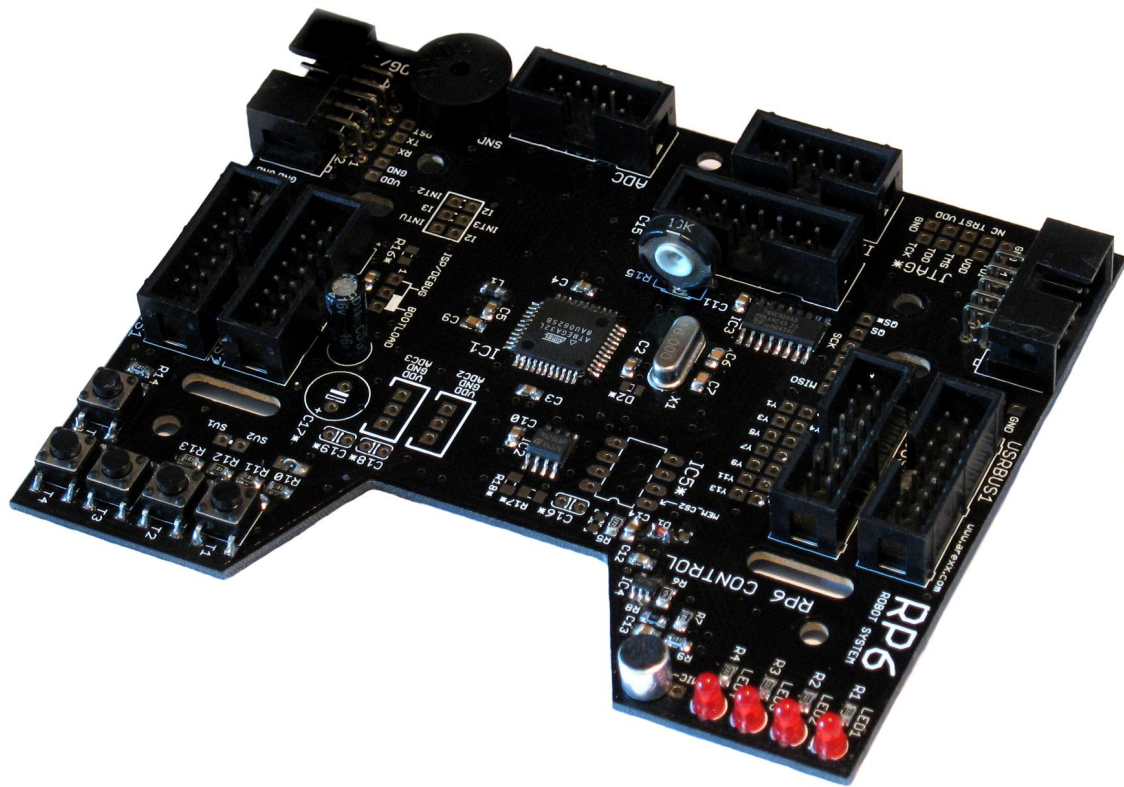


# RP6 ROBOT SYSTEM

## RP6 CONTROL M32 Erweiterungsmodul



**RP6-M32**

©2007 AREXX Engineering

[www.arexx.com](http://www.arexx.com)

# RP6 CONTROL M32

## Bedienungsanleitung

- Deutsch (German) -

Version RP6-M32-DE-20071031



### **WICHTIGE INFORMATION! Bitte unbedingt lesen!**

**Bevor Sie dieses RP6 Erweiterungsmodul in Betrieb nehmen, lesen Sie bitte diese Anleitung UND die RP6 ROBOT SYSTEM Bedienungsanleitung vollständig durch! Sie erläutert Ihnen die korrekte Verwendung und weist auf mögliche Gefahren hin! Weiterhin enthält sie wichtige Informationen, die für viele Anwender keineswegs offensichtlich sein dürften. Die RP6 CONTROL M32 Anleitung ist nur eine Ergänzung!**

**Bei Nichtbeachtung dieser Anleitung und der RP6 ROBOT SYSTEM Anleitung erlischt jeglicher Garantieanspruch! Weiterhin übernimmt AR-EXX 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

### ©2007 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/Socket 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 CONTROL M32 Erweiterungsmodul .....	5
1.1. Technischer Support .....	6
1.2. Lieferumfang .....	6
1.3. Features und technische Daten .....	7
2. Montage des Erweiterungsmoduls .....	9
3. RP6 CONTROL Library .....	11
3.1.1. Mikrocontroller initialisieren.....	12
3.1.2. Status LEDs.....	12
3.1.3. Taster.....	13
3.1.4. Beeper.....	13
3.1.5. Mikrofonsensor.....	14
3.1.6. LC-Display.....	14
3.1.7. SPI Bus und SPI EEPROM.....	16
3.1.8. ADCs.....	18
3.1.9. I/O Ports.....	18
4. Beispielprogramme .....	20
ANHANG .....	27
A – Anschlussbelegungen.....	27

## **1. Das RP6 CONTROL M32 Erweiterungsmodul**

Mit dem RP6 CONTROL M32 (oder kurz „RP6-M32“) Erweiterungsmodul können Sie dem Roboter einen zweiten Atmel ATMEGA32 Mikrocontroller hinzufügen, der jedoch doppelt so schnell getaktet ist wie der Controller auf dem Mainboard. Auch sonst steht dem Anwender auf dem RP6-M32 mehr Rechenzeit zur Verfügung, da der Controller nicht auch noch mit Motorregelung, ACS, IRCOMM, etc. beschäftigt ist.

Mit dem externen 32KB SPI EEPROM befindet sich ein sehr oft wiederbeschreibbarer (1 Mio. Zyklen) Festspeicher auf dem Modul, den man beispielsweise zum Datenloggen oder als Programmspeicher für Bytecode Interpreter verwenden kann (wie z.B. die NanoVM für Java). Optional kann man sogar noch einen 8 poligen DIP Sockel auf das Modul löten und ein zweites EEPROM im DIP 8 Gehäuse hinzufügen.

Weitere interessante Möglichkeiten bieten die Eingabetaster, die LEDs, der Piezo Piepser und das optional erhältliche LC-Display. Damit kann man den Roboter auch direkt bedienen und z.B. ein kleines Auswahlménü programmieren um verschiedene Programme über die Taster zu starten – und natürlich auch Messwerte und Statusmeldungen anzeigen. Der Piepser kann verschiedene Töne erzeugen und z.B. eine Begrüßungsmelodie abspielen wenn das Programm gestartet wird oder bei niedriger Akkuspannung warnen.

Eigene Schaltungen auf den Lochraster Erweiterungsmodulen kann man mit den 14 freien I/O Ports ansteuern, die auf zwei 10 polige Wannenstecker herausgeführt sind. Von den 14 I/Os sind 6 als Analog/Digital Wandler Kanäle verwendbar.

Abgerundet wird die Ausstattung des Moduls mit dem Mikrofonsensor, der auch schon auf dem alten CCRP5 vorhanden war. Damit kann man den RP6 beispielsweise starten indem man in die Hände klatscht oder ähnliche Dinge. Die Schaltung ist als „Peak Detektor“ ausgelegt, es werden also nur die Maximalwerte ermittelt. So kann die Lautstärke von Umgebungsgeräuschen grob gemessen und entsprechend darauf reagiert werden. (das klappt natürlich nur dann gut, wenn die Motoren nicht laufen weil über Körperschall vom Roboter selbst erzeugte Geräusche besonders gut vom Mikrofon detektiert werden...)

**Bevor Sie mit dem RP6-M32 loslegen**, sollten Sie sich unbedingt mit dem Roboter an sich vertraut machen und alle Beispielprogramme des Roboters OHNE montiertes RP6-M32 Erweiterungsmodul ausprobieren! Diese Anleitung versteht sich als kleine Ergänzung zur großen RP6 Bedienungsanleitung. Lesen Sie diese auf jeden Fall bevor Sie mit dem RP6-M32 anfangen!

**Ein wichtiger Hinweis für Anfänger:** Für das RP6-M32 geschriebene Programme laufen natürlich NICHT korrekt auf dem Mikrocontroller der Basiseinheit und umgekehrt (komplett andere Pinbelegung und Taktfrequenz)!



**ACHTUNG: Wenn Sie ein Programm in den falschen Controller laden, könnte dieser oder die gesteuerten Schaltungen durchaus beschädigt werden! Beispielsweise wenn ein I/O Pin normalerweise als Eingang verwendet, im Programm für den falschen Controller aber als Ausgang geschaltet und dann aufgrund der angeschlossenen Schaltung überlastet wird!**

Normalerweise passiert nichts schlimmes wenn Sie sich mal vertun sollten, aber es kann hier für nichts garantiert werden! Der RP6Loader kann nicht unterscheiden für welchen Controller die Programme bestimmt sind, da die Hexfiles alle gleich aufgebaut sind. Er verhindert also nicht, dass Sie Programme in den falschen Controller laden! Nutzen Sie die Funktion des RP6Loaders, verschiedene Kategorien anzulegen! Für jedes Erweiterungsmodul eine eigene Kategorie ...

### 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](mailto: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/>**

und auf der Homepage des Roboters:

**<http://www.arexx.com/rp6>**

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 CONTROL M32 Packung vorfinden:

- Fertig aufgebautes RP6-M32 Modul
- 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 RP6 CD-ROM. Aktualisierte Versionen der Software und dieser Anleitung gibt es auf unserer Homepage!

### **1.3. Features und technische Daten**

Dieser Abschnitt gibt einen Überblick darüber, was das RP6 CONTROL M32 zu bieten hat und dient gleichzeitig der Einführung einiger Begriffe und Bezeichnungen von Komponenten des Moduls.

#### ***Features, Komponenten und technische Daten des RP6 CONTROL M32:***

- **Leistungsfähiger Atmel ATMEGA32 8-Bit Mikrocontroller**

- ◇ Geschwindigkeit 16 MIPS (=16 Millionen Instruktionen pro Sekunde) bei 16MHz Takt, also doppelt so schnell getaktet wie der Controller auf dem RP6 Mainboard!
- ◇ Speicher: 32KB Flash ROM, 2KB SRAM, 1KB EEPROM
- ◇ Frei in C programmierbar (mit WinAVR / avr-gcc)!
- ◇ ... und vieles mehr (s. Datenblatt)!

- **Externes 32KB SPI EEPROM**

- ◇ Sehr schnelles SPI Interface (8MHz Taktrate)
- ◇ Jede Speicherzelle mindestens 1.000.000 mal wiederbeschreibbar
- ◇ ... mehr technische Infos im Datenblatt auf der CD-ROM (AT25256A)!
- ◇ Ideal für Datenlogging oder als Speicher für Bytecode Interpreter (z.B. eine Java VM wie die NanoVM: <http://www.harbaum.org/till/nanovm/> . Allerdings müsste diese erst noch angepasst werden damit auch das externe EEPROM verwendet wird... )

- **I<sup>2</sup>C-Bus Erweiterungsanschlüsse**

- ◇ Kann beliebige I<sup>2</sup>C Bus Slaves ansteuern.
- ◇ Der MEGA32 auf dem Modul 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).

- **Mikrofonsensor**

- ◇ um Geräusche zu detektieren, z.B. in die Hände klatschen o.ä.

- **Piezo Beeper**

- ◇ um einfache Töne und Melodien zu erzeugen
- ◇ Signalgeber z.B. um Fehler oder Statuswechsel zu melden

- **4 Status LEDs**

- **5 Eingabetaster**

- **LC-Display Port**

- ◇ Zum Anschluss eines 16x2 Zeichen LC-Displays. Auch andere LC-Displays lassen sich hier anschließen, z.B. 16x4 o.ä., allerdings lassen sich diese dann nur mit zwei Distanzbolzen befestigen und könnten an einer Seite überstehen... Am besten vor dem Kauf genau ausmessen und passendes Montagematerial mitbestellen! Ausserdem müssen Sie bei anderen Display Formaten als 16x2 evtl. ein paar Anpassungen in der Library vornehmen, damit es korrekt funktioniert (Hauptsächlich die Initialisierung des Displays und evtl. die Cursor Positionierung). Die Beispielprogramme sind alle auf ein 16x2 Zeichen Display zugeschnitten. Das lässt sich aber leicht ändern.
- ◇ Das Display kann z.B. Textmeldungen/Menüs, Programmzustände oder Sensorwerte anzeigen.

- **14 freie I/O Ports zur Ansteuerung eigener Schaltungen und Sensoren**

- ◇ **6** davon sind als **Analog/Digital Wandler (ADC) Kanäle** verwendbar

- **Bis zu 3 externe Interrupts am XBUS Anschluss lassen sich verwenden.**

- **Anschluss für das USB PC Interface** für den Programupload

- ◇ Der Programupload läuft genau wie beim Roboter selbst, schnell und einfach über das USB Interface und die komfortable RP6Loader Software.

Weiterhin werden wie beim Roboter einige C Beispielprogramme sowie eine Funktionsbibliothek mitgeliefert, welche die Programmierung stark erleichtert.

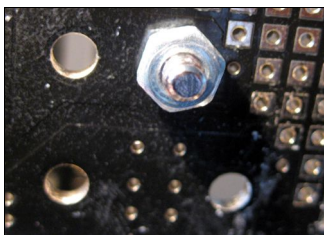
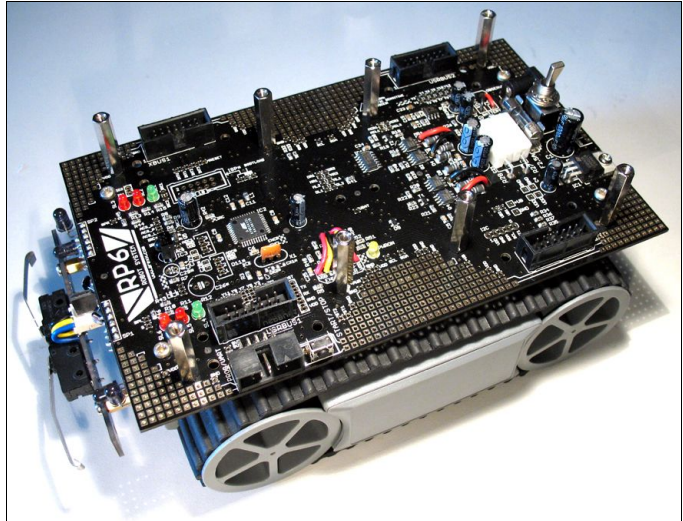
Auf der Website zum Roboter werden demnächst evtl. weitere Programme und Updates für den Roboter und seine Erweiterungsmodule zur Verfügung stehen. Sie können natürlich auch gerne Ihre eigenen Programme über das Internet mit anderen RP6 Anwendern austauschen! Die RP6ControlLibrary und die Beispielprogramme stehen unter der Open Source Lizenz GPL!

## 2. Montage des Erweiterungsmoduls

Wie Sie das Modul auf dem Roboter befestigen, hängt natürlich 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. Eventuell 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...

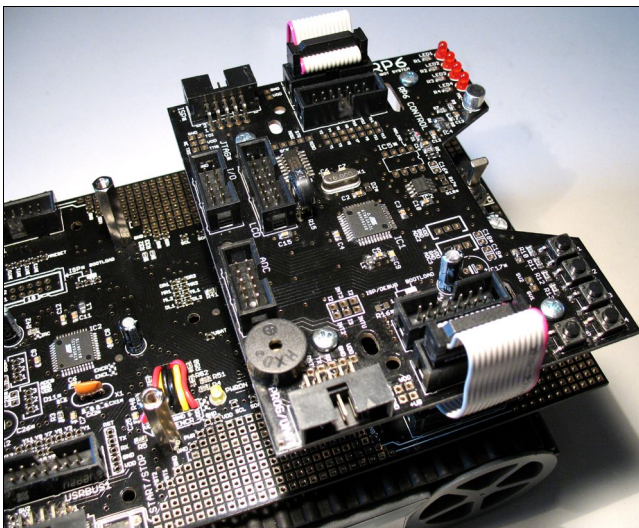


Dann können Sie die vier 25mm M3 Distanzbolzen nacheinander mit M3 Muttern in den Befestigungslöchern im Mainboard festschrauben wie auf dem Foto gezeigt.

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

Anschließend setzen 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 CONTROL M32 auf dem hinteren Erweiterungsstapel auf dem Roboter zu montieren – als oberstes Modul, damit die Taster und ggf. das Display zugänglich bleiben. 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 auf der nächsten Seite für eine Beispielkonfiguration).

## RP6 ROBOT SYSTEM - 2. Montage des Erweiterungsmoduls

Wenn Sie zusätzlich das 16x2 Zeichen LC-Display gekauft haben, sollten Sie es VOR der Montage auf dem Roboter an das Erweiterungsmodul anschließen und montieren.

Das 14 polige Flachbandkabel des Displays ist sehr flexibel und kann problemlos gefaltet werden. Damit das Kabel gut unter das Display passt, sollten Sie es für das RP6 CONTROL M32 Erweiterungsmodul in etwa wie auf dem Foto gezeigt falten.

Dann kann das Display z.B. mit 20mm oder 25mm Distanzbolzen, passenden Muttern und Schrauben am Erweiterungsmodul befestigt werden.

Sie können auch ein beliebiges anderes Text LC-Display mit HD44780 kompatibel Controller verwenden. Sie müssen dazu nur ein eigenes Kabel an das Display anlöten.

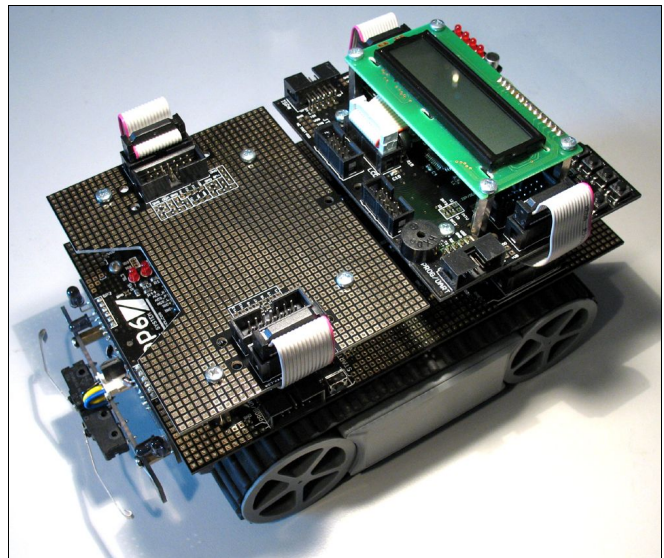
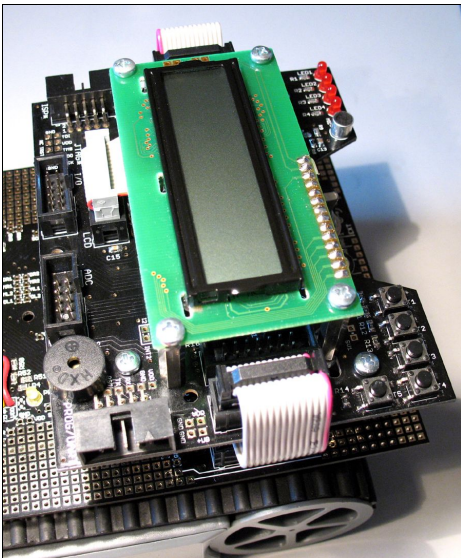
*Dabei bitte unbedingt auf die richtige Anschlussbelegung achten!*

Es werden nicht unbedingt 4 Distanzbolzen benötigt wie auf dem Foto! Zwei Stück (beide vorne oder hinten) reichen bereits aus, um das Display zu befestigen.



*Tipp: Es werden zwar keine zusätzlichen Distanzbolzen für die Display Montage mitgeliefert, aber bei jedem Erweiterungsmodul sind vier Distanzbolzen, Muttern und Schrauben inklusive. Normalerweise befestigt man ein Erweiterungsmodul wie auf den Bildern gezeigt mit 4 Distanzbolzen. Es reichen aber auch drei – zwei vorne und einer hinten in der Mitte! Mit Experimentiermodul und RP6-M32 hätten Sie so zwei 25mm Distanzbolzen, Schrauben und Muttern für das Display übrig...*

Fertig auf dem Roboter montiert, könnte das z.B. wie folgt aussehen:

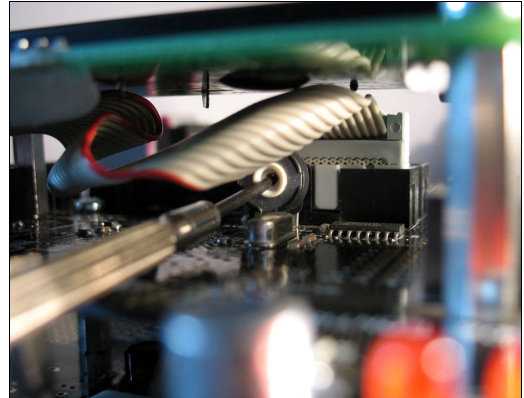


Die zwei 10 poligen Erweiterungsstecker mit den freien I/O Ports kann man nun einfach über kleine 10 polige Flachbandkabel mit dem Erweiterungsmodul verbinden und die I/Os und ADCs dort zur Auswertung von Sensoren o.ä. verwenden. Das geht natürlich auch mit Modulen die auf anderen Ebenen montiert sind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

Jetzt können Sie kurz einen kleinen **Funktionstest** durchführen:

Zunächst das am PC angeschlossene USB Interface über das 10 polige Flachbandkabel mit dem PROG/UART Anschluss auf dem RP6-M32 verbinden und den RP6Loader starten. Dann den Roboter anschalten – auf dem LC-Display sollte eine Textnachricht erscheinen und eine der LEDs sollte blinken. Hat das geklappt, klicken Sie im RP6Loader bitte auf Verbinden – es sollte in der Statusbox eine Meldung erscheinen „Connected to RP6 Control“. Wenn auch das geklappt hat, können Sie nachdem Sie die Anleitung zuende gelesen haben mit den Beispielprogramm anfangen!

Wenn das Display nichts oder nur zwei Reihen komplett dunkle Kästchen anzeigen sollte, aber die LEDs blinken und das Verbinden mit dem RP6Loader klappt, müssen Sie den Kontrast des Displays einstellen (oder Sie haben ein anderes Display verwendet und die Anschlussbelegung ist falsch...). Das geht mit Poti R16 auf der Platine. Das Poti können Sie mit einem kleinen Schlitzschraubendreher verstellen. Kreuzschlitz geht auch, klappt dann aber mit vielen Schraubendrehern nicht, weil diese keinen guten Halt im Poti finden...



Damit das Poti besser erreichbar ist, können Sie das Display wieder losschrauben (oder es erst gar nicht festschrauben bevor es nicht korrekt eingestellt ist). Der Stecker muss dabei aber angeschlossen bleiben! Dann können Sie mit einem Schraubendreher am Poti drehen, während der Roboter eingeschaltet ist.

Oder Sie lassen das Display einfach festgeschraubt und nehmen einen kleinen Schraubendreher dafür wie auf dem Foto gezeigt. **Dabei unbedingt darauf achten keine Bauteile und Kontakte auf der Platine mit dem Schraubendreher zu berühren!** Am besten erst den Roboter kurz ausschalten, dann den Schraubendreher in Position bringen, den Roboter wieder anschalten und schließlich den Kontrast einstellen...

## 3. RP6 CONTROL Library

Wie für den Roboter selbst, gibt es auch für das RP6 CONTROL M32 eine Funktionsbibliothek mit vielen hilfreichen Funktionen, die es Einsteigern bedeutend einfacher machen. Die Bibliothek heisst RP6ControlLibrary oder kürzer RP6ControlLib. Die Stopwatch, Delay, UART und I<sup>2</sup>C-Bus Funktionen sind identisch mit denen der normalen RP6Library. Es werden für UART und I<sup>2</sup>C-Bus sogar die identischen Dateien verwendet. Diese liegen in der RP6Library im Unterordner RP6common. Also werden wir diese hier nicht nochmal beschreiben – schauen Sie sich bitte das entsprechende Kapitel in der RP6 Bedienungsanleitung und die Beispielprogramme an! Hier beschreiben wir nur Funktionen, die es nur für das RP6Control gibt, oder die etwas anders funktionieren als in der RP6Lib.

Trotz der vielen vorgefertigten Funktionen, ist die RP6ControlLib nur eine Grundlage! Perfekt ist auch diese Library bei weitem noch nicht. Man kann noch vieles verbessern und hinzufügen! Hier sind Ihre eigenen Programmierfähigkeiten gefordert!

### 3.1.1. Mikrocontroller initialisieren

```
void initRP6Control(void)
```

Wie schon von der RP6Lib bekannt, muss diese Funktion IMMER als erstes in der Main Funktion aufgerufen werden! Sie heisst hier nur etwas anders...

Die Funktion initialisiert die Hardwaremodule des Mikrocontrollers auf dem RP6-M32. Nur wenn Sie diese Funktion als erstes aufrufen, wird der Mikrocontroller korrekt arbeiten! Ein Teil wird zwar schon vom Bootloader initialisiert, aber nicht alles.

Beispiel:

```
1  #include "RP6ControlLib.h"
2
3  int main(void)
4  {
5      initRP6Control(); // Initialisierung - IMMER ALS ERSTES AUFRUFEN!
6
7      // [...] Programmcode...
8
9      while(true);      // Endlosschleife
10     return 0;
11 }
```

**Jedes Programm für das RP6 CONTROL M32 muss mindestens so ausschauen! Die Endlosschleife in Zeile 9 ist notwendig, um ein definiertes Ende des Programms zu garantieren!** Ohne diese Schleife könnte sich das Programm anders verhalten wie erwartet! *Genau wie mit dem Controller auf dem Mainboard!*

### 3.1.2. Status LEDs

Die LEDs lassen sich ähnlich wie auf dem Mainboard steuern, allerdings sind nur vier LEDs verfügbar und die Bezeichnungen sind etwas anders, da die LEDs an einem externen Schieberegister angeschlossen sind, welches auch gleichzeitig das LC-Display mit ansteuert. Das 8-Bit Schieberegister wird als „External Port“ bezeichnet.

Auch für das RP6-M32 gibt es die Funktion „setLEDs“:

```
void setLEDs(uint8_t leds)
```

Beispiel:

```
setLEDs(0b0000); // Dieser Befehl schaltet alle LEDs aus.
setLEDs(0b0001); // Und dieser schaltet LED1 an und alle anderen aus.
setLEDs(0b0010); // LED2
setLEDs(0b0100); // LED3
setLEDs(0b1010); // LED4 und LED2
```

Eine alternative Möglichkeit ist folgendes:

```
externalPort.LED1 = true; // LED1 im „External Port“ Register aktivieren
externalPort.LED2 = false; // LED2 im „External Port“ Register deaktivieren
outputExt();              // Änderungen übernehmen!

// Mit der Funktion outputExt() wird der Inhalt der Variablen externalPort
// an das Schieberegister gesendet – analog zu updateLEDs() aus der RP6Lib.
// Hier werden allerdings auch Änderungen der LCD Datenleitungen mitgesendet.
```

### 3.1.3. Taster

Anders als z.B. die Bumper, sind die 5 Taster auf dem RP6-M32 an einen ADC Kanal angeschlossen. Das hat den Vorteil, das nur ein Pin für alle 5 Taster benötigt wird. Der Nachteil ist allerdings, dass bei der einfachen Schaltung mit 5 gleichen Widerständen (s. Schaltplan auf der CD-ROM) die wir hier verwenden nur ein Taster gleichzeitig gedrückt werden kann. Das reicht für Bedientaster aber völlig aus!

```
uint8_t getPressedKeyNumber(void)
```

Diese Funktion ermittelt welcher Taster gerade gedrückt ist. Dazu wird der ADC ausgelesen und mit einigen voreingestellten Schwellwerten verglichen. Diese Schwellwerte muss man evtl. in der Library anpassen damit es auf dem eigenen RP6-M32 korrekt funktioniert! Der gemessene ADC Wert kann nämlich aufgrund der üblichen Fertigungstoleranz von Widerständen abweichen. Die Schwellwerte sind direkt in der Funktion `getPressedKeyNumber` zu finden.

```
uint8_t checkPressedKeyEvent(void)
```

Diese Funktion prüft ob ein Taster gedrückt worden ist und gibt dann ein einziges mal die Tastennummer zurück – im Gegensatz zu `getPressedKeyNumber`, wo die Tastennummer ständig zurückgegeben wird. Das ist nützlich um in der Hauptschleife die Tasten abzufragen ohne den Programmfluss zu unterbrechen.

Ähnlich ist es auch mit der Funktion:

```
uint8_t checkReleasedKeyEvent(void)
```

Hier wird allerdings der Tastenwert erst dann genau einmal zurückgegeben wenn die Taste gedrückt und danach auch wieder losgelassen wurde. Auch diese Funktion blockiert den normalen Programmfluss nicht – man muss nicht mit einer Schleife darauf warten bis die Taste wieder losgelassen wurde.

### 3.1.4. Beeper

Der Signalgeber auf dem RP6-M32 kann mit der Funktion

```
void beep(uint8_t pitch, uint16_t time)
```

gesteuert werden. Allerdings ist diese Funktion nicht blockierend – d.h. Sie setzt nur die Tonhöhe und die Zeitspanne für die dieser Ton erzeugt werden soll, und wird dann verlassen. Der Beeper wird nach der vorgegebenen Zeitspanne automatisch abgeschaltet. Allerdings überschreibt jeder weitere Aufruf dieser Funktion die Einstellungen. Um Melodien oder einzelne Töne zu erzeugen ist es daher meist einfacher, das Makro

```
sound(pitch, time, delay)
```

zu verwenden. Hier können Tonhöhe, Zeitspanne und Pause nach dem Ton angegeben werden. Es wird `mSleep` verwendet um die Pausen zu erzeugen – also dauert es `time + delay` Millisekunden bis das Programm fortfährt.

Wenn man gar keine Pause oder Tondauer einstellen will, kann man mit

```
void setBeeperPitch(uint8_t pitch)
```

nur die Tonhöhe einstellen. Das ist für kontinuierliche Tonfolgen (z.B. Sirengeräusch o.ä.) nützlich. Achtung: Bei allen Beeper Funktionen liegt der zulässige Bereich von „pitch“ zwischen 0 und 255, wobei 255 die maximale Frequenz ist.

### 3.1.5. Mikrofonsensor

Das RP6 CONTROL kann nicht nur Schall erzeugen, sondern auch darauf reagieren. Zwar nicht auf die Tonhöhe, aber auf die Lautstärke. Man kann z.B. den Roboter bei lauten Geräuschen losfahren lassen.

Die Schaltung ist als „Peak Detektor“ ausgelegt. Es wird über einen variablen Zeitraum die Amplitude des Mikrofondsignals gemessen und der Maximalwert gehalten. Dann kann der Mikrocontroller mit einem ADC den Maximalwert messen und danach löschen. Der Maximalwert wird einfach kurz in einem kleinen Kondensator gespeichert und zum „Löschen“ des Maximalwerts wird dieser Kondensator wieder entladen.

Mit der Funktion

```
void dischargePeakDetector(void)
```

muss man den Kondensator zunächst entladen. Danach kann man in festen Intervallen mit der Funktion

```
uint16_t getMicrophonePeak(void)
```

den aktuell gemessenen Maximalwert ermitteln. Die Funktion ruft nachdem der Wert gemessen wurde direkt `dischargePeakDetector()` auf.

In einem der Beispielprogramme sieht man wie man das anwenden kann.

### 3.1.6. LC-Display

Das LC-Display ist ideal um Sensorwerte und Statusmeldungen auszugeben, während der Roboter nicht am Rechner angeschlossen ist. Die Ausgabe auf das LC-Display funktioniert ähnlich wie bei der seriellen Schnittstelle – aber natürlich gibt es ein paar kleine Besonderheiten. Schauen Sie sich am besten die Beispielprogramme an, dann wird schnell klar wie man das LCD verwenden kann.

```
void initLCD(void)
```

Diese Funktion muss immer zu Beginn des Programms aufgerufen werden, um das LCD zu initialisieren.

```
void setLCDD(uint8_t lcdd)
```

Diese Funktion (und `write4BitLCDDData`) benötigen Sie normalerweise nicht – wir beschreiben Sie hier nur um kurz zu erläutern wie das Display angesteuert wird.

Das LCD wird im 4-Bit Modus betrieben. Es werden daher nur vier Datenleitungen und zwei Steuerleitungen benötigt (Enable (EN) und Register Select (RS), Read/Write (R/W) ist dauerhaft auf Masse geschaltet wodurch das LCD ausschließlich beschrieben werden kann. Lesen ist nicht möglich und auch nicht notwendig). Die vier Datenleitungen sind wie die LEDs am Schieberegister angeschlossen, um Ports zu sparen. Analog zur `setLEDs` Funktion, setzt `setLCDD` die Datenleitungen des LCDs. Allerdings setzt diese Funktion auch kurz das Enable Signal, damit das LCD die Daten übernimmt.

```
void write4BitLCDDData(uint8_t data)
```

Da wir dem LCD eigentlich 8 Bit Befehle und Daten senden müssen, müssen die zu übertragenden Bytes aufgeteilt werden. Die Funktion `write4BitLCDDData` übernimmt genau das – 8 Bit Daten werden in zwei 4 Bit „Nibbles“ aufgeteilt und übertragen.

## RP6 ROBOT SYSTEM - 3. RP6 CONTROL Library

---

```
void writeLCDCommand(uint8_t cmd)
```

Diese Funktion ruft write4BitLCDData auf, setzt allerdings die RS Leitung auf low, um einen Befehl an das LCD zu senden.

```
void clearLCD(void)
```

Sendet den Befehl zum Löschen des Display Inhalts an das LCD.

```
void clearPosLCD(uint8_t line, uint8_t pos, uint8_t length)
```

Löscht einen bestimmten Bereich des Displays. Die Parameter sind: Zeile, Startposition in der Zeile und Länge des zu löschenden Bereichs.

Beispiel:

```
clearPosLCD(0,10,5);    // löscht in der ersten Zeile des Displays  
                        // die letzten 5 Zeichen!
```

```
void setCursorPosLCD(uint8_t line, uint8_t pos)
```

Setzt den Textcursor an eine bestimmte Position auf dem Display. Der Parameter line kann 0 für die obere, oder 1 für die untere Zeile sein. Der Parameter pos darf für 2x16er LCDs im Bereich von 0 bis 15 liegen.

```
void writeCharLCD(uint8_t ch)
```

Sendet ein einzelnes Zeichen an das LCD – das funktioniert analog zur writeChar Funktion für die serielle Schnittstelle. Allerdings muss man hier zunächst sicherstellen dass der Cursor des Displays an der richtigen Position ist, denn sonst sieht man den Text nicht!

Beispiel:

```
setCursorPosLCD(1,5);  // positioniere Cursor in der zweiten Zeile, Zeichen 5.  
writeCharLCD('R');     // jetzt wird „RP6“ ausgegeben, und zwar  
writeCharLCD('P');     // beginnend an der Cursorposition!  
writeCharLCD('6');
```

```
void writeStringLCD(char *string)
```

Analog zur entsprechenden Funktion für die serielle Schnittstelle, sendet writeStringLCD eine nullterminierte Zeichenkette aus dem SRAM an das LCD. Also sollten Sie diese Funktion nur verwenden, wenn der Text auch wirklich im RAM liegt und nicht nur fest vordefiniert ist. Dazu ist das Makro:

```
writeStringLCD_P (STRING)
```

besser geeignet, da hier der Text direkt aus dem Flashspeicher gelesen wird, ohne den Umweg über den Arbeitsspeicher.

```
void writeStringLengthLCD(char *string, uint8_t length, uint8_t offset)
```

Mit dieser Funktion kann ein Text mit einer bestimmten Länge auf dem LCD ausgegeben werden. Die Parameter sind identisch zu denen der entsprechenden Funktion für die serielle Schnittstelle.

```
showScreenLCD (LINE1, LINE2)
```

Um die Textausgabe auf dem LCD etwas zu vereinfachen, kann man mit dieser Funkti-

on beide Zeilen des LCDs mit nur einem Aufruf beschreiben. Der Cursor wird automatisch richtig platziert und der Inhalt des Displays vorher gelöscht.

Beispiel:

```
showScreenLCD("LCD Zeile 1", "LCD Zeile 2");
```

```
void writeIntegerLCD(int16_t number, uint8_t base)
```

Die schon von der seriellen Schnittstelle bekannte Funktion um Zahlen in den Formaten BIN, OCT, DEC oder HEX auf dem LCD auszugeben.

```
void writeIntegerLengthLCD(int16_t number, uint8_t base, uint8_t length)
```

Auch writeIntegerLengthLCD ist bis auf den Namen identisch zur bereits bekannten Funktion für die seriellen Schnittstelle.

### 3.1.7. SPI Bus und SPI EEPROM

Am SPI (=Serial Peripheral Interface) Bus sind das EEPROM und das 8-Bit Schieberegister angeschlossen. Ein Sockel für ein weiteres AT25256 kompatibles EEPROM (z.B. ST M95256) im 8 poligen DIP Gehäuse kann optional aufgelötet werden. Man könnte auch weitere SPI ICs ansteuern und ein Schieberegister mit dem auf dem Board vorhandenen kaskadieren – das sollte man aber nur dann machen wenn man eine spezielle Anwendung dafür hat und nicht den I<sup>2</sup>C-Bus verwenden kann!

Da es für das EEPROM und das Schieberegister spezielle Funktionen gibt, muss man eigentlich gar nicht direkt aus dem eigenen Programm auf den SPI Bus zugreifen. Wenn das dennoch erforderlich ist, können folgende Funktionen verwendet werden.

```
void writeSPI(uint8_t data)
```

Überträgt ein Datenbyte über den SPI Bus.

```
writeWordSPI(uint16_t data)
```

Überträgt zwei Datenbytes die in einer 16 Bit Variablen übergeben werden über den SPI Bus, wobei das High Byte zuerst gesendet wird.

```
void writeBufferSPI(uint8_t *buffer, uint8_t length)
```

Überträgt bis zu 255 Bytes über den SPI Bus aus einem entsprechend großen Array. Die Anzahl der zu übertragenden Bytes im Array „buffer“ wird mit dem Parameter „length“ angegeben.

```
uint8_t readSPI(void)
```

Liest ein Datenbyte vom SPI Bus.

```
uint16_t readWordSPI(void)
```

Liest zwei Bytes vom SPI Bus und gibt diese als 16 Bit Variable zurück. Das zuerst gelesene Byte ist dabei das High Byte.

```
void readBufferSPI(uint8_t *buffer, uint8_t length)
```

Liest bis zu 255 Bytes vom SPI Bus in ein entsprechend großes Array.

### RP6 ROBOT SYSTEM - 3. RP6 CONTROL Library

Wie gesagt – normalerweise benötigen Sie die SPI Funktionen nicht. Diese werden aber von den im Folgenden beschriebenen Funktionen verwendet, um auf das am SPI Bus angeschlossene EEPROM zuzugreifen.

```
uint8_t SPI_EEPROM_readByte(uint16_t memAddr)
```

Liest ein einzelnes Byte an Adresse „memAddr“ aus dem EEPROM aus. Die Adresse darf bei unserem 32 KB großen EEPROM im Bereich von 0 bis 32767 liegen.

Beispiel:

```
// Wir lesen in der folgenden Zeile ein Byte von Adresse 13860 aus dem EEPROM:  
uint8_t data = SPI_EEPROM_readByte(13860);
```

```
void SPI_EEPROM_readBytes(uint16_t startAddr, uint8_t *buffer, uint8_t length)
```

Liest beginnend bei Adresse „startAdr“ bis zu 255 Bytes (length) in ein entsprechend großes Array (buffer).

```
void SPI_EEPROM_writeByte(uint16_t memAddr, uint8_t data)
```

Speichert ein Byte (data) an einer bestimmten Adresse (memAddr) im EEPROM.

```
void SPI_EEPROM_writeBytes(uint16_t startAddr, uint8_t *buffer, uint8_t length)
```

Speichert bis zu 64 Bytes (length) im Array „buffer“ beginnend bei „startAddr“ im EEPROM.



Bitte beachten Sie, dass nur 64 Bytes auf einmal geschrieben werden können. 64 Bytes ist die Seitengröße (Pagesize) des EEPROMs und mehr kann es nicht zwischenspeichern bevor es die Daten schreibt. Ausserdem müssen die Daten die hintereinander geschrieben werden sollen immer innerhalb einer Seite liegen, also z.B. zwischen 0 und 63, 64 und 127, 128 und 191 ...! Bei überschreiten der Seitengröße, überschreibt das EEPROM sonst wieder die Daten zu beginn der jeweiligen Seite. Sie können natürlich z.B. bei Adresse 50 mit dem Schreiben beginnen, aber wenn mehr als 14 Bytes geschrieben werden, beginnt der Adresszähler wieder bei 0 und überschreibt die dort im Zwischenspeicher bereits vorhandenen Daten.

Beim Lesen von Daten aus dem EEPROM müssen Sie die Seitengröße übrigens nicht beachten und können theoretisch auch das komplette EEPROM auf einmal auslesen.

Das EEPROM benötigt 5ms um die Daten zu schreiben. In dieser Zeit kann man nicht auf das EEPROM zugreifen. Um den aktuellen Status abzufragen kann die Funktion

```
uint8_t SPI_EEPROM_getStatus(void);
```

verwendet werden. Man kann dann z.B. mit

```
if(!(SPI_EEPROM_getStatus() & SPI_EEPROM_STAT_WIP)) {  
    // ...  
}
```

abfragen ob das EEPROM nicht mehr mit dem Schreiben von Daten beschäftigt ist. Das machen die Funktionen oben aber auch schon selbst, also benötigt man das nur, wenn man in der Zeit noch andere Dinge erledigen muss!

### 3.1.8. ADCs

Die ADC Kanäle lassen sich mit der schon von der RP6Lib bekannten Funktion:

```
uint16_t readADC(uint8_t channel)
```

auslesen. Eine automatische Variante die der Reihe nach die ADC Kanäle im Hintergrund ausliest gibt es für das RP6-M32 (noch) nicht.

Die Kanäle sind natürlich anders bezeichnet als in der RP6Lib – folgende Kanäle sind verfügbar:

```
ADC_7      --> ADC Kanal 7 - auf dem 10 poligen Wannenstecker „ADC“ verfügbar!
ADC_6      --> ADC Kanal 6 ...
ADC_5
ADC_4
ADC_3
ADC_2      --> ADC Kanal 2
ADC_KEYPAD --> Hier sind die Taster angeschlossen
ADC_MIC    --> Und hier das Mikrofon.
```

### 3.1.9. I/O Ports

Da auf dem RP6 CONTROL insgesamt 14 freie I/O Ports verfügbar sind, beschreiben wir hier noch kurz wie man allgemein auf I/O Ports eines AVR zugreifen kann.

Der ATMEGA32 hat 4 I/O Ports zu je 8 Bit. Jeder Port wird über 3 Register gesteuert. Ein Register für die „Richtung“ der I/O Pins (DDRx), also ob ein Pin als Eingang oder Ausgang geschaltet ist, ein Register zum Schreiben (PORTx) und ein Register zum Lesen (PINx).

Wenn man einen I/O Pin als Ausgang verwenden will, um z.B. eine LED zu schalten, muss man das entsprechende Bit im DDRx Register auf 1 setzen.

Beispiel:

```
DDRC |= IO_PC7; // PC7 ist nun Ausgang
DDRC = IO_PC7 | IO_PC6 | IO_PC5; // PC5, PC6, PC7 sind nun Ausgang,
                                // alle anderen Pins sind Eingänge!
```

Dann kann man über das PORTx Register den Ausgang auf high oder low pegel schalten.

Beispiel:

```
PORTC |= IO_PC7; // High
PORTC &= ~IO_PC7; // Low
```

Ist ein Bit im DDRx Register 0, so ist der zugehörige Pin als Eingang konfiguriert.

Beispiel:

```
DDRC &= ~IO_PC6; // PC6 ist nun Eingang
```

Dann kann man über das PINx Register den Zustand des Pins auslesen, also ob high oder low Pegel am Pin anliegt.

```
if(PINC & IO_PC6)
    writeString_P("PC6 is HIGH!\n");
else
    writeString_P("PC6 is LOW!\n");
```

### **RP6 ROBOT SYSTEM - 3. RP6 CONTROL Library**

---

Man kann weiterhin die im Mikrocontroller integrierten Pullup Widerstände aktivieren indem man die Bits im PORTx Register setzt. Das ist z.B. für Tastsensoren oder ähnliche Sensoren sinnvoll.

Folgende I/O Pins sind auf dem RP6 CONTROL M32 frei verfügbar (genaue Definition finden Sie in der Header Datei RP6Control.h):

IO\_PC7  
IO\_PC6  
IO\_PC5  
IO\_PC4  
IO\_PC3  
IO\_PC2  
  
IO\_PD6  
IO\_PD5

Die ADC Kanäle können ebenfalls als I/O Pins verwendet werden! Bitte beachten Sie die unterschiedlichen Schreibweisen verglichen mit den Bezeichnungen oben (ADC\_7 vs. ADC7)!

ADC7  
ADC6  
ADC5  
ADC4  
ADC3  
ADC2

**Wichtiger Hinweis:** Die einzelnen I/O Pins sind für einen maximalen Strom von 20mA ausgelegt. Insgesamt sollte ein 8 Bit Port nicht mit mehr als 100mA belastet werden! Wenn Sie also größere Verbraucher schalten wollen, müssen Sie dies über externe Transistoren tun!

Für genauere Informationen müssen wir hier auf das Datenblatt des MEGA32 verweisen, das auf der CD-ROM zu finden ist!

## 4. Beispielprogramme

Auf der CD finden Sie einige Beispielprogramme. Diese Beispielprogramme demonstrieren die grundlegenden Funktionen des RP6 CONTROL M32. Genau wie schon beim Roboter stellen sie keinesfalls die optimale Lösung dar und 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...

Sie können Ihre eigenen Programme selbstverständlich mit anderen Anwendern über das Internet austauschen. Die RP6ControlLib und alle Beispielprogramme stehen unter der Open Source Lizenz „GPL“ (General Public License) und daher sind Sie berechtigt, die Programme unter den Bedingungen der GPL zu modifizieren, zu veröffentlichen und anderen Anwendern zur Verfügung zu stellen.

Allgemein gibt es für den MEGA32 schon sehr viele Beispielprogramme im Internet, da die Controller aus der AVR Controllerfamilie bei Hobby Anwendern sehr beliebt sind. Allerdings muss man hier immer darauf achten, andere Beispielprogramme auch an die Hardware des RP6 CONTROL und die RP6ControlLib anzupassen – sonst wird es oft nicht funktionieren (die offensichtlichsten Probleme sind andere Pinbelegungen, Verwendung von bereits anderweitig verwendeten Hardwaremodulen wie Timern, andere Taktfrequenz, etc. pp.)!

Beispiel 1: „Hello World“-Programm mit LCD Textausgabe und LED Lauflicht

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_01\_LCD\

Datei: RP6Control\_LCD.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle und auf dem LCD-Display, sie sollten den Roboter also an den PC anschließen und sich die Ausgaben im Terminal der RP6Loader Software ansehen! Optional können Sie auch das LC-Display anschließen!

Der Roboter bewegt sich in diesem Beispielprogramm nicht – sofern Sie nur das I<sup>2</sup>C-Bus Slave Programm in den Controller auf dem Mainboard geladen haben! Sie können ihn also z.B. auf einen Tisch neben dem Computer stellen.

Dieses Beispielprogramm gibt einen kurzen „Hello World“ Text über die serielle Schnittstelle aus, und anschließend wird ein Lauflicht ausgeführt. Zusätzlich wird auf dem LCD zunächst ein statischer Text ausgegeben und später ein sich bewegendes „Hello World“ Text – die beiden Wörter „HELLO“ und „WORLD“ bewegen sich langsam hin und her. Nach etwa 16 Sekunden wird eine kurze Pause gemacht und dies mit zwei kurzen Piepsen signalisiert. Nach 8 Sekunden wird weitergemacht – ebenfalls signalisiert durch zwei kurze Piepsen.

## RP6 ROBOT SYSTEM - 4. Beispielprogramme

### Beispiel 2: Taster und Sound

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_02\_Buttons\

Datei: RP6Control\_Buttons.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle und auf dem LC-Display!

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Dieses Beispielprogramm demonstriert die Verwendung der 5 Taster auf dem RP6 CONTROL. Bei jedem Tastendruck wird die Tastennummer im Display angezeigt und eine Tonfolge mit dem Piezo ausgegeben

(Achtung: Auf T4 zu drücken kann ganz schön nervig werden ;-) ).

### Beispiel 3: Mikrofonsensor

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_03\_Microphone\

Datei: RP6Control\_Microphone.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle und auf dem LC-Display!

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Der Mikrofonsensor kann zum Detektieren lauter Geräusche verwendet werden. Dieses Programm stellt die gemessene Lautstärke als Balken dar. Sowohl auf dem LCD als auch mit den LEDs. Dabei wird auch der gemessene Wert angezeigt. Tippen Sie mal mit dem Finger auf das Mikrofon, um zu testen ob es korrekt funktioniert. Klatschen Sie auch mal in die Hände oder ähnliches und schauen Sie sich die Reaktion auf dem Display und den LEDs an.

### Beispiel 4: Externes EEPROM

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_04\_EEPROM\

Datei: RP6Control\_04\_EEPROM.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle und auf dem LC-Display!

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Wie man grundlegend auf das externe EEPROM schreibend und lesend zugreifen kann, wird in diesem Beispielprogramm demonstriert. Zunächst werden die ersten zwei „Pages“ (=Seiten) zu je 64 Byte ausgelesen und ausgegeben. Als Demonstration, dass das EEPROM seinen Inhalt auch nach dem Ausschalten des Roboters behält, schalten Sie den Roboter nach Ausführung des Programms bitte einfach mal aus und wieder an! Die zuletzt geschriebenen Daten bleiben dabei im EEPROM erhalten und werden zu Beginn wieder ausgegeben.

## RP6 ROBOT SYSTEM - 4. Beispielprogramme

---

Dann wird im Programm die erste Page mit 64 Bytes beschrieben und danach wieder die ersten 128 Bytes gelesen, um zu verifizieren, dass die Bytes auch nur in der ersten Page geschrieben wurden und der Rest des EEPROMs nicht verändert wurde (dann ist der Inhalt jeder Speicherzelle 255).

Auch einzelne Bytes können geschrieben und gelesen werden, was im Anschluss demonstriert wird. Es wird die Speicherzelle mit Adresse 4 mit 128 beschrieben und danach wieder die ersten beiden Pages ausgelesen – diesmal allerdings Byte für Byte – was natürlich langsamer ist als eine Page komplett zu lesen.

Wenn alles erledigt ist, wird noch ein kleines Lauflicht ausgeführt.

### Beispiel 5: Analog Digital Wandler und I/O Ports

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_05\_IO\_ADC\

Datei: RP6Control\_05\_IO\_ADC.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle und auf dem LC-Display!

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Es funktioniert zwar eigentlich genau wie auf dem Roboter – aber weil es vor allem bei diesem Erweiterungsmodul sehr häufig benötigt wird, demonstriert dieses Beispielprogramm noch kurz wie die freien ADCs und I/Os ausgewertet werden können.

Die ADCs und I/Os haben im Programm folgende Bezeichnungen:

```
ADC7   (1 << PINA7) // ADC Kanal 7 - auch als normaler I/O Pin verwendbar
ADC6   (1 << PINA6) // Kanal 6 ...
ADC5   (1 << PINA5)
ADC4   (1 << PINA4)
ADC3   (1 << PINA3)
ADC2   (1 << PINA2) // Kanal 2. Die Kanäle 0 und 1 sind mit Tastatur und Mikrophon belegt.
```

```
IO_PC7 (1 << PINC7) // I/O Pin 7 von PORTC
IO_PC6 (1 << PINC6) // Pin 6 ...
IO_PC5 (1 << PINC5)
IO_PC4 (1 << PINC4)
IO_PC3 (1 << PINC3)
IO_PC2 (1 << PINC2) // I/O Pin 2 von PORTC
IO_PD6 (1 << PIND6) // I/O Pin 6 von PORTD
IO_PD5 (1 << PIND5) // I/O Pin 5 von PORTD
```

(aus der Datei RP6Control.h)

Das Programm tut nichts wirklich sinnvolles – also brauchen Sie es nicht unbedingt auszuprobieren. An den I/O Pins ist nichts angeschlossen, daher macht es erst Sinn etwas damit anzufangen, wenn man eigene Hardware mit den I/Os steuern will.

## RP6 ROBOT SYSTEM - 4. Beispielprogramme

### Beispiel 6: I<sup>2</sup>C Bus Interface – Master Modus

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_06\_I2CMaster\

Datei: RP6Control\_06\_I2CMaster.c

Dieses Programm demonstriert wie man den Master Modus des I<sup>2</sup>C Busses verwenden kann. Der Controller auf dem Roboter Mainboard muss das I2C Slave Beispielprogramm geladen haben!

Dieses Programm zeigt, wie man den Controller auf dem Mainboard im Slave Modus steuern kann. Das klappt natürlich nur dann, wenn auch das I<sup>2</sup>C Slave Beispielprogramm aus den RP6Base Beispielen in den Controller auf dem Mainboard geladen ist.

Der Zugriff auf den I<sup>2</sup>C Bus funktioniert fast genau wie auf dem Controller auf dem Mainboard auch – es sind dieselben Funktionen.

In diesem Beispielprogramm kann man verschiedene I<sup>2</sup>C Bus Kommandos an den mit dem Slave Beispielprogramm programmierten Controller auf dem Mainboard schicken, indem man auf einen der 5 Taster drückt. Taster T1 erhöht einen Zähler um eins und sendet dann den setLEDs Befehl an den Slave mit diesem Zählerwert. So wird mit den 6 Status LEDs auf dem Mainboard ein Binärzähler angezeigt.

Mit einem Druck auf Taster T2 werden alle Register gelesen und über die serielle Schnittstelle ausgegeben. Mit T3 hingegen, werden nur die Werte der Lichtsensoren ausgelesen und auch auf dem LC-Display angezeigt.

Mit T4 und T5 wird der „Rotate“ Befehl gesendet und der Roboter dreht sich ein kleines Stückchen nach links oder rechts (man kann auch mehrmals auf den Taster drücken, dann dreht er sich weiter... ).

Das Programm kann – wie alle anderen auch – natürlich beliebig verändert werden und eignet sich sehr gut zum Testen neuer I<sup>2</sup>C Geräte oder Funktionen die man dem Slave Beispielprogramm hinzufügt.

### Beispiel 7: I<sup>2</sup>C Bus Interface – Master Modus – auf Interrupts reagieren

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_07\_I2CMaster\

Datei: RP6Control\_07\_I2CMaster.c

Dieses Programm demonstriert wie man den Master Modus des I<sup>2</sup>C Busses verwenden kann. Der Controller auf dem Roboter Mainboard muss das I2C Slave Beispielprogramm geladen haben!

Vielleicht sind Ihnen schon die Interrupt Signale auf den XBUS Anschlüssen des RP6 aufgefallen? Diese kann man nutzen um ohne ständiges Abfragen der Slaves auf Sensoränderungen zu reagieren. Jeder Buszugriff kostet schließlich Zeit.

Ein gutes Beispiel dafür ist das ACS auf dem Roboter. Der Sensorzustand ändert sich hier nur *relativ* selten und es wäre nicht besonders effektiv, ständig über den Bus abzufragen ob sich etwas geändert hat. Sobald sich der Status des ACS ändert, wird vom Slave Beispielprogramm das INT1 Signal auf High Pegel gesetzt. Da INT1 an den Interrupt Eingang 0 vom MEGA32 auf dem RP6 CONTROL M32 gelegt wurde, kann der Controller direkt auf dieses Ereignis reagieren und den Status des Controllers auf dem Mainboard abfragen.

Wir nutzen in den Beispielprogrammen allerdings KEINE Interrupt Routinen um auf das Ereignis zu reagieren, sondern fragen den Zustand des Pins bei jedem Durchlauf der Hauptschleife ab. Da die I2C Bus Transfers selbst auch Interruptgesteuert ablaufen, könnte innerhalb der Interrupt Routine keine neue Übertragung stattfinden. Es

muss in der Hauptschleife immer die `task_I2CTWI()` Funktion aufgerufen werden, die den Ablauf der I2C Transfers regelt. Daher hätte es keinen großen Vorteil eine Interrupt Routine zu verwenden. Um genau zu sein, würde es sogar Probleme bereiten, denn so könnten bereits laufende I2C Bus Übertragungen unterbrochen werden. Also wird nur die Funktion `task_checkINT0()` verwendet um ständig das Interrupt Signal auszuwerten und ggf. eine Abfrage zu starten. Sobald das Statusregister 0 des Slaves gelesen wird, wird das Interrupt Signal zurückgesetzt. Über die ersten drei Register des Slaves kann man herausfinden, was den Interrupt ausgelöst hat.

Genau das macht dieses Beispielprogramm auch. Resultat ist, dass der aktuelle Zustand des ACS über die 4 LEDs, das LCD, die serielle Schnittstelle und den Beeper auf dem RP6 CONTROL angezeigt und hörbar gemacht wird.

Zu Beginn des Programms wird noch die Sendeleistung des ACS über den I<sup>2</sup>C Bus eingestellt. Neben dem ACS reagiert das Programm auch auf die Bumper und auf eventuell gesendete RC5 Übertragungen von einer Fernbedienung oder anderen Robotern.

Das kann man so ähnlich natürlich auch mit allen anderen Sensoren des Roboters handhaben. Auch mit evtl. zukünftig erhältlichen Erweiterungsmodulen mit weiteren Sensoren.

Ein weiteres Detail des Programms ist die „Heartbeat“ Anzeige. Also „Herzschlag“ Anzeige. Die `task_LCDHeartbeat()` Funktion sorgt dafür, dass auf dem LCD ständig das Zeichen '\*' mit einer Frequenz von 1Hz blinkt. Das ist sehr hilfreich um festzustellen ob sich das Programm komplett aufgehängt hat, oder ob nur ein bestimmter Teil der Software fehlerhaft ist. Wenn Sie eigene Programme schreiben und sich das Programm dann anscheinend komplett aufhängt, kann das sehr hilfreich sein um die Fehlerquelle einzugrenzen. Das sich das Programm aufhängt, kann während der Entwicklung durchaus mal passieren.

Deshalb ist im I<sup>2</sup>C Slave Beispielprogramm für den Controller auf dem Mainboard auch eine „Software Watchdog“ Funktion eingebaut. Reagiert der Master innerhalb einer festgelegten Zeit nicht auf ein Interrupt Ereignis (indem Register 0 gelesen wird), werden alle Systeme der Roboter Basiseinheit ausgeschaltet und das Programm gestoppt. Vor allem werden die Motoren ausgeschaltet! Denn sollte sich der Master Controller aufhängen, aber vorher noch den Befehl mit 10cm/s vorwärts zu fahren senden, fährt der Roboter ungebremst vor das nächste Hindernis und stoppt auch nicht bei einer Kollision ...

Der Software Watchdog Timer ist zunächst deaktiviert. Man muss vorher noch einen Befehl über den I<sup>2</sup>C Bus senden um den Watchdog zu aktivieren. Es ist auch möglich den Watchdog so zu konfigurieren, dass dieser alle 500ms ein Interrupt Ereignis auslöst um zu überprüfen ob der Mastercontroller noch darauf reagiert. Das werden wir im nächsten Beispiel verwenden.

## RP6 ROBOT SYSTEM - 4. Beispielprogramme

### Beispiel 8: I<sup>2</sup>C Bus Interface – kleine RP6 Library

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_08\_I2CMaster\

Datei: RP6Control\_08\_I2CMaster.c

Dieses Programm demonstriert wie man den Master Modus des I<sup>2</sup>C Busses verwenden kann. Der Controller auf dem Roboter Mainboard muss das I2C Slave Beispielprogramm geladen haben!

Da Programme schnell etwas unübersichtlich werden können, wenn man soviel in eine Datei reinpackt, wird jetzt das Beispiel 7 in zwei C Dateien aufgeteilt und noch etwas ergänzt. Dabei wurde es gleich so angelegt, das man eine kleine Bibliothek zur Steuerung des Roboters über den I<sup>2</sup>C Bus hat, die sich fast wie die normale RP6Lib für den Controller auf dem Mainboard verwenden lässt. Viele Funktionen und Variablen sind identisch zur RP6Lib benannt. Das vereinfacht es, Teile von Programmen für die RP6Lib auch mit der RP6ControlLib weiterzuverwenden. Auch die bekannten Event Handler für ACS, IRCOMM und Bumper sind wieder verfügbar. Neu sind Event Handler für niedrigen Akkuladestatus und die Watchdog Requests. Im nächsten Beispiel werden noch die Funktionen für die Bewegungssteuerung hinzugefügt.

Das Programm ist ansonsten ähnlich zu Beispiel 7. Neu hinzugekommen sind nur wie schon erwähnt der weitere Watchdog Timer dessen Requests auch auf dem LC-Display dargestellt werden und es werden alle Sensorregister ausgelesen und über die serielle Schnittstelle ausgegeben. Wie gehabt werden auch weiterhin ACS, Bumper und RC5 Events dargestellt.

### Beispiel 9: I<sup>2</sup>C Bus Interface – Bewegungsfunktionen

Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_09\_Move\

Datei: RP6Control\_09\_Move.c

Dieses Programm demonstriert wie man den Master Modus des I<sup>2</sup>C Busses verwenden kann. Der Controller auf dem Roboter Mainboard muss das I2C Slave Beispielprogramm geladen haben!

**ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!**

Jetzt fügen wir der neuen Library ein paar der bereits von der RP6Lib bekannten Bewegungsfunktionen hinzu: move, rotate, moveAtSpeed, changeDirection und stop. Die Funktionen können identisch zu denen in der RP6Lib benutzt werden. In diesem Beispiel haben wir alles aus den vorherigen Beispielen bis auf die Anzeige für den Watchdog wieder entfernt, damit es übersichtlicher ist und ausserdem der blockierende Modus der Bewegungsfunktionen verwendet wird (dann würden ein paar Sachen wie die Heartbeat Anzeige ohnehin nicht funktionieren). Der Roboter bewegt sich in diesem Beispiel hin und her und dreht sich dabei um etwa 180° - genau wie in Beispiel 7 zur RP6Lib („RP6Base\_Move\_02.c“).

## RP6 ROBOT SYSTEM - 4. Beispielprogramme

---

Beispiel 10: I<sup>2</sup>C Bus Interface – Verhaltensbasierter Roboter  
Verzeichnis: <RP6Examples>\RP6ControlExamples\Example\_10\_Move2\  
Datei: RP6Control\_10\_Move2.c

Dieses Programm demonstriert wie man den Master Modus des I<sup>2</sup>C Busses verwenden kann. Der Controller auf dem Roboter Mainboard muss das I2C Slave Beispielprogramm geladen haben!

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Mit der neuen Library kann man jetzt schon fast 1:1 die Beispielprogramme zum Verhaltensbasierten Roboter übernehmen. Genau das wurde hier mit Beispielprogramm RP6Base\_05\_Move\_05 getan. Es waren nur kleinere Änderungen notwendig – u.a. müssen die LEDs auf dem Mainboard über die Funktion setRP6LEDs gesteuert werden, da setLEDs schon für die LEDs auf dem RP6-M32 reserviert ist...

Ansonsten ist das Programm nahezu identisch zum bereits bekannten Programm – der Roboter fährt herum und weicht dabei Hindernissen aus. Nur das er diesmal vom RP6-M32 gesteuert wird.

Neu hinzugekommen ist die Darstellung des aktuell aktiven Verhaltens auf dem LC-Display und mit den LEDs. So kann man direkt sehen welches Verhalten gerade aktiv ist. Dazu gibt es eine kleine Hilfsfunktion die sicherstellt, dass der Text immer nur einmal an das LC-Display gesendet wird. Sonst würde der Text auf dem Display flimmern. Während das Verhalten „Cruise“ aktiv ist, wird mit den 4 Roten Status LEDs ein kleines Lauflicht ausgeführt.

Es wird auch der Akkuzustand überwacht. Wenn der Ladezustand sehr niedrig ist, wird der Roboter angehalten. Das dauert bei frisch aufgeladenen Akkus allerdings eine Weile...

Weiterhin wartet das Programm zu beginn auf drei laute Geräusche (dann steht WAIT in der zweiten Displayzeile, daneben der Counter für die lauten Geräusche) – z.B. dreimal in die Hände klatschen. Alternativ kann man auch einen beliebigen Taster auf dem RP6-M32 drücken. Auch das wurde mit einem weiteren Verhalten realisiert.

*Damit sind wir auch 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-M32 steuern können oder etwas ganz anderes damit anstellen.*

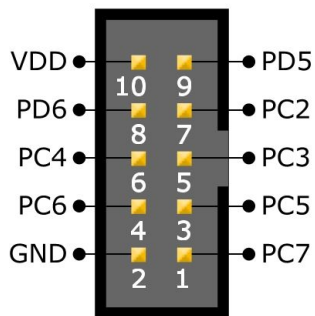
# ANHANG

## A – Anschlussbelegungen

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

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

### I/O Ports:

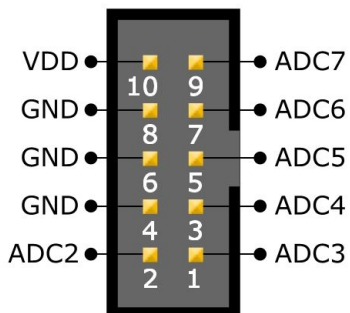


Am Anschluss I/O sind alle freien I/O Ports und die Betriebsspannung verfügbar.

PC2, PC3, PC4, PC5, PC6, PC7, PD5 und PD6.

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

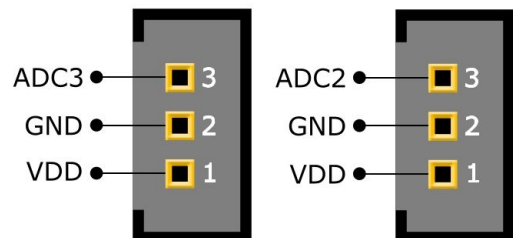
### ADC Kanäle:



Die 6 freien ADC Kanäle (die natürlich auch als I/O Pins verwendbar sind) sind am 10 poligen ADC Stecker verfügbar. Ebenfalls zusammen mit der Betriebsspannung.

Zwei der ADCs sind auch auf zwei unbestückte Anschlüsse wie auf dem Mainboard

gelegt. Daran kann man eigene Stecker im 2.54mm Raster anlöten.

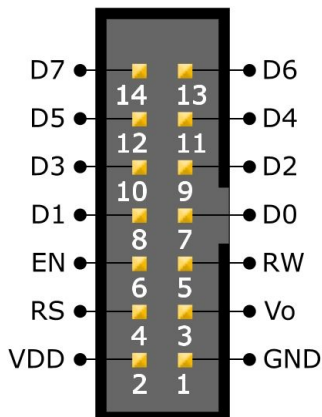


Aber vorsichtig sein und nichts „kaputtlöten“! Das sollte man nur machen wenn man schon etwas Löterfahrung hat.

Sie können an die Anschlüsse zwei beliebige analoge oder auch digitale Sensoren anschließen (Ausgangsspannung der Sensoren kann im Bereich von 0 bis 5V liegen) und mit 5V Betriebsspannung versorgen. Eventuell sollte man noch den großen Elko bestücken – 220 bis 470µF (mehr nicht!) eignet sich gut für die meisten Anwendungen.

Das ist aber nur dann nötig, wenn Sensoren mit hohem Spitzenstrom eingesetzt werden – wie z.B. die beliebten Sharp IR Abstandssensoren. Abblockkondensatoren (100nF) auf dem Mainboard, eignen sich nur für sehr kurze Zuleitungen – bei längeren Leitungen sollten diese direkt an die Sensoren angelötet werden (was aber auch schon bei kurzen Leitungen sehr zu empfehlen ist!).

### LCD Anschluss:



Wenn Sie nicht das Standard LCD einsetzen möchten, können Sie anhand der nebenstehenden Anschlussbelegung ein eigenes 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 lesen brauchen.

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

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