Anhang B

Ergänzungen zur Motorendokumentation

Der folgende Anhang enthält Zusammenfassungen, Übersetzungen und – als Reverse Engeneering-Ergebnisse – Ergänzungen zur Dokumentation der Motorcontroller, die von besonderem Interesse für das Verständnis sowohl der Motoren- als auch der Simulationskomponente sind. Die vollständige Dokumentation liegt in [PI93a] und [PI93b] vor.

B.1 Zum C812er Controller

B.1.1 Kommunikation

Die ISA-Bus-Kommunikation mit dem C812er Controller läuft über einen *dual port RAM* oder Direktspeicher genannten Speicherbereich, der sowohl vom Controller als auch von der Software über den PC-Bus angesprochen werden kann. Die Basisadresse dieses Speicherbereiches wird über Dip-Schalter auf der Controllerkarte eingestellt. Üblicherweise liegt sie bei `0xD8000`. Die folgenden Abbildungen geben Speicheradressen stets als Offset zu dieser Adresse an.

Die Hauptkommunikation wird über Handshake-Flags und Mailboxadressen synchonisiert. Daneben gibt es noch den unsynchronisierten *direct access*.

Lesen (synchron) Zum Lesen eines Bytes vom C812er Controller muss die Steuersoftware darauf warten, dass das *data available flag* des Statusregisters gesetzt wurde, und kann dann von der *Leseadresse* ein Byte lesen (s. Abb. B.1).

Schreiben (synchron) Um ein Byte an den Controller zu senden, muss die Software darauf warten, dass das *busy flag* des Statusregisters zurückgesetzt wurde. Dann wird das zu schreibende Byte an die *Mailboxadresse 1* geschrieben und als nächstes an die *Mailboxadresse 2*; erst das Schreiben eines identischen Bytes an die zweite Adresse führt zur Interpretation durch den Controller (s. Abb. B.1).

direct access Neben der eben beschriebenen synchronisierten Kommunikation gibt es die *direct access*-Kommunikation. Dabei werden einige Register sowohl

Offset	Bedeutung
0x03FC	Schreibadresse (Mailboxadresse 1)
0x03FE	Leseadresse
0x03FF	Schreibadresse (Mailboxadresse 2)
0x0800	Statusregister Byte 0: busy flag Byte 1: data available flag

Abbildung B.1: dual-port-RAM des C812 (Teil 1, I/O, Status)

vom Controller als von der Software unsynchronisiert gelesen oder beschrieben. Vom XCTL-System wird diese problematische Kommunikation nur für das *position register* (Encoder-Position) und das *failure register* (Positionfehler) verwendet und fast ausschließlich lesend. Die Abbildungen B.2 und B.3 zeigen die Adressen dieser Register. (Weitere *direct access*-Register listet theoretisch das Kommando `HE6` auf.)

Offset	Bedeutung
0x006C	Positionsfehler Motor 1, Byte 0
0x006D	Positionsfehler Motor 2, Byte 0
0x006E	Positionsfehler Motor 3, Byte 0
0x006F	Positionsfehler Motor 4, Byte 0
0x0070	Positionsfehler Motor 1, Byte 1
0x0071	Positionsfehler Motor 2, Byte 1
0x0072	Positionsfehler Motor 3, Byte 1
0x0073	Positionsfehler Motor 4, Byte 1
0x0074	Positionsfehler Motor 1, Byte 2
0x0075	Positionsfehler Motor 2, Byte 2
0x0076	Positionsfehler Motor 3, Byte 2
0x0077	Positionsfehler Motor 4, Byte 2
0x0078	Positionsfehler Motor 1, Byte 3
0x0079	Positionsfehler Motor 2, Byte 3
0x007A	Positionsfehler Motor 3, Byte 3
0x007B	Positionsfehler Motor 4, Byte 3

Abbildung B.2: dual-port-RAM des C812 (Teil 2, *direct access*, Positionsfehler)

Offset	Bedeutung	
0x010A	Position	Motor 1, Byte 0
0x010B	Position	Motor 2, Byte 0
0x010C	Position	Motor 3, Byte 0
0x010D	Position	Motor 4, Byte 0
0x010E	Position	Motor 1, Byte 1
0x010F	Position	Motor 2, Byte 1
0x0110	Position	Motor 3, Byte 1
0x0111	Position	Motor 4, Byte 1
0x0112	Position	Motor 1, Byte 2
0x0113	Position	Motor 2, Byte 2
0x0114	Position	Motor 3, Byte 2
0x0115	Position	Motor 4, Byte 2
0x0116	Position	Motor 1, Byte 3
0x0117	Position	Motor 2, Byte 3
0x0118	Position	Motor 3, Byte 3
0x0119	Position	Motor 4, Byte 3

Abbildung B.3: dual-port-RAM des C812 (Teil 3, *direct access*, Motorposition)

B.1.2 Kommandos (allgemein)

Der C812er Controller wird über Kommandos in einer einfachen formalen Sprache gesteuert. Es gibt Einzelkommandos, zusammengesetzte Kommandos und Makrokommandos sowie für einige Einzelkommandos Abkürzungen. Die Makrokommandos werden hier nicht weiter behandelt, da sie weder von der Motorenkomponente verwendet noch von der Simulationskomponente zur Verfügung gestellt werden. Ein *Einzelkommando* hat folgende Struktur:

```
{<a>}<CMD>{<n>}<CR>
```

`a` Die Achse (d.h. der Motor) auf die sich der Befehl bezieht. Bei Kommandos, die sich auf den Controller beziehen, ist die Achsenangabe unzulässig. Einige motorbezogenen Kommandos erlauben die Achsenangabe wegzulassen; dann bezieht sich das Kommando auf alle Achsen. Wird eine Achse angegeben ist eine Zahl von 1 bis 4 zulässig.

`CMD` Zweibuchstabencode für das Kommando.

`n` Parameter für das Kommando. Ganze Zahl. Nicht alle Kommandos haben Parameter.

`CR` *carriage return*-Zeichen.

Bit	Bedeutung
0	*on taget flag*: 1 falls Ist- gleich Sollposition, bzw. falls Istposition innerhalb des *death band* liegt, also „beinahe“ gleich der Sollposition ist; 0 sonst
1	*limit flag*: 1 falls ein Endlagenschalter erreicht wurde; wird durch nächstes Bewegungskommando zurückgesetzt
2	*motor off flag*: 1 falls Motor im OFF-Zusand ist; 0 falls Motor im ON-Zustand ist
3	
4	*command error flag*: 1 falls das letzte Kommando fehlerhaft war
5	
6	
7	
8	

Abbildung B.4: Statusregister für einen Motor des C812

Ein *zusammengesetztes Kommando* besteht aus beliebig vielen Einzelkommandos, die durch Kommata getrennt sind. Es wird durch ein *carriage return* abgeschlossen. Ein zusammengesetztes Kommando hat also folgende Struktur, wobei `SNGCMD` für ein Einzelkommando steht:

```
<SNGCMD>{,<SNGCMD>}<CR>
```

Einige Beispiele für Kommandos:

`1MR2000`	bewege Motor 1 um 2000 Schritte nach rechts
`AB`	stoppe alle 4 Motoren
`EF`	schalte das Echo aus
`4SA1000,4SD1000`	setzte Be- und Entschleunigung für Motor 4 auf 1000

Abb. B.5 listet die Einzelkommandos auf, die von der Motorenkomponente verwendet werden. Für eine vollständige Liste s. [PI93a].

B.1.3 Ausgaben des Controllers (Reportkommandos)

Einige Kommandos des C812er Controllers liefern Informationen über den Controller bzw. die Motoren – die sogen. *Reportkommandos*. Der Ausgabestring, der wie oben beschrieben byteweise ausgelesen werden muss, hat folgende Struktur:

```
 <A><LBL><VAL><CRLF>
{<A><LBL><VAL><CRLF>}<ETX1><ETX2>
```

Kommando	Erläuterung
AB	*Abort motion*; stoppt den Motor abrupt.
CF	*Channel off*
CN	*Channel on*
CP	*Calculate point*; berechnet Entschleunigungsrampe.
CT	*Channel out*
DB	*Dead band*
DH	*Define home*; setzt die aktuelle Position auf Null.
DM	*Decimal mode*; Dezimalmode ein; Ausgaben werden als Dezimalzahlen dargestellt.
EF	*Echo off*
HE	*Help*
LS	*Limit set*; Anzahl der Motorschritte, die der Motor zurückgefahren wird, wenn er eine Endlagensignal empfangen wird.
MA	*Move absolute*; bewegt den Motor zu einer Position.
MN	*Motor off*
MR	*Move relative*; bewegt den Motor un eine Distanz.
SA	*Set acceleration*; setzt die Beschleunigung.
SD	*Set deceleration*; setzt die Entschleunigung.
SE	*Set dynamic gain*
SG	*Set gain*
SP	*Set point*; setzt den Entschleunigungspunkt.
SQ	*Set Torque*
SV	*Set Velocity*; setzt die Geschwindigkeit.
TE	*Tell error*; liefert den Positionsfehler, d.h. die Abweichung von der Soll-Position.
TP	*Tell position*; liefert die aktuele Position.
TS	*Tell status*; liefert Status des Motors.

Abbildung B.5: Kommandos des C812 die von der Motorenkomponente verwendet werden

`A` Achse. Zweistellige Dezimalzahl von 1 bis 4 die angibt zu welcher Achse die Information gehört.

`LBL` Label. Buchstabe der angibt welche Information gerade reportiert wird (z.B. `T` für die Zielposition; für Details s. Dokumentation der Kommandos).

`VAL` Value. Der reportierte Wert. Je nachdem in welchem Modus der Controller sich gerade befindet eine dezimale oder eine hexadezimale ganze Zahl, eventuell mit Vorzeichen.

`CRLF` Wertendemarke. Ein *carriage return* und ein *new line* Zeichen. Ein Reportkommando liefert entweder für genau einen Motor einen Wert oder für alle vier Motoren.

`ETX1` Kommandoendemarke. Programmiertes *end of text*-Zeichen, das am Ende der Ausgabe eines Reportkommandos steht; d.h. bei zusammengesetzten Kommandos erscheint dieses Zeichen mehrfach und trennt die verschiedenen Ausgaben.

`ETX2` Kommandosequenzendemarke. Programmiertes *end of text*-Zeichen, das am Ende der Ausgabe einer Komamndosequenz steht.

Einige Beispiele für Reports:
Für Einzelkommando mit Achsenangabe: `1TP`.

```
01P0000000001{CRLF}{ETX}{ETX}
```

Für Einzelkommando ohne Achsenangabe: `TP`.

```
01P0000000001{CRLF}
02P0000001000{CRLF}
03P0000000232{CRLF}
04P0000000000{CRLF}{ETX}{ETX}
```

Für zusammengesetztes Kommando: `1TP,1TT,1TE`.

```
01P0000000100{CRLF}{ETX}
01T0000001000{CRLF}{ETX}
01E0000000900{CRLF}{ETX}{ETX}
```

B.2 Zum C832er Controller

B.2.1 Kommunikation

Die Kommunikation mit dem C832er Controller wird ausschließlich über ein Adressregister und ein Datenregister geführt. Die Adresse des Adressregisters kann über Dip-Schalter auf der Controllerkarte eingestellt werden. Das Datenregister liegt stets an der dem Adressregister folgenden Adresse. Typischerweise liegt das Adressregister bei 0x210 und damit das Datenregister bei 0x211.

Um vom Controller ein Byte zu lesen oder eines an ihn zu senden, muss als erstes ein Byte ins Adressregister geschrieben werden, das angibt, was als nächstes gelesen oder geschrieben werden soll. Abb. B.7 zeigt die Belegungen der

Bits 0 - 2 und deren Bedeutung. Wird z.B. ein Byte mit 000 in den niedrigsten 3 Bits ins Adressregister geschrieben, bedeutet dies, dass die nächste Leseoperation vom Datenregister das Statusregister des Motors 1 liefert, bzw. die nächste Schreiboperation auf das Datenregister ein Kommando an den Motor 1 sendet. Wird dagegen ein auf 001 endendes Byte in das Adressregister geschrieben, bezieht sich die nächste Lese- oder Schreiboperation auf das Datenregister des ersten Motors. Analoges gilt für den Motor 2. Außerdem kann auf diese Weise das Interruptregister des Controllers ausgelesen werden, welches Informationen über die Quelle des letzten vom Controller ausgelösten Interrupts und über die Endlagenschalter enthält (s. Abb. B.8).

Bevor Daten gelesen oder geschrieben bzw. Kommandos gesendet werden können, muss getestet werden, ob das *busy bit* (Bit 0) des Statusregisters zurückgesetzt ist.

Die Struktur der Kommandos des C832er Controllers ist leider nicht so klar durch eine formale Sprache beschrieben, wie die des C812er Controllers. Jedes Kommando wird durch ein *command code* identifiziert – ein 8bit-Integerwert. Einige der Kommandos erwarten eine bestimmte Anzahl von geschriebenen Bytes als Parameter und einige stellen eine Bestimmte Anzahl von Bytes zum Auslesen zur Verfügung. Eine Liste der relevanten Motorkommandos zeigt Abb. B.6.

Beispiel Als Erklärung des grundlegenden Prinzips der Kommunikation soll hier ein in C++-ähnlichem Pseudocode geschriebener Bespielablauf dienen. Die Zahlen sind in Binärdarstellung gegeben. `outp` steht für eine Schreiboperation und `inp` für eine Leseoperation. `ADDR_REG` ist das Adressregister und `DATA_REG` das Datenregister.

```
 1. outp(ADDR_REG, 00000010); // Kommandoregister Motor 2 aktiv
 2. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
 3. outp(DATA_REG, 00011111); // 0x1F, LTRJ

 4. outp(ADDR_REG, 00000010); // Kommandoregister Motor 2 aktiv
 5. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
 6. outp(ADDR_REG, 00000011); // Datenregister Motor 2 aktiv
 7. outp(DATA_REG, 00000000); // Byte 1 des control word
 8. outp(DATA_REG, 00000010); // Byte 0 des control word

 9. outp(ADDR_REG, 00000010); // Kommandoregister Motor 2 aktiv
10. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
11. outp(ADDR_REG, 00000011); // Datenregister Motor 2 aktiv
12. outp(DATA_REG, 00000000); // Byte 3 des 1. Parameters
13. outp(DATA_REG, 00111101); // Byte 2 des 1. Parameters

14. outp(ADDR_REG, 00000010); // Kommandoregister Motor 2 aktiv
15. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
16. outp(ADDR_REG, 00000011); // Datenregister Motor 2 aktiv
17. outp(DATA_REG, 00001001); // Byte 1 des 1. Parameters
18. outp(DATA_REG, 00000000); // Byte 0 des 1. Parameters

19. outp(ADDR_REG, 00000010); // Kommandoregister Motor 2 aktiv
20. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
```

```
21. outp(DATA_REG, 00000001); // 0x1, STT
```

Im ersten Teil des Beispiels soll der Motor 2 an eine absolute Position gefahren werden. Dazu wird in Zeile 1 und 2 (und ebenso 4 und 5, 9 und 10 ...) zuerst das Kommandoregister des Motors 2 aktiviert, damit als nächstes das Statusregister ausgelesen werden und dessen *busy*-Byte getestet werden kann. Ist dieses zurückgesetzt, darf geschrieben werden. In Zeile 3 wird dann der *command code* des *load trajectory*-Kommandos gesendet.

Dieses Kommando benötigt als Parameter ein 16bit-*controll word*, das angibt wie eventuell folgende Parameter zu interpretieren sind. In diesem Beispiel ist Bit 1 des *control words* gesetzt, d.h. dass ein weiterer 32-bit Parameter für die Zielposition folgt. In Zeile 6 wird daher das Datenregister des Motors 2 aktiviert und in Zeile 7 und 8 werden die beiden Bytes des *control words* geschrieben, zuerst das höherwertige, dann das niederwertige Byte. Nach demselben Schema werden dann Byte 3 bis 0 des Positionsparameters geschrieben, der den Wert $4000000_{10} = 11110100001001000000000_2$ hat. In Zeile 19 bis 21 wird dann das *start motion*-Kommando gesendet, das die Parameter der vorhergehenden *load trajectory*-Kommandos auswertet und die Bewegung auslöst.

```
22. outp(ADDR_REG, 00000010);
23. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
24. outp(DATA_REG, 00001010);    // 0xA; RDRP

25. union { INT32 value, BYTE byte[4] } pos;

26. outp(ADDR_REG, 00000010);
27. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
28. outp(ADDR_REG, 00000011);    // Datenregister Motor 2 aktiv

29. pos.byte[3] = inp(DATA_REG); // Byte 3 des Ergebnisses
30. pos.byte[2] = inp(DATA_REG); // Byte 2 des Ergebnisses

31. outp(ADDR_REG, 00000010);
32. do { status = inp(DATA_REG) } while (bit 0 in status gesetzt);
33. outp(ADDR_REG, 00000011);    // Datenregister Motor 2 aktiv

34. pos.byte[1] = inp(DATA_REG); // Byte 1 des Ergebnisses
35. pos.byte[0] = inp(DATA_REG); // Byte 0 des Ergebnisses
```

Im zweiten Teil des Beispiels soll die aktuelle Motorposition des Motors 2 ausgelesen werden. Dazu wird in Zeile 22 bis 24 analog zur bisherigen Vorgehensweise, das *read real position*-Kommando gesendet. Danach werden nacheinander Byte 3 bis Byte 0 des gelieferten Positionswertes gelesen und in der Variablen `pos` gespeichert.

B.2.2 Steuermodi

Der C832er Controller unterstützt zwei Steuermodi, a) den *position mode*, bei dem eine Bewegung durch Vorgabe einer Sollposition, die von der Istposition abweicht, ausgelöst wird und die Bewegung bei Erreichen dieser Zielposition beendet wird, und b) den *velocity mode*, bei dem eine Bewegung durch Vorgabe

Abk.	**Erläuterung**
LTRJ	*Load Trajectory Parameters*; Laden der Bewegungsparameter; durch wiederholte Anwendung können mehrere Parameter gesendet werden; die Parameter werden erst durch ein folgendes STT-Kommando verwendet
STT	*Start Motion*; wendet die Parameter der vorangegangenen LTRJ-Kommandos an
LFIL	*Load Filter Parameters*; Laden der Koeffizienten der Gleichung des PID-Filters (proportional integral derivative filter), der vom Algorithmus zur Bestimmung des Motorstroms verwendet wird; die Koeffizienten werden erst nach dem nächsten UDF-Kommando aktiv
UDF	*Update Filter*; Aktiviert die durch die letzten LFIL-Kommandos geladenen Filter-Parameter
RESET	*Reset the LM628*; Zurücksetzen der Controllerhardware
DFH	*Define Home*; setzt die Encoder-Position auf Null
RDRP	*Read Real Position*; liefert die aktuelle Encoder-Position
RDDP	*Read Desired Position*; liefert die Zielposition
SIP	*Set Index Position*; nach diesem Kommando wird das nächste Auftreten des Index Signals (ausgelöst durch den Indexschalter) durch ein Flag im Statusregister signalisiert; danach kann die Indexposition mit RDIP ausgelesen werden
RDIP	*Read Index Position*; liefert die Encoder-Position, an der das erstemal nach dem letzten SIP-Kommando das Indexsignal empfangen wurde.
RDSIGS	*Read Signals Register*
MSKI	*Mask Interrupts*
RSTI	*Reset Interrupts*

Abbildung B.6: Kommandos des C832, die die Simulationskomponente realisiert

einer Geschwindigkeit gestartet wird und durch explizites Stoppen beendet wird. Das aktuelle XCTL-System verwendet nur den *position mode*.

Registerwert	**Bedeutung**
000	Kommando oder Statusregister für Motor 1
001	Datenregister für Motor 1
010	Kommando oder Statusregister für Motor 2
011	Datenregister für Motor 2
111	Interruptregister

Abbildung B.7: Adressregister des C832 (Bit 2 - 0)

Bit	**Bedeutung**
0	Endlagenschalter für Motor 1 ist angefahren worden
1	Endlagenschalter für Motor 2 ist angefahren worden
2	letztes Interrupt kam vom Controller für Moter 1
3	letztes Interrupt kam vom Controller für Moter 2

Abbildung B.8: Interruptregister des C832

Bit	Bedeutung
0	*aquire next index* – 1 falls Controller auf ein Signal vom Indexeingang wartet (nach SIP-Kommando)
1	
2	*trajectory complete* – 1 falls das letzte *start trajectory*-Kommando abgearbeitet ist; der Motor steht nicht unbedingt an der Soll-Position
3	*index pulse aquired* – 1 falls nach letztem SIP-Kommando das Indexsignal empfangen wurde; das *index position register* enthält dann die Indexposition
4	
5	
6	
7	*motor off* – 1 falls der Motor im OFF-Zustandist
8	
9	
10	*on target* – 1 falls das letzte *start trajectory*-Kommando abgearbeitet ist und der Motor an der Soll-Position steht
11	*velocity mode* – 1 falls der Motor im *velocity mode* ist, 0 falls er im *position mode* ist
12	
13	
14	
15	

Abbildung B.9: Signalregister für einen Motor des C832