Librairie C de Robopoly

Ce document présente un résumé des différentes fonctions présentes dans la librairie fournie par Robopoly ainsi que des exemples d'utilisation pour chacune d'elles.

Fonctions d'entrée/sortie:

digitalWrite(port,bit, value) Cette fonction permet de forcer la valeur d'une ligne d'entrée/sortie du cerveau à 1 ou à 0 et en sortie. On lui définit sur quel port on désire mettre la valeur (A, B, C ou D en fonction du port sur lequel on veut travailler), ensuite la ligne du port de 0 à 7 (qui correspond à la ligne du connecteur sur laquelle on va venir brancher quelque chose) et la valeur à mettre à savoir soit un état logique haut 1, soit un état logique bas 0.

Ex. :
```
int main(void)
{
        digitalWrite(C, 2, 1) ;  // Allume la LED du cerveau sur la ligne PORTC2.
        while(1) ;
        return 0 ;
}
```

On peut également écrire sur le port en entier, pour cela il suffit de donner le mot clé BYTE à l'endroit où on indique sur quel bit on veut travailler. La valeur à fournir est donc dès lors sur 8bit.

Ex. :
```
int main(void)
{
        digitalWrite(C, BYTE, 0xAA) ;    // Mets une ligne sur deux du PORTC à 1. (On aurait aussi bien pu mettre
                                         // 170 ou 0b10101010 à la place de 0xAA pour la
                                         // valeur au format décimal, binaire ou hexadécimal).
        while(1) ;
        return 0 ;
}
```

unsigned char result = digitalRead(port, bit) Cette fonction lit la valeur sur une ligne d'entrée/sortie du microcontrôleur. Il faut lui dire sur quel port on désire lire la valeur (A, B, C ou D) et sur quelle ligne du port (entre 0 et 7). Ensuite on récupère la valeur logique lue sur la broche du connecteur dans la variable result (un 1 ou un 0).

Ex. :
```
int main(void)
{
        unsigned char resultat ;
        while(1)
        {
                resultat=digitalRead(B, 0) ;    // Allume la LED du cerveau si on lit un 1 sur PORTB0.
                if (resultat == 1)
                {
                        digitalWrite(C, 2, 1) ;
                }
                else
                {
                        digitalWrite(C, 2, 0) ;
                }
        }
        return 0 ;
}
```

On peut comme pour la fonction digitalWrite, lire le contenu entier d'un port en remplaçant la valeur du bit par le mot clé BYTE.

unsigned char result = analogReadPortA(bit) Cette fonction effectue une mesure analogique sur l'une des lignes du PORTA. Il suffit de lui donner le numéro de la ligne du PORTA que l'on veut lire et elle retourne une valeur entre 0 et 255 dans la variable result. La règle de conversion est linéaire et fait correspondre la tension 0V avec la valeur 0 et la tension d'alimentation du cerveau avec la valeur 255.

Ex. :
```
int main(void)
{
    unsigned char resultat ;
    while(1)
    {
        // Allume la LED du cerveau si on lit une tension plus grande ou égale à Vcc/2.
        resultat = analogReadPortA(0) ;
        if (resultat >= 127)
        {
            digitalWrite(C, 2, 1) ;
        }
        else
        {
            digitalWrite(C, 2, 0) ;
        }
    }
    return 0 ;
}
```

Fonctions de timing:

waitms(unsigned int temps) Cette fonction fait attendre le microcontrôleur un temps que l'on peut choisir avant de passer à la ligne suivante. L'unité de temps est la milliseconde. La gamme de temps d'attente va de 0ms à 65'536ms. Attention le temps d'attente n'est qu'approximatif.

Ex. :
```
int main(void)
{
    while(1)
    {
        // Fait clignoter la LED du cerveau avec un intervalle d'une demi seconde.
        digitalWrite(C, 2, 1) ;
        waitms(500) ;
        digitalWrite(C, 2, 0) ;
        waitms(500) ;
    }
    return 0 ;
}
```

waitus(unsigned char temps) Cette fonction est similaire à waitms à la seule différence près que la base de temps n'est plus la milliseconde mais la microseconde. Le temps d'attente est de 0us à 255us environ.

Ex. :
```
{
    …
    waitus(40) ;
    …
}
```

Fonctions de gestion de moteur :

Moteur de traction :

setupMotorPWM(int vLeft, int vRight) Cette fonction définit la vitesse de rotation des moteurs gauche et droite. La vitesse est un nombre entier signé de -100 à +100. Cette valeur représente la vitesse en pourcent de la vitesse maximale des moteurs ; 0 le moteur est arrêté, +100 le moteur est à pleine vitesse dans le sens de rotation positif et -100 le moteur est à pleine vitesse

dans le sens de rotation négatif. Les moteurs doivent être branchés sur les pins PD4-7 selon le schéma de connexion du kit à disposition sur le site internet de Robopoly.

(Cette fonction utilise le module hardware TIMER1. Cette fonction n'utilise aucune fonction d'interruption. Pour pouvoir utiliser le TIMER1 pour une autre application, il suffit de le configurer comme souhaité et de ne jamais faire appel à la fonction *setupMotorPWM* par la suite.)

Ex. :
```
int main(void)
{
	while(1)
	{
		// Le robot avance à 80% de sa vitesse maximale pendant 1.5s puis recule
		// pour revenir à sa position initiale.
		setupMotorPWM(80, 80) ;
		waitms(1500) ;
		setupMotorPWM(-80,-80) ;
		waitms(1500) ;
	}
	return 0 ;
}
```

Servomoteur :

set_servo(unsigned char num_servo, char angle_servo) Cette fonction permet de commander séparément 10 servomoteurs branchés sur le cerveau selon le tableau de connexion ci-dessous. Le numéro du servomoteur doit être compris entre 0 et 9 et la valeur de consigne entre 0 et 100 correspondant à l'angle désiré en pourcent. Notons que 0 et 100 correspondent aux positions extrêmes gauche et droite du servomoteur alors que 50 devrait placer ce dernier vers le centre.

Cette fonction utilise le TIMER2. Pour utiliser librement le TIMER2 il faut désactiver les servomoteurs en ajoutant dans les options de compilation les informations suivantes (cf. fonctions de timing avancées) : -D noservo=definition

Ligne du PRisme	Servo #
PORTC3	0
PORTC4	1
PORTC5	2
PORTC6	3
PORTC7	4
PORTB0	5
PORTB1	6
PORTB2	7
PORTB3	8
PORTB4	9

Ex :
```
int main(void)
{
	set_servo(0, 50);		//place le servo 0 en position milieu
	set_servo(3, 0);		//place le servo 3 en position extrême gauche
	set_servo(7, 100);		//place le servo 7 en position extrême droite
	while(1) ;
	return 0 ;
}
```

Fonctions de communication :

uartSendByte(unsigned char valeur) Cette fonction envoie un byte par le bus de communication UART. On peut facilement récupérer des informations sur un PC via le programmateur USB. Pour cela, il faut brancher le programmateur USB comme si l'on voulait programmer le cerveau. Ensuite il suffit d'ouvrir un hyperterminal et de le configurer de la manière suivante : 9600bps, 8bit, 1 stop bit, pas de contrôle de parité, pas de contrôle de flux. A chaque appel de la fonction le microcontrôleur enverra le byte passé en argument.

Ex. :
```
int main(void)
{
	while(1)
	{
		// Envoie en boucle HELLO sur le bus de communication UART.
		uartSendByte('H') ;
		uartSendByte('E') ;
		uartSendByte('L') ;
		uartSendByte('L') ;
		uartSendByte('O') ;
		uartSendByte(' ') ;
	}
	return 0 ;
}
```

uartSendString(const char *text) Cette fonction envoie une chaîne de caractère par le bus de communication UART. La configuration du bus est la même que pour la fonction précédente.

Ex. :
```
int main(void)
{
	while(1)
	{
		// Envoie en boucle HELLO sur le bus de communication UART.
		uartSendString("HELLO ") ;
	}
	return 0 ;
}
```

unsigned char result = uartGetByte() Cette fonction lit un byte sur le bus de communication UART. La configuration du bus est la même que pour les fonctions précédentes. Attention, cette fonction est bloquante. Cela signifie que tant qu'aucune valeur n'est lue par le cerveau, le programme restera bloqué dans une boucle en attendant l'arrivée d'une valeur. Le résultat est ensuite placé dans la variable résultat.

Ex. :
```
int main(void)
{
	unsigned char resultat ;
	while(1)
	{
		// Renvoie en écho tout ce qui arrive sur le bus UART.
		resultat = uartGetByte() ;
		uartSendByte(resultat) ;

		// Test le caractère reçu et active ou désactive la LED si la bonne valeur est reçue.
		switch (resultat)
		{
		case 'n' :
			digitalWrite(C, 2, 1) ; // Allume la LED
			break ;
		case 'f' :
			digitalWrite(C, 2, 0) ; // Eteint la LED
			break ;
		}
	}
	return 0 ;
}
```

Fonctions de timing avancées :

char addNewCallback (void(*newcallbackaddr)(void), unsigned int duration, unsigned char executionNumber)

Cette fonction réalise une tâche équivalente à un agenda et permet de définir une fonction à appeler périodiquement. La fonction appelée ne doit retourner aucune valeur et n'avoir aucun argument.

Il faut définir l'intervalle de temps entre chaque appel de la fonction (temps entre appel = *duration·2ms*) et le nombre de fois que l'on veut que cette fonction soit appelée (0 signifie que la fonction est appelée sans limite du nombre d'exécution, autrement le nombre maximum d'exécution que l'on peut fixer est de 255). Le temps absolu maximum est de 8589934s à savoir environ 100 jours et l'intervalle maximum du temps entre appel est de 131s.

On peut définir au maximum 8 fonctions à appeler dans la liste de l'agenda. La fonction *addNewCallback* retourne un numéro correspondant à la place occupée par la fonction dans l'agenda et -1 en cas d'erreur (par exemple lorsqu'il n'y a plus de place disponible). Lorsqu'une fonction a été appelée le nombre maximum de fois défini, elle est automatiquement retirée de la liste de l'agenda et on peut placer une autre fonction à sa place en utilisant la fonction *addNewCallback* à nouveau.

Attention, l'exécution de la fonction appelée est prioritaire et bloque l'entier du cerveau. Il faut donc veiller à ne jamais rester bloquer dans une boucle d'attente à cet endroit par exemple.

Ex. :

```
void callbackBlink(void)
{
        // Fait clignoter la LED du cerveau
        static unsigned char blinkstatus = 0 ;
        if (blinkstatus == 0)
        {
                digitalWrite(C, 2, 1) ;
                blinkstatus = 1 ;
        }
        else
        {
                digitalWrite(C, 2, 0) ;
                blinkstatus = 0 ;
        }
}

int main(void)
{
        // Appel la fonction callbackBlink chaque seconde.
        char hndl ;
        hndl = addNewCallback(callbackBlink, 500, 0) ;          // 500·2ms=1s
        while(1) ;
        return 0 ;
}
```

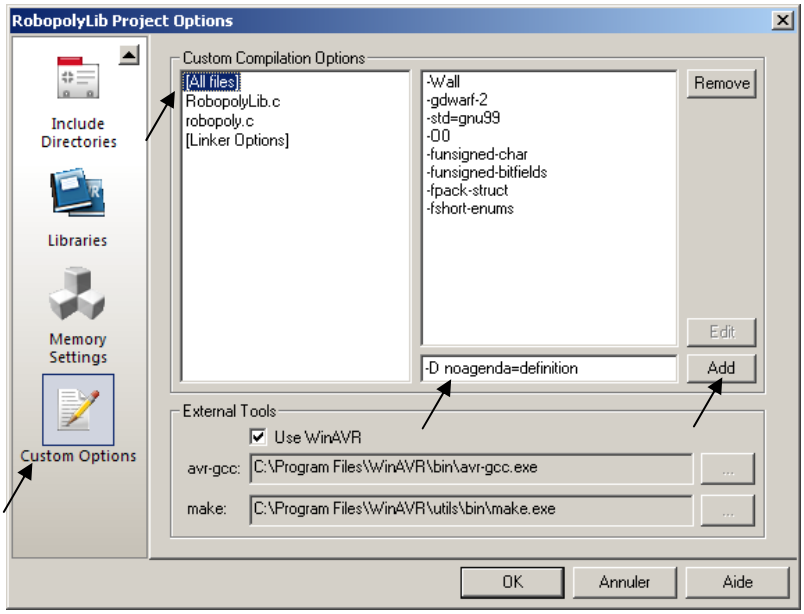
stopCallback(char callbackNumber) Permet de forcer l'arrêt d'une fonction de l'agenda (en particulier lorsque son nombre d'exécution est infini). Pour pouvoir utiliser cette fonction, il faut avoir récupéré le numéro identifiant la fonction lors de sa mise dans l'agenda (utilisation de la variable *hndl*).

Ex. :

```
…
int main(void)
{
        …
        hndl = addNewCallback(…) ;
        …
        stopCallback(hndl) ;
        …
}
```

Attention : Les deux fonctions précédentes utilisent le TIMER0. Pour utiliser librement le TIMER0 il faut désactiver l'agenda en ajoutant dans les options de compilation les informations suivantes :

-D noagenda=definition



Résumé des fonctions

Entrée/Sortie	**digitalWrite(port,bit, value)** port = {A,B,C,D} bit = {0,1,2,3,4,5,6,7, *BYTE*} value = {0,1, *0..255*}
	result = digitalRead(port, bit) port = {A,B,C,D} bit = {0,1,2,3,4,5,6,7, *BYTE*} result = {0,1, *0..255*}
	result = analogReadPortA(bit) bit = {0,1,2,3,4,5,6,7} result = {0..255}
Timing	**waitms(temps)** temps = {0..65536}
	waitus(temps) temps = {0..255}
Gestion moteur	**setupMotorPWM(vLeft, vRight)** vLeft = {-100..+100} vRight = {-100..+100}
	set_servo(num_servo, angle_servo) num_servo = {0,1,2,3,4,5,6,7,8,9} angle_servo = {0..100}
Communication	**uartSendByte(valeur)** valeur = {0..255, 'A..Z..0..9..'}
	uartSendString(const char *text) const char *text = {"your string here"}
	result = uartGetByte() result = {0..255, 'A..Z..0..9..'}
Timing avancé	char addNewCallback (void(*newcallbackaddr)(void), unsigned int duration, unsigned char executionNumber)
	stopCallback(char callbackNumber)