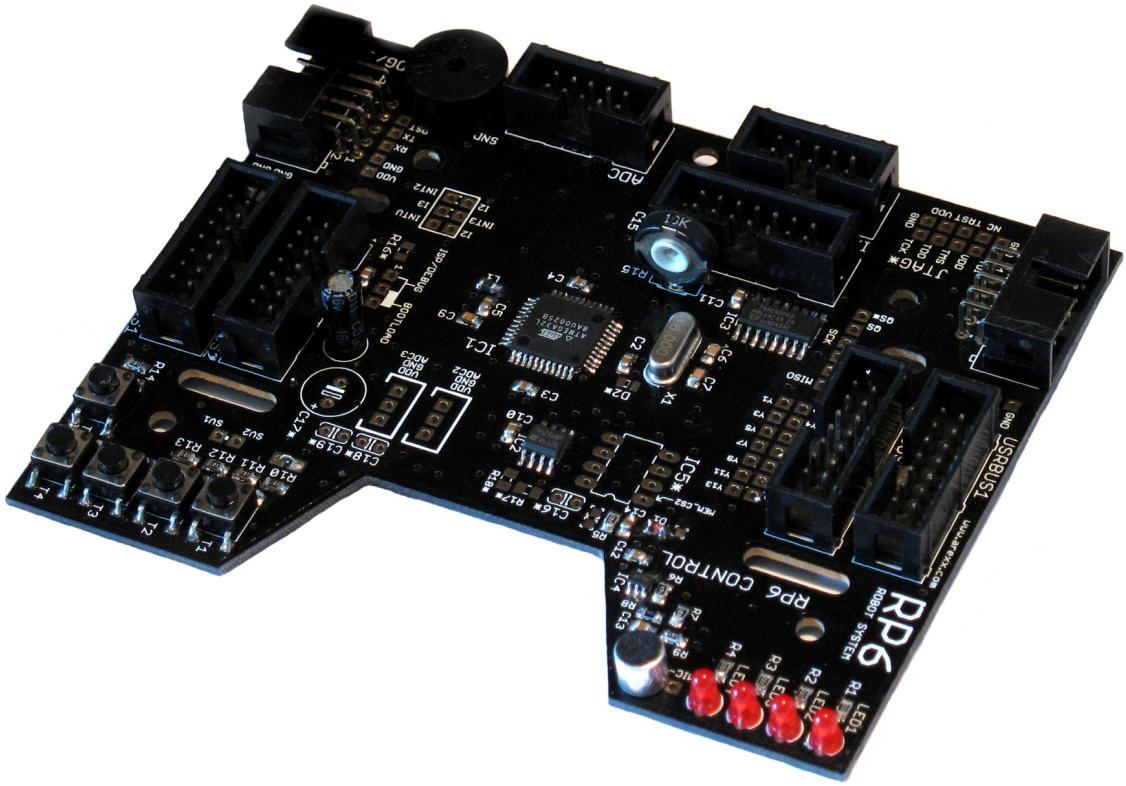
RP6 ROBOT SYSTEM

RP6 CONTROL M32

Module d'Extension



AREXX Engineering

RP6-M32

www.arexx.com

RP6 CONTROL M32

Manuel d'Utilisation

- Français (French) -

Version RP6-M32-FR-20071031

INFORMATIONS IMPORTANTES!
A lire absolument!

Avant la mise en service de ce module d'extension du RP6, lisez attentivement ce manuel ET le manuel du RP6! Il vous explique la bonne utilisation et attire votre attention sur des dangers éventuels! Par ailleurs, il contient des informations importantes qui ne sont pas forcément connues de tous les utilisateurs. Le manuel du module RP6 CONTROL M32 n'est qu'un complément!

Le non-respect des consignes contenues dans ce manuel et celui du robot RP6 invalide la garantie! Par ailleurs, AREXX Engineering décline toute responsabilité pour des dommages quels qu'ils soient, qui résultent du non-respect de ce manuel.

Lisez surtout le chapitre « Consignes de Sécurité » dans le manuel du système robot RP6!

Impressum

©2007 AREXX Engineering

Nervistraat 16
8013 RS Zwolle
The Netherlands

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

Restrictions de garantie et de responsabilité

La garantie d'AREXX Engineering se limite au remplacement ou à la réparation du robot pendant la période légale de garantie si la défaillance provient d'un défaut de fabrication tel qu'un défaut mécanique ou une implantation erronée ou manquante de composants électroniques à l'exception de tous les composants implantés par des connecteurs. AREXX Engineering décline toute responsabilité pour des dommages résultant directement ou indirectement de l'utilisation du robot à l'exception des droits basés sur les obligations légales de responsabilité du fabricant.
Si vous modifiez le robot d'une manière irréversible (p.ex. soudage d'autres composants, perçage de trous, etc) ou le robot est endommagé suite au non-respect de ce manuel, tout droit à garantie s'éteint!

Nous ne pouvons pas garantir que le logiciel fourni répondra aux attentes individuelles ou pourra travailler sans aucune interruption, ni défaillance.
Par ailleurs, le logiciel est librement modifiable et chargé dans l'appareil par l'utilisateur. Par conséquent, l'utilisateur assume la totalité du risque concernant la qualité et la performance de l'appareil y compris du logiciel.
AREXX Engineering garantit la fonctionnalité des exemples d'application fournis à condition de respecter les conditions spécifiées dans les caractéristiques techniques. Si, au-delà, le robot ou le logiciel PC s'avèrent défaillants ou insuffisants, tous les frais de service, de réparation ou de correction sont à la charge du client.
Respectez également les accords de licence indiqués sur le CD ROM!

Symboles

Les symboles suivants sont utilisés dans ce manuel:

Le point d'exclamation dans le triangle attire l'attention sur des consignes très importantes qu'il faut respecter scrupuleusement. Une erreur pourrait entraîner la destruction du robot ou de ses accessoires et même mettre en danger votre santé ou celle d'autrui!

Le « i » dans un cercle attire l'attention sur des chapitres qui contiennent des astuces et conseils utiles ou des informations de fond. Ce n'est pas toujours essentiel de tout comprendre mais généralement très utile.

Table des Matières

1. Le Module d'Extension RP6 CONTROL M32

Le module d'extension RP6 CONTROL M32 (ou en abrégé „RP6-M32“) vous permet d'ajouter au robot un deuxième microcontrôleur Atmel ATMEGA32 qui est cependant deux fois plus rapide que le contrôleur sur la carte-mère. En outre, le RP6-M32 offre plus de temps de calcul puisqu'il n'est pas occupé par le réglage du moteur, l'ACS, l'IRCOMM, etc.

L'EEPROM SPI 32KB externe est une mémoire ROM externe inscriptible de nombreuses fois (1 millions de cycles) que vous pouvez utiliser p.ex. pour enregistrer des données (data logger) ou comme zone programme pour des interprètes de bytecode (tel que le NanoVM pour Java). Il est même possible de braser un commutateur DIP à 8 broches en option sur le module et ajouter un deuxième EEPROM dans le boîtier DIP 8.

Les touches d'entrée, les LED, le buzzer Piézo et l'écran LCD en option offrent d'autres possibilités intéressantes. Elles permettent de commander le robot directement en écrivant p.ex. un petit programme afin de démarrer différents programmes par les touches et bien sûr également afficher des valeurs de mesure et des messages d'état. Le buzzer peut générer plusieurs sons et jouer p.ex. une mélodie de bienvenue lorsque le programme est démarré ou avertir en cas de sous-tension des accus.

Vous pourrez piloter votre propres circuits avec les 14 ports I/O libres sur les modules d'extension à grille perforée qui sont montés sur deux fiches à 10 broches. Sur les 14 I/O, vous pouvez utiliser 6 comme canaux CAN.

L'équipement du module est complété par un détecteur de microphone qui existait déjà sur l'ancien CCRP5. Il permet p.ex. de démarrer le RP6 en frappant dans les mains ou par un autre bruit. Le circuit est conçu comme « peak detector » ce qui signifie qu'il ne détecte que les sons les plus forts. Ainsi, le volume des bruits environnants est grossièrement mesuré et la réaction sera en conséquence (cela ne fonctionne bien que si les moteurs ne tournent pas parce que le microphone capte surtout les bruits faits par le robot lui-même par réverbération de son corps...).

Avant d'attaquer le RP6-M32, vous devez absolument vous familiariser avec le robot lui-même et essayer tous les exemples de programme du robot SANS le module d'extension RP6-M32. Ce manuel n'est qu'un petit complément au grand manuel RP6. Lisez-le absolument avant de commencer avec le RP6-M32.

Information importante pour débutants: Les programmes écrits pour le RP6-M32 NE tournent PAS correctement sur le microcontrôleur de l'unité de base et inversement (affectation des broches et cycle horloge tout à fait différents).

ATTENTION: Si vous chargez un programme dans le mauvais contrôleur, vous courrez le risque d'endommager le contrôleur ou les circuits! C'est le cas si une broche I/O est normalement utilisée comme entrée mais est commutée en sortie dans le programme destiné à l'autre contrôleur et est surchargée en raison du circuit connecté.

Normalement il ne se passe rien de très grave en cas d'erreur mais nous ne pouvons rien garantir. Le RP6Loader ne peut pas distinguer quel programme est destiné à quel

contrôleur puisque les fichiers hex sont tous construits de la même manière. Il n'empêchera donc pas que vous chargiez le programme dans le mauvais contrôleur! Profitez de la fonction du RP6Loader de créer différentes catégories. Pour chaque module d'extension sa propre catégorie...

1.1. Support technique

Si vous avez des questions ou rencontrez des problèmes, vous pouvez joindre notre assistance technique par Internet (Avant de nous contacter, **lisez entièrement le mode d'emploi!** Par expérience nous savons que la plupart des questions y trouveront une réponse!
- par notre forum: `http://www.arexx.com/forum/`

- par courrier électronique: `info@arexx.nl`

Notre adresse postale figure dans l'impressum de ce manuel. Des informations de contact plus actualisées, des mises à jour de logiciel et autres informations figurent sur notre page d'accueil:

http://www.arexx.com/

et sur la page d'accueil du robot:

http://www.arexx.com/rp6

1.2. Contenu du carton

Vous devez trouver les pièces suivantes dans le carton du RP6 CONTROL M32:

- Module RP6-M32 fini
- 4 entretoises M3 25mm
- 4 vis M3
- 4 écrous M3
- 2 câbles en nappe à 14 contacts

Le logiciel et le mode d'emploi en PDF se trouvent sur le CD-ROM du RP6. Des versions mises à jour du logiciel et de ce manuel sont disponibles sur notre page d'accueil.

1.3. Propriétés et caractéristiques techniques

Ce paragraphe donne un aperçu des capacités du RP6 CONTROL M32 et introduit en même temps quelques notions et désignations de composants du module.

Propriétés, Composants et Caractéristiques techniques du RP6 CONTROL M32:

- **Microcontrôleur performant Atmel ATMEGA32 8-Bit**
  - ◊ Vitesse 16 MIPS (=16 Millions d'Instructions Par Seconde) à un cycle horloge de 16MHz, donc le double par rapport au contrôleur sur la carte-mère!
  - ◊ Mémoire: Flash ROM 32KB, SRAM 2KB, EEPROM 1KB
  - ◊ Librement programmable en C (avec WinAVR / avr-gcc)!
  - ◊ ... et beaucoup d'autres choses (voir fiche technique)!
- EEPROM SPI 32KB Externe
  - ◊ Interface SPE très rapide (Cycle horloge 8MHz)
  - ◊ Chaque cellule de mémoire est ré-inscriptible au moins 1.000.000 de fois
  - ◊ ... plus d'informations techniques sur la fiche technique du CD-ROM (AT25256A)!
  - ◊ Idéal pour enregistrer des données (data logger) ou comme mémoire pour l'interprète de codes byte (p.ex. un Java NM comme le NanoVM: http://www.harbaum.org/till/nanovm/ . Toutefois, il faudra encore l'adapter afin qu'il utilise l'EEPROM externe...)
- **Fiches d'Extension du bus I²C**
  - ◊ Peut commander des esclaves de bus I²C.
  - ◊ Le MEGA32 sur le module peut servir de maître ou d'esclave. Il est cependant plus judicieux de l'utiliser comme maître afin qu'il commande entièrement le robot (le contrôleur sur la carte-mère se charge toutefois automatiquement de la régulation de la vitesse des moteurs, de l'ACS, de l'IRCOMM, de la surveillance des accumulateurs, etc. et décharge ainsi le contrôleur sur le module d'extension).
- **Détecteur à Microphone**
  - ◊ Pour détecter des bruits tels que frapper dans les mains, etc..
- **Beeper Piézo**
  - ◊ Pour générer des sons et mélodies simples
  - ◊ Avertisseur sonore afin d'annoncer une erreur ou un changement d'état
- **4 LED d'état**
- **5 touches de saisie**

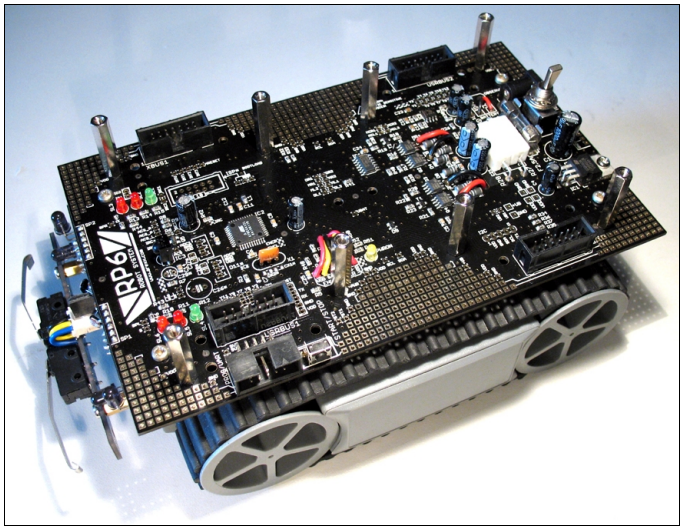
- **Port de l'écran à Cristaux Liquides (LCD)**
  - ◊ Permet la connexion d'un LCD externe de 16x2 caractères. Vous pouvez également connecter d'autres LCD tels qu'un 16x4 mais dans ce cas, il faudra le fixer à l'aide de deux entretoises et il risque de dépasser d'un côté... Il est recommandé de prendre les dimensions exactes avant l'achat et de commander le matériel de montage adéquat en même temps. Par ailleurs, pour pouvoir fonctionner correctement, il peut être nécessaire de procéder à quelques adaptations dans la bibliothèque si vous utilisez un format d'écran autre que le 16x2 (principalement l'initialisation de l'afficheur et éventuellement le positionnement du curseur). Tous les programmes de démo sont conçus pour un afficheur à 16x2 caractères mais c'est facile à changer.
  - ◊ Le LCD peut p.ex. afficher des messages de texte/menus, états de programme ou valeurs des détecteurs.
- **14 ports I/O libres pour commander vos propres circuits et détecteurs**
  - ◊ Dont **6** sont utilisables comme **canaux de convertisseurs analogiques/numériques (CAN)**
- **Vous pouvez utiliser jusqu'à 3 interruptions externes sur le connecteur XBUS.**
- **Connexion pour l'interface PC USB** pour le chargement du programme
  - ◊ Le chargement du programme se déroule comme sur le robot, rapidement et simplement par l'interface USB et l'agréable logiciel RP6Loader.

Tout comme pour le robot, le module est livré avec quelques exemples de programme en C ainsi qu'une bibliothèque de fonctions qui facilite largement la programmation.

D'autres programmes et mises à jour pour le robot et ses modules d'extension sont en cours et figureront sur le site Internet du robot dès qu'ils seront disponibles. Vous avez également la possibilité d'échanger vos propres programmes par Internet avec d'autres utilisateurs du RP6! La RP6ControlLibrary et les exemples de programmes sont sous licence Open Source GPL.

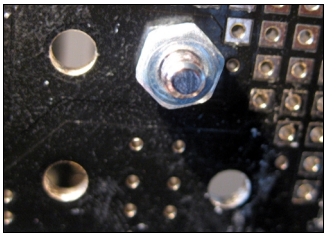
2. Montage du Module d'Extension

La façon dont vous allez fixer le module sur le robot dépend bien sûr aussi du nombre de modules que vous avez peut-être déjà installés sur le robot.



Commencez par retirer les quatre vis de la carte-mère. Vous pouvez éventuellement également débrancher la petite fiche de la platine des pare-chocs afin de pouvoir soulever la carte-mère tout doucement si vous n'arrivez pas à passer les doigts sous la carte-mère pour visser les entretoises sur les écrous M3.

Attention: Lors de la remise en place du câble de la platine des pare-chocs, vous devez absolument appuyer avec un doigt de l'autre côté de la platine afin qu'elle ne soit pas trop fortement poussée vers l'arrière. Il est également possible de retirer les deux vis de la platine des pare-chocs et de laisser le câble branché ...

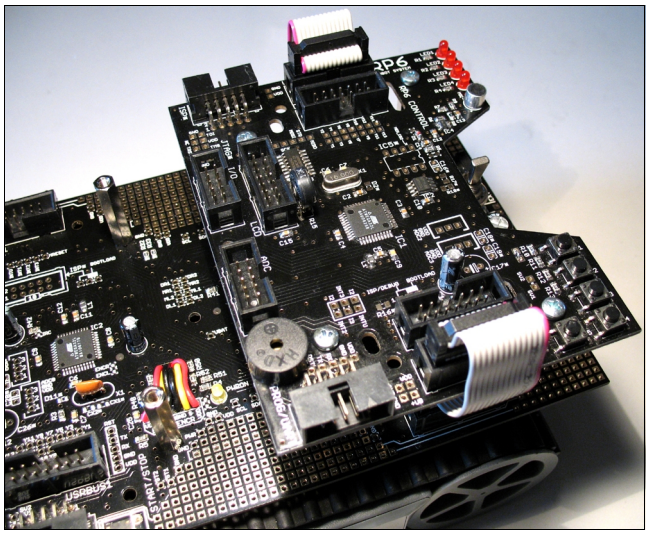


Ensuite vous pouvez visser l'une après l'autre les quatre entretoises M3 de 25mm sur les écrous M3 dans les trous de fixation de la carte-mère, comme indiqué sur la photo.

Sur la photo ci-dessus, les 8 entretoises sont montées, donc aussi celles du module d'extension à grille perforée!

Ensuite vous placez le module d'extension sur les entretoises et vous le vissez avec les quatre vis M3.

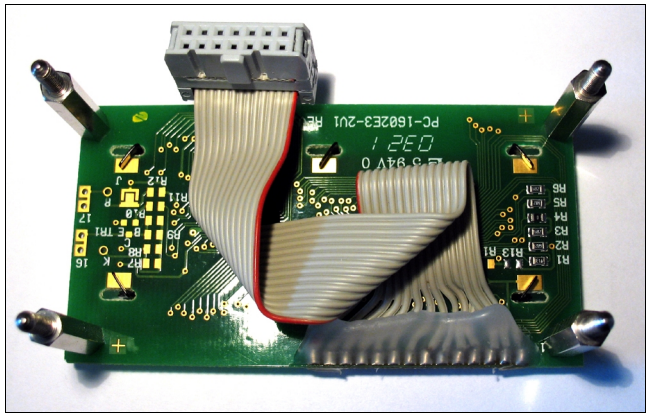
Maintenant vous n'avez plus qu'à brancher les deux câbles en nappe et voilà!



Nous vous conseillons de monter le RP6 CONTROL M32 sur la pile d'extension arrière comme premier module pour que les capteurs et éventuellement aussi l'afficheur restent accessibles. Ainsi, les deux connexions de programmation restent accessibles sur le même côté du robot. Sur l'avant, vous pouvez installer le module d'expérimentation livré avec le robot (voir photo sur la page suivante pour un exemple de configuration).

Si vous avez acheté aussi l'afficheur 16x2 caractères, vous devez le brancher et le monter sur le module d'extension AVANT le montage sur le robot.

Le câble en nappe à 14 points de l'afficheur est très flexible et se plie sans problème. Afin qu'il passe bien sous le LCD, vous devriez le plier pour le module d'extension RP6 CONTROL M32 comme indiqué sur la photo.



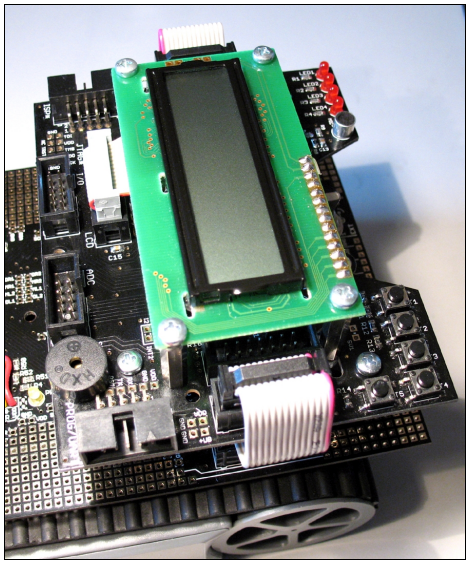
Ensuite vous pouvez fixer l'afficheur à l'aide d'entretoises de 20mm ou 25mm, d'écrous et de vis appropriées sur le module d'extension.

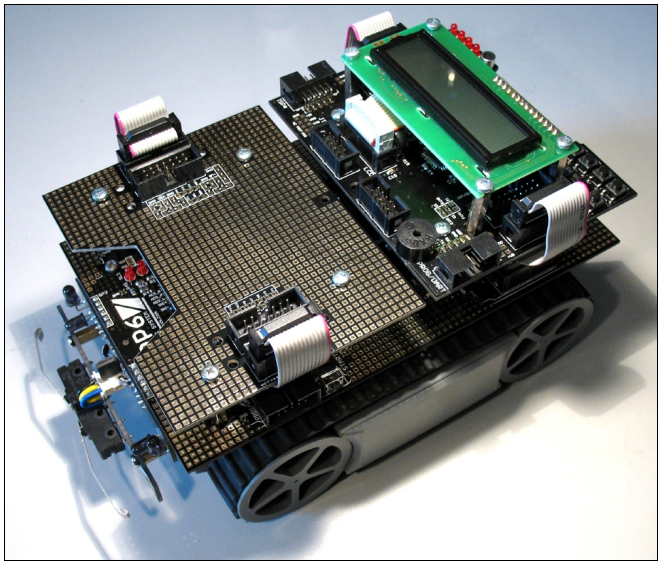
Vous pouvez également utiliser un autre afficheur de texte équipé d'un contrôleur compatible HD44780. Vous devez simplement souder un câble approprié sur l'afficheur. *Respectez scrupuleusement l'affectation des broches!*

Vous n'avez pas impérativement besoin de 4 entretoises comme indiqué sur la photo. Deux pièces (les deux à l'avant *ou* à l'arrière) suffisent déjà à fixer l'écran LCD.

Astuce: Le produit n'est pas livré avec des entretoises supplémentaires pour le montage de l'écran LCD mais chaque module d'extension comprend quatre entretoises, écrous et vis. Normalement un module d'extension est fixé par 4 entretoises comme indiqué sur la photo mais trois suffisent déjà – deux à l'avant et un à l'arrière au milieu. Avec le module d'extension et le PR6-M32, il vous resterait donc deux entretoises de 25mm, vis et écrous pour le LCD...

L'installation finie sur le robot pourrait ressembler à ceci:

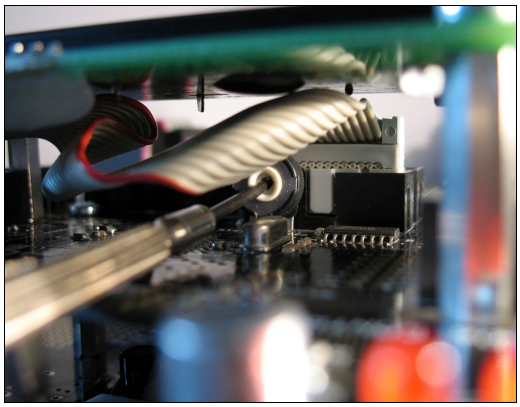




Il ne vous reste plus qu'à brancher les deux connecteurs d'extension à 10 contacts avec les ports I/O libres sur le module d'extension à l'aide de petits câbles en nappe à 10 points et d'utiliser les I/O et CAN pour l'évaluation de détecteurs et autres. Cela fonctionne aussi avec des modules montés sur d'autres niveaux. Les câbles en nappe doivent juste passer facilement par l'ouverture au milieu entre deux piles de modules.

Maintenant vous pouvez effectuer un petit **test de fonctionnement**:

Reliez d'abord l'interface USB connectée sur le PC via le câble en nappe à 10 points avec le connecteur PROG/UART sur le RP6-M32 et démarrez le RP6Loader. Mettez en suite le robot sous tension. Un message de texte doit s'afficher sur le LCD et une des LED doit clignoter. Si cela a fonctionné, cliquez sur « Connecter » dans le RP6Loader. Dans la fenêtre d'état un message doit apparaître « Connected to RP6 Control » Si cela a fonctionné aussi, vous devez lire le manuel en entier! Ensuite vous pouvez commencer avec l'exemple de programme.



Si le LCD n'affiche rien ou seulement deux rangées complètes de cases noires, mais que les LED clignotent et que la connexion au RP6Loader a fonctionné, vous devez régler le contraste du LCD (ou bien vous avez utilisé un autre LCD et l'affectation des broches est erronée). Cela se fait avec le potentiomètre R16 sur la platine. Vous pouvez régler le potentiomètre à l'aide d'un petit tournevis à lame plate. Un cruciforme fera aussi l'affaire mais cela se fonctionne pas avec tous les cruciformes parce qu'ils ne s'agrippent pas suffisamment...

Pour que le potentiomètre soit plus accessible, vous pouvez dévisser le LCD (ou ne pas le fixer avant d'être parfaitement réglé). Cependant, la fiche doit rester connectée. Ensuite vous pouvez tourner le potentiomètre avec un tournevis pendant que le robot est tous tension.

Ou bien vous laissez le LCD vissé et vous prenez un petit tournevis comme présenté sur la photo. **Veillez à ne pas toucher de composants ou de contacts sur la platine avec le tournevis!** Il est conseillé d'éteindre le robot, de mettre le tournevis en place, de remettre le robot sous tension et de régler le contraste...

3. RP6 CONTROL Library

Tout comme pour le robot, le RP6 CONTROL M32 dispose d'une bibliothèque de fonctions contenant de nombreuses fonctions utiles qui facilitent largement la tâche des débutants. La bibliothèque s'appelle RP6ControlLibrary ou, en abrégé, RP6ControlLib. Les fonctions Stopwatch, Delay, UART et Bus I²C sont identiques à celles de la RP6Library normale. Quant à l'UART et au bus I²C, les fichiers sont même identiques. Ils se trouvent dans la RP6Library dans le sous-dossier RP6common. Nous n'allons pas les redécrire ici. Reportez-vous au chapitre correspondant dans le manuel du RP6 et aux exemples de programme. Nous ne parlerons ici que des fonctions qui existent uniquement pour le RP6Control ou qui diffèrent un peu de celles contenues dans la RP6Lib.

En dépit des nombreuses fonctions toutes prêtes, la RP6ControlLib n'est qu'une base! Cette bibliothèque est loin d'être parfaite. Il reste encore beaucoup de choses à améliorer et à ajouter. Vous devrez montrer vos propres capacités de programmation.

3.1.1. Initialisation du Microcontrôleur

```
void initRP6Control(void)
```

Comme vous le savez déjà par la RP6Lib, cette fonction doit TOUJOURS être appelée en premier dans la fonction Main. Elle porte seulement un autre nom..

La fonction initialise les modules de matériel du microcontrôleur sur le RP6-M32. Le microcontrôleur ne fonctionnera correctement que si vous appelez cette fonction en premier! Une partie est certes déjà initialisée par le chargeur d'amorçage mais pas tout.

Exemple:

```
1  #include "RP6ControlLib.h"
2
3  int main(void)
4  {
5        initRP6Control(); // Initialisation - APPELER TOUJOURS EN PREMIER!
6
7  // [...] Code programme...
8
9        while(true);      // Boucle sans fin
10       return 0;
11 }
```

Chaque programme pour le RP6 CONTROL M32 doit se présenter au minimum ainsi! La boucle sans fin en ligne 9 est nécessaire afin de garantir une fin définie du programme! Sans cette boucle, le programme risque de se comporter autrement que prévu. *Exactement comme pour le contrôleur sur la carte-mère!*

3.1.2. LED d'Etat

La commande des LED est similaire à celle sur la carte-mère mais il n'y a que 4 LED et les désignations sont un peu différentes puisque les LED sont connectées à un registre à décalage externe qui commande en même temps le LCD. Le registre à décalage de 8 octets est appelé « External Port »

La fonction „setLEDs" existe aussi pour le RP6-M32:

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

Exemple:

```
setLEDs(0b0000); // Cette commande éteint toutes les LED.
setLEDs(0b0001); // Celle-ci allume la LED1 et éteint toutes les autres.
setLEDs(0b0010); // LED2
setLEDs(0b0100); // LED3
setLEDs(0b1010);   // LED4 et LED2
```

Voici une alternative:

```
externalPort.LED1 = true;  // Activer la LED1 dans le registre „External Port"
externalPort.LED2 = false;  // Désactiver LED2 dans le registre „External Port"
outputExt();               // Appliquer les modifications!

// La fonction outputExt() envoie le contenu de la variable externalPort
// au registre à décalage - comme le updateLEDs() de la RP6Lib.
// Cependant, des modifications des lignes de données LCD sont aussi envoyées.
```

3.1.3. Touches

A la différence des pare-chocs, les 5 touches sur le RP6-M32 sont connectées sur un canal CAN. Cela présente l'avantage qu'une seule broche suffit pour les 5 touches. L'inconvénient est que sur le circuit simple équipé de 5 résistances égales (voir schéma technique sur le CD-ROM) que nous utilisons ici, une seule touche peut être actionnée à la fois. Mais cela suffit amplement pour des touches de service.

```
uint8_t getPressedKeyNumber(void)
```

Cette fonction détermine quelle touche est actuellement appuyée. A cet effet, le CAN est lu et comparé à quelques valeurs de seuil pré-réglées. Il faut éventuellement adapter ces valeurs de seuil dans la Library pour que cela fonctionne bien avec votre propre RP6-M32, car la valeur CAN mesurée peut différer des résistances en raison des écarts de fabrication habituels. Vous trouverez les valeurs de seuil directement dans la fonction getPressedKeyNumber.

```
uint8_t checkPressedKeyEvent(void)
```

Cette fonction vérifie si une touche a été appuyée et renvoie une seule fois le numéro de touche – à la différence de getPressedKeyNumber, où le numéro de touche est renvoyé en permanence. C'est utile pour interroger les touches dans la boucle principale sans interrompre le flux du programme.

Cette fonction est un peu similaire:

```
uint8_t checkReleasedKeyEvent(void)
```

Toutefois, ici la valeur de la touche n'est renvoyée qu'une seule fois après que la touche a été relâchée. Cette fonction ne bloque pas non plus le flux normal du programme. Ce n'est pas nécessaire d'attendre avec une boucle jusqu'à ce que la touche soit relâchée.

3.1.4. Beeper

L'avertisseur sonore sur le RP6-M32 peut être commandé avec la fonction

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

Toutefois, cette fonction n'est pas bloquante, c'est-à-dire elle ne fixe que la fréquence du son et la durée pendant laquelle le son sera généré. Ensuite elle quitte. Le beeper s'arrête automatiquement après la durée déterminée. Toutefois chaque nouvel appel de cette fonction écrase les réglages. Pour générer des mélodies ou des sons distincts, il est souvent plus simple d'utiliser la macro

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

Ici vous pouvez déterminer la hauteur du son, la durée et la pause après le son. La génération des pauses se fait avec mSleep. Cela durera donc time+delay millièmes de secondes avant que le programme ne poursuive.

Si vous ne voulez pas du tout régler des pauses ou durées, vous pouvez régler juste la fréquence du son avec

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

C'est utile pour des séries de sons continues (p.ex. sirène ou autre). Attention: Dans toutes les fonctions Beeper, la plage de réglage admise du « pitch » se situe entre 0 et 255, 255 étant la fréquence maximale.

3.1.5. Capteur de Microphone

Le RP6 CONTROL peut non seulement générer du son mais aussi y réagir. Certes pas à la fréquence mais au volume. Ainsi vous pouvez faire démarrer le robot par des sons forts.

Le circuit est conçu comme un « Peak Detector » (détecteur de crête). Il mesure l'amplitude du signal de microphone sur une durée variable et retient la valeur maximale. Ensuite le microcontrôleur peut mesurer la valeur maximale à l'aide d'un CAN et l'effacer ensuite. La valeur maximale est mémorisée dans un petit condensateur. Pour « effacer » cette valeur maximale, ce condensateur est déchargé.

Il faut d'abord décharger le condensateur avec la fonction

```
void dischargePeakDetector(void)
```

Ensuite vous pouvez, à des intervalles réguliers, déterminer la valeur maximale actuellement mesurée avec la fonction

```
uint16_t getMicrophonePeak(void)
```

Après avoir mesuré la valeur, la fonction appelle directement dischargePeakDetector ().

Un des exemples de programme montre comment cela peut être utilisé.

3.1.6. Ecran LCD

Le LCD est idéal pour afficher des valeurs de détecteurs et des messages d'état pendant que le robot n'est pas connecté sur le PC. La sortie sur le LCD fonctionne un peu comme pour l'interface série – mais il y a quand-même quelques particularités. Regardez les exemples de programme et vous comprendrez rapidement l'intérêt du LCD.

```
void initLCD(void)
```

Cette fonction doit toujours être appelée au début du programme afin d'initialiser le LCD.

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

Normalement, vous n'avez pas besoin de cette fonction (et de write4BitLCDData) – nous ne la décrivons ici que pour vous expliquer brièvement comment le LCD est commandé.

Le LCD fonctionne en mode 4-bit. Quatre lignes de données et deux lignes de commande suffisent donc (Enable (EN) et Register Select (RS), Read/Write (R/W) sont en permanence commutés sur la masse ce qui explique pourquoi le LCD peut être exclusivement écrit mais jamais lu - ce qui n'est pas nécessaire non plus). Tout comme les LED, les quatre lignes de données sont connectées sur le registre à décalage afin d'économiser des ports. A l'instar de la fonction setLEDs, setLCDD définit les lignes de données du LCD. Toutefois cette fonction place aussi brièvement le signal Enable pour que le LCD reprenne les données.

```
void write4BitLCDData(uint8_t data)
```

Puisque nous devons en fait envoyer des commandes et données de 8 bits, les octets à transmettre devront être répartis – et c'est exactement ce que fait la fonction wri-

te4BitLCDData: Les données à 8 bits sont réparties sur deux « nibbles » de 4 bits et transmises.

void writeLCDCommand(uint8_t cmd)

Cette fonction appelle write4BitLCDData mais commute la ligne RS sur low pour envoyer une commande au LCD.

void clearLCD(void)

Envoie au LCD la commande d'effacer le contenu affiché.

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

Efface une partie déterminée du LCD. Les paramètres sont: Ligne, position de départ sur la ligne et longueur de la zone à effacer..

Exemple:

```
clearPosLCD(0,10,5);    // Efface les 5 derniers caractères sur la
                        // première ligne du LCD!
```

void setCursorPosLCD(uint8_t line, uint8_t pos)

Place le curseur sur une certaine position sur le LCD. Le paramètre line peut être 0 pour la ligne supérieure et 1 pour la ligne inférieure. Le paramètre pos peut se situer entre 0 et 15 pour les LCD de 2x16.

void writeCharLCD(uint8_t ch)

Envoie un seul caractère au LCD. Cela se passe de la même façon que pour la fonction writeChar avec l'interface série. Toutefois il faut déjà s'assurer que le curseur du LCD se trouve sur la bonne position, car sinon vous ne verrez pas le texte!

Exemple:

```
setCursorPosLCD(1,5);  // Positionne le curseur sur la deuxième ligne, caractère
5.
writeCharLCD('R');     // maintenant „RP6" est sorti et commence
writeCharLCD('P');     // à partir de la position du curseur!
writeCharLCD('6');
```

void writeStringLCD(char *string)

A l'instar de la fonction correspondante pour l'interface série, writeStringLCD envoie une chaîne de caractères se terminant par zéro du SRAM vers le LCD. Vous ne devez donc utiliser cette fonction uniquement si le texte se trouve vraiment dans la RAM et n'est pas seulement pré-défini. A cet effet, la macro:

writeStringLCD_P(STRING)

s'y prête mieux car le texte est lu directement de la mémoire flash, sans faire le détour par la mémoire de travail.

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

Cette fonction permet de sortir un texte d'une longueur définie sur le LCD. Les paramètres sont identiques à ceux de la fonction correspondante pour l'interface série.

showScreenLCD(LINE1,LINE2)

Afin de simplifier la sortie de texte sur le LCD, cette fonction permet d'écrire les deux lignes du LCD avec seulement un appel. Le curseur se place automatiquement au bon endroit et le contenu de l'écran précédent est effacé auparavant.

Exemple:

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

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

C'est la fonction déjà connue de l'interface série afin de sortir des chiffres dans les formats BIN, OCT, DEC ou HEX sur l'écran.

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

Mis à part le nom, writeIntegerLengthLCD est identique à la fonction déjà connue pour l'interface série.

3.1.7. Bus SPI et EEPROM SPI

L'EEPROM et le registre à décalage 8 bits sont connectés sur le bus SPI (= Serial Peripheral Interface). Vous pouvez braser en option un socle pour un autre EEPROM compatible AT25256 (p.ex. ST M95256) dans un boîtier DIP à 8 broches. Vous pourriez aussi commander d'autres CI SPI et monter en cascade un registre à décalage avec celui sur la carte-mère – mais vous ne devriez faire cela qui si vous visez une application spéciale et vous ne pouvez pas utiliser le bus I²C!

Puisqu'il existe des fonctions spéciales pour l'EEPROM et le registre à décalage, ce n'est pas nécessaire d'accéder directement à partir de son propre programme au bus SPI. Si jamais cela s'avère quand-même nécessaire, vous pouvez utiliser les fonctions suivantes.

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

Transmet un octet de données par le bus SPI.

```
writeWordSPI(uint16_t data)
```

Transmet deux octets de données qui sont transférés dans une variable à 16 bits via le bus SPI en commençant par le High Byte.

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

Transmet jusqu'à 255 octets d'un tableau via le bus SPI. Le nombre d'octets à transmettre dans le « buffer » du tableau est indiqué par le paramètre « length ».

```
uint8_t readSPI(void)
```

Lit un octet de données du bus SPI.

```
uint16_t readWordSPI(void)
```

Lit deux octets du bus SPI et les renvoie comme variable à 16 bits. Le High Byte est l'octet qui sera lu en premier.

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

Lit jusqu'à 255 octets du bus SPI dans un tableau de dimension appropriée.

Comme déjà dit, normalement vous n'avez pas besoin des fonctions SPI mais celles-ci seront utilisées dans les fonctions décrites ci-après afin d'accéder à l'EEPROM connecté sur le bus SPI.

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

Lit un seul octet à l'adresse « memAddr » de l'EEPROM. Pour notre EEPROM de 32 kb, l'adresse peut se situer entre 0 et 32767,

Exemple:

```
// Dans la ligne suivante, nous lisons un octet à l'adresse 13860 de l'EEPROM:
uint8_t data = SPI_EEPROM_readByte(13860);
```

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

En commençant par l'adresse « startAdr », lit jusqu'à 255 octets (length) dans un tableau de dimension suffisante (buffer).

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

Mémorise un octet (data) dans une adresse donnée (memAdr) dans l'EEPROM.

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

Mémorise jusqu'à 64 octets (length) dans le tableau « buffer » en commençant par « startAddr » dans l'EEPROM.

N'oubliez pas que seulement 64 octets peuvent être écrits en même temps. 64 octets correspondent à la dimension d'une page (pagesize) de l'EEPROM et il est impossible de l'enregistrer temporairement avant de l'écrire. En plus, les données qu'il faut écrire à la suite doivent toujours se trouver sur la même page, donc p.ex. entre 0 et 63, 64 et 127, 128 et 191...! Si la dimension de page est dépassée, l'EEPROM écrase les données au début de la page en cours. Vous êtes libre de commencer à écrire p.ex. à l'adresse 50 mais s'il faut écrire plus de 14 octets, le compteur d'adresse recommence à 0 et écrase les données qui s'y trouvent déjà.

Lors de la lecture des données de l'EEPROM, la dimension de page n'a pas d'importance et vous pouvez théoriquement lire la totalité de l'EEPROM en une seule fois.

Il faut 5ms à l'EEPROM pour écrire les données. Pendant cette période, vous ne pouvez pas accéder à l'EEPROM. Afin de connaître l'état actuel, vous pouvez utiliser la fonction

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

Ainsi, avec

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

vous pouvez interroger pour savoir si l'EEPROM n'est plus occupé à écrire des données. Cependant, les fonctions s'en chargent déjà d'elles-mêmes. Vous n'avez besoin de cette fonction qui si vous voulez faire d'autres choses pendant ce temps.

3.1.8. Les CAN

Les CAN sont lus avec la fonction déjà connue de la RP6Lib:

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

Une variante automatique qui lit dans l'ordre les canaux CAN en tâche de fond n'existe pas (encore) pour le RP6-M32.

Les canaux sont évidemment appelés autrement que dans la RP6Lib. Les canaux suivants sont disponibles:

```
ADC_7       --> ADC canal 7 – disponible sur la fiche à 10 broches „ADC“!
ADC_6       --> ADC canal 6 ...
ADC_5
ADC_4
ADC_3
ADC_2       --> ADC canal 2
ADC_KEYPAD  --> Les touches sont branchées ici
ADC_MIC     --> et le microphone ici.
```

3.1.9. Ports I/O

Puisque le RP6 CONTROL dispose de 14 ports I/O libres, nous ne décrivons ici que succinctement comment on accède généralement aux ports I/O d'un AVR.

L'ATMEGA32 possède 4 ports I/O à 8 bits chacun. Chaque port est commandé via 3 registres, à savoir un registre pour la « direction » des broches I/O (DDRx), donc si une broche est commutée en entrée ou en sortie, un registre pour l'écriture (PORTx) et un registre pour la lecture (PINx).

Si vous voulez utiliser une broche I/O comme sortie p.ex. Pour commuter une LED, vous devez mettre le bit correspondant dans le registre DDRx sur 1.

```
Exemple:
DDRC |= IO_PC7; // PC7 est maintenant une sortie
DDRC = IO_PC7 | IO_PC6 | IO_PC5; // PC5, PC6, PC7 sont maintenant des sorties,
                                 // toutes les autres broches sont des entrées!
```

Ensuite vous pouvez commuter la sortie sur le niveau high ou low via le registre PORTx.

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

Si un bit est à 0 dans le registre DDRx, la broche correspondante est configurée en entrée.

```
Exemple:
DDRC &= ~IO_PC6;  // PC6 est maintenant une entrée
```

Ensuite, le registre PINx nous renseigne sur l'état de la broche, à savoir si un haut (high) ou bas (low) niveau est appliqué à la broche:

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

Par ailleurs, vous pouvez également activer les résistances de tirage intégrées dans le microcontrôleur en définissant les bits dans le registre PORTx. C'est notamment très utile pour des détecteurs tactiles et autres.

Les broches I/O sont librement disponibles sur le RP6 CONTROL M32 (vous trouverez les définitions exactes dans le fichier des en-têtes RP6Control.h):

```
IO_PC7
IO_PC6
IO_PC5
IO_PC4
IO_PC3
IO_PC2

IO_PD6
IO_PD5
```

Les canaux CAN peuvent également servir de broches I/O! Regardez les différences d'écriture par rapport aux désignations ci-dessus (ADC_7 vs. ADC7)!

```
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
```

Information importante: Chaque broche I/O est conçue pour un courant maximum de 20mA. Donc, un port de 8 bits ne doit pas être chargé à plus de 100mA. Si vous souhaitez commuter des appareils de plus forte consommation, vous devez utiliser des transistors externes!

Pour avoir des informations plus précises, vous devez consulter la fiche technique du MEGA32 qui se trouve sur le CD-ROM.

4. Exemples de Programme

Vous trouverez quelques exemples de programme sur le CD. Ils présentent les fonctions de base du RP6 CONTROL M32. Tout comme pour le robot, ils ne constituent pas la solution optimale et s'entendent comme points de départ pour vos propres programmes. C'est fait exprès pour vous laisser un peu de travail. Ce serait tout de même ennuyeux d'essayer tout simplement des programmes pré-écrits...

Vous avez la possibilité d'échanger vos programmes avec d'autres utilisateurs par Internet. Le RP6ControlLib et tous les exemples sont sous Licence Open Source « GPL » (General Public Licence) et vous êtes donc autorisés de modifier, publier et mettre à la disposition d'autres utilisateurs les programmes sous les conditions de la GPL.

D'une manière générale, ils existent déjà de nombreux exemples de programme sur Internet pour le MEGA32 puisque le contrôleur de la famille AVR est très populaire auprès des utilisateurs amateurs. Toutefois, il faut toujours veiller à adapter d'autres exemples de programme au matériel du RP6 CONTROL et de la RP6ControlLib sinon, il y aura souvent des problèmes (les plus courants sont des affectations différentes des broches, utilisation de modules de matériel déjà utilisés ailleurs tels que des timers, autre cycle horloge, etc.).

Exemple 1: Programme „Hello World" avec sortie de texte et séquenceur à LED
Répertoire: <RP6Examples>\RP6ControlExamples\Example_01_LCD
Fichier: RP6Control_LCD.c

Le programme génère des sorties sur l'interface série ou l'écran LCD, vous devriez donc connecter le robot aussi sur le PC et regarder les sorties sur le terminal du logiciel RP6Loader! En option, vous pouvez connecter le LCD!

Le robot ne bouge pas dans cet exemple – dans la mesure où vous n'avez chargé que le programme esclave du bus I²C dans le contrôleur sur la carte-mère! **Vous pouvez donc le placer sur une table à côté de l'ordinateur.**

Ce programme sort un petit texte « Hello World » via l'interface série et ensuite il exécute une séquence de lumières. Par ailleurs le LCD affiche d'abord un texte statique et ensuite un texte mobile où les deux mots « HELLO » et « WORLD » se déplacent lentement vers la gauche et la droite. Au bout de 16 secondes, il y a une courte pause qui est signalée par 2 bips courts. 8 secondes plus tard, le programme recommence, également indiqué par deux bips courts.

Exemple 2: Touches et Sons
Répertoire: <RP6Examples>\RP6ControlExamples\Example_02_Buttons
Fichier: RP6Control_Buttons.c

Le programme génère des sorties sur l'interface série et le LCD

Le robot ne bouge pas dans ce programme!

Cet programme démontre l'utilisation des 5 touches sur le RP6 CONTROL. A chaque pression sur une touche, le numéro de touche est indiqué sur le LCD et le Piézo émet une suite de sons.

(Attention: Cela peut devenir enquiquinant d'appuyer sur T4 ;-)).

Exemple 3: Détecteur à micro
Répertoire: <RP6Examples>\RP6ControlExamples\Example_03_Microphone
Fichier: RP6Control_Microphone.c

Le programme génère des sorties sur l'interface série et le LCD!

Le robot ne bouge pas dans ce programme!

Le détecteur à microphone permet de capter des bruits forts. Ce programme représente également le volume mesuré sous forme d'échelle, aussi bien sur le LCD qu'avec les LED. Il indique en même temps la valeur mesurée. Tapotez avec le doigt sur le microphone afin de tester s'il fonctionne bien. Frappez dans les mains ou faites un autre bruits fort et regardez la réaction sur le LCD et les LED.

Exemple 4: EEPROM externe
Répertoire: <RP6Examples>\RP6ControlExamples\Example_04_EEPROM
Fichier: RP6Control_04_EEPROM.c

Le programme génère des sorties sur l'interface série et sur le LCD!

Le robot ne se déplace pas dans cet exemple!

Ce programme illustre comment on peut accéder en principe à l'EEPROM externe en écriture et en lecture. D'abord, on sort et lit les deux premières pages à 64 octets chacune. A titre de preuve que l'EEPROM conserve son contenu même après la mise hors tension du robot, éteignez tout simplement le robot après exécution du programme et remettez-le sous tension. Les données écrites en dernier sont conservées dans l'EEPROM et sont sorties en premier.

Ensuite la première page du programme est écrite avec 64 octets. Ensuite les 128 premiers octets sont lus pour vérifier qu'ils n'ont été écrits que sur la première page et que le reste de l'EEPROM n'a pas été modifié (alors le contenu de la ligne de mémoire est de 255).

Il est également possible d'écrire et de lire des octets individuels ce qui sera démontré ci-après. La ligne de mémoire portant l'adresse 4 sera écrite avec 128 et les deux premières pages seront lues, cette fois-ci toutefois octet par octet ce qui prend bien sûr plus de temps qu'une page complète.

Lorsque tout cela a été fait, un petit séquenceur de lumière est exécuté.

Exemple 5: Convertisseur analogique/numérique et Ports I/O
Répertoire: <RP6Examples>\RP6ControlExamples\Example_05_IO_ADC
Fichier: RP6Control_05_IO_ADC.c

Le programme génère des sorties sur l'interface série et l'écran LCD!

Le robot ne se déplace pas dans cet exemple!

Le fonctionnement est identique à celui du robot mais comme il est fréquemment utilisé dans ce module d'extension, cet exemple démontre brièvement comment les CAN et I/O libres peuvent être utilisés.

Les CAN et I/O portent les désignations suivantes dans le programme:

```
ADC7    (1 << PINA7) // ADC canal 7  - utilisable également comme broche I/O normale
ADC6    (1 << PINA6) // Canal 6 ...
ADC5    (1 << PINA5)
ADC4    (1 << PINA4)
ADC3    (1 << PINA3)
ADC2    (1 << PINA2) // Canal 2. Les canaux 0 et 1 sont pris par le clavier et le microphone.


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

(du fichier RP6Control.h)

Le programme ne fait rien de particulier – donc vous n'êtes pas obligé de l'essayer. Rien n'est connecté sur les broches I/O. Ce programme ne sert que si l'on veut commander son propre matériel via les I/O.

Exemple 6: Interface Bus I²C – Mode Maître
Répertoire: <RP6Examples>\RP6ControlExamples\Example_06_I2CMaster
Fichier: RP6Control_06_I2CMaster.c

Ce programme démontre comment on peut utiliser le mode maître du bus I²C. Le contrôleur sur la carte-mère du robot doit avoir chargé le programme Esclave I²C!

Ce programme montre comment on peut commander le contrôleur sur la carte-mère en mode Esclave. Cela ne peut fonctionner que si le programme I²C Slave des exemples de la RP6Base a été chargé dans le contrôleur sur la carte-mère.

L'accès au bus I²C fonctionne presque de la même façon que pour le contrôleur sur la carte-mère – et il s'agit des mêmes fonctions.

Ce programme permet d'envoyer différentes commandes du bus I²C au contrôleur programmé avec le programme Esclave sur la carte-mère en appuyant sur l'une des 5 touches. Touche T1 augmente un compteur de 1 et envoie la commande setLEDs à l'esclave qui possède cette valeur de compteur. Ainsi les 6 LED d'état sur la carte-mère représentent un compteur binaire.

Appuyez sur la touche T2 pour lire tous les registres et les sortir par l'interface série. T3 par contre ne lit et affiche sur le LCD que les valeurs des détecteurs de lumière.

T4 et T5 envoient la commande « rotate » et le robot tourne un peu à droite ou à gauche (vous pouvez également appuyer plusieurs fois sur la touche et il tourne davantage...).

Comme tous les autres, ce programme est librement modifiable et convient parfaitement au test de nouveaux équipements I²C ou fonctions à ajouter au programme Esclave.

Exemple 7: Interface de Bus I²C – Mode Maître – Réagir aux interruptions
Répertoire: <RP6Examples>\RP6ControlExamples\Example_07_I2CMaster
Fichier: RP6Control_07_I2CMaster.c

Ce programme démontre comment on peut utiliser le mode Maître du bus I²C. Le contrôleur sur la carte-mère du robot doit avoir chargé le programme Esclave I²C!

Vous avez peut-être déjà remarqué les signaux d'interruption sur les connecteurs XBUS du RP6? Vous pouvez les utiliser afin de réagir aux changements des détecteurs sans interroger constamment les esclaves. Après tout, chaque accès au bus prend du temps.

Le CAN sur le robot en est un bon exemple. L'état du détecteur ne change que *relativement* peu souvent et ce ne serait pas très efficace d'envoyer constamment une requête via le bus s'il y a eu un changement. Dès que l'état du CAN change, le programme Esclave met le signal INT sur le niveau High. Puisque INT1 a été appliqué à l'entrée d'interruption 0 du MEGA32 sur le RP6 CONTROL M32, le contrôleur peut réagir directement à cet événement et interroger l'état du contrôleur sur la carte-mère.

Toutefois, nous n'utilisons PAS de routines d'interruption dans les exemples de programme pour réagir à un événement mais interrogeons l'état de la broche à chaque passage de la boucle principale. Puisque les transferts de bus I²C sont également commandés par des interruptions, aucun nouveau transfert ne pourrait avoir lieu au sein de la routine d'interruption. Il est nécessaire d'appeler impérativement la fonction

task_I2CTWI() dans la boucle principale qui règle le déroulement du transfert I2C. Il n'y aura donc aucun intérêt à utiliser une routine d'interruption. Pour être précis, cela risquerait même de créer des problèmes car des transferts de bus I2C en cours seraient interrompus. C'est pourquoi nous utilisons la fonction task_checkINT0() pour évaluer constamment le signal d'interruption et d'envoyer éventuellement une requête. Dès la lecture du registre d'état 0 de l'esclave, le signal d'interruption est remis à zéro. Les trois premiers registres de l'esclave nous disent ce qui a déclenché l'interruption.

C'est exactement ce que fait ce programme. Le résultat en est que l'état actuel du CAN est indiqué par les 4 LED, le LCD, l'interface série et par le buzzer sur le RP6 CONTROL

La puissance de transmission du CAN est réglée au début du programme par le bus I²C. Outre le CAN, le programme réagit aussi aux pare-chocs et aux éventuelles transmissions RC5 provenant d'une télécommande ou d'autres robots.

Vous pouvez procéder d'une manière similaire avec d'autres détecteurs du robot ainsi que éventuellement avec d'autres modules d'extension qui seront disponibles avec d'autres détecteurs.

Un autre détail du programme est l'indication « Heartbeat », donc « battement de cœur ». La fonction task_LCDHeartbeat() fait clignoter sur le LCD en permanence le caractère '*' à une fréquence de 1Hz. C'est très utile pour savoir si le programme a planté en totalité ou si seulement une petite partie du logiciel présente des erreurs. Si vous écrivez vos propres programmes et ce programme semble planter complètement, cette fonction peut s'avérer très utile pour délimiter la source d'erreur. Le plantage du programme est tout à fait courant pendant le développement.

C'est pourquoi le programme esclave I²C pour le contrôleur sur la carte-mère contient une fonction « software watchdog ». Si le maître ne réagit pas pendant un laps de temps déterminé à l'événement d'interruption (en lisant le registre 0), tous les systèmes de l'unité de base du robot sont coupés et le programme s'arrête. Avant tout, les moteurs sont arrêtés! Car si le contrôleur Maître plante mais a eu le temps d'envoyer la commande d'avancer de 10cm/sec., le robot roule sans retenue vers le prochain obstacle et ne s'arrête même pas dans le cas d'une collision ...

Le timer du logiciel watchdog est d'abord désactivé. Il faut d'abord envoyer une instruction via le bus I²C pour activer le watchdog. Il est également possible de configurer le watchdog de façon à ce que celui déclenche toutes les 500ms un événement d'interruption afin de vérifier si le contrôleur maître y réagit encore. C'est ce que nous allons utiliser dans l'exemple suivant.

Exemple 8: Interface du Bus I²C – petite RP6 Library
Répertoire: <RP6Examples>\RP6ControlExamples\Example_08_I2CMaster
Fichier: RP6Control_08_I2CMaster.c

Ce programme démontre l'utilisation du mode maître du bus I²C. Le contrôleur sur la carte-mère du robot doit avoir chargé l'exemple de programme Esclave I²C!

Etant donné que les programmes perdent rapidement de leur clarté lorsque l'on y met trop de choses, l'exemple 7 sera divisé en deux fichiers C et un peu complété. Dès le début, il a été conçu de telle façon que l'on dispose d'une petite bibliothèque pour la commande du robot via le bus I²C qui s'utilise presque comme la RP6Lib normale pour le contrôleur sur la carte-mère. Un grand nombre de fonctions et de variables portent le même nom que celles dans la RP6Lib. Cela facilite l'utilisation commune de parties du programmes par la RP6Lib et la RP6ControlLib. Les fameux Event Handler pour l'ACS, l'IRCOMM et les pare-chocs sont à nouveau disponibles. A cela s'ajoutent de nouveaux Event Handlers pour l'état de faible charge des accus et les requêtes Watchdog. Dans l'exemple suivant, nous ajouterons encore les fonctions pour le contrôle de mouvement.

A part cela, le programme est similaire à l'exemple 7. Les seuls changements sont le rajout du timer Watchdog dont les requêtes apparaissent également sur le LCD, et la lecture de tous les registres des détecteurs et leur sortie sur l'interface série. L'ACS, les pare-chocs et les événements RC5 continuent à être représentés.

Exemple 9: Interface du bus I²C – Fonctions de mouvement
Répertoire: <RP6Examples>\RP6ControlExamples\Example_09_Move
Fichier: RP6Control_09_Move.c

Ce programme illustre comment on peut utiliser le mode maître du bus I²C. Le contrôleur sur la carte-mère du robot doit avoir chargé l'exemple de programme Esclave I²C!

ATTENTION: Le robot se déplace dans cet exemple!

Maintenant nous ajoutons à la nouvelle bibliothèque quelques fonctions de mouvement qui sont déjà connues par la RP6Lib: move, rotate, moveAtSpeed, changeDirection et stop. L'utilisation de ces fonctions est identique à celles de la RP6Lib. Dans cet exemple, nous avons tout retiré ce qui était déjà dans les autres exemples afin d'améliorer la clarté, à l'exception de l'indication du Watchdog, et utiliser le mode bloquant des fonctions de mouvement (quelques fonctions comme l'indication Heartbeat ne fonctionneraient de toute manière pas). Dans cet exemple, le robot avance et recule et tourne à 180° - tout comme dans l'exemple 7 pour la RP6Lib („RP6Base_Move_02.c“).

Exemple 10: Interface bus bus I²C – Robot basé sur le comportement
Répertoire: <RP6Examples>\RP6ControlExamples\Example_10_Move2
Ficher: RP6Control_10_Move2.c

Ce programme illustre comment on peut utiliser le mode maître du bus I²C. Le contrôleur sur la carte-mère du robot doit avoir chargé l'exemple de programme Esclave I²C!

ATTENTION: Le robot se déplace dans cet exemple!

La nouvelle bibliothèque permet de reprendre les exemples de programme relatifs au robot basé sur le comportement pratiquement à l'identique. Et c'est exactement ce qui a été fait ici avec le programme RP6Base_05_Move_05. Il n'y a eu que de petites modifications – entre autres, il faut commander les LED sur la carte-mère via la fonction setRP6Leds puisque setLEDs est déjà réservé pour les LED sur le RP6-M32...

Sinon, le programme est quasiment identique au programme déjà connu. Le robot se déplace et évite les obstacles. La différence est qu'il est commandé cette fois-ci par le RP6-M32.

La nouveauté est la représentation du comportement actuel sur le LCD et par les LED. Ainsi, vous voyez immédiatement quel comportement est actif. A cet effet, il existe une petite fonction auxiliaire qui assure que le texte n'est envoyé qu'une fois au LCD sinon il scintillerait sur l'écran. Pendant que le comportement « Cruise » est actif, les 4 LED d'état rouges exécutent une séquence de lumière.

L'état des accus est également surveillé. Si l'état de charge est très faible, le robot s'arrête. Toutefois, si les accus sont fraîchement chargés, il faut attendre un petit moment ...

Par ailleurs, le programme attend au début trois bruits forts (WAIT est alors écrit dans la deuxième ligne du LCD à côté du compteur de bruits forts) p.ex. frapper trois fois dans les mains. Il est également possible d'appuyer sur une touche quelconque sur le RP6-M32. Cela aussi a été réalisé avec un autre comportement.

Nous voici arrivés à la fin de ce petit additif. Maintenant vous pouvez laisser libre cours à votre propre créativité, écrire de nouveaux programmes et monter des détecteurs tout nouveaux sur le RP6 que vous pouvez commander avec le RP6-M32 et bien d'autres choses encore.

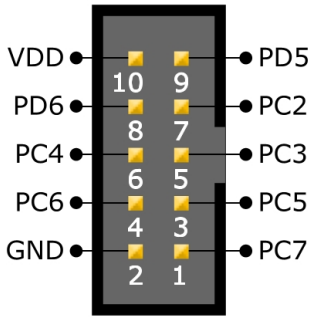
ANNEXE

A – Affectation des Broches

Ce chapitre contient les affectations des broches des fiches et cosses à souder les plus importantes.

Les broches du connecteur de l'interface série sont affectées exactement de la même façon que sur la carte-mère. Cela s'applique bien sûr également aux connecteurs XBUS et USRBUS!

Ports I/O:

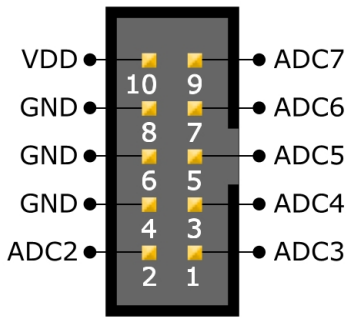


Sur le connecteur I/O, tous les ports I/O libres et la tension d'alimentation sont disponibles.

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

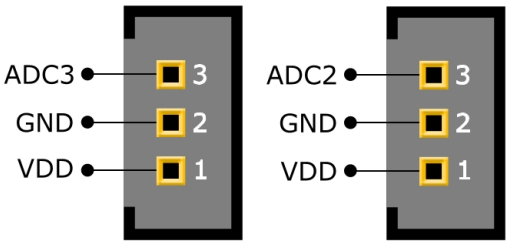
ATTENTION: Il est déconseillé de relier les broches de tension d'un autre module d'extension (p.ex. d'un module d'expérimentation) aux broches de tension de ces fiches afin d'éviter des boucles de masse et autres.

Canaux CAN:



Les 6 canaux CAN (qui sont évidemment utilisables comme broches I/O) sont tous disponibles sur la fiche CAN à 10 broches, également avec la tension d'alimentation

Deux des CAN sont appliqués à des broches non-implantées comme sur la carte-mère. Vous pouvez y souder vos propres connecteurs à l'espacement de 2,54mm.



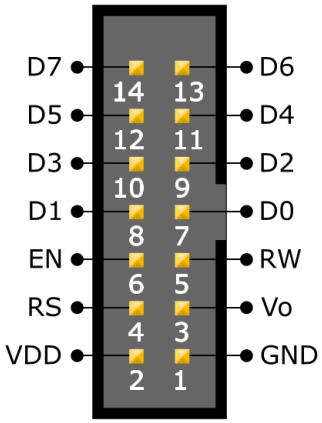
Mais soyez prudent et ne faites pas de dégâts! Il faut avoir déjà une certaine expérience dans la soudure pour ce faire.

Vous pouvez brancher deux détecteurs analogiques ou numériques sur les connecteurs (la tension de sortie des détecteurs peut se situer dans la plage de 0 à 5V) et les alimenter en 5V. Vous pouvez éventuellement encore implanter le gros condensateurs électrolytique. 220 à 470µF (pas plus!) conviennent à la plupart des applications.

Cela n'est nécessaire que si vous utilisez des détecteurs présentant un courant crête très élevé tels que les détecteurs de distance IR très populaires de chez Sharp. Des condensateurs de dérivation (100nF) sur la carte-mère ne se prêtent qu'à des longueurs de câbles très courts. Lorsque les chemins sont plus longs, il est préférable de les souder directement sur les détecteurs (ce qui est même vivement conseillé pour

des chemins de câble plus courts!).

Connecteur du LCD :



Si vous ne souhaitez pas utiliser le LCD standard, vous pouvez confectionner votre propre câble pour le LCD à l'aide des affections ci-contre.

Les fils D0, D1, D2 ,D3, RW sont fermement connectés sur la masse (GND) puisque nous n'utilisons le LCD qu'en mode 4 bits et n'avons pas besoin de le lire.

Respectez scrupuleusement les affectations et veillez à ne pas connecter la fiche à l'envers!

Les désignations des différentes broches peuvent varier d'un fabricant à l'autre mais normalement les désignations sont identiques à celles utilisées ici et vous pouvez brancher les broches 1:1 à l'afficheur.