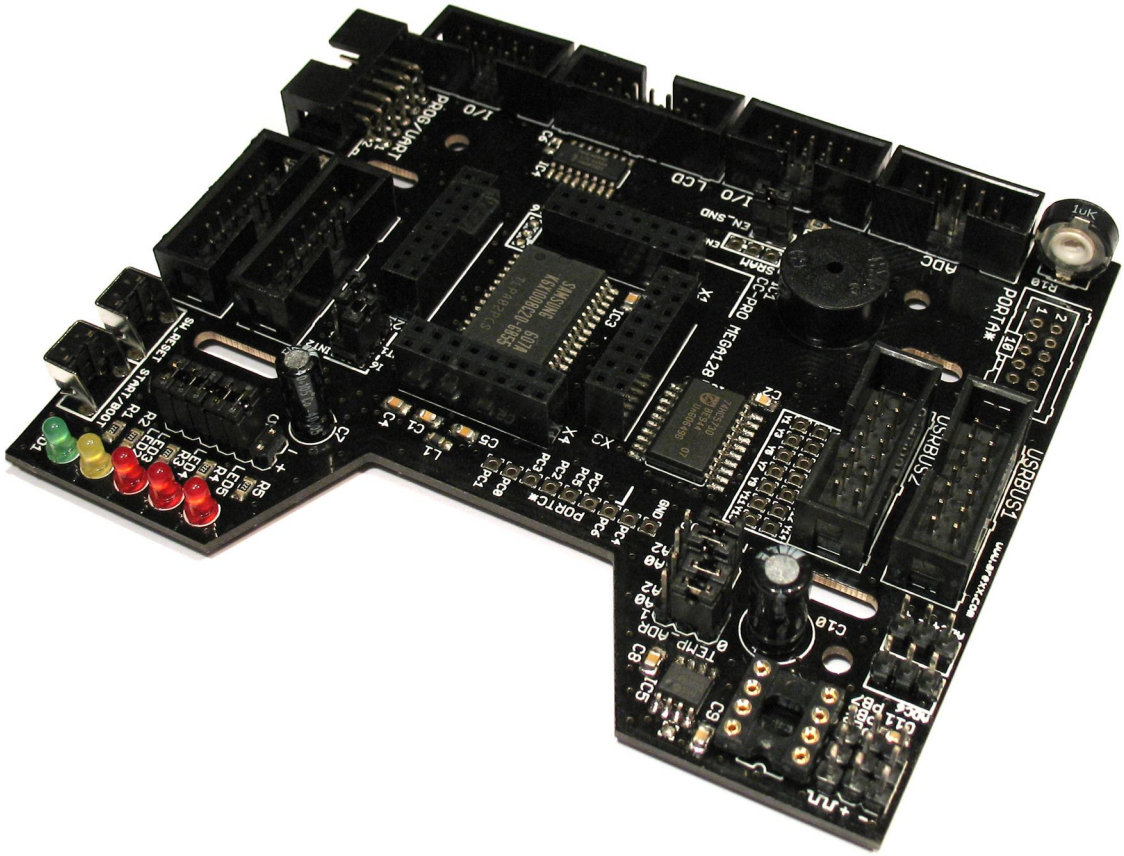
RP6 ROBOT SYSTEM

RP6 C-Control PRO M128

Erweiterungsmodul



AREXX Engineering

RP6-CCPRO-M128

©2008 AREXX Engineering

www.arexx.com

RP6 CCPRO M128

Bedienungsanleitung

- Deutsch (German) -

Version RP6-M128-DE-20081022

WICHTIGE INFORMATION!
Bitte unbedingt lesen!

Bevor Sie dieses RP6 Erweiterungsmodul in Betrieb nehmen, lesen Sie bitte diese Anleitung, die RP6 ROBOT SYSTEM Bedienungsanleitung und die C-Control PRO Anleitung vollständig durch! Diese Anleitungen erläutern Ihnen die korrekte Verwendung und weisen auf mögliche Gefahren hin! Weiterhin enthalten sie wichtige Informationen, die für viele Anwender keineswegs offensichtlich sein dürften. Die RP6 CCPRO M128 Anleitung ist nur eine Ergänzung!

Bei Nichtbeachtung dieser Anleitung und der RP6 ROBOT SYSTEM Anleitung erlischt jeglicher Garantieanspruch! Weiterhin übernimmt AREXX Engineering keinerlei Haftung für Schäden jeglicher Art, die aus Nichtbeachtung dieser Anleitung resultieren!

Bitte beachten Sie vor allem den Abschnitt "Sicherheitshinweise" in der RP6 ROBOT SYSTEM Bedienungsanleitung!

Impressum

©2008 AREXX Engineering

Nervistraat 16
8013 RS Zwolle
The Netherlands

Tel.: +31 (0) 38 454 2028
Fax.: +31 (0) 38 452 4482

"RP6 Robot System" ist eingetragenes Warenzeichen von AREXX Engineering.
Alle anderen Warenzeichen stehen im Besitz ihrer jeweiligen Eigentümer.

Diese Bedienungsanleitung ist urheberrechtlich geschützt. Der Inhalt darf ohne vorherige schriftliche Zustimmung des Herausgebers auch nicht teilweise kopiert oder übernommen werden!

Änderungen an Produktspezifikationen und Lieferumfang vorbehalten.
Der Inhalt dieser Bedienungsanleitung kann jederzeit ohne vorherige Ankündigung geändert werden.
Neue Versionen dieser Anleitung erhalten Sie kostenlos auf http://www.arexx.com/

Wir sind nicht verantwortlich für den Inhalt von externen Webseiten, auf die in dieser Anleitung verlinkt wird!

Hinweise zur beschränkten Garantie und Haftung

Die Gewährleistung von AREXX Engineering beschränkt sich auf Austausch oder Reparatur des Roboters und seines Zubehörs innerhalb der gesetzlichen Gewährleistungsfrist bei nachweislichen Produktionsfehlern, wie mechanischer Beschädigung und fehlender oder falscher Bestückung elektronischer Bauteile, ausgenommen aller über Steckverbinder/Sockel angeschlossenen Komponenten.
Es besteht keine Haftbarkeit für Schäden, die unmittelbar durch, oder in Folge der Anwendung des Roboters entstehen. Unberührt davon bleiben Ansprüche, die auf unabdingbaren gesetzlichen Vorschriften zur Produkthaftung beruhen.

Sobald Sie irreversible Veränderungen (z.B. Anlöten von weiteren Bauteilen, Bohren von Löchern etc.) am Roboter oder seinem Zubehör vornehmen oder der Roboter Schaden infolge von Nichtbeachtung dieser Anleitung nimmt, erlischt jeglicher Garantieanspruch!

Es kann nicht garantiert werden, dass die mitgelieferte Software individuellen Ansprüchen genügt oder komplett unterbrechungs und fehlerfrei arbeiten kann.
Weiterhin ist die Software beliebig veränderbar und wird vom Anwender in das Gerät geladen. Daher trägt der Anwender das gesamte Risiko bezüglich der Qualität und der Leistungsfähigkeit des Gerätes inklusive aller Software.
AREXX Engineering garantiert die Funktionalität der mitgelieferten Applikationsbeispiele unter Einhaltung der in den technischen Daten spezifizierten Bedingungen. Sollte sich der Roboter oder die PC-Software darüber hinaus als fehlerhaft oder unzureichend erweisen, so übernimmt der Kunde alle entstehenden Kosten für Service, Reparatur oder Korrektur.
Bitte beachten Sie auch die entsprechenden Lizenzvereinbarungen auf der CD-ROM!

Symbole

Im Handbuch werden folgende Symbole verwendet:

Das "Achtung!" Symbol weist auf besonders wichtige Abschnitte hin, die sorgfältig beachtet werden müssen. Wenn Sie hier Fehler machen, könnte dies ggf. zur Zerstörung des Roboters oder seinem Zubehör führen und sogar Ihre eigene oder die Gesundheit anderer gefährden!

Das "Information" Symbol weist auf Abschnitte hin, die nützliche Tipps und Tricks oder Hintergrundinformationen enthalten. Hier ist es nicht immer essentiell alles zu verstehen, aber meist sehr nützlich.

Inhaltsverzeichnis

1. Das RP6 CCPRO M128 Erweiterungsmodul

Mit dem RP6 CCPRO M128 (oder kurz „RP6 CCPRO M128") Erweiterungsmodul können Sie dem RP6 Robot System ein leistungsfähiges C-Control PRO MEGA128 Modul von Conrad Electronic hinzufügen (nicht im Lieferumfang enthalten, Best.Nr. 198219-62). Dieses Modul enthält einen Atmel ATMEGA128 Controller mit 128KB Flash ROM, 4KB SRAM und zahlreichen I/O Ports. Auf dem Erweiterungsmodul ist eine Speichererweiterung vorhanden, die den Arbeitsspeicher auf 64KB erhöht und somit auch die Ausführung umfangreicherer Algorithmen ermöglicht, wie z.B. eine einfache Wegplanung oder ähnliches. Auch die Ansteuerung komplexerer Hardware wird so möglich.

Auf dem Modul befinden sich neben diesen zentralen Komponenten auch noch ein 12 Bit Temperatursensor, ein Piezo zur Erzeugung von Tonsignalen, 5 LEDs und ein LC-Display Port, an den man direkt standard Text LCDs anschließen kann.

An Erweiterungsmöglichkeiten stehen 3 Anschlüsse für Servos, 19 freie I/Os (davon 8 ADC Kanäle), ein UART (zweite serielle Schnittstelle), ein DIL8 Sockel für I²C Bus EEPROMs der 24LCxxx Serie, sowie ein SPI Bus Anschluss zur Ansteuerung weiterer Hardware wie Schieberegister, ADCs, DACs, Flash-ROM (Speicherkarten) oder anderen Dingen zur Verfügung. Die Schnittstellen sind auf 10 poligen Wannensteckern verfügbar und können leicht mit Lochraster Experimentierplatinen verbunden werden.

Durch Deaktivierung einiger Komponenten auf dem Mainboard lassen sich bis zu 19 weitere I/O Ports und 3 Schaltausgänge freigeben (16 I/O Ports entfallen auf das SRAM, die Anschlüsse für anderweitige Verwendung dieser Ports müssen allerdings selbst mit Steckverbindern bestückt werden).

Als Entwicklungsumgebung kommt die von Conrad Electronic entwickelte CCPRO IDE zum Einsatz. Diese ist sehr leicht zu bedienen und bietet viele komfortable Funktionen. Es stehen zwei verschiedene Programmiersprachen zur Auswahl: Basic und CompactC. Basic ist für Einsteiger etwas leichter zu erlernen als C, hat für die CCPRO Unit aber dennoch nahezu identischen Funktionsumfang. Die Funktionsbibliothek der CCPRO Units ist sehr umfangreich und unterstützt sogar Multithreading. Mehr dazu entnehmen Sie bitte der ausführlichen C-Control PRO Dokumentation!

Bevor Sie mit dem RP6 CCPRO M128 loslegen, sollten Sie sich kurz mit dem Roboter an sich vertraut machen und einige der Beispielprogramme des Roboters OHNE montiertes Erweiterungsmodul ausprobieren! Diese Anleitung versteht sich als Ergänzung zur ausführlichen RP6 Bedienungsanleitung UND zur C-Control PRO Bedienungsanleitung. Lesen Sie diese auf jeden Fall bevor Sie mit dem RP6-M128 anfangen! Die Kapitel zur Programmierung des Roboters können Sie bei Bedarf nur kurz überfliegen und stattdessen die Anleitung der CCPRO Unit lesen. Bitte beachten Sie aber, dass die Beispielprogramme der CCPRO Unit dort nicht speziell für den RP6 geschrieben wurden. Nur der Programmcode in dieser ergänzenden Anleitung und in den Beispielprogrammen ist direkt auf dem Erweiterungsmodul lauffähig. Alle anderen Programme müssen jedoch zuvor leicht angepasst werden (vor allem an die andere Anschlussbelegung).

Ein wichtiger Hinweis für Anfänger: Für das RP6 CCPRO M128 geschriebene Programme laufen natürlich NICHT auf dem Mikrocontroller der Basiseinheit und umgekehrt (anderer Prozessor, andere Taktfrequenz und eine völlig andere Programmierumgebung)!

1.1. Technischer Support

Bei Fragen oder Problemen erreichen Sie unser Support Team wie folgt über das Internet (bevor Sie uns kontaktieren **lesen Sie aber bitte diese Bedienungsanleitung vollständig durch!** Erfahrungsgemäß erledigen sich viele Fragen so bereits von selbst!):

- über unser Forum: `http://www.arexx.com/forum/`

- per E-Mail: `info@arexx.nl`

Unsere Postanschrift finden Sie im Impressum dieses Handbuchs. Aktuellere Kontaktinformation, Softwareupdates und weitere Informationen gibt es auf unserer Homepage:

http://www.arexx.com/

auf der Homepage des Roboters:

http://www.arexx.com/rp6

und auf der C-Control Homepage:

http://www.c-control.de/

sowie dem (inoffiziellen) C-Control Forum:

http://ccpro.cc2net.de/

Auch dem Roboternetz, der größten deutschsprachigen Robotik Community, kann man auf jeden Fall mal einen Besuch abstatten:

http://www.roboternetz.de/

1.2. Lieferumfang

Folgende Dinge sollten Sie in Ihrer RP6 CCPRO M128 Packung vorfinden:

- Fertig aufgebautes RP6 CCPRO M128 Modul
- CD-ROM mit Software und Dokumentation
- 4 Stück 25mm M3 Distanzbolzen
- 4 Stück M3 Schrauben
- 4 Stück M3 Muttern
- 2 Stück 14 pol Flachbandkabel

Die Software und die PDF Anleitung befinden sich auf der CD-ROM. Aktualisierte Versionen der Software und dieser Anleitung gibt es auf unserer Homepage. Die Software für die C-Control PRO ist unter `http://www.c-robotics.com/` bzw. `http://www.c-control.de/` erhältlich!

1.3. Features und technische Daten

Dieser Abschnitt gibt einen Überblick darüber, was das RP6 CCPRO M128 Erweiterungsmodul zu bieten hat und dient gleichzeitig der Einführung einiger Begriffe und Bezeichnungen von Komponenten des Moduls.

Features, Komponenten und technische Daten des RP6 CCPRO M128:

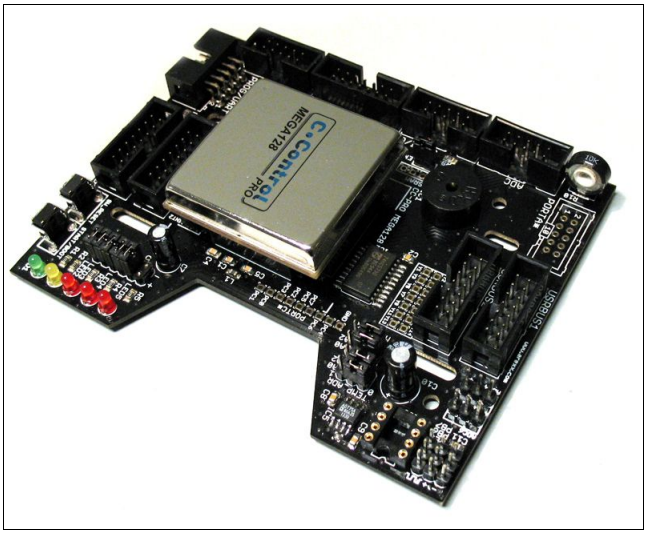
- **Steckplatz für leistungsfähiges C-CONTROL PRO MEGA128 Modul mit einem Atmel ATMEGA128 8-Bit Mikrocontroller**
  - ◊ Taktfrequnz 14,7456 Mhz (sog. Baudratenquartz um möglichst genau bestimmte Bitraten mit der seriellen Schnittstelle zu erreichen)
  - ◊ Speicher: 128KB Flash ROM, 4KB SRAM intern und **64KB externe SRAM Erweiterung auf dem Modul**, 4KB EEPROM.
  - ◊ In BASIC und CompactC programmierbar!
  - ◊ ... und vieles mehr (s. Dokumentation der CCPRO Unit)!
- **I²C-Bus Erweiterungsanschlüsse**
  - ◊ Kann beliebige I²C Bus Slaves ansteuern.
  - ◊ Der MEGA128 kann als Master oder Slave verwendet werden. Sinnvollerweise sollte er als Master verwendet werden und den Roboter komplett steuern (der Controller auf dem Mainboard übernimmt allerdings die Regelung der Motorgeschwindigkeit, ACS, IRCOMM, Akku Überwachung etc. selbstständig und entlastet so den Controller auf dem Erweiterungsmodul).
- **I²C-Bus 12 Bit Temperatursensor**
  - ◊ um präzise die aktuelle Umgebungstemperatur messen zu können.
  - ◊ Wählbare Auflösung von 0.5 bis 0.0625°C. Messgenauigkeit etwa 1°C.
- **Sockel für I²C Bus EEPROMs vom Typ 24(L)Cxxx** (nicht im Lieferumfang)
  - ◊ Daten bleiben bei EEPRMs nach dem Ausschalten erhalten
  - ◊ Verfügbar mit bis zu 1MBit (=128KByte) Kapazität – ideal für Datenlogger o.ä.
  - ◊ Typ. 1 Mio. mal wiederbeschreibbar
- **Piezo Tongeber**
  - ◊ um einfache Töne und Melodien zu erzeugen
  - ◊ Signalgeber z.B. um Fehler oder Statuswechsel zu melden
  - ◊ Kann deaktiviert werden – der I/O Port ist dann auf einem der 10poligen Steckverbinder verfügbar.

- **5 Status LEDs**
  - ◊ Zur Darstellung von Programm- oder Sensorzuständen
  - ◊ Zwei der LEDs sind an I/O Ports angeschlossen, die drei anderen an den Ausgängen eines Schieberegisters. Die LEDs lassen sich per Jumper deaktivieren – die I/Os bzw. die Schaltausgänge sind dann für andere Anwendungen verfügbar.
- **LC-Display Port**
  - ◊ Zum Anschluss eines standard Text LC-Displays. Es lassen sich beliebige HD44870 kompatible LCDs ansteuern, z.B. mit 16x2 oder 16x4 Zeichen. Bitte vor dem Kauf ausmessen bzw. in die Dokumentation des Displays schauen und passendes Montagematerial mitbestellen!
  - ◊ Das Display kann z.B. Textmeldungen/Menüs, Programmzustände oder Sensorwerte anzeigen.
- **19 freie I/O Ports zur Ansteuerung eigener Schaltungen und Sensoren**
  - ◊ **8** davon sind als **Analog/Digital Wandler (ADC) Kanäle** verwendbar
  - ◊ **3** können als **PWM Ausgänge** z.B. zur Steuerung von Servos benutzt werden
  - ◊ An zwei I/Os ist eine **weitere serielle Schnittstelle (UART)** verfügbar.
  - ◊ Bis zu 19 weitere I/O Ports lassen sich durch Deaktivieren von Komponenten auf dem Modul freigeben. 16 davon entfallen allerdings auf das SRAM. Die Steckverbinder für diese I/Os sind nicht auf dem Modul aufgelötet da man nur in Ausnahmefällen diese I/Os verwenden wird. Das große SRAM ist meist nützlicher (I/Os kann man bei Bedarf auch über I²C Bus Port Expander hinzufügen!).
- **Bis zu 2 externe Interrupts am XBUS Anschluss lassen sich verwenden.**
- **Ein Taster zum Starten des Programms und für andere Zwecke.**
- **Anschluss für das RP6 USB Interface** für den Programupload
  - ◊ Der Programmupload läuft wie beim Roboter über das USB Interface, allerdings nicht mit der RP6Loader Software sondern direkt aus der CCPRO IDE heraus. Alternativ lässt sich auch das CCPRO Applicationboard zum Programmieren der CCPRO Unit verwenden, dazu muss diese allerdings umgesteckt werden (daher ist das vor allem wenn noch ein LCD montiert wurde, nicht sinnvoll)

Es werden je 20 Basic und CompactC Beispielprogramme mitgeliefert um einen schnellen Einstieg zu ermöglichen.

Auf der Website zum Roboter und zur C-Control PRO werden demnächst evtl. weitere Programme und Updates zur Verfügung stehen. Sie können natürlich auch gerne Ihre eigenen Programme über das Internet mit anderen RP6 Anwendern austauschen!

2. Montage des Erweiterungsmoduls



Zunächst müssen Sie die CCPRO MEGA128 Unit richtig auf das Erweiterungsmodul aufstecken. Bitte fassen Sie zur Sicherheit vor dem Hantieren mit dem Modul einen geerdeten Gegenstand an, um sich zu entladen. Beispielsweise ein blankes Rohr eines Heizkörpers oder ein Metallteil eines geerdeten PC Gehäuses (mit 3 poligem Netzstecker!).

Kontrollieren Sie bitte *VOR* der Montage die Position aller Jumper auf dem Modul (s. Anhang A)! Diese sollten sich in den Standard Positionen befinden.

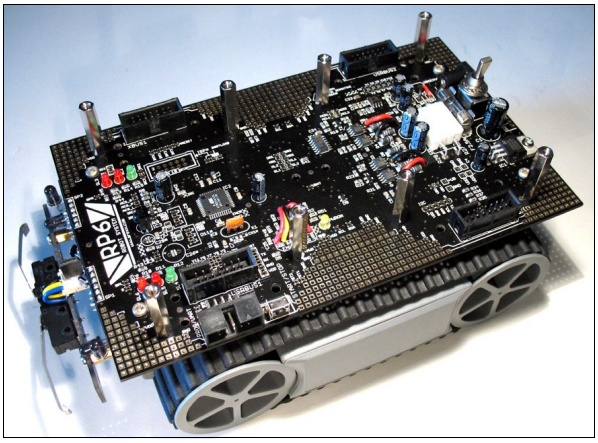
Es gibt nur eine Ausrichtung des CCPRO Moduls die passt (s. Abb.). Achten Sie dennoch darauf, dass alle Pins korrekt in den Buchsen auf der Platine sitzen!

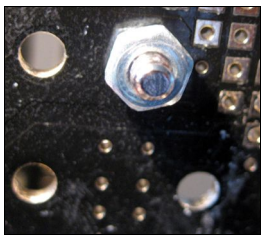
Das Aufstecken erfordert ein wenig Kraft, **aber bitte nicht zu stark drücken!** Auf der flachen Platinenunterseite direkt unter dem Modul kann man mit einem oder zwei Fingern gegenhalten. **Vorsicht:** Die Lötstellen rundherum sind ein wenig spitz! Achten Sie also besser darauf, nur die freie Fläche unter dem Modul zu berühren!

Wie Sie das Erweiterungsmodul auf dem Roboter befestigen, hängt davon ab, welche anderen Erweiterungsmodule Sie evtl. bereits auf dem Roboter montiert haben.

Um das Modul auf dem Roboter zu montieren, müssen Sie zunächst die vier Schrauben vom Mainboard lösen. Bei Bedarf können Sie auch den kleinen Stecker der Bumperplatine vorsichtig lösen, damit Sie das Mainboard komplett anheben können. Das ist aber nicht unbedingt erforderlich wenn Sie mit den Fingern auch so unter das Mainboard greifen können um die Distanzbolzen mit den M3 Muttern festzuschrauben.

Achtung: Beim wieder Anstecken des Kabels der Bumperplatine unbedingt mit einem Finger hinter der Sensorplatine gegenhalten damit diese nicht zu stark nach hinten gedrückt wird! Alternativ kann man auch die zwei Schrauben der Bumperplatine lösen und das Kabel dran lassen.



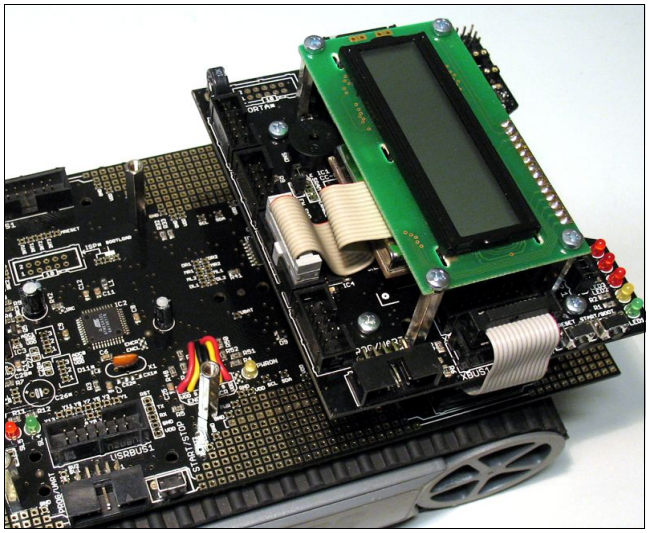


Dann können Sie die vier 25mm M3 Distanzbolzen nacheinander mit M3 Muttern in den Befestigungslöchern im Mainboard festschrauben wie auf dem Foto gezeigt.

Auf dem Bild oben sind alle 8 Distanzbolzen angeschraubt, auch die vom Lochraster Erweiterungsmodul!

Anschließend setzten Sie das Erweiterungsmodul oben auf die Distanzbolzen und Schrauben es mit den vier M3 Schrauben fest.

Jetzt müssen noch die zwei Flachbandkabel angesteckt werden – fertig.

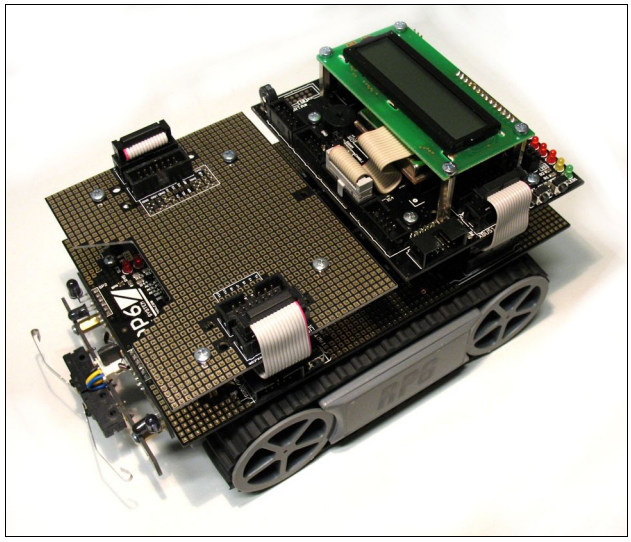


Wir empfehlen das RP6 CCPRO M128 auf dem hinteren Erweiterungsstapel auf dem Roboter zu montieren. Dann sind auch beide Programmieranschlüsse auf der gleichen Seite des Roboters zugänglich. Vorne können Sie das bereits im Lieferumfang des Roboters enthaltene Experimentiermodul anbringen (s. Foto unten für eine Beispielkonfiguration).

Wenn Sie zusätzlich ein LC-Display gekauft haben, sollten Sie es VOR der Montage auf dem Roboter an das Erweiterungsmodul anschließen und montieren. Je nach Display Format müssen Sie sich evtl. Adapter bauen um das Display zu befestigen, das gilt vor allem für größere Displays. Kleinere 2x16 Zeichen LCDs im standard Format können Sie direkt mit zwei bis vier Distanzbolzen befestigen. Um das Display noch oberhalb des CCPRO Moduls zu befestigen, benötigen Sie mindestens zwei 25mm Distanzbolzen, zwei Schrauben und zwei Muttern. Bei größeren Displays benötigen Sie ggf. zusätzliches Montagematerial.

Das bereits fertig vorkonfektionierte 14 polige Flachbandkabel des standard 2x16 Zeichen LC-Displays ist sehr flexibel und kann problemlos gefaltet werden. Sie können das Display also auch anders montieren als wir es hier demonstrieren.

Sie können auch ein beliebiges anderes Text LC-Display mit HD44780 kompatiblem Controller verwenden. Sie müssen dazu nur ein eigenes Kabel an das Display anlöten. *Dabei bitte unbedingt genauestens auf die richtige Anschlussbelegung achten! Bei Falschpolung können das Display und der Controller irreversibel beschädigt werden!*



Es werden nicht unbedingt 4 Distanzbolzen benötigt wie auf den Fotos! Zwei Stück (beide vorne *oder* hinten) reichen bereits aus, um das Display zu befestigen. Fertig auf dem Roboter montiert, könnte das Beispielsweise wie auf dem Foto aussehen.

Die drei 10 poligen Erweiterungsstecker mit den freien I/O Ports und dem SPI Bus kann man einfach über kleine 10 polige Flachbandkabel mit dem Lochraster Erweiterungsmodul verbinden und die I/Os und ADCs dort zur Auswertung von Sensoren o.ä. verwenden. Das geht natürlich auch mit Modulen die auf anderen Ebenen montiert sind – die Flachbandkabel sollten gut durch die Lücke in der Mitte zwischen den zwei Modulstapeln passen.

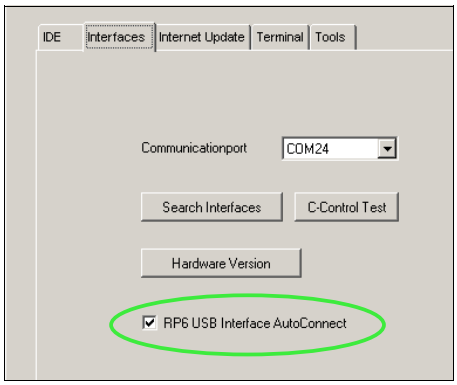
2.1. Funktionstest

Jetzt können Sie einen kleinen Funktionstest durchführen, um zu überprüfen ob das Erweiterungsmodul korrekt arbeitet.

Dazu müssen Sie sich allerdings zunächst schonmal ein wenig mit der CCPRO IDE vertraut machen (und diese natürlich auch installieren wie in der CCPRO Anleitung beschrieben. Den USB Treiber können Sie von der RP6 CD-ROM verwenden, mehr zur Installation finden Sie in der normalen RP6 Anleitung)! Um dann den Test durchzuführen, müssen Sie das am PC angeschlossene USB Interface über das 10 polige Flachbandkabel mit dem PROG/UART Anschluss auf dem RP6 CCPRO M128 verbinden und die CCPRO IDE starten. Öffnen Sie dann bitte das Projekt

`<RP6_CCPRO_Beispiele>/CompactC/RP6_CCPRO_SELFTEST/RP6_CCPRO_Selftest.cprj`

von der CD und lassen es den Compiler übersetzen (oben der blaue Pfeil in der Symbolleiste).



In den Einstellungen muss nun das richtige Interface gewählt werden (COMxx, *nicht* USB0 wählen, dies klappt nur mit dem CCPRO Application Board) und zusätzlich die Checkbox „RP6 USB Interface AutoConnect" aktiviert werden! Dies ist wichtig, damit die IDE die CCPRO Unit automatisch in den Bootload Modus versetzen kann. So muss man nicht jedesmal vor dem Programmupload auf die zwei Taster des Moduls drücken.

Sollte die CCPRO Unit nicht beim ersten Versuch erkannt werden, klicken Sie einfach nocheinmal auf den Button „Schnittstellensuche" und stellen Sie sicher, dass die CCPRO Unit zuvor korrekt in den Bootload Modus versetzt wurde (im Ausgaben Fenster unten sollte „Conrad C-Control Pro 2008" ausgegeben werden). Falls das nicht automatisch passiert sein sollte, können Sie dazu alternativ auch die beiden Taster auf dem CCPRO Modul in folgender Reihenfolge drücken um das Modul in den Bootload Modus zu versetzen: START/BOOT drücken und halten(!), RESET drücken, RESET loslassen und erst *danach* START/BOOT loslassen.

Wenn das CCPRO Modul im Bootload Modus ist, leuchtet die gelbe LED permanent. Dann können Sie das Programm in die CCPRO Unit laden. Starten Sie nach erfolgreichem Upload das Programm entweder per Druck auf den Start Taster vom CCPRO Modul oder vom Roboter selbst. Vorher muss ggf. noch einmal kurz Reset gedrückt werden, falls sich das Modul noch im Bootload Modus befindet (alternativ das „Blitz" Symbol in der Symbolleiste der CCPRO IDE anklicken). Die grüne LED sollte blinken wenn das Modul bereit zum Starten des geladenen Programms ist. Das gilt aber nur wenn Sie die RP6CCLib verwenden! Dort sorgt eine spezielle Routine dafür, dass alle Controller auf dem RP6 gleichzeitig starten.

Der Piezo Tongeber sollte direkt nach Druck auf den Start Taster eine kurze Tonfolge ausgeben. Es sollte dann ein LED Lauflicht ausgeführt werden und auf dem LCD (sofern vorhanden) ein Text „RP6 CCPRO M128" angezeigt werden.

Es werden auch Statusmeldungen über die serielle Schnittstelle ausgegeben, die Sie sich in der CCPRO IDE (oder im Terminal des RP6Loaders) anschauen können. **Achten Sie hierbei bitte auf mögliche Fehlermeldungen!**

Wenn das Display nichts oder nur zwei Reihen komplett dunkle Kästchen anzeigen sollte, aber die LEDs blinken und das Verbinden mit der CCPRO IDE klappt, müssen Sie den Kontrast des Displays einstellen (oder die Anschlussbelegung des Displays ist falsch...). Das geht mit Drehpotentiometer R10 hinten auf der Platine.

Das Poti können Sie mit einem kleinen Schlitzschraubendreher verstellen. Kreuzschlitz ist auch geeignet, klappt dann aber mit vielen Schraubendrehern nicht, da diese keinen guten Halt im Poti finden.

Drehen Sie einfach während der Roboter eingeschaltet ist am Poti solange bis etwas auf dem Display zu erkennen ist. Starten Sie das Selbsttestprogramm dann erneut!

Wenn der Selbsttest erfolgreich abgeschlossen wurde, sollte auf dem LC-Display die Meldung „SELFTEST OK!" erscheinen!

Die Ausgaben des Programms sollten ungefähr wie folgt aussehen:

```
   _____________________
   \| RP6  ROBOT SYSTEM |/
    \_-_-_-_-_-_-_-_-_-_/

  RP6 C-Control PRO M128 Selftest
  Writing Text to LCD...
  Testing Beeper...
  Testing LEDs...
  Testing Temperature Sensor...

Temperature: 19.6250°C --> OK
Temperature: 19.6875°C --> OK
Temperature: 19.8750°C --> OK
Temperature: 19.8125°C --> OK
Temperature: 19.8750°C --> OK
[...]
Temperature: 19.8125°C --> OK
Temperature: 19.6250°C --> OK

  Testing External 64K Memory...
MEM PATTERN: 17 - Writing...
Reading...
OK
MEM PATTERN: 34 - Writing...
Reading...
OK
MEM PATTERN: 68 - Writing...
[...]
MEM PATTERN: 64 - Writing...
Reading...
OK
MEM PATTERN: 128 - Writing...
Reading...
OK
MEMTEST PASSED!

  Testing I2C Communication...

############################################################
 Selftest finished successfully!
```

Die Temperaturmesswerte werden natürlich anders sein und die Ausgabe wurde ein wenig gekürzt!

3. Programmierung

Da es für die CCPRO Unit bereits eine ausführliche Dokumentation zur allgemeinen Anwendung und Programmierung gibt, kommen wir an dieser Stelle direkt zu den RP6 spezifischen Funktionen. Wir gehen hier nicht auf CompactC oder BASIC Grundlagen ein oder wie man einzelne Ports, ADCs, die serielle Schnittstelle u.ä. ansteuert oder mit Threads arbeitet! Dies ist ausführlich in der C-Control PRO Anleitung und der in der CCPRO IDE vorhandenen Hilfsfunktion beschrieben.

Im Folgenden beschreiben wir die Funktionen in der RP6 CCPRO Library (oder kurz „RP6CCLib"). Es wird meistens zu jeder Funktion ein kurzer Beispielcode in CompactC und BASIC angegeben. Die Funktionsprototypen werden in C und BASIC Notation aufgelistet. Ausführlichere Beispiele finden Sie separat auf der CD-ROM.

3.1.1. Start Taster abfragen und Programm starten

```
void RP6_waitForStart(void)

Sub RP6_waitForStart()
```

Diese Funktion ist wichtig um den Programmstart mit den anderen Controllern auf dem RP6 zu synchronisieren. Schließlich sollten alle Controller ihr Programm möglichst gemeinsam starten und vor allem sollte man die Programmausführung kontrolliert stoppen können ohne den Roboter komplett auszuschalten.

Diese Funktion wartet also zunächst auf ein bestimmtes Startsignal. Dies kann ein Druck auf den Start Taster auf dem RP6 CCPRO M128 Modul oder ein über den I²C Bus empfangenes Signal von einem anderen Controller (z.B. dem auf dem Mainboard) sein. Wird der Start Taster gedrückt, wird dies den anderen Controllern über den I²C Bus signalisiert.

3.1.2. Initialisierung

```
void RP6_CCPRO_Init(void)

Sub RP6_CCPRO_Init()
```

Anstatt RP6_waitForStart direkt aufzurufen, sollten Sie immer diese Funktion aufrufen. Sie initialisiert die Hardwaremodule und ruft anschließend RP6_waitForStart auf.

Ein typisches Programm-Grundgerüst für das RP6 CCPRO M128 kann folgendermaßen ausehen:

Beispiel in CompactC:

```
1  #include "../../RP6CCLib/RP6CCLib.cc"
2
3  void main(void)
4  {
5        RP6_CCPRO_Init(); // Initialisierung - IMMER ALS ERSTES AUFRUFEN!
6
7        // [...] Programmcode...
8
9        while(true)      // Endlosschleife
10       {
11            // [...] Programmcode...
12       }
13 }
```

Beispiel in Basic:

```
1  #include "../../RP6CCLib/RP6CCLib.cbas"
2
3  Sub main()
4        RP6_CCPRO_Init() ' Initialisierung - IMMER ALS ERSTES AUFRUFEN!
5
6        ' [...] Programmcode...
7
8        Do While True      ' Endlosschleife
9            ' [...] Programmcode...
10       End While
11 End Sub
```

Diese beiden Programme sind von der Funktionalität her identisch. Jeder kann für sich selbst entscheiden ob er lieber in BASIC oder CompactC Programmieren möchte.

*Hinweis: Man kann die RP6_CCPRO_lib (Zeile 1 in den Beispielen) auch über die Projekteinstellungen einbinden. Es ist aber sinnvoller anstatt des absoluten Pfads eine relative Pfadangabe zu verwenden („../../lib/", zwei Verzeichnisebenen höher). So kann man das Projekt mitsamt Library problemlos verschieben oder auch sofort auf einem anderen Computer verwenden. Bitte achten Sie darauf, das es für BASIC und CompactC zwei verschiedene Libraries gibt – die für CompactC heisst RP6CCLib**.cc** und die für Basic RP6CCLib**.cbas**.*

3.1.3. Text ausgeben

```
print(STRING)
println(STRING)
newline()
printInteger(Int)
```

In CompactC und BASIC gibt es noch keine Textkonstanten – daher müsste man eigentlich immer zuerst einem Array einen Text zuweisen und dann dieses der Serial_WriteText Funktion übergeben.

Um die Programme übersichtlicher zu halten und Schreibarbeit zu sparen, stellt die RP6CCLib Makros zur Verfügung, die genau das für Texte bis 64 Zeichen (und Integer Zahlenwerte) übernehmen. „print" gibt den übergebenen Text aus und „println" hängt zusätzlich noch die Sonderzeichen „\r\n" an, damit im Terminal eine neue Zeile begonnen wird.

CompactC Beispiel:

```
println("Hallo Welt!");
print("Test1: ");
printInteger(x); // Gibt den Wert der Variablen x als ASCII Text aus
print(" ... Test2");
newline(); // Neue Zeile
```

Basic Beispiel:

```
println("Hallo Welt!")
print("Test1: ")
printInteger(x)    ' Gibt den Wert der Variablen x als ASCII Text aus
print(" ... Test2")
newline()        ' Neue Zeile
```

3.1.4. Status LEDs

```
void setLEDs(byte leds)
Sub setLEDs(leds As Byte)
```

Die 5 Status LEDs auf dem Mainboard lassen sich über diese Funktion steuern.

CompactC Beispiel:

```
setLEDs(LED1 | LED2 | LED5); // LED1, LED2 und LED5 an, LED3 und LED 4 aus.
SetLEDs(0);  // Alle LEDs aus
setLEDs(31); // Alle LEDs an (31 = Binär 11111)
```

Basic Beispiel:

```
setLEDs(LED1 Or LED2 Or LED5) ' LED1, LED2 und LED5 an, LED3 und LED 4 aus.
SetLEDs(4) ' LED3 anschalten
setLEDs(5) ' LED1 und 3 anschalten (5 = Binär 00101)
```

LED1 und 2 sind direkt an normalen I/O Ports angeschlossen und können daher auch wie normale I/O Ports angesteuert werden. Aber LED3, 4 und 5 sind zusammen mit dem LCD mit einem Schieberegister am SPI Bus verbunden. Das wurde so realisiert um I/O Ports zu sparen, die sich besser für andere Zwecke verwenden lassen. Das LCD allein würde ansonsten 6 I/Os blockieren, dazu noch 5 LEDs, macht 11 I/Os insgesamt. Mit Schieberegister sind es nur 6 I/Os und der SPI Bus kann auch noch weitere externe Hardware anbinden.

3.1.5. Beeper

Der Signalgeber auf dem RP6-M128 kann mit der Funktion

```
void beep(word pitch, word time, word pause)
Sub beep(pitch As Word, time As Word, pause As Word)
```

gesteuert werden. Allerdings ist diese Funktion blockierend – d.h. Sie setzt die Tonhöhe, blockiert für die vorgegebene Zeitspanne, schaltet den Signalgeber wieder aus und pausiert nochmals um die zweite gegebene Zeitspanne. Hierzu wird die Funktion AbsDelay aus der CCPRO Funktionsbibliothek verwendet. Die Zeiten werden hier in ms angegeben.

CompactC Beispiel:

```
#define Tone_A1    262     // 440Hz
beep(Tone_A1, 100, 200)    // 100ms Ton ausgeben, 200ms Pause
beep(Tone_A1, 355, 422)    // 355ms Ton ausgeben, 422ms Pause
```

Basic Beispiel:

```
#define Tone_A1    262     '440Hz
beep(Tone_A1, 100, 200)    ' 100ms Ton ausgeben, 200ms Pause
beep(Tone_A1, 355, 422)    ' 355ms Ton ausgeben, 422ms Pause
```

Die Funktion beep ist hauptsächlich für die Ausgabe von Melodien u.ä. gedacht. Um andere Dinge parallel laufen zu lassen, könnte man auch Threads verwenden. Man muss dann aber darauf achten, diese Funktion nicht gleichzeitig in zwei verschiedenen Threads zu verwenden (man kann nicht aus zwei verschiedenen Threads gleichzeitig auf dieselbe Hardwarekomponente zugreifen)!

Allerdings funktioniert das mit der normalen beep Funktion ohnehin nicht, da hier wie oben schon erwähnt wurde AbsDelay verwendet wird. AbsDelay unterbricht aber auch die Ausführung aller Threads. Daher gibt es eine spezielle Funktion:

```
void beep_t(word pitch, word time, word pause)

Sub beep_t(pitch As Word, time As Word, pause As Word)
```

die anstatt von AbsDelay, Thread_Delay verwendet. Der Nachteil ist, dass Thread_Delay nur auf 10ms genau pausiert. D.h. mit beep_t sähe obiges Beispiel folgendermaßen aus:

CompactC Beispiel:

```
#define Tone_A1     262 // 440Hz
beep_t(Tone_A1, 10, 20) // 100ms Ton ausgeben, 200ms Pause
beep_t(Tone_A1, 35, 42) // 350ms Ton ausgeben, 420ms Pause
```

Basic Beispiel:

```
#define Tone_A1     262 '440Hz
beep_t(Tone_A1, 10, 20) ' 100ms Ton ausgeben, 200ms Pause
beep_t(Tone_A1, 35, 42) ' 350ms Ton ausgeben, 420ms Pause
```

Anstatt der beep und beep_t Funktion, kann man auch nur die Makros

```
sound(pitch)

und

sound_off()
```

verwenden. Hier kann man nur die Tonhöhe einstellen. Das ist für kontinuierliche Tonfolgen (z.B. Sirenengeräusch o.ä.) nützlich. Die Funktion ist NICHT blockierend! Es wird also nur die Tonhöhe gesetzt und sofort mit dem Programm fortgefahren. Mit sound_off() kann man den Tongenerator wieder abschalten.

Mit dem Makro

```
tone(PITCH, TIME)
```

gibt es noch eine weitere Alternative. Diese wird z.B. in einem Beispielprogramm verwendet um längere Melodien abzuspielen, bei denen der Beeper nicht zwischendrin abgeschaltet werden darf.

Achtung: Bei allen Beeper Funktionen liegt der zulässige Bereich von „pitch" (Tonhöhe) zwischen 0 und 65535. *Allerdings ist 0 die höchste Frequenz und 65535 die niedrigste!*

Details finden Sie in der C-Control PRO Dokumentation im Kapitel über Timer. Von den Beep Funktionen wird Timer3 mit Prescaler 64 verwendet.

3.1.6. I²C Bus

Neben den I²C Bus Funktionen der normalen CCPRO Library, gibt es einige zusätzliche Funktionen speziell um den Roboter über das „RP6Base_I2CSlave" Programm steuern zu können. Dieses Programm muss man dazu natürlich mit dem RP6Loader (s. RP6 Anleitung) in den Controller auf dem Mainboard laden. Es stellt eine Reihe von Registern bereit, über die sich der Roboter komplett über den Bus steuern lässt. Man kann ihm damit Befehle senden und alle Sensordaten auslesen.

Der CCPRO Interpreter stellt natürlich noch einige weitere Funktionen zur Verfügung mit denen sich Daten über den I²C Bus übertragen lassen.

3.1.6.1. Befehle senden

Ein einzelner Befehl lässt sich mit folgender Funktion übertragen:

```
void RP6_writeCMD(byte adr, byte cmd)
Sub RP6_writeCMD(adr As Byte, cmd As Byte)
```

Meistens muss man allerdings nicht nur ein einzelnes Kommando übertragen, sondern zusätzlich auch noch Werte übergeben – z.B. um die LEDs auf dem Mainboard zu setzen, die Geschwindigkeit einzustellen o.ä.

Ein zusätzlicher Parameter kann mit dieser Funktion übergeben werden:

```
void RP6_writeCMD_1param(byte adr, byte cmd, byte param)
Sub RP6_writeCMD_1param(adr As Byte, cmd As Byte, param As Byte)
```

CompactC Beispiel:

```
// I2C Bus Adresse des Controllers auf dem Mainboard:
#define RP6_BASE_ADR 10

RP6_writeCMD_1param(RP6_BASE_ADR, CMD_SETLEDS, 0x09);
```

Basic Beispiel:

```
#define RP6_BASE_ADR 10

RP6_writeCMD_1param(RP6_BASE_ADR, CMD_SETLEDS, 0x09)
```

0x09 ist Hexadezimal für Binär: 00001001. Die 6 LEDs auf dem Mainboard sind dabei den ersten 6 Bits eines Bytes zugeordnet.

0x01 würde also LED1 einschalten. 0x02 steht für LED2, 0x04 für LED3, 0x08 für LED4, 0x10 für LED5 und 0x20 für LED6.

Das kann man natürlich auch dezimal schreiben: 1 für LED1, 4 für LED3, 16 für LED5 und 32 für LED6. Wenn man mehrere LEDs anschalten möchte, muss man diese einzelnen Werte einfach addieren. So kommt man auf 63 um alle LEDs anzuschalten bzw. 0x3F in hexadezimaler Schreibweise. 0x24 schaltet LED6 und LED3 ein.

(0x24 = 00100100 Binär)

Alternativ gibt es auch passende Definitionen für die LEDs, so dass man auch schreiben kann:

```
RP6_writeCMD_1param(RP6_BASE_ADR, CMD_SETLEDS, LED1 | LED4);  // CompactC

RP6_writeCMD_1param(RP6_BASE_ADR, CMD_SETLEDS, LED1 Or LED4)  // Basic
```

Das ist natürlich etwas leichter lesbar. Die anderen Beispiele oben sollten dem Verständnis dienen, denn LED1 ist nichts anderes als der Dezimalwert 1, LED2 = 2, LED3 = 4, LED4 = 8 und LED5 = 16.

Also werden mit obigem Befehl (0x09) die beiden grünen LEDs 1 und 4 angeschaltet.

Analog zu writeCMD_1param gibt es Funktionen für 2:

```
void RP6_writeCMD_2params(byte adr, byte cmd, byte param1, byte param2)

Sub  RP6_writeCMD_2params(adr  As  Byte,  cmd  As  Byte,  param1  As  Byte,
                          param2 As Byte)
```

und für 3 Paramter:

```
void RP6_writeCMD_3params(byte adr, byte cmd, byte param1,
                          byte param2, byte param3)

Sub RP6_writeCMD_3params(adr As Byte, cmd As Byte, param1 As Byte,
                          param2 As Byte, param3 As Byte)
```

Damit sind bereits die meisten Dinge abgedeckt. In speziellen Fällen mit mehr Parametern kann man die Funktion

```
void RP6_writeCommand_params(byte adr, byte cmd,
                          byte params[], byte param_count)

Sub RP6_writeCommand_params(adr As Byte, cmd As Byte, ByRef params As Byte,
                          param_count As Byte)
```

verwenden. Hier können bis zu 255 Parameter in einem Array übergeben werden.

CompactC Beispiel:

```
byte params[16];
params[0] = 10;
params[1] = 44;
params[2] = 10;

// ...

params[15] = 255;
// Befehl „40“ mit 16 Parameter Werten an Adresse „10“ senden:
RP6_writeCommand_params(10, 40, params, 16);
```

Basic Beispiel:

```
Byte params(16)
params(0) = 10
params(1) = 42
params(2) = 10

' ...

params(15) = 255
' Befehl „40“ mit 16 Parameter Werten an Adresse „10“ senden:
RP6_writeCommand_params(10, 40, params, 16)
```

Das RP6I2CSlave Programm kennt momentan 12 Befehle, mit denen sich der Roboter komplett steuern lässt. Diese sind in folgender Tabelle aufgelistet:

Befehl	Code	Beschreibung
CMD_POWER_OFF	0	Encoder, Stromsensoren und Power LED abschalten
CMD_POWER_ON	1	Encoder, Stromsensoren und Power LED einschalten
CMD_SETLEDS	3	LEDs setzen Parameter 1: LED Zustand – erste 6 Bits Bit 0 ist LED1, Bit 6 ist LED6
CMD_STOP	4	Motoren und Geschwindigkeitsregelung stoppen
CMD_MOVE_AT_SPEED	5	Mit bestimmter Geschwindigkeit bewegen Parameter 1: Geschwindigkeit links Parameter 2: Geschwindigkeit rechts Geschwindigkeitswert in Encoder Zählschritten pro 200ms
CMD_CHANGE_DIR	6	Drehrichtung der Motoren ändern Paramter 1: Richtung – FWD, BWD, LEFT oder RIGHT (Konstanten sind in RP6CCLib definiert)
CMD_MOVE	7	Eine bestimmte Distanz abfahren Parameter 1: Geschwindigkeit Parameter 2: Richtung, FWD oder BWD Parameter 3: High Byte der zurückzulegenden Strecke Parameter 4: Low Byte
CMD_ROTATE	8	Um einen bestimmten Winkel rotieren Parameter 1: Geschwindigkeit Parameter 2: Richtung, LEFT oder RIGHT Parameter 3: High Byte des Winkels Parameter 4: Low Byte des Winkels
CMD_SET_ACS_POWER	9	ACS Sendeleistung einstellen Parameter 1: Sendeleistung ACS_PWR_OFF, ACS_PWR_LOW, ACS_PWR_MED, ACS_PWR_HIGH
CMD_SEND_RC5	10	RC5 Signal mit dem IRCOMM senden Parameter 1: Adresse und Togglebit Parameter 2: Daten
CMD_SET_WDT	11	Watchdog Timer aktivieren Parameter 1: true oder false
CMD_SET_WDT_RQ	12	Watchdog Timer request aktiviere Parameter 1: true oder false

Wie man die einzelnen Befehle verwenden kann, wird in den Beispielprogrammen demonstriert.

Um einen 16 Bit Wert wie z.B. beim CMD_MOVE Befehl zu schreiben, muss man diesen Wert in zwei 8 Bit Werte aufteilen.

CompactC Beispiel:

```
word distance; // 16 Bit Wert der geschrieben werden soll
distance = 4000;  // 4000 * 0.25mm = 1 Meter
byte params[5];
params[0] = 80;   // 10 cm/s
params[1] = FWD;  // Vorwärts
params[2] = distance >> 8;     // obere 8 Bit der Distanz
params[3] =  distance & 0x0F;  // untere 8 Bit
RP6_writeCommand_params(10, CMD_MOVE, params, 4);
```

3.1.6.2. Daten auslesen

Natürlich kann man nicht nur Befehle übertragen, sondern auch Daten wie z.B. Sensorwerte auslesen. Ein einzelnes Register kann man mit folgender Funktion auslesen:

```
byte RP6_readRegister(byte adr, byte reg)

Sub RP6_readRegister(adr As Byte, reg As Byte) As Byte
```

Mehrere Register lassen sich mit:

```
void RP6_readRegisters(byte adr, byte reg, char readBuffer[], byte reg_count)

Sub RP6_readRegisters(adr As Byte, reg As Byte,
                      ByRef readBuffer As Byte, reg_count As Byte)
```

auslesen und werden in das Array „readBuffer" geschrieben.

CompactC Beispiel:

```
// Den Wert des Spannungssensors auslesen:

byte messageBuf[3]; // Buffer für Datenempfang

RP6_readRegisters(RP6_BASE_ADR, I2C_REG_ADC_UBAT_L, messageBuf, 2);

// Der Messwert besteht aus zwei Bytes, daher muss man diese nun wieder
// zu einem Messwert zusammensetzten:
adcBat = (messageBuf[1] << 8) | (messageBuf[0]);
// (dazu wird das höherwertige Byte um 8 Stellen nach links verschoben und
// an diese Stellen über eine ODER Verknüpfung das niederwertige
// Byte geschrieben)

// Jetzt kann man den Wert ausgeben:
print("Battery Sensor Value:");
printInteger(adcBat);
println("  ");

// Es handelt sich dabei NICHT um den direkten Spannungswert, sondern um den
// rohen Messwert des Analog Digital Wandlers. Das kann man noch umrechnen
// um den realen Wert zu erhalten (durch 102.4 teilen).
```

Basic Beispiel:

```
// Den Wert des Spannungssensors auslesen:

Dim messageBuf(3) As Byte; // Buffer für Datenempfang

RP6_readRegisters(RP6_BASE_ADR, I2C_REG_ADC_UBAT_L, messageBuf, 2)

// Der Messwert besteht aus zwei Bytes, daher muss man diese nun wieder
// zu einem Messwert zusammensetzten:
adcBat = (messageBuf(1) << 8) Or (messageBuf(0));
// (dazu wird das höherwertige Byte um 8 Stellen nach links verschoben und
// an diese Stellen über eine ODER Verknüpfung das niederwertige
// Byte geschrieben)

// Jetzt kann man den Wert ausgeben:
print("Battery Sensor Value:")
printInteger(adcBat)
println("  ")

// Es handelt sich dabei NICHT um den direkten Spannungswert, sondern um den
// rohen Messwert des Analog Digital Wandlers. Das kann man noch umrechnen
// um den realen Wert zu erhalten (durch 102.4 teilen).
```

Das Standard RP6_I2C_Slave Programm stellt 30 Register zur Verfügung, mit denen die aktuellen Sensorwerte und Programmzustände ausgelesen werden können. Diese sind in folgender Tabelle aufgelistet:

Registername	#	Beschreibung
I2C_REG_STATUS1	0	Status Register 1 Bit 0: batLow; 1: bumperLeft; 2: bumperRight; 3: RC5reception; 4: RC5transmitReady; 5: obstacleLeft; 6: obstacleRight; 7: driveSystemChange
I2C_REG_STATUS2	1	Status Register 2 Bit 0: powerOn; 1: ACSactive; 2: watchDogTimer; 3: wdtRequest; 4: wdtRequestEnable;
I2C_REG_MOTION_STATUS	2	Status Register für Antriebssystem Bit 0: movementComplete; 1: motorsOn; 2: motorOvercurrent; 3+4: direction;
I2C_REG_POWER_LEFT	3	Aktuell eingestellter PWM Wert Links
I2C_REG_POWER_RIGHT	4	Aktuell eingestellter PWM Wert Rechts
I2C_REG_SPEED_LEFT	5	Encoder Messwert Links (Enc. Zählschritte / 200ms)
I2C_REG_SPEED_RIGHT	6	Encoder Messwert Rechts (Enc. Zählschritte / 200ms)
I2C_REG_DES_SPEED_LEFT	7	Geschwindigkeits Sollwert Links
I2C_REG_DES_SPEED_RIGHT	8	Geschwindigkeits Sollwert Rechts
I2C_REG_DIST_LEFT_L I2C_REG_DIST_LEFT_H	9, 10	Zurückgelegte Distanz Links (High und Low Byte)
I2C_REG_DIST_RIGHT_L I2C_REG_DIST_RIGHT_H	11, 12	Zurückgelegte Distanz Rechts (High und Low Byte)
I2C_REG_ADC_LSL_L I2C_REG_ADC_LSL_H	13, 14	ADC Messwert, Lichtsensor Links (High und Low Byte)
I2C_REG_ADC_LSR_L I2C_REG_ADC_LSR_H	15, 16	ADC Messwert, Lichtsensor Rechts (High und Low Byte)
I2C_REG_ADC_MOTOR_CURL_L I2C_REG_ADC_MOTOR_CURL_H	17, 18	ADC Messwert Stromsensor Links (High und Low Byte)
I2C_REG_ADC_MOTOR_CURR_L I2C_REG_ADC_MOTOR_CURR_H	19, 20	ADC Messwert Stromsensor Rechts (High und Low Byte)
I2C_REG_ADC_UBAT_L I2C_REG_ADC_UBAT_H	21, 22	ADC Messwert Batterie Spannung (High und Low Byte)
I2C_REG_ADC_ADC0_L I2C_REG_ADC_ADC0_H	23, 24	ADC Messwert, freier ADC Kanal 0 (High und Low Byte)
I2C_REG_ADC_ADC1_L I2C_REG_ADC_ADC1_H	25, 26	ADC Messwert, freier ADC Kanal 1 (High und Low Byte)
I2C_REG_RC5_ADR	27	Adresse und Togglebit des zuletzt empfangenen RC5 Datenpakets
I2C_REG_RC5_DATA	28	Daten des zuletzt empfangenen RC5 Datenpakets
I2C_REG_LEDS	29	Zustand der Status LEDs

3.1.6.3. Temperatursensor

Für den per I²C Bus angeschlossenen Temperatursensor auf der Platine gibt es auch bereits vorbereitete Funktionen. Mit:

```
TCN75_write_cfg(byte adr, byte config)
Sub TCN75_write_cfg(adr As Byte, config As Byte)
```

kann man allgemein in das Konfigurations-Register des Sensors schreiben (nähere Informationen dazu sind im Datenblatt des TCN75 Sensors zu finden. In der Library wurden bereits einige Einstellungen als Konstanten hinterlegt).

Besser lesbar wird der Code allerdings, wenn man folgende Makros verwendet:

```
TCN75_run(ADR, CONFIG)
```

und

```
TCN75_shutdown(ADR)
```

um die Auswertung zu starten und zu stoppen. Diese Makros rufen aber auch nur TCN75_write_cfg auf. Bei TCN75_run kann man neben der Adresse des Sensors auch das Konfigurationsbyte übergeben.

Um den Sensor dann auszulesen, wird folgende Funktion verwendet:

```
void TCN75_read(byte adr)
Sub TCN75_read(adr As Byte)
```

CompactC Beispiel:

```
TCN75_run(TCN75_ADR, TCN75A_CONFIG_RES_12);  // Messung starten...
AbsDelay(250);                               // Pause
TCN75_shutdown(TCN75_ADR);                   // Messung stoppen
TCN75_read(TCN75_ADR);                       // Auslesen...
```

Basic Beispiel:

```
TCN75_run(TCN75_ADR, TCN75A_CONFIG_RES_12)   ' Messung starten...
AbsDelay(250)                                ' Pause
TCN75_shutdown(TCN75_ADR)                    ' Messung stoppen
TCN75_read(TCN75_ADR)                        ' Auslesen...
```

Das liest allerdings den aktuellen Temperaturwert nur aus. Er muss dann natürlich noch irgendwie weiterverarbeitet werden...

Nachdem der aktuelle Temperaturwert ausgelesen ist kann mit den Funktionen:

```
byte getTemperatureHigh(void)
Sub getTemperatureHigh() As Byte
```

und

```
byte getTemperatureLow(void)
Sub getTemperatureLow() As Byte
```

auf „Low" und „High" Byte des Messwerts zugegriffen werden. Diese beiden Bytes enthalten den Sensorwert allerdings noch nicht in einem passenden Format – man kann diesen nun z.B. noch in eine Fließpunktzahl konvertieren.

CompactC Beispiel:

```
float temperature;
char result[32];
int temp_low;

// Messwert in "float" Wert konvertieren (12 Bit Messung):
temp_low = getTemperatureLow();
if(!temp_low & 128)
        temp_low = (temp_low & 63) - 127;
else
        temp_low = temp_low & 63;
temperature = temp_low + (0.0625 * (getTemperatureHigh()>>4));

// Messwert ausgeben:
newline();
print("Temperature: ");
Str_WriteFloat(temperature,4,result,0);
Serial_WriteText(0,result);
print("°C");
newline();
```

Basic Beispiel:

```
Dim temperature As Single ' Floatingpoint Variable für Temperaturwert
Dim result(32) As Char
Dim temp_low As Integer

' [...]

' Messwert in "float" Wert konvertieren (12 Bit Messung):
temp_low = getTemperatureLow()
If Not temp_low And 128 Then
        temp_low = (temp_low And 63) - 127
Else
        temp_low = temp_low And 63
End If
temperature = temp_low + (0.0625 * (getTemperatureHigh()>>4))

' Messwert ausgeben:
newline()
print("Temperature: ")
Str_WriteFloat(temperature,4,result,0)
Serial_WriteText(0,result)
print("°C")
newline()
```

In beiden Beispielprogrammen wird der Messwert auf der seriellen Schnittstelle ausgegeben.

3.1.7. SPI Bus

Am SPI (=Serial Peripheral Interface) Bus können verschiedene Komponenten angeschlossen werden. Unter anderem Analog/Digital Wandler, Digital/Analog Wandler, EEPROMs, Flash Speicher, Speicherkarten, LC-Displays oder Schieberegister.

Das optionale LC-Display des CCPRO Erweiterungsmoduls ist Beispielsweise an einem 8 Bit Schieberegister angeschlossen, zusammen mit drei der Status LEDs. Die Funktionen zur Ansteuerung des Displays werden im nächsten Abschnitt besprochen.

Der Zugriff auf den SPI Bus ist recht einfach, man muss nur die Chipselect Leitung des jeweiligen Slaves auf high Pegel schalten und kann dann Daten lesen und schreiben.

ACHTUNG: Es darf immer nur GENAU EINE Chipselect Leitung zur selben Zeit aktiviert sein! Andernfalls funktioniert zumindest das Auslesen von Daten nicht korrekt, da in diesem Fall zwei Slaves Daten senden würden.

Die SPI Funktionen sind interne Interpreter Funktionen, also nicht in der RP6CCLib enthalten! Da diese allerdings noch nicht lange im CCPRO Interpreter verfügbar sind, werden diese hier auch noch kurz beschrieben. Sie sollten auch bald in die CCPRO Dokumentation mit aufgenommen werden.

```
void SPI_Enable(byte ctrl)
```

Initialisiert das SPI Hardware Modul mit dem Konfigurationsbyte „ctrl". Der Standardwert für das CCPRO Erweiterungsmodul ist 0x50, es werden also die SPE (Enable) und MSTR (Master Modus) Bits im Konfigurationsregister gesetzt (s. MEGA128 Datenblatt). Die Taktfrequenz ist auf die halbe CPU Taktfrequenz eingestellt, also 7.3728MHz.

```
byte SPI_Read(void)
```

Liest ein Byte vom gerade aktivierten SPI Slave.

```
void SPI_Write(byte data)
```

Sendet ein Byte an den gerade aktivierten SPI Slave.

```
void SPI_ReadBuf(byte buf[], byte length)
```

Liest „length" Bytes vom gerade aktivierten SPI Slave und speichert diese im Array „buf".

```
void SPI_WriteBuf(byte buf[], byte length)
```

Sendet „length" Bytes aus dem Array „buf" an den gerade aktivierten SPI Slave.

Bei einem Schieberegister wie dem 74HC4094, das auf dem CCPRO Erweiterungsmodul verwendet wird, muss man eine „Strobe" Leitung aktivieren *nachdem* man die Daten geschrieben hat. Also z.B.

```
Thread_Lock(1);              // Wichtig: Threadumschaltung sperren!
SPI_Write(external_port);    // Daten senden
Port_WriteBit(PORT_STR, 1);  // Strobe Leitung aktivieren
delayCycles(5);              // Sehr kurze Pause
Port_WriteBit(PORT_STR, 0);  // Strobe Leitung wieder deaktivieren
Thread_Lock(0);              // Wichtig: Threadumschaltung wieder freigeben!
```

So wird es auch in der RP6CCLib erledigt. Bei anderen SPI Slaves wie z.B. Flash Speicherchips oder EEPROMs ist es meist erforderlich die Chipselect Leitung VOR dem Bus-

zugriff zu setzen und danach zu deaktivieren.

Das könnte z.B. folgendermaßen aussehen:

```
Thread_Lock(1);                  // Wichtig: Threadumschaltung sperren!
Port_WriteBit(PORT_FLASH_CS, 1);    // CS Leitung aktivieren
SPI_Write(10);                      // Daten senden
result = SPI_Read();                // Daten lesen
SPI_WriteBuf(buffer, 32);           // 32 Bytes senden.
SPI_ReadBuf(buffer, 16);            // 16 Bytes lesen.
Port_WriteBit(PORT_STR, 0);         // CS Leitung wieder deaktivieren
Thread_Lock(0);                  // Wichtig: Threadumschaltung wieder freigeben!
```

Das ist natürlich kein vollständiges Beispiel und muss an den jeweiligen Slave angepasst werden!

Hier müssen Sie das entsprechende Datenblatt des Herstellers beachten. Wichtig ist für die komplette Zeit des Zugriffs die Threadumschaltung zu sperren (sofern Threads verwendet werden)! Sonst könnten zwei Threads gleichzeitig versuchen auf den SPI Bus zuzugreifen, was nicht funktionieren wird. Insbesondere muss man hier beachten, das jeder Zugriff auf das LCD und die LEDs auch einen SPI Zugriff erfordert!

3.1.8. LC-Display

Das LC-Display ist ideal um Sensorwerte und Statusmeldungen auszugeben. Vor allem wenn der Roboter nicht am Rechner angeschlossen ist, kann das sehr nützlich sein um Informationen über den aktuellen Programmzustand zu erhalten.

Das LCD ist NICHT wie auf dem CCPRO Application Board angeschlossen, daher können (noch) nicht direkt die internen LCD Funktionen verwendet werden. Das soll in einer späteren Version des CCPRO Interpreters aber auch möglich sein.

Als allererstes muss man das LCD natürlich intialisieren:

```
void RP6_initLCD(void)

Sub void RP6_initLCD()
```

Das wird schon von RP6_CCPRO_Init() erledigt – man muss es also normalerweise nicht nocheinmal tun.

Das LCD wird im 4 Bit Modus betrieben. Daher muss man stets:

```
void setLCDD(byte lcdd)

Sub setLCDD(lcdd As Byte)
```

zweimal aufrufen um ein komplettes Datenbyte an das LCD zu übertragen. Das erledigen die beiden Funktionen:

```
void writeLCDCommand(byte cmd)

Sub writeLCDCommand(cmd As Byte)
```

und

```
void writeCharLCD(char ch)

Sub writeCharLCD(ch As Char)
```

Beide übertragen ein Byte an das LCD, bei writeLCDCommand wird es als Steuerbefehl interpretiert und bei writeCharLCD als Zeichencode.

Mit

```
void setCursorPosLCD(byte text_line, byte pos)
Sub setCursorPosLCD(text_line As Byte, pos As Byte)
```

wir der Text Cursor an eine bestimmte Position des LCDs gesetzt.

CompactC Beispiel:

```
setCursorPosLCD(1, 5);  // Setzt den Cursor in Zeile 1, Position 5
writeCharLCD(65);       // Schreibt ein „A“ an diese Position
```

Basic Beispiel:

```
setCursorPosLCD(1, 5)   ' Setzt den Cursor in Zeile 1, Position 5
writeCharLCD(65)        ' Schreibt ein „A“ an diese Position
```

```
void clearLCD()
Sub clearLCD()
```

Löscht den gesamten Inhalt des LCDs.

```
void clearPosLCD(byte text_line, byte pos, byte length)
Sub clearPosLCD(text_line As Byte, pos As Byte, length As Byte)
```

Löscht nur einen bestimmten Bereich des LCD. Beginnend in Zeile „text_line“ an Position „pos“ werden „length“ Zeichen gelöscht.

Möchte man mehrere Zeichen nacheinander an das LCD senden, kann man

```
void writeStringLCD(char text[])
Sub writeStringLCD(ByRef text As Char)
```

verwenden.

Möchte man mehrere Zeichen nacheinander an eine bestimmte Position schreiben, kann man

```
void writeLineLCD(char text[], byte text_line, byte pos)
Sub writeLineLCD(ByRef text As Char, text_line As Byte, pos As Byte)
```

verwenden.

Für alle String Funktionen gibt es Makros die die Anwendung stark vereinfachen:

```
printLCD(__STRING__)
printlnLCD(__STRING__, __LINE__, __POS__)
showScreenLCD(__STRING1__, __STRING2__)
printIntegerLCD(__INT__)
```

Diese Makros funktionieren ganz ähnlich wie diejenigen für die serielle Schnittstelle. Allerdings ist bei einem LCD natürlich die Position der Zeichen wichtig.

printLCD tut nichts anderes, als ein temporäres Array mit dem übergebenen konstanten Text zu initialisieren und diese dann mit writeStringLCD an das LCD zu senden.

printlnLCD ruft writeLineLCD auf.

ShowScreenLCD löscht den Inhalt des gesamten Displays und schreibt dann die beiden übergebenen Texte in Zeile 1 und Zeile2. Das ist natürlich nur für zweizeilige Displays brauchbar – möchte man ein Display mit 4 oder mehr Zeilen verwenden, muss man das entsprechend anpassen.

Schließlich gibt es mit printIntegerLCD noch ein äquivalent zu printInteger für die serielle Schnittstelle. Mit diesem Makro wird der Wert einer übergebenen Variable als Text an das LCD gesendet. Das ist sehr hilfreich wenn man Sensorwerte darstellen möchte.

Sie können sehr viele Beispiele zur Anwendung des LCDs in den Beispielprogrammen auf der CD-ROM bzw. auf der RP6 Homepage finden. Das LCD wird in jedem der Beispielprogramme mehr oder weniger intensiv verwendet.

Dieses Kapitel diente nur als grober Überblick über die Funktionen, die als Ergänzung zu den bereits im C-Control Interpreter integrierten Funktionen in der RP6CCLib vorhanden sind. Wie diese in einem kompletten Programm verwendet werden können, wird ausführlich in den Beispielprogrammen gezeigt.

Die RP6CCLib können Sie durchaus selbst verändern und bei Bedarf neue Funktionen hinzufügen. Es wäre in diesem Fall allerdings evtl. besser, eine neue Basic bzw. C Datei anzulegen und diese separat einzubinden. Wie das geht, wird ebenfalls in den Beispielprogrammen demonstriert. Dort wird eine weitere kleine Library erstellt, welche die Sensorauswertung und Steuerung des Roboters über den I²C Bus weiter vereinfacht. Dort gibt es dann Hilfsfunktionen wie RP6_rotate oder RP6_move, mit denen man den Roboter direkt um bestimmte Winkel rotieren oder bestimmte Strecken abfahren lassen kann. Auch fehlt es nicht an Funktionen, welche auf Änderungen wichtiger Sensordaten automatisch reagieren können (z.B. wenn das ACS ein Hindernis detektiert, wird eine bestimmte Funktion aufgerufen o.ä.).

Diese zusätzlichen Dinge werden hier NICHT weiter besprochen, da diese Funktionen in den Beispielprogrammen nach und nach hinzugefügt und beschrieben werden. Lesen Sie sich dazu bitte die Kommentare in den Beispielprogrammen durch!

4. Beispielprogramme

Auf der CD finden Sie jeweils 20 vollständig lauffähige Beispielprogramme in CompactC und Basic. Die Beispielprogramme für das CCPRO Modul sind ausführlich auf Deutsch kommentiert.

Diese Beispielprogramme demonstrieren die grundlegenden Funktionen des RP6 CCPRO M128 Erweiterungsmoduls. Genau wie beim Roboter selbst, stellen sie keinesfalls die optimale Lösung dar, sondern verstehen sich als Ausgangspunkt für eigene Programme. Das ist absichtlich so, damit Ihnen auch noch etwas zu tun bleibt – wäre ja langweilig einfach nur vorgefertigte Programme auszuprobieren...

Allgemein gibt es für die C-Control PRO schon einige Bespielprogramme im Internet und im Lieferumfang der CCPRO IDE. Allerdings muss man hier immer darauf achten, andere Beispielprogramme auch an die Hardware des RP6 CCPRO M128 Moduls anzupassen – sonst wird es oft nicht funktionieren (die offensichtlichsten Probleme sind andere Pinbelegungen, Verwendung von bereits anderweitig verwendeten Hardwaremodulen wie Timern usw.)!

Sie können Ihre eigenen Programme selbstverständlich auch mit anderen Anwendern über das Internet austauschen.

Die Beispielprogramme bauen aufeinander auf – d.h. „Example_01_HelloWorld" sollte man vor „Example_02_Beeper" angeschaut, ausprobiert und verstanden haben! Ab „Example_07_RP6_Sensors1" bis „Example_09_Behaviour4" wird nach und nach ein etwas komplexeres Beispielprogramm aufgebaut, in dem der Roboter Hindernissen mithilfe der Infrarotsensorik ausweichen kann.

Bitte beachten Sie, dass nicht alle der Beispiele für den RP6 sofort für Anfänger verständlich sind! Es wird empfohlen, zunächst mit den einfachen Programmen zu beginnen und sich langsam Schritt für Schritt vorzuarbeiten. Vor allem die normalen C-Control PRO Beispielprogramme sind für das Grundlagenverständnis sehr wichtig und sollten zuvor durchgearbeitet werden. Ausprobieren klappt dabei mit der RP6 Hardware leider nur in den seltensten Fällen, da die Programme für das C-Control PRO Application Board geschrieben wurden (andere Hardware, andere Pinbelegung). Aber die Kommentare lesen und versuchen nachzuvollziehen was genau gemacht wird ist zum Verständnis dennoch sehr hilfreich!

Die komplexeren Beispiele verwenden ausserdem das von der CCPRO bereitgestellte Multithreading. Hier (und für viele andere Dinge) empfiehlt es sich, die CCPRO Dokumentation zu lesen bevor man die Funktionsweise der RP6 spezifischen Beispielprogramme nachvollzieht!

Für eigene Programme kann man die Beispiele als Vorlage verwenden – dazu gibt es auch von jedem Beispielprogramm eine komplett unkommentierte Version.

Damit sind wir auch schon am Ende dieser kleinen Zusatzanleitung angelangt. Jetzt können Sie Ihre eigene Kreativität walten lassen, neue Programme schreiben und neue Sensoren am RP6 anbringen die Sie mit dem RP6-CCRPO-M128 steuern können - oder etwas ganz anderes damit anstellen!

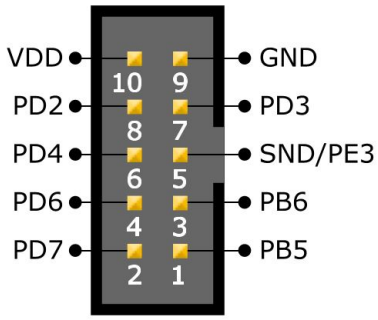
ANHANG

A – Anschlussbelegungen

In diesem Abschnitt finden Sie die Anschlussbelegungen der wichtigsten Stecker und Lötpads.

Der Anschluss der seriellen Schnittstelle hat genau die gleiche Pinbelegung wie auf dem Mainboard. Das gilt natürlich auch für die XBUS und USRBUS Anschlüsse!

I/O Ports:



Am „I/O" Wannenstecker sind einige der freien I/O Pins und die 5V Betriebsspannung verfügbar.

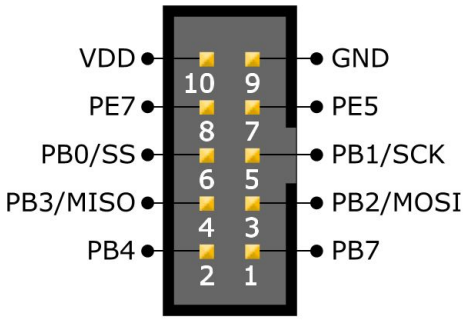
SND/PE3, PB5, PB6, PD2, PD3, PD4, PD6 und PD7.

PE3 (OC3A) ist mit dem Piezo Tongeber verbunden (SND), dieser kann per Jumper deaktiviert werden um so diesen I/O Pin freizugeben. PB6 und PB5 sind zusätzlich auf Anschlüssen (z.B. für Servos verwendbar) im vorderen Bereich der Platine verfügbar. Natürlich können diese Pins nur für eine Sache gleichzeitig verwendet werden – entweder als normaler I/O Pin ODER als Servo Ausgang. Also NICHT die jeweiligen Pins auf beiden Steckverbindern gleichzeitig verwenden!

ACHTUNG: Verbinden Sie lieber nicht die Betriebsspannungspins eines anderen Erweiterungsmoduls (z.B. eines Experimentiermoduls) mit den Betriebsspannungspins an diesen Steckern, um Masseschleifen zu vermeiden.

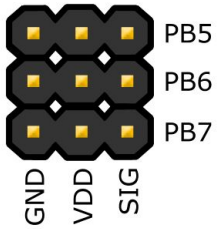
SPI Bus und I/O Ports:

Am SPI Bus Anschluss können bis zu 5 SPI kompatible Chips angeschlossen werden (auch mehr, mit externen Adressdekodern). Der SPI Bus kann z.B. für schnelle Schieberegister, EEPROM und FLASH Speicher, D/A und A/D Wandler o.ä. verwendet werden. Es sind 5 I/O Pins verfügbar (PB0/SS ist nur im SPI Slave Modus für den SPI Bus reserviert und kann sonst auch als I/O verwendet werden), die entweder für „Chip Select" Signale verwendet werden können, oder als ganz normale I/O Pins. Die sind PB0, PB4, PB7, PE5 und PE7.



PB1, PB2 und PB3 können NICHT als normale I/O Pins verwendet werden, auf dem Modul ist hier bereits ein Schieberegister angeschlossen um das LCD und einige der LEDs anzusteuern (wenn man diese Komponenten nicht benötigt, wäre allerdings auch eine Nutzung als normale I/O Pins denkbar. Bitte den Schaltplan beachten!). PB7 ist zusätzlich auf einem der Servo Anschlüsse verfügbar, kann also nur entweder für einen Servo ODER als Chipselect Leitung verwendet werden.

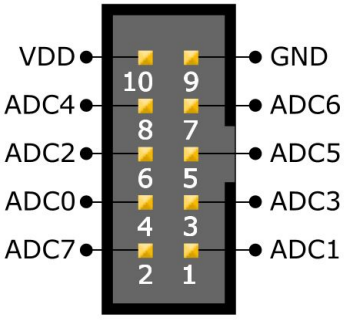
Servos



Wie bereits erwähnt, sind PB5, PB6, und PB7 auch an 3 poligen Servo Anschlüssen verfügbar. Hier kann man direkt Servos mit passender Steckerbelegung anschließen.

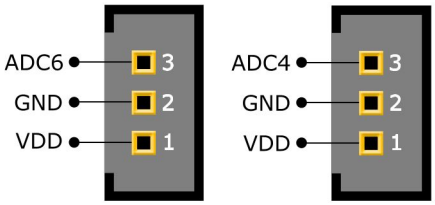
ACHTUNG: *Es gibt verschiedene Servo Varianten! Bitte vorher die Anschlussbelegung überprüfen und auf korrekte Polung achten!*

ADC Kanäle:

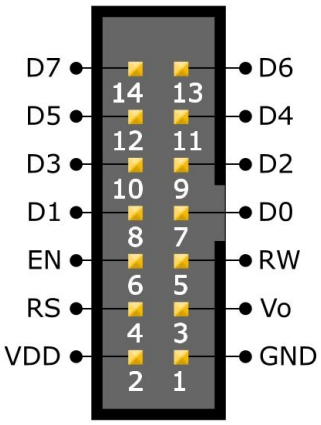


Die 8 freien ADC Kanäle (die natürlich auch als I/O Pins verwendbar sind) sind zusammen mit der Betriebsspannung am 10 poligen ADC Stecker verfügbar.

Zwei der ADCs (ADC4 und ADC6) sind zusätzlich auch auf zwei 3 poligen Steckern verfügbar um z.B. direkt analoge IR Distanzsensoren anzuschließen.



LCD Port:



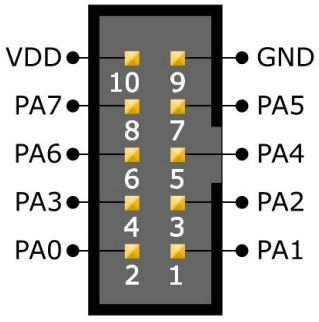
Wenn Sie nicht das Standard LCD einsetzen möchten, können Sie anhand der nebenstehenden Anschlussbelegung ein eigenes 14 poliges Kabel für das LCD zusammenbauen.

Die Leitungen D0, D1, D2 ,D3, RW sind fest mit GND verbunden, da wir das LCD nur im 4 Bit Modus betreiben und nicht davon zu lesen brauchen (RW ist auch fest auf Masse).

Achten Sie unbedingt auf korrekte Anschlussbelegung und darauf den Stecker nicht spiegelverkehrt anzuschließen!

Die Bezeichnungen der einzelnen Pins können je nach Hersteller auch anders sein, aber normalerweise sind die Bezeichnungen mit den hier verwendeten identisch und Sie können die Pins 1:1 mit dem Display verbinden!

PORTA und PORTC

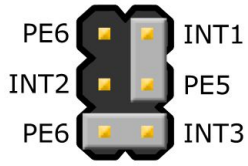


Die Stecker für PORTA und PORTC sind NICHT bestückt. Diese sind nur optional, falls man tatsächlich sehr viele I/O Pins benötigen sollte, aber auf das externe SRAM verzichten kann. Dieses kann man ggf. über einen ebenfalls unbestückten Jumper deaktivieren (die Verbindung dieses Jumpers ist per Leiterbahn auf der Rückseite der Platine gesetzt – diese kann man leicht mit einem Cutter Messer durchtrennen und anschließend den Jumper auflöten).

Die Anschlussbelegung von PORTC ist auf dem Mainboard aufgedruckt. Hier kann bei Bedarf z.B. eine 9 polige einreihige Stiftleiste aufgelötet werden.

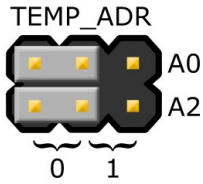
Jumper

Auf dem Modul sind einige Jumper zu finden um die Komponenten auf dem Modul zu konfigurieren.



Interrupts:

Dieser Jumperblock kann verwendet werden um PE6 und PE5 mit den Interrupt Leitungen des XBUS zu verbinden. Das kann z.B. verwendet werden um ständiges Abfragen („Polling") von I²C Slaves zu vermeiden. Stattdessen setzen diese bei Zustandsänderungen eine der Interruptleitungen auf Low Pegel. In der Abbildung ist PE6 mit INT3 verbunden und PE5 mit INT1, dies ist die **Standard Konfiguration! INT1 muss mit PE5 verbunden sein, damit die Beispielprogramme korrekt funktioneren!** Sie sollten PE6 besser nicht gleichzeitig mit zwei Interrupt Leitungen verbinden, auch wenn dies mit dem Jumperblock möglich wäre!



I²C Bus Adresse Temperatursensor:

Zwei Bits (A0 und A2) der Adresse des Temperatursensors können verändert werden um mögliche Konflikte (d.h. identische Adressen) mit anderen Geräten am I²C Bus beheben zu können.

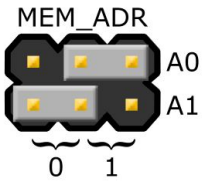
Bitte das Datenblatt des Temperatursensors beachten!

Die Adresse des Temperatursensors lautet in binärer Schreibweise: 1001[A2]0[A0]0

Eigentlich ist die Adresse nur 7 Bit lang also 1001000 --> 0x48. Im Programm muss aber direkt auch das Schreib/Lese Bit mit eingeplant werden (wird automatisch gesetzt), daher 10010000 --> 0x90 (144 Dezimal).

Die Abbildung zeigt die **Standard Einstellung** der Adresse auf 0x90.

Weitere mögliche Adressen sind: 0x92, 0x98 und 0x9A.



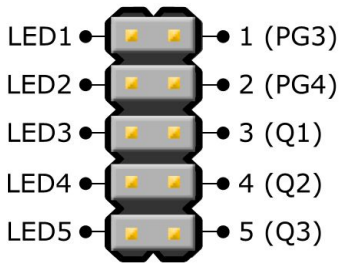
I²C Bus Adresse EEPROM:

Auch zwei Bits (A0 und A1) der Adresse eines optional aufsteckbaren EEPROMs können verändert werden.

Bitte das Datenblatt des EEPROMs beachten! Das Adressierungsschema kann hier je nach Hersteller durchaus abweichen!

Die Adresse des EEPROMs lautet in binärer Schreibweise: 10100[A1][A0]0

Also sind die möglichen Adressen: 0xA6, 0xA4, 0xA2 und 0xA0 (160 Dezimal)



LEDs:

Die LEDs lassen sich per Jumper deaktivieren. An diesem Jumperfeld kann man dann direkt über Steckverbinder o.ä. die I/Os bzw. Schaltausgänge verwenden. LED1 und 2 sind mit den I/O Ports PG3 und PG4 verbunden, LED3, 4 und 5 nur mit reinen Ausgängen des Schieberegisters an dem auch das LCD angeschlossen ist.

B – Entsorgungs- und Sicherheitshinweise

Entsorgungshinweise

Der RP6 und zugehörige Komponenten dürfen nicht im normalen Hausmüll entsorgt werden! Der RP6 und Zubehör muss wie alle Elektrogeräte beim Wertstoffhof oder an einer anderen Elektro-Altgeräte Sammelstelle abgegeben werden!

Falls Sie Fragen dazu haben, wenden Sie sich an Ihren Händler.

Sicherheitshinweise für Akkus und Batterien

Akkus und Batterien gehören nicht in Kinderhände! Lassen Sie Batterien/Akkus nicht offen herumliegen, es besteht die Gefahr, dass diese von Kindern oder Haustieren verschluckt werden. Suchen Sie im Falle eines Verschluckens sofort einen Arzt auf!

Ausgelaufene oder beschädigte Batterien können bei Berührung mit der Haut Verätzungen verursachen, benutzen Sie deshalb in diesem Fall geeignete Schutzhandschuhe. Achten Sie darauf, dass die Batterien/Akkus nicht kurzgeschlossen oder ins Feuer geworfen werden. Normale Batterien dürfen außerdem nicht aufgeladen werden! Es besteht Explosionsgefahr! Nur darauf ausgelegte, wiederaufladbare Akkumulatoren wie z.B. NiMH Akkus dürfen mit einem passenden Ladegerät geladen werden!

Entsorgungshinweise für Akkus und Batterien

Akkus und Batterien gehören genausowenig in den Hausmüll wie der Roboter selbst! Sie als Endverbraucher sind gesetzlich (Batterieverordnung) zur Rückgabe aller gebrauchten Batterien und Akkus verpflichtet. Eine Entsorgung über den Hausmüll ist untersagt!

Geben Sie defekte/alte Akkus bzw. leere Batterien deshalb nur bei Ihrem Händler, oder einer Batteriesammelstelle Ihrer Gemeinde ab! Sie können gebrauchte Akkus und Batterien auch überall dort abgeben, wo Batterien und Akkus verkauft werden.

Sie erfüllen damit die gesetzlichen Verpflichtungen und leisten Ihren Beitrag zum Umweltschutz!