Mikroschrittsteuerung

SM32

Handbuch

INHALT

Wichtige Hinweise

Handhabung

- Statische Aufladungen:
  Die SM32 ist zwar bis zu einem gewissen Grade gegen statische Entladungen geschützt, dennoch sollte man vor dem Aufnehmen/Ablegen einer Platine immer ein geerdetes Metallteil berühren.

- **Auf die Platine und in den PC dürfen keinerlei Metallsplitter, Bleistiftminenstücke oder andere leitende Teilchen gelangen.**

- Ein-/Ausbau:
  Ziehen Sie vor dem Öffnen des PC den Netzstecker.
  - Bei neueren PC's ist auch nach Betätigen des Ausschalters nicht gewährleistet, daß die Hauptplatine tatsächlich spannungsfrei ist.

  Einstecken/Ausziehen einer Platine **bei laufendem PC** führt zur **Zerstörung des Rechners und der Karte.**
  Schrauben Sie das Slotblech fest, andernfalls kann die Karte durch Bewegen des Motorkabels aus dem Platinenstecker gezogen werden.
  Achten Sie darauf, daß **keine Kurzschlüsse zwischen den Steckkarten** im PC entstehen.

Anwendungsbeschränkungen

- Die SM32 ist schon aufgrund des Zusammenspiels mit Standard-PC's nicht zugelassen für lebenserhaltende Systeme oder andere Anwendungen, bei denen Fehlfunktionen zu Sach- oder Personenschäden führen können.

- Beim Bremsen wirkt der Motor als Generator, die freiwerdende elektrische Energie müßte von der Steuerung vernichtet werden. Motorsteuerungen dieser Leistungsklasse sind hierfür generell nicht ausgelegt!
  Bei allen normalen Anwendungen ergeben Führungen, Spindeln und Getriebe hinreichend energievernichtende Reibung, lassen Sie aber beispielsweise kein Gewicht von einer Seilscheibe ab. Exzessives Bremsen könnte zur Beschädigung der Endstufen, im Extremfall sogar des Netzteiles führen

Funktionen und Leistungsmerkmale

- Steckkarte für PC zum Betrieb von
  drei bipolaren 2/4Phasen-Schrittmotoren.
  - Endstufen auf der Karte zum direkten Anschluss der Motoren.
  - Eigener Prozessor für Positionierfahrten und Motorregelung.
  - Pro Motor sind zwei Endschalter anschließbar.
  - Notaus-Taster (Schließer-Kontakt) anschließbar.
  - Drei zusätzliche Digitaleingänge

- Vorgesehene Verwendung: Programmierung mit C / C++ / Pascal / Delphi

- Software-Schnittstellen für C, C++, Pascal, Delphi und Basic.
  Plattformen: DOS, Windows 3.x , 9.x, NT4, Windows2000

Systemvoraussetzungen

- PC mit einem freien PCI Steckplatz

Lieferumfang

- Eine Steckkarte SM32.
- Handbuch.
- Diskette:
  Softwareversionen für
  - DOS.
    Einschließlich Quellen der Hardware-Schnittstelle, welche gegebenenfalls für andere Plattformen anpassbar sind.
  - Windows32 (9x / NT/ 2000) , mit Port-Treiber für Windows NT.
    Zugriff auf die SM32 wird über eine DLL koordiniert, mehrere Programme oder Threads können daher gleichzeitig dieselbe SM32 ansprechen.

  Dateien:
  - Konfigurations- und Testprogramm, Ausführungen:
    sm32win.exe für Windows32
    sm32dos.exe für DOS
  - Programmierschnittstelle:
    sm32.h / sm32.lib / sm32.c für VC++ / ANSI-C
    usm32.pas für Delphi / Pascal
    sm32.bas für Visual Basic (nur Windows32)
  - Weitere Dateien: siehe READ.ME-Dateien

Bedienungsanleitung

Installation

Handhabung

- Statische Aufladungen:
  Die SM32 ist zwar bis zu einem gewissen Grade gegen statische Entladungen geschützt, dennoch sollte man vor dem Aufnehmen/Ablegen einer Platine immer ein geerdetes Metallteil berühren.

- Auf die Platine und in den PC dürfen keinerlei Metallsplitter, Bleistiftminenstücke oder andere leitende Teilchen gelangen.

Einbau

Nach Ausschalten des Rechners und Ausstecken des Netzkabels kann der Rechner geöffnet werden.
Bei einem freien PCI-Steckplatz wird ein Abdeckblech abgeschraubt, die SM32 hier eingesetzt und festgeschraubt. Erst hierdurch ist sichergestellt, daß die Karte richtig in der Buchse steckt, auch wird verhindert, daß sie später durch Zug am Motorkabel aus der Buchse gezogen werden kann.

> Einstecken/Ausziehen einer Steckkarte bei laufendem PC führt zur Zerstörung des Rechners und der Karte.

Abschließend wird zur Versorgung der SM32 mit Betriebsstrom am Stromversorgungsanschluß oben rechts ein Stromversorgungsstecker angeschlossen (Kabelfarben: rot-schwarz-schwarz-gelb).
Meist ist im Rechner noch ein derartiger Anschluß frei, gegebenenfalls kann ein handelsübliches Abzweigstück eingesetzt werden.

Rechner einschalten

Falls der Rechner nach dem Einschalten nicht sofort normal anläuft, ist er umgehend auszuschalten, darauf muß die SM32 wieder ausgebaut und nochmals bei "Einbau" begonnen werden.

Windows 2000 / 98 erkennen, dass neue Hardware installiert ist:
Zur Installation des Treibers ist
A:\Win32\Drivers\Win2000.98\Pci9030.inf
zu suchen und anzuklicken.

Windows 95:
Hier wird kein Treiber benötigt, eine Aufforderung durch Windows, einen Treiber zu installieren, kann ignoriert gegebenenfalls werden.

Software kopieren

Zur Installation unter Windows 9x / NT / 2000 steht ein Installationsprogram zur Verfügung, dieses

- installiert einen Porttreiber (nur Windows NT),
- kopiert DLLs für Portzugriff und SM32-Funktionen in das Systemverzeichnis,
- kopiert die zugehörigen Dateien in ein anzugebendes Verzeichnis,
- erzeugt auf dem Desktop eine Verknüpfung dorthin.

Für DOS ist keine Setup-Software vorhanden, die wenigen Dateien lassen sich einfacher nach Bedarf manuell kopieren.

Externe Anschlüsse

Entsprechend der Steckerbelegung auf Seite 49 sind anzuschließen :

- Motoren mit Endschaltern
- Gegebenenfalls ein Notaus-Taster

Software

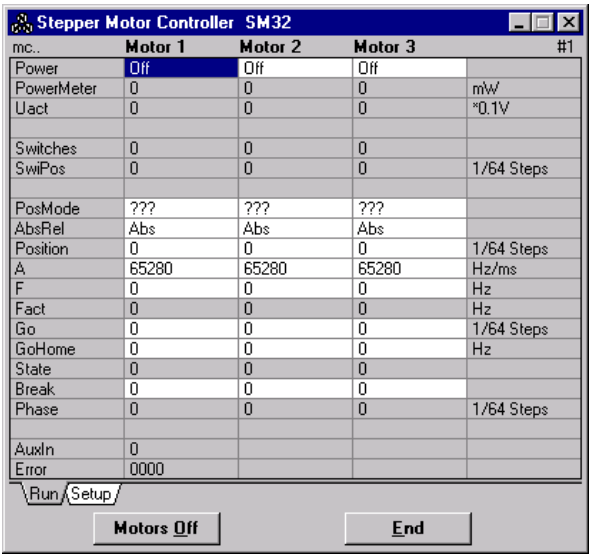
SM32WIN.EXE	für Windows 9x / NT
SM32DOS.EXE	für DOS

Diese Programme dienen zum ersten Kennenlernen der Befehle sowie zur Konfiguration.
Beim Einstieg wird man nach der Nummer der SM32 im System gefragt, alternativ kann diese auch in der Kommandozeile angegeben werden, Beispiel:

SM32WIN –n1

Im Hauptbildschirm (hier die Windows-Version) steht die volle Funktionalität zur Verfügung.

Im hier gezeigten Registerblatt \Run/ , das nach dem Starten des Programmes immer aktiv ist, finden sich alle Befehle, welche für den Normalbetrieb von Interesse sind. Seltener benötigte Befehle sind in das Registerblatt \Setup/ verlagert.



Programmierschnittstelle

Übersicht

Auf der Diskette befinden sich Schnittstellendateien für:

- Windows 32 (9.x / NT / 2000).
  - Port-Treiber für Windows 98 / NT / 2000.
  - DLLs welche Zugriffe auf die SM32 vermitteln und koordinieren; beide werden durch das Setup-Programm installiert.
  - ANSI C/C++ : sm32.h / sm32.lib (getestet unter VC++ 5.0)
  - Delphi: uSM32.pas (getestet unter Delphi III)
  - Visual Basic: sm32.bas (getestet mit VB5.0CCE)
- DOS.
  - ANSI C : sm32.c / sm32.h (getestet unter Turbo C 2.01).
    Hier sind möglicherweise kleine Anpassungen an die Erfordernisse des verwendeten Compilers vonnöten: Am Anfang von sm32.h werden die benutzten Variablentypen definiert, wie etwa
    ```
    #define WORD16 unsigned short
    ```
    In sm32.c muß dafür gesorgt werden, daß die passenden Port-I/O-Funktionen angesprochen werden; gegebenenfalls sind zwei in Assembler geschriebene BIOS-Zugriffsfunktionen anzupassen.
  - Pascal, Delphi: uSM32.PAS (getestet unter BP7, Delphi I, Delphi III)

Die Schnittstellendateien stellen zur Kommunikation mit der SM32 zur Verfügung:

- Einen Typ `tSM32Motor` → Seite 5
- Kommunikationsfunktionen → Seite 6
- Befehls- und sonstige Konstanten → Befehlsreferenz, Seite 19

Übliches Programmierschema:

- Für jeden Motor wird eine Variable des `tSM32Motor`-Typs definiert
- Die Variablen werden mit `SM32Init(..)` mit der SM32 verbunden
- Mit `SM32Write(..)` und `SM32Read(..)` werden Daten unter Zuhilfenahme der Befehlskonstanten an die SM32 gesendet und von ihr abgefragt

Beispiele:

C
```
#include "sm32.h"
tSM32Motor Motor1;
main() {
  SM32Init( Motor1, 1, 1 );
  SM32Write(Motor1, mcPower, 1); /* Motor 1 einschalten*/
}
```

Pascal
```
uses uSM32;
var Motor1 :tSM32Motor;
begin
  SM32Init( Motor1, 1, 1 );
  SM32Write(Motor1, mcPower, 1); (* Motor 1 einschalten*)
end.
```

Variablentyp tSM32Motor

Für jeden Motor muß eine Variable dieses Typs definiert werden.
Beispiel:

```
C         tSM32Motor Motor1;

Pascal    var Motor1 :tSM32Motor;
```

C-Definition:

```
typedef long tSM32Motor,  *pSM32Motor;
```

Pascal-Definition:

```
type  pSM32Motor = ^tSM32Motor;
      tSM32Motor = longint;
```

Beschreibung:

Nach `SM32Init(...)` können alle anderen Funktionen den Motor mittels dieser Variable ansprechen.

Windows32 :

`SM32Init` ruft die DLL auf und fordert einen Handle (Zugriffsnummer) für den Motor an, dieser wird, zusammen mit Verwaltungsinformationen, in der Motorvariable gespeichert.

In der Folge muß immer das Original der Variable verwendet werden:

- Zuweisungen wie Kopie:=Original sind unzulässig.
- Die Übergabe einer Motorvariable an Funktionen/Prozeduren darf nur mittels Referenzparametern geschehen ("VAR", "ByRef", "*").

Andererseits muß für jedes unabhängige Programm / jeden Thread eine neue Motorvariable initialisiert werden – wenn stattdessen eine Referenz der Variable übergeben würde, könnte die DLL nicht entscheiden, welchem Thread angeforderte Daten gehören.

DOS / Windows 3x :

`SM32Init` speichert Motornummer und die Portadresse der SM32 in der Variable.

Funktionen

`SM32Init`	Verbindet eine tSM32Motor-Variable mit der tatsächlichen SM32-Steuerung.
`SM32Done`	Markiert eine tSM32Motor-Variable als ungültig.
`SM32Write` `SM32Read`	Daten an die Steuerung senden und von ihr abfragen.

`SM32Post`
`SM32Request`
`SM32Replied`
`SM32Fetch`
	Daten an die Steuerung senden und von ihr abfragen, unter besserer Ausnutzung auftretender Wartezeiten.

Die SM32 benötigt etwa ein bis zwei Millisekunden, um mit `SM32Write` gesendete Daten zu verarbeiten oder in `SM32Read` angeforderte Daten zurückzugeben. Diese Funktionen verbringen die Wartezeit in Warteschleifen. Für komplexere Anwendungen, bei denen ein höherer Datendurchsatz erwünscht ist, stehen zum Schreiben `SM32Post`, zum Lesen `SM32Request` / `SM32Replied` / `SM32Fetch` zur Verfügung. Hiermit kann so programmiert werden, daß auftretende Wartezeiten zur Erledigung anderen Aufgaben nutzbar sind.
Die Windows32-DLL ruft während Wartezeiten die Windows-Funktion sleep() auf, so daß Wartezeiten anderen Programmen/Threads zugute kommen, dennoch mag es bei kritischen Applikationen besser sein, eigene Warteschleifen zu schreiben, da sleep() keine Windows-Nachrichten wie WM_LBUTTONCLICK verarbeitet.

Hilfsfunktionen

`SM32CmdToName`	Ermittle Kommandoname aus Kommandonummer.
`SM32NameToCmd`	Ermittle Kommandonummer aus Kommandoname.
`SM32CmdFlags`	Ermittle einige Flags eines Kommandos.
`SM32GetTimeout`	Ermittle Timeout-Wert für `SM32Read` und `SM32Write`.
`SM32SetTimeout`	Setze Timeout-Wert für `SM32Read` und `SM32Write`.

Auf den folgenden Seiten werden die Definitionen im Detail beschrieben.

```
long     SM32Init( tSM32Motor *Motor, long Mno, long Bno);

function SM32Init( var Motor :tSM32Motor;
                   Mno, Bno  :longint     ) :longint;
```

Aufgabe: `tSM32Motor` -Variable mit realer SM32 verbinden

Parameter:

`Motor`:	Zu initialisierende SM32Motor-Variable
`Mno`:	Nummer des Motors (1..3)
`Bno`:	Nummer der SM32 im Rechner

Rückgabewerte :

`mcrOk`	: Init erfolgreich
`mcrIllegalMno`	: Nummer nicht in (1..3)
`mcrIllegalBno`	: Nummer nicht in (1..10)
`mcrNoSM32`	: Keine SM32 gefunden
`mcrNoNTDriver`	: Windows NT: Treiber nicht gefunden
`mcrTooManyHandles`	: Windows32: Mehr als 255 Handles angefordert
`mcrTooManyBoards`	: Windows32: Interner Fehler (sollte nicht vorkommen)

Beschreibung:

Jede tSM32Motor-Variable muß vor der Verwendung mit `SM32Init` initialisiert werden.

SM32Init durchsucht die Steckplätze des Rechner nach SM32-Karten, bis die SM32 Nummer <Bno> gefunden wird.

- Im Erfolgsfall wird `mcrOk` zurückgegeben; `Motor` ist mit der SM32 verbunden und kann von den anderen Funktionen verwendet werden.
- Andernfalls wird eine der anderen Fehlerkonstanten zurückgegeben.

Test-Modus:
Bei erfolglosem `SM32Init` wird in `Motor` eine spezielle Konstante gespeichert, welche den anderen Funktionen "Test-Modus" signalisiert. Sie verhalten sich dann so, als wäre alles in Ordnung, es wird immer `mcrOk` zurückgegeben. Dies kann man ausnutzen, um eine Software ohne Steuerung zu testen:
`SM32Write` und `SM32Post` verwerfen Befehl und Daten,
`SM32Read` und `SM32Fetch` liefern als gelesenes Datum immer Null,
Ausnahme: Einlesen von `mcError` ergibt `meNotFound` .

Beispiel: Verbinde Motorvariable mit erstem Motor der ersten SM32

C

```
tSM32Motor Motor1;
main()
{
  if( SM32Init( & Motor1, 1, 1 ) != mcrOk  ) {
    printf("Keine SM32 Nummer 1 gefunden\n");
    exit(1);
  };
  .
  .
};
```

Pascal

```
var Motor1 :tSM32Motor;
begin
  if SM32Init( Motor1, 1, 1 ) <> mcrOk then begin
    writeln( 'Keine SM32 Nummer 1 gefunden' );
    HALT(1);
  end;
  .
  .
end.
```

```
long     SM32Done( tSM32Motor *Motor );

function SM32Done( var Motor :tSM32Motor );
```

Aufgabe: `tSM32Motor`-Variable als ungültig markieren

Parameter:
`Motor`: tSM32Motor-Variable

Rückgabewerte:
`mcrOk` : SM32Done war erfolgreich.
`mcrNotInitialised`: Variable war nicht initialised.

Beschreibung:

Die tSM32Motor-Variable wird als 'uninitialisiert' markiert.

Windows 9.x / NT:

In der DLL wird der Handle (Zugriffsnummer) freigegeben, so daß er wieder für andere `SM32Init()` zur Verfügung steht.

Bei Programmende ruft die DLL automatisch SM32Done für alle noch mit dem Programm verbundenen Handles auf.

Beispiel:

C

```
SM32Done( &Motor1 );
```

Pascal

```
SM32Done( Motor1 );
```

```
long      SM32Write( tSM32Motor *Motor,
                     long Cmd, long Data );
```

```
function  SM32Write( var Motor :tSM32Motor;
                     Cmd, Data :longint    ) :longint;
```

Aufgabe: Daten an die SM32 senden

Parameter:

`Motor`: Angesprochener Motor
`Cmd`: Befehlsnummer (`mcMinCmd..mcMaxCmd`)
`Data`: Zu sendende Daten

Rückgabewerte:

`mcrOk` : Write erfolgreich

`mcrNotInitialised`: Motorvariable nicht initialisiert
`mcrTimeout` : SM32 reagiert nicht

Beschreibung:

Mit `SM32Write` können Daten an die Steuerung gesendet werden.

`Cmd` gibt an, um welche Art von Datum es sich handelt.

- Zulässige Werte siehe Befehlsreferenz
- Unzulässige Befehlsnummern werden von der SM32 ignoriert

Anmerkungen:

`SM32Write` benötigt im Normalfall ein bis zwei Millisekunden.
Wenn die Steuerung nicht reagiert, versucht `SM32Read` maximal dreißig Sekunden lang, die Steuerung anzusprechen und bricht dann mit Rückgabewert `mcrTimeout` ab. Ursachen können sein:

- Während `SM32Write` ausgeführt wird, blockiert ein anderes Programm den PC für längere Zeit.
- Der Computer auf der SM32 ist abgestürzt. Dies trat bisher noch nicht auf.

Beispiel: Motor 1 einschalten

C

```
tSM32Motor Motor1;
main()
{
  if( SM32Init( &Motor1, 1, 1 ) != mcrOk ) exit(1);
  SM32Write( & Motor1, mcPower, 1 );
};
```

Pascal

```
var Motor1 :tSM32Motor;
begin
  if SM32Init( Motor1, 1, 1 ) <> mcrOk then HALT(1);
  SM32Write( Motor1, mcPower, 1 );
end.
```

```
long     SM32Read( tSM32Motor *Motor, long Cmd, long *Data );

function SM32Read( var Motor :tSM32Motor;
                   Cmd :longint; var Data:longint ) :longint;
```

Aufgabe: Daten von der SM32 auslesen

Parameter:

`Motor`:	Angesprochener Motor
`Cmd`:	Befehlsnummer (`mcMinCmd..mcMaxCmd`)
Data:	Variable zur Aufnahme der empfangenenen Daten

Rückgabewerte:

`mcrOk`	: Read erfolgreich
`mcrNotInitialised`	: Motorvariable nicht initialisiert
`mcrTimeout`	: SM32 reagiert nicht
`mcrInterference`	: Kommunikation wurde gestört

Beschreibung:

Mit `SM32Read` können Werte von der Steuerung gelesen werden.

Mit `Cmd` wird angegeben, um welche Art von Wert es sich handelt

- Zulässige Werte siehe Befehlsreferenz
- Bei unbekannten Befehlsnummern gibt die SM32 in Data den Wert 0 zurück

Anmerkungen:

- `SM32Read` benötigt im Normalfall ein bis zwei Millisekunden, bei `Cmd=mcNone` nur 2 Mikrosekunden.
  Wenn die Steuerung nicht reagiert, versucht `SM32Read` maximal dreißig Sekunden lang, die Steuerung anzusprechen und bricht dann mit Rückgabewert `mcrTimeout` ab.
  Ursachen können sein:
  - Während `SM32Read` ausgeführt wird, blockiert ein anderes Programm den PC für längere Zeit
  - Der Computer der SM32 ist abgestürzt. Dies trat bisher noch nicht auf.

- Die SM32 stellt sicher, daß ein vorangegangenes `SM32Write` bearbeitet wurde, bevor `SM32Read` ein Ergebnis liefert.

- `mcrInterference` kann auftreten, mehrere Programme gleichzeitig mittels der DOS-Schnittstellendateien auf die SM32 zugreifen.

Beispiel: Ist Motor 1 eingeschaltet?

C

```
tSM32Motor Motor1;
long Data;
main()
{
  if( SM32Init( &Motor1, 1, 1 ) !=mcrOk ) exit(1);
  SM32Read( &Motor1, mcPower, &Data );
  if( Data !=0 ) printf( “Motor 1 ist eingeschaltet” );
  else           printf( “Motor 1 ist ausgeschaltet” );
};
```

Pascal

```
var Motor1 :tSM32Motor;
    Data   :longint;
begin
  if SM32Init( Motor1, 1, 1 ) <>mcrOk then HALT(1);
  SM32Read( Motor1, mcPower, Data )
  if Data <>0  then writeln( 'Motor 1 ist eingeschaltet' )
  else              writeln( 'Motor 1 ist ausgeschaltet' );
end.
```

```
long     SM32Post( tSM32Motor *Motor,
                   long aCmd, long aData );

function SM32Post( var Motor :tSM32Motor;
                   Cmd, Data :longint    ) :longint;
```

Aufgabe: Daten an die SM32 senden

Parameter:

`Motor`:	Angesprochener Motor
`Cmd`:	Befehlsnummer (`mcMinCmd..mcMaxCmd`)
`Data`:	Zu sendende Daten

Rückgabewerte:

`mcrOk`	: Post erfolgreich
`mcrBusy`	: Schnittstelle belegt
`mcrNotInitialised`	: Motorvariable nicht initialisiert

Beschreibung:

Wenn die SM32 die zuvor gesendeten Daten noch nicht abgeholt hat, kann `SM32Post` die Daten noch nicht abgeben und kehrt mit `mcrBusy` zurück. Um die Daten doch an die SM32 zu senden, muß `SM32Post` solange wiederholt werden, bis es `mcrOk` liefert; zwischen den Versuchen kann man andere Programmteile ausführen.

Beispiel: Motor 1 einschalten und `mcPosMode` auf `mmPos` setzen

C

```
tSM32Motor Motor1;
main() {
  if( SM32Init( &Motor1, 1, 1 ) !=mcrOk ) exit(1);
  SM32Write( &Motor1, mcPower, 1 );
    /* Verarbeitung dauert 1..2 Millisekunden */
  while( SM32Post( &Motor1,mcPosMode,mmPos) ==mcrBusy) {
    /* waehrend der Wartezeit andere Aufgaben erledigen */
  };
};
```

Pascal

```
var Motor1 :tSM32Motor;
begin
  if SM32Init( Motor1, 1, 1 ) <>mcrOk then HALT(1);
  SM32Write( Motor1, mcPower, 1 );
    (* Verarbeitung dauert 1..2 Millisekunden *)
  while SM32Post( Motor1, mcPosMode, mmPos) =mcrBusy
  do (* waehrend der Wartezeit andere Aufgaben erledigen *)
end.
```

```
long     SM32Request( tSM32Motor *Motor, long Cmd );

function SM32Request( var Motor :tSM32Motor;
                      Cmd       :longint    ) :longint;
```

Aufgabe: Daten von der SM32 anfordern

Parameter:
- `Motor`: Angesprochener Motor
- `Cmd`: Befehlsnummer (`mcMinCmd..mcMaxCmd`)

Rückgabewerte:
- `mcrOk` : Anforderung wurde an die SM32 gesendet
- `mcrBusy` : Anforderung nicht gesendet da Schnittstelle belegt
- `mcrNotInitialised`: Motorvariable nicht initialisiert

```
long     SM32Replied( tSM32Motor *Motor );

function SM32Replied( var Motor :tSM32Motor) :longint;
```

Aufgabe: Test: Hat die SM32 die letzte Anfrage beantwortet?

Parameter:
- `Motor`: Angesprochener Motor

Rückgabewerte :
- `mcrOk` : SM32 hat Daten geliefert
- `mcrBusy` : Daten noch nicht geliefert
- `mcrNotInitialised`: Motorvariable nicht initialisiert
- `mcrInterference` : Kommunikation wurde gestört

```
long     SM32Fetch( tSM32Motor *Motor, long *Data );

function SM32Fetch( var Motor :tSM32Motor;
                    var Data  :longint    ) :longint;
```

Aufgabe: Mit `SM32Request` angeforderte Daten abholen

Parameter:
- `Motor`: Angesprochener Motor
- Data: Variable zur Aufnahme der empfangenenen Daten

Rückgabewerte:
- `mcrOk` : SM32 hat Daten geliefert
- `mcrBusy` : Daten noch nicht geliefert
- `mcrNotInitialised`: Motorvariable nicht initialisiert
- `mcrInterference` : Kommunikation wurde gestört

Beschreibung:

Wenn `SM32Read` Daten von der SM32 anfordert, dauert es bis zu 2 Millisekunden bis diese von der SM32 geliefert werden; solange würde `SM32Read` in einer Warteschleife warten. Mit Hilfe von `SM32Request` / `SM32Replied` / `SM32Fetch` kann diese Wartezeit für andere Aufgaben verwendet werden:

- Mit `SM32Request` wird ein Datum angefordert
- `SM32Replied` ermittelt, ob das angeforderte Datum geliefert wurde
- Wenn `SM32Replied` TRUE ergibt, kann das Datum mit `SM32Fetch` abgeholt werden

Beispiel: Interna von `SM32Read`

`SM32Read` ist im Prinzip wie folgt programmiert :

C

```
long SM32Read( tSM32Motor *Motor, long Cmd. long *Data )
{
  while( SM32Request( Motor, Cmd ) ==mcrBusy ) {
    /* hier koennte man andere Aufgaben erledigen */
  };

  while( SM32Replied( Motor ) ==mcrBusy ) {
    /* hier koennte man andere Aufgaben erledigen */
  };

  SM32Fetch( &Motor, mcNone, Data );
  return mcrOk;
};
```

Pascal

```
function SM32Read( var Motor :tSM32Motor;
                   Cmd:longint; var Data:longint ):longint;
begin
  while SM32Request( Motor, Cmd ) =mcrBusy do begin
    (* hier koennte man andere Aufgaben erledigen *)
  end;

  while SM32Replied( Motor ) =mcrBusy do begin
    (* hier koennte man andere Aufgaben erledigen *)
  end;

  SM32Fetch( Motor, mcNone, Data );
  SM32Read := mcrOk;
end.
```

```
char*    SM32CmdToName( long  Cmd  );

function SM32CmdToName( Cmd :longint ) :pChar;
```

Aufgabe: Kommandoname aus Kommandonummer ermitteln.

Parameter:
`Cmd`: Kommandonnummer

Rückgabewert: Name des Kommandos

Beschreibung:
Eine Zeichenkette wird zurückgegeben, welche den Namen des Kommandos ohne führendes 'mc' enthält. Bei unbekannten Kommandonummern wird eine leere Zeichenkette zurückgegeben.

Beispiel: Drucke Namen eines Kommandos
C
```
printf( "Der Name ist mc%s\n", SM32CmdToName(mcPower) );
```

Pascal
```
writeln( 'Der Name ist mc',    SM32CmdToName(mcPower) );
```

```
long     SM32NameToCmd( char* Name );

function SM32NameToCmd( Name:pChar   ) :longint;
```

Aufgabe: Kommandonummer aus Kommandoname ermitteln.

Parameter:
`Name`: Kommandoname, mit oder ohne führendem 'mc'.

Rückgabewert: Kommandonummer

Beschreibung:
Die Nummer des Kommandos wird zurückgegeben. Wenn der angegebene Name keinem Kommando entspricht, wird `mcNone` zurückgegeben.

Beispiel: Drucke Nummer eines Kommandos
C
```
printf("Nummer: %d\n", (int)SM32NameToCmd("power") );
```

Pascal
```
writeln('Nummer: ', SM32NameToCmd('power') );
```

```
long     SM32CmdFlags(  long  Cmd  );

function SM32CmdFlags(  Cmd :longint ) :longint;
```

Aufgabe: Verschiedene Flags ermitteln

Parameter:
`Cmd`: Kommandonummer

Rückgabewert: 0 .. (1+2+4)

Beschreibung:

Ein Bitfeld wird zurückgegeben, welches aussagt, ob
- mit diesem Kommando Daten von SM32 gelesen werden können: 1
- mit diesem Kommando Daten zur SM32 gesendet werden können: 2
- es sich um ein Systemkommando handelt: 4

Bei unbekannten Kommandos wird Null zurückgegeben.

Beispiel: Kommandoflags eines Kommandos ausdrucken

C
```
printf("Flags: %d\n", (int)SM32CmdFlags(mcSysPower) );
```

Pascal
```
writeln('Flags: ', SM32CmdFlags(mcSysPower) );
```

```
long     SM32GetTimeout(void);

long     SM32SetTimeout( long t );

function SM32GetTimeout :longint;

function SM32SetTimeout(t:longint) :longint;
```

Aufgabe: Timeout-Wert für `SM32Read` and `SM32Write` setzen/lesen

Parameter:
`t`: Timeout in Millisekunden

Rückgabewert: Gegenwärtige Einstellung in Millisekunden

Vorgabe: 30.000 = 30 Sekunden

Anhänge

Befehlsreferenz

Übersicht

Bei den Konstanten ist in Klammern angegeben, ob
(R) : ...Daten mit `SM32Read` ausgelesen werden können und
(W) : ...Daten mit `SM32Write` an die SM32 übergeben werden können.
(S) : bezeichnet Systemkommandos, welche von jedem Motor angesprochen werden können.
(M) : Die Werkseinstellung solcher Daten kann mit mcStoreCfg überschrieben werden.

Eine Auflistung aller Befehle findet sich im Befehlsindex am Ende des Handbuchs.

Die Beispiele der folgenden Befehlsbeschreibungen gehen davon aus, daß eine `tSM32Motor`-Variable `Motor1` vorhanden ist und diese ordnungsgemäß initialisiert wurde, hierzu könnten etwa folgende Codestücke dienen:

C

```
tSM32Motor Motor1;
main()
{
  if( SM32Init( &Motor1, 1, 1 ) !=mcrOk ) exit(1);
  /* hier kann das Beispiel eingefuegt werden */
};
```

Pascal

```
var Motor1 :tSM32Motor;
begin
  if SM32Init( Motor1, 1, 1 ) <>mcrOk then HALT(1);
  (* hier kann das Beispiel eingefuegt werden *)
end.
```

`mcError (R S )`

Aufgabe: Systemfehlermeldungen auslesen

Werte: Siehe unten

Vorgabe: 0

Beschreibung:

Konstante	Wert (hex)	Beschreibung
`meNone`	0	Kein Fehler
Vorübergehende Fehler:		
`meIM1`	1	Überstrom Motor 1
`meIM2`	2	Überstrom Motor 2
`meIM3`	4	Überstrom Motor 3
`meIover`	8	Allgemeine Strombegrenzung
Andauernde Fehler; diese Meldungen können nur durch Abschalten aller Motoren mit `mcPower` beseitigt werden:		
`meSupply`	10	5V-Ausgang kurzgeschlossen oder 12V-Versorgung fehlt
`meSwitch`	20	Endschalter wurde im Nothaltmodus betätigt
`meWatchdog`	40	Kommunikationsüberwachung angesprochen
`meReset`	80	Interner Fehler
`meHalted`	f0	Maske: Andauernder Fehler

Beispiel: Liegt ein andauernder Fehler vor?

C

```
long Data;
SM32Read( &Motor1, mcSysError, &Data );
if( Data & meHalted ) { /* Fehler */ };
```

Pascal

```
var Data :longint
SM32Read( Motor1, mcSysError, Data );
if Data and meHalted <> 0 then begin (* Fehler *) end.
```

mcWatchdog (RWS)

Aufgabe: Kommunikationsüberwachung aktivieren

Werte: 0=inaktiv, 1..n (Millisekunden)=aktiv

Vorgabe: 0

Beschreibung:

Die SM32 kann zu Aktivitäten veranlaßt werden, welche eine permanente Überwachung durch den PC erfordern, etwa Fahren im Betriebsmodus `mmMove` bei abgeschalteten Endschaltern. Um katastrophalen Folgen bei Absturz des PC's vorzubeugen, besitzt die SM32 eine Kommunikationsüberwachung. Wenn sie aktiviert ist (Werte >0), muß immer innerhalb der eingestellten Zeit eine Kommunikation mit der SM32 stattfinden. Bleibt diese aus, werden alle Motoren mit mcBreak angehalten, bei `mcError` erscheint die Fehlermeldung `meWatchdog` . Dieser Zustand kann nur durch Abschalter aller Motoren beseitigt werden.

Beispiel: Kommunikationsüberwachung auf zwei Sekunden einstellen

C

```
SM32Write( &Motor1, mcWatchdog, 2000 );
```

Pascal

```
SM32Write( Motor1, mcSysWatchdog, 2000 );
```

mcVersion (R S)

Aufgabe: Firmware-Version auslesen

Werte: ?

Vorgabe: —

Beschreibung:

Kann beispielsweise verwendet werden, um sicherzustellen, daß die eingesetzte SM32 zu der PC-Software paßt.

Der ausgelesene Wert durch 1000 dividiert ergibt in den Vorkommastellen die Hauptversion, in den Nachkommastellen die Unterversion.

Beispiel:
Das Anwendungsprogramm verlangt nach einer SM32 mit mindestens Firmware-Version 1.000

C

```
long Data;
SM32Read( &Motor1, mcSysVersion, &Data );
if( Data < 1000 ) {
   /* Fehler */
};
```

Pascal

```
var Data :longint;
SM32Read( Motor1, mcSysVersion, Data );
if Data < 1000 then begin
   (* Fehler *)
end;
```

mcAuxIn (R S)

Aufgabe: Digitaleingänge lesen

Werte: 0 ... (1+2+4)

Vorgabe: —

Beschreibung:

Die Digitaleingänge werden eingelesen.
Das Ergebnis ist eine Kombination aus
1 : Eingang "1" = High
2 : Eingang "2" = High
4 : Eingang "3" = High

Im unbeschalteten Zustand sind alle drei Eingänge high, das Ergebnis von `mcAuxIn` ist daher 7.

Beispiel: Digitaleingänge einlesen

C

```
long Data;
SM32Read( &Motor1, mcAuxIn, &Data );
```

Pascal

```
var Data :longint;
SM32Write( Motor1, mcAuxIn, Data );
```

mcReset (WS)

Aufgabe: Steuerung in Einschaltzustand versetzen

Werte: 12345678

Vorgabe: —

Beschreibung:

Die SM32 wird in den Einschaltzustand versetzt:

- Die Stromversorgung wird abgeschaltet.
- Alle Einstellungen gehen verloren.
- Um eine versehentliche Fehlbedienung auszuschliessen, wird das Kommando nur erkannt wenn der 'magische Wert' 12345678 übergeben wird.

Beispiel: SM32 in den Einschaltzustand versetzen

C

```
SM32Write( &Motor1, mcReset, 12345678 );
```

Pascal

```
SM32Write( Motor1, mcReset, 12345678 );
```

Bedienen

mcPower (RW)

Aufgabe: Motor ein/ausschalten oder anhalten

Werte: 0, 1, -1

Vorgabe: 0

Beschreibung:

Konstante	Wert	Beschreibung
mpOff	0	Die Stromversorgung des Motors wird ausgeschaltet
mpOn	1	Die Stromversorgung des Motors wird eingeschaltet
mpShutdown	-1	Der Motor wird auf Geschwindigkeit Null gebremst, dann wird die Stromversorgung des Motors ausgeschaltet

Solange der Motor ausgeschaltet ist, werden Fahrbefehle ignoriert.

Beispiel: Motor 1 einschalten

C

```
SM32Write( &Motor1, mcPower, 1 );
```

Pascal

```
SM32Write( Motor1, mcPower, 1 );
```

mcPosMode (RW)

Aufgabe: Betriebsmodus einstellen

Werte: mmMove, mmPos

Vorgabe: mmMove

Beschreibung:

Im Modus `mmMove` fährt der Motor mit der eingestellten Geschwindigkeit und Beschleunigung, Angaben `mcF` und `mcA`.

Im Modus `mmPos` fährt der Motor zu einer mit `mcGo` anzugebenden Position und hält dort an.

Bei Ausführung dieses Kommandos werden laufende Motoren auf Geschwindigkeit Null gebremst, hierauf wird bei
- `mmMove` die Geschwindigkeit mit `mcF` auf Null,
- `mmPos` die Geschwindigkeit mit `mcF` auf die Angabe bei `mcFmax` gesetzt.

Beispiel: Motor 1 in den Positioniermodus schalten

C

```
SM32Write( &Motor1, mcPosMode, mmPos );
```

Pascal

```
SM32Write( Motor1, mcPosMode, mmPos );
```

mcAbsRel (RW)

Aufgabe: Absoluten oder relativen Positioniermodus auswählen

Werte: `mmAbs`, `mmRel`

Vorgabe: `mmAbs`

Beschreibung:

Im Modus `mmAbs` fährt der Motor zu der mit `mcGo` angegebenen Position.

Im Modus `mmRel` verfährt der Motor um die mit `mcGo` angegebenen Strecke. Falls hierbei der Motor gerade ein anderes Fahrkommando ausführt, wird relativ zum gerade durchfahrenen Punkt positioniert.

Beispiel: Motor 1 in den Absolutmodus schalten

C

```
SM32Write( &Motor1, mcAbsRel, mmAbs );
```

Pascal

```
SM32Write( Motor1, mcAbsRel, mmAbs );
```

mcPosition (RW)

Aufgabe: Positionszähler auslesen oder setzen

Werte: -2'147'483'648 .. 2'147'483'647

Vorgabe: 0

Beschreibung:

- Durch Auslesen erhält man den gegenwärtigen Stand des Schrittzählers.
- Durch Beschreiben kann man ihn auf einen beliebigen Wert setzen.
  Ein mit `mcGo` angegebenes Fahrziel wird um dieselbe Distanz verstellt, daher ändert sich nichts an der Fahrstrecke, falls ein Motor soeben in einer Positionierfahrt unterwegs ist.

Beispiel: Schrittzähler auslesen

C

```
long Data;
SM32Read( &Motor1, mcPosition, &Data )
printf( "Position Motor1 ist %d\n" , Data );
```

Pascal

```
var Data :longint;
SM32Read( Motor1, mcPosition, Data );
writeln( 'Position Motor1 ist ' , Data );
```

Beispiel: Nullpunkt definieren

C

```
SM32Write( &Motor1, mcPosition,  0  );
```

Pascal

```
SM32Write( Motor1, mcPosition,  0  );
```

mcState (R)

Aufgabe: Status lesen

Werte: siehe Unten

Vorgabe: 0

Beschreibung:

mcState kann folgende Werte annehmen:

```
mstNone  =0 :  Motor steht
mstMove  =1 :  Motor läuft im Positioniermodus mmMove
mstPos   =3 :  Motor läuft im Positioniermodus mmPos
mstHome  =8 :  Motor führt Heimfahrt aus
mstBreak =9 :  Motor führt Schnellbremsung aus
```

Beispiel: Steht Motor 1 ?

C

```
long Data;

SM32Read( &Motor1, mcState, &Data );
if( Data==mstNone ) printf("Motor1 steht\n");
else                printf("Motor1 fährt\n");
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcState, Data );
if Data=mstNone then writeln('Motor1 steht')
else                 writeln('Motor1 fährt');
```

mcGo (RW)

Aufgabe: Zielfahrt

Werte: -2’147’483’648 .. 2’147’483’647

Vorgabe: --

Beschreibung:

Im Positioniermodus `mmPos` :

Bei `mcAbsRel=mmAbs` fährt der Motor zu der hier angegebenen Position. Falls der Motor gerade ein anderes Fahrkommando ausführt, wird das alte Fahrziel verworfen.

Bei `mcAbsRel=mmRel` verfährt der Motor um die hier angegebenen Strecke. Falls der Motor gerade ein anderes Fahrkommando ausführt, wird relativ zum gerade durchfahrenen Punkt positioniert.

Im Positioniermodus `mmMove` wird das Kommando ignoriert.

Beispiel: Bei `mcAbsRel=mmAbs` zum Nullpunkt fahren

C

```
SM32Write( &Motor1, mcGo, 0 );
```

Pascal

```
SM32Write(  Motor1, mcGo, 0 );
```

mcGoHome (W)

Aufgabe: Heimfahrt

Werte: -16000 .. 16000 Hz

Vorgabe: --

Beschreibung:

Ein Endschalter wird angefahren.
Der angegebene Wert bestimmt die Geschwindigkeit der Suchfahrt, das Vorzeichen die Suchrichtung.

Bei Erreichen des Endschalters wird mit verminderter Geschwindigkeit in entgegengesetzter Richtung gefahren bis der Endschalter wieder abfällt.
Der Schrittzähler wird nicht automatisch auf Null gesetzt, dies ist gegebenenfalls mit `mcPosition` vorzunehmen.

Zum Beschleunigen wird die mit `mcAmax` vorgenommene Einstellung verwendet.

Falls der betreffende Endschalter nicht mit `mcSwitchMode` aktiviert wurde bleibt `mcGoHome` wirkungslos.
Ein eventuell mit `mcSwitchMode` aktivierter Nothalt-Modus wird vorübergehend außer Kraft gesetzt.

Die Fahrt kann mit `mcGoHome` mit Geschwindigkeit 0 abgebrochen werden.
Auch alle anderen Kommandos brechen die Fahrt ab, können aber verloren gehen.

Beispiel: Mit 1000 Hz zum negativen Endschalter fahren, Nullpunkt setzen

C

```
long Data;

SM32Write( &Motor1, mcGoHome, -1000 );
do SM32Read( &Motor1, mcState, &Data ); while(Data!=0);
SM32Write( &Motor1, mcPosition, 0  );
```

Pascal

```
var Data :longint;

SM32Write( Motor1, mcGoHome, -1000 );
repeat SM32Read( Motor1, mcState, Data ) until Data=0;
SM32Write( Motor1, mcPosition, 0  );
```

mcBreak (W)

Aufgabe: Alle Vorgänge abbrechen, Motor schnell abbremsen

Werte: --

Vorgabe: --

Beschreibung:

Jeglicher Positioniervorgang oder Heimfahrt wird abgebrochen, der Motor wird sofort mit der bei `mcAmax` angegebenen Beschleunigung auf Null gebremst.

Beispiel: Motor 1 anhalten

C

```
SM32Write( &Motor1, mcBreak, 0 );
```

Pascal

```
SM32Write( Motor1, mcBreak, 0 );
```

mcFact (R)

Aufgabe: Gegenwärtige Geschwindigkeit lesen

Werte: -16000 .. 16000 Hz

Vorgabe: --

Beschreibung:

Vornehmlich für Anzeigezwecke: Gegenwärtige Geschwindigkeit lesen

Hinweis: Um abzufragen ob ein Positioniervorgang beendet ist, wird vorzugsweise `mcState` verwendet: eine unmittelbar auf `mcGo` folgende Abfrage mit `mcFact` liefert noch Null, falls der Beschleunigungsvorgang augenblicklich noch nicht begonnen hat.

Schrittgeschwindigkeiten werden, unabhängig von der Schrittweiteneinstellung, in "Vollschritten pro Sekunde" angegeben.

Beispiel: Gegenwärtige Geschwindigkeit drucken

C

```
long Data;

SM32Read( &Motor1, mcSysVersion, &Data );
printf("Geschwindigkeit: %d\n", Data );
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcSysVersion, Data );
writeln('Geschwindigkeit: ', Data );
```

mcPhase (R)

Aufgabe: Motorschrittwinkel lesen

Werte: 0 .. 255

Vorgabe: --

Beschreibung:

Der aktuelle Schrittwinkel wird ausgelesen, wobei die Werte 0,64,128,192 jeweils Vollschritten entsprechen.

Nutzen in besonders kritischen Aufbauten:
Beim manuellen Drehen eines Motors sind stabile und instabile Positionen zu erkennen.
Wenn man ermittelt, welcher Schrttwinkel einer stabilen Position entspricht, kann man vor dem Abschalten gezielt auf eine stabile Position fahren, um zu vermeiden, daß der Motor nach dem Abschalten unkontrolliert von selbst eine kleine Bewegung zur nächsten stabilen Position ausführt.

Beispiel: Aktuellen Motorschrittwinkel drucken

C

```
long Data;

SM32Read( &Motor1, mcPhase, &Data );
printf("Aktueller Schrittwinkel: %d\n", Data );
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcPhase, Data );
writeln('Aktueller Schrittwinkel: ', Data );
```

mcPowerMeter (R)

Aufgabe: Leistungsverbrauch des Motors in Milliwatt lesen

Werte: 0 .. 18 000

Vorgabe: --

Beschreibung:

Für Informations-/Anzeigezwecke:
gegenwärtigen Leistungsverbrauch des Motors in Milliwatt lesen.

Der gelieferte Wert ist eine grobe Abschätzung aus internen Daten der Steuerung und daher für Regelungszwecke eher ungeeignet.

Beispiel: Motorleistung anzeigen

C

```
long Data;

SM32Read( &Motor1, mcPowerMeter, &Data );
printf("Leistung Motor1 : ungefähr %d mW\n", Data );
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcPowerMeter, Data );
writeln('Leistung Motor1 : ungefähr ', Data, ' mW' );
```

`mcU` (RW M)

Aufgabe: Motor-Nennspannung in 1/10 Volt einstellen

Werte: 0 .. 120

Vorgabe: 0

Beschreibung:

- Wenn der Motorstrom bei der angegebenen Spannung so groß wird, daß das Netzteil ihn nicht liefern kann, wird die Spannung intern heruntergeregelt, bei mcError erscheint die Fehlermeldung `meIM(x)`.

- Bei höheren Drehzahlen stellt sich eine höhere Spannung ein als angegeben, da an der Motorinduktivität ein zusätzlicher Spannungsabfall auftritt.

Beispiel: Spannung von Motor 1 auf 6.0 Volt setzen

C

```
SM32Write( &Motor1, mcU, 60 );
```

Pascal

```
SM32Write( Motor1, mcU, 60 );
```

mcUhold (RW M)

Aufgabe: Motor-Haltespannung in Prozent der Nennspannung einstellen

Werte: 0 .. 100

Vorgabe: 0

Beschreibung:

Wenn der Motor zwanzig Millisekunden nicht gelaufen ist, wird die Spannung auf diesen Wert abgesenkt, um unnötigen Leistungsverbrauch zu minimieren.

Beispiel: Haltespannung von Motor 1 auf 30% der Nennspannung setzen

C

```
SM32Write( &Motor1, mcUhold, 30 );
```

Pascal

```
SM32Write( Motor1, mcUhold, 30 );
```

mcStepWidth (RW M)

Aufgabe: Schrittweite und Drehrichtung festlegen

Werte: ± 1, 2, 4, 8, 16, 32, 64 Mikroschritte pro Vollschritt

Vorgabe: 64 Mikroschritte pro Vollschritt

Beschreibung:

- Die SM32 arbeitet intern immer im 1/64 -Mikroschrittbetrieb. Falls diese feine Schrittauflösung nicht benötigt wird, ergeben sich dadurch unpraktikable Positionsangaben. Mit mcStepwidth kann ein Umrechnungsfaktor eingestellt werden, so daß nach aussen hin die Schrittauflösung verringert erscheint.
- Durch die Angabe von negativen Werten kann die Drehrichtung des Motors umgedreht werden.

Beispiel: Drehrichtung Motor1 umdrehen

C

```
SM32Write( &Motor1, mcStepWidth, -64 );
```

Pascal

```
SM32Write( Motor1, mcStepWidth, -64 );
```

mcAmax (RW M)

Aufgabe: Maximalbeschleunigung setzen

Werte: 1 .. 1000 Hz / Millisekunde

Vorgabe: 25 Hz / Millisekunde

Beschreibung:

- Zur Absicherung gegen Bedienungsfehler werden mit `mcA` angegebene Werte (unter Berücksichtigung des Vorzeichens) auf den hier festgelegten Wert begrenzt.
- Beim Ansprechen eines Endschalters und bei Auslösen von `mcBreak` wird mit der hier angegebenen Beschleunigung gebremst.

Falls die eingestellte Beschleunigung zu hoch ist, kann der Motor den erzeugten Schrittimpulsen nicht folgen und gerät ausser Tritt; die gezählte Schrittposition stimmt nicht mehr mit der tatsächlichen überein.

Beschleunigungen werden, unabhängig von der Schrittweiteneinstellung, in "Vollschritte/Sekunde pro Millisekunde" angegeben.

Beispiel: Maximalbeschleunigung Motor1 auf 10 Hz/ms setzen

C

```
SM32Write( &Motor1, mcAmax, 10 );
```

Pascal

```
SM32Write( Motor1, mcAmax, 10 );
```

mcFmax (RW M)

Aufgabe: Maximalgeschwindigkeit setzen

Werte: 1 .. 16000 Hz

Vorgabe: 16000 Hz

Beschreibung:

- Zur Absicherung gegen Bedienungsfehler werden mit `mcF` angegebene Werte (unter Berücksichtigung des Vorzeichens) auf den hier festgelegten Wert begrenzt.
- Beim Umschalten in den Positioniermodus wird die Sollgeschwindigkeit auf den hier angegebenen Wert eingestellt.

Schrittgeschwindigkeiten werden, unabhängig von der Schrittweiteneinstellung, in "Vollschritten pro Sekunde" angegeben.

Beispiel: Maximalgeschwindigkeit Motor1 auf 2000 Hz setzen

C

```
SM32Write( &Motor1, mcFmax, 2000 );
```

Pascal

```
SM32Write( Motor1, mcFmax, 2000 );
```

mcA (RW M)

Aufgabe: Beschleunigung wählen

Werte: 1 .. 1000 Hz / Millisekunde

Vorgabe: 25 Hz / Millisekunde

Beschreibung:

Stellt den Beschleunigungswert des Schrittgenerators ein.

Zum Bremsen (negative Beschleunigung) wird der negative Wert verwendet.

Beschleunigungen werden, unabhängig von der Schrittweiteneinstellung, in "Vollschritte/Sekunde pro Millisekunde" angegeben.

Beispiel: Beschleunigung Motor1 auf 10 Hz/ms setzen

C

```
SM32Write( &Motor1, mcA, 10 );
```

Pascal

```
SM32Write(  Motor1, mcA, 10 );
```

mcF (RW M)

Aufgabe: Geschwindigkeit wählen

Werte: -16000 .. 16000 Hz

Vorgabe: 0 Hz

Beschreibung:

- Im Positioniermodus `mmMove` :

  Der Motor beschleunigt sofort mit der bei `mcA` eingestellten Beschleunigung auf die hier angegebene Geschwindigkeit.
  Negative Werte lassen den Motor rückwärts drehen, Angabe von Null führt zu Stillstand.

- Im Positioniermodus `mmPos` :

  Die hier angegebene Geschwindigkeit wird als Sollwert für Positionierfahrten verwendet.
  Auch der Sollwert eines momentan ablaufenden Positioniervorgangs wird gegebenenfalls geändert.
  Zulässige Werte : 1 .. 16000 Hz .

Übergebene Geschwindigkeitsangaben werden auf den mit `mcFmax` gesetzten Grenzwert begrenzt.

Schrittgeschwindigkeiten werden, unabhängig von der Schrittweiteneinstellung, in "Vollschritten pro Sekunde" angegeben.

Beispiel: Im Positioniermodus `mmMove` auf 2000 Hz beschleunigen
C

```
SM32Write( &Motor1, mcF, 2000 );
```

Pascal

```
SM32Write(  Motor1, mcF, 2000 );
```

Beispiel: Im Positioniermodus `mmPos` Sollgeschwindigkeit auf 2000 Hz setzen
C

```
SM32Write( &Motor1, mcF, 2000 );
```

Pascal

```
SM32Write(  Motor1, mcF, 2000 );
```

mcFg (RW M)

Aufgabe: Motoreckfrequenz angeben

Werte: 100 .. 500 Hz

Vorgabe: 380 Hz

Beschreibung:

Diese Angabe dürfte selten gebraucht werden; hiermit wird zur Unterstützung der Regelung angegeben, ab welcher Schrittfrequenz der Spannungsbedarf aufgrund der Motorinduktivität stark erhöht ist.
Die Voreinstellung ist praktisch immer der beste Kompromiss, experimentell könnte man versuchen, bei mittleren Drehzahlen noch ein wenig mehr Drehmoment zu erreichen, indem man den Wert so wählt, daß sich mit unbelastetem Motor bei 500Hz der selbe Leistungsverbrauch ergibt wie bei 1Hz .

Beispiel: Motoreckfrequenz auf 200 Hz einstellen

C

```
SM32Write( &Motor1, mcFg, 200 );
```

Pascal

```
SM32Write( Motor1, mcFg, 200 );
```

Schalter

`mcSwitchMode` (RW M)

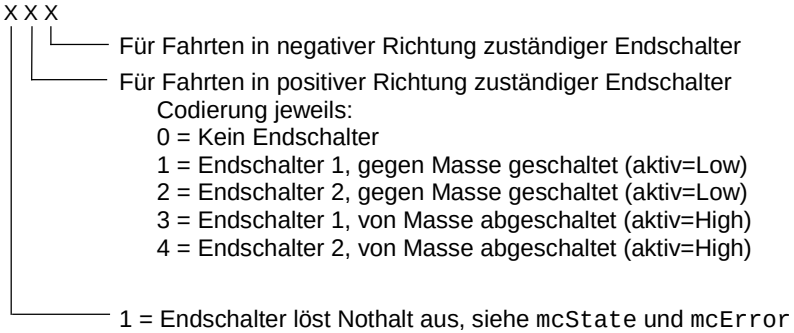
Aufgabe: Endschalter konfigurieren

Werte: 0 .. 121

Vorgabe: 0 = Endschalter ignorieren

Beschreibung:

Durch Angabe einer dreistelligen Dezimalzahl werden
die zum Motor gehörigen Endschalter konfiguriert:



Beispiel:
In positiver Richtung Endschalter 2, Schließer gegen Masse
In negativer Richtung Endschalter 1, Schließer gegen Masse

C

```
SM32Write( &Motor1, mcSwitchMode, 21 );
```

Pascal

```
SM32Write( Motor1, mcSwitchMode, 21 );
```

mcSwitches (R)

Aufgabe: Gegenwärtige Schalterzustände lesen

Werte: 0 .. (1+2)

Vorgabe: --

Beschreibung:

Gegenwärtige Schalterzustände lesen.

Werte:
1: Schalter 1 = high,
2: Schalter 2 = high,
3: beide Schalter high.

Der erhaltene Wert ist unabhängig von der Einstellung mit mcSwitchMode:

Beispiel: Schalter anzeigen

C

```
long Data;

SM32Read( &Motor1, mcSwitches, &Data );
printf("Schalter 1: ");
if(Data & 1) printf("high\n"); else printf("low\n");
printf("Schalter 2: ");
if(Data & 2) printf("high\n"); else printf("low\n");
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcSwicthes, Data );
write('Schalter 1: ');
if Data and 1<>0 then writeln('high') else writeln('low');
write('Schalter 2: ');
if Data and 2<>0 then writeln('high') else writeln('low');
```

mcSwiPos (R)

Aufgabe: Position bei letztem Schaltpunkt lesen

Werte: -2’147’483’648 .. 2’147’483’647

Vorgabe: --

Beschreibung:

Bei jedem Umschalten eines zum Motor gehörenden Schalters wird die Motorposition abgespeichert, der erfasste Wert kann mit diesem Befehl ausgelesen werden.
mcSwiPos arbeitet unabhängig von der Einstellung mit `mcSwitchMode`.

Der Zeitpunkt wird nur auf etwa 1/10 000 Sekunde genau erfasst, bei höheren Geschwindigkeiten ergibt sich ein entsprechener Ablesefehler.
Bei 100 Hz liegt der Fehler unterhalb 1/64 Schritt.

Beispiel: Letzte Schaltposition anzeigen

C

```
long Data;

SM32Read( &Motor1, mcSwiPos, &Data );
printf("Letzte Schaltposition: %d\n", Data );
```

Pascal

```
var Data :longint;

SM32Read( Motor1, mcPowerMeter, Data );
writeln(’Letzte Schaltposition: ’, Data );
```

`mcStoreCfg (RWS )`

Aufgabe: Konfiguration speichern

Werte: siehe unten

Vorgabe: —

Beschreibung:

Die gegenwärtige Konfiguration wird im Programmspeicher der SM32 abgelegt und überschreibt die werksseitigen Voreinstellungen.
Betroffen sind die Einstellungen von

F, Fmax,	U, Uhold,	SwitchMode,
A, Amax,	StepWidth,	Fg.

- Die Verarbeitung dieses Kommandos dauert typisch etwa eine zehntel, fallweise aber auch bis zu 30 Sekunden; während dieser Zeit ist die SM32 nicht ansprechbar.
- Um eine versehentliche Fehlbedienung auszuschliessen, wird das Kommando nur erkannt wenn der 'magische Wert' 87654321 übergeben wird.
- Alle Motoren müssen ausgeschaltet sein.

Nach korrekter Ausführung des Befehls wird 0 zurückgegeben, andernfalls ein negativer Wert.

Beispiel: Konfiguration speichern

C

```
SM32Write( &Motor1, mcStoreCfg, 87654321 );
```

Pascal

```
SM32Write( Motor1, mcStoreCfg, 87654321 );
```

Wichtiger Hinweis:
Dieses Kommando ist nur für gelegentliche Anwendung vorgesehen, etwa bei Inbetriebnahme einer Anlage oder Wartungsarbeiten.

- Der Programmspeicher läßt zuverlässig nur einige tausend Umprogrammierungen zu.
- Bei Ausfall der Stromversorgung oder Reset des Rechners während Ausführung des Kommandos kann eine Fehlprogrammierung des Programmspeichers nicht ausgeschlossen werden.

mcNone ( )

Aufgabe: Leerer Befehl

Werte: —

Vorgabe: —

Beschreibung:

Der Konstantenwert dieses Befehls ist Null.
Er kann in Programmen zu Kennzeichnungszwecken verwendet werden.

Beispiel:

C

```
char LetztesKommando;
.
.
.
  LetztesKommando = mcNone;
.
.
.
  if( LetztesKommando==mcNone ) {
    printf("Es wurde noch nichts an die SM32 uebertragen");
  };
```

Pascal

```
var LetztesKommando :byte;
.
.
.
  LetztesKommando := mcNone;
.
.
.
  if LetztesKommando=mcNone then begin
    write("Es wurde noch nichts an die SM32 uebertragen");
  end;
```

Steckerbelegung

Stromversorgungsanschluß

Zur Versorgung der SM32 mit Betriebsstrom muß am Stromversorgungsanschluß oben rechts ein Stromversorgungsstecker angeschlossen werden (Vierpolig, Kabelfarben: rot-schwarz-schwarz-gelb).
Meist ist im Rechner noch ein derartiger Anschluß frei, gegebenenfalls kann ein handelsübliches Abzweigstück eingesetzt werden.

Motoranschlußbuchse

Signalname	Sub-D - Anschluß		Signalname
Motor1 B	1		
		14	Motor 1 Schalter 2
Motor1 A	2		
		15	Motor 1 Schalter 1
Motor1 B'	3		
		16	Masse
Motor1 A'	4		
		17	AuxIn 0
Motor2 B	5		
		18	Motor 2 Schalter 2
Motor2 A	6		
		19	Motor 2 Schalter 1
Motor2 B'	7		
		20	Masse
Motor2 A'	8		
		21	AuxIn 1
Motor3 B	9		
		22	Motor 3 Schalter 2
Motor3 A	10		
		23	Motor 3 Schalter 1
Motor3 B'	11		
		24	Masse
Motor3 A'	12		
		25	AuxIn 2
5V 100mA (NOTAUS)	13		

Zur Einhaltung der CE-Norm bezüglich Funkstörungen ist ein einfaches abgeschirmtes Kabel erforderlich.

Motoranschlüsse

- Motoren mit 4 Anschlüssen:

```
A --#######-- A'        B --#######-- B'
```

- Bei Sechsleiter-Motoren wird folgender Anschluß empfohlen:

```
       A'                      B'
       |                       |
A --#######-- frei      B --#######-- frei
```

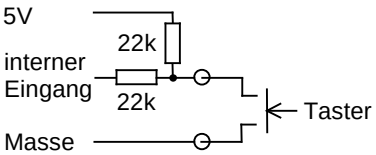
- Bei Achtleiter-Motoren sollte man je zwei Spulen parallelschalten, so daß im Effekt ein Vierleitermotor vorliegt.

```
A -+-#######-+- A'      B -+-#######-+- B'
  |         |             |         |
C  +-#######-+  C'      D  +-#######-+  D'
```

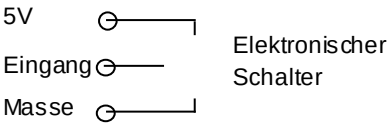
- Fünfleiter-Motoren sind nicht für den Betrieb mit der SM32 geeignet.

Endschalter- und zusätzliche Digitaleingänge

Diese Eingänge arbeiten mit TTL-Pegel (Low=0.4V, High=2V) und sind intern mit Pullup- und Schutzwiderständen versehen.
Im einfachsten Fall ist lediglich ein Taster gegen Masse anzuschliessen:

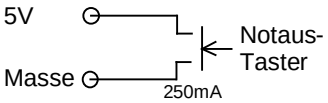


Auch eine Beschaltung mit elektronischen Schaltern (Näherungsschaltern) ist möglich:



Notaus-Taster

Der 5V-Ausgang arbeitet auch als Notaus-Eingang: Bei Kurzschluß gegen Masse erscheint ein "me5V"-Fehler und alle Motorn halten an.



Kompatibilität zu SM30

Hardware:

- Kurze PCI 2.2-Karte statt XT/AT Bus.
- Nur für bipolare Motoren ausgelegt (4/6/8-Draht).
- Höhere Motorströme (1,8 A kontinuierlich).
- Höhere Versorgungsleistung (3,6 W kontinuierlich).
- Anschluss eines Floppystromversorgungssteckers erforderlich.
- Drei zusätzliche Digitaleingänge.
- Hilfsspannungsausgang fest auf 5V eingestellt.

Software:

- Nahezu vollständig kompatibel mit SM32 Softwareversion 2.0;
  SM30 Softwareversion 1.0 wird nicht unterstützt.
- Alle Funktionen und Konstanten von SM30... auf SM32... umbenannt.
- SM32Init nimmt Kartennummer statt Portadresse.
- Geänderte Kommandos:
  - mc510V : entfernt.
  - mcAuxIn : Neuer Befehl zum Lesen der Digitaleingänge.
  - mcStoreCfg : Neuer Befehl zum Abspeichern der Konfiguration.

Treiber / DLLs:

- Windows:
  Neue Treiber für Windows 98 / NT / 2000.
  PCISM32.DLL statt PCSM30.DLL.
  Neue PlxApi.DLL..
- DOS, sm32.h /.c und uSM32.pas:
  Erhebliche Änderungen aufgrund Erfordernissen der PCI-Schnittstelle.

Technische Daten

- Steckkarte für PCI –Steckplatz

- Der vom PC-Netzteil entnommene Versorgungsstrom wird auf der Karte aktiv auf 3 Ampere begrenzt.

- Gesamt-Anschlussleistung : 36 W, mit etwas Reserve zum Beispiel
  3 * 12V,0.8A oder 3 * 6V,1.6A oder 3* 4V,2.4A .

- Motorstrom max. 1.8 A / Phase im Dauerbetrieb.

- Motorspannung per Software wählbar in 0.1V-Schritten bis 12V.

- Stromabsenkung im Stillstand durch Software einstellbar.

- Schrittauflösung 1/64 Schritt.

- Schrittfrequenz intern Vollschritt bis 16000 Hz, entsprechend 1.024.000 Hz bei 1/64 Mikroschritt, am Motor nur wegen induktivem Spannungsabfall an der Motorinduktivität begrenzt.

- Eigener Prozessor für Schritterzeugung und Positionierfahrten.

- Alle Motoren unabhängig steuerbar.

- Je Motor zwei Eingänge für (gegen Masse schaltende) Endschalter, per Software als Schließer / Öffner / Unausgewertet konfigurierbar.

- Ausgang 5 Volt,100mA zur Versorgung elektronischer Endschalter. Dient auch als NOTAUS-Eingang.

Befehlsindex

Legende: R=lesbar, W=schreibbar, S=Systemweit wirksam, M=speicherbar