position des Achskanals

Die Funktion dieser Qualifizierer kann bei den entsprechenden *rdxxxx()*- bzw *wrxxxx()*-Befehlen in der Funktionenreferenzliste der PCAP-Programmierung nachgelesen werden. Beispielsweise wird die Bedeutung des Qualifizierers *digo* beim Befehl *wrdigo()* erklärt.

Ausnahme: Die PIDF-Filterkoeffizienten werden gemeinsam mit dem SAP-Befehl *UF()* wirksam. Das Lesen bzw. Beschreiben dieser Koeffizienten erfolgt auf PCAP-Ebene mit den Befehlen *rdf()* und *uf()*.

6.3.4 Strukturierte Achsen-Qualifizierer

Die nachfolgend aufgeführten Systemparameter verstehen sich als strukturierte Achsen-Qualifizierer und sind deshalb für alle im System vorhandenen Achskanäle und damit für alle Achsen-Spezifizierer verfügbar. Mit Hilfe dieser Parameter können verschiedene achsspezifische Daten bitweise abgefragt bzw. gesetzt werden. Zu beachten ist, daß der *NCC*-Compiler bei diesen Parametern zwischen Klein- und Großschreibung unterscheidet. Der Bezug auf einen strukturierten Achsen-Qualifizierer wird aus folgendem Beispiel ersichtlich:

```
...
const
     enable = 1;
var
     input: boolean;
...
input := A2.digib.enable;    // Digitaleingang 1 von Achskanal 2 (I1)
                             // einlesen
A1.digob.7 := TRUE;          // Digitalausgang 7 (O7) setzen
...
```

Tabelle 33: Strukturierte Achsen-Qualifizierer

Name	Typ	Kurzwortbedeutung	Funktion
digib	boolean	digital-input-bit	Digital-Eingänge der MCU-3T (bitweise)
digob	boolean	digital-output-bit	Digital-Ausgänge der MCU-3T (bitweise)
ifsb	boolean	interface-status-bit	Status-Flags der MCU-3T (bitweise)
axstb	boolean	axis status-bit	Error-, Status- und Profil-Flags (bitweise)

Die Funktion dieser Qualifizierer kann bei den entsprechenden *rdxxxxb()*- bzw *wrxxxxb()*-Befehlen in der Funktionenreferenzliste der PCAP-Programmierung nachgelesen werden. Beispielsweise wird die Bedeutung des Qualifizierers *digib* beim Befehl *rddigib()* erklärt.
Anmerkung: Die Bit-Zählweise bei den strukturierten Achsenqualifizieren beginnt bei 1!

6.3.5 Abkürzungen

Vorab werden einige in der Funktionen-Referenzliste verwendete Abkürzungen erläutert:

Tabelle 34: Abkürzungen.

A1	Symbolischer Name für den ersten Achskanal. Dieser Name kann in mcfg.exe frei gewählt werden. Wird hauptsächlich bei Beispielen verwendet.
A2	Symbolischer Name für den zweiten Achskanal. Sonst wie A1.
Spec	Achsen-Spezifizierer wie z.B. A1 oder A2
Qual	Achsen-Qualifizierer wie z.B. digi, digib, digo, digob, axst usw.
Pos	Positionssollwert (Datentyp double)
Event	Prozedur mit Funktion als Eventhandler

6.4 Reservierte Prozedur-Namen mit Event-Funktion

In *rw_SymPas* sind eine Reihe von Prozedur-Namen mit Ereignis-Funktion vordefiniert. Sofern Prozedurdefinitionen mit diesen Prozedur-Namen im Anwenderprogramm erfolgen, kann die CNC-Task mit einem Freigabebefehl dazu veranlasst werden, diese Prozeduren beim Eintreffen eines prozedurspezifischen Ereignisses (Event) automatisch aufzurufen. Diese Prozeduren werden deshalb auch als Event-Handler bezeichnet.
Anmerkung: Die Events werden nach jeder Ausführung einer rw_SymPas-Anweisung abgeprüft.

6.4.1 Event-Prozedur EVEO

Die Definition der Prozedur EVEO und Freigabe des entsprechenden Ereignisses bewirkt den automatischen Aufruf dieser Prozedur. Das Ereignis EO (Emergency Out. Noto-Aus) tritt dann auf, wenn ein mit EO-Funktion projektierter Digital-Eingang aktiviert wird [BHB / Kapitel 4.4.3.1]. Sofern mehrere EO-Eingänge im System vorhanden sind, kann mit dem Statusregister *axst* geprüft werden, welcher EO-Eingang den Fehler verursacht. Nachfolgend wird ein einfaches Beispielprogramm für die Implementierung eines EO-Handlers aufgelistet:

```
...
procedure EVEO;     // vordefinierter Name für
                    // Timeout-EVENT-Handler
begin
     CI0 := 999;    // Common Variable
                    // signalisiert Pogrammabbruch
     abort;         // Benutzerprogramm abbrechen
end;

...
begin
     ...
     CI0 := 0;      // Common Variable
                    // löschen
     enev(EVEO);    // Timeout-Handler freigeben
     ...
end.
```

6.4.2 Event-Prozedur EVDNR

Die Funktionsweise der Event-Prozedur EVDNR ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Drive Not Ready (Antrieb nicht bereit) automatisch abgearbeitet. Das Ereignis DNR tritt dann auf, wenn ein mit DR-Funktion projektierter Digital-Eingang inaktiv wird [BHB / Kapitel 4.4.3.1].

6.4.3 Event-Prozedur EVLSH

Die Funktionsweise der Event-Prozedur EVLSH ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Limit Switch Hardware (Hardware-Endschalter) automatisch abgearbeitet. Das Ereignis LSH tritt dann auf, wenn ein mit LSL_TOM, LSL_SMA, LSR_TOM oder LSR_SMA-Funktion projektierter Digital-Eingang aktiviert wird [BHB / Kapitel 4.4.3.1].

6.4.4 Event-Prozedur EVLSS

Die Funktionsweise der Event-Prozedur EVLSS ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Limit Switch Software (Software-Endlage) automatisch abgearbeitet. Das Ereignis LSS tritt dann auf, wenn die aktuelle Position eines Achssystems einen im TOOLSET-Programm *mcfg.exe* spezifizierten Grenzwert überschreitet und der entsprechende Grenzwert mit der Funktion TOM oder SMA projektiert wurde [BHB / Kapitel 4.4.3.1].

6.4.5 Event-Prozedur EVMPE

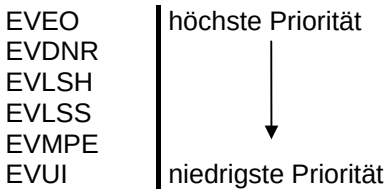
Die Funktionsweise der Event-Prozedur EVMPE ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Maximum Position Error (Maximaler Schleppfehler) automatisch abgearbeitet. Das Ereignis MPE tritt dann auf, wenn der Regelkreis geschlossen ist und die Differenz von Soll- und Ist-Position eines Achssystems den im TOOLSET-Programm *mcfg.exe* spezifizierten Grenzwert überschreitet [BHB / Kapitel 4.4.2.10].

6.4.6 Event-Prozedur EVUI

Die Funktionsweise der Event-Prozedur EVUI ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses User Input automatisch abgearbeitet. Das Ereignis UI tritt dann auf, wenn ein mit UI projektierter Digital-Eingang aktiviert wird [BHB / Kapitel 4.4.3.1]. Der Benutzer hat die Möglichkeit mit UI-projektierten Digital-Eingängen benutzerspezifische Spezial-Funktionen im SAP-Programm aufzubauen. Das alternative zyklische Abfragen (Polling) kann entfallen.

6.4.7 Priorität und Abarbeitungsreihenfolge der Event-Prozeduren

Es ist möglich, daß verschiedene Ereignisse zum gleichen Zeitpunkt eintreffen. In diesem Fall ist die Priorität wie folgt gegeben:



Sofern momentan eine Event-Prozedur (Event 1) in Bearbeitung ist, wird das Eintreffen eines anderen Events (Event 2) mit niederer oder höherer Priorität ignoriert und erst nach Abarbeitung des momentanen Eventhandlers (Event 1) durchgeführt. Event 2 muß dann aber noch aktiv sein!
Anmerkung: Nach den *STOP* und *ABORT* SAP-Befehlen und während der Ausführung des *WT()* SAP-Befehls werden keine Event-Handler abgearbeitet.

6.5 SAP-Satz-Befehle

In nachfolgend aufgelisteter Befehls-Referenzliste sind eine Reihe von Befehlen enthalten, mit denen ein Programmaufbau mit Satzstruktur erzielt werden kann. Dies sind alle Befehle, deren Befehlsname mit dem Zeichen 'W' endet. Dazu gehören beispielsweise die SAP-Befehle *MLAW()*, *JAW()* oder *SSMSW().* Diese Befehle warten automatisch das Profilende aller beteiligten Achsen ab, d.h. die nächste Anweisung wird erst beim Erreichen der Zielpositionen der angewählten Achsen abgearbeitet. Dazu prüft die CNC-Task zyklisch die Profil-Ende-Flags dieser Achsen ab und setzt das Programm ggf. bei der nächsten Anweisung fort. Bei diesem Abprüfen werden auch die oben freigegebenen EVENT-Handler berücksichtigt und im Bedarfsfall automatisch abgearbeitet.
Anmerkung: Eine andere Möglichkeit der Profilende-Abprüfung ist das Auswerten des *axst*-Achsenqualifizierers.

6.6 SAP-Befehls-Referenzliste *rw_SymPas*

6.6.1 Aufbau der Referenzliste

Die Referenzliste ist wie folgt aufgebaut:

FUNKTIONSNAME: Dies ist der Name, mit dessen Hilfe die nachfolgend beschriebene Funktion aufgerufen wird.

KURZWORTBEDEUTUNG: Hier steht die ausführliche Beschreibung des entsprechenden Funktionsnamens.

FUNKTIONSPARAMETER: Sofern die Funktion eine Parameterübergabe verlangt, werden diese hier aufgeführt.

SYSTEMPARAMETER: Verschiedene Funktionen werden unter Berücksichtigung verschiedener Systemparameter ausgeführt. Diese werden hier aufgeführt.

SIMULTANFUNKTION: Bei verschiedenen Funktion ist es gestattet, eine oder mehrere Achsen zu spezifizieren, für welche die entsprechende Funktion auszuführen ist.

REFERENZEN: Verweise auf andere Funktionen und Kapitel.

DEKLARATION: Die formale Deklaration von vordefinierten Systemfunktionen, benutzerdefinierte Elemente sind kursiv gesetzt.

ERGEBNISTYP: Der Typ des zurückgelieferten Wertes (nur bei Systemfunktionen).

BESCHREIBUNG: Klartextbeschreibung des Befehls.

ANMERKUNG: Immer wiederkehrende Anmerkungen und Erläuterungen verweisen hier auf die entsprechenden Kapitel.

BEISPIEL: Ein Beispiel für die entsprechende Funktion.

6.6.2 ABORT, abort

BESCHREIBUNG: Dieser Befehl bewirkt das Abbrechen eines laufenden SAP-Programmes. Im Gegensatz zur *STOP*-Anweisung kann das Programm nicht mit dem PCAP-Befehl *contcnct()* bzw. SAP-Befehl *CONTCNCT()* fortgesetzt werden. Dies ist nur durch den PCAP-Befehl *startcnct()* bzw. PCAP-Befehl *STARTCNCT()* möglich.

ANMERKUNG: Nach Ausführen des Befehls werden die freigegebenen EVENT-Handler-Prozeduren nicht mehr abgearbeitet.

BEISPIEL: *ABORT;*

6.6.3 ABS, absolute function

DEKLARATION: abs(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den absoluten Wert von *value* zurück.

BEISPIEL:
```
...
var
        d1, d2: double;
...
d1 := -5.0;
d2 := ABS(d1);          // d2 := 5.0
```

6.6.4 ACOS, arc cosine function

DEKLARATION: acos(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Arcuscosinus von *value* zurück. Das Argument *Value* muß im Bereich [-1..+1] liegen. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [0..pi].

6.6.5 ASIN, arc sine function

DEKLARATION: asin(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Arcussinus von *value* zurück. Das Argument *Value* muß im Bereich [-1..+1] liegen. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [-pi/2..+pi/2].

6.6.6 ATAN, arc tangent function

DEKLARATION: atan(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Arcustangens von *value* zurück. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [-pi/2..+pi/2].

6.6.7 AZO, activate zero offsets

FUNKTIONSPARAMETER: Integer-Konstante im Wertebereich 0..4

BESCHREIBUNG: PCAP-Befehl *azo()*

BEISPIEL:
```
const Offsets1 = 1;

azo(Offsets1);// Nullpunktverschiebungen Satz 1 aktivieren
```

6.6.8 CL, close loop

FUNKTIONSPARAMETER: *Spec*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *cl()* [Kapitel 4.4.4]

BEISPIEL: *CL(A1, A2); // Achskanäle 1 und 2 in Lageregelung bringen*

6.6.9 CONTCNCT, continue CNC-Task

FUNKTIONSPARAMETER: Integer-Konstante im Bereich 0..3

BESCHREIBUNG: Dieser Befehl setzt die im Parameter übergebene CNC-Task fort.

ANMERKUNG: Der Befehl kann benutzt werden, um ein gestopptes SAP-Programm fortzusetzen. Ein SAP-Programm welches mit dem SAP-Befehl *ABORT* angehalten wurde, kann nur mit dem SAP-Befehl *STARTCNCT()* oder PCAP-Befehl *startcnct()* neu gestartet, also nicht fortgesetzt, werden. Das selbsttätige Fortsetzen einer gestoppten Task ist ebenfalls nicht möglich.

BEISPIEL:
```
...
const
        TASK0 = 0;
...
CONTCNCT(TASK0);        // Task 0 fortsetzen
CONTCNCT(1);            // Task 1 fortsetzen
```

6.6.10 COS, cosine function

DEKLARATION: cos(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Cosinus von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360 Grad) interpretiert.

ANMERKUNG *Sin()*, *Tan()*-Funktion

BEISPIEL:
```
...
var
        d1, d2: double;
...
d1 := 3.1415;
d2 := COS(d1);          // d2 := -1.0 (gerundet)
```

6.6.11 COSH, hyperbolic cosine function

DEKLARATION: cos(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den hyperbolischen Cosinus von *value* zurück.

6.6.12 DISEV, disable event

FUNKTIONSPARAMETER: *Event*

REFERENZEN: Kapitel 6.4 und SAP-Befehl *ENEV()*

BESCHREIBUNG: sperrt den spezifizierten Event-Handler

BEISPIEL: *DISEV(EVEO); // Emergency Out Handler ignorieren*

6.6.13 ENEV, enable event

FUNKTIONSPARAMETER: *Event*

REFERENZEN: Kapitel 6.4 und SAP-Befehl *DISEV()*

BESCHREIBUNG: gibt den spezifizierten Event-Handler frei

BEISPIEL: *ENEV(EVEO); // Emergency Out Handler freigeben*

6.6.14 EXP, exponential function

DEKLARATION: exp(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Wert e^{value} zurück, wobei *e* die Basis des natürlichen Logarithmus ist (2.718281...).

ANMERKUNG: Funktion *Ln()*

6.6.15 JA, jog absolute

FUNKTIONSPARAMETER: *Spec* und *Pos*

SYSTEMPARAMETER: Qualifizierer: *jac, jvl* und *jtvl*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *ja()*, SAP-Befehl *JAW()*

BESCHREIBUNG: Der oder die ausgewählten Achskanäle werden auf die angegebenen Positionssollwerte absolut verfahren. Dazu wird der Motor mit der achsspezifischen Beschleunigung *jac* auf die Geschwindigkeit *jvl* hochbeschleunigt und auf die spezifizierte Zielposition *Pos* verfahren. Zusätzlich kann mit dem Parameter *jtl* eine Zielgeschwindigkeit spezifiziert werden. Die Angabe der Bahnparameter erfolgt in den achsenspezifischen Einheiten.

ANMERKUNG: PCAP-Befehl *ja()*

BEISPIEL: *JA(A1:=100.0); // Achse 1 absolut auf Position 100 verfahren*
JA(A1:=100.0, A2:=100.0);

6.6.16 JAW, jog absolute waiting

FUNKTIONSPARAMETER: *Spec, Pos*

SYSTEMPARAMETER: Qualifizierer: *jac, jvl* und *jtvl*

SIMULTANFUNKTION: ja

REFERENZEN: *JA*

BESCHREIBUNG: Dieser Befehl ist mit dem SAP-Befehl *JA()* und PCAP-Befehl *ja()* identisch, wartet jedoch zusätzlich das Profilende aller beteiligten Achsen ab. Durch den Einsatz dieses Befehls erhält das SAP-Programm eine satzähnliche Gestalt wie sie bei den handelsüblichen CNC-Steuerungen zu finden ist.

ANMERKUNG: Der Benutzer sollte durch den Einsatz von EVENT-Handlern sicherstellen, daß der Antrieb auch in Ausnahmesituationen korrekt bedient wird, da das CNC-Programm gerade bei Positioniervorgängen mit großem Zeitbedarf entsprechend lange bei diesem Befehl verweilt.

BEISPIEL:
```
JAW(A2:=-1000.0);          // Achse 1 absolut auf Position -1000.0 verfahren
                           // und warten bis das Profilende erreicht wird
JAW(A1:=1e3, A2 := 1.3e4);
```

6.6.17 JHI, jog home index

FUNKTIONSPARAMETER: *Spec, Pos*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *jhi()*, SAP-Befehl *JHIW()*

BESCHREIBUNG: Der Referenzsuchlauf auf die Nullspur (Index) des Drehgebers oder des Linearmaßstabes aller selektierten Achskanäle wird gestartet. Der Suchlauf wird abgebrochen, sobald der in *Pos* spezifizierte Verfahrweg bzw. Winkel überschritten wird.

BEISPIEL:
```
JHI(A1 := 1.0, A2 := 1.5);  // Referenzsuchlauf von Achse 1 und 2
                            // starten.
```

6.6.18 JHIW, jog home index waiting

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *jhi()* und SAP-Befehl *JHI()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.

ANMERKUNG: SAP-Befehl *JA()*

BEISPIEL:
```
JHIW(A1 := 5.0);
```

6.6.19 JHL, jog home left

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *jhl()*, SAP-Befehl *JHLW()*

BESCHREIBUNG: Der Referenzsuchlauf auf einen mit REF projektierten Digitaleingang aller selektierten Achskanäle wird in die linke Verfahrrichtung gestartet.

BEISPIEL: *JHL(A1);*

6.6.20 JHLW, jog home left waiting

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *jhl()*, SAP-Befehl *JHL()*

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *jhl()* und SAP-Befehl *JHL()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.

ANMERKUNG: SAP-Befehl *JA()*

BEISPIEL: *JHLW(A2);*

6.6.21 JHR, jog home right

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *jhr()*, SAP-Befehl *JHRW()*

BESCHREIBUNG: Der Referenzsuchlauf auf einen mit REF projektierten Digitaleingang aller selektierten Achskanäle wird in die rechte Verfahrrichtung gestartet.

BEISPIEL: *JHR(A2);*

6.6.22 JHRW, jog home right waiting

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *hac* und *hvl*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *jhr()* und SAP-Befehl *JHR()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.

ANMERKUNG: SAP-Befehl *JA()*

BEISPIEL: *JHRW(A1);*

6.6.23 JR, jog relative

FUNKTIONSPARAMETER: *Spec, Pos*

SYSTEMPARAMETER: Qualifizierer: *jac, jvl* und *jtvl*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Die Beschreibung erfolgt beim PCAP-Befehl *jr()*.

BEISPIEL: *JR(A1);*

6.6.24 JRW, jog relative waiting

FUNKTIONSPARAMETER: *Spec, Pos*

SYSTEMPARAMETER: Qualifizierer: *jac, jvl* und *jtvl*

SIMULTANFUNKTION: ja

REFERENZEN: JR

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *jr()* und SAP-Befehl *JR()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.

6.6.25 JS, jog stop

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *sdec*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Die Beschreibung erfolgt beim PCAP-Befehl *js()*.

BEISPIEL: *JS(A1);*

6.6.26 JSW, jog stop waiting

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *sdec*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *js()* und SAP-Befehl *JS()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet

BEISPIEL: *JSW(A1);*

6.6.27 LN, natural logarithm function

DEKLARATION: ln(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den natürlichen Logarithmus von *value* zurück, d.h. der Wert, mit dem die Konstante 2.71828... potenziert werden muß, um *value* zu erhalten.

ANMERKUNG: Wert kleiner/gleich 0.0 für *value* sind mathematisch nicht definiert. Die Funktion hat in diesem Fall keinen gültigen Rückgabewert.
Funktion *Exp()*

6.6.28 MCA, move circular absolute
SMCA, spool motion circular absolute

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: PCAP-Befehl *mca()*, *smca()*

BEISPIEL: *MCA(A1 := 50.0, A2 := 0.0, PHI := 720.0);*
SMCA(A1 := 0.0, A2 := 10.0, PHI := 0.1);

6.6.29 MCAW, move circular absolute waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

REFERENZEN: PCAP-Befehl *mca()*

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MCA()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.

6.6.30 MCR, move circular relative
SMCR, spool motion circular relative

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: PCAP-Befehl *mcr()*, *smcr()*

BEISPIEL:
```
MCR(A1 := 50.0, A2 := 0.0, PHI := 360.0);
SMCR(A1 := 0.0, A2 := 10.0, PHI := 45.0);
```

6.6.31 MCRW, move circular relative waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MCR()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.

6.6.32 MHA, move helical absolute
SMHA, spool motion helical absolute

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: PCAP-Befehl *mha()*, *smha()*

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert.

6.6.33 MHAW, move helical absolute waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MHA()*, jedoch wird hier zusätzlich das Profilende der drei beteiligten Achsen abgewartet.

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert.

6.6.34 MHR, move helical relative
SMHR, spool motion helical relative

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: PCAP-Befehl *mhr()*, *smhr()*

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert.

6.6.35 MHRW, move helical relative waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL, PHI*

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MHR()*, jedoch wird hier zusätzlich das Profilende der drei beteiligten Achsen abgewartet.

ANMERKUNG: Dieser Befehl ist momentan noch nicht implementiert.

6.6.36 MLA, move linear absolute
SMLA, spool motion linear absolute

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Die Beschreibung erfolgt bei dem PCAP-Befehl *mla()* bzw. *smla()*.

BEISPIEL:
```
MLA(A1:=1000.0, A2:=3.2e2);
SMLA(A1:=100.0, A2:=-335.0);
```

6.6.37 MLAW, move linear absolute waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MLA()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.

BEISPIEL:
```
MLAW(A1:=-0.3e3, A2:=100.4);
```

6.6.38 MLR, move linear relative
SMLR, spool motion linear relative

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Die Beschreibung erfolgt bei dem PCAP-Befehl *mlr()* bzw. *smlr()*.

BEISPIEL:
```
MLR(A1:=2000.0, A2:=3.2e2);
SMLR(A1:=300.0, A2:=-35.3);
```

6.6.39 MLRW, move linear relative waiting

FUNKTIONSPARAMETER: *Spec*, *Pos*

SYSTEMPARAMETER: *TRAC, TRVL, TRTVL*

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem SAP-Befehl *MLRW()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.

BEISPIEL: *MLRW(A1:=-3.45e3, A2:=100.4e-1);*

6.6.40 MS, motion stop

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: keine

SIMULTANFUNKTION: ja

BESCHREIBUNG: Die Beschreibung erfolgt beim PCAP-Befehl *ms()* [Kapitel 4.4.23].

BEISPIEL: *MS(A1, A2);*

6.6.41 MSW, motion stop waiting

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: keine

SIMULTANFUNKTION: ja

BESCHREIBUNG: Dieser Befehl ist identisch mit dem PCAP-Befehl *ms()* und SAP-Befehl *MS()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.

BEISPIEL: *MSW(A1, A2);*

6.6.42 OL, open loop

FUNKTIONSPARAMETER: *Spec*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *ol()* [Kapitel 4.4.24]

BEISPIEL: *OL(A1, A2); // Lageregelkreis von A1 und A2 öffnen*

6.6.43 RA, reset axis

FUNKTIONSPARAMETER: *Spec*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *ra()*

BEISPIEL: *RA(A1, A2); // Achsen A1 und A2 zurücksetzen*

6.6.44 RDCBD, read COMMON BUFFER double function

DEKLARATION: RDCBD(*offset*:integer)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert einen Gleitpunktwert mit doppelter Genauigkeit (double) aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.

Der Double-Datentyp belegt 8 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
...
var
        cbd: double;
...
cbd := RDCBD(500);          // Double-Variable einlesen ab offset 500
```

6.6.45 RDCBI, read COMMON BUFFER integer function

DEKLARATION: RDCBI(*offset*:integer)

ERGEBNISTYP: integer

BESCHREIBUNG: Die Funktion liefert einen Integerwert aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.

Der Integer-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
...
var
        cbi: integer;
...
cbi := RDCBI(500);          // Integer-Variable einlesen ab offset 500
```

6.6.46 RDCBS, read COMMON BUFFER single function

DEKLARATION: RDCBS(*offset*:integer)

ERGEBNISTYP: single

BESCHREIBUNG: Die Funktion liefert einen Gleitpunktwert mit einfacher Genauigkeit (single) aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.
Der Single-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
...
var
        cbs: single;
...
cbs := RDCBS(500);          // Single-Variable einlesen ab offset 500
```

6.6.47 RS, reset system

BESCHREIBUNG: PCAP-Befehl *rs()*

ANMERKUNG: Sofern dieser Befehl ausgeführt wird, ist keine Kontrolle durch das Standalone-Applikations-Programm mehr möglich, da die CNC-Task durch diesen Befehl angehalten wird.

BEISPIEL:
```
RS;    // komplettes Achssytem zurücksetzen
```

6.6.48 SHP, set home position

FUNKTIONSPARAMETER: *Spec, Pos*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *shp()*

BEISPIEL:
```
SHP(A2:=1000.0);
```

6.6.49 SIN, sine function

DEKLARATION: sin(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Sinus von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360 Grad) interpretiert.

ANMERKUNG: *Cos()*, *Tan()*-Funktion

BEISPIEL:
```
...
var
        d1, d2: double;
...
d1 := 3.1415;
d2 := SIN(d1);          // d2 := 0.0 (gerundet)
```

6.6.50 SINH, hyperbolic sine function

DEKLARATION: cos(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den hyperbolischen Sinus von *value* zurück.

6.6.51 SQRT, square root function

DEKLARATION: sqrt(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert die Quadratwurzel (»square root«) von value zurück.

ANMERKUNG: Negative Werte von *value* sind mathematisch nicht definiert. Die Funktion hat in diesem Fall keinen gültigen Rückgabewert.

BEISPIEL:
```
...
var
        d1, d2: double;
...
d1 := 9.0;
d2 := SQRT(d1);         // d2 := 3.0
```

6.6.52 SSMS, start spooled motions synchronous

FUNKTIONSPARAMETER: *Spec*

SIMULTANFUNKTION: ja

REFERENZEN: PCAP-Befehl *ssms()*, SAP-Befehl *SSMSW()*

BESCHREIBUNG: Mit Hilfe von *spool*-Befehlen können Kommandos an die einzelnen Achskanäle der MCU-3T übertragen werden. Diese werden in einer Warteschlange eingetragen. Der Befehl *SSMS()* veranlaßt den Synchronstart für die Spoolerbefehlsabarbeitung aller in *AS* spezifizierten Achsen.

BEISPIEL:
```
...
SMLA(A1:=1000.0, A2:=1000.0);      // Verfahrbefehl spoolen
SMLR(A1:=200.0, A2:=500.0);              // Verfahrbefehl spoolen
...
SSMS(A1, A2);                      // Spooler starten
```

6.6.53 SSMSW, start spooled motions synchronous waiting

FUNKTIONSPARAMETER: *Spec*

SIMULTANFUNKTION: ja

REFERENZEN: SAP-Befehl *SSMS()*

BESCHREIBUNG: Synchronstart aller angewählten Achsen und abwarten, bis sämtliche gespoolten Bewegungsprofile dieser Achsen komplett abgefahren sind und das Profilende aller beteiligten Achsen erreicht wird.

ANMERKUNG: SPOOL-Modus

BEISPIEL:
```
...
SMLR(A1:=1000.0, A2:=1000.0);      // Verfahrbefehl spoolen
SMLR(A1:=200.0; A2:=500.0);              // Verfahrbefehl spoolen
...
SSMSW(A1, A2);                     // Spooler starten
```

6.6.54 STARTCNCT, start CNC-Task

FUNKTIONSPARAMETER: Integer-Konstante im Bereich 0..3

BESCHREIBUNG: Dieser Befehl startet die im Parameter übergebene CNC-Task und führt das dort abgelegte SAP-Programm vom Programmbeginn an aus.

ANMERKUNG: Ein SAP-Programm kann sich mit diesem Befehl auch selbsttätig vom Programmbeginn an starten.

BEISPIEL:
```
...
const
        Task1 = 1;
...
STARTCNCT(Task1);
```

6.6.55 STOP, stop

BESCHREIBUNG: Dieser Befehl bewirkt den Programmstop des momentan ablaufenden Standalone-Applikations-Programms. Zusätzlich wird die entsprechende CNC-Task (Task 0, 1, 2, oder 3) in den Idle-Zustand gebracht.
Das Applikations-Programm kann mit Hilfe des *contcnct()*-PCAP-Befehls, des CONTCNCT()-SAP-Befehls oder im TOOLSET-Programm *mcfg.exe* wieder fortgesetzt werden.

ANMERKUNG: Eventuell freigegebene EVENT-Handling-Prozeduren werden nach Ausführen des Stop-Befehls nicht mehr abgearbeitet. Der Antrieb sollte vor Ausführung dieses Befehls deshalb in einen sicheren Betriebszustand gebracht werden.

BEISPIEL:
```
STOP;          // Stoppt das SAP-Programm
```

6.6.56 STOPCNCT, stop CNC-Task

FUNKTIONSPARAMETER: Integer-Konstante im Bereich 0..3

BESCHREIBUNG: Dieser Befehl hält die im Parameter übergebene CNC-Task an und damit auch das dort abgelegte SAP-Programm.

ANMERKUNG: Eventuell freigegebene EVENT-Handling-Prozeduren werden nach Ausführung des *STOPCNCT()* von der entsprechend selektierten Task nicht mehr abgearbeitet.

BEISPIEL:
```
...
const
        Task3 = 3;
...

STOPCNCT(Task3);
```

6.6.57 TAN, tangent function

DEKLARATION: tan(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den Tangens von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360) Grad interpretiert.

ANMERKUNG: *Sin()*, *Cos()*-Funktion

BEISPIEL:
```
...
var
        d1, d2: double;
...
d1 := 0.5;
d2 := TAN(d1);          // d2 := 0.5463 (gerundet)
```

6.6.58 TANH, hyperbolic tangent function

DEKLARATION: tan(*value*:double)

ERGEBNISTYP: double

BESCHREIBUNG: Die Funktion liefert den hyperbolischen Tangens von *value* zurück.

6.6.59 UF, update filter

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: Qualifizierer: *kp, ki, kd, kpl, kfca, kfcv*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *uf()*

ANMERKUNG: Zur Aktualisierung der PIDF-Filter-Koeffizienten müssen alle oben aufgeführten Qualifizierer vor Ausführung des Befehls initialisiert werden.

BEISPIEL:
```
...
A1.kp := 5.0;  // Proportionalverstärkung ändern
A1.ki := 0.0;
A1.kd := 0.0;
A1.kpl := 0.0;
A1.kfca := 0.0;
A1.kfcv := 0.0;
UF(A1);
...
```

6.6.60 UTROVR, update trajectory override

FUNKTIONSPARAMETER: *Spec*

SYSTEMPARAMETER: *TROVR*

SIMULTANFUNKTION: ja

BESCHREIBUNG: PCAP-Befehl *utrovr()*

BEISPIEL:
```
...
TROVR := 0.9;          // Bahngeschwindigkeitsoverride = -10%
UTROVR(A1, A2);        // reduzierte Bahngeschwindigkeit für Achsen A1 und A2
...
```

6.6.61 WRCBI, write COMMON BUFFER integer procedure

DEKLARATION: WRCBI(*offset*:integer; *value*:integer)

BESCHREIBUNG: Die Prozedur beschreibt eine Speicherzelle vom Typ integer mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.

Der Integer-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
WRCBI(500, -1000);      // Integer-Variable beschreiben ab offset 500
                        // mit Wert -1000
```

6.6.62 WRCBS, write COMMON BUFFER single procedure

DEKLARATION: WRCBS(*offset*:integer; *value*:single)

BESCHREIBUNG: Die Prozedur beschreibt eine Speicherzelle vom Typ single (Gleitpunkt-Zahl mit einfacher Genauigkeit) mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.

Der Single-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
WRCBS(500, 3.99);       // Single-Variable beschreiben ab offset 500
                        // mit Wert 3.99
```

6.6.63 WRCBD, write COMMON BUFFER double procedure

DEKLARATION: WRCBD(*offset*:integer; *value*:double)

BESCHREIBUNG: Die Prozedur beschreibt eine Speicherzelle vom Typ double (Gleitpunkt-Zahl mit doppelter Genauigkeit) mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER.

Der Double-Datentyp belegt 8 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das MCU-3T-CPU-System zu ermöglichen, muß *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.

ANMERKUNG: Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:
```
WRCBD(500, 100.2e-128);   // Double-Variable beschreiben ab offset 500
                          // mit Wert 100.2e-128
```

6.6.64 WT, wait timer

FUNKTIONSPARAMETER: Integer-Wert mit Einheit 64µs

BESCHREIBUNG: Die als Parameter übergebene Verweilzeit abwarten, bis das SAP-Programm wieder fortgesetzt wird. Dieser Befehl versetzt die CNC-Task in einen inaktiven Zustand und benötigt deshalb keine CPU-Zeit. Um das Master-CPU-System zu entlasten kann dieser Befehl ggf. in Warteschleifen o.ä. eingesetzt werden.

ANMERKUNG: Die EVENT-Handling-Prozeduren werden während der Ausführung dieses Befehls nicht abgearbeitet. Sollen diese jedoch überwacht werden, so kann dies beispielsweise durch mehrere *WT()*-Aufrufe mit kürzeren Verweilzeiten (evtl. in einer Schleife) erzwungen werden.

BEISPIEL:
```
...
CONST sec = 15625;
...
WT(5*sec);    // 5s warten
...
```

6.7 Compilerbefehle

Wie der Name vermuten läßt, weist ein Compilerbefehl den Compiler während der Übersetzung eines Quelltextes an, bestimmte Operationen auszuführen (oder zu unterlassen). In *rw_SymPas* wird ein Compilerbefehl wie folgt aktiviert:

Innerhalb des SAP-Quelltextprogramms wird innerhalb eines Kommentars eine spezielle Syntax formuliert: Direkt auf die öffnende Klammer ({) folgen ein Dollarzeichen ($) und der Name des Befehls, der aus einem oder mehreren Buchstaben besteht. Diese »Kommentare« können - von einigen Ausnahmen abgesehen - an jeder Stelle des Quelltextes erscheinen, an der auch ein normaler Kommentar zulässig ist.

6.7.1 Include-Datei

SYNTAX: {$I Dateiname}

Dieser Compilerbefehl weist den Compiler an, die durch *Dateiname* bezeichnete Datei einzulesen. Der Compiler verhält sich dabei im Prinzip so, als ob der gelesene Text anstelle des {$I}-Befehls stünde. *rw_SymPas* erlaubt die Verschachtelung von Include-Dateien bis zu 15 Ebenen. Eine via {$I} eingefügte Datei kann also ihrerseits weitere Dateien einfügen, die wiederum {$I}-Befehle enthalten.
Anmerkung: Sofern in der *mcfg.exe* NCC-Editor-Umgebung bereits eine Include-Datei in einem der drei Editor-Fenster eröffnet wurde, wird der SAP-Quelltext dieses Editors - und nicht der Inhalt der entsprechenden Datei - eingebunden.

6.7.2 Task-Auswahl

SYNTAX: {$TASK *TaskNr*}

BESCHREIBUNG: Mit diesem Compilerbefehl kann angegeben werden, in welcher Task (*TaskNr*, Werte 0..3) das entsprechende SAP-Programm ablaufen soll. Die Information wird im Autocodefile *„filename.cnc“* abgelegt. Mit Hilfe des PCAP-Befehls *txbf()* wird dieses File automatisch in die richtige Task übertragen.

ANMERKUNG: Sofern das entsprechende SAP-Programm diese Anweisung nicht enthält, wird die momentan selektierte Tasknummer zur Übersetzung herangezogen. Erfolgt aber der ($TASK)-Befehl, wird die entsprechend angewählte Tasknummer auch Default-Tasknummer für alle nachfolgenden Anzeige-, Start- und Stoppbefehle. Kapitel 3.2

BEISPIEL:

```
...
const
        Task1 = 1;
...

{$TASK Task1};          // oder
{$TASK 1};
```