Conrad sur INTERNET
www.conrad.fr

Note de l'éditeur
Cette notice est une publication de la société Conrad, 59800 Lille/France.

Cette notice est conforme à l'état du produit au moment de l'impression.
Données techniques et conditionnement soumis à modifications sans avis préalable.
 XXX/06-11/JV

N O T I C E

Version 06/11

Module d'extension RP6 Control M32

Code : 191550

Cette notice fait partie du produit. Elle contient des informations importantes concernant son utilisation. Tenez-en compte, même si vous transmettez le produit à un tiers.

Conservez cette notice pour tout report ultérieur !

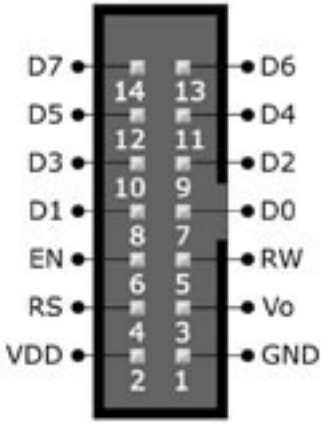
IMPORTANT
À lire absolument

Avant de mettre en marche votre module d'extension RP6, il est important de lire entièrement ce mode d'emploi ET celui du SYSTEME ROBOT RP6 ! Elles contiennent des explications importantes quant à l'utilisation et vous informent d'éventuels dangers possibles ! De plus, elles possèdent des informations importantes, qui ne sont nullement évidentes pour la plupart des utilisateurs. Le mode d'emploi du RP6 CONTROL M32 n'est qu'un complément !

En cas de non observation de ce mode d'emploi et de celui du système robot RP6, toute garantie prend fin ! De plus, AREXX Engineering ne sera tenu pour responsable pour tout type de dommages, résultant d'une non observation de ce mode d'emploi !

Prêtez toute votre attention au paragraphe « consignes de sécurité » du mode d'emploi du SYSTEME ROBOT RP6 !

Connexion LCD :



Si vous ne souhaitez pas installez le LCD standard, vous pouvez faire votre propre câble tout un utilisant le bon pin pour le LCD.

Les lignes D0, D1, D2, D3 et RW fermement reliées au GND, puisque nous n'utilisons que le mode 4 bit du LCD et qu'il n'est pas nécessaire de lire.

Veillez absolument à utiliser les bonnes affectations de broches et à ne pas relier les connecteurs à l'envers !

Les désignations des pins varient d'un fabricant à un autre, mais elles sont généralement identiques à celles utilisées ici. Dans ce cas, vous pouvez connectez le pin 1:1 à l'écran !

Remarques sur la garantie limitée et la responsabilité

La garantie de AREXX Engineering se limite à l'échange ou à la réparation du robot et de ses accessoires, dans le délai de garantie légale en cas de fautes de production avérées, comme des dommages mécaniques, des pièces d'équipement manquantes ou fausses, exceptés tous les composants branchés à des connecteurs/socles.
Il n'existe aucune responsabilité pour des dommages résultant directement ou après l'utilisation du robot. Les réclamations reposant sur des consignes légales nécessaires sur la responsabilité du fait des produits restent intactes.

Dès que vous procédez à des modifications irréversibles sur le robot ou accessoire (par exemple : soudage de pièces supplémentaires, forage d'autres trous, etc.) ou que le robot subit des dommages suite à la non observation de ce mode d'emploi, toute garantie prend fin !

Il ne peut être garanti, que le logiciel fourni avec chaque version suffit ou qu'il est possible de travailler entièrement avec sans interruption ou erreur.
De plus, le logiciel fourni est modulable et peut être chargé dans l'appareil par l'utilisateur. Ainsi, l'utilisateur prend un risque quant à la qualité et l'efficacité de l'appareil y compris tous les logiciels.
AREXX Engineering garantit la fonctionnalité de l'exemple d'application fourni tout en observant les conditions spécifiées dans les données techniques. S'il est prouvé que le robot ou le logiciel pc est en plus défectueux ou insuffisamment, le client prend en charge tous les coûts pour le service, la réparation ou la correction.
Tenez également compte de l'accord de licence du CD-ROM !

Symboles

Les différents symboles suivants sont utilisés dans ce mode d'emploi :

Le symbole « attention » indique un paragraphe particulièrement important, que vous devez soigneusement respecter. Si vous effectuez une erreur, cela peut conduire à une destruction du robot ou de ses accessoires et même menacer votre personne ou la santé des autres !

Le symbole « information » vous indique un paragraphe contenant des trucs et astuces ou des informations de fond. Il n'est pas essentiel de tout comprendre, mais cela reste tout de même très utile.

SOMMAIRE

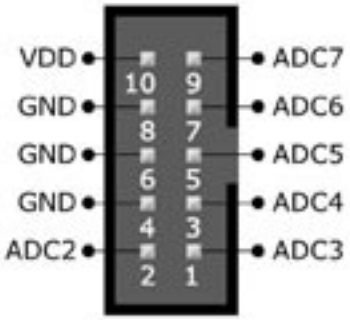
1. Le module d'extension RP6 CONTROL M32

Grâce au module d'extension RP6 CONTROL M32 (ou RP6-M32), vous pouvez ajouter à votre robot un deuxième micro processeur Atmel ATMEGA32, qui est pourtant cadencé deux fois plus rapidement que le contrôleur sur la carte mère. Sinon, l'utilisateur dispose également d'un temps de calcul plus long, puisque le contrôleur ne s'occupe pas du réglage moteur, ACS, IRCOMM, etc.

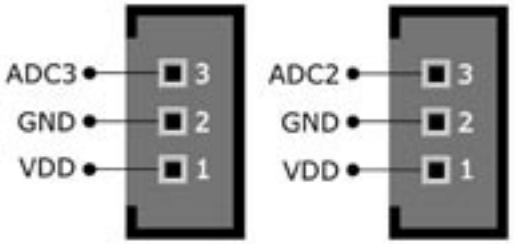
Le SPI EEPROM 32 Ko externe fournit au module une mémoire permanente réinscriptible (1 million de cycles), pouvant être utilisée comme enregistreur de données ou sauvegarde de programme pour Byte code Interpreter (par exemple NanoVM pour Java). Un socle DIP à 8 pôles peut être soudé au module si vous le souhaitez, et également ajouter un deuxième EEPROM dans un boîtier DIP 8.

Les boutons, les LED, les bipeurs piézoélectriques et l'écran LC optionnel vous offrent d'autres possibilités intéressantes. Ces derniers vous permettent de contrôler le robot et par exemple de programmer un petit menu de commande permettant de démarrer des programmes différents en appuyant sur les boutons – et bien naturellement il peut être affiché les mesures et les messages de statuts. Le bipeur peut générer différentes tonalités et peut, par exemple, jouer une mélodie au démarrage du programme ou un son d'avertissement lorsque l'accu est presque déchargé.
Afin de contrôler votre propre circuit sur les modules d'extension, vous avez 14 ports I/O libres de disponible, sur 2 connecteurs standard à 10 pôles. 6 de ces 14 I/O peuvent être utilisés comme canaux comme canaux convertisseurs analogique/numérique.

Canaux ADC :



Les 6 canaux ADC libres (utilisables également comme pins I/O) sont disponibles sur un connecteur ADC à 10 pôles, fournissant aussi la tension d'alimentation.



Tout comme sur la carte mère, deux des ADC sont disponibles sur des connecteurs nus. Cela vous permet de souder vos propres connecteurs sur une grille de 2,54mm.

Mais faites bien attention et éviter toute soudure excessive ! Il vaut mieux avoir de l'expérience en soudure avant d'effectuer cette manipulation.

Vous pouvez aussi relier deux capteurs analogiques ou numériques (la tension de sortie des capteurs doit se trouver en 0 et 5 V) et les alimenter avec une tension de 5 V. Vous pouvez également l'équiper d'un grand condensateur électrolyte - une valeur de 220 à 470 µF (pas plus !) suffit amplement pour la plupart des utilisations.

Toutefois, cela n'est pas nécessaire, à moins que vous n'utilisiez des capteurs avec un haut courant de pointe – comme par exemple le populaire capteur de distance Sharp IR. Les condensateurs de découplage (100nF) de la carte mère ne sont adaptés que pour de courtes alimentations – Pour de plus longues alimentations, il faut les souder directement sur les capteurs (ce qui est également recommandé pour de courtes alimentations !).

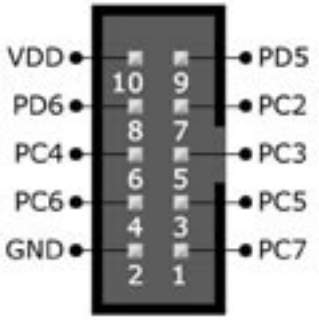
ANNEXE

Affectations des broches

Vous trouverez dans ce chapitre les affectations des broches des connecteurs et des blocs à souder les plus importants.

La connexion de l'interface séquentielle a exactement la même affectation des broches que sur la carte mère. Cela fonctionne aussi pour les connexions XBUS et USRBUS !

Ports I/O :



Les connecteurs I/O fournissent une disponibilité des ports I/O libres et la tension d'alimentation.

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

ATTENTION : Veuillez ne pas relier un connecteur de tension d'un autre module d'extension (par exemple un module d'expérimentation), afin d'éviter les boucles de masse ou problèmes similaires.

De plus, le module possède un capteur microphone, déjà présent sur le prédécesseur CCRP5. Il vous permet de démarrer le RP6, par exemple, en tapant dans vos mains ou des choses similaires. Le circuit passe pour un « détecteur Peak », seules les valeurs acoustiques maximales sont identifiées. Ainsi les volumes des bruits environnants peuvent être sommairement mesurés et bien évidemment réagir en fonction. (Cela ne fonctionne correctement que lorsque les moteurs sont éteints, car le microphone est particulièrement sensible aux sons émis par le robot…)

Avant que vous ne démarriez quoique ce soit avec le RP6-M32, vous devez absolument vous familiariser avec le robot en testant tous les programmes d'exemple SANS que le module d'extension RP6-M32 ne soit monté dessus. Il faut considérer ce mode d'emploi comme un petit ajout au grand mode d'emploi du RP6. Dans tous les cas, veuillez lire ce manuel avant d'utiliser votre RP6-M32 !

Indication importante pour les débutants : Les programmes écrits pour le RP6-M32 ne fonctionnent ABSOLUMENT PAS correctement sur le micro contrôleur de l'unité de base et vice et versa (disposition des broches et fréquence élémentaire totalement différentes) !

Attention : Charger un programme dans le mauvais contrôleur peut endommager celui-ci ou les circuits commandés ! Par exemple, lorsqu'une broche I/O est normalement utilisée comme entrée et que le programme du mauvais contrôleur la considère comme sortie, la broche I/O sera en surcharge ou causera des dommages au circuit !

Généralement, il n'arrive rien de grave si vous vous trompez, mais nous ne pouvons pas vous le garantir ! Le Loader RP6 ne peut pas distinguer les programmes adaptés pour tels contrôleurs, puisque les fichiers hex sont similaires. Il ne vous empêchera donc pas de charger des programmes sur le mauvais contrôleur ! Afin d'éviter cela, utilisez les fonctions du Loader RP6 afin de créer différentes catégories ! A chaque module d'extension sa catégorie…

1.1. Support technique

Pour toute question ou problème, vous pouvez joindre notre Support Team comme suit sur Internet (avant de nous contacter, veuillez lire attentivement et entièrement ce mode d'emploi ! La plupart des réponses à vos questions se trouvent dedans !) :

- sur notre forum : http://www.arexx.com/forum
- par e-mail : info@arexx.nl
- adresse postale :
  AREXX Engineering
  Nervistraat 16
  8013 RS ZWOLLE
  PAYS-BAS
- Pour toutes les informations de contact, les mises à jour logiciel et autres informations, rendez-vous 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

Vous devez trouver les articles suivants dans la boîte du RP6 CONTROL M32 :

• Le module RP6-M32
• 4 x boulons d'écartement 25 mm M3
• 4 x vis M3
• 4 x écrou M3
• 2 x câble ruban 14 pôles

Le logiciel et le mode d'emploi PDF se trouvent sur le CD-ROM du RP6. Les versions actualisées du logiciel et ce mode d'emploi se trouvent également sur notre page d'accueil !

1.3. Caractéristiques et données techniques

Cette partie vous offre un aperçu de ce que le RP6 CONTROL M32 vous offre et sert en même temps d'introduction de certains termes et appellations des composants du module.

Caractéristiques, composants et données techniques du RP6 CONTROL M32 :

• Micro contrôleur Atmel ATMEGA32 8Bit plus performant
  o Vitesse de 16 MIPS (= 16 millions d'instruction par seconde) à une cadence de 16 MHz, deux fois plus rapides que le contrôleur présent sur la carte mère du RP6 !
  o Mémoire : flash Rom 32 Ko, SRAM 2Ko, EEPROM 1Ko
  o Librement programmable en C (avec WinAVR / avr-gcc)
  o … et bien plus encore (cf. feuilles de données)

• EEPROM SPI 32Ko externe
  o Interface SPI très rapide (fréquence de cadence de 8MHz)
  o Chaque cellule mémoire est 1 000 000 fois réinscriptible.
  o …retrouvez plus d'infos techniques sur la fiche technique du CD-ROM (AT25256A) !
  o Idéal pour les enregistrements de données ou comme sauvegarde pour les Byte code Interpreter (par exemple, un Java VM tel que le Nano VM : http://www.harbaum.org/till/nanovm/ . Ce VM doit de toutes façons devenir adapté pour utilisé avec l'EEPROM externe…)

• Connecteurs d'extension Bus I²C
  o Peut contrôler tous les Bus Slaves I²C.
  o Le MEGA32 du module peut être utilisé comme Master ou Slave. Généralement, il vaut mieux l'utiliser comme Master pour un contrôle complet du robot (le contrôleur de la carte mère se charge en effet de la régulation de la vitesse du moteur, ACS, IRCOMM, surveillance de l'accu, etc. de façon autonome et décharge ainsi le contrôleur du module d'extension).

fonction qui s'assure que le texte n'est envoyé qu'une seule fois à l'écran, sinon le texte à l'écran scintillerait. Lorsque le comportement « Cruise » est actif, les 4 LED rouges de statut réalisent un message défilant.

Le programme surveille également l'état de l'accu. Lorsque le niveau de l'accu est faible, le robot est arrêté. Toutefois, cela prend du temps avant qu'un accu récemment chargé n'atteigne ce niveau…

De plus, le programme attend trois bruits forts pour démarrer (un message WAIT s'affiche sur la deuxième ligne de l'écran, suivi d'un compteur de ces sons) – par exemple, en tapant trois fois des mains. Vous pouvez aussi appuyer sur un bouton du RP6-M32 comme solution alternative. Cela s'effectue par la mise en œuvre d'un comportement supplémentaire.

Nous avons atteint la fin de ce petit mode d'emploi additionnel. Vous pouvez désormais agir selon votre propre créativité et écrire de nouveaux programmes ou placer de nouveaux capteurs sur le RP6, contrôlés par le RP6-M32 ou mettre d'autres choses en place.

gestionnaires d'événements pour ACS, IRCOMM et Bumper sont également disponibles ici, mais il en existe de nouveaux pour niveau d'accu faible et les requêtes Watchdog. Le prochain exemple montre comment contrôler les mouvements du robot en utilisant ces fonctions.

Le programme est similaire à l'exemple 7. Comme nous l'avons déjà dit, nous avons ajouté une nouvelle fonction Watchdog Timer, dont les requêtes sont affichées sur le LCD. Le programme lit également tous les registres de capteur et affiche les résultats sur l'interface séquentielle, tout comme les événements ACS, Bumper et RC5.

```
Example 9: I²C Bus Interface - controlling the robot's movements
Directory: <RP6Examples>\RP6ControlExamples\Example_09_Move\
Source file: RP6Control_09_Move.c

This program demonstrates how to use the I²C Bus in master mode. The I2C-Slave
example program must be loaded to controller on the mainboard before you can
run this example!

ATTENTION: The Robot will move in this example program!
```

Là, nous ajoutons à la nouvelle Library quelques fonctions de mouvements déjà présentes dans le RP6Lib : move (bouger), rotate (tourne), moveAtSpeed, changeDirection (changement de direction) et stop. Ces fonctions sont identiques à celles utilisées dans le RP6Lib. Pour cet exemple, nous avons enlevé depuis l'affichage jusqu'au Watchdog des exemples précédents, afin d'utiliser le mode blocage des fonctions de mouvements (de toute façon, certaines choses telles que l'affichage Heartbeat n'aurait pas fonctionné). Le robot s'agite et se tourne sur environ 180° - exactement comme pour l'exemple 7 du RP6Lib (« RP6Base_Move_02.c »).

```
Example 10: I²C Bus Interface - Verhaltensbasierter Roboter
Directory: <RP6Examples>\RP6ControlExamples\Example_10_Move2\
Source file: RP6Control_10_Move2.c

This program demonstrates how to use the I²C Bus in master mode. The I2C-Slave
example program must be loaded to controller on the mainboard before you can
run this example!

ATTENTION: The Robot will move in this example program!
```

Ce nouvelle Library vous permet de copier pratiquement l'intégralité des programmes d'exemples pour votre robot à commande. C'est exactement ce qui a été fait ici avec le programme d'exemple «RP6Base_05_Move_05. Quelques modifications étaient nécessaires – les LED de la carte mère devaient être, entre autres, contrôlées par la fonction setRP6LEDs, puisque setLEDs est déjà réservé pour les LED du RP6-M32…

Pour le reste, le programme est à peu de choses près identique aux programmes que vous connaissez déjà – le robot tourne tout en essayant d'éviter les obstacles. Sauf que cette fois le robot est contrôlé par le RP6-M32.

L'écran LC et les LED affichent désormais le comportement actif actuel. Ainsi, vous pouvez voir en temps réel quel comportement est actif. Pour cela, il existe une petite

- **Capteur microphone**
  - o Afin de détecter les bruits, par exemple, taper dans les mains, etc.

- **Bipeur piezo**
  - o Pour créer des sonneries simples
  - o Le signaleur, par exemple pour indiquer une erreur ou un changement de statut

- **4 LED de statut**

- **5 boutons**

- **Port écran LC**
  - o Pour la connexion d'un écran LC 16 x 2 lignes, mais d'autres sont également compatibles (par exemple le 16 x 4 lignes). L'écran se relie par deux boulons d'écartement et peuvent dépasser d'un côté… Nous vous conseillons toutefois de prendre les dimensions avant l'achat et de commander le matériel de montage adapté ! De plus, pour les formats autre que 16 x 2 lignes, il est possible que vous ayez quelques petites modifications à effectuer dans la bibliothèque, afin que tout fonctionne correctement (principalement pour l'initialisation de l'écran et éventuellement le curseur de positionnement). Les programmes d'exemple sont conçus pour un écran 16 x 2 lignes, mais vous pouvez facilement le changer.
  - o L'écran peut afficher, par exemple, des messages textes, des menus, des statuts de programme ou encore des valeurs de capteur.

- **14 ports I/O disponibles pour contrôle votre circuit et vos capteurs**
  - o 6 d'entre eux peuvent être utilisés comme canaux convertisseurs analogique/numérique (ADC).

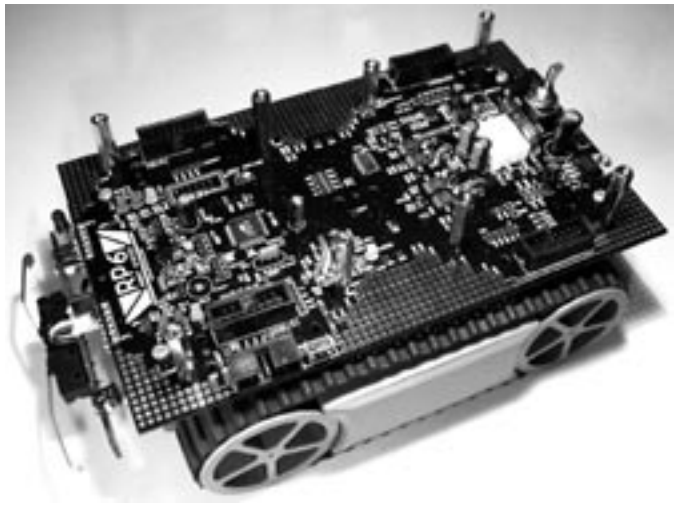
- **Jusqu'à 3 interrupteurs externes sont disponibles pour la connexion XBUS.**

- **Branchement pour l'interface PC USB, disponible pour le téléchargement de programme**
  - o Le téléchargement de programme travaille aussi rapidement et facilement que le robot seul, grâce à une interface USB et le logiciel pratique RP6Loader.

De plus, comme avec le robot, nous fournissons quelques programmes C d'exemple, comme la bibliothèque fonctionnelle, qui facilite grandement la programmation.

Nous avons prévu de rendre disponible plus de programmes et de mises à jour du robot et du module d'extension sur la page web du robot. Vous pouvez aussi bien évidemment échanger votre propre programme avec d'autres utilisateurs ! La RP6ControlLibray et les programmes d'exemple sont également disponibles sur le GPL Licence Open Source !

2. Montage du module d'extension



La façon dont vous allez montez le module sur le robot dépend des autres modules d'extension, que vous pourriez déjà avoir installé sur le robot.

Afin de monter le module sur le robot, vous devez d'abord dévisser 4 vis de la carte mère. Vous pouvez également, si vous le souhaitez, détacher prudemment un petit connecteur de la platine bumper, afin de soulever entièrement la carte mère. Cependant, cela n'est pas nécessaire, si vous vous servez de vos doigts pour atteindre ce qu'il y a en dessous de la carte mère et pour visser les boulons d'écartement avec les écrous M3.

Attention : Lorsque vous rebranchez le câble de la platine bumper, maintenez le platine du capteur par derrière avec un doigt afin d'éviter qu'elle ne soit poussée trop fortement vers l'arrière ! Autrement, vous pouvez également dévissez les 2 vis de la platine bumper et faire passer le câble…

Ce programme d'exemple permet d'effectuer ceci, en affichant l'état actuel des ACS sur les 4 LED, l'écran, l'interface séquentielle et le bipeur duRP6 CONTROL et en le rendant audible.

L'énergie d'émission du ACS est encore réglée sur un bus I²C au démarrage du programme. En plus du ACS, le programme réagit aussi au bumper et aux éventuels transferts RC5 émis depuis une radio commande ou un autre robot.

Vous pouvez vous en servir de la même façon que tous les autres capteurs du robot. Egalement pour de futurs modules d'extension avec des capteurs supplémentaires.

Un autre aspect du programme est l'affichage « Heartbeat », c'est-à-dire « battements de cœur ». La fonction task8LCDHeartbeat() permet de faire clignoter de manière continue le caractère '*' sur le LCD, avec une fréquence de 1 Hz. Cela est très utile afin de constater si le programme a entièrement planté ou si il s'agit juste d'une partie précise qui présente une erreur. Lorsque vous écrivez votre propre programme et qu'il semble s'être entièrement planté, il peut être très utile pour délimiter une source d'erreurs. Le fait que le programme plante peut mal passer pendant le développement.

C'est pour cela que le programme d'exemple pour le Slave I²C du contrôleur de la carte mère intègre aussi une fonction « Software watchdog ». Le master ne réagit pas dans un laps de temps déterminé (dans la mesure où le registre 0 est lu), tous les systèmes de l'unité de base du robot sont éteints et le programme stoppé. Il faut avant tout éteindre les moteurs ! Imaginez que le slave reçoive la commande d'avancer à une vitesse de 10cm/s et que le master plante ! Cela provoquerait un crash du robot contre le prochain obstacle rencontré et ce, à haute vitesse…

Désactivez tout d'abord le logiciel Watchdog Timer. Afin d'activer le Watchdog, vous devez préalablement envoyer une commande sur le bus I²C. Il est également possible de configurer le Watchdog, de manière à ce que l'interruption se déclenche afin de vérifier que le contrôleur master y réagit encore. C'est ce que nous allons utiliser dans l'exemple suivant.

```
Example 8: I²C Bus Interface - reduced RP6 Library
Directory: <RP6Examples>\RP6ControlExamples\Example_08_I2CMaster\
Source file: RP6Control_08_I2CMaster.c

This program demonstrates how to use the I²C Bus in master mode. The I2C-Slave
example program must be loaded to controller on the mainboard before you can
run this example!
```

Le programme peut être rapidement embrouillé, lorsqu'on ajoute plusieurs fonctions dans un seul fichier. C'est pour cela que l'exemple 7 est partagé entre deux fichiers C et d'y ajouter quelques fonctionnalités. C'est une sorte de RP6Lib réduite, pour le contrôle du robot via les bus IC et qui vous permet de l'utiliser de façon identique que la bibliothèque du RP6Lib normal pour le contrôleur de la carte mère. De nombreuses fonctions et variables sont nommées de la même façon que dans le RP6Lib. Cela facilite de la réutilisation de parties de programme du RP6Lib avec le RP6ControlLib. Les

appuyant sur l'une des 5 touches. La touche T1 rehausse d'un un compteur et envoie ensuite les commandes setLED avec cette valeur de compteur au Slave. Cette procédure affiche un compteur binaire avec les 6 LED de statut de la carte mère.

En appuyant sur la touche T2, tous les registres sont lus et transmis sur l'interface séquentielle. Par contre, uniquement les valeurs des capteurs de lumières sont lues et affichées sur l'écran LC.

Avec T4 et T5 est envoyée une commande « Rotate » et le robot effectue une petite rotation vers la droite ou la gauche (vous pouvez appuyer plusieurs fois sur les touches afin de continuer à faire tourner le robot…).

Le programme peut bien évidemment être modifié – tout comme les autres – et est parfaitement adapté pour tester votre nouvel appareil I²C ou les fonctions que vous avez ajoutées sur votre programme d'exemple.

```
Example 7: I²C Bus Interface - Master Modus - react on external Interrupts
Directory: <RP6Examples>\RP6ControlExamples\Example_07_I2CMaster\
Source file: RP6Control_07_I2CMaster.c

This program demonstrates how to use the I²C Bus in master mode. The I2C-Slave
example program must be loaded to controller on the mainboard before you can
run this example!
```

Peut-être avez-vous déjà remarqué le signal d'interruption sur le connecteur XBUS du RP6 ? Vous pouvez l'utiliser pour réagir sur les modifications de capteur, sans consulter constamment le Slave. Chaque accès aux Bus vous fait perdre du temps.

Un bon exemple pour l'utilisation du signal d'interruption est l'ACS du robot. Le statut du capteur ne change relativement que très rarement et il ne serait pas très efficace de consulter le bus pour vérifier si quelque chose a changé. Dès que le statut ACS change ; le programme Slave passe le signal INT1 au plus haut niveau. Puisque INT1 est relié à l'entrée d'interruption 0 du MEGA32 du RP6 CONTROL M32, le contrôleur peut directement réagir à cet événement et vérifier le statut du contrôleur de la carte mère.

Toutefois, nous n'avons pas besoin d'un sous-programme d'interruption dans le programme d'exemple, pour réagir à cet événement, mais pour consulter le statut des pins à chaque exécution de la boucle principale. Puisque les transferts Bus I2C sont interrompus et il ne peut y avoir lieu de nouvelle transmission pendant un sous-programme d'interruption.

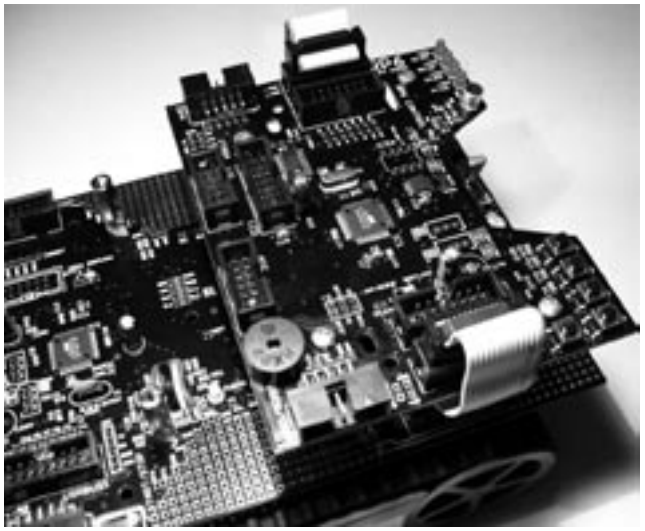
La fonction task_I2CTWI() doit toujours être appelée dans la boucle principale, puisqu'elle règle le déroulement des transferts I2C. Ainsi, il n'y a aucun avantage à utiliser un sous-programme d'interruption, cela serait même un problème, car les transferts bus I2C déjà en cours pourraient être interrompus. La fonction task_checkINT0() sera donc la seule à être utilisée pour réévaluer le signal d'interruption et effectuer une consultation le cas échéant. Dès que le registre de statut 0 du slave est lu, le signal d'interruption est réinitialisé. Vous pouvez trouver dans les trois premiers registres du slave, ce qui a déclenché l'interruption.

Vous pouvez ensuite vissez les 4 boulons d'écartement 25mm M3 avec les écrous M3 dans les trous de fixation de la carte mère, comme représenté sur la photo.

Sur l'image au dessus, les 8 boulons d'écartement sont vissés, et également ceux du circuit imprimé du module d'extension !

Installez le module d'extension sur les boulons d'écartement et fixez le avec les 4 vis M3.

Enfin, connectez les deux câbles ruban – c'est fini.



Nous vous recommandons de monter le RP6 CONTROL M32 à l'arrière du monceau d'extension du robot – comme module principal, afin que les touches et l'écran restent accessibles. Les deux branchements de programmation sont alors aussi accessibles depuis la même face du robot. Vous pouvez fixer à l'avant du robot les modules d'expérimentation fournis avec le robot.

Si vous avez acheté l'écran LC 16x2 lignes, vous devez d'abord le brancher et le monter au module d'extension avant de l'installer sur le robot.

Le câble ruban à 14 pôles est très flexible et peut être plié sans problème. Afin que le câble passe correctement sous l'écran, vous devez le plier comme indiqué sur la photo sur le module d'extension RP6 CONTROL M32.

L'écran peut alors être fixé au module d'extension grâce à, par exemple, des boulons d'écartement de 20 ou 25 mm, des écrous adaptés et de vis.

Vous pouvez également utiliser un autre écran LC texte compatible avec le contrôleur HD44780. Vous devez simplement souder un câble à l'écran. Veillez absolument à la bonne affectation des broches !

4 boulons d'écartement ne sont pas forcément nécessaires, comme sur la photo !
2 suffisent amplement (tous les deux devant ou à l'arrière) pour fixer l'écran.

Astuce : Il n'y a pas de boulon d'écartement fourni pour le montage de l'écran, mais pour chaque module d'extension, 4 sont fournis, ainsi que des écrous et des vis. Généralement, il faut fixer le module d'extension avec 4 boulons d'écartement, comme représenté sur la photo. Mais 3 suffisent – deux devant et un à l'arrière, au centre ! Ainsi, avec le module d'expérimentation et de le RP6-M32, vous disposeriez de 2 boulons d'écartement de 25 mm, de vis et d'écrous pour l'écran…

Une fois fixé sur le robot, cela peut ressembler aux montages suivants :


```
Example 5: Analog/Digital Converter (ADC) and I/O Ports
Directory: <RP6Examples>\RP6ControlExamples\Example_05_IO_ADC\
Source file: RP6Control_05_IO_ADC.c

The program will output messages to the serial interface and the LC-Display!

The Robot will not move in this example program!
```

Même si cela fonctionne de la même façon que sur le robot, nous allons vous montrer dans, cet exemple, comment utiliser les ADC libres et les I/O. Ils sont fréquemment utilisés pour ce module, ainsi un exemple entièrement dédier sur ce sujet peut s'avérer très utile pour les débutants.

Le programme utilise les appellations suivantes pour les ADC et les I/O.

```
ADC7    (1 << PINA7) // ADC Channel #7  - may also be used as a standard I/O Pin
ADC6    (1 << PINA6) // Channel # 6 ...
ADC5    (1 << PINA5)
ADC4    (1 << PINA4)
ADC3    (1 << PINA3)
ADC2    (1 << PINA2) // Channel # 2, Channels #0 and #1 have been reserved for keys and microphone.

IO_PC7  (1 << PINC7) // I/O-Pin 7 at 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 at PORTC
IO_PD6  (1 << PIND6) // I/O Pin #6 at PORTD
IO_PD5  (1 << PIND5) // I/O Pin #5 at PORTD
```

(cf. le dossier RP6Control.h)

Le programme ne fait rien de vraiment intéressant – vous n'avez donc pas nécessairement besoin de le tester. Il n'y a rien de brancher sur les pins I/O, il est donc sage de débuter lorsque votre matériel sera connecté aux I/O et écrire ainsi vos propres programmes de commande.

```
Example 6: I²C Bus Interface - Master Modus
Directory: <RP6Examples>\RP6ControlExamples\Example_06_I2CMaster\
Source file: RP6Control_06_I2CMaster.c

This program demonstrates how to use the I²C Bus in master mode. The I2C-Slave
example program must be loaded to controller on the mainboard before you can
run this example!
```

Ce programme vous montre comment commander le contrôleur sur la carte mère en mode Slave. Cela ne fonctionne bien évidemment, que lorsque le programme d'exemple I²C Slave des exemples du RP6Base est chargé dans le contrôleur sur la carte mère.

L'accès aux bus I²C s'effectue pratiquement de la même manière comme sur le contrôleur de la carte mère – il s'agit des mêmes fonctions.

Dans ce programme d'exemple, vous pouvez envoyer différentes commandes au contrôleur sur la carte mère, programmé avec le programme d'exemple Slave, en

Ce programme montre l'utilisation des 5 touches sur le RP6 CONTROL. A chaque pression de touches, le programme affiche à l'écran le numéro de touche et une gamme est jouée par le piezo.

(Attention : appuyer sur la touche 4 risque de vous mettre sur les nerfs ;-)).

```
Example 3: Microphone sensor
Directory: <RP6Examples>\RP6ControlExamples\Example_03_Microphone\
Source file: RP6Control_Microphone.c

The program will output messages to the serial interface and the LC-Display!

The Robot will not move in this example program!
```

Le capteur microphone peut être utilisé pour la détection de bruits forts. Ce programme représente les volumes de bruit mesurés sous forme d'un graphique en barres, aussi bien sur le LCD qu'avec les LED. Tapotez du doigt sur le microphone pour vérifier que cela fonctionne correctement. Tapez également des mains et observez le résultat sur l'écran et les LED.

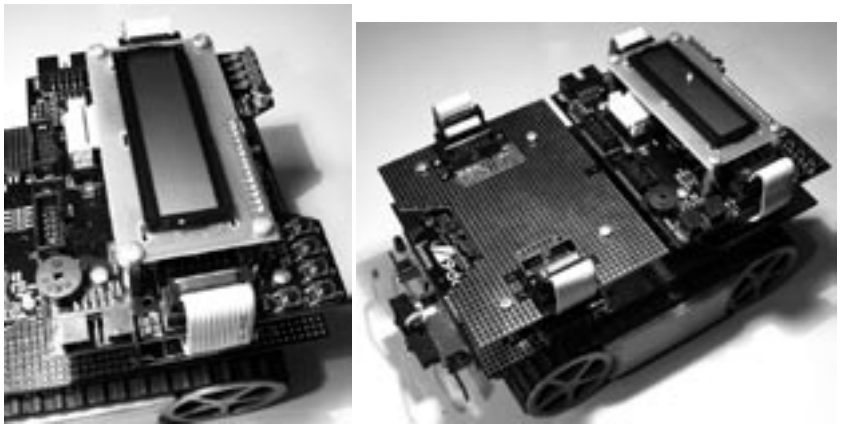
```
Example 4: External EEPROM
Directory: <RP6Examples>\RP6ControlExamples\Example_04_EEPROM\
Source file: RP6Control_04_EEPROM.c

The program will output messages to the serial interface and the LC-Display!

The Robot will not move in this example program!
```

Ce programme d'exemple montre comment lire et écrire sur l'EEPROM externe. Dans un premier temps, le programme lit et affiche les deux premières « pages », chacune d'une longueur de 64 byte. Afin de vérifier que l'EEPROM conserve bien son contenu lorsqu'il est éteint, éteignez puis rallumez le robot après l'exécution du programme ! Les dernières données écrites sont conservées dans l'EEPROM et sont montrées au redémarrage.

Le programme réinscrit ensuite la première page avec 64 bytes et relis les 128 premiers bytes afin de vérifier que les bytes n'ont été écrits que dans la première page et que le reste de l'EEPROM n'a pas été modifié (ce qui modifierait le contenu de ces cellules mémoires à 255).

Des bytes séparés peuvent également être écrits et lus. La valeur 128 est écrite sur les cellules mémoire avec l'adresse 4, puis le programme lit les deux premières pages – en utilisant cette fois-ci une lecture byte par byte – ce qui est, bien évidemment, beaucoup plus lent que de lire entièrement la page.

Quand tout est terminé, un petit message va à nouveau défiler.



Les deux connecteurs d'extension à 10 pôles avec les ports I/O libres peuvent être simplement reliés au module d'extension grâce à un petit ruban câble à 10 pôles, et les I/O et les ADC peuvent être alors utilisés comme capteurs, par exemple. Cela fonctionne également pour les modules montés sur d'autres niveaux – les câbles ruban devant normalement passer dans l'espace entre les deux monceaux de module.

Maintenant vous pouvez effectuer un petit test des fonctions :

Reliez l'interface USB au connecteur PROG/UART sur le RP6-M32 avec le câble ruban et démarrez le RP6Loader. Ensuite, mettez en route le robot – un message texte doit apparaître sur l'écran et une des LED doit clignoter. Si cela fonctionne, cliquez sur connecter dans le RP6Loader – un message de confirmation « Connected to RP6 Control » s'affiche. Si cela fonctionne également, finissez de lire intégralement le mode d'emploi et démarrez avec les programmes d'exemple !

Si l'écran n'affiche aucun texte ou juste une ou deux rangées de carrés grisés mais que la LED clignote et que la connexion avec le RP6Loader est effective, vous devez régler le contraste de l'écran (ou bien vous avez utilisé un autre type d'écran et l'affectation des broches est erronée…). Ajustez le contraste avec le potentiomètre R16 sur la platine. Il peut être réglé grâce à un petit tournevis plat. Vous pouvez également utiliser un petit tournevis cruciforme, mais vous rencontrerez des difficultés à le positionner par rapport au potentiomètre.

Afin d'atteindre plus facilement le potentiomètre, vous pouvez dévisser l'écran (ou ne pas le fixer du tout tant qu'il n'est pas correctement réglé). La fiche doit cependant restée branchée ! Cela vous permet d'ajuster le potentiomètre alors que le robot est allumé.

Ou bien vous fixez tout simplement l'écran avec un petit tournevis comme représenté sur la photo ci-dessus. Veillez à ne surtout pas toucher des composants ou à entrer en contact avec la platine avec le tournevis ! Nous vous conseillons d'éteindre le robot d'abord, puis d'installer le tournevis dans la bonne position, de rallumer le robot et d'ajuster le contraste…

4. Programmes d'exemple

Vous trouverez sur le CD quelques programmes d'exemple. Ces programmes d'exemple démontrent les fonctions de base du RP6 CONTROL M32. Comme pour un robot, ils ne doivent pas être considérés comme des solutions optimales mais comme points de départ pour votre propre programme. C'est intentionnel de notre part, afin qu'il vous reste quelque chose à faire – il serait en effet ennuyeux de tester seulement des programmes préfabriqués, n'est-ce pas ?

Vous pouvez bien évidemment échanger votre programme avec d'autres utilisateurs sur internet. Le RP6ControlLib et tous les programmes d'exemple sont inscrits sous la licence Open Source « GPL » (General Public License) et vous êtes autorisés à modifier les programmes et à les rendre disponibles pour d'autres utilisateurs, selon des conditions GPL.

Un grand nombre de programmes d'exemple pour le MEGA32 sont disponibles sur Internet, car la famille de contrôleur AVR est l'une des plus populaires parmi les usagers de ce loisir. Toutefois, vous devrez faire attention à adapter ces exemples au RP6ControlLib et en fonction du matériel du RP6 CONTROL – sinon cela risque souvent de ne pas fonctionner (les problèmes récurrents sont, entre autres, l'affectation des pins, l'utilisation de modules déjà utilisés comme les minuteurs, d'autres cadences d'horloge, etc.) !

```
Example 1: "Hello World"-Program with LCD text output and LED running light
Directory: <RP6Examples>\RP6ControlExamples\Example_01_LCD\
Source file: RP6Control_LCD.c

The program will output messages through the serial interface and on the LC-
Display, you should connect the robot to your PC and observe the messages ap-
pearing in the terminal of the RP6Loader Software! Optionally you may also con-
nect the LC-Display.

The Robot will not move in this example program - as long as you only load the
I²C-Bus Slave Program into the controller on the mainboard! Therefore you are
allowed to place the robot on top of a table near your PC.
```

Ce programme d'exemple transmet un petit texte « Hello World » sur l'interface séquentielle et le message défile ensuite (running light). De plus, un texte statique sera affiché sur le LCD puis un texte animé « Hello World » - les deux mots « HELLO » et « WORLD » se balancent lentement. Après environ 16 secondes, une petite pause est marquée, signalée par deux petits bips. Après 8 secondes continue – également signalé par deux petits bips.

```
Example 2: Buttons and Sound
Directory: <RP6Examples>\RP6ControlExamples\Example_02_Buttons\
Source file: RP6Control_Buttons.c

The program will output messages to the serial interface and the LC-Display!

The Robot will not move in this example program!
```

Vous pouvez ensuite commuter la sortie sur le niveau high (haut) ou low (bas) dans le registre PORTx.

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

Si, dans le registre DDRx, le bit est à 0, le pin correspondant est alors configuré pour l'entrée.

```
Exemple:
DDRC &= ~IO_PC6;  // PC6 is set to input-mode
```

Vous pouvez alors lire dans le registre PINx le statut du pin et déterminer si vous souhaitez le régler sur le niveau high ou low.

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

Vous pouvez encore activer les résistances pull up intégrées au micro contrôleur, en réglant les bits dans le registre PORTx. Cela est intéressant, parx exemple, pour des palpeurs ou des capteurs similaires.

Les pins I/O suivants sont disponibles sur le RP6 CONTROL M32 (les définitions exactes figurent dans le fichier Header RP6Control.h) :

```
IO_PC7
IO_PC6
IO_PC5
IO_PC4
IO_PC3
IO_PC2

IO_PD6
IO_PD5
```

Vous pouvez également utiliser les canaux ADC comme pins I/O ! Veillez aux noms utilisés différents des appellations ci-dessus (ADC_7 / ADC7)

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

Remarque importante : Les pins I/O séparés sont prévus pour un courant maximum de 20mA. Au total, un port 8 bit ne doit être chargé à plus de 100 mA ! Si vous souhaitez utiliser plus de courant, vous devez utiliser un transistor externe !

Pour des informations plus précises, veuillez vous référencer à la fiche de données du MEGA32, figurant sur le CD-ROM !

3. Documentation RP6 CONTROL

Tout comme pour le robot en lui-même le RP6 CONTROL M32 dispose également d'une bibliothèque fonctionnelle comportant de nombreuses fonctions très utiles pour les débutants. La bibliothèque s'intitule RP6ControlLibrary ou RP6ControlLib en raccourci. Les fonctions stop-watch, delay, UART et bus I²C sont identiques à la RP6Library normale. Les fichiers utilisés pour UART et bus I²C sont d'ailleurs les mêmes, et sont localisés dans le sous-dossier RP6common. Nous ne décrirons donc pas à nouveau ces deux fonctions – veuillez vous référer à nouveau au chapitre concerné du mode d'emploi du RP6 et aux programmes d'exemple !
Nous allons vous décrire ici les fonctions spécialement dédiées au RP6Control ou qui ne fonctionnent pas dans le RP6Lib.

Malgré un grand nombre de fonctions préétablies, le RP6ControlLib n'est qu'une base ! Cette bibliothèque est de loin parfaite et vous pouvez nettement l'améliorer et la compléter ! Nous faisons ici appel à vos capacités de programmation !

3.1. Initialiser le micro contrôleur

```
void initRP6Control(void)
```

Comme déjà indiqué dans le RP6Lib, votre programme doit TOUJOURS appeler cette fonction en premier ! Pour le RP6ControlLib, seul le nom change…

Cette fonction initialise les modules hardware du micro contrôleur du RP6-M32. Le micro contrôleur ne fonctionnera correctement que si vous appelez cette fonction en premier ! Le bootloader peut en initialiser une partie, mais tout ne sera pas effectué.

Exemple :

```
1  #include "RP6ControlLib.h"
2
3  int main(void)
4  {
5   initRP6Control(); // Initialisation - ALWAYS call this function FIRST!
6
7  // [...] Program code...
8
9        while(true);      // Infinite loop
10       return 0;
11 }
```

Chaque programme du RP6 CONTROL M32 doit au minimum ressembler à ça ! Le « endless loop » (« boucle infinie ») ligne 9 est nécessaire pour garantir une fin définie du programme ! Sans cette boucle, le programme pourrait fonctionner différemment de ce qui est attendu ! Tout comme le contrôleur sur la carte mère !

3.2. Statuts LED

Les LED fonctionnent de la même façon que sur la carte mère du robot. Dans tous les cas il n'y a que 4 LED de disponible et les appellations seront différentes car ces LED sont connectées à un registre à glissement externe, également utilisé pour l'écran LC. Ce registre à glissement est aussi désigné comme « External Port ».

Le RP6-M32 dispose d'une fonction « setLEDs » :

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

Exemple :

```
setLEDs(0b0000); // This command will turn off all LEDs.
setLEDs(0b0001); // turn on LED1 only
setLEDs(0b0010); // LED2
setLEDs(0b0100); // LED3
setLEDs(0b1010); // both LED4 and LED2
```

Voici une autre alternative possible :

```
externalPort.LED1 = true;  // activate LED1 in "External Port" Register
externalPort.LED2 = false;  // deactivate LED2 in "External Port" Register
outputExt();               // Commit the settings!

// outputExt() will send the contents of the externalPort variable
// to the shift register - in analogy to updateLEDs() in the RP6Lib.
// However this function will also update the LCD data lines.
```

3.3. Touches

Contrairement au RP6, 5 touches du RP6-M32 ont été connectées à un canal ADC, ce qui a pour avantage de n'utiliser qu'une broche pour les 5. L'inconvénient est, par contre, que ce circuit simplifié utilise 5 résistances identiques (cf. plan du circuit sur le CD) ne permettant pas d'appuyer sur plus d'une touche à la fois. Cela suffit amplement pour les touches de commande !

```
uint8_t getPressedKeyNumber(void)
```

Cette fonction détermine quelle touche est pressée. L'ADC est alors évaluée et comparée avec des valeurs seuil prédéfinies. Ces valeurs seuil peuvent être adaptées dans la Library afin que cela fonctionne correctement avec le RP6-M32. La valeur ADC mesurée peut dévier en raison de la tolérance de fabrication habituelle de la résistance. Les valeurs seuil se trouvent dans la fonction getPressedKeyNumber.

```
uint8_t checkPressedKeyEvent(void)
```

Cette fonction vérifie lorsqu'une touche est pressée et restitue une fois le numéro de la touche – contrairement au getPressedKeyNumber, où les numéros des touches sont restitués tant que la touche est pressée. Cette fonction est très utile pour l'évaluation d'une valeur de touche dans la boucle principale sans interruption du flux du programme. Il en va de même pour cette fonction :

```
uint8_t checkReleasedKeyEvent(void)
```


L'EEPROM nécessite 5ms pour écrire les données, temps pendant lequel vous ne pouvez pas accéder à l'EEPROM. Afin d'évaluer le statut actuel, vous pouvez utiliser cette fonction :

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

avec par exemple

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

vous pouvez vérifier si l'EEPROM n'est plus occupé par l'écriture de données. Les précédentes fonctions ci-dessus le font également, donc vous n'avez besoin de cette fonctionnalité que si vous avez besoin d'effectuer d'autres choses dans ce laps de temps.

3.8. ADC

Les canaux ADC peuvent être lus avec la fonction suivante, similaire à celle du RP6Lib :

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

Il n'y a cependant pas de variante automatique permettant de lire séquentiellement les canaux ADC en arrière plan pour le RP6-M32.

Bien évidemment, les canaux sont intitulés différemment que ceux du RP6Lib :

```
ADC_7       --> ADC Channel 7 - available on the 10-pin connector "ADC"!
ADC_6       --> ADC Channel 6 ...
ADC_5
ADC_4
ADC_3
ADC_2       --> ADC Channel 2
ADC_KEYPAD  --> This channel is connected to the keypad
ADC_MIC     --> This channel serves the microphone.
```

3.9. Ports I/O

Le RP6 CONTROL fournit le système avec 14 ports I/O disponibles et nous allons vous décrire brièvement comment accéder de manière générale aux ports I/O de l'AVR.

Le ATMEGA32 possède 4 ports I/O. Chacun de ses ports va jusque 8 bit et est contrôlé par 3 registres : un registre contrôle la « direction » des pins I/O (DDRx), c'est-à-dire savoir si un pin est utilisé comme entrée ou sortie ; un registre pour écrire (PORTx) et un registre pour lire (PINx).

Si vous souhaitez utiliser un pin I/O comme sortie, par exemple pour allumer une LED, vous devez d'abord positionner le bit correspondant sur 1 dans le registre DDRx.

```
Exemple:
DDRC |= IO_PC7; // PC7 is set to output-mode
DDRC = IO_PC7 | IO_PC6 | IO_PC5; // PC5, PC6, PC7 are set to output-mode,
                                 // all other pins are set to input-mode!
```

```
writeWordSPI(uint16_t data)
```

Transfère deux bytes de données, qui seront transmis dans une variable 16 bits sur le bus SPI, en même temps que le high byte.

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

Transfère jusqu'à 255 bytes sur un bus SPI d'un grand circuit prétraité. Le nombre de bytes à transférer de ce circuit « buffer » est à mentionner avec le paramètre « length ».

```
uint8_t readSPI(void)
```

Lit un byte de donnée du bus SPI.

```
uint16_t readWordSPI(void)
```

Lit deux bytes du bus SPI et les restitue en tant que variable 16 bits. Le byte lu en priorité est le High Byte.

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

Lit jusqu'à 255 bytes d'un bus SPI d'un grand circuit prétraité.
Vous n'avez normalement pas besoin des fonctions SPI. Celles-ci seront toutefois utilisées dans les prochaines fonctions que nous allons décrire, afin d'avoir accès à l'EEPROM connecté sur le bus SPI.

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

Lit un byte unique à l'adresse « memAddr » depuis l'EEPROM. L'adresse peut se trouver dans le domaine de 0 à 32767 de notre grand EEPROM 32 Ko.

Exemple :

```
// The next program line will read a byte from EEPROM address 13860:
uint8_t data = SPI_EEPROM_readByte(13860);
```

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

Lit jusqu'à 255 bytes (length) débutant à l'adresse « startAdr » dans un réseau prédéfini (buffer).

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

Sauvegarde un byte (donnée) à une adresse précise (memAddr) de l'EEPROM.

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

sauvegarde jusqu'à 64 bytes (lenght) dans le réseau « buffer » commençant par « startAddr » de l'EEPROM.

Veuillez respecter la limite de 64 bytes maximums inscriptibles en une seule fois. 64 bytes correspondent à la taille de la page (Page size) de l'EEPROM et rien de plus ne peut être mis en mémoire tampon avant que les données ne soient écrites. De plus, les données qui doivent être gravées les unes après les autres doivent tenir dans une page, c'est-à-dire entre 0 et 63, 64 et 127, 128 et 191… ! En cas de dépassement de la taille de la page, l'EEPROM réécrit les données depuis le début de la page. Bien évidemment, vous avez la possibilité de commencer à écrire par exemple à partir de l'adresse 50, mais si vous souhaitez graver plus de 14 Bytes, le compteur d'adresse recommencera à zéro et gravera par-dessus les données existantes dans la mémoire tampon.

À la lecture des données de l'EEPROM, il n'est pas nécessaire de veiller à la taille de la page et vous pouvez, en théorie, lire le contenu entier de l'EEPROM en une seule fois.

Dans le cas présent, la valeur de touche n'est restituée uniquement lorsque la touche a été pressée puis relâchée. Cette fonction aussi ne bloque pas le flux de programme – vous n'aurez pas à attendre une boucle pour que la touche soit relâchée.

3.4. Bipeur

Le signaleur du RP6-M32 peut être contrôlé grâce à cette fonction :

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

Cette une fonction non bloquante – Elle donne la hauteur et le laps de temps pour un son généré et ensuite abandonné. Le bipeur se coupe automatiquement après un laps de temps donné. Chaque appel supplémentaire de cette fonction écrase les réglages. Pour jouer des mélodies ou des tonalités, utilisez le macro suivant :

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

Les hauteurs de sons, les laps de temps et les pauses après le son peuvent être ici configurés. Le macro utilise mSleep pour générer les pauses – ainsi cela dure time + delay millisecondes jusqu'à ce que le programme continue.

Si vous ne souhaitez pas régler de pause ou de durée de sons, vous pouvez régler uniquement la hauteur de sons avec ceci :

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

Très pratique pour générer un son continu (par exemple une sirène d'alarme). Attention : toutes les fonctions du bipeur n'acceptent que les valeurs de pitch dans une plage de 0 à 255, 255 représentant la fréquence maximale.

3.5. Capteur microphone

Le RP6 CONTROL ne produit pas uniquement des sons, mais y réagit également. Pas selon la hauteur de sons mais selon le volume. Vous pouvez par exemple faire partir votre robot en produisant un bruit fort.

Le circuit est conçu comme « peak-detektor ». Il sera mesuré dans une période variable l'amplitude du signal microphone et la valeur maximale sera conservée. Le micro contrôleur peut être mesuré avec un ADC de la valeur maximale, puis effacé. La valeur maximale sera rapidement sauvegardée dans un petit condensateur et afin « d'effacer » cette valeur maximale, il faudra décharger ce condensateur.
Avec la fonction suivant :

```
void dischargePeakDetector(void)
```

Vous devez d'abord décharger le condensateur. La fonction

```
uint16_t getMicrophonePeak(void)
```

Vous permet de déterminer la valeur maximale dans un intervalle précis. Après que la valeur soit mesurée, la fonction appelle directement dischargePeakDetector().
Vous pouvez voir comment cela fonctionne dans l'un des programmes d'exemple.

3.6. Écran LC

L'écran LC est idéal pour donner les valeurs du capteur et les messages de notifications, lorsque le robot n'est pas relié à un ordinateur. La sortie de l'écran LC fonctionne exactement comme sur une interface séquentielle – mais il y a bien évidemment quelques particularités. Regardez au maximum les programmes d'exemples, vous comprendrez plus facilement comment vous pouvez utiliser l'écran LC.

```
void initLCD(void)
```

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

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

Cette fonction (et write4BitLCDData) ne nécessite rien de particulier – nous ne faisons que décrire ici pour vous expliquer rapidement comment est commandé l'écran.

Le LCD doit fonctionner en mode 4 Bit, impliquant l'utilisation de 4 lignes de données et de 2 circuits de commande (Enable (EN) et Register Select (RS), Read/Write (R/W) est relié en permanence à la masse, le LCD ne servant qu'à l'écriture. La lecture n'est pas possible et pas nécessaire). Les 4 lignes de données sont branchées au registre de glissement comme les LED, afin de garder des ports libres. De façon analogue à la fonction setLED, setLCDD fixe les lignes de données du LCD. Toutefois, cette fonction fixe également le signal Enable, afin que le LCD prenne en charge les données.

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

Les bytes à transmettre sont partagé afin de transférer les ordres et les données 8 bit au LCD. La fonction write4BitLCDData prend en charge ce processus : les données 8 bit sont partagées en 4 « Nibbles » puis transférées.

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

Cette fonction appelle write4BitLCDData et fixe les lignes RS sur low afin d'émettre une commande au LCD.

```
void clearLCD(void)
```

Envoie au LCD la commande d'effacer son contenu.

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

Efface une plage particulière de l'écran. Les paramètres sont : ligne, position de départ de la ligne et longueur de la plage à effacer.

Exemple :

```
clearPosLCD(0,10,5); // deletes the trailing 5 chars in the first LCD line
```

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

Fixe le curseur texte à une position précise sur l'écran. Le paramètre « line » est 0 pour la ligne supérieur et 1 pour la ligne inférieure. Le paramètre « pos » doit être compris entre 0 et 15 pour les LCD 2 x 16.

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

Envoie un caractère à l'écran - cela fonctionne de la même manière que la fonction writeChar pour les interfaces séquentielles. Vous devez vous assurer que le curseur de l'écran se trouve à la bonne position. Sinon le module n'affichera pas votre message !

Exemple :

```
setCursorPosLCD(1,5);  // Set the cursor to the second line, character #5.
writeCharLCD('R');     // now display "RP6" starting
writeCharLCD('P');     // at the cursor's location!
writeCharLCD('6');
```

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

De la même manière que pour la fonction des interfaces séquentielles, writeStringLCD envoie une chaîne de caractères terminée par zéro du SRAM au LCD. Vous ne devez utiliser cette fonction que si le texte est réellement dans la RAM et non pas prédéfini. Le macro :

```
writeStringLCD_P(STRING)
```

Convient mieux, puisque du texte va directement être lu depuis la mémoire, sans un détour par la mémoire vive.

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

Grâce à cette fonction, vous pouvez afficher un texte d'une longueur précise sur le LCD. Les paramètres sont identiques dans la mesure où il s'agit de la même fonction pour les interfaces séquentielles.

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

Afin de simplifier l'émission du texte sur le LCD, vous pouvez écrire sur les deux lignes du LCD avec cette fonction et un appel.

Exemple

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

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

Qui est déjà une fonction connue pour les interfaces séquentielles pour afficher les calculs dans les formats BIN, OCT, DEC ou HEX sur le LCD.

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

Le writeIntegerLenghtLCD est identique, jusqu'au nom, à une fonction connue pour les interfaces séquentielles.

3.7. Bus SPI et EEPROM SPI

Sur le bus SPI (=Serial Peripheral Interface) sont reliés l'EEPROM et le registre à glissement 8 Bit. Un socle pour un autre AT25256 compatible EEPROM (par exemple ST M 95256 avec un boîtier DIP à 8 pôles) peut être soudé. Vous pourriez contrôler d'autres IC SPI, montés en cascade sur la carte mère avec le registre à glissement – nous vous conseillons cependant de préférer l'usage de bus I²C et de limiter ces méthodes alternatives pour des utilisations précises, dans lesquelles un bus I²C ne peut être utilisé ! Nous avons prévu le système avec les fonctions spécifiques pour accéder à l'EEPROM et le registre à glissement. Ainsi, nous vous déconseillons d'utiliser directement le bus SPI pour votre programme. Mais si vous avez besoin d'accéder au bus SPI, les fonctions suivantes vous seront utiles.

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

Transfère un byte de données sur un bus SPI.