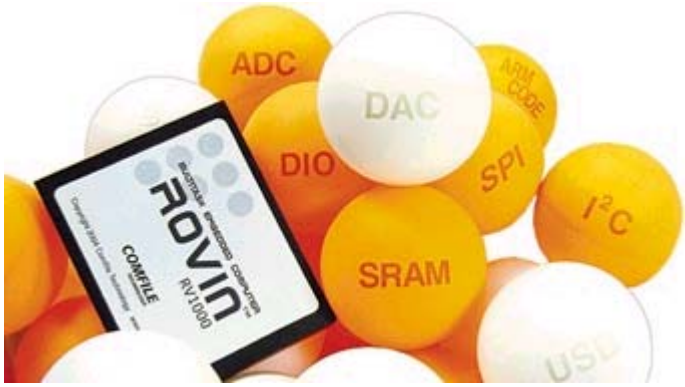
COMFILE
TECHNOLOGY

Manuel syntaxique du ROVIN

Version 0.03
20/01/2005

Déchaînez dès maintenant les possibilités multitâches et la puissance du ROVIN™ !



Version 0.02
- Le manuel du « cœur » du ROVIN™ a été ajouté.

Version 0.03
- Le manuel du « ROVIN-IDE » a été ajouté.

Copyrights et appellations commerciales

Informations techniques

Ce manuel a été conçu avec la plus grande attention. Tous les efforts ont été mis en oeuvre pour éviter les anomalies. Toutefois, nous ne pouvons garantir que ce dernier soit à 100% exempt de toute erreur. Les informations présentes dans ce manuel sont données à titre indicatif. Les caractéristiques techniques des modules "ROVIN™", la nature, les possibilités et le nombre de leurs instructions, ainsi que les possibilités de leurs logiciels de programmation et les caractéristiques des modules périphériques associés aux modules "ROVIN™", peuvent changer à tout moment sans aucun préavis dans le but d'améliorer la qualité et les possibilités de ces derniers. Ces produits sont protégés par des brevets à travers le monde.

Limitation de responsabilité

En aucun cas le Fabricant et LEXTRONIC ne pourront être tenus responsables de dommages quels qu'ils soient (intégrant, mais sans limitation, les dommages pour perte de bénéfice commercial, interruption d'exploitation commerciale, perte d'informations et de données à caractère commercial ou de toute autre perte financière) provenant de l'utilisation ou de l'incapacité à pouvoir utiliser les modules "ROVIN™" et leurs logiciels associés ainsi que leurs platines et modules optionnels associés, même si le Fabricant ou LEXTRONIC ont été informés de la possibilité de tels dommages.

Les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés sont destinés à être utilisés en milieu résidentiel dans les gammes de températures +10 à +50 °C. Les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés ne sont pas conçus, ni destinés, ni autorisés pour être utilisés au sein d'applications militaires, ni au sein d'applications à caractère médical, ni au sein d'applications d'alerte incendie, ni au sein d'applications pour ascenseurs, ni au sein d'applications sur machine outils, ni au sein d'applications embarquées dans des véhicules (automobiles, camions, bateaux, scooters, motos, kart, scooters des mers, avions, hélicoptères, ULM, etc...), ni au sein d'applications embarquées sur des maquettes volantes de modèles réduits (type avions, hélicoptères, planeurs, etc...).

De même, les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés ne sont pas conçus, ni destinés, ni autorisés pour expérimenter, développer ou être intégrés au sein d'applications dans lesquelles une défaillance de ces derniers pourrait créer une situation dangereuse pouvant entraîner des pertes financières, des dégâts matériels, des blessures corporelles ou la mort de personnes ou d'animaux. Si vous utilisez les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés volontairement ou involontairement pour de telles applications non autorisées, vous vous engagez à soustraire le Fabricant et LEXTRONIC de toute responsabilité et de toute demande de dédommagement.

En cas de litige, l'entière responsabilité du Fabricant et de LEXTRONIC vis-à-vis de votre recours se limitera exclusivement selon le choix du Fabricant et de LEXTRONIC au remboursement du module "ROVIN™" et/ou de ses platines et modules optionnels associés et/ou de leur réparation et/ou de leur échange. Le Fabricant et LEXTRONIC démentent toutes autres garanties, exprimées ou implicites.

L'utilisateur des modules "ROVIN™" et de ses platines et modules optionnels associés est entièrement et seul responsable des développements logiciels (de l'écriture de son programme) ainsi que de l'intégration matérielle, des modifications et ajouts de périphériques qu'il effectuera sur les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés. S'agissant de matériel "OEM", Il incombera à l'utilisateur de vérifier que l'application finie complète développée avec les modules "ROVIN™" ainsi que leurs platines et modules optionnels soient conformes aux normes de sécurité et CEM en vigueur.

Tous les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés sont testés avant expédition. Toute inversion de polarité, dépassement des valeurs limites des tensions d'alimentation, courts-circuits, utilisation en dehors des spécifications et limites indiquées dans ce document ou utilisation pour des applications non prévues pourront affecter la fiabilité, créer des dysfonctionnements et/ou endommager les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés sans que la responsabilité du Fabricant et de LEXTRONIC ne puisse être mise en cause, ni que les produits puissent être échangés au titre de la garantie.

Rappel sur l'évacuation des équipements électroniques usagés

Ce symbole présent sur les modules "ROVIN™" ainsi que leurs platines et modules optionnels associés et/ou leurs emballages indique que vous ne pouvez pas vous débarraser de ces produits de la même façon que vos déchets courants. Au contraire, vous êtes responsable de l'évacuation de ces produits lorsqu'ils arrivent en fin de vie (ou qu'ils sont hors d'usage) et à cet effet, vous êtes tenu de le remettre à un point de collecte agréé pour le recyclage des équipements électriques et électroniques usagés. Le tri, l'évacuation et le recyclage séparés de vos quipements usagés permettent de préserver les ressources naturelles et de s'assurer que ces équipements sont recyclés dans le respect de la santé humaine et de l'environnement. Pour plus d'informations sur les lieux de collecte des équipements électroniques usagés, veuillez contacter votre mairie ou votre service local de traitement des déchets.

Note for all residents of the European Union

This symbol on the product or on its packaging indicates that this product must not be disposed of with other household waste. Instead, it is your responsability to dispose of your waste equipement by handing it over to designated collection point for the recycling of waste electrical and electric equipement. The separate collection and recycling of your waste equipement at the time of disposal will help to conserve natural resources and ensure that it is recycled in a manner that protects human health and environnement. For more information about where you can drop off your waste equipement for recycling, please contact your local city office or your local hold waste disposal service.

Sommaire

INTRODUCTION

Bienvenue dans le « monde » du ROVIN™ de Comfile Technology, lequel s'apparente au produit le plus novateur que nous ayons développé grâce à nos années d'expériences en matière de modules microcontrôlés 8 bits. Le ROVIN™ est par contre très différent de nos précédents produits en ce sens que son circuit principal soit un processeur ARM™ 32 bits qui intègre un système d'exploitation multitâches (RTOS). Ce dernier permettra aux utilisateurs de développer des programmes indépendants sans avoir à se soucier de la gestion interne multitâches proprement dite.

Ainsi vous pourrez écrire très simplement de multiples programmes qui fonctionneront automatiquement et simultanément, sans avoir à vous préoccuper de la façon dont fonctionne le processeur ARM™ ou le système d'exploitation (RTOS) tout en bénéficiant d'une capacité mémoire de 512 KB et d'une vitesse d'exécution avoisinant les 80 MHz !

Nous travaillons constamment pour devenir un développeur leader de modules microcontrôlés embarqués. Merci d'avoir fait l'acquisition du module ROVIN™. Nous espérons que vous apprécierez ce dernier autant que nous...

A propos de ce manuel…

Ce manuel est consacré à la description détaillée de la syntaxe du langage de programmation « ROVIN-C ». Son but principal et de vous en apprendre la syntaxe générale.

Nous continuerons à mettre à jour et à améliorer ce manuel. Toutes les mises à jour seront téléchargeables sur le www.comfile.co.kr.

A PROPOS DU ROVIN™

Le ROVIN™ est un « super » module microcontrôlé compact. Ce dernier est principalement destiné à être intégré au sein d'applications professionnelles, de systèmes d'automatisation, de contrôle de process, de collecte de données ainsi que pour tous les systèmes nécessitant un puissant traitement multitâches. Le ROVIN™ utilise un système d'exploitation développé par Comfile Technology™ (Virtual Operating System – une sorte de système d'exploitation temps réel RTOS). Ce dernier peut également gérer les calculs mathématiques en mode 64 bits.

Le ROVIN™ dispose de son propre compilateur « C » avec environnement de développement IDE intégrant un éditeur, un compilateur et un débuggeur fonctionnant sur PC (sous environnement Windows™ XP et prochainement Windows™2000). Vous disposez ainsi d'une chaîne de développement complète de A jusqu'à Z. Un câble USB spécial sera utilisé pour télécharger vos programmes au sein de la mémoire Flash du ROVIN™. Ces programmes seront sauvegardés même en cas de coupure d'alimentation du fait que la mémoire Flash du module ne soit pas volatile.

Le débuggeur présent dans l'environnement de développement du ROVIN™ permet l'utilisation des modes SINGLE-STEP, ON-THE-FLY DEBUG et la visualisation des variables permettant ainsi de disposer de fonctions similaires à celles d'un émulateur. Le câble USB pourra être déconnecté du module ROVIN™ une fois le développement terminé afin que ce dernier devienne autonome. SI des modifications mineures sont nécessaires il vous suffira à nouveau de relier le câble USB au ROVIN™ pour effectuer celles-ci.

Le ROVIN™ dispose de base de 64 E/S (8 bits), de 2 liaisons séries UART (RS-232), d'une gestion logiciel I2C™/SPI™, de divers convertisseurs « A/N », d'un comparateur analogique (ANCOMP), de sorties PWM, d'une mémoire EEprom, etc…

De plus le langage « C » du ROVIN™ dispose d'une large gamme de librairies faciles à assimiler et à utiliser. Dès lors tout utilisateur (même ne disposant pas d'une grande connaissance en langage « C ») pourra créer très facilement des programmes multitâches en quelques minutes au lieu de quelques semaines ou de quelques mois comme sur les systèmes conventionnels.

ROVIN VS. MCU

Voyons maintenant les différences entre un module microcontrôlé (MCU) et le module ROVIN™. Les MCU et le ROVIN™ sont tous les deux utilisables pour des applications microcontrôlées embarquées. Toutefois en fonction de la nature de cette application, il vous sera préférable d'utiliser le ROVIN™ ou un MCU standard.

En premier lieu, détaillons les avantages de l'environnement de développement du module ROVIN™.

Le développement sur un MCU standard nécessite généralement une expérience de haut niveau. De plus il vous faudra acquérir un émulateur, un programmateur et un compilateur « c » dont les coûts peuvent être très élevés. De plus, si vous désirez développer un dispositif doté de possibilités multitâches, vous devrez également vous fournir un système d'exploitation temps réel (RTOS).

Quand bien même vous serez en mesure de pouvoir vous « offrir » cet ensemble complet, il vous faudra encore passer un très grand nombre d'heures, de jours, de semaines (voir de mois) pour maîtriser l'ensemble en retardant ainsi d'avantage la phase de production de votre application.

Le ROVIN™

Du fait que le ROVIN™ intègre à la base des possibilités de debuggage ainsi que son propre système d'exploitation multitâches, il ne vous sera pas nécessaire d'acquérir d'autres dispositifs additionnels. L'environnement de développement « ROVIN-C » est qui plus ait disponible en libre téléchargement et en version complète (vous n'utilisez pas ici de compilateur limité ou de démo…). Tout ce que vous aurez besoin pour démarrer sera un compatible PC et le câble de téléchargement USB du module ROVIN™. Si vous connaissez ou êtes en mesure d'assimiler les bases du langage « c », vous pourrez développer des applications très complexes à l'aide des librairies du ROVIN™. Ces librairies ont été conçues avec soin afin de pouvoir vous aider à développer vos projets encore plus rapidement et facilement.

Le ROVIN™ permet également d'améliorer les possibilités de maintenance de votre application.

Prenons par exemple le cas d'un problème survenant sur le système que vous avez développé après que ce dernier ait été commercialisé… Dans la « vrai » vie, les problèmes surviennent toujours aux moments où on s'y attend le moins. Avec la plupart des MCU standards, il vous sera impossible de modifier ou de débuguer votre application sur site. Vous devrez stopper le système et amener la carte en laboratoire ou amener le système de développement sur site (si cela est possible). Dans ce cas, soit il vous faudra remplacer la carte temporairement ou stopper complètement le système suivant les cas.

Avec le ROVIN, un simple PC portable doté d'un port USB pourra être utilisé pour apporter les modifications logiciel sur site en très peu de temps. Ceci est un des autres principaux avantages du module ROVIN™ vis-à-vis des solutions MCU standards.

Un des autres avantages du ROVIN™ est bien évidemment ses possibilités multitâches. Et bien qu'il soit possible de développer ce type de fonctionnalité sur un MCU standard, ceci relève d'une très grande complexité alors qu'avec le ROVIN™ vous n'aurez à vous soucier de rien et pourrez bénéficier de ces avantages sans connaissance particulière.

Le multitâches signifie que vous pouvez exécuter diverses actions simultanément. Ainsi il y a une multitude de systèmes qui requièrent ce type de possibilités, surtout sur les machines qui sont dotées de plusieurs capteurs dont les états doivent être surveillés en même temps. Et si un des capteurs venait à signaler une anomalie, le système devra être en mesure de la signaler immédiatement.

Avec un MCU standard, il est courant d'utiliser des interruptions pour gérer ce type d'applications. Avec le ROVIN, tout sera plus simple de par son système d'exploitation temps réel qui s'occupera de tout. Le ROVIN utilise un puissant processeur ARM™ 32 bits qui gère un système d'exploitation propriétaire stable ; le ROVIN-VMS (une sorte de système d'exploitation RTOS spécialement conçu pour le ROVIN), lequel s'occupe de toutes les actions courantes : opérations mathématiques, librairies de fonctions, gestion des E/S…

Si un seul programme est téléchargé dans le ROVIN™, ce dernier fonctionnera comme un programme MONO-TACHE. Si 2 programmes sont téléchargés, alors ils fonctionneront en mode multitâches ; chaque programme fonctionnant indépendément l'un de l'autre. Si un des programmes venait à stopper en raison d'un bug, d'une erreur de programmation ou autre, l'autre programme (tâche) ne serait pas affecté et continuerait de fonctionner normalement.

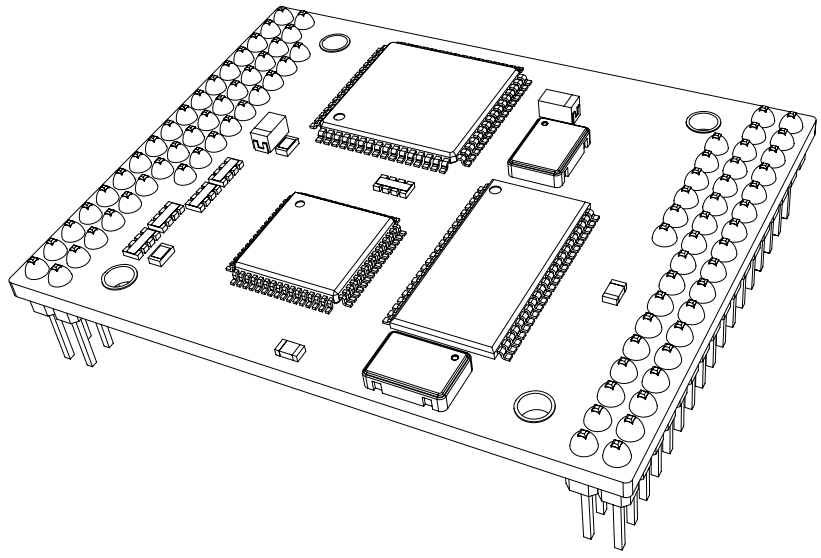
Chaque programme téléchargé dans le ROVIN™ sera exécuté indépendamment des programmes déjà téléchargés (qu'il y en ait 2 ou 3 ou … 10). Lorsque l'on désigne un programme, la notion de fonctionnement multitâches interne n'a pas besoin d'être prise en compte. La conception est alors plus sûre et plus stable avec ce type de fonctionnement du fait que chaque programme fonctionne indépendamment les uns des autres.

De part ses très nombreuses librairies capables de prendre en charge les communications SPI™, I2C™, des afficheurs LCD à commandes série, des interruptions diverses… le ROVIN™ est très simple à utiliser et il vous permettra de développer des applications en quelques minutes là où il vous faudra des heures pour d'autres systèmes.

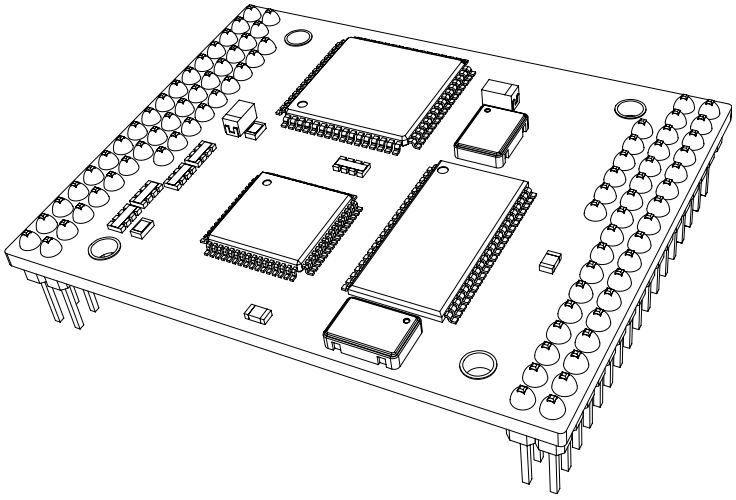
Comme vous pouvez le constater, il y a de très nombreux avantages à utiliser le ROVIN plutôt que des microcontrôleurs traditionnels. Une fois que vous aurez commencé à développé avec le ROVIN™… il vous sera alors très difficile de réutiliser des MCU traditionnels !

COMFILE
TECHNOLOGY

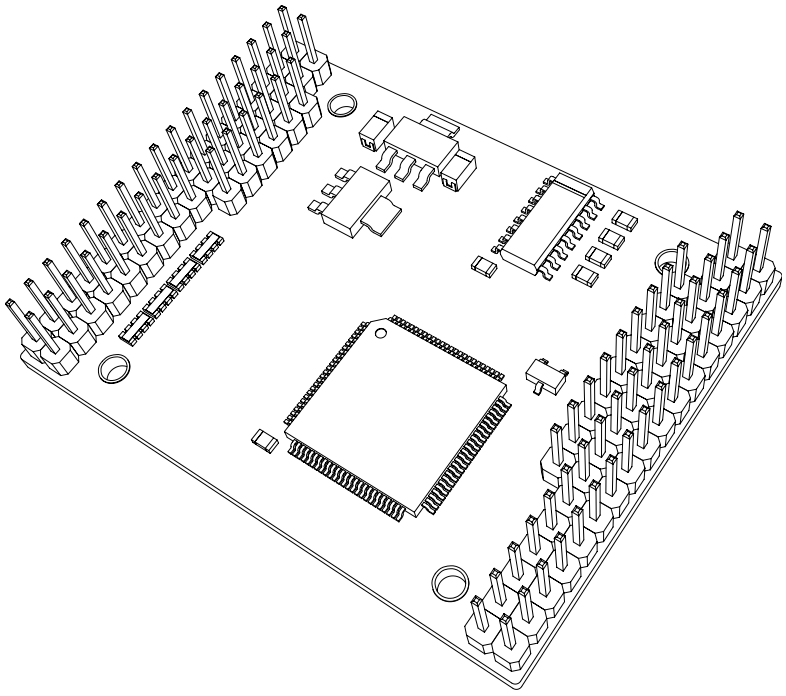
LE «CŒUR » DU
DU ROVIN



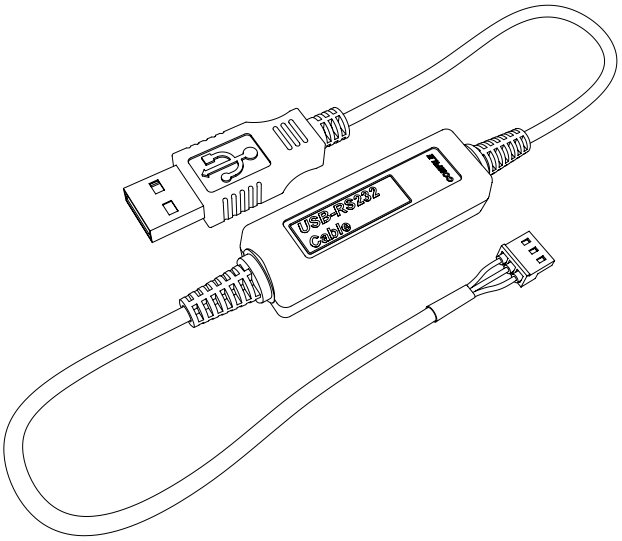
ROVIN & CABLE USB



DESSUS



DESSOUS



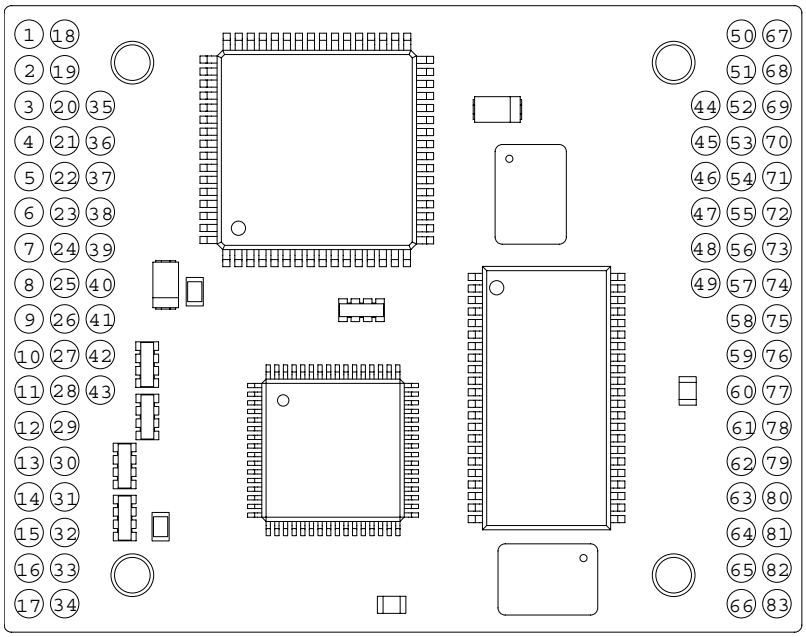
CABLE USB

Caractéristiques du ROVIN

- MODULE MICROCONTROLE MULTITACHE
- Langage supporté : « C »
- Processeur principal : ARM™ (32 bits)
- Connexion au PC: via port USB

- MEMOIRE FLASH: 128 K
- MEMOIRE DONNEES: 192 K
- MEMOIRE HEAP: 256 K
- EEPROM : 4 K
- PORTS « E/S » : 56 PORTS
- CONVERTISSEURS « A/N »: 8 CANAUX RESOLUTION 10 BITS
- PWM : 7 CANAUX AVEC 8~16 BITS
- COMPARATEUR ANALOGIQUE
- RTC, ALARME
- 8 ENTREES INTERRUPTION EXTERNE
- 2 CANAUX RS-232
- CONTROLE SERVOMOTEURS : 6 SORTIES
- GESTION PROTOCOLE : SPI™, I2C™
- CAPTURE, COMPTEUR
- COMMANDE POUR GESTION D'AFFICHEURS SERIE « ALCD »
- 2 CANAUX « SONS »

BROCHAGE DU ROVIN



BROCHAGE DU ROVIN™ (vue du dessus)

Rôle de chaque broche:

Numéro	Nom	Possibilités Supplément.	Explication:
1	EXPA7	ADC7	E/S générale ou conversion « A/N ».
2	EXPA6	ADC6	E/S générale ou conversion « A/N ».
3	EXPA5	ADC5	E/S générale ou conversion « A/N ».
4	EXPA4	ADC4	E/S générale ou conversion « A/N ».
5	EXPA3	ADC3	E/S générale ou conversion « A/N ».
6	EXPA2	ADC2	E/S générale ou conversion « A/N ».
7	EXPA1	ADC1	E/S générale ou conversion « A/N ».
8	EXPA0	ADC0	E/S générale ou conversion « A/N ».
9	VCCIO-PA		Tension de PULL-UP du port PA. (Moins de 5V)
10	PA7		E/S générale.
11	PA6		E/S générale.
12	PA5		E/S générale.
13	PA4		E/S générale.
14	PA3		E/S générale.
15	PA2		E/S générale.
16	PA1		E/S générale.
17	PA0		E/S générale.
18	EXPB7	INT7, CP1	E/S générale, CP1, ou interruption externe.

19	EXPB6	INT6, CT0	E/S générale, CT0, ou Interruption externe.
20	EXPB5	PWM/INT5	E/S générale, PWM3C, ou Interruption externe.
21	EXPB4	PWM/INT4	E/S générale, PWM3B, ou Interruption externe.
22	EXPB3	PWM/ AIN1 (-)	E/S générale, PWM3A, Tension entrée ANCOMP (-), ou Interruption externe.
23	EXPB2	AIN0(+)	E/S générale ou Tension entrée ANCOMP (+).
24	EXPB1	TXD0	E/S générale ou TXD0. (UART)
25	EXPB0	RXD0	E/S générale ou RXD0. (UART)
26	VCCIO-PB		Tension de PULL-UP du port PB. (Moins de 5V)
27	PB7		E/S générale.
28	PB6		E/S générale.
29	PB5		E/S générale.
30	PB4		E/S générale.
31	PB3		E/S générale.
32	PB2		E/S générale.
33	PB1		E/S générale.
34	PB0		E/S générale.
35	GND(0.0V)		GND (A connecter au 0 V)
36	XRESET		RESET DU MODULE ROVIN™.
37	PC-TXD		TXD pour connexion du ROVIN™ au PC
38	PC-RXD		RXD pour connexion du ROVIN™ au PC
39	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
40	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
41	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
42	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
43	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
44	VCC(5.0V)		A connecter au 5.0 V.
45	3.3V OUT		Sortie 3.3 V. A n'utiliser que pour alimenter des systèmes très faible consommation.
46	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
47	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
48	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
49	SYSTEM		Laissez libre (Ne rien connecter sur cette broche !)
50	PWM1C	PWM2	Broche PWM1C/PWM2 uniquement de sortie.
51	PWM1B		Broche de sortie PWM1B uniquement.
52	PWM1A		Broche de sortie PWM1A uniquement.
53	PWM0		Broche de sortie uniquement.
54	XBUS		Laissez libre (Ne rien connecter sur cette broche !)
55	XBUS		Laissez libre (Ne rien connecter sur cette broche !)
56	XBUS		Laissez libre (Ne rien connecter sur cette broche !)
57	XBUS		Laissez libre (Ne rien connecter sur cette broche !)
58	VCCIO-PC		Tension de PULL-UP du port PC. (Moins de 5V)
59	PC7		E/S générale.
60	PC6		E/S générale.
61	PC5		E/S générale.
62	PC4		E/S générale.
63	PC3		E/S générale.
64	PC2		E/S générale.

65	PC1		E/S générale.
66	PC0		E/S générale.
67	EXPD7	CT2	E/S générale ou CT2.
68	EXPD6	CT0	E/S générale ou CT0.
69	EXPD5		E/S générale.
70	EXPD4	CP0	E/S générale ou CP0.
71	EXPD3	TXD1/INT3	E/S générale, TXD1, broche Interruption externe.
72	EXPD2	RXD1/INT2	E/S générale, RXD1, broche Interruption externe.
73	EXPD1	INT1	E/S générale ou broche Interruption externe.
74	EXPD0	INT0	E/S générale ou broche Interruption externe.
75	VCCIO-PD		Tension de PULL-UP du port PD. (Moins de 5V)
76	PD7		E/S générale.
77	PD6		E/S générale.
78	PD5		E/S générale.
79	PD4		E/S générale.
80	PD3		E/S générale.
81	PD2		E/S générale.
82	PD1		E/S générale.
83	PD0		E/S générale.

SYSTEM

Les broches SYSTEM sont utilisées pour la conception « en usine » du module ROVIN™. Ne connectez jamais rien sur ces broches sous peine de destruction irrémédiable du module ROVIN™.

XBUS

Les broches XBUS sont utilisées pour des communications entre les circuits internes du module ROVIN™. Ne connectez jamais rien sur ces broches sous peine de destruction irrémédiable du module ROVIN™.

VCC(5V)

Ceci est la broche d'alimentation du module ROVIN™. La tension appliquée doit être régulée et correctement filtrée. Cette dernière doit être comprise 4,5 et 5,5 volts. De part la présence d'un circuit de gestion interne de RESET, la tension ne doit jamais descendre en dessous de 4 V sans quoi le module ROVIN™ effectuera un « RESET » automatiquement. Il vous faudra dès lors en tenir compte lors l'intégration de votre module dans votre application finale.

GND

La broche GND devra être connectée au 0 volt (masse). Il est impératif de relier cette dernière, sans quoi le module ROVIN™ ne fonctionnera pas correctement. Pensez à faire une piste « large » si vous concevez vous-même votre circuit imprimé.

VCCIO

Les ports d'entrées/sorties du ROVIN™ utilisent des tensions de référence déterminées par les broches VCCIO. Ces tensions peuvent être établies à 1.8 V, 2.5 V, 3.3 V, 5.0 V (jusqu'à 5.5 V max.). Les broches VCCIO pourront être indépendemment attribués aux ports PA, PB, PC et PD. Vous pourrez ainsi par exemple configurer les ports comme suit : VCCIO-PA = 5.0V, VCCIO-PB = 3.3V, VCCIO-PC = 2.5V, VCCIO-PD = 1.8V.

Ceci est encore un des avantages du ROVIN™ par rapport à la plupart des autres systèmes qui disposent généralement de tensions de références internes. Dans le passé, les dispositifs mirocontrôlés 5 V étaient considérés comme la « référence ». Maintenant les microcontrôleurs 3 V sont très commun et les modèles 5 V sont plus difficile à trouver (mise à part pour les CPU 8 bits). Dés lors, si vous développez en utilisant des microcontrôleurs avec différentes tensions, ceci peut vite s'avérer problématique. En revanche, avec le ROVIN™ vous ne serez pas confronté à ce type d'inconvénient et il ne vous sera pas nécessaire d'avoir recours à des circuits « switch » ou autres 74LVCXXX d'interfaçage supplémentaires.

3.3V OUT

Cette broche est une sortie 3.3V. Cette dernière pourra être utilisée pour alimenter un dispositif externe à condition que sa consommation n'excède pas quelques milliampères sous peine de dysfonctionnement sévère (voir irrémédiable) du module ROVIN™. Utilisez donc cette sortie avec beaucoup de précaution…

PA/PB/PC/PD

Ces ports correspondent aux ports d'E/S du ROVIN™. Vous pourrez non seulement déterminer l'état d'entrée/sortie de chaque bit, mais également les utiliser pour de nombreuses autres possibilités de communication tels SPI™, I2C™, etc…

EXPA/EXPB/EXPD

Les ports EXPA, EXPB, EXPD sont assez différents des entrées/sorties standards en ce sens qu'ils sont destinés à s'interfacer avec des périphériques 5 V afin de pouvoir fournir des entrées/sorties à usage standard our des fonctions speciales telles que UART, conversion « A/N », comparateur analogique et PWM. Notez que le port EXPC n'existe pas.

XRESET

La broche XRESET est utilisée pour initialiser le module ROVIN™. Cette broche est similaire à celle de la broche RESET d'un microcontrôleur. Un signal BAS réalisera un Reset du module ROVIN™. Cette broche doit être appliquée au niveau logique HAUT (5 V) en fonctionnement normal.

PC-TXD/PC-RXD

Ces broches sont utilisées pour la communication entre le PC et le module ROVIN™ via le câble USB pour la programmation lors de vos phases de développement. Utilisez toujours le câble de programmation USB dédié spécialement conçu pour le module ROVIN™.

PWM

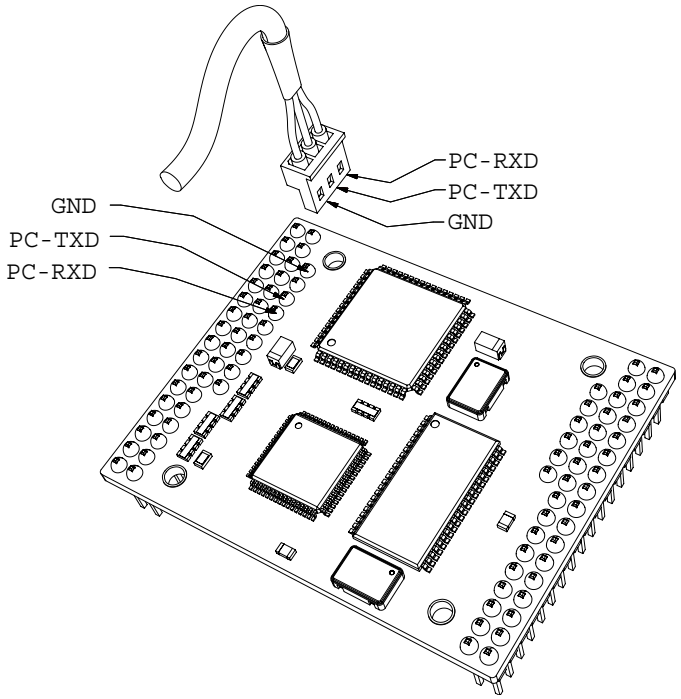
Ces broches sont destinées à générer des signaux PWM susceptibles de piloter des moteurs « CC » ou d'autres dispositifs.

CONNEXION DU CABLE USB

Vous devrez vous procurer le câble de connexion USB du ROVIN™ si vous avez achetez le module ROVIN™ seul. Installez le driver USB lors de la première connexion du câble USB sur le PC (une fenêtre d'installation s'affichera alors pour vous demander d'installer ce dernier). Sélectionnez le répertoire relatif au driver du câble USB dans le répertoire du ROVIN (généralement X:/Program Files/ROVIN).

Si vous utilisez une platine d'évaluation telle que la « ROVIN-STUDY-BOARD » ou la « ROVIN Q-Start Board » ou une autre platine similaire, insérez le connecteur 3 points en bout de câble sur le connecteur prévu à cet effet sur la platine.

Si vous réalisez vous-même votre propre circuit imprimé, vous pouvez acquérir le connecteur mâle associé afin que vous puissiez réaliser la connexion ci-dessous.



Raccordement du module ROVIN™ au câble USB

Structure mémoire

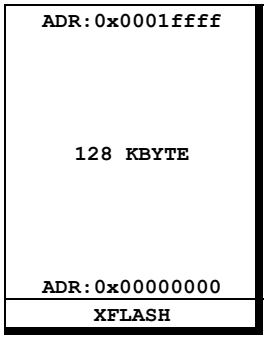
Le ROVIN dispose de 4 types de mémoire: **XFLASH, XSRAM, XHEAP et EEPROM**. Chaque mémoire dispose d'une fonction différente.

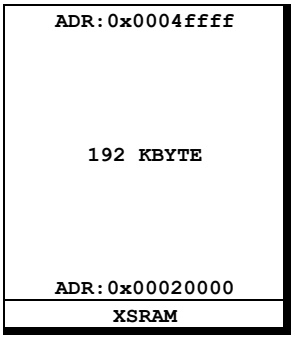
La mémoire **XFLASH** servira à stocker les programmes. Il s'agit d'une mémoire non volatile (sauvegardée même en cas de coupure d'alimentation).

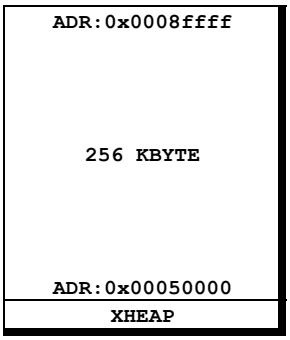
La mémoire **XSRAM** (**X-Static RAM**) sert à mémoriser les caractères, les nombres et autres variables (données) pendant l'exécution des programmes. Du fait qu'elle soit volatile (les données sont perdues en cas de coupure d'alimentation), il est recommandé de sauvegarder certaines données dans la mémoire EEPROM si nécessaire.

La mémoire **EEPROM** est une mémoire non volatile pouvant être utilisée pour sauvegarder des constantes et autres données.

La mémoire **XHEAP** est spécialement conçue pour sauvegarder des données durant l'exécution du programme. De grande capacité, cette dernière pourra être utilisée librement pour échanger des informations entre les différents programmes (tâches). Cette mémoire est de type SRAM (donc volatile).







XFLASH

La capacité de la mémoire XFLAH est de **128 K (0x20000)**.
Cette mémoire utilise les emplacements compris entre 0 à 1ffff. Il vous sera possible d'utiliser un pointeur pour y lire une valeur depuis une adresse (mais pas pour y écrire).

XSRAM

La capacité de la mémoire XRAM est de **192 K (0x30000)**.
L'utilisateur devra faire attention à ne pas dépasser la capacité limite de cette mémoire (surtout pour les applications multitâches)… sans quoi le système pourrait fonctionner de façon anormal. Cette mémoire utilise les emplacements compris de 00020000h ~ 0004ffffh.

L'utilisation de pointeurs pour le stockage des données en mémoire XSRAM requière une grande attention et leurs recours nécessiteront de déclarer des adresses impérativement comprises dans la plage mémoire autorisée sous peine de dysfonctionnements sévères du module ROVIN™ (il en est de même pour les adressages aléatoires de la mémoire qui pourront créer des anomalies de fonctionnement importants s'ils sont mal utilisés).

XHEAP

La capacité de la mémoire XHEAP est de **256 K (0x40000)**.
L'utilisateur pourra exploiter librement cet espace et y accéder de façon arbitraire sans aucune déclaration. Le système d'exploitation « ROVIN-VMS » génère simplement cet espace mémoire sans y avoir recours (vous laissant ainsi l'opportunité d'accéder aux 256 K). Sachant que les accès à cette mémoire (entrée/sortie) sont très rapides, la mémoire XHEAP pourra être utilisée pour la création de buffers temporaires ou pour « travailler » et modifier une large quantité de données. Des pointeurs pourront être utilisés afin de pouvoir stocker des données « n'importe où ». De nombreuses fonctions exploitent d'ores et déjà la mémoire XHEAP afin que vous puissiez profiter pleinement de ses capacités.

Le multitâches selon le ROVIN™

Le module ROVIN™ supporte les applications MULTITACHES grâce à un MANAGEUR DE TACHES conçu autour d'un NANO-OS interne. Son mode de fonctionnement est de type « Round-Robin ».

Le langage « ROVIN-C » est un dérivé du langage « C » spécialement adapté au système d'exploitation (ROVIN-VMS) interne au module. Quoi que légèrement différent sur certains points, il est pratiquement entièrement compatible avec les standards « ANSI-C ». Toutefois, le système d'exploitation du ROVIN™ reste différent des machines virtuelles Java™, des RTOS embarqués et de la plupart des autres systèmes similaires.

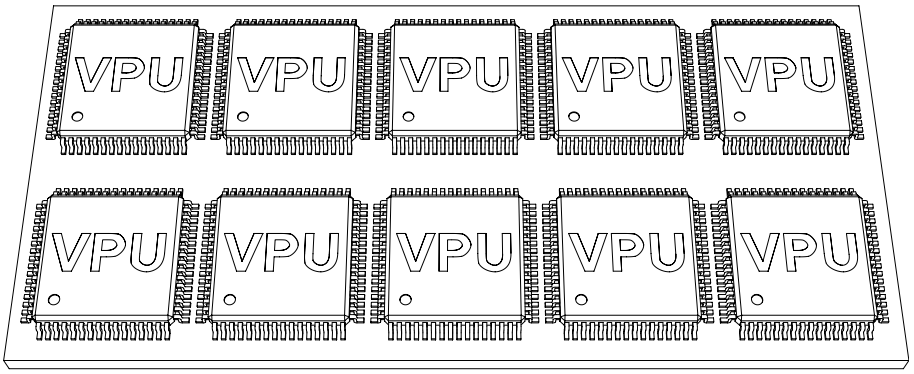
Le mode de fonctionnement MULTITACHES du ROVIN™ peut être à titre d'image être comparé au fonctionnement de Linux en ce sens qu'il est capable de traiter des programmes exécutables en même temps.

Ainsi l'utilisateur n'aura pas à se soucier de la façon dont fonctionne le mode multitâches. La seule chose dont il aura à s'assurer sera que chaque programme exécuté n'utilise pas plus que la mémoire autorisée par le système d'exploitation « ROVIN-VMS ».

D'un point de vue utilisateur, vous ne verrez pas la différence entre la création d'un seul programme ou de plusieurs programmes à la fois. Chaque programme sera exécuté indépendamment les uns des autres et la fonctionnalité multitâches sera automatiquement gérée et prise en charge par le module. Dès lors même si vos connaissances en ce qui concerne les systèmes multitâches sont inexistantes il vous sera tout de même possible de développer des applications professionnelles qui pourront soutenir la comparaison avec bon nombre d'autres systèmes beaucoup plus complexes à maîtriser.

Le module ROVIN™ est capable à ce jour (août 2004) de supporter jusqu'à 10 tâches simultanément. Dans le futur, nous disposerons de nouvelles versions capables de supporter d'avantages de tâches.

VPU (Virtual Process Unit)



ROVIN

Par exemple si 10 programmes s'exécutent en même temps il y aura 10 unités processeurs virtuels (VPU) de créés. Chaque VPU exécutera indépendamment son OPCODE qui n'affectera pas les autres. Le ROVIN™ utilise des CYCLES MACHINES plutôt que des TOURS D'HORLOGES généralement utilisés dans la plupart des autres systèmes d'exploitation embarqués. 1 OPCODE monopolise 1 CYCLE MACHINE au processeur.

Les possibilités multitâches du ROVIN™ sont effectuées par chaque VPU qui exécute à son tour un OPCODE. Le VPU est sélectionné par le MANAGEUR DE TACHES pour prendre en charge un certain OPCODE. Chaque VPU dispose d'un code identification (ID) unique. Le MANAGEUR DE TACHE appel un VPU spécifique par ordre de son numéro d'ID. Toutes les TACHES disposent de leur propre SET DE REGISTRES et MEMOIRE DE DONNEES qui sont utilisés indépendamment les uns des autres. Contrairement à la plupart des autres RTOS embarqués, les opérations s'exécutent de la même façon que s'il y avait plusieurs processeurs en même temps comme sur la représentation ci-dessus.

A ce jour, le module ROVIN™ peut supporter jusqu'à 10 tâches simultanément (la limitation venant de sa capacité mémoire et de sa vitesse d'exécution). Dans un futur proche, le ROVIN™ pourra supporter plus de tâches du fait que nous améliorons constamment la quantité de mémoire et la vitesse de nos CPU.

MULTITACHES ?

Le mode de fonctionnement multitâches du ROVIN™ est différent de celui réalisé par un RTOS embarqué. En effet les RTOS embarqués utilisent de nombreuses fonctions pour pouvoir gérer les TACHES. En d'autres termes, la gestion d'applications multitâches requière une connaissance parfaite de ces fonctions et la prise en compte des TACHES doit être directement intégrée dans le code source. Dès lors une compétence professionnelle sérieuse est indispensable et un nombre limité de personnes est vraiment à même de « manipuler » ce type de code.

Comme indiqué précédemment, le mode de fonctionnement multitâches supporté par le ROVIN™ est très similaire à celui du système d'exploitation d'un PC. Par exemple, vous pouvez très bien exécuter en même temps un logiciel de traitement de texte, un logiciel de dessin et un programme de navigation Internet ou tout aussi bien exécuté qu'un seul de ces programmes à la fois. Chacun d'entre eux étant supporté par le système d'exploitation.

Le ROVIN™ est conçu autour d'une vision similaire. Comme pour le système d'exploitation d'un PC, le ROVIN™ pourra exécuter un seul et unique programme ou plusieurs programmes à la fois de façon totalement indépendants. La seule différence est que le ROVIN™ executera tous les programmes depuis leur début.

UTILISATION D'« E/S » COMMUNES

Lorsqu'on exécute de multiples TACHES qui ont accès à de multiples ports d'E/S, il peut se poser des problèmes si plusieurs tâches essaient par exemple d'utiliser en même temps un port de communication de type UART… Du fait que le ROVIN™ ne dispose pas d'une gestion prioritaire des TACHES, si 2 TACHES envoient une commande à l'UART au même moment, l'UART ne saura pas quel « processeur interne » est le premier.

Sur les systèmes d'exploitation de type RTOS, des « signaux de communication » sont utilisés pour résoudre ce type de contexte. Ainsi avec le ROVIN™ vous pourrez utiliser la mémoire XHEAP pour réaliser vos propres « signaux de communication ». Sauvegardez simplement la valeur « TRUE » à l'adresse 00000000h lorsqu'une tâche utilise l'UART et les autres taches pourront être facilement programmées pour attendre que cette dernière « passe » à « FALSE ».

EVENEMENTS

Le module ROVIN™ est capable de gérer des « EVENEMENTS ». Ces derniers sont similaires aux interruptions présentent sur d'autres microcontrôleurs avec toutefois quelques différences mineures. Ainsi, le ROVIN™ exploite la couche supérieure du système d'exploitation « ROVIN-VMS ». Lorsque le « ROVIN-VMS » reçoit une interruption, cette dernière est convertie en un MESSAGE associé, lequel sera placé dans une FILE D'ATTENTE POUR MESSAGE. Après quoi le « ROVIN-VMS » transformera le MESSAGE en un EVENEMENT fonctionnel que l'utilisateur pourra exploiter.

LE ROVIN™ ne dispose que d'une seule fonction EVENEMENT, ce qui veut dire que ce dernier ne dispose que d'un seul VECTEUR D'EVENEMENT. Cette méthode est utilisée par certains microcontrôleurs ou processeurs tels que les PIC™ et les ARM™ ; Méthode qui, si elle n'est pas la plus efficace permet de béféficier d'une plus grande flexibilité dans le contrôle des EVENEMENTS prioritaires. L'utilisateur pourra ainsi lire le MESSAGE dans la FILE D'ATTENTE DE MESSAGES lorsqu'un EVENEMENT surviendra et pourra trouver de quel sorte d'EVENEMENT il s'agit. L'utilisateur n'aura pas à se préoccuper des registres qui seront créés lors de l'appel d'un EVENEMENT, du fait que tout est géré en interne pour vous. Vous trouverez ainsi une librairie complète de fonctions capables de gérer ces MESSAGES.

FILE D'ATTENTE POUR MESSAGE

MSGQUEUE – Le buffer FIFO dispose de 512 octets. Chaque message prend 1 octet. L'utilisateur devra utiliser les librairies fournies et chaque TACHE utilisera la même file d'attente pour messages (MSGQUEUE).

Avec 512 octets d'espace mémoire, le ROVIN™ dispose de toute la place nécessaire pour gérer les EVENEMENTS les plus complexes. Comme c'est le cas avec la plupart des processeurs, UN EVENEMENT ne peut pas survenir à l'intérieur d'un autre EVENEMENT en cours. En d'autres termes, une fois qu'une fonction EVENEMENT est exécutée, si un autre EVENEMENT survient ce dernier sera stocké dans la file d'attente de messages (MSGQUEUE) en tant que MESSAGE (sans que le module ne saute directement à cet EVENEMENT).

Les MESSAGES D'EVENEMENTS sont stockés dans l'ordre de leur création, puis récupérés dans le même ordre du fait que la file d'attente des messages (MSGQUEUE) soit de type FIFO (First-In-First-Out). L'utilisateur ne pourra pas changer cet ordre. Notez qu'un dépassement (overflow) peut intervenir si la vitesse de traitement des MESSAGES est inférieure à celle du remplissage de la file d'attente des messages (MSGQUEUE). Si un dépassement intervient (overflow), plus aucun message ne pourra être admis dans la file d'attente des messages. En mode simple tâche, les cas de dépassement sont très rares. En mode MULTITACHES, les dépassements (overflow) auront pour conséquence de ralentir la vitesse d'exécution.

FONCTION EVENEMENT

EVENT Fonction format

```
void IRQ_EVENT_(void)
{
}
```

IRQ_EVENT_ est une fonction réservée. Vous ne devrez pas utiliser d'autre nom pour la fonction EVENEMENT.

Exemple:

```
Void IRQ_EVENT_(void)
{
unsigned char MSG;

  //--- Récupération d'un MESSAGE EVENEMENT.
MSG = GetMsg();
switch(MSG)
  {
    case MSG_UART0RX : <Appel la Fonction appropriée>
                           break;
    case MSG_EXINT0  : < Appel la Fonction appropriée >
                           break;
    ...
 }
```

La fonction **GetMsg()** récupère un MESSAGE de la file d'attente des messages (MSGQUEUE). Une fois récupéré, le message est totalement effacé la file d'attente des messages. Comme indiqué ci avant, si vous ne récupérez pas les MESSAGES de la file d'attente, celle-ci risque de dépasser et plus aucun EVENEMENT ne pourra être géré.

Avant de pouvoir utiliser les EVENEMENTS, vous devez au préalable activer ceux-ci avec la fonction **EventOn()**.

ACTIVER LES EVENEMENTS

Les fonctions RTC, ALARM, VMTIMER, ANCOMP, EXINT, UART, et TIMER disposent de leur propre EVENEMENT. Mais afin que chaque EVENEMENT puisse être opérationnel, l'EVENEMENT devra être activé avec la fonction **EventOn()** .

Utilisez les fonctions **EventOn()** et **EventOff()** pour activer / désactiver tous les EVENEMENTS.

LIBRAIRIES

L'environnement « ROVIN-C » dispose de librairies EVENEMENTS.

Fonctions supportées:

void EvtnOn(void);
void EventOff(void);
unsigned char GetMsg(void);
void ClearAllMsg(void)
unsigned char CheckMsg(void);

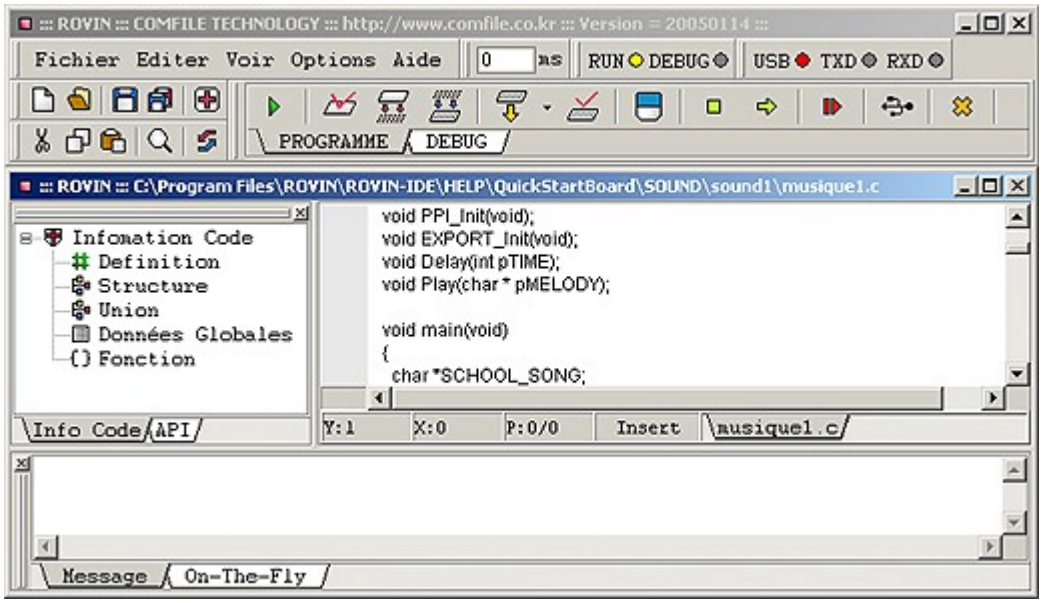
Constantes supportées:

MESSAGE

```
[ MSG NULL         : MESSAGE NUL.
[ MSG ALARM        : APPARITION EVENEMENT ALARM.
[ MSG_VMTIMER      : APPARITION EVENEMENT VMTIMER.
[ MSG_ANCOMP       : APPARITION EVENEMENT ANCOMP.
[ MSG_EXINT0       : APPARITION EVENEMENT EXINT0.
[ MSG EXINT1       : APPARITION EVENEMENT EXINT1.
[ MSG EXINT2       : APPARITION EVENEMENT EXINT2.
[ MSG EXINT3       : APPARITION EVENEMENT EXINT3.
[ MSG EXINT4       : APPARITION EVENEMENT EXINT4.
[ MSG EXINT5       : APPARITION EVENEMENT EXINT5.
[ MSG_EXINT6       : APPARITION EVENEMENT EXINT6.
[ MSG_EXINT7       : APPARITION EVENEMENT EXINT7.

[ MSG UART0RX      : UART0 RECEIVE EVENT OCCURENCE.
[ MSG UART1RX      : UART1 RECEIVE EVENT OCCURRENCE.
[ MSG RTC           : RTC EVENT OCCURRENCE.
[ MSG TIMER0       : TIMER0 EVENT OCCURRENCE.
[ MSG TIMER1       : TIMER1 EVENT OCCURRENCE.
[ MSG_TIMER2       : TIMER2 EVENT OCCURRENCE.
[ MSG_TIMER3       : TIMER3 EVENT OCCURRENCE.
[ MSG_COUNTER0     : COUNTER0 EVENT OCCURRENCE.
[ MSG COUNTER1     : COUNTER1 EVENT OCCURRENCE.
[ MSG COUNTER2     : COUNTER2 EVENT OCCURRENCE.
[ MSG CAPTURE0     : CAPTURE0 EVENT OCCURRENCE.
[ MSG CAPTURE1     : CAPTURE1 EVENT OCCURRENCE.
```

L'INTERFACE DE DEVELOPPEMENT « ROVIN-IDE »



Disponible en Coréen, en Anglais et en Français (traduction effectuée par la société Lextronic pour cette dernière version), l'environnement "ROVIN-IDE" développé par Comfile Technology™ intègre un compileur "C" (ROVIN-C) ainsi qu'un puissant module de DEBUG qui vous permettrons ainsi de bénéficier d'une plate-forme de développement complète.

Les différentes versions de l'environnement de développement "ROVIN-IDE" sont librement téléchargeables depuis le site www.comfile.co.kr . L'installation du "ROVIN-IDE" est extrêmement simple (ce dernier intègre également une aide en ligne très complète).

Dès lors en utilisant le "ROVIN-IDE", il vous sera possible d'écrire vos programmes en langage « C », de les télécharger dans le module et d'effectuer du «Débug » en langage « C » via le câble USB dédié à cet usage.

IMPORTANT

Configuration nécessaire avant d'utiliser le « ROVIN-IDE »

Le programme « ROVIN-IDE » nécessite que vous reconfiguriez la façon dont les nombres à virgules devront être interprétés.

Pour ce faire, sans qu'aucun programme ne soit en cours d'exécution, sélectionnez dans le menu de Windows™ :

« Démarrer » -> « Paramètres » -> « Panneau de configuration ».

Une fois le panneau ouvert, double-cliquez sur :

« Options régionales et linguistiques »

Cliquez ensuite sur le bouton « Personnaliser… »

A l'emplacement « Symbole décimal : »
Mettez un « . » (un point) à la place de la « , » (virgule).

Cliquez ensuite sur le bouton « OK ».

Puis à nouveau sur le bouton « OK » pour fermer la boîte de dialogue.

Refermez également la fenêtre du « panneau de configuration ».

A ce stade, vous êtes prêt à pouvoir utiliser correctement l'environnement de développement « ROVIN-IDE » en exécutant ce dernier depuis :

« Démarrer » -> « Programmes » -> « ROVIN » -> « ROVIN-IDE »

Le « ROVIN-IDE » s'utilise sous environnement window™ XP. Chaque barre d'outils et chaque fenêtre pourront dès lors être déplacés et positionnés selon vos préférences. Ainsi la barre d'outils de l'éditeur de code initialement positionnée en haut à gauche pourra être déplacée tout en haut de l'écran ou complètement à gauche selon vos habitudes de travail. Le « ROVIN-IDE » vous permettra d'ouvrir de multiples fichiers « source C » afin que vous puissiez les compiler et les « débugger » indépendamment les uns des autres et tout ceci, sans quitter l'environnement principal.

Barre d'outils de l'éditeur

Barre d'outils accessible en mode « Débug »

Ces fenêtres correspondent à celles disponibles lors de l'exécution du « ROVIN-IDE » (ROVIN.EXE). La plupart des fonctions du « ROVIN-IDE » sont accessibles au moyen de boutons (certains d'entre eux étant « désactivés » en fonction de l'état de vos travaux).

Menu

Le menu principal du « ROVIN-IDE » comprend les sélections : Fichier, Editer, Voir, Options et Aide. La sélection « Voir » vous permettra d'ouvrir des fenêtres de visualisation spéciales (initialement fermées afin de ne pas surcharger l'écran). La sélection « Options » vous permettra entre autre de créer des fichiers « Log » pendant les phases de « débuggage » ou encore de configurer les fontes et les couleurs des points d'arrêt et de l'éditeur principal.

La sélection « Aide » vous permettra d'avoir accès non seulement à ce manuel, mais également à une version simplifiée « html » et surtout à une rubrique d'exemples de programmes extrêmement complète et détaillée.

>> Détail des sélections du menu (Fichier)

Ce menu permet de créer, d'ouvrir ou d'enregistrer des fichiers sources en 'c' ou des projets complets.

(Nouveau Fichier) Permet de créer un fichier source 'c' en vue de son exécution sur le module ROVIN™.

(Ouvrir) Ouvre un fichier source '*.c' existant. Plusieurs fichiers peuvent être ouverts en même temps.

(Sauver) Sauver le fichier source sélectionné et uniquement celui-ci.

(Tout Sauver) Sauver tous les fichiers sources ouverts.

(Sauver Sous) Sauver le fichier source sélectionné sous un nom différent.

(Ouvrir Projet) Ouvre un fichier Download-Manager (*.dwn). Tous les fichiers sources ' *.c' associés au fichier Download- Manager seront également automatiquement ouvert.

(Sauver Projet) Sauve le fichier Download-Manager en cours d'utilisation ainsi que les fichiers sources 'c' associés.

Attention notez que tous les changements apportés sur vos fichiers seront auto-matiquement sauvegardés lorsqu'un fichier Download-Manager ' *.dwn' est fermé (même si vous n'utilisez pas la fonction 'Sauver Projet').

(Sauver LogFile) Sauve le contenu de la fenêtre « On-The-Fly » dans un fichier log (*.log).

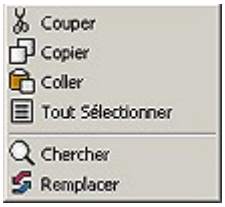
(Imprimer) Imprime le contenu de la fenêtre sélectionnée.

(Configurer Imprimante) Permet la configuration de l'imprimante.

(Mise à jour Firmware) Permet la mise à jour du Firmware du module ROVIN™ Vous pouvez télécharger la dernière version sur le www.comfile.co.kr.

(Sortir) Quitter l'environnement « ROVIN-IDE ».

>> Détail des sélections du menu (Editer)



(Couper) Couper le texte sélectionné.

(Copier) Copier le texte et le place dans le « presse-papiers ».

(Coller) Récupère le texte du « presse-papiers ».

(Tout sélectionner) Sélectionne tout le texte.

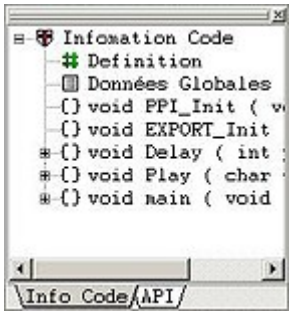
(Chercher) Cherche une expression dans le texte.

(Remplacer) Remplace une expression dans le texte.

>> Détail des sélections du menu (Voir)

Ce menu permet d'afficher différentes fenêtres dans l'environnement de développement. Certaines de ces fenêtres sont déjà pré-affichées dès que vous ouvrez un fichier source.

(Fenêtre Code Explorer)

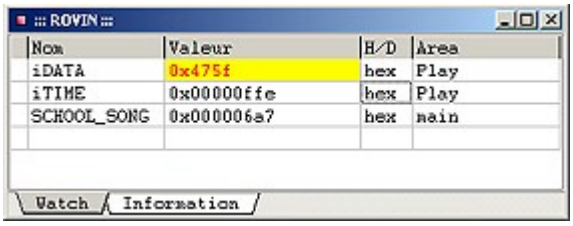


Cette sélection permet d'afficher la fenêtre « Explorateur de code » qui permet (APRES une compilation) d'afficher la liste des fonctions et des variables utilisées dans votre programme. En cliquant sur l'une d'entres-elles, vous vous retrouverez directement « dessus » au niveau de l'éditeur de code. Un onglet « API » au bas de la fenêtre permet d'avoir un accès à un rappel sur la syntaxe des fonctions reconnues par le « ROVIN-C ».

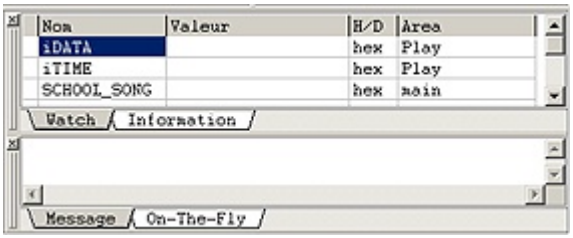
(Fenêtre Message)

Cette sélection affichage de la fenêtre des messages (cette dernière est normalement pré-affichée d'office à l'écran). Elle permet de connaître le résultat des actions effectuées(erreurs de compilation, téléchargement vers le ROVIN™ réussi… Vous pouvez effacer le contenu de la fenêtre par un click droit de souris). Un onglet « On-The-Fly » au bas de la fenêtre est automatiquement utilisé en mode DEBUG (voir description ci-après).

(Fenêtre Watch)

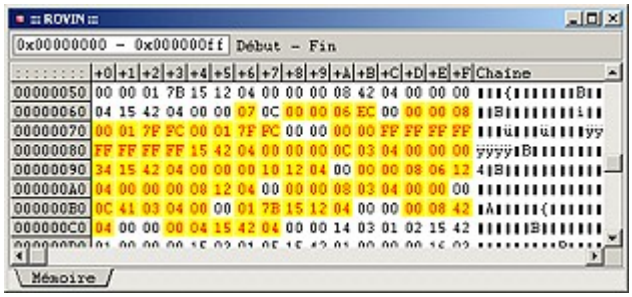


Cette sélection permet d'afficher la fenêtre « Watch ». Cette dernière vous permettra (en mode DEBUG) de visualiser/modifier les variables de votre programme. La fenêtre peut également être accolée à la fenêtre « Message » en cliquant sur son cadre bleu (et tout en restant appuyé sur le bouton gauche de la souris) en déplaçant le cadre à l'intérieur de la fenêtre « Message » puis en relâchant le bouton de la souris.



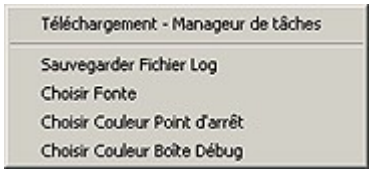
Pour ajouter une variable dans la fenêtre il faut (APRES avoir compilé le programme), surligner la variable en question dans l'éditeur de code en restant appuyé sur le bouton gauche de la souris et en « passant » sur toutes les lettres de la variable (comme par exemple iTIME), puis relâcher le bouton de la souris et réaliser un click droit.

Puis sélectionnez 'Ajouter fenêtre Watch' . Si la variable n'est pas acceptée, c'est que soit le programme n'a pas été compilé, soit que la variable n'est pas une variable ! (ou qu'elle n'a pas été soulignée complètement ou avec des lettre en plus).
(Fenêtre Mémoire)

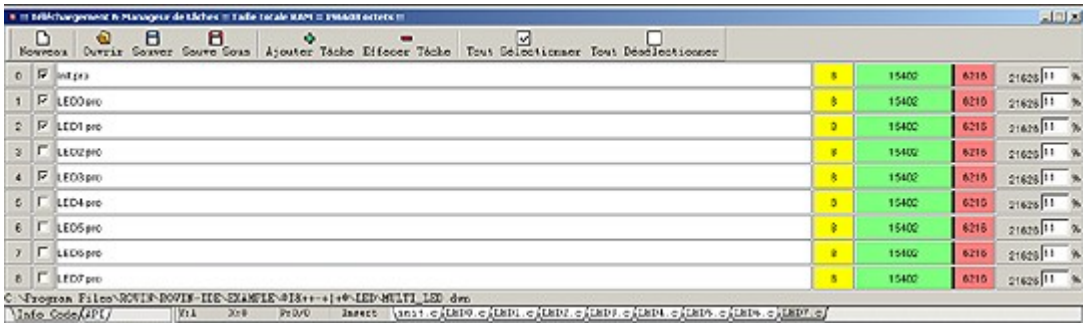


Cette sélection permet l'affichage de la fenêtre « Mémoire » (pérationnelle durant le mode DEBUG). Il vous sera possible si vous le désirez, d'attraper celle-ci par son cadre bleu afin de la placer dans la fenêtre « Message » au bas de l'écran. Saisissez la plage mémoire à visualiser (début – fin) dans le cadre prévu à cet effet en haut de l'écran de la fenêtre (référez-vous aux différents types (RAM / EEPROM….) et emplacements mémoires respectifs donnés au début du manuel.

>> Détail des sélections du menu (Option)



(Téléchargement – Manageur de tâches)

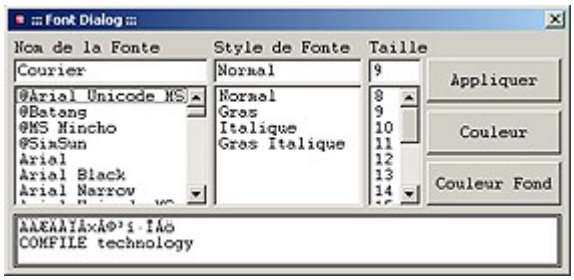


Cette sélection permer l'affichage de la fenêtre « Téléchargement – Manageur de tâches » (dont le fonctionnementest détaillé ci-après).

Les réglages qui suivent vous permettrons de personnaliser l'éditeur du « ROVIN-IDE ». Ces derniers seront réutilisés automatiquement lors de la prochaine exécution du « ROVIN-IDE ».

(Sauvegarder Fichier Log)
Cette option permet d'autoriser la sauvegarde du contenu de la fenêtre 'On-The-Fly' dans un fichier 'Log'. Veuillez surveiller tout de même la taille de ce fichier si la fenêtre 'On-The-Fly' est souvent utilisée.

(Choisir fonte)



Permet la sélection de la couleur des fontes et du fond de l'éditeur (ne fonctionne que si vous avez ouvert un fichier source '*.c'). Attention, conservez IMPERATIVEMENT une fonte de taille 9 sans quoi des problèmes d'affichage surviendront lors des phases de debuggage.

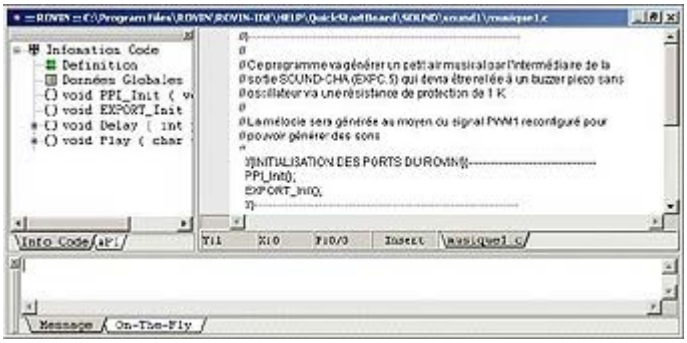
(Choisir couleur Point d'arrêt)
Permet de choisir la couleur des points d'arrêts du mode Débug.

(Choisir couleur Cadre DEBUG)
Permet de choisir la couleur de la boîte de Débug.

>> Détail de la sélection du menu (Aide)

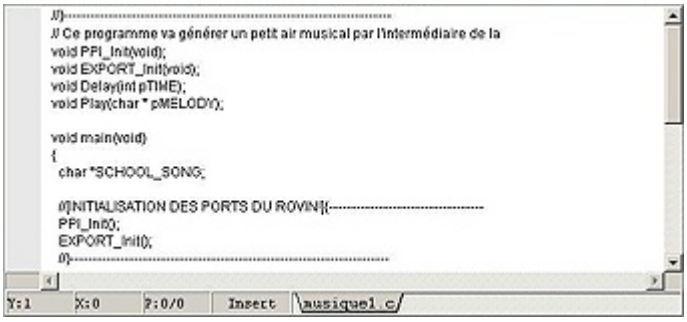
Permet d'afficher l'aide du ROVIN.

A PROPOS DU MODULE « EDITEUR » DU « ROVIN-IDE »



Le module « Editeur » du « ROVIN-IDE » se compose des fenêtres « Editeur de code source », « Explorateur de code » et « Messages ». Toutes ces fenêtres initialement accolées les unes aux autres peuvent être librement déplacées et positionnées selon vos préférences.

(La fenêtre « Editeur de code source »)



La fenêtre « Editeur de code source » vous permettra de saisir votre programme. Le bas de la fenêtre fait apparaître la rangée et la colonne où se trouve le curseur ainsi que le mode de saisie (Insert/Overwrite). Il est possible d'ouvrir simultanément jusqu'à 100 fichiers sources, sélectionnables à partir d'onglets au bas de la fenêtre.

>> Détail des autres boutons et sélection du « ROVIN-IDE »

Durant le mode DEBUG, il est possible de déterminer une durée générée en mode « pas-à-pas Automatique » (exprimée en milisecondes). Vérifiez que la touche « Enter » soit pressée après avoir choisi la valeur.

Ces icones vous afficherons automatiquement l'état du ROVIN™ (mode RUN ou DEBUG). Aucune sélection n'est possible sur ces icônes.

Le 'voyant ' USB indique la présence du câble USB du module ROVIN. Ce dernier est opérationnel lorsque le voyant est rouge. Si le voyant n'est pas allumé, cela signifie que le câble n'est pas connecté ou que vous n'avez pas utilisé le bon driver (aucun dialogue avec le module ROVIN™ ne sera possible). Le voyant vert TXD et le voyant jaune RXD s'allument quant à eux de façon fugitive lors des échanges d'informations et du pilotage du ROVIN™ via l'interface de développement « ROVIN-IDE ».

Vous disposez également d'un menu dont les icônes vous fournissent un « raccourci » de la plupart des options du menu Fichier. Ces derniers vous aideront à travailler encore plus rapidement.

Un second menu permet de disposer d'icônes offrant un raccourci de la plupart des options du menu Editer.

Ces boutons accessibles avec l'onglet «PROGRAMME » (lorsqu'un fichier source est chargé dans la fenêtre de l'éditeur) permettent d'avoir accès à la plupart des commandes du compilateur présent dans le « ROVIN-IDE ».

(Cycle Automatique)
Permet de réaliser automatiquement le processus :
Vérifier Syntaxe -> Compiler -> Linker -> Télécharger -> Programme -> Vérifier.

Ce bouton est très utile lors des phases de modification et de mise au point de votre programme puisqu'il vous permettra d'automatiser le processus de traitement de l'environnement « ROVIN-IDE ».

(Vérifier syntaxe)
Permet de vérifier la syntaxe du programme dont l'onglet a été sélectionné en accord avec les règles grammaticales du langage « ROVIN-C ».

(Compiler)
Permet de vérifier la syntaxe, puis de compiler le programme afin de pouvoir créer un code « objet ». Le fichier exécutable n'est pas encore créé à ce stade.

(Linker)
Permet de vérifier la syntaxe, puis de compiler le programme (afin de pouvoir créer un code « objet »), puis de réaliser le fichier exécutable/ téléchargeable (*.pro) et le fichier nécessaire au mode Débug (*.dbg).

(Télécharger programme(s))
Permet de télécharger le(s) fichier (*.dwn) dans le module ROVIN™. Vous pouvez utiliser ce bouton si n'avez pas modifié le code source et si vous voulez juste télécharger le programme dans le ROVIN™ (pour la production en série par exemple).

(Verifier)
Permet de vérifier si le programme a été correctement téléchargé dans le module ROVIN™.

(Effacer Flash du ROVIN)
Permet d'effacer tous les programmes de la mémoire Flash du ROVIN™.

Une fenêtre vous demandera de confirmer l'opération afin d'éviter toute mauvaise manipulation involontaire.

(Stopper ROVIN !)
Stoppe l'éxécution du (des) programme(s) du ROVIN™.

(Démarrer ROVIN !)
Démarre l'éxécution du (des) programme(s) du ROVIN™ si elle a été stoppée.

 **(Reset)**

Permet d'avoir le même effet que le bouton RESET (physique) du module ROVIN™. Tout est initialisé (l'ensemble des points d'arrêt sont perdus) et tous les programmes en mémoire redémarrent depuis le début).

 **(Initialiser Cable USB)**

Permet d'initialiser le câble USB du ROVIN™ (si ce dernier a été déconnecté par exemple en ours d'utilisation). Le voyant rouge USB doit être allumé lorsque le câble est correctement détecté et prêt à l'emploi.

 **(Stopper Communication)**

Stoppe immédiatement la communication entre le module ROVIN™ et l'environnement de développement « ROVIN-IDE ».

NOTA : Une fois sollicité, ce bouton **RESTE ENFONCE**. Il devra donc être sollicité à nouveau avant de pouvoir réaliser une autre opération (sans quoi les commandes n'auront aucun effet).

ATTENTION lorsqu'une erreur de traitement intervient (par exemple si vous essayez de télécharger un programme qui n'existe pas ou qui n'est pas présent dans le répertoire attendu), l'environnement « ROVIN-IDE » stoppe automatiquement la communication avec le ROVIN™ et « enfonce » le bouton « Stopper Communication ».

Dès lors, si lorsque vous essayez d'effectuer des actions sur le ROVIN™, ce dernier ne semble pas réagir, vérifiez si le bouton « Stopper Communication » n'est pas enfoncé… auquel cas, cliquez dessus pour le relâcher !

>> Les commandes du mode DEBUG

Ces boutons sont disponibles à partir de l'onglet DEBUG. Le fichier téléchargeable relatif à votre programme ainsi que celui dédié au mode Débug doivent avoir été générés pour pouvoir activer cet onglet et exploiter ces boutons (consultez l'aide en ligne de l'environnement de développement « ROVIN-IDE » qui détaille toutes ces procédures).

 **(Stop)**

Stoppe l'éxécution de la tâche dont le fichier source a été sélectionné dans l'éditeur de code. Si il y a plusieurs tâches en cours d'exécution, les autres tâches ne seront pas affectées et continueront de s'exécuter normalement.

 **(Run)**

Reprend l'exécution de la tâche à l'endroit où elle a été stoppée.

 **(1 pas)**

Exécute une ligne de code à chaque solicitation.

 **(Pas-à-pas automatique)**

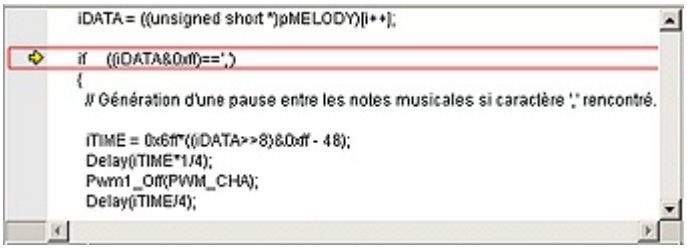
Exécute les lignes de code de la tâche sélectionnée avec un delai entre chaque ligne. Attention, une fois sollicitée, la touche reste enfoncée, il faut la sollicité à nouveau (pour la libérer) si on désire réaliser une autre action.

Le délai est réglable en milisecondes.

 **(Reset Tâche)**

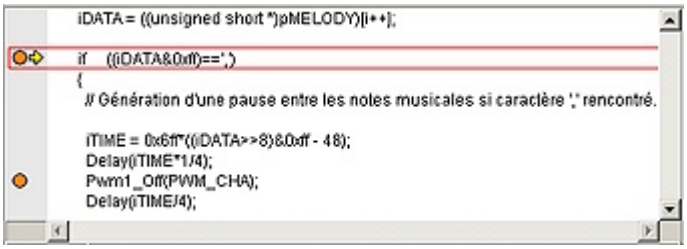
Réinitialise la tâche au début du code (la tâche n'est pas exécutée, le pointeur est juste remis au début du code).

>> Exploitation du mode DEBUG



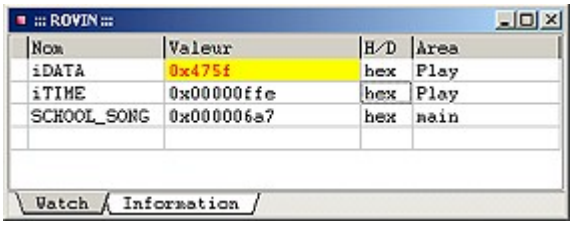
La fenêtre de DEBUG

La fenêtre de DEBUG affiche la ligne courante du code qui va être exécutée. Cette dernière est entourée d'un cadre rouge (il est possible de modifier la couleur de ce cadre dans le menu (Options) -> (Choisir couleur cadre DEBUG).

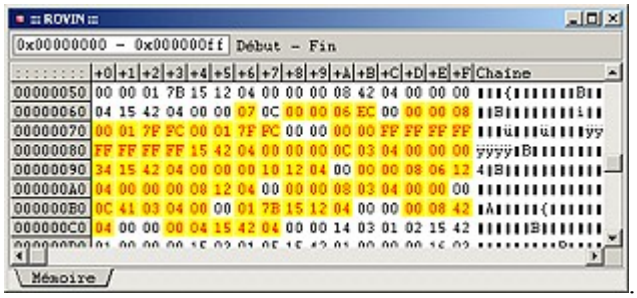


Les points d'arrêt

Il est possible d'utiliser jusqu'à 10 points d'arrêts au sein d'une tâche. Pour mettre/enlever un point d'arrêt, il suffit de cliquer à gauche de la ligne d'instruction concernée pour faire apparraître/disparaître un point d'arrêt orange (la couleur de ce dernier est modifiable dans le menu (Options) -> (Choisir Couleur Point d'arrêt). Lorsque la tâche est en cours d'exécution et qu'elle rencontre un point d'arrêt, cette dernière stoppe son exéction et se positionne sur la ligne en question dans l'attente de vos commandes.



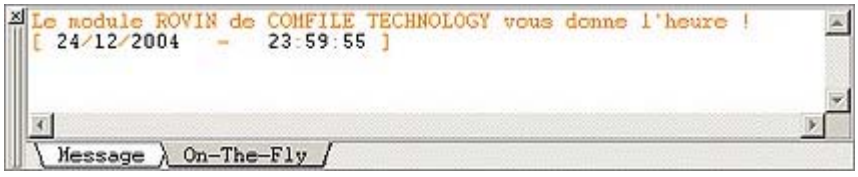
A noter que pendant le mode « pas-à-pas » et « pas-à-pas Automatique », il vous sera possible de visualiser l'évolution des variables déclarées dans la fenêtre « Watch ». En mode « pas-à-pas », il vous sera même possible de modifier la valeur des variables avant l'exécution de la prochaine instruction. On remarquera que seules les variables affectées par un changement d'état au cours de l'exécution de l'instruction sont en vidéo inverse jaune.



A noter que pendant le mode « pas-à-pas » et « pas-à-pas Automatique », il vous sera possible de visualiser l'évolution des données de la fenêtre « Memoire » où seules les données affectées par un changement d'état au cours de l'exécution de l'instruction sont en vidéo inverse jaune.

Note concernant l'utilisation des modes : « pas-à-pas » et « pas-à-pas Automatique »

Attention, lors de l'utilisation de ces boutons, vous pouvez être amené à croire que ces derniers ne fonctionnent pas correctement du fait qu'aucune action ne se déroule à l'écran. Ceci peut être le cas si vous passez sur une instruction nécessitant normalement un nombre important de « cycles machines » avant d'être exécutée : comme par exemple l'instruction while(pTIME--) ; Pour vous en assurer ajoutez la variable pTIME dans la fenêtre Watch et vous verrez effectivement que l'instruction while(pTIME--) est bien en train de s'exécuter à chaque sollicitation des boutons « pas-à-pas » ou « pas-à-pas automatique », même si rien ne semble se passer dans l'évolution de la fenêtre de DEBUG.



La fenêtre « On-The-Fly »

La fenêtre « On-The-Fly » dispose de fonctionnalités similaires à celles que l'on peut trouver sur des émulateurs professionnels. Cette dernière vous permettra ainsi de "remonter" des informations de la part du module ROVIN™ via le câble USB à partir d'instructions que vous pourrez disséminer au sein de votre code.

DebugPrint Affiche des caractères issus du ROVIN™ dans le fenêtre Debug du PC
DebugClear Efface le contenu de la fenêtre Debug du PC
DebugCHAR Affiche variable char / unsigned char (hex ou déc.) dans fenêtre Débug
DebugSHORT Affiche variable short / unsigned short char (hex ou déc.)
DebugINT Affiche variable int / unsigned int char (hex ou déc.) dans fenêtre Débug
DebugLONG Affiche variable long / unsigned long char (hex ou déc.) dans fenêtre Débug
DebugFLOAT Affiche variable float char (en hexa ou décimal.) dans fenêtre Débug DebugLONG
Affiche variable Double char (en hexa ou décimal) dans fenêtre Débug

Ainsi à tout moment ce dernier pourra lors de l'exécution d'une tâche envoyer des messages ASCII ou des variables dans la fenêtre « On-The-Fly ». Dans l'exemple ci-dessus, le module ROVIN™ envoie le message ASCII "**Le Module ROVIN de COMFILE TECHNOLOGY vous donne l'heure !**", puis ce dernier envoi des variables (issues de son horloge temps réel). L'ensemble s'affiche alors en boucle dans la fenêtre "On-the-Fly" du PC. On notera que les couleurs des caractères et des variables sont différentes afin de simplifier leur repérage. Vous pourrez ainsi créer des fenêtres de DEBUG entièrement personnalisées, lesquelles pourront être automatiquement sauvegardées dans un fichier « Log » afin de pouvoir étudier leur évolution après coups (Prenez toutefois garde à la taille de ce fichier si la fenêtre est susceptible d'évoluer souvent). Vous pouvez également effacer manuellement le contenu de la fenêtre « On-The-Fly » en réalisant un click droit dessus.

>> La fenêtre « Téléchargement et Manageur de tâches »

Accessible soit depuis le menu (Options) -> (Téléchargement – Manageur de tâches) ou par la petite flèche pointant vers le bas (à droite du bouton « Télécharger Programme »), la fenêtre « Téléchargement et Manageur de tâches » permet de sélectionner les programmes (tâches) qui devront être transférés au sein du module ROVIN afin de pouvoir être exécutés simultanément. Ces programmes sont dotés d'une extension (*.pro).

La liste des programmes en question sera sauvegardée dans un fichier de téléchargement avec l'extension (*.dwn). Si vous venez de démarrer un programme et que vous ne l'avez pas encore enregistré sous la forme d'un projet, la fenêtre « Téléchargement et Manageur de tâches » est vide.

Le chapitre ci-dessous donne une description « sommaire » des différents boutons de la fenêtre, consultez le chapitre « Méthodes de travail » pour avoir une idée précise sur la façon dont la fenêtre doit être précisément exploitée.

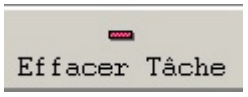
Permet de créer un fichier de téléchargement (*.dwn) – comprenant la liste des programmes (*.pro) à transférer dans le module ROVIN™. Si le fichier a déjà été sauvegardé, il sera sauvé à nouveau automatiquement (sinon une fenêtre de sauvegarde s'ouvrira).

Ouvre un fichier de téléchargement existant (*.dwn). Tous les programmes présents dans le fichier de téléchargement doivent exister et être présents dans le même répertoire, sans quoi une erreur interviendra lors du traitement.

Sauvegarde le fichier de téléchargement en cours (*.dwn).

Sauvegarde le fichier en cours (*.dwn) sous un nom différent.

Ajoute une tâche (un programme) de type (*.pro) à télécharger au sein du module ROVIN™ dans la liste du fichier de téléchargement. Tous les programmes doivent exister et être présents dans le même répertoire que le fichier de téléchargement.



Efface de la liste la tâche sélectionnéée.

Sélectionne tous les programmes afin de les ajouter à la liste.

Enlève tous les programmes de la liste.

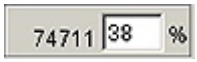
Lorsque vous sélectionnez les tâches (programmes) qui devront être téléchargés au sein du module ROVIN™ (avec le bouton « Ajouter Tâche » , celles-ci s'affichent les unes au dessus des autres avec divers paramètres.

Cette case renferme le N° de la tâche attribué arbitrairement par le « ROVIN-IDE ». Il est possible de télécharger jusqu'à 10 tâches max. au sein du module ROVIN™. Vous ne pouvez pas modifier ces numéros.

Cette case permet d'inhiber le transfert de la tâche qui y est associée (si la case n'est pas cochée) ou d'autoriser son transfert. Ceci idéal pour faire différents essais avec plus ou moins de tâches simultanément.

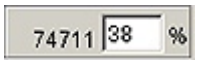
Cette case contient le nom de la tâche (du programme). Ce dernier dispose d'une extension (*.pro). En cliquant dessus vous ferez apparaître le nom complet (avec l'emplacement sur le disque dur).

La taille de l'emplacement globale du programme est affichée ici.



Cette case vous permettra de déterminer la repartition mémoire qu'il faudra faire entre toutes les tâches. Ce paramètre indispensable en mode multitâches devra également être configuré en mode « simple » tâche (la tâche disposera alors de 100 % de la capacité).

Lorsque vous modifierez la répartition mémoire (en pourcentage) attribuée à une tâche, cela affectera directement la capacité mémoire mise à sa disposition. Cette capacité mémoire est matérialisée par 2 cases de différentes couleurs. Celle pour la mémoire destiné à un usage 'standard' (verte) et celle (rouge) utilisée pour l'exploitation d'interruptions (EVENEMENTS). En fonction de la nature de votre tâche, il sera nécessaire d'effectuer un réequilibrage du ratio entre les 2 types de mémoire en déplaçant la barre de séparation centrale vers la droite ou vers la gauche. Si par exemple votre tâche n'utilise aucune interruption (EVENEMENT), il vous faudra faire en sorte que la barre verte soit plus importante que la barre rouge. Dans tous les cas, ne mettez jamais une des 2 barres à 0 (le minimum requis est de 500 octets).



Pour information : la capacité mémoire indiquée à gauche de la répartition en pourcentage correspond au calcul suivant:

74711 = 8 (Taille emplacement globale) + 37351 (Mémoire usage standard) + 37352 (Mémoire usage d'interruption)

Les Bases

Type de données

Le « ROVIN-C » supporte 6 types de données.

Type	Taille octets	Plage
char	1	-128 <= X <= 127
unsigned char	1	0 <= X <= 255
short	2	-32,768 <= X <= 32,767
unsigned short	2	0 <= X <= 65,535
int	4	-2,147,483,648 <= X <= 2,147,483,647
unsigned int	4	0 <= X <= 4,294,967,295
long	8	-923372036854775807 <= X 9223372036854775807
unsigned long	8	0 <= X <=18,446,744,073,709,551,615
float	8	3.37*10-4932 < |X| < 1.18*104932
double	8	3.37*10-4932 < |X| < 1.18*104932

Type numérique :

Décimal

Les nombres 0 ~ 9 peuvent être utilisés.

Héxadécimal

Le préfixe “0x” doit être utilisé. 0 ~ 9 et a ~ f peuvent être utilisés.

Binaire

Le préfixe “0b” doit être utilisé. 1 ou 0 peuvent être utilisés.

Exemple:

“123” est un Décimal
“0xa5” est un Héxadécimal
“0b1001” est un Binary

Déclaration des caractères

```
char c;
c = 'a';
c = 0x41;
```

```
long A;
A = 'ABCDEFGH';          // 8 octets
```

Caractères spéciaux

\0	NULL, 0x00, EOF.
\n	LF (Line Feed), 0x0a Nouvelle ligne.
\r	CR (Retour chariot), 0x0d.
\"	"
\'	'
\\	\

Virgule flottante
Près de 15 digits peuvent être utilisés pour les nombres à virgule flottante.

Exemple:
3.14, 0.1, 123.45678
123. & .14 sont traduit en 123.0, 0.14.

DEFINE Constante
#define Nom de la constante Valeur

CHAINES

Exemple:

```
Char    * STR;     //Création pointeur.
short     DATA;

STR = "0123456789";
STR = "ABCDEFG";
STR = "Hello World!";
STR = "Comfile Technology";
DATA = ((short *)"Comfile Technology")[4];
```

COMMENTAIRE

/* */ et // peuvent être utilisés pour en début de ligne de vos commentaires.

Exemple:

```
/* Ceci est
considéré comme un commentaire  */

// Ceci est un commentaire
// Ceci est un autre commentaire
```

MOTS CLES

Mots clés supportés:
char, if, struct, void, short, else, union, return, int, switch, goto, long, case, float, default, continue, double, while, for, do, signed, unsigned, break, include, define, undef

Mots clés supportés non supportés:
auto, extern, static, register, volatile, enum, const, sizeof, typedef

True / False

La valeur de True est 1 – Celle de False est 0.

OPERATEUR

Le « ROVIN-C » utilise des opérateurs compatibles avec le « ANSI-C ». L'ordre de priorité des opérateurs est donné dans le tableau ci-dessous.

Type d'opérateurs et ordre de priorité

Place	Operateur
Haute	[] -> .
	! ~ ++ -- + - * & (typecast)
↑	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
↓	||
Basse	= += -= *= /= %= &= ^= |= <<= >>=

Comment utiliser les operateurs:

->

```
struct dot
{
  int X, Y;
} * DOT;

void main(void)
{
  int x, y;
  x = DOT->X;
  y = DOT->Y;
}
```

•

```
struct dot
{ int X, Y;
} DOT;

void main(void)
{
  int x, y;

  x = DOT.X;
  y = DOT.Y;
}
```

~

```
void main(void)
{
  char A;
  A = 0xf0;
  A = ~A;        // A prend la valeur 0x0f.
}
```

++ & --

Usage opérateur	Calcule opérateur
T = ++OPERANDE;	OPERANDE = OPERANDE + 1; T = OPERANDE;
T = OPERANDE++;	T = OPERANDE; OPERANDE = OPERANDE + 1;

*Note: fonctionne de la même façon avec --

*

```
void main(void)
{
  char A, B, *P;
  A = -1;
  P = &A;
  B = *P;        //B prend la valeur -1.
}
```

&

```
void main(void)
{
  char A, B, *P, ARY[3];
  A = -1;
  P = &A;          //P prend la valeur de l'adresse de A.
  B = *P;          //B prend la valeur -1.
  P = &(ARY[1]); //P prend la valeur de l'adresse ARY[1].
}
```

(type de catégories)

TYPES

DONNES	(char), (short), (int), (long), chacun sont de type signé ou non signé. (float), (double).
POINTEUR	(char *), (short *), (int *), (long), chacun son de type signé ou non signé. (float *), (double *)

```
void main(void)
{
  char      A, *P;
  unsigned int I;

  I = (unsigned int)A;//Change A en int. Non signé.
  I = (unsigned int)P;//Change P en int.
  A = *((char *)I);   //Change I en caractère pointeur
//et lit la valeur à nouveau.
}
```

%

% Calcule les restes après la division.
Le résultat suite le plus grand nombre de A et B.

```
void main(void)
{
  int A, B, C;

  A = 7;
  B = 2;
  C = A % B;          //C prend la valeur 1 andis que 7/2 = 3
  //et 1 est le reste.
}
```

<<

A << B

<< opérateur de décalage du nombre A, de B bits vers la gauche.
Le « ROVIN-C » peut décaler 64 bits (8 octets).
Lorsque les bits sont décalés, les nouveaux bits apparaissant à droite seront à 0.

```
void main(void)
{
  unsigned char   V;

  V = 0x01;
  V = V << 3;   //V becomes 0x08.
}
```

>>

A >> B

>> opérateur de décalage du nombre A, de B bits vers la droite.
Le « ROVIN-C » peut décaler 64 bits (8 octets).
Lorsque les bits sont décalés, les nouveaux bits apparaissant à droite seront à 1.

```
void main(void)
{
  unsigned char   V;

  V = 0x80;
  V = V >> 3;   //V becomes 0xf0.
}
```

<

A < B

```
void main(void)
{
  unsigned char   R;

  R = 3 < 7;      //R prend la valeur True.
}
```

<=

A <= B

```
void main(void)
{
  unsigned char   R;

  R = 3 <= 3;      // R prend la valeur True.
}
```

>

A > B

```
void main(void)
{
  unsigned char   R;

  R = 3 > 7;      //R prend la valeur False.
}
```

>=

A >= B

```
void main(void)
{
  unsigned char   R;

  R = 7 >= 7;       //R prend la valeur True.
}
```

^

Logic XOR

A ^ B

A	B	A^B
1	1	0
1	0	1
0	1	1
0	0	0

```
void main(void)
{
  unsigned char   A, B, C;

  A = 0xf0;
  B = 0xff;
  C = A ^ B;       //C prend la valeur 0x0f.
  C = A & B;       //C prend la valeur 0xf0.
}
```

+=

```
void main(void)
{
  unsigned char   A;

  A = 5;
  A += 5;      //A = 10.
}
```

-=

```
void main(void)
{
  unsigned char   A;

  A = 5;
  A -= 5;     //A = 0.
}
```

*=

```
void main(void)
{
  unsigned char   A;
  A = 5;
  A *= 5;     //A = 25.
}
```

/=

```
void main(void)
{
  unsigned char   A;
  A = 5;
  A /= 5;     //A = 1.
}
```

%=

```
void main(void)
{  unsigned char   A;
  A = 9;
  A %= 5;     //A = 4.
}
```

&=

```
void main(void)
{
  unsigned char   A;
  A = 0xff
  A &= 0xf0;      //A = 0xf0.
}
```

^=

A ^= B

^= est identique à A= A^B.

```
void main(void)
{
  unsigned char   A;
  A = 0xff;
  A ^= 0xf0;      //A est à 0x0f.
}
```

|=

A |= B

|= est identique à A=A|B.

```
void main(void)
{  unsigned char   A;
  A = 0xf0;
  A |= 0x0f;      //A est à 0xff.
}
```

<<=

A <<= B

<<= est identique à A = A << B

```
void main(void)
{
  unsigned char   A;

  A = 0x01;
  A <<= 3;      //A est à 0x08.
}
```

>>=

A >>= B

>>= est identique à A = A >> B.

```
void main(void)
{
  unsigned char   A;

  A = 0x80;
  A >>= 3;      //A est à 0xf0.
}
```

DECLARATION CONDITIONNELLES

Les déclaration conditionnelles if(), switch(), while(), do(), for(), et goto sont les mêmes que ceux du langage « C » standard.

Les étiquette devront être écrite comme ci-dessous:

```
LabelA:
    <commands>
        .
        .
        .
        .
        .
        goto LabelA;
        .
        .
        .
        .
        .
```

Pré-processeur

Le « pré-processeur » du « ROVIN-C » est utilisé pour inclure des fichiers externes.

Fichiers inclus

```
#include   < FILE-NAME >
#include   " FILE-NAME "
#include   " FILE-PATH "
```

#include Exemple:

```
//Normal.
#include < xlibrary.h >

//Si le fichier est dans le même répertoire que le fichier source.
#include " ABC.h "

// Si le fichier est dans un répertoire différent du fichier source.
#include " C:\Program Files\ROVIN\my\myfile.h "
```

#define

Définir une constante

```
#define   NOM   VALEUR
#define   NOM   CHAINE
#define   NOM   CODE
```

#undef

« Désactive » la définition d'une contsante, le NOM ne sera plus valide.

```
#undef   NOM
```

```
char *dogNAME; //Déclare un pointeur pour stocker une chaîne d'adresse.

#define   DOG_NAME   "Doggy1"
void Print_MyDogName(void)
{
  dogNAME=DOG_NAME;
}

#undef DOG_NAME      //DOG NAME N'EXISTE PLUS

#define DOG_NAME     "Doggy2"   //Re-défini le nom DOG NAME.
void Print_YourDogName(void)
{
dogNAME=DOG_NAME;
}
```

Exemples de déclarations

```
struct point
{ int    X;
  int    Y;
};

void main(void)
{ struct point P;
  P.X = 100;            //X prend la valeur 100.
  P.Y = 50;             //Y prend la valeur 50.
}
```

```
union xdata
{
  char     CHAR;
  short    SHORT;
  int      INT;
  long     LONG8;
  double   DOUBLE;
} XDATA;
union xdata U1,U2,*U3,U4[5];
```

```
union data
{
  long     L;
  double   F;
};

void main(void)
{
  int         iLONG;
union data U,*PU;
U.F = 3.14;
PU = &U;
ILONG = PU->L;
}
```

COMFILE TECHNOLOGY

LA SYNTAXE

AdcSet, AdcRead,
AdcOn, AdcOff

Fonction:
void AdcSet(unsigned char **pMUX**,
unsigned char **pFACTOR**);
unsigned short AdcRead(void);

void AdcOn(void);

void AdcOff(void);

Explication:

AdcSet
Pour utiliser les convertisseurs « A/N », les paramètres **pMUX** et **pFACTOR** devront être configurés.

Le réglage de **pMUX** est destiné à déterminer quelle broche du convertisseur sera utilisée et si un OPAMP sera employé. Le convertisseur « A/N » général utilise les broches ADC0 à ADC7. La fréquence de conversion « A/N » est réglé par le paramètre **pFACTOR**. Vous pouvez généralement utiliser ADCDF_128 pour la valeur de **pFACTOR**.

AdcRead
Lit et retourne la valeur de la conversion « A/N » actuelle (sur 10 bits)
Si vous utilisez un OPAMP, une valeur amplifiée de la conversion sera retournée.

AdcOn
Active le convertisseur « A/N ».

AdcOff
Désactive le convertisseur « A/N ».
Ceci permet d'exploiter toutes les broches de conversion « A/N » en tant qu'entrées/sorties à usage général.

Exemple:

```
void Delay(int pTIME)
{ while(pTIME--);      // Attend un certain temps.
}

void main(void)
{
unsigned short ADC_DATA;  // Conversion sur 10 bits donc declare en short.
PPI_SetMode(PD, 0x00);    // met le port PPI-PD tout en sortie.

DebugClear();

//Configure les valeurs MUX et DIVISION FACTOR pour utilisation « A/N ».
//Utilise la broche ADC0.
//Facteur de division = 128.

AdcSet(ADC0, ADCDF_128);

  AdcOn();     // Active les convertisseurs « A/N ».

  while(1)
{
    ADC_DATA = AdcRead();     // Lecture de la valeur convertie.
    PPI_Out(PD, ~ADC_DATA);   // Sort la valeur convertie sur port PPI-PD.

    // Réalise un voltmètre sur PC en utilisant le mode Débug On-The-Fly.
    DebugClear();
    DebugPrint("La tension sur l'entrée ADC0(EXPA0)est de: ");

    // Conversion valeur sur 10 bits en format décimal.
    DebugDOUBLE((ADC_DATA*5)/pow(2,10),DEC);

    DebugPrint("Volts.\n");

    // Légère tempo nécessaire pour éviter des problèmes d'affichage.
    Delay(0x3ff);
  }
}
```

AncompSet, AncompOn, AncompOff

Fonction:
void AncompSet(unsigned char **pMODE**);

void AncompOn(void);

void AncompOff(void);

Explication:

AncompSet

Configure le mode ANCOMP (voir exemple dans l'aide en ligne du « ROVIN-IDE ».

AncompOn

Active la fonction ANCOMP. Un évènement ANCOMP est seulement valable pour un EVENEMENT. Après cela, l'utilisateur devra à nouveau activer la fonction ANCOMP (du fait qu'elle a été précédemment désactivée). EventOn() doit être appelé afin de pouvoir faire fonctionner correctement la fonction ANCOMP.

AncompOff

Désactive la fonction ANCOMP.
Les broches ANCOMP pourront être utilisées comme E/S générales une fois que la fonction ANCOMP aura été désactivée.

Exemple:

```
// EVENT Fonction call.
void IRQ_EVENT_(void)
{
  if(GetMsg() == MSG_ANCOMP)
  { DebugPrint("Les tensions sont identiques !\n");
     AncompOn();    // EVENEMENT ANCOMP valable que pour un seul EVENEMENT
                    // Active encore ANCOMP pour un nouvel EVENEMENT.
  }
}

void main(void)
{ AncompSet(ANCOMP_TOGGLE);  // Configure le mode ANCOMP.
  AncompOn();                // Active ANCOMP.
  RtcOn();    // Active l'horloge RTC comme ANCOMP dépend de lui.
  EventOn();  // Active tous les EVENEMENTS.

  while(1);   // Attend un évenement.
}
```

AlarmSetTime, AlarmGetTime
AlarmOn, AlarmOff

Fonction:

void AlarmSetTime(unsigned char **pHOUR**,
unsigned char **pMINUTE**,
unsigned char **pSECOND**);

void AlarmGetTime(unsigned char * **pHOUR**,
unsigned char * **pMINUTE**,
unsigned char * **pSECOND**);

void AlarmOn(void);

void AlarmOff(void);

Explication:

AlarmSetTime
Configure l'heure d'alarme. (Hour, Minute, Seconds = **pHOUR, pMINUTE, pSECOND**)

AlarmGetTime
Lecture de l'heure d'alarme via pointeurs.

AlarmOn
Active l'ALARME. L'EVENEMENT D'ALARME sera automatiquement activé.

AlarmOff
Désactive l'ALARME.

Exemple:

```
void IRQ_EVENT_(void) {
  if(GetMsg() == MSG_ALARM) {
     DebugPrint("Joyeux Noel!!!\nMeilleurs vœux !\n\n");
  }
}
void main(void) {
  RtcSetDate(2005,12,24,23,59,50);  // Réglage de l'horloge RTC.
  //Activera l'alarme au bout de 10 sec. Pour le 25 Décembre !
  AlarmSetTime(00,00,00);             // réglage de l'heure d'ALARME.
  RtcOn();
  AlarmOn();
  EventOn();                          // Active tous les EVENEMENTS.
   // Attend un èvenement.
while(1)
}
```

AlcdInit

Fonction:
void AlcdInit (unsigned char **pUARTport**)

Explication:
pUARTport N° du port UART (0 ou 1) sur lequel l'aficheur ALCD est connecté.

Pour rappel, le module ROVIN™ est capable de piloter les afficheurs LCD à commandes séries « ALCD » de Comfile Technology. L'initialisation des afficheurs « ALCD » dure environ 500 ms. Durant cette période, l'afficheur effacera son écran, positionnera son curseur en haut à gauche et désactivera le rétro-éclairage. L'afficheur devra être initialisé au moins une fois avantque vouspuissiez l'utiliser. En cas de problème particulier, il vous sera possible d'effectuer à nouveau cette initialisation.

Exemple:

```
void main(void)
{
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);        // Initialise afficheur « ALCD » connecté à l'UART0.
}
```

AlcdLocate

Fonction:
void AlcdLocate(unsigned char **pUARTport**,
unsigned char **pX**,
unsigned char **pY**);

Explication:

pUARTport	N° du port UART sur lequel l'afficheur ALCD est connecté.
pX	Position « X » du curseur de l'afficheur « ALCD ».
pY	Position « Y » du curseur de l'afficheur « ALCD ».

Définition de la position du curseur de l'afficheur à commandes séries « ALCD ».

AlcdPrint

Fonction:
void AlcdPrint(unsigned char **pUARTport,** char * **pSTR**);

Explication:
pUARTport N° du port UART sur lequel l'afficheur ALCD est connecté.
pSTR Chaîne de sortie.

Affiche un caractère ou une chaîne à l'endroit du curseur.

```
void main(void)
{
char    * iSTR;

  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

AlcdInit(0);
AlcdLocate(0, 0, 0);
AlcdPrint(0,"Module serie « ALCD ».");

iSTR = "COMFILE TECHNOLOGY.";
AlcdLocate(0, 0, 1);
AlcdPrint(0, iSTR);
}
```

AlcdPrintCode

Fonction:
void AlcdPrintCode(unsigned char **pUARTport**, unsigned char **pCODE**);

Explication:

pUARTport	N° du port UART sur lequel l'afficheur ALCD est connecté.
pCODE	Code ASCII à afficheur sur le « ALCD ».

Exemple:

```
void main(void)
{
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);

  Alcdlocate(0, 0, 0);
  AlcdPrint(0,'A');

  AlcdLocate(0, 0, 1);
  AlcdPrinCode(0, 0x41);      // Le code ASCII de A est 0x41.
}
```

AlcdClear

Fonction:
void AlcdClear(unsigned char **pUARTport**);

Explication:
pUARTport N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.

Efface l'écran et place le curseur sur la position (0,0).

Exemple:

```
void Delay(int pTIME){
while(pTIME--);
}

void main(void){
  char  * iSTR;
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

AlcdInit(0);
AlcdLocate(0, 0, 0);
AlcdPrint(0,"Module série « ALCD ».");

iSTR = "COMFILE TECHNOLOGY.";
AlcdLocate(0, 0, 1);
AlcdPrint(0,iSTR);            // Affiche la chaîne via un pointeur.

Delay(0xfff);                 // Durée d'affichage.

AlcdClear(0);                 // Efface l'écran LCD.

AlcdLocate(0, 0, 0);
AlcdPrint(0,"= SITE WEB =");
AlcdLocate(0, 0, 1);
AlcdPrint(0,"http://www.comfile.co.kr/");
}
```

AlcdCursor

Fonction:
void AlcdCursor(unsigned char **pUARTport**, unsigned char **pOF**);

Explication:

pUARTport	N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pOF	ON ou OFF.

Active/désactive le curseur.
Après une initialisation par AlcdInit(), le curseur est activé par défaut, mais vous pouvez le désactiver avec cette fonction.

Exemple:

```
void main(void)
{
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);

  AlcdCursor(0,OFF);  // Fait disparaître le curseur.

  AlcdLocate(0, 0, 1);// Positionne le curseur à un emplacement donné.

  AlcdPrint(0,"EFFACE CURSEUR.");
}
```

AlcdBlink

Fonction:
void AlcdBlink(unsigned char **pUARTport,** unsigned char **pOF**);

Explication:

PUARTport	N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pOF	ON ou OFF

Active/désactive le clignotement du curseur de l'affiheur « ALCD ».

Exemple:

```
void main(void)
{
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);

  AlcdBlink(0, ON);           // Active le clignotement du curseur
}
```

AlcdCgWrite

Fonction:
void AlcdCgWrite(unsigned char **pUARTport**,
unsigned char **pCODE**,
unsigned char **pCG[]**);

Explication:

pUARTport	N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pCODE	Valeur entre 1 et 8 qui contiendra le nouveau caractère.
pCG	8 octets contenant le dessin du caractère 5x8 redéfini.

L'utilisateur peu redéfinir 8 caractères (composés chacun de 5 x 8 points) au sein de la mémoire CGRAM de l'afficheur « ALCD ». La redéfinition se faisant en mémoire RAM, il en résultera qu'une coupure d'alimentation provoquera leur disparition. Une fois mémoriser, il vous sera possible d'utiliser ces caractères de la même façon qu'un caractère standard.

Exemple:

```
void main(void)
{
  unsigned char       iCGdata[8];    // Declare caractère de taille 5x8.
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  // Rédéfinition d'un caractère en forme de croix.
  iCGdata[0] = 0b01110;              // Utilisation format binaire.
  iCGdata[1] = 0b00100;
  iCGdata[2] = 0b10101;
  iCGdata[3] = 0b11111;
  iCGdata[4] = 0b10101;
  iCGdata[5] = 0b00100;
  iCGdata[6] = 0b00100;
  iCGdata[7] = 0b01110;

  AlcdInit(0);
  AlcdCgWrite(0, 1, iCGdata);        // Mémorise à l'adresse 1 en CGRAM.
  AlcdLocate(0, 0, 0);
  AlcdPrintCode(0, 1);        // Affiche le caractère à l'écran.
}
```

AlcdBackLight

Fonction:
void AlcdBackLight(unsigned char **pUARTport**, unsigned char **pOF**);

Explication:

pUARTport	N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pOF	ON/OFF.

Cette fonction active / désactive le rétro-élairage.

AlcdKey

Fonction:
void AlcdKey(unsigned char **pUARTport**, unsigned char **pKEY**);

Explication:
pUARTport N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pKEY Fonction touches virtuelles pour l'afficheur « ALCD ».

L'afficheur « ALCD » supporte les touches de fonctions classiques d'un éditeur de texte tels que « insert/overwrite », « backspace », etc… Toutes les touches de fonctions supportées sont au format VK_XXXX.

Exemple:

```
void Delay(int pTIME)
{
  while(pTIME--);
}
void main(void)
{
  int   i;

  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);
  AlcdLocate(0, 0, 0);
  AlcdPrint(0,"0123456789");

  // Effacement des caractères un par un avec fonction BACKSPACE.
  for(i=0;i<10;i++)
  {
    AlcdKey(0, VK_BACKSPACE);
    Delay(0xfff);
  }
}
```

AlcdMode

Fonction:
void AlcdMode(unsigned char **pUARTport**, unsigned char **pMODE**);

Explication:
pUARTport N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pMODE VK_INSERT ou VK_OVERWRITE.

L'afficheur « ALCD » supporte les mêmes modes d'édition que les éditeurs de textes.

Exemple:

```
void main(void)
{

  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);

  AlcdMode(0, VK_OVERWRITE);  // Configure le ALCD en mode OVERWRITE.

  AlcdLocate(0, 0, 0);
  AlcdPrint(0,"0123456789");

  AlcdMode(0, VK_INSERT);     // Configure ALCD en mode OVERWRITE INSERT.

AlcdLocate(0, 3, 0);  // Place le curseur sur la position 3.
AlcdPrint(0,"ABC");   // Lors de l'écriture, les nombres se décalent !
}
```

AlcdSetBigFont

Fonction:
void AlcdSetBigFont(unsigned char **pUARTport**);

Explication:
pUARTport N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.

Cette fonction permet d'écrire des « grandes » fontes à l'aide des caractères redéfinissables de la mémoire CGRAM du « ALCD ». La fonction doit être déclarée avant de pouvoir écrire de gros caractères. La mémoire CGRAM ne sera bien évidemment plus dans ce cas, disponible pour une redéfinition personnelle.

Les « grandes » fontes permettent d'afficher les caractères 0 à 9. Un caractère occupe 3 x 4 cases de l'écran (vous devez donc disposer au moins 'un afficheur « ALCD » de 4 x 20 caractères). Cette fonction est très utile pour afficher une horloge, une tension ou toute autre application nécessitant une grande visibilité.

Une « grande » fonte initialisée par AlcdSetBigFont() nepourra être exploitée que par la fonction AlcdPrintBig().

Exemple:

```
void main(void)
{
  UartSetBaud(0,115200);      // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                  // Active l'UART0.

  AlcdInit(0);
  AlcdSetBigFont(0);  // Active les grandes fontes sur le « ALCD ».
}
```

AlcdPrintBig

Fonction:
void AlcdPrintBig(unsigned char **pUARTport**,
unsigned char **pX**,
unsigned char **pCODE**);

Explication:

pUARTpor	N° du port UART (0 ou 1) sur lequel l'ALCD est connecté.
pX	Position X sur laquelle on va écrire.
pCODE	Nombres 0 à 9 ou caractères '0' à '9'.

Cette fonction permet d'afficher de gros caractère sur le « ALCD ». AlcdSetBigFont() doit être appelé avant cette fonction (sans quoi des caractères « bizarroïde » s'afficheront à la place des nombres sur le « ALCD »).

Exemple:

```
void main(void)
{
UartSetBaud(0,115200);        // Configure vitesse UART0 à 115200 BPS.
  UartOn(0);                   // Active l'UART0.

  AlcdInit(0);

  AlcdSetBigFont(0);

  for(i=0;i<10;i++)
  {
    AlcdLocate(0, 0, 0);
    AlcdPrintBig(0,i*4,i);
  }
}
```

GetMsg, CheckMsg

Fonction:
unsigned char GetMsg(void);

unsigned char CheckMsg(void);

Explication:

GetMsg

Récupère un MESSAGE de la file d'attente des messages (MESSAGE-QUEUE).
Cette fonction retire le MESSAGE de la lsiteen même temps.
Si la list d'attente est vide, un message nul (MSG_NULL) sera retourné.

CheckMsg

Contrairement à la fonction GetMsg, CheckMsg permet de connaître le MESSAGE present dans la file d'attente, sans pour autant retirer ce dernier de la file (seul la fonction GetMsg permettra de retirer ce dernier de la liste).

ClearAllMsg

Fonction:
void ClearAllMsg(void)

Explication:
Vide complètement la file d'attente des MESSAGES du ROVIN™ et réinitialise cette file d'attente. Le module ROVIN™ génère un EVENEMENT s'il y a au moins un MESSAGE de présent. En effectuant cette fonction, aucun EVENEMENT ne sera généré puisqu'il n'y aura plus de MESSAGE.

CounterSet, CounterCount

Fonction:

void CounterSet(unsigned char **pID**, unsigned char **pMODE**);
void CounterCount(unsigned char **pID**, unsigned short **pCOUNT**);

Explication:

CounterSet

pID	Utilisez 0, 1, ou 2 pour le COUNTER0, COUNTER1, ou COUNTER2.
PMODE	4 bits poids forts = COUNTER_1EVENT ou COUNTER_XEVENT. 4 bits poids faibles = COUNTER_HIGHEDGE ou COUNTER_LOWEDGE.

Utilisez les 4 bits de poids forts de **pMODE** pour configurer le mode EVENT et les 4 bits de poids faibles pour configurer le mode COUNT-CONDITION.

COUNTER_1EVENT est utilisé pour compter et créer un EVENEMENT.
COUNTER_XEVENT sera utilisé pour compter et créer continuellement un EVENEMENT plutôt qu'une seule fois.

Si COUNT-CONDITION est configuré en COUNTER_HIGHEDGE, le compteur comptera à chaque front montant alors que si COUNTER-LOWEDGE est choisi le compteur comptera à chaque front descendant.

Exemple d'utilisation**:** CounterSet(0, COUNTER_1EVENT| COUNTER_HIGHEDGE);

CounterCount

pID	Utilisez 0, 1, ou 2 pour le COUNTER0, COUNTER1 ou le COUNTER2.
PCOUNT	Détermine combien on doit compter avant de déclencher un EVENEMENT.

La fonction CounterCount() compte la valeur définie en pCOUNT avant de créer un évenement COUNTER-EVENT. Avec COUNTER0 et COUNTER1, une valeur sur 16 bits (1 à 65535) peut être utilisée et avec COUNTER2, une valeur sur 8 bits (1 à 255) put être utilisée.

Exemple d'utilisation**:** Pour configurer le COUNTER2 pour qu'il compte jusqu'à 5 avant de créer un EVENEMENT, il vous faudra écrire:
CounterCount(2,5);

Exemple de programme:

Utilisons le COUNTER0 du module ROVIN™ pour comptabiliser le nombre de fois qu'un bouton-poussoir a été sollicité. Le nombre sera affiché sur 8 leds connectées au port PPI-PD. Le COUNTER0 sera configuré en mode COUNTER_XEVENT et le comptage effectif sur le front descendant. Le bouton-poussoir devra bien évidemment être connecté à la broche COUNTER0 du module. Lorsque le bouton-poussoir sera sollicité 3 fois, un EVENEMENT COUNTER surviendra et le nombre de fois que cet EVENEMENT sera survenu s'affichera sur les Leds.

```
int vmCOUNT;   // Variable pour compter les évenements de counter0

void IRQ_EVENT_(void)
{ // Un èvenement survient !
  // On en prend connaissance et on le traite.

switch(GetMsg()){
case MSG_COUNTER0 :          // C'est un EVENEMENT COUNTER0 !!!

      vmCOUNT++;             // incrémente le compteur d'EVENEMENT !
      PPI_Out(PD,~vmCOUNT); // Sort contenu du compteur sur port xppi-pd !
          break;
  }
}
void main(void)
{
  vmCOUNT = 0;

  // Met le port PPI-PD en sortie.
  PPI_SetMode(PD, 0x00);
  PPI_Out(PD,0xff);       //  Eteind toutes les Leds

//[COUNTER0-INIT]{----------------------------------------
// Configure en COUNTER_XEVENT pour le mode continu,
// et pour une action sur un front descendant


  CounterSet(0, COUNTER_XEVENT|COUNTER_LOWEDGE);

// Configure compteur pour déclencher èvenement après 3 impulsions sur BP

   CounterCount(0, 3);

// Active le compteur.

  CounterOn(0); // Broches utilisées par COUNTER0 automatiquement
                // placées en mode sortie / pull-up !
   // Autorise les EVENEMENTS !
  EventOn();
  // Attend un EVENEMENT !
 while(1);
}
```

CounterRead

Fonction:
unsigned short CounterRead(unsigned char **pID**);

Explication:
CounterRead

pID	Utilisez 0, 1, ou 2 pour COUNTER0, COUNTER1 ou COUNTER2.
Retourned Value	La valeur du compte à rebours (Valeur du compteur – valeur déjà comptée)

CounterRead() vous retournera la valeur du compte à rebours (Valeur du compteur – valeur déjà comptée). En d'autre terme, le compteur retournera ne nombre de comptage nécessaire avant qu'un EVENEMENT ne soit généré.

Exemple D'utilisation:

```
unsigned short COUNT0;
COUNT0 = CounterRead(0);
```

CounterOn, CounterOff

Fonction:
void CounterOn(unsigned char **pID**);
void CounterOff(unsigned char **pID**);

Explication:

CounterOn

pID Utilisez 0, 1, 2 pour COUNTER0, COUNTER1 ou COUNTER2.

La fonction CounterOn() active le COUNTER. Les broches utilisées par les cmpteurs sont automatiquement configurée en entrée avec pull-up. Les ENVENEMENT relatifs aux compteurs sont aussi activés en même temps.

CounterOff

pID Utilisez 0, 1, 2 pour COUNTER0, COUNTER1 ou COUNTER2.

La fonction CounterOff() désactive le COUNTER. La valeur du compteur est conservée (même lorsque ce dernier est désactivé). Lorsque le compteur est de nouveau activé, ce dernier continuera son action à partir de sa valeur précédente.

DebugPrint, DebugClear

Fonction:
void DebugPrint(char * **pSTR**);
void DebugClear(void);

Explication:

DebugPrint

Affiche des caractères dans la fenêtre de débug du PC.

DebugClear

Efface la fenêtre débug du PC.

Exemple:

```
void main(void)
{
char    *iSTR;

iSTR = "Comfile Technology Inc.\n";

DebugClear();

DenugPrint(iSTR);
DebugPrint("COMFILE TECHNOLOGY\n");
DebugPrint("http://www.comfile.co.kr/\n");
Break ;
}
```

DebugCHAR, DebugSHORT, DebugINT, DebugLONG, DebugDOUBLE

Fonction:
void DebugCHAR(unsigned char **pVALUE**, unsigned char **pVIEWtype**);

void DebugSHORT(unsigned short **pVALUE,** unsigned char **pVIEWtype**);

void DebugINT(unsigned int **pVALUE**, unsigned char **pVIEWtype**);

void DebugLONG(unsigned long **pVALUE**, unsigned char **pVIEWtype**);

void DebugFLOAT(float **pVALUE**, unsigned char **pVIEWtype**);

void DebugDOUBLE(double **pVALUE**, unsigned char **pVIEWtype**);

Explication:

DebugCHAR

Affiche une variable de type **char** et **unsigned char** au format hex ou décimal dans la fenêtre Débug du PC.

DebugSHORT

Affiche une variable de type **short** and **unsigned short char** au format hex ou décimal dans la fenêtre Débug du PC.

DebugINT

Affiche une variable de type **int** and **unsigned int char** au format hex ou décimal dans la fenêtre Débug du PC.

DebugLONG

Affiche une variable de type **long** and **unsigned long char** au format hex ou décimal dans la fenêtre Débug du PC.

DebugFLOAT

Affiche une variable de type **float char** au format hex ou décimal dans la fenêtre Débug du PC.

DebugDOUBLE

Affiche une variable de type **double char** au format hex ou décimal dans la fenêtre Débug du PC**.**

Exemple:

```
void main(void)
{
char            iCHAR;
short           iSHORT;
int             iINT;
long            iLONG;
double          iDOUBLE;

iCHAR = 0x12;
iSHORT = 0x1234;
iINT = 0x12345678;
iLONG = 0x123456789abcdef0;
idouble = 3.14;

DebugClear();
DebugPrint("ichar est [ "];   DebugCHAR(iCHAR,HEX);
DebugPrint(" ). \n\n");

DebugPrint("iSHORT est [ "];  DebugCHAR(iSHORT,HEX);
DebugPrint(" ). \n\n");

DebugPrint("iINT est [ "];    DebugCHAR(iINT,HEX);
DebugPrint(" ).\n\n");

DebugPrint("iLONG est [ "];   DebugCHAR(iLONG,HEX);
DebugPrint(" ).\n\n");

DebugPrint("iDOUBLE est [ "]; DebugCHAR(iDOUBLE,DEC);
DebugPrint(" ). \n\n");
Break ;
}
```

EepSetAdr

Fonction:
void EepSetAdr(unsigned short **pADDRESS**);

Explication:
Détermine l'adresse avant de lire ou d'écrire dans la mémoire EEprom. Le ROVIN™ dispose d'une mémoire EEPRON de 4 K, cete adresse peut être comprise entre 0x000 et 0xfff.

Cette adresse est automatiquement mise à jour lors d'une opération de lecture ou d'écriture de données. Par exemple si vous écrivez 2 octets de donnée, alors ADDRESS = ADDRESS + 2. Dès lors, si vous effectuez une lecture ou une écriture continue dans cette mémoire, il ne vous sera pas nécessaire de modifier l'adresse à chaque fois, il vous suffira simplement d'utiliser les fonctions de lecture et d'écriture.

EepWrite

Fonction:
void EepWrite(unsigned char **pDATA**);

Explication:
Ecrit 1 octet **pDATA** à l'adresse courante de la mémoire EEPROM et ajoute 1 octet à l'ADRESSE de la mémoire de pointage de l'EEprom. Vous devez impérativement déterminer l'adresse de la mémoire EEprom avant cette fonction, sans quoi vous risquez d'écrire à n'importe quel endroit de la mémoire de façon intempestive.

Exemple:

```
void main(void)
{
  int   i;

EepSetAdr(0x000);      // Démarre à l'adresse 0x000 de l'EEPROM.

for(i=0;i<=0xff;i++)
{
  EepWrite(i);
}
}
```

EepRead

Fonction:
unsigned char EepRead(void);

Explication:
Ecrit 1 octet à l'adresse courante de la mémoire EEPROM et ajoute 1 octet à l'ADRESSE de la mémoire de pointage de l'EEprom. Vous devez impérativement déterminer l'adresse de la mémoire EEprom avant cette fonction, sans quoi vous risquez de lire une donnée à n'importe quel endroit de la mémoire de façon intempestive.

Exemple:

```
void main(void)
{
  int   i;

DebugClear();

EepSetAdr(0x000);      // Démarre à l'adresse 0x000 de l'EEPROM.

for(i=0;i<=0xff;i++)
{
  EepWrite(i);         // Ecrit dans l'EEprom.
}

EepSetAdr(0x000);      // Démarre à nouveau à l'adresse 0x000 de l'EEPROM.

for(i=0;i<=0xff;i++)
{
  DebugPrint("Affiche 256 octets de données depuis l'adresse 0x000.\n");
  if((i%16)==0)               // Affiche 16 octets par rangée.
  { DebugPrint("\n");
  }

  DebugCHAR(EepRead(),HEX);
  DebugPrint(" ");     // Laisse un espace entre les données.
}
}
```

EepBufWrite

Fonction:
void EepBufWrite(unsigned char * **pBUFFER**, unsigned char **pCOUNT**);

void EepBulkWrite(void);

Explication:

EepBufWrite

Ecrit **pBUFFER** (**pCOUNT** octets) dans le buffer temporaire du ROVIN™. Vous devez utiliser cette fonction en ligne avec EepBulkWrite(). Ce buffer temporaire utilise la fonction EepBulkWrite() pour écrire immédiatement dans l'EEPROM (Durant cette période, le ROVIN™ peut effectuer d'autres tâches. Bien que pCOUNT soit de type unsigned char, vous devez utiliser une valeur comprise entre 1 et 255. En d'autre terme, vous pouvez transférer jusqu'à 255 octets de données à la fois.

Si vous écrivez un octet à la fois en EEprom de façon cyclique et répétitif, vous pourrez être amené à constater des ralentissements dans votre programme (le ROVIN devra attente la fin des cycles d'écriture). Dès lors, si un grand nombre de tâches sont exécutées en même temps, le traitement sera plus lent. En revanche, en utilisant EepBulkWrite() de façon appropriée (en écrivant d'un coup toutes vos données), le ROVIN™ pourra faire d'autres choses sans attendre la fin des cycles d'écriture en EEPROM.

EepBulkWrite

Ecrit les données du buffer temporaire en mémoire EEprom à partir de l'adresse (ADDRESS).

Exemple:

```
void main(void)
{
  unsigned char        iBUFFER[256];
  int          i;

DebugClear();

  for(i=0;i<0xff;i++)
  { iBUFFER[i] = i;            // Rempli le tableau de données.
  }

  EepBufWrite(iBUFFER, 255) ;  // Rempli le buffer temporaire).

  DebugPrint("L'écriture EEprom à partir de l'addresse 0x000!\n");

  EepSetAdr(0x000);    // Fixe l'adresse de départ pour l'écriture.

  EepBulkWrite();      // Transfert des données en EEPROM.

  DebugPrint("Ecriture de toutes les données en Eprom !!!\n");

  DebugPrint("Lecture depuis l'adresse 0x000 de l'EEPROM!!!\n");

  EepSetAdr(0x000);    // Fixe l'adresse de départ pour la lecture.

  for(i=0;i<0xff;i++)
  { if((i%16)==0)      // Affichage 16 octets sur chaque ligne.
      DebugPrint("\n");
    }

    DebugCHAR(EepRead(),HEX); DebugPrint(" ");  //Lit et affiche données.
  }

  DebugPrint("\n\nLecture effectuée !\n");
}
```

EepWrChar, EepWrShort, EepWrInt
EepWrLong, EepWrDouble

Fonction:
void EepWrChar(unsigned char **pCHARdata**);

void EepWrShort(unsigned short **pSHORTdata**);

void EepWrInt(unsigned int **pINTdata**);

void EepWrLong(unsigned long **pLONGdata**);

void EepWrDouble(double **pDOUBLEdata**);

Explication:

EepWrChar
Ecrit **pCHARdata** à l'adresse ADDRESS en EEPROM et incrémente ADDRESS par 1.

EepWrShort
Ecrit **pSHORTdata** sur l'octet haut de ADDRESS et incrémente ADDRESS par 2.

EepWrInt
Ecrit **pINTdata** sur l'octet haut de ADDRESS et incrémente ADDRESS par 4.

EepWrLong
Ecrit **pLONGdata** sur l'octet haut de ADDRESS et incrémente ADDRESS par 8.

EepWrDouble
Ecrit **pDOUBLEdata** sur l'octet haut de ADDRESS et incrémente ADDRESS par 8. Double est différent de Long car il peut stocker des nombres à virgule flottante tels 3.14, 0.231, etc…

Exemple:

```
void main(void)
{
DebugClear();

// Configure adresse de début.
EepSetAdr(0x000);

EepWrChar(0x12);                    // Ecrit en 0x000.
EepWrShort(0x1234);                 // Ecrit en 0x001.
EepWrInt(0x12345678);               // Ecrit en 0x003.
EepWrLong(0x123456789abcdef0);      // Ecrit en 0x007.
EepWrDouble(3.14);                  // Ecrit en 0x00f.

// Lecture et affichage des données
EepSetAdr(0x0000);
DebugPrint("L'addresse 0x000 ecrite avec EepWrChar() est à: ");
DebugSHORT(EepRdChar(),HEX);
DebugPrint(" .\n\n");

EepSetAdr(0x0001);
DebugPrint("L'addresse 0x001 ecrite avec EepWrShort()est à: ");
DebugSHORT(EepRdShort(),HEX);
DebugPrint(" .\n\n");

EepSetAdr(0x0003);
DebugPrint("L'addresse 0x003 ecrite avec EepWrInt()est à: ");
DebugSHORT(EepRdInt(),HEX);
DebugPrint(".\n\n");

EepSetAdr(0x0007);
DebugPrint("L'addresse 0x007 ecrite avec EepWrLong()est à: ");
DebugSHORT(EepRdLong(),HEX);
DebugPrint(" .\n\n");

EepSetAdr(0x000f);
DebugPrint("Address 0x00f ecrite avec EepWrDouble()est à: ");
DebugSHORT(EepRdDouble(),HEX);
DebugPrint(".\n\n");
}
```

EepRdChar, EepRdShort, EepRdInt EepRdLong, EepRdDouble

Fonction:
unsigned char EepRdChar(void);

unsigned short EepRdShort(void);

unsigned int EepRdInt(void);

unsigned long EepRdLong(void);

double EepRdDouble(void);

Explication:

EepRdChar

Lit 1 octet depuis la mémoire EEPROM et incrément ADDRESS de 1.

EepRdShort

Lit 2 octets depuis la mémoire EEPROM et incrément ADDRESS de 2.

EepRdInt

Lit 4 octets depuis la mémoire EEPROM et incrément ADDRESS de 4.

EepRdLong

Lit 8 octets depuis la mémoire EEPROM et incrément ADDRESS de 8.

EepRdDouble

Lit 8 octets depuis la mémoire EEPROM et incrément. ADDRESS by 8. Double est différent de Long car il peut lire des nombres à virgule flottante tels 3.14, 0.231, etc……

Exemple:

```
void main(void)
{
  DebugClear();

EepSetAdr(0x000);     // Configure adresse d'écriture.

DebugPrint("Enregistre à l'adresse 0x0000 de la mémoire EEPROM "
            "La valeur 16 bits 3.14, 0x40091eb851eb851f! \n\n");
EepWrLong(0x40091eb851eb851f);

  EepSetAdr(0x000);   // Configure adresse de lecture.
  DebugPrint("Le resultat de EepRdChar() est: ");
  DebugCHAR(EepRdChar(),HEX);
  DebugPrint(" .\n\n");

  EepSetAdr(0x000);   // Configure adresse de lecture..
  DebugPrint("Le resultat de EepRdShort() est: ");
  DebugSHORT(EepRdShort(),HEX);
  DebugPrint(" .\n\n");

  EepSetAdr(0x000);   // Configure adresse de lecture.
  DebugPrint("Le resultat de EepRdInt() est: ");
  DebugINT(EepRdInt(),HEX);
  DebugPrint(" .\n\n");

  EepSetAdr(0x000);   // Configure adresse de lecture.
DebugPrint("Le resultat de EepRdLong() est: ");
  DebugLONG(EepRdLong(),HEX);
  DebugPrint(".\n\n");

  EepSetAdr(0x000);   // Configure adresse de lecture.
DebugPrint("Le resultat de EepRdDouble() est: ");
  DebugDOUBLE(EepRdDouble(),DEC);
  DebugPrint(" .\n\n");
}
```

EepFill

Fonction:
void EepFill(unsigned char **pFILLvalue**, unsigned short **pAREAsize**);

Explication:
Remplit l'espace **pAREAsize** de l'EEPROM avec **pFILLvalue**. L'espace **pAREAsize** peut être compris entre 1 et 4096 (soit la capacité max. de l'Eeprm du module ROVIN™).

Exemple:

```
void main(void)
{
int     i;
DebugClear();
// Ecrit 0x00 dans l'EEPROM depuis l'addresse 0x000 jusqu'à 0xff !

  EepSetAdr(0x000);
  DebugPrint("Ecriture de 0x000 ~ 0xff… \n");
  EepFill(0x00,256);

  // Lecture pour vérification de la manipulation.
  EepSetAdr(0x0000);
  for(i=0;i<256;i++)
  { if((i%16)==0) DebugPrint("\n");  // Affiche 16 octets par ligne.

    DebugCHAR(EepRdChar(),HEX);
    DebugPrint(" ");                 // Affiche un espace entre octets.
  }
}
```

EventOn, EventOff

Fonction:
void EventOn(void);

void EventOff(void);

Explication:

EventOn

Active (Autorise) tous les EVENEMENTS.
Le ROVIN™ est capable de gérer de nombreux EVENEMENTS indépendants. En réalisant cette fonction, tous les EVENEMENTS pourront fonctionner.

EventOff

Désactive (Inhibe) tous les EVENEMENTS.
Ceci désative tous les EVENEMENTS.

Exemple:

```
void IRQ_EVENT_(void)
{
  if(GetMsg() == MSG_EXINT4)
  {
    DebugPrint("EVENEMENT EXINT4 survenu !\n");
  }
}

void main(void)
{
  PortSetMode(EXPB,0xf0);            // Broche interruption en entrée..
  PortOut(EXPB,0xff);                // Prévoir resistance de Pull-Up.

  ExintSet(4,EXINT_HEDGE);           // Pin EXINT.4 en mode FRONT montant.
  ExintOn(4);

  EventOn();                         // utorise tous les EVENEMENTS.

  while(1);                          // ATTEND un EVENEMENT.
}
```

HeapWrChar, HeapWrShort, HeapWrInt, HeapWrLong, HeapWrDouble

Fonction:
void HeapWrChar(unsigned int **pOFSadr**, unsigned char **pCHARdata**);

void HeapWrShort(unsigned int **pOFSadr**, unsigned short **pSHORTdata**);

void HeapWrInt(unsigned int **pOFSadr**, unsigned int pINTdata);

void HeapWrLong(unsigned int **pOFSadr**, unsigned long **pLONGdata**);

void HeapWrDouble(unsigned int **pOFSadr**, double **pDOUBLEdata**);

Explication:

HeapWrChar
Ecrit 1 octet **pCHARdata** à l'adresse **pOFSadr** de la mémoire RAM XHEAP.

HeapWrShort
Ecrit 2 octets **pSHORTdata** à l'adresse **pOFSadr** de la mémoire RAM XHEAP.

HeapWrInt
Ecrit 4 octets **pINTdata** à l'adresse **pOFSadr** de la mémoire RAM XHEAP.

HeapWrLong
Ecrit 8 octets **pLONGdata** à l'adresse **pOFSadr** de la mémoire RAM XHEAP.

HeapWrDouble
Ecrit 8 octets avec virgule flottante **pDOUBLEdata** à l'adresse **pOFSadr** de la mémoire XHEAP.

Exemple:

```
void main(void)
{ HeapWrChar(0x0000, 0x12);
  HeapWrShort(0x0001, 0x1234);
  HeapWrInt(0x0003, 0x12345678);
  HeapWrLong(0x0007, 0x123456789abcdef0);
  HeapWrDouble(0x000f, 3.14);
}
```

HeapRdChar, HeapRdShort, HeapRdInt, HeapRdLong, HeapRdDouble

Fonctions:
unsigned char HeapRdChar(unsigned int **pOFSadr**);

unsigned short HeapRdShort(unsigned int **pOFSadr**);

unsigned int HeapRdInt(unsigned int **pOFSadr**);

unsigned long HeapRdLong(unsigned int **pOFSadr**);

double HeapRdDouble(unsigned int **pOFSadr**);

Explication:

HeapRdChar
Lit et retourne 1 octet depuis l'adresse **pOFSadr** en mémoire RAM HEAP.

HeapRdShort
Lit et retourne 2 octets depuis l'adresse **pOFSadr** en mémoire RAM HEAP.

HeapRdInt
Lit et retourne 4 octets depuis l'adresse **pOFSadr** en mémoire RAM HEAP.

HeapRdLong
Lit et retourne 8 octets depuis l'adresse **pOFSadr** en mémoire RAM HEAP.

HeapRdDouble
Lit et retourne 8 octets depuis l'adresse **pOFSadr** en mémoire RAM HEAP et le converti en virgule flottante (ex. : 0x40091eb851eb851f est converti en 3.14).

Exemple:

```
void main(void)
{
  DebugClear();

  // Enregsitre 3.14 à l'adresse 0x00000000 sur 16 bits.
  // Pour écrire tout en une seule fois, utilisez le format LONG.
  HeapWrLong(0x00000000,0x40091eb851eb851f);

  DebugPrint("Le resultat de HeapRdChar() est: ");
  DebugCHAR(HeapRdChar(0x00000000),HEX);
  DebugPrint(" .\n\n");

  DebugPrint("Le resultat de HeapRdShort() est: ");
  DebugSHORT(HeapRdShort(0x00000000),HEX);
  DebugPrint(" .\n\n");

  DebugPrint("Le resultat de HeapRdInt() est: ");
  DebugINT(HeapRdInt(0x00000000),HEX);
  DebugPrint(" .\n\n");

  DebugPrint("Le resultat de HeapRdLong() est: ");
  DebugLONG(HeapRdLong(0x00000000),HEX);
  DebugPrint(" .\n\n");

  DebugPrint("Le resultat de HeapRdDouble() est: ");
  DebugDOUBLE(HeapRdDouble(0x00000000),DEC);
  DebugPrint(" .\n\n");
}
```

HeapBufWrite

Fonction:
void HeapBufWrite(unsigned int **pOFSadr**,
unsigned char *** pBUF**,
unsigned int **pSIZE**);

Explication:

pOFSadr	Adresse d'écriture (depuis 0x00000000) en RAM-XHEAP.
pBUF	Pointeur du buffeur où mémoriser les données.
pSIZE	Taille des octets à utiliser lors du transfert.

Ecrit un buffer de données à l'ADDRESS en mémoire RAM-XHEAP. Vous pouvez déterminer la taille **pSIZE** des données à transférer.

La mémoire RAM-XHEAP peut être considérée comme un espace librement utilisable pour des stockages temporaire. La vitesse d'accès à cette mémoire est très rapide (toutefois gardez à l'esprit que cette mémoire est volatile).

Exemple:

```
void main(void)
{
  unsigned char iBUF[100];

  HeapBufWrite(0x00000000, iBUF, 100);       // Ecrit dans le buffer.

  HeapBufWrite(0x00000000+ 100,
                          (unsigned char *)"COMFILE TECHNOLOGY.", 20);

// Ecrit encore après 1er buffer, puis '\0' après la chaine de caractères.
```

HeapBufRead

Fonction:
void HeapBufRead(unsigned int **pOFSadr**,
unsigned char * **pBUF**,
unsigned int **pSIZE**);

Explication:

pOFSadr	Adresse lecture (depuis 0x00000000) en RAM-XHEAP
pBUF	Pointeur du buffeur où mémoriser les données.
pSIZE	Taille des octets à lire.

Lit **pSIZE** octets de données depuis l'adresse **pOFSadr** et les stock dans **pBUF**.
La taille de **pBUF** doit être la même ou plus grande que **pSIZE**, sans quoi, une erreur fatale interviendra dans le traitement de la TACHE.

Exemple:

```
void main(void)
{
  unsigned char iBUF[20];

  DebugClear();

  // Ecrit une chaine.
  HeapBufWrite(0x00000000,
                (unsigned char *)"COMFILE TECHNOLOGY.",
                20);

  HeapBufRead(0x00000000,
               iBUF,
               20);         // Lit la chaine.

  DebugPrint("iBUF[] contient: ");
  DebugPrint((char *)iBUF);   // Print iBUF.
  DebugPrint("\n");
}
```

HeapClear

Fonction:
void HeapClear(unsigned int **pOFSadr,** unsigned int **pSIZE**);

Explication:
POFSadr Addresse où en doit commencer à effacer la RAM-XHEAP.
pSIZE Taille de la plage à effacer (Utiliser 1 à 0x4000).

Remplit la mémoire RAM-XHEAP avec 0x00 à partir de l'adresse (**pOFSadr**) jusqu'à l'adresse (**pOFSadr**+**pSIZE**). Cette fonction est extrêmement rapide et efficace. Il est recommandé d'utiliser cette dernière pour initialiser la mémoire avant de l'utiliser (gardez à l'esprit que la mémoire RAM-XHEAP peut contenir des valeurs aléatoires à la mise sous tension, d'où l'intérêt d'avoir à l'initialiser au préalable pour être sûr des valeurs.

Exemple:

```
void main(void)
{
// Efface et initialise 256 K de la mémoire RAM-XHEAP avec 0x00.
HeapClear(0x00000000, 0x3fff);
}
```

HeapFill

Fonction:
void HeapFill(unsigned int **pOFSadr**,
unsigned char **pFILLvalue**,
unsigned int **pAREAsize**);

Explication:

pOFSadr	Addresse où en doit commencer à remplir la RAM-XHEAP.
pFILLvalue	Valeur à remplir.
pAREAsize	Taille de la plage à remplir (de 1 à 0x4000).

Exemple:

```
void main(void)
{
  // Remplit 256 K de lamémoire RAM-XHEAP avec 0xff.
  HeapFill(0x0000,0xff, 0x4000);
}
```

ModuleReset, TaskReset

Fonction:
void ModuleReset(void);
void TaskReset(void);

Explication:

ModuleReset

Reset le module ROVIN™ (équivalent Reset matériel via BP RESET).

TaskReset

Reset la TACHE dans laquelle est appelée cette fonction. La TACHE redémarre depuis le début en « sautant » à son VECTEUR de RESET (sans affecter le fonctionnement des autres TACHES).

Exemple:

```
void main(void)
{
int i;

DebugClear();
DebugPrint("Démmare une NOUVELLE TACHE !!!\n");

for(i=0;i<=0xffff;i++);      // Attend un petit delai
                             // en faisant une boucle.

DebugPrint("Reset de la TACHE en cours !!!\n");

// Reset la tâche en cours
TaskReset();
}
```

PPI_SetMode, PPI_GetMode

Fonction:
void PPI_SetMode(unsigned char **pPORT**, unsigned char **pMODE**);

unsigned char PPI_GetMode(unsigned char **pPORT**);

Explication:

SetMode

Configure le port **pPORT** en Entrée/Sortie.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
0 : Sortie.
1 : Entrée.

PPI_GetMode

Lit et retourne le mode de configuration du port **pPORT**.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
0 : Sortie.
1 : Entrée.

Exemple:

```
void main(void){
unsigned char iSETmode;
unsigned char iGETmode;

  iSETmode = 0xff;

DebugClear();

PPI_SetMode(PA,iSETmode);
PPI_SetMode(PD,0x00);

DebugPrint("Le port PPI-PA est configuré en : [");
iGETmode = PPI_GetMode(PA);
DebugCHAR(iGETmode, HEX);
DebugPrint(" ).\n");
}
```

PPI_SetBitMode, PPI_GetBitMode

Fonction:
void PPI_SetBitMode(unsigned char **pPORT**,
unsigned char **pBIT**,
unsigned char **pBITmode**);

unsigned char PPI_GetBitMode(unsigned char **pPORT**,
unsigned char **pBIT**);
Explication:

PPI_SetBitMode

Configure la broche (comprise entre 0 et 7) **pBIT** en entrée ou en sortie.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
0 : Sortie
1 : Entrée

PPI_GetBitMode

Lit et retourne le mode (entrée ou sortie) d'un bit **pBIT** (compris entre 0 et7) de **pPORT**.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
0 : Sortie
1 : Entrée

Exemple:

```
void main(void) {
unsigned char iBITmode;

DebugCleear();

PPI_SetBitMode(PA, 0, 1);
PPI_SetBitMode(PD, 0, 0);

DebugPrint("La broche 0 de PPI-PA. est en mode : ");
iBITmode = PPI_GetBitMode(PA, 0);
DebugCHAR(iBITmode, DEC);
DebugPrint(" . \n");
}
```

PPI_Out, PPI_In

Fonction:
void PPI_Out(unsigned char **pPORT**, unsigned char **pDATA**);

unsigned char PPI_In(unsigned char **pPORT**);

Explication:

PPI_Out

Sort la donnée **pDATA** sur le port **pPORT**
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
Le mode de fonctionnement entrée/sortie doit être préalablement défini.

PPI_In

Lit et retourne la valeur de l'octet présent sur le port **pPORT**.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
Le mode de fonctionnement entrée/sortie doit être préalablement défini.

Exemple:

```
void main(void)
{
unsigned char iINDATA;
unsigned char iOUTDATA;

DebugClear();

PPI_SetMode(PA, 0xff);
PPI_SetMode(PD, 0x00);

PPI_Out(PD,0x12);

DebugPrint("Les entrées du port PPI-PA sont: ");
iINDATA = PPI_In(PA);
DebugCHAR(iINDATA, HEX);
DebugPrint(".\n");

iOUTDTA = 0x34;
PPI_Out(PD, iOUTDATA);

}
```

PPI_BitOut, PPI_BitIn

Fonction:
void PPI_BitOut(unsigned char **pPORT**,
unsigned char **pBIT**,
unsigned char **pBITdata**);

unsigned char PPI_BitIn(unsigned char **pPORT**,
unsigned char **pBIT**);

Explication:

PPI_BitOut

Place la broche **pBIT** du port **pPORT** à 0 ou 1
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
pBIT doit être compris entre 0 et 7.

PPI_BitIn

Lit et retourne la valeur (1 ou 0) de la broche **pBIT** du port **pPORT**.
pPORT concerne les ports PA / PB / PC ou PD du module ROVIN™.
pBIT doit être compris entre 0 et 7.

Exemple:

```
void main(void)
{
unsigned char iBITvalue;

DebugClear();

PPI_SetMode(PA, 0xff);
PPI_SetMode(PD, 0x00);

PPI_BitOut(PD, 3, 1);

DebugPrint("La broche 3 du port PPI_PA est à [ ");
iBITvalue = PPI_BitIn(PA, 3);
DebugCHAR(iBITvalue, DEC);
DebugPrint(" ). \n");
}
```

PortSetMode, PortGetMode

Fonction:
void PortSetMode(unsigned char **pPORT**, unsigned char **pMODE**);

unsigned char PortGetMode(unsigned char **pPORT**);

Explication:

PortSetMode
Configure le port **pPORT** en Entrée/Sortie.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
0 : Sortie.
1 : Entrée.

PortGetMode
Lit et retourne le mode de configuration du port **pPORT**.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
0 : Sortie.
1 : Entrée

Exemple:

```
void main(void)
{
unsigned char iSETmode;
  unsigned char iGETmode;

  iSETmode = 0xff;

DebugClear();

PortSetMode(EXPA,iSETmode);
PortSetMode(EXPD,0x00);

DebugPrint("Le port EXPA est configuré en: ");
iGETmode = PortGetMode(EXPA);
DebugCHAR(iGETmode, HEX);
DebugPrint("].\n");
}
```

PortSetBitMode, PortGetBitMode

Fonction:
void PortSetBitMode(unsigned char **pPORT**,
unsigned char **pBIT**,
unsigned char **pBITmode**);

unsigned char PortGetBitMode(unsigned char **pPORT**,
unsigned char **pBIT**);

Explication:

PortSetBitMode
Configure la broche (comprise entre 0 et 7) **pBIT** de **pPORT** en entrée ou en sortie.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
0 : Sortie.
1 : Entrée.

PortGetBitMode
Lit et retourne le mode (entrée ou sortie) d'un bit **pBIT** (compris entre 0 et7) de **pPORT**.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
0 : Sortie.
1 : Entrée.

Exemple:

```
void main(void)
{
unsigned char iBITmode;

DebugClear();

PortSetBitMode(PA, 0, 1);
PortSetBitMode(PD, 0, 0);

DebugPrint("La broche 0 de PPI-PA. est en mode [ ");
iBITmode = PortGetBitMode(EXPA, 0);
DebugCHAR(iBITmode, DEC);
DebugPrint(" ].\n");
}
```

PortOut, PortIn

Fonction:
void PortOut(unsigned char **pPORT**, unsigned char **pDATA**);

unsigned char PortIn(unsigned char **pPORT**);

Explication:
PortOut
Sort la donnée **pDATA** sur le port **pPORT**
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
Le mode de fonctionnement entrée/sortie doit être préalablement défini.

PortIn
Lit et retourne la valeur de l'octet présent sur le port **pPORT**.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
Le mode de fonctionnement entrée/sortie doit être préalablement défini.

Exemple:

```
void main(void)
{
unsigned char iINDATA;
unsigned char iOUTDATA;

DebugClear();

PortSetMode(EXPA, 0xff);
PortSetMode(EXPD, 0x00);
PortOut(EXPD,0x12);

DebugPrint("Les entrées du port EXPA sont : ");
iINDATA = PortIn(EXPA);
DebugCHAR(iINDATA, HEX);
DebugPrint(".\n");

iOUTDTA = 0x34;
PortOut(EXPD, iOUTDATA);

}
```

PortBitOut, PortBitIn

Fonction:
void PortBitOut(unsigned char **pPORT**,
unsigned char **pBIT**,
unsigned char **pBITdata**);

unsigned char PortBitIn(unsigned char **pPORT**, unsigned char **pBIT**);

Explication:

PortBitOut

Place la broche **pBIT** du port **pPORT** à 0 ou 1
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
pBIT doit être compris entre 0 et 7.

PortBitIn

Lit et retourne la valeur (1 ou 0) de la broche **pBIT** du port **pPORT**.
pPORT concerne les ports XPA / XPB ou XPD du module ROVIN™.
pBIT doit être compris entre 0 et 7.

Exemple:

```
void main(void)
{
unsigned char iBITvalue;

DebugClear();

PortSetMode(EXPA, 0xff);
PortSetMode(EXPD, 0x00);

PortBitOut(EXPD, 3, 1);

DebugPrint("La broche 3 du port EXPA est à: ");
iBITvalue = PortBitIn(EXPA, 3);
DebugCHAR(iBITvalue, DEC);
DebugPrint(".\n");

}
```

Pwm0_Set

Fonction:
void Pwm0_Set (unsigned char **pMODE**, unsigned char **pFREQ**);

Explication:
pMODE Configure le mode de PWM0.
pFREQ Détermine la fréquence de PWM0.

Pwm0_Duty

Fonction:
void Pwm0_Duty (unsigned char **pDUTY**);

Explication:
pDUTY Détermine le rapport cyclique du signal PWM0 (0 à 255)

Le signal PWM0 est en mode 8 bits (sa précision est d'1/256).

Pwm0_On, Pwm0_Off

Fonction:
void Pwm0_On (void);

void Pwm0_Off (void);

Explication:

Pwm0_On

Active le signal PWM0. Du fait que le signal PWM0 partage ses ressources avec l'horloge temps réel, il ne sera pas possible d'utiliser les deux en même temps. Lorsque PWM0 est utilisé, l'horloge RTC sera coupée automatiquement. Lorsque le PWM0 sera stoppé, l'horloge RTC sera à nouveau activée.

Pwm0_Off

Stoppe le signal PWM0. Si vous activez à nouveau le signal PWM0 après l'avoir préalablement stoppé, ce dernier redémarre avec les mêmes paramètres que précédemment..

Exemple:

```
void main(void) {
// Utilisation de l'horloge 18.4320 MHz !
// Une fréquende de 57.6K hz est générée.

  Pwm0_Set(PWM_FAST, PWM0_FREQ_1);

  Pwm0_Duty(0xff/2);  // Rapport cyclique de 50 %.

  Pwm0_On();          // Démarre le signal.

  while(1);
}
```

Pwm2_Set

Fonction:
void Pwm2_Set (unsigned char **pMODE** , unsigned char **pFREQ**);

Explication:
PMODE Configure le mode de PWM2.
pFREQ Détermine la fréquence de PWM2.

Du fait que le signal PWM2 partage ses ressources avec le TIMER, ces derniers ne pourront pas êtres utilisés en même temps. De façon similaire, PWM2 ne pourra pas être utilisé en même temps que PWM1C du fait que PWM2 et PWM1-C utilisent la même broche.

Pwm2_Duty

Fonction:
void Pwm2_Duty (unsigned char **pDUTY**);

Explication:
pDUTY Détermine le rapport cyclique du signal PWM2 (0 ~ 255)

Le signal PWM0 est en mode 8 bits (sa précision est d'1/256)..

Pwm2_On, Pwm2_Off

Fonction:
void Pwm2_On (void);

void Pwm2_Off (void);

Explication:

Pwm2_On

Active la sortie PWM2. Du fait que PWM2 utilise les mêmes ressource que le TIMER, le TIMER doit être préalablement désactivé avant d'exploiter PWM2.

Pwm2_Off

Stoppe le signal PWM2. Si vous activez à nouveau le signal PWM2 après l'avoir préalablement stoppé, ce dernier redémarre avec les mêmes paramètres que précédemment.

Pwm1_Set

Fonction:
void Pwm1_Set (unsigned char **pMODE**, unsigned char **pFREQ**);

Explication:

pMODE	Configure le mode de PWM1.
pFREQ	Détermine la fréquence de PWM1.

Configure le mode de fonctionnement et la fréquence du signal PWM1 (voir ci-après). PWM1 dispose d'une résolution max. de 16 bits. Celle-ci peut être sélectionner pour fonctionner en mode 8, 9, 10 ou 16 bits et sur trois canaux (CHA, CHB et CHC). Chaque canal dispose d'une broche de sortie capable de générer un signal PWM. Parmi ces broches, PWM1-C est commune avec PWM2 et de ce fait ils ne pourront pas être utilisés en même temps. Les broches PWM1 figées en sortie et il ne sera pas nécessaire de modifier leur mode de fonctionnement en sortie.

Pwm1_Duty

Fonction:

void Pwm1_Duty (unsigned char **pCHANNEL** , unsigned short **pDUTY**);

Explication:

pCHANNEL	Sélectionne un canal PWM1
PDUTY	Sélectionne un rapport cyclique pour le canal **pCHANNEL** (Différent pour les modes 8, 9, 10 et 16 bits).

Le rapport cyclique de chaque canal peut être sélectionné.
Du fait que PWM1 puisse être configuré en mode 8, 9, 10 ou 16 bits, le rapport cyclique devra être configuré en conséquence. Par exemple le mode 16 bits permettra une sélection de 0 à 2^{16}. Il n'est pas nécessaire de configurer le rapport cylcique pour un canal non utilisé.

Pwm1_On, Pwm1_Off

Fonction:
void Pwm1_On (unsigned char **pCHANNEL**);

void Pwm1_Off (unsigned char **pCHANNEL**);

Explication:

Pwm1_On

pCHANNEL Active un canal PWM1

Sort le signal PWM sur le canal **pCHANNEL**.

Pwm1_Off

pCHANNEL Désactive un canal PWM1

Stop le signal sur la canal **pCHANNEL**.

Pwm1_AllOn, Pwm1_AllOff

Fonction:
void Pwm1_AllOn (void);
void Pwm1_AllOff (void);

Explication:
Pwm1_AllOn
Active les signaux PWM sur toutes les canaux PWM1.

Pwm1_AllOff
Désactive tous les signaux PWM des canaux PWM1.

Exemple:

```
void main(void)
{
  // Utilise une horloge de 18.4320 MHz !
  // Utilise le mode 16 BITS - FAST MODE!
  Pwm1_Set(PWM_16BIT_FAST, PWM_FREQ_1);

  // Détermine les rapports cycliques PWM.
  Pwm1_Duty(PWM_CHA, ((float)0xffff*10)/100); // Rapport 10 % !
  Pwm1_Duty(PWM_CHB, ((float)0xffff*50)/100); // Rapport 50 % !
  Pwm1_Duty(PWM_CHC, ((float)0xffff*90)/100); // Rapport 90 % !

  // Active tous les canaux en même temps.
  Pwm1_AllOn();

  while(1);
}
```

Pwm3_Set

Fonction:
void Pwm3_Set (unsigned char **pMODE** , unsigned char **pFREQ**);

Explication:

pMODE	Configure le mode de PWM3.
pFREQ	Détermine la fréquence de PWM3.

Configure le mode de fonctionnement et la fréquence du signal PWM3 (voir ci-après). PWM1 dispose d'une résolution max. de 16 bits. Celle-ci peut être sélectionner pour fonctionner en mode 8, 9, 10 ou 16 bits et sur trois canaux (CHA, CHB et CHC). Chaque canal dispose d'une broche de sortie capable de générer un signal PWM. Contrairement aux autres signaux PWM, chaque broche/canal PWM3 devra être configuré sortie (car ceux-ci utilisent des broches E/S à usage général, sans quoi le signal PWM ne sera pas généré).

Pwm3_Duty

Fonction:
void Pwm3_Duty (unsigned char **pCHANNEL**, unsigned short **pDUTY**);

Explication:

pCHANNEL	Sélectionne un canal PWM3.
PDUTY	Sélectionne un rapport cyclique pourle canal **pCHANNEL** (Différent pour les modes 8, 9, 10 et 16 bits).

Le rapport cyclique de chaque canal peut être sélectionné.
Du fait que PWM3 puisse être configuré en mode 8, 9, 10 ou 16 bits, le rapport cyclique devra être configuré en conséquence. Par exemple le mode 16 bits permettra une sélection de 0 à 2^{16}. Il n'est pas nécessaire de configurer le rapport cylcique pour un canal non utilisé.

Pwm3_On, Pwm3_Off

Fonction:
void Pwm3_On (unsigned char **pCHANNEL**);
void Pwm3_Off (unsigned char **pCHANNEL**);

Explication:

Pwm3_On

pCHANNEL Active un canal PWM3.

Sort le signal PWM sur le canal **pCHANNEL**.

Pwm3_Off

pCHANNEL Désactive un canal PWM3.

Stop le signal sur la canal **pCHANNEL**.

Pwm3_AllOn, Pwm3_AllOff

Fonction:
void Pwm3_AllOn (void);
void Pwm3_AllOff (void);

Explication:

Pwm3_AllOn

Active les signaux PWM sur toutes les canaux PWM3.

Pwm3_AllOff

Désactive tous les signaux PWM des canaux PWM3.

Exemple:

```
void main(void)
{ // EXPB[3..5] sont partagée avec les sorties PWM3.
  PortSetMode(EXPB, 0xc7);  // Place les broches appropriées en sortie.
  // Utilise une horloge de 18.4320 MHz !
  // Utilise le mode 16 bits - fast mode !
Pwm3_Set(PWM_16BIT_FAST, PWM_FREQ_1);
Pwm3_Duty(PWM_CHA, ((float)0xffff*10)/100); // Rapport 10 % !
Pwm3_Duty(PWM_CHB, ((float)0xffff*50)/100); // Rapport 50 % !
Pwm3_Duty(PWM_CHC, ((float)0xffff*90)/100); // Rapport 90 % !

// Active tous les canaux en même temps !
Pwm3_AllOn();

  while(1);
}
```

RtcSetDate, RtcGetDate, RtcOn, RtcOff

Fonction:

```
void RtcSetDate( unsigned short pYEAR,
                 unsigned char pMONTH,
                 unsigned char pDATE,
                 unsigned char pHOUR,
                 unsigned char pMINUTE,
                 unsigned char pSECOND);

void RtcGetDate( unsigned short * pYEAR,
                 unsigned char * pMONTH,
                 unsigned char * pDATE,
                 unsigned char * pHOUR,
                 unsigned char * pMINUTE,
                 unsigned char * pSECOND);

void RtcOn(void);

void RtcOff(void);
```

Explication:

RtcSetDate

Configure la date courante et l'heure de l'horloge temps réel RTC (Real Time Clock) du ROVIN™ sous le format année, mois, date, heure, minutes, secondes en utilisant respectivement **pYEAR, pMONTH, pDATE, pHOUR, pMINUTE, pSECOND**.

RtcGetDate

Lit la date et l'heure depuis l'horloge RTC (Real Time Clock) sous le format année, mois, date, heure, minutes, secondes en retournant respectivement **pYEAR, pMONTH, pDATE, pHOUR, pMINUTE, pSECOND**.

RtcOn

Active l'horloge RTC.

RtcOff

Désactive l'horloge RTC.

RtcEventOn, RtcEventOff

Fonction:
void RtcEventOn(void);

void RtcEventOff(void);

Explication:

RtcEventOn

Active l'EVENEMENT RTC. Ceci permet de générer un EVENEMENT RTC toutes les secondes (lequel se traduit par l'aparition d'un MESSAGE RTC dans la file d'attente des EVENEMENTS).

Pour utiliser cette fonction, vous devez avoir préalablement activé la fonction EventOn() avant d'activer en premier tous les EVENEMENTS. Du fait que l'horloge RTC partage ses ressources avec la génération du signal PWM0, vous ne pourrez pas exploiter l'horloge en même temps que la génération de PWM0. **Si cette fonction est utilisée, PWM0 sera automatiquement stoppé**. De même, l'horloge RTC sera stoppée si le signal PWM0 est actif.

RtcEventOff

Désactive les EVEMENTS relatif à l'horloge RTC.

Exemple:

```
// Declare les variables pour le stockage de l'heure et de la date.

unsigned short YEAR;
unsigned char  MONTH, DATE, HOUR, MINUTE, SECOND;

// Gestion des EVENEMENT RTC.

void IRQ_EVENT_(void)
{
  if(GetMsg() == MSG_RTC) // Lorsqu'un EVENEMENT RTC survient...
{
  // Lecture de l'heure et de la date.
  RtcGetDate(&YEAR, &MONTH, &DATE, &HOUR, &MINUTE, &SECOND);

  DebugClear();
  DebugPrint("Le ROVIN de COMFILE TECHNOLOGY vous donne l'heure, \n");
  DebugSHORT(YEAR,DEC);
  DebugPrint(", ");
  DebugCHAR(MONTH,DEC);
  DebugPrint(", ");
  DebugCHAR(DATE,DEC);
  DebugPrint(", \n");
  DebugCHAR(HOUR,DEC);
  DebugPrint(" : ");
  DebugCHAR(MINUTE,DEC);
  DebugPrint(" : ");
  DebugCHAR(SECOND,DEC);
  DebugPrint(" : ");

beforeSECOND=SECOND;
}

void main(void)
{
// Initialise l'horloge RTC en la mettant à l'heure.
RtcSetDate(2005,00,00,00,00,00);
// Active les evenements RTC.
RtcOn();

// Active tous les EVENEMENTS.
EventOn();

// Attend un evènement.
  while(1);
}
```

Timer_SourceOn, Timer_SourceOff

Fonction:
void Timer_SourceOn (void);
void Timer_SourceOff (void);

Explication:

Timer_SourceOn

Active Timer_Source qui autorisera le fonctionnement des timers. Les Timers ne fonctionneront pas sans avoir appelé cette fonction.

Timer_SourceOff
Désactive Timer_Source et stoppe tous les Timers. Ceci aura pour conséquence de désactiver tous les EVENEMENTS Timers. Lorsque Timer_SourceOn() est appelé à nouveau, tous les Timers seront réactivés.

Timer_On, Timer_Off

Fonction:
void Timer_On (unsigned char **pNUM**);
void Timer_Off (unsigned char **pNUM**);

Explication:

Timer_On

pNUM Numéro du TIMER.

Chaque TIMER peut être indépendamment activé. Une fois activé, le réglage du Timer prendra effet et un EVENEMENT Timer pourra survenir.

Timer_Off

pNUM Numéro du TIMER.
Chaque TIMER peut être indépendamment désactivé. Aucun EVENEMENT sur ce Timer ne sera généré (les réglages du Timer restent sauvegardés).

Timer_SetTime

Fonction:
void Timer_SetTime (unsigned char **pNUM**, unsigned short **pTIME**);

Explication:

pNUM	Numéro du TIMER (T0 à T3).
pTIME	Valeur temporelle TIMER (Période TIMER EVENT d'unité 0.01 seconde.)

Configure les durées des TIMERS.
L'utilisateur peut configurer les valeurs temporelles des TIMERS avec des multiples de 0.01 seconds. Par exemple 100 unités créeront un EVENEMENT TIMER toutes les secondes.

Exemple:

```
void IRQ_EVENT_(void)
{
  if(GetMsg() == MSG_TIMER0)
  {
     DebugPrint("Un évenement TIMER0 est survenu !\n");
  }
}

void main(void)
{
  // Utilise le TIMER0.
  // 0.01 sec * 100 = 1 seconde.

  Timer_SetTime(T0, 100);     // Configure pour EVENEMENT chaque seconde.
  Timer_On(T0);               // TIMER0 ACTIF.

  Timer_SourceOn();           // Active la source du Timer.

  EventOn();                  // Autorise tous les EVENEMENTS.

  while(1);                   // Attend un EVENEMENT.
}
```

UartSetBaud

Fonction:
void UartSetBaud(unsigned char **pUARTport**, unsigned int **pBAUD**);

Explication:

pUARTport	N° du port UART (0 ou 1).
pBAUD	Vitesse en Bds (1200 à 57600 BPS).

Sélectionne le N° du port UART et sa vitesse de communication.

UartOn, UartOff

Fonctions:
void UartOn(unsigned char **pUARTport**);

void UartOff(unsigned char **pUARTport**);

Explication:
UartOn
Active l'UART N° **pUARTport** (0 ou 1).

UartOff
Désactive l'UART **pUARTport** (0 ou 1).
Lorsque l'UART est désactivé, il vous est possible d'utiliser ses broches associées pour des applications d'E/S générales.

UartWrite

Fonction:
void UartWrite(unsigned char **pUARTport**, unsigned char **pDATA**);

Explication:
Envoi un octet **pDATA** sur l'UART **pUARTpor**t (0 ou 1).

Exemple:

```
void main(void)
{
  int   i;

  UartSetBaud(0, 9600);       // Configure débit UART0 à 9600 bps.
  UartOn(0);                  // Active l'UART.

  // Envoie octets 0 à 255 à l'UART.
  for(i=0; i<=0xff; i++)
  {
    UartWrite(0, i);          // Envoi la valeur de i sur l'UART0.
  }
}
```

UartSetPacketSize

Fonction:

void UartSetPacketSize(unsigned char **pUARTport**, unsigned char **pSIZE**);

Explication:
Détermine la taille de réception du buffer de l'UART. Bien que vous n'ayez pas besoin de cette fonction lorsque vous envoyez des données, le buffer devra être utilisé en réception afin de pouvoir générer un EVENEMENT (interruption) lorsque ce dernier est plein. Il conviendra de sélectionner la taille de celui-ci avec soin pour obtenir une efficacité et un traitement optimale. Si par exemple vous choisissez un buffer de 1 octet, vous risquez en cas de réception d'un grand nombre de données de créer un dépassement de la capacité de la file d'attente des EVENEMENT liés à l'UART (puisqu'une interruption sera générée à chaque octet !). La taille du buffer peut être configurée de 1 à 255 octets.

Exemple:

```
void IRQ_EVENT_(void) {
 if(GetMsg() == MSG_UART0RX) // Test si réception EVENEMENT de l'UART0.
{ DebugPrint("Un paquet de 100 octets est arrivé via l'UART0.\n");
  }
}

void main(void)
{ UartSetPacketSize(0, 100);  // Taille buffer de l'UART0 = 100 octets.
  UartSetBaud(0, 115200);     // Configure débit à 115200 BPS.
  UartRxOn(0);                // Active l'UART0 !
  UartEventOn(0);             // Autorise EVENEMENT sur UART0.

  EventOn();                  // Autorise tous les EVENEMENTS.
  DebugClear();               // Efface fenêtre Debug du PC.

  while(1);                   // Attend un EVENEMENT.
}
```

UartRxBufRead

Fonction:
void UartRxBufRead (unsigned char **pUARTport**, unsigned char * **pBUFFER**);

Explication:

pUARTport	N° du port UART (0 ou 1).
pBUFFER	Buffer de récupération des données reçues

Lit le paquet de données depuis le buffer de l'UART.

Exemple:

```
#define  PACKET_SIZE  100            // Taille buffer = 100 octets.

void IRQ_EVENT_(void)
{ unsigned char iBUF[PACKET_SIZE];
  if(GetMsg() == MSG_UART0RX)          // Test réception EVENEMENT UART0?
  { UartRxBufRead(0, iBUF);               // Récupère données dans Buffer
     UartBufOut(0, iBUF, PACKET_SIZE);    // Renvoi les données en retour.
  }
}

void main(void)
{ UartSetPacketSize(0, PACKET_SIZE);  // Configure la taille du biffer.
  UartSetBaud(0, 115200);            // Configure débit à 115200 BPS.
  UartOn(0);                         // Active l'UART0.
  UartEventOn(0);                    // Autorise EVENEMENT sur UART0.

  EventOn();                         // Autorise tous les EVENEMENTS.
  while(1);                          // Attend un EVENEMENT.
}
```

UartRxBufCountRead

Fonction:
unsigned short UartRxBufCountRead(unsigned char **pUARTport**);

Explication:
pUARTport N° du port UART (0 ou 1).
Retourne Valeur Permet de connaître le nombre de données présente dans le buffer.

Exemple:

```
void main(void)
{
  unsigned char iDATA;

  UartSetBaud(0, 115200);            // Configure débit à 115200 BPS.

  UartOn(0);                         // Active l'UART0.

  while(UartRxBufCountRead(0)<1);    // Atttend qu'une données soit reçue

  iDATA = UartRead(0);        // Récupère cet octet dans le buffer.
}
```

UartTxBufWrite, UartBulkOut

Fonction:
void UartTxBufWrite(unsigned char * **pBUFFER**, unsigned char **pCOUNT**);

void UartBulkOut(unsigned char **pUARTport**);

Explication:

UartTxBufWrite

pBUFFER Buffer à envoyer.
pCOUNT Taille du buffer en octets.

Place les données **pBUFFER** dans un buffer temporaire (les données ne sont pas envoyées vers l'UART, mais seulement stockée dans le buffer d'envoi).

Cette fonction doit être employée de « concert » avec UartBulkOut(). UartBulkOut() enverra les données placées dans le buffer temporaire vers l'UART. La taille maxi. Du buffer est de 255 octets.

UartBulkOut

pUARTport N° du port UART (0 ou 1).

Envoi immédiatement les données placées dans le buffer temporaire du ROVIN™ via la fonction UartTxBufWrite() vers l'UART.

Exemple:

```
void main(void)
{
  unsigned char       iBUFFER[100];
  int          i;

  UartSetBaud(0, 230400);            // Configure débit à 230400 BPS.

  UartOn(0);                         // Active l'UART0.

  for(i=0;i<100;i++)
  {
    iBUFFER[i]=i;                    // Sauve 0 à 99 dans le tableau.
  }

  UartTxBufWrite(iBUFFER, 100);     // Rempli buffer avec contenu tableau.

  UartBulkOut(0);                    // Envoi le tout vert UART0.
}
```

UartStringOut

Fonction:
void UartStringOut(unsigned char **pUARTport**, char * **pSTR**);

Explication:

pUARTport	N° du port UART (0 ou 1).
pSTR	Chaîne à envoyer.

Envoie une chaîne de caractères vers l'UART.

UartBufOut

Fonction:
void UartBufOut(unsigned char **pUARTport**,
unsigned char * **pBUF**,
unsigned short **pBUFsize**);

Explication:

pUARTport	N° du port UART (0 ou 1).
pBUF	Données à envoyer.
pBUFsize	Nombre de données à envoyer.

Envoie les données l'UART (nécessite de savoir combien de données on envoi). Taille max. : 255 octets.

UartWrChar, UartWrShort, UartWrInt, UartWrLong, UartWrDouble

Fonction:
void UartWrChar(unsigned char **pUARTport**, unsigned char **pCHARdata**);

void UartWrShort(unsigned char **pUARTport**, unsigned short **pSHORTdata**);

void UartWrInt(unsigned char **pUARTport**, unsigned int **pINTdata**);

void UartWrLong(unsigned char **pUARTport**, unsigned long **pLONGdata**);

void UartWrDouble(unsigned char **pUARTport**, double **pDOUBLEdata**);

Explication:
UartWrChar
Envoi donnée d'1 octet vers **pUARTport** .

UartWrShort
Envoi donnée de 2 octets vers **pUARTport**.

UartWrInt
Envoi donnée de 4 octets vers **pUARTport**.

UartWrLong
Envoi donnée 8 octets vers **pUARTport** en commençant par octets de poid fort.

UartWrDouble
Envoi 8 octets vers (avec virgule flottante) **pDOUBLEdata** depuis la mémoire.

Exemple:

```
void main(void) {
  UartSetBaud(0, 115200);     // Configure débit 115200 BPS.
  UartOn(0);                  // Active l'UART0.
  UartWrChar(0, 0x12);
  UartWrShort(0, 0x1234);
  UartWrInt(0, 0x12345678);
  UartWrLong(0, 0x123456789abcdef0);
  UartWrDouble(0, 3.14);
}
```

UartTxOn, UartTxOff

Fonction:
void UartTxOn(unsigned char **pUARTport**);
void UartTxOff(unsigned char **pUARTport**);

Explication:

UartTxOn

pUARTport N° du port UART (0 ou 1).

Permet la communication de la broche TXD de l'UART. Pour envoyer des données via l'UART, il vous faudra utiliser cette fonction et la broche TXD devra être configurée en sortie (sinon, les données ne seront pas émises).

UartTxOff

PUARTport N° du port UART (0 ou 1).

Stoppe la communication TXD de l'UART.

Exemple:

```
void main(void)
{
  UartSetBaud(0, 115200);          // Configure le débit à 115200 BPS.
  UartTxOn(0);                     // Active TXD de l'UART0.

  UartWrite(0, 0xab);        // Trasfert 0xab vers l'UART0.

  UartTxOff(0);              // Stoppe l'acces TXD de l'UART0.

  while(1);
}
```

UartRxOn, UartRxOff

Fonction:
void UartRxOn(unsigned char **pUARTport**);

void UartRxOff(unsigned char **pUARTport**);

Explication:

UartRxOn

pUARTport N° du port UART (0 ou 1).

Permet la communication de la broche RXD de l'UART. Pour recevoir des données via l'UART, il vous faudra utiliser cette fonction sinon les données ne seront pas reçues.

UartRxOff

PUARTport N° du port UART (0 ou 1).

Stoppe la communication RXD de l'UART. La réception est ignorée et les données ne sont plus stockées dans le buffer.

Exemple:

```
void IRQ_EVENT_(void)
{ if(GetMsg() == UART0RX)
  { UartRxOff(0); // Après la réception d'1 octet, stoppe la réception.
  }
}

void main(void)
{
  UartSetPacketSize(0, 1);    // Configure taille buffer pour interruption.
  UartSetBaud(0, 115200);     // Configure débit à 115200 BPS.
  UartRxOn(0);                // Active l'UART0.

  UartEventOn(0);             // Autorise génération EVENEMENT UART0.
  EventOn();                  // Autorise tous les EVENEMENTS.

  while(1);                   // Attend un EVENEMENT.
}
```

UartEventOn, UartEventOff

Fonction:
void UartEventOn(unsigned char **pUARTport**);

void UartEventOff(unsigned char **pUARTport**);

Explication:

UartEventOn

pUARTport N° du port UART (0 ou 1).

Permet la génération d'EVENEMENTS UART (interruption) lorsqu'un « paquet » de données a rempli le buffer (la taille du buffer devra être préalablement définie avec la fonction UartSetPacketSize()). De plus EventOn() devra être appelé pour que la fonction puisse fonctionner.

UartEventOff

pUARTport N° du port UART (0 ou 1).

Stoppe la génération d'EVENEMENTS UART (interruption). Ceci stoppe uniquement la génération des EVENEMENTS, les données continuent tout de même à remplir le buffer de réception. **Dès lors un dépassement de la capacité du buffer peut intervenir puisque les données peuvent toujours être reçues.**

UartBufClear

Fonction:
void UartBufClear(unsigned char **pUARTport**);

Explication:
Efface le contenu du buffer de réception de l'UART **pUARTport**.

Exemple

```
void main(void) {
UartBufClear(0);              // Efface le buffer de réception de l'UART0.
UartSetPacketSize(0, 1);      // Taille buffer de réception = 1 octet.
UartSetBaud(0, 115200);       // Débit de communication = 115200 BPS .
UartRxOn(0);                  // Active l'UART0.
UartEventOn(0);               // Permet génération EVENEMENTS UART0.
EventOn();                    // Permet génération EVENEMENTS

  while(1);                   // Attend un évenement
}
```

VmTimerSetTime, VmTimerGetTime

Fonction:
void VmTimerSetTime(unsigned short **pSECOND**);
unsigned short VmTimerGetTime(void);

Explication:

VmTimerSetTime

Charge le TIMER VMTIMER avec un multiple de 10 ms.
Lorsque VTIMER arrive à zéro, un EVENEMENT survient.

VmTimerGetTime

Lit et retourne la durée rstante dans VMTIMER.

Exemple:

```
int     vmCOUNT;

void IRQ_EVENT_(void)
{ if(GetMsg() == MSG_VMTIMER)    // Regarde si EVENEMENT VTIMER ?
   { vmCOUNT++;                  // Oui -> Comptabilise ces EVENEMENT.
     DebugClear();
     DebugPrint("VMTIMER a ete déclenché ");
     DebugINT(vmCOUNT, DEC);
     DebugPrint(" fois.\n");
   }
}

void main(void)
{
  vmCOUNT = 0;   // Initialise variable vmCOUNT.

RtcOn();         // RTC doit être activé pour utiliser VTIMER.

VmTimerSetTime(100);  // VMTIMER initialisé pour 100 x 10 ms = 1 sec.
  VmTimerAutoLoadOn();// Recharge automatique de VMTIMER.
  VmTimerOn();        // Démarre VMTIMER.

  EventOn();          // Active tous les évenements.

  while(1);           // Attend pour un èvenement.
}
```

VmTimerOn, VmTimerOff

Fonction:
void VmTimerOn(void);

void VmTimerOff(void);

Explication:

VmTimerOn

Active le Timer VMTIMER. Les EVENEMENTS VMTIMER seront autorisés.
RtcOn() doit être appelé pour utiliser VmTimer car ils partagent les mêmes ressources.

void VmTimerOff

Désactive le Timer VMTIMER. Les EVENEMENTS VMTIMER seront inhibés.

VmTimerAutoLoadOn, VmTimerAutoLoadOff

Fonction:
void VmTimerAutoLoadOn(void);

void VmTimerAutoLoadOff(void);

Explication:

VmTimerAutoLoadOn

Si vous lancez la fonction VmTimerAutoLoadOn, Le Timer sera automatiquement initialisé avec la valeur initiale (configurée par VmTimerSetTime()) lorsque ce dernier atteind zéro.

VmTimerAutoLoadOff

Désactive cette fonction.

COMFILE
TECHNOLOGY

MATH

Le « ROVIN-C » supporte la plupart des fonctions mathématiques du standard « ANSI-C ». Tous les calculs mathématiques sur font en mode 8 octets doubles.

Fonctions supportée:

- long abs(long pX); - Nombre absolu
- double fabs(double pX); - Nombre absolu (virgule flottante)
- double acos(double pX); - Arc cosinus
- double asin(double pX); - Arc sinus
- double atan(double pX); - Arc tangente
- double atan2(double pY, double pX); - Double Arc tangente
- double cos(double pX); - Cosinus
- double sin(double pX); - Sinus
- double tan(double pX); - Tangente
- double cosh(double pX); - Cosinus hyperbolique
- double sinh(double pX); - Sinus hyperbolique
- double tanh(double pX); - Tangente hyperbolique
- double exp(double pX); - e^{pX}
- double frexp(double pX, int * pEXPONENT); $2^{pEXPONENT} * Y = pX$
- double ldexp(double pX, int pEXPONENT); $2^{pEXPONENT} * Y = pX$
- double log(double pX); - $log_1(pX)$
- double log10(double pX); - $log_{10}(pX)$
- double logb(double pX); - $log_2(pX)$
- double modf(double pX, double * pIPART); F = modf(3.14, I); // I = 3, F = 0.14
- double pow(double pX, double pY); - pX^{pY}
- double sqrt(double pX); - Racine carée
- double ceil(double pX); - Arrondi vers le haut
- double floor(double pX); - Arrondi vers le bas
- double fmod(double pX, double pY); - Mod point décimal, Idem mod(), mais avec calcul point décimal.
- double cbrt(double **pX**); - $\sqrt[3]{\mathbf{pX}}$ = Racine cubique de la valeur

Constantes mathématiques suportées
- _pi : 3.14159265358979323846
- _e : 2.71828182845904523536

srand, rand

Fonction:
void srand(unsigned int **pSEED**);

int rand(void);

Explication:

srand

pSEED : Initialise la procédure de génération de nombres aléatoires.
La fonction Srand() doit être appelée au moins une fois avant d'appelé la fonction rand().

rand

Crée un nombre aléatoire.

Exemple:

```
void Delay(int pTIME)
{ while(pTIME--);
}

void main(void)
{
  int iRANDnumber;

  srand(1234); // Initialisation fonction pour création NB aléatoire.

  DebugClear();
  while(1)
  {
    iRANDnumber=rand();              // Génère un nombre aléatoire.
    DebugINT(iRANDnumber,DEC);       // Affiche un nombre aléatoire.
    DebugPrint("\n");
    Delay(0xfff);                    // Délai pour affichage.
  }
}
```

itoa

Fonction:
void itoa(int **pVALUE**, char * **pBUF**, char **pBUFsize**);

Explication:

pVALUE	Valeur à convertir en une chaîne.
pBUF	Buffer pour sauvegarder la chaîne sauvegardée.
pBUFsize	Taille du buffer.

Conversion valeur numérique type « int » (4 octets) en une chaîne.

Exemple:

```
void main(void)
{
  char iINTstr[50];

  DebugClear();

  itoa(-2147483647,&iINTstr,50);
  DebugPrint("Résultat de itoa(-2147483647,&iINTstr,50); \n\n[ ");
  DebugPrint(iINTstr);
  DebugPrint(" ]\n\n");

  itoa(2147483647,&iINTstr,50);
  DebugPrint("Résultat de itoa (2147483647,&iINTstr,50); \n\n[ ");
  DebugPrint(iINTstr);
  DebugPrint(" )\n\n");

  itoa(-002147.1234,&iINTstr,50);
  DebugPrint("Résultat de itoa (-002147.1234,&iINTstr,50); \n\n[ ");
  DebugPrint(iINTstr);
  DebugPrint(" ]\n\n");
}
```

ltoa

Fonction:
void ltoa(long p**VALUE**, char * **pBUF**, char **pBUFsize**);

Explication:
pVALUE Valeur à convertir en une chaîne.
pBUF Buffer pour sauvegarder la chaîne sauvegardée.
pBUFsize Taille du buffer.

Conversion valeur numérique type « long » (8 octets max.) en une chaîne.

ftoa

Fonction:
void ftoa(double **pVALUE**,
char * **pBUF**,
char **pBUFsize**,
char **pDIGIT**);

Explication:
pVALUE Valeur à convertir en une chaîne.
pBUF Buffer pour auvegarder la chaîne sauvegardée.
pBUFsize Taille du buffer.
pDIGIT Nombre de digits à droite du point décimal.

Conversion valeur numérique type « long avec virgule flottante» (8 octets max.) en une chaîne. La conversion s'effectue sur 15 digits (y compris les chiffres après le point décimal).

Exemple:

```
void main(void)
{
  char iDOUBLEstr[50];

  DebugClear();

  // Affiche jusqu'à 12 position après le point décimal
  ftoa(-3.14159265359,&iDOUBLEstr,50,12);
  DebugPrint("Résultat de ftoa(-3.14159265359,&iDOUBLEstr,50,12);"
          "is:\n\n[ ");
  DebugPrint(iDOUBLEstr);
  DebugPrint(" ]\n\n");
}
```

SOUND

Pwm1_Set,Pwm3_Set (SOUND), Pwm1_On, Pwm1_Off, Pwm3_On, Pwm3_Off

Fonction:
void Pwm1_Set (unsigned char **pMODE**, unsigned char **pFREQ**);

void Pwm3_Set (unsigned char **pMODE**, unsigned char **pFREQ**);

void Pwm1_On (unsigned char **pCHANNEL**);

void Pwm1_Off (unsigned char **pCHANNEL**);

void Pwm3_On (unsigned char p**CHANNEL**);

void Pwm3_Off (unsigned char **pCHANNEL**);

Explication:

Pwm1_Set

pMODE Configure en mode "**PWM_SOUND"** SEULEMENT.
pFREQ La valeur de **pFREQ** n'a pas d'importance.

Cette fonction permet de configurer uniquement le canal SOUND-CHA. Contrairement aux autres réglages PWM, vous devrez utiliser **"PWM_SOUND"** pour **pMODE** et mettre la valeur 0 pour **pFREQ.** En utilisant cette fonction PWM1 sera automatiquement configuré pour générer des sons sur le canal SOUND-CHA.

Exemple Usage: Pwm1_Set(PWM_SOUND, 0);

Pwm3_Set

pMODE Configure en mode "**PWM_SOUND"** SEULEMENT.
pFREQ La valeur de **pFREQ** n'a pas d'importance.

Cette fonction permet de configurer uniquement le canal SOUND-CHB. Contrairement aux autres réglages PWM, vous devrez utiliser **"PWM_SOUND"** pour **pMODE** et mettre la valeur 0 pour **pFREQ.** En utilisant cette fonction PWM3 sera automatiquement configuré pour générer des sons sur le canal SOUND-CHB.

Exemple Usage: Pwm3_Set(PWM_SOUND, 0);

Pwm1_On, Pwm1_Off
Active/Désactive le canal « sons » SOUND-CHA.

Pwm3_On, Pwm3_Off
Active/Désactive le canal « sons » SOUND-CHA.

Sound, SoundFreq

Fonction:
void Sound(unsigned char **pCHANNEL**,
unsigned char **pOCTAVE**,
unsigned short **pTONE**);

void SoundFreq(unsigned char **pCHANNEL**,
double **pFREQ**);

Explication:

Sound
pCHANNEL Utilisez SOUND-CHA ou SOUND-CHB SEULEMENT.
pOCTAVE Sélectionnez l'octave à utiliser pour pOCTAVE.
pTONE Sélectionnez la tonalité (selon un format de 2 octets pré-définis)

Génère une tonalité avec un octave donné sur un canal SON. L'octave peut être choisi entre 0 to 255 (mis si la valeur est trop grande le son rendu ne sera pas réaliste). Un octave avec une valeur de 2 correspond « au milieu » de piano.

Exemple Usage: Sound(SOUND-CHA, 2, 'G_'); // Génère tonalité 'G' (octave 2).

SoundFreq
pCHANNEL Utilisez SOUND-CHA ou SOUND-CHB SEULEMENT.
pDOUBLE Valeur avec virgule flottante pour la fréquence.

Sélection de la fréquence du canal SOUND. Contrairement à la fonction Sound(), l'utilisateur pourra utiliser SoundFreq s'il connaît la fréquence exacte à générer. La fréquence doit être comprise dans la gamme **140Hz ~ 900KHz**.

Exemple Usage:
SoundFreq(SOUND-CHB, 164.8); // Génère fréquence 164.8 Hz sur le canal SOUND-B.

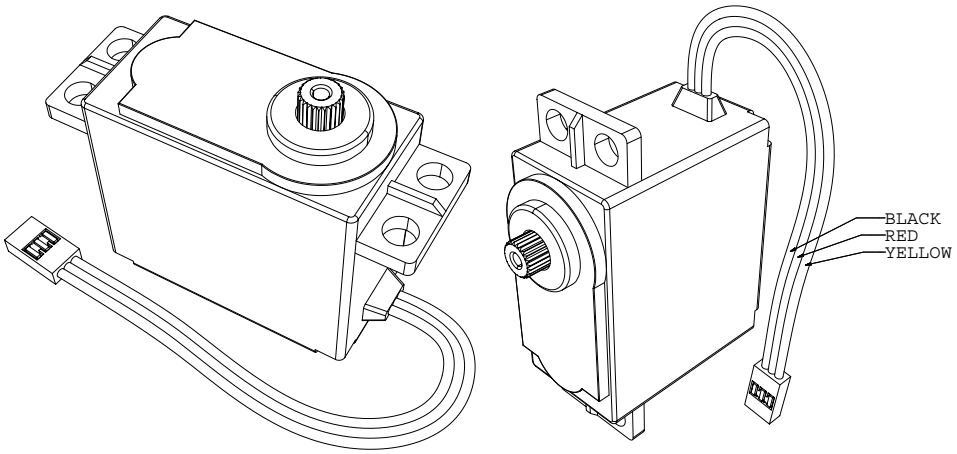
Exemple:

```
void main(void)
{
  Pwm1_Set(PWM_SOUND, 0);     // Configure PWM1 en SOUND-CHA.
  Pwm1_On(PWM_CHA);           // Active canal SOUND-CHA.
  Pwm3_Set(PWM_SOUND, 0);     // Configure PWM3 en SOUND-CHB.
  Pwm3_On(PWM_CHA);           // Active canal SOUND-CHB.

  Sound(SOUND_CHA, 2, 'G_');  // Génère "G" sur octave 2.
  SoundFreq(SOUND_CHB, 196);  // Génère "G" très bas en fréquence.
    // 196 Hz correspond à la note "G" et l'octave 0.
  while(1);
}
```

COMFILE
TECHNOLOGY

RC-SERVO



RC-SERVO (Servomoteur type modélisme)
Black = noir Red = Rouge Yellow = Jaune (ou blanc)

Pwm1_Set,Pwm3_Set (RC-SERVO)
Pwm1_On, Pwm1_Off, Pwm3_On, Pwm3_Off

Fonction:
void Pwm1_Set (unsigned char **pMODE**, unsigned char **pFREQ**);
void Pwm3_Set (unsigned char **pMODE**, unsigned char **pFREQ**);

void Pwm1_On (unsigned char **pCHANNEL**);
void Pwm1_Off (unsigned char **pCHANNEL**);

void Pwm3_On (unsigned char **pCHANNEL**);
void Pwm3_Off (unsigned char **pCHANNEL**);

Explication:

Pwm1_Set

pMODE Configure **PWM_RCSERVO** pour gestion de servos.
pFREQ Détermine période PWM pour les SERVOS en millisecondes. Généralement compris entre 10 et 20 ms (consultez la notice de votre servomoteur pour connaître la valeur exacte).

Pour le pilotage de 3 servos, RC-SERVO[0:2] les sorties PWM1A/B/C seront en mode RC-SERVO.

Exemple:
```
Pwm1_Set(PWM_RCSERVO, 20);      // Configure ports PWM1 en mode RC-SERVO.
                                // Et sélection période à 20 ms.
```

Pwm3_Set

pMODE Configure **PWM_RCSERVO** pour gestion de servos.
pFREQ Détermine période PWM pour les SERVOS en millisecondes. Généralement compris entre 10 et 20 ms (consultez la notice de votre servomoteur pour connaître la valeur exacte).

Pour le pilotage de 3 autres servos, RC-SERVO[3:5] les sorties PWM3A/B/C seront en mode RC-SERVO.

Exemple:
```
Pwm1_Set(PWM_RCSERV3, 20);      // Configure ports PWM3 en mode RC-SERVO.
                                // Et sélection période à 20 ms.
```

Pwm1_On, Pwm1_Off

Active/désactive le signal sur les sorties PWM1-CH[A:C] destiné au pilotage des servo-moteurs RC-SERVO[0:2]. En absence de signaux les servos ne bougent pas.

Exemple Usage:

```
Pwm1_On(PWM_CHA);   // Génère PWM pour RC-SERVO-0.
Pwm1_Off(PWM_CHC);  // Stoppe PWM sur RC-SERVO-2.
```

Pwm3_On, Pwm3_Off

Active/désactive le signal sur les sorties PWM3-CH[A:C] destiné au pilotage des servo-moteurs RC-SERVO [3:5]. En absence de signaux les servos ne bougent pas.

Exemple Usage:

```
Pwm3_On(PWM_CHA);   // Génère PWM pour RC-SERVO -3.
Pwm3_Off(PWM_CHB);  // Stoppe PWM sur RC-SERVO -4.
```

Pwm1_Duty, Pwm3_Duty (RC-SERVO)

Fonction:
void Pwm1_Duty (unsigned char **pCHANNEL**, unsigned short **pDUTY**);

void Pwm3_Duty (unsigned char **pCHANNEL**, unsigned short **pDUTY**);

Explication:

Pwm1_Duty

pCHANNEL	Sélectionne canal PWM1 (Utilisez PWM-CHA , PWM-CHB ou PWM-CHC).
PDUTY	Sélectionnez le rapport cyclique du signal 16 bits PWM.

Cette fonction généralement utilisée après l'appel à Pwm1_set(), génère un signal PWM sur la sortie PWM1 (pour piloter les servos RC-SERVO[0:2], suivant le canal sélectionné) avec un rapport cyclique sur 16 bits. Il vous faudra dans un premier temps trouver la correspondance entre le rapport cyclique du signal et la position du servomoteur en degré. Vous devrez ensuite utiliser la fonction RcServo_Regulate() pour pouvoir piloter directement les servomoteurs en indiquant leur position en degré (voir explication RcServo_Regulate()).

Exemple:

```
Pwm1_Duty(PWM_CHA, 0x600);
// Rapport cyclique pour position 0 degré (peut être différent suivant votre servo)
```

Pwm3_Duty

PCHANNEL Sélectionne canal PWM3 (Utilisez PWM-CHA , PWM-CHB ou PWM-CHC).

PDUTY Sélectionnez le rapport cyclique du signal 16 bits PWM.

Cette fonction généralement utilisée après l'appel à Pwm1_set(), génère un signal PWM sur la sortie PWM3 (pour piloter les servos RC-SERVO[3:5], suivant le canal sélectionné) avec un rapport cyclique sur 16 bits. Il vous faudra dans un premier temps trouver la correspondance entre le rapport cyclique du signal et la position du servomoteur en degré. Vous devrez ensuite utiliser la fonction RcServo_Regulate() pour pouvoir piloter directement les servomoteurs en indiquant leur position en degré (voir explication RcServo_Regulate()).

Exemple:

```
Pwm3_Duty(PWM_CHC, 0x1390);
// Rapport cyclique pour position 180 degré (peut être différent suivant les servos)
```

RcServo_Regulate

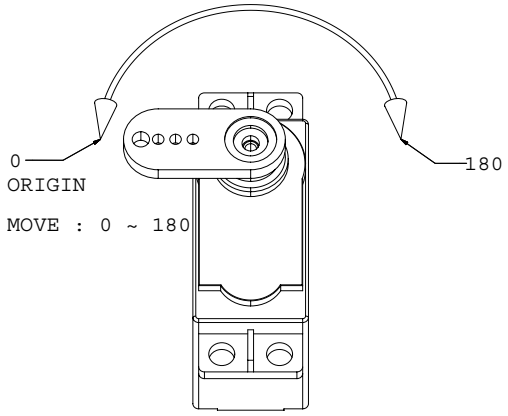
Fonction:
void RcServo_Regulate(unsigned char **pID**,
unsigned short **pZEROduty**,
unsigned char **pCOMPdeg**,
unsigned short **pCOMPduty**);

Explication:

pID	N° du SERVO (Entre 0 et 5).
pZEROduty	Valeur du rapport cyclique du signal PWM lorque le SERVO est en position de départ.
pCOMPdeg	Angle au point d'arrivé du servomoteur.
pCOMPduty	Valeur du rapport cyclique correspondant à **pCOMPdeg**.

Cette fonction permet de déterminer automatiquement et avec précision la correspondance entre chaque position d'un servomoteur et les valeurs du rapport cycliques du signal PWM à générer. Une fois exécutée il vous sera alors possible de piloter chaque servo (avec la fonction RcServo_Move()) directement en indiquant l'angle sur lequel il doit se positionner. Avec cette méthode, il est possible d'obtenir une précision de l'ordre de 0.03 degré (pour un déplacement de 0 à 180 degré – en fonction des servomoteurs).

Méthode d'utilisation pour le servomoteur 0.



Positionnez le servomoteur complètement à gauche (origine 0 degré) en utilisant la fonction Pwm1_Duty (déterminez la valeur du rapport cyclique pour obtenir la position exacte : notez cette valeur que nous appelerons DUTY-A). Positionnez ensuite le servomoteur à l'extrême droite (angle 180 degré) toujours en utilisant la fonction Pwm1_Duty (déterminez la valeur du rapport cyclique pour obtenir cette position exacte : notez cette valeur que nous appelerons DUTY-B).

Il vous suffit alors d'appeler la fonction RcServo_Regulate() comme indiqué ci-dessous.

Exemple:
RcServo_Regulate(0, DUTY-A, 180, DUTY-B);

D'autres position d'origine et de « fin » peuvent être utilisés suivant vos applications.

RcServo_Move

Fonction:
void RcServo_Move(unsigned char **pID**, double **pDEGREE**);

Explication:

pID	N° du SERVO (Entre 0 et 5).
pDEGREE	Angle à atteindre. Un nombre avec virgule flottante sur 8 octets peut être utilisé pour une précision extrême.

Demande au servomoteur de se positionner sur l'angle déterminé par **pDEGREE.** La valeur de **pDEGREE** peut être négative (exemple plage : -90 à 90 degré).

En fonction de votre servomoteur, une précision de 0,03 degré pourra être obtenue pour un déplacement sur une plage globale de 0 à 180 °. Avant d'utiliser cette fonction, il vous faudra utiliser RcServo_Regulate() et activer la sortie PWM adéquate.

Exemple:

```
void main(void)
{
 // Initialise RC-SERVO[3:5].
   Pwm3_Set(PWM_RCSERVO,15);          //  Configure PWM3 en mode RC-SERVO.
                                      // Configure période du servo à 15 ms.

RcServo_Regulate(3, 0x600, 180,   0x1390);
    // Utilise le servomoteur N° 3.
    // Détermine valeur rapport cyclique du signal PWM pour position 0 degré  (0x600)
    // Détermine la position extrême 180 degré
    // Détermine valeur rapport cyclique du signal PWM pour position 180 degré  (0x1390)

   // On peut ensuite piloter le servomoteur avec précision !

 Pwm3_On(PWM_CHA); // Autorise l'activation sur signal PWM pour le SERVO N° 3.

 RcServo_Move(3, 90);              // Déplace servo sur position 90° (milieu).
 Delay(0xfff);

 RcServo_Move(3, 180);             // Déplace servo sur position 180° (à droite).
 Delay(0xfff);

RcServo_Move(3, 90);               // Déplace servo sur position 0° (à gauche).
 Delay(0xfff*2);
}
```

COMFILE
TECHNOLOGY

SPI™

SPI™ - Serial Peripheral Interface

Le port SPI™ est un mode de communication série développé par la société Motorola™ (Freescale™). **Le ROVIN™ est capable de gérer jusqu'à 8 ports SPI™ simultanément.** Seul le mode de communication SPI™ maître est supporté par le ROVIN™. Vous pouvez créer et contrôler indépendamment vos ports de communication SPI™ de façon dynamique en fonction de vos applications. Les ports XPPI PA à PD seront utilisables pour gérer les signaux SPI™.

SpiCreate

Fonction:

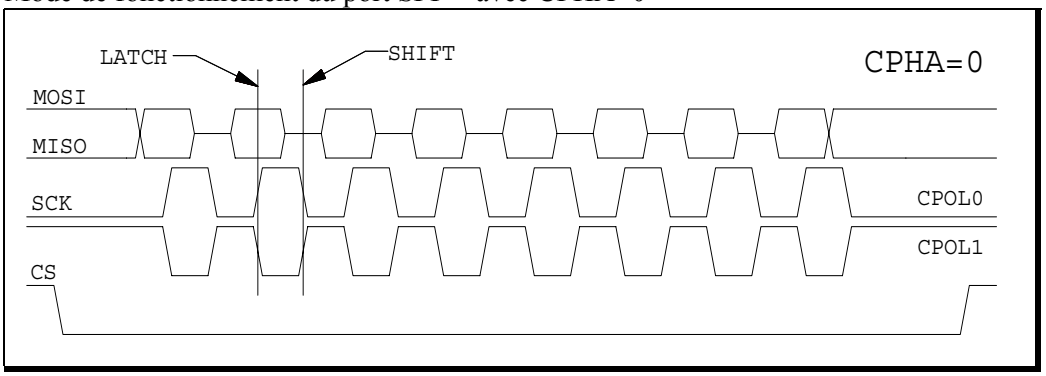
void SpiCreate(unsigned char **pSPIid**,
unsigned char **pUSEport**,
unsigned char **pMOSIpin**,
unsigned char **pMISOpin**,
unsigned char **pSCKpin**,
unsigned char **pCSpin**,
unsigned char **pSPImode**,
unsigned char **pOUTmode**,
unsigned int **pSLOWtime**);

Explication:

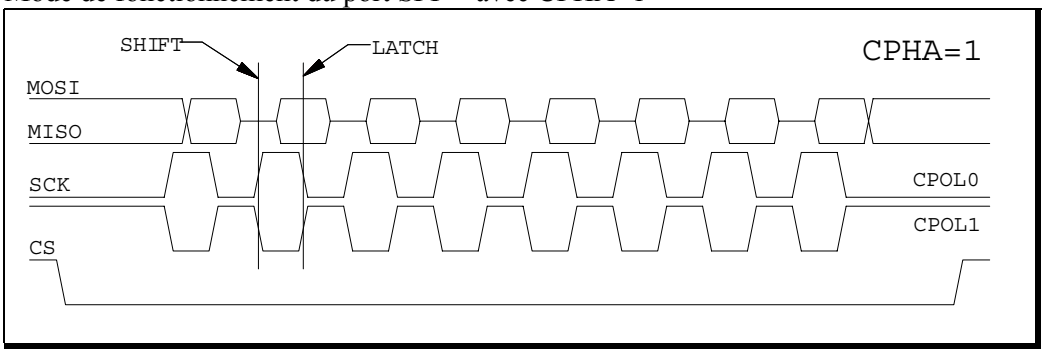
pSPIid	Port SPI™ à créer (Utilisez 0 à 7).
PUSEport	Port XPPI monopolisé pour la gestion SPI (PA, PB, PC, ou PD).
pMOSIpin	Broche utilisée pour signal MOSI du port *pUSEport* (4 bits poids faible). 0 ou 1 pour valeur d'initialisation (4 bits poids fort).
pMISOpin	Broche utilisée pour signal MISO du port *pUSEport*. Pas de valeur d'initialisation car c'est une entrée.
pSCKpin	Broche utilisée pour signal SCK du port *pUSEport* (4 bits poids faible). 0 ou 1 pour valeur d'initialisation (4 bits poids fort).
pCSpin	Broche utilisée pour signal CS du port *pUSEport* (4 bits poids faible). 0 ou 1 pour valeur d'initialisation (4 bits poids fort).
pSPImode	Détermine mode trigger du SPI™. 4 bits poids fort: SPI_CPHA0, SPI_CPHA1 4 bits poids faible: SPI_CPOL0, SPI_SPOL1. Utilisez le catatère '|' entre CPHA et CPOL. Par exemple le mode "SPI_CPHA0|SPI_CPOL0".
pOUTmode	SPI_MSBFIRST ou SPI_LSBFIRST, décidera si vous désirez sortir en premier les MSB ou LSB.
pSLOWtime	Détermine la vitesse du port SPI™. 0 correspond à la vitesse max. En augmentant ce paramètre, la vitesse de communication décroîtra.

SpiCreate() créé un port SPI™ sur un des ports XPPI (PA, PB, PC, ou PD) du module ROVIN™ en initialisant les broches MOSI, MISO, SCK et CS et en les attribuant à des broches du module. Le mode « SPI trigger » et les modes « MSB/LSB » sont également sélectionnés avec cette fonction.

Mode de fonctionnement du port SPI™ avec CPHA=0



Mode de fonctionnement du port SPI™ avec CPHA=1



Exemple:

```
SpiCreate(2, PA,      // Création port SPI™ N° 2 en utilisant le port XPPI-PA du ROVIN™
     (0<<4)|3,              // Assigne signal MOSI sur broche PA.3 et initialise au niveau 0.
      2,                    // Assigne signal MISO sur broche PA.2.
      (0<<4)|1,             // Assigne signal  SCK sur broche PA.1 et initialise au niveau 0.
      (1<<4)|0,             // Assigne signal CS sur broche PA.0 et initialise au niveau 1.
      SPI_CPHA0|SPI_CPOL0,      // Utilise modes CPHA0 et CPOL0.
      SPI_MSBFIRST,         // Configure pour sortie des MSB en premier
      0);                   // Configure vitesse maximale du port SPI™.
```

SpiOut, SpiIn

Fonction:
void SpiOut(unsigned char pSPIid, unsigned char **pDATA**);

unsigned char SpiIn(unsigned char **pSPIid**);

Explication:

pSPIid	Port SPI™ à utiliser (Utilisez 0 à 7).
pDATA	Donnée (d'1 octet) à envoyer.

SpiOut() envoi un octet sur le port SPI™. Le port SPI™ doit être préalablement créé avec la fonction SpiCreate().

SpiIn

pSPIid Port SPI™ à utiliser (Utilisez 0 à 7).

SpiIn() lit un octet depuis le port SPI™. Le port SPI™ doit être préalablement créé avec la fonction SpiCreate().

SpiBitOut, SpiBitIn

Fonction:
void SpiBitOut(unsigned char **pSPIid**,
unsigned char **pBITsize**,
unsigned long **pDATA**);

unsigned long SpiBitIn(unsigned char **pSPIid**, unsigned char **pBITsize**);

Explication:

SpiBitOut

pSPIid	:	Port SPI™ à utiliser (Utilisez 0 à 7).
pBITsize		Nombre de bit(s) à envoyer (1 bit au minimum).
pDATA		Donnée à envoyer (max. 64 bits).

La fonction SpiBitOut() permet d'envoyer jusqu'à 64 bits vers le port SPI™. Elle pourra être utilisée pour communiquer avec certains composants bénéficiant d'une interface proche du SPI™ (comme le « Micro-Wire™ » par exemple). En utilisant 8 circuits intégrés externes 74HC595 connectés en série sur le module ROVIN™, il vous sera alors possible d'envoyer 8 octets dans une seule trame de sortie et étendant ainsi le nombre de sorties disponibles sur le ROVIN™.

Exemple:
SpiBitOut(2, 10, 0b1101110001); // Sortie de 10 bits de données sur le port SPI™ N° 2.

SpiBitIn

pSPIid	Port SPI™ à utiliser (Utilisez 0 à 7).
pBITsize	Nombre de bit(s) à recevoir (1 bit au minimum).
Valeur returnée	La valeur sera sur 64 bits max.

SpiBitIn() lit jusqu'à 64 bits max. depuis le port SPI™ désigné. 8 circuits intégrés externes 74HC597 ou 74HC165 pourront être connectés en série sur le module ROVIN™ pour lire jusqu'à 8 octets en une seule fois en étendant ainsi le nombre d'entrées disponibles sur le ROVIN™.

Exemple:

```
unsigned long spiDATA;
spiDATA = SpiBitIn(2, 64);      // Lecture de 64 bits de donnée sur SPI™ N° 2
                                // et les stocke dans une variable de type « long ».
```

SpiRw

Fonction:
unsigned long SpiRw(unsigned char **pSPIid**,
unsigned char **pBITsize**,
unsigned long **pDATA**);

Explication:

pSPIid	Port SPI™ à utiliser (Utilisez 0 à 7).
pBITsize	Nombre de bit(s) à envoyer (1 à 64 bits max.).
pDATA	Donnée à envoyer (max. 64 bits).
Valeur returnée	La valeur sera sur 64 bits max.

La fonction SpiRw() permet d'envoyer jusqu'à 64 bits vers le port SPI™ désigné tandis qu'elle lit simultanément et retourne la même quantité de bits qu'elle a envoyée. Cette dernière fonctionne comme si vous réalisez en même temps SpiBitOut() et SpiBitIn().

SpiCs

Fonction:
void SpiCs(unsigned char **pSPIid**, unsigned char **pLEVEL**);

Explication:

SpiCs

pSPIid	Port SPI™ à utiliser (Utilisez 0 à 7).
pLEVEL	Choisir NIVEAU CS. Utiliez 0 ou 1.

SpiCs() sort 0 ou 1 sur la broche CS (pCSpin) crée par la fonction SpiCreate().

Exemple:

```
SpiCs(2, 0);      // Place la broche CS à 0 sur le port SPI™ N° 2.
SpiCs(2, 1);      // Place la broche CS à 1 sur le port SPI™ N° 2.
```

SpiLatch

Fonction:
void SpiLatch(unsigned char **pSPIid**, unsigned char **pLATCHpulse**);

Explication:

pSPIid	Port SPI™ à utiliser (Utilisez 0 à 7).
PLATCHpulse	Utilisez SPI_HIGHPULSE ou SPI_LOWPULSE.

SpiLatch() permet d'étendre les possibilités d'interfaçage du ROVIN™. En utilisant des circuits intégrés externes 74HC595, 74HC597 ou 74HC165 cette fonction activera leur entrée « latch » par le biais d'une impulsion générée sur la broche « CS » (laquelle aura été créee par la fonction SpiCreate()). Le paramètre SPI_HIGHPULSE est à utiliser pour transférer les données lorsque « CS » est au niveau « haut ». A l'inverse le paramètre SPI_LOWPULSE est à utiliser pour trans-férer les données lorsque « CS » est au niveau « bas ».

Exemple:

```
SpiLatch(2, SPI_HIGHPULSE);     // Envoi une impulsion « haute » sur CS.
SpiLatch(2, SPI_LOWPULSE);      // Envoi une impulsion « basse » sur CS .
```

COMFILE
TECHNOLOGY

I2C™

I2C™-Inter Ic Control

L'I2C™ est un port de communication développé par la société Phillips™. Ce dernier utilise 2 fils pour établir des communications entre divers composants. Ce type de communication a été développé après le protocole SPI™. Il est très simple à utiliser et à exploiter. Le module ROVIN™ peut supporter jusqu'à 16 ports de communication I2C™ en même temps (uniquement en mode MASTER).

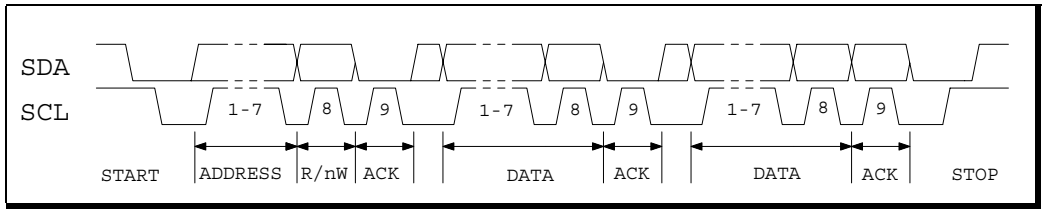
Chaque port I2C™ disposera de son propre jeu de registre de tel sorte que ces derniers soient totalement indépendants les uns des autres port. Les ports I2C™ exploiteront les ports XPPI PA, PB, PC, PD du module ROVIN™.

Broches I2C™

PA	Port XPPI-PA.
PB	Port XPPI-PB.
PC	Port XPPI-PC.
PD	Port XPPI-PD.

Constante	Explication:
I2C_MSBFIRST	Sortie des bits les plus significatifs en premier.
I2C_LSBFIRST	Sortie des bits les moins significatifs en premier.

Protocole I2C™



Description des broches I2C™:

Nom	Explication:
SDA	Ligne de données (E/S) du MAITRE vers/depuis circuit I2C™ ESCLAVE.
SCL	Ligne de sortie d'horloge du MAITRE vers circuit I2C™ ESCLAVE.

Le module ROVIN™ peut supporter jusqu'à 16 ports de communication I2C™ en même temps.

I2cCreate

Fonction:
void I2cCreate(unsigned char **pI2Cid**,
unsigned char **pUSEport**,
unsigned char **pSDApin**,
unsigned char **pSCLpin**,
unsigned char **pOUTmode**,
unsigned int **pSLOWtime**);

Explication:

I2cCreate

pI2Cid :	N° du port I2C™ à créer (Utilisez 0 à 15).
pUSEport	N° du port XPPI du ROVN™ à exploiter pour la création du port I2C™ (Utilisez PA, PB, PC ou PD).
pSDApin	Broche pUSEport à utiliser en tant que signal SDA.
pSCLpin	Broche pUSEport à utiliser en tant que signal SCL.
pOUTmode	Utilisez I2C_MSBFIRST ou I2C_LSBFIRST (voir tableau page précédente).
pSLOWtime	Vitesse de communication I2C™.

La fonction I2cCreate() est utilisée pour créer les ports I2C™ à partir des ports XPPI du module ROVIN™ (PA, PB, PC et PD). Pour utiliser cette fonction, il vous suffira simplement d'indiquer la correspondance des SDA, SCL vis à vis de celles du module ROVIN™ ainsi que le mode de sortie des données (MSB/LSB) et la vitesse de communication désirée (0 correspond à la vitesse max. En augmentant ce paramètre, la vitesse de communication décroîtra).

Tous les paramètres sont modifiables à tout moment dans votre programme (que ce soit l'attribution des broches SDA, SCL, le mode de sortie des données ou encore la vitesse de communication). Ceci vous offre une fléxibilité maximale.

Exemple:

```
I2cCreate(2, PA,          // Création port I2C™ N° 2 en utilisant XPPI-PA.
   5,                     // SDA sera la broche PA.5 (valeur initiale = 0).
   4,                     // SCL sera la broche  PA.4.
   SPI_MSBFIRST,          // On choisi la sortie MSB en premier
   0);                    // Vitesse de communication I2C™ maximale.
```

I2cOut, I2cIn

Fonction:
void I2cOut(unsigned char **pI2Cid**, unsigned char **pDATA**);
unsigned char I2cIn(unsigned char **pI2Cid**);

Explication:

I2cOut

pI2Cid	N° du port I2C™ (0 à 15).
pDATA	Donnée 8 bits à envoyer.

I2cOut() envoi un octet sur le port I2C™ désigné. Le port I2C™ doit être préalablement créé avec la fonction I2cCreate().

I2cIn

pI2Cid	N° du port I2C™ (0 à 15).
Return Value	Donnée 8 bits reçue.

I2cIn() lit et retourne un octet depuis le I2C™. Le port I2C™ doit être préalablement créé avec la fonction I2cCreate().

I2cBitOut, I2cBitIn

Fonction:
void I2cBitOut(unsigned char **pI2Cid**,
unsigned char **pBITsize**,
unsigned long **pDATA**);

unsigned long I2cBitIn(unsigned char **pI2Cid**, unsigned char **pBITsize**);

Explication:

I2cBitOut

pI2Cid	N° du port I2C™ (0 à 15).
pBITsize	Nombre bits à envoyer (1 à 64).
pDATA	Données à envoyer (64 bits max.).

I2cBitOut() permet d'envoyer de 1 à 64 bits sur le bus I2C™. Des données de type « long » (sur 8 octets soit 64 bits) pourront donc être envoyées en une seule commande.

A noter que si le paramètre pBITsize relatif au nombre de bit à envoyer est par exemple configuré à 13 mais que la données à envoyer pDATA disposent de 16 bits, alors le ROVIN™ n'enverra que 13 bits.

Exemple:

```
// Envoi 10 bits sur le port  I2C™ N° 2.
I2cBitOut(2, 10, 0b1101110001);
```

I2cBitIn

pI2Cid	N° du port I2C™ (0 à 15).
pBITsize	Nombre de bits à lire (1 à 64).
Returned Value	Donnée reçue.

I2cBitIn() peut être utilisé pour lire de 1 à 64 bits depuis le port I2C™ désigné. La fonction I2cCreate() devra être préalablement utilisée pour créer le port I2C™. Les données reçues pourront être stockées dans une variable de type « long » (sur 64 bits – soit 8 octets).

Exemple Use:

```
unsigned long i2cDATA;
i2cDATA = I2cBitIn(2, 64);          // Lecture de 64 bits de données
                                    // et stockage dans variable de type « long ».
```

I2cStart, I2cStop, I2cAck

Fonction:
void I2cStart(unsigned char **pI2Cid**);

void I2cStop(unsigned char **pI2Cid**);

unsigned char I2cAck(unsigned char **pI2Cid**, unsigned int **pTIMEOUT**);

Explication:

I2cStart

pI2Cid	N° du port I2C™ (0 à 15).

I2cStart() créé une condition « START » sur le port I2C™ désigné **pI2Cid**. Lors d'une communication I2C™, la condition START est à utiliser avant chaque début de « paquet ».

I2cStop

pI2Cid :	N° du port I2C™ (0 à 15).

I2cStop()créé une condition « STOP » sur le port I2C™ désigné **pI2Cid**. Lors d'une communication I2C™, la condition STOP est à utiliser à la fin de chaque « paquet ».

I2cAck

pI2Cid	N° du port I2C™ (0 à 15).
pTIMEout	TIME-OUT signal ACK
Return Value	TRUE(ACK Reçu), FALSE(ACK TIME-OUT)

I2cAck() surveille l'arrivé d'une condition ACK (Acquittement) envoyé par le composant « ESCLAVE » adressé sur le port I2C™ **pI2Cid**. Généralement, lors d'une communication I2C™, le MAITRE envoi des données à un module ESCLAVE et le module ESCLAVE envoi un acquittement par le biais d'une mise au niveau logique bas de la ligne SDA. A ce stade, le module MAITRE doit donner un coup d'horloge pour effectuer la lecture de ce signal d'acquittement. La fonction I2cAck permettra donc de récupérer cette information.

Exemple d'adresage d'une mémoire EEprom à commane série I2C™:

```
Consultez la documentation de la mémoire 24C32 pour plus d'informations.

// Configure SDA = PA.5 et SCL = PA.4.

void Delay(int pDELAYtime)
{ while(pDELAYtime--);
}

void main(void)
{
  int i;

       // Configure port PPI-PD en sortie.
  PPI_SetMode(PD, 0x00);

       // Création port I2C™ N° 0.  Utilise broches PC.5 et PA.4.
       // Sortie MSB en premier et vitesse I2C™ en premier.

  I2cCreate(0, PA, 5, 4, I2C_MSBFIRST, 0);

       // [Ecriture dans le mémoire !!!]{--------------------------------
       // Ecriture données 0 à 255 aux adresses appropriées.
       // Si pas d'acquittement de la part de la mémoire stoppe procédure.

  for(i=0; i<=0xff; i++)
  { I2cStart(0);                    // Conditon START pour l'I2C™ !
    I2cBitOut(0, 8, 0b10100000);    // Bit de contrôle pour 24LC32 !
    if(!I2cAck(0,0xff)) break;      // Test aquittement (TIME OUT ~ 32ms)

    I2cBitOut(0, 8, i>>8);          // ADDRESSE HAUTE !
    if(!I2cAck(0,0xff)) break;

    I2cBitOut(0, 8, i);             // ADDRESSE BASSE !
    if(!I2cAck(0,0xff)) break;

    I2cBitOut(0, 8, i);             // Envoie les données !
    if(!I2cAck(0,0xff)) break;

    I2cStop(0);                     // Condition STOP pour l'I2C™ !

    // Un délai de 5 ms mini. Doit être crée pour laisser le temps à
    // l'EEPROM de stocker les données. (pas nécessaire ne mode lecture).

    Delay(0x1a); // Délai de 6 ms (pour se laisser de la marge).
}
```

```
       // [Lecture dans le mémoire !!!]{--------------------------------
// Lecture depuis la mémoire et sortie des données sur le port PPI-PD.
// Si pas d'acquittement de la part de la mémoire stoppe procédure.

  for(i=0; i<=0xff; i++)
  { I2cStart(0);                              // Conditon START pour I2C™ !

    I2cBitOut(0, 8, 0b10100000);              // Bit de contrôle pour 24LC32
    if(!I2cAck(0,0xff)) break;                // Test aquittement

    I2cBitOut(0, 8, i>>8);                   // ADDRESSE HAUTE !
    if(!I2cAck(0,0xff)) break;

    I2cBitOut(0, 8, i);                       // ADDRESSE BASSE !
    if(!I2cAck(0,0xff)) break;

    // Commence cycle de lecture !

    I2cStart(0);                              // Conditon START pour I2C™ !

    I2cBitOut(0, 8, 0b10100001);              // Bit de contrôle pour 24LC32
    if(!I2cAck(0,0xff)) break;                // Test aquittement

    PPI_Out(PD, ~I2cBitIn(0, 8));    // Lit 1 octet et le sort sur PPI-PD!

    I2cAck(0,0);                              // Envoi un ACK !

    // Création coup d'horloge CLOCK avec le ACK !

    I2cStop(0);                               // Condition STOP pour l'I2C™

    Delay(0xff);                              // Delai pour voir les sorties
  }
  //}-------------------------------------------------------------------
}
```

CAPTURE

CaptureSet

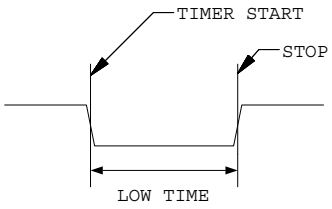
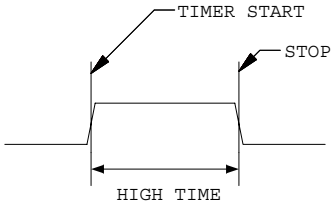
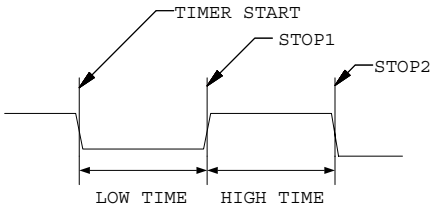
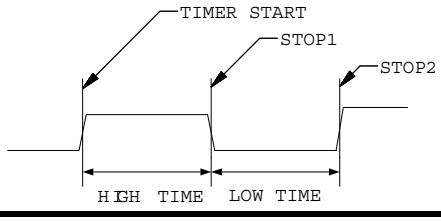
Fonction:
void CaptureSet(unsigned char **pID**, unsigned char **pSET**);

Explication:
CaptureSet

pID	Utilisez l'entrée de capture (0 ou 1).
pSET	Sélection du mode CAPTURE (voir table ci-dessous). Les 4 bits de poids fort sélectionnent le MODE et les4 bits de poids faible le facteur de division du TIMER.

CaptureSet() sélectionne le mode de fonctionnement pour CAPTURE0 ou CAPTURE1.

4 possibilités de configuration

	CAPTURE_LOW Mesure la durée lorsque le signal est au niveau « BAS ».
	CAPTURE_HIGH Mesure la durée lorsque le signal est au niveau « HAUT ».
	CAPTURE_LOWCYCLE Mesure la durée lorsque le signal est au niveau « BAS », puis mesure la durée lorsque le signal est au niveau « HAUT ».
	CAPTURE_HIGHCYCLE Mesure la durée lorsque le signal est au niveau « HAUT », puis mesure la durée lorsque le signal est au niveau « BAS ».

Exemple:

```
// Utilise CAPTURE0 et mesure niveau bas avec un facteur de division de  1024.
CaptureSet(0, CAPTURE_LOW | CAPTURE_DIV1024);
```

CaptureRead, CaptureOn, CaptureOff

Fonction:
void CaptureRead(unsigned char **pID**,
unsigned short ***pLOW**,
unsigned short ***pHIGH**);

void CaptureOn(unsigned char **pID**);
void CaptureOff(unsigned char **pID**);

Explication:

CaptureRead

pID : N° CAPTURE (0 ou 1).
pLOW : Mesure « BAS ».
pHIGH : Mesure « HAUT ».

CaptureRead() lit et retourne les mesures « BAS » et « HAUT ». Cette fonction doit être utilisée après la génération d'un EVENEMENT CAPTURE sinon des valeurs erronées seront retournées.

Exemple:

```
unsigned short LOW, HIGH;
CaptureRead(0, &LOW, &HIGH);
```

CaptureOn

pID : N° CAPTURE (0 ou 1).

Active CAPTURE0 ou CAPTURE1. La broche CAPTURE sera automatiquement configurée en entrée avec résistance de « PULL-UP ». Les EVENEMENT type CAPTURE seront également activés en même temps. La « capture » de largeur d'impulsion débutera automatiquement dès que la (les) condition(s) définie(s) par la fonction CaptureSet() seront remplie(s) et lorsque la (les) mesure(s) sera finie(s) un EVENEMENT CAPTURE sera généré.

CaptureOff

pID : N° CAPTURE (0 ou 1.)
Désactive CAPTURE0 ou CAPTURE1. Le mode de l'entrée CAPTURE reste configuré mais la (les) mesure(s) cesse(s) et les EVENEMENTS ne sont plus générés.

COMFILE
TECHNOLOGY

Interruptions

ExintSet

Fonction:
void ExintSet(unsigned char **pID**, unsigned char **pSET**);

Explication:

ExintSet

pID	:	N° broche interruption (0 à 7).
pSET	:	Mode de fonctionnement.

Le module ROVIN™ dispose de 8 entrées d'interruption (EXINT0 … EXINT7). Chaque entrée peut être configurée pour fonctionner selon plusieurs modes.

Si pSET = EXINT_CHANGE
l'entrée génrera une interruption (EVENEMENT) à chaque changement d'état.

Si pSET = EXINT_LOW
l'entrée génrera une interruption (EVENEMENT) tant qu'elle sera au niveau logique bas.

Si pSET = EXINT_HEDGE
l'entrée génrera une interruption (EVENEMENT) sur un front montant.

Si pSET = EXINT_LEDGE
l'entrée génrera une interruption (EVENEMENT) sur un front descendant.

Exemple:

```
ExintSet(6,EXINT_LOW);     // Interrution (EVENEMENT) sur niveau bas
ExintOn(6) ;               // Autorise Interruption sur EXINT6
EventOn() ;                // Autorise les Evenements
```

COMFILE
TECHNOLOGY

Méthode de Travail

Le chapitre qui suit expose brièvement les « grandes lignes » à suivre pour utiliser le « ROVIN-IDE » dans le cadre du développement d'une application. Consultez l'aide en ligne (et plus particulièrement les sélections « Démarrage rapide » qui vous donneront des exemples concrets détaillés).

Cas de figure 1 : **Développement d'une nouvelle application « mono tâche »** :

- Soit vous sélectionnez le menu (Fichier) -> (Nouveau Fichier) et vous commencez à écrire votre nouveau programme.

- Soit vous ouvrez un fichier source déjà écrit (*.c) et vous le sauvegardez sous un autre nom - avec (Fichier) -> (Sauver Sous) et sous un autre répertoire (à créer par vos soins). Il ne vous reste plus alors qu'à le modifier et à l'adapter à votre guise pour les besoins de votre nouvelle application.

Une fois le programme écrit et sauvegardé, cliquez sur le bouton « Linker » dans la boîte à outils du haut de l'écran.

Si vous n'avez fait aucune erreur de syntaxe, le « ROVIN-IDE » exécute tout son processus et vous indiquera que tout s'est correctement déroulé. En cas contraire, modifiez votre programme source.

Ouvrez alors la fenêtre « Téléchargement – Manageur de tâches » à l'aide de la petite flèche à droite du bouton « Télécharger programme » (ou par la sélection (Options) -> (Téléchargerment – Manageur de tâches).

Cliquez alors sur le bouton « Ajouter Tâche », puis sélectionnez le nom du fichier que vous venez de compiler (ce dernier doit être doté d'une extension '.pro'). Le nom du fichier s'affiche alors sous le menu de la fenêtre « Téléchargement – Manageur de tâches ». Cochez la case à gauche du nom du programme pour sélectionner ce dernier (afin qu'il puisse faire partie de la liste des programmes à télécharger dans le ROVIN™).

Configurer la répartition mémoire à attribuer pour les programmes au sein du ROVIN™. Ici pas de problème, du fait que vous n'avez qu'un seul programme, saisissez 100% dans la case de droite. Vous pouvez également définir pour chaque tâche (ici vous n'en avez qu'une) la répartition mémoire pour gérer les opérations standards et celles nécessitant la gestion d'interruptions, en déplaçant la ligne noire de séparation des cadres vert et rouge.

Il vous faut maintenant enregistrer l'ensemble de ces informations dans un fichier de type PROJET. Un PROJET renferme en fait le ou les noms des fichiers sources (*.c) contenus dans le PROJET (et qui apparaittrons dans la fenêtre de l'éditeur lorsque vous rechargerez le PROJET), mais également la liste des programmes compilés (*.pro) sélectionnés pour être transférés dans le module ROVIN™ ainsi que les différentes répartitions mémoires que vous leurs aurez attribués).

Cliquez alors sur (Sauver), une fenêtre s'ouvre dans laquelle vous pourrez saisir le nom de votre PROJET (vous pouvez éventuellement saisir le même nom que celui de votre fichier source). Saisissez le nom du fichier **sans aucune extension**, le « ROVIN-IDE » s'occupera d'ajouter tout seul une extension '*.dwn'. Si vous saisissez le même nom que votre fichier source (par exemple **essai**), vous pourrez vous rendre compte que le « ROVIN-IDE » aura au final créé 4 fichiers :

essai.c -> Votre fichier source
essai.pro -> Le fichier compilé (celui qui sera téléchargé dans le module ROVIN™)
essai.dbg -> Fichier utilisé par le « ROVI N-IDE » lors du mode DEBUG
essai.dwn -> Fichier PROJET renfermant la plupart des informations sur les autres fichiers

Fermez alors la fenêtre « Téléchargement – Manageur de tâches » à l'aide de la petite croix en haut à droite de celle-ci.

Vérifiez que le câble USB du ROVIN™ est bien relié d'une part au PC et d'autre par au module ROVIN™. Vérifiez également que le module ROVIN™ est correctement alimenté.

Vérifiez que pour une quelconque raison la communication du câble n'ait pas été coupée (la touche « Stopper Communication » ne doit pas être enfoncée) si tel est le cas, cliquez dessus pour la relâcher.

Cliquez alors sur le bouton « Télécharger Programme(s) » (pas sur la petite flèche de droite, mais sur la flèche jaune de gauche). A ce stade, si tout se passe normalement le « ROVIN-IDE » après un petit traitement vous demandera de cliquer sur le bouton « Reset » de sa barre d'outils pour que le ROVIN™ puisse démarrer.

En cliquant sur le bouton en question, le ROVIN™ doit réaliser votre programme ! ! !

Si une des étapes ci-dessus ne se déroule pas correctement, recommencez les opérations depuis le début… c'est que vous avez ratez une étape.. Au besoin consultez la rubrique Problèmes et Solutions en fin de manuel.

Cas de figure 2 : **Debuguer la tâche** :

- Soit la tâche est en cours d'exécution car vous avez suivi le processus doné en exemple ci-avant, auquel cas, il suffit simplement se cliquer sur l'onglet DEBUG dans la boîte à outils du haut de l'écran.

Après un petit traitement le « ROVIN-IDE » changera la nature des boutons accessible dans la boîte à outils du haut de l'écran. Il vous suffit alors d'utiliser les nouveaux boutons pour exploiter les différentes possibilités du mode DEBUG (comme décrit dans le manuel ci avant). Pour revenir à l'adition de votre programme, il vous suffit simplement de cliquer sur l'onglet « PROGRAMME »

- Vous pouvez également entrer en mode DEBUG en rechargeant simplement un fichier PROJET, puis en se cliquant sur l'onglet DEBUG dans la boîte à outils.

Cas de figure 3 : **Développement d'une nouvelle application « multitâches »**

Le développement d'applications en mode multitâches fonctionne exactement de la même façon qu'en mode « simple tâche ». Il vous suffit d'ouvrir les différents fichier « .c », de les éditer et de les sauvegarder TOUS dans un même répertoire que vous aurez préalablement créé. Sélectionnez ensuite les programmes un par un par et pour chacun d'eux, cliquez sur la touche « Linker » . Lorsque tous les programmes ont été correctement compilés sans erreur, activez la fenêtre « Téléchargement – Manageur de Tâches ». A l'aide du bouton « Ajouter Tâche », sélectionnez tous les noms de fichier (.pro) devant être téléchargé au sein du ROVIN™ (10 max.). Cochez le nom des différents programmes sélectionnés pour être réellement transférés dans le ROVIN™. Equilibrez la répartition mémoire en % entre les différentes tâches et l'équilibrage mémoire propre à chacune des tâches (jauge verte/rouge). Sauvegardez l'ensemble des informations dans un fichier PROJET. Fermez la fenêtre « Téléchargement – Manageur de Tâches » et cliquez sur le bouton « Télécharger Programme(s) », puis surle bouton « Reset ».

Cas de figure 4 : **Debuguer une des tâches**

Sous la fenêtre de l'éditeur, cliquez sur l'onglet du programme (tâche) vous intéresse, puis cliquez sur l'onglet « DEBUG » de la boîte à outils du haut de l'écran. Débuguez votre tâche comme indiqué précédemment (seule cette tâche sera concernée), les autres continueront à s'exécuter normalement.

Dans le cadre d'un développement en mode multitâches, les erreurs de manipulation les plus courrantes consitent à récupérer les fichiers sources '.c' directement dans les répertoire de projet déjà existant et de ne pas les resauvegarder dans le répertoire courant du PROJET actuel. Dès lors, une erreur interviendra dans la fenêtre « Téléchargement – Manageur de Tâches ». Rappelez-vous que tous les fichiers ('.c' – ''.pro' – '.dwn ' – 'dbg') d'un même projet doivent être TOUS dansle même répertoire). Cette erreur peut causer plusieurs effets.

- Soit désactiver la communication USB.

Vérifiez que la touche « Stopper Communication » ne soit pas enfoncée (si tel est le cas, cliquez dessus pour la relâcher). Sauvegardez à nouveau les fichiers dans le même répertoire et recompilez l'ensemble pour recréer tous les fichiers.

- Soit cette erreur aura pour effet de corrompre le fichier PROJET que vous avez voulu créer et ce dernier ne comportera aucune des informations qu'il est censé contenir. Dès lors, suite à l'erreur vous vous apercevrez en le rechargeant que ce dernier ne fait plus afficher les fichiers sources à l'écran ! Pas de panique, il suffit sous Windows™ d'effacer ce dernier (.dwn) et éventuellement les fichiers '.pro' et '.dbg' associés (SURTOUT pas les sources '.c') et de remettre les sources dans un seul et même répertoire afin de recréer le projet de façon cohérente.

Attention ceci peut aussi arriver lorsque vous développez en mode simple tâche. Il est en effet courant d'ouvrir d'autres fichiers sources pour récupérer telle ou telle partie de code afin de l'intégrer au fichier en cours d'édition. Toutefois le « ROVIN-IDE » conservera en mémoire l'adresse du dossier relatif au dernier fichier ouvert ! Aussi lors de la création du PROJET dans la fenêtre « Téléchargement – Manageur de Tâches » pensez à vérifiez que tous les fichiers '.c', 'dwn ', '.pro' et '.dbg' relatifs à votre source soient bien dans le même répertoire !

Notes relatives aux développements en mode multitâches.

Il est important de garder à l'esprit que chaque tâche développée sera vraiment exécutée de façon totalement indépendante vis à vis des autres tâches. Il est donc impératif de surveiller comment vos différentes tâches vont être amenées à exploiter les différents ports d'entrées-sorties du module ROVIN™ afin d'éviter tout conflit pouvant intervenir par exemple si une tâche exploite une broche du ROVIN™ en sortie et si dans le même temps, une autre tâche reconfigure cette broche en entrée… ou si 2 (ou plus) tâches essaient d'avoir accès à la même broche en même temps (pour envoyer des informations sur le port série par exemple).

La solution la plus simple consiste dans un premier temps à définir l'état des ports du ROVIN™ dans une seule des tâches et (du fait que toutes les tâches démarrent en même temps), à retarder l'exécution des autres tâches de quelques milli-secondes à l'aide d'instructions de « temporisation » au début de celles-ci») afin d'être sur que la configuration des ports ait eu temps de s'effectuer.

Si plusieurs tâches doivent vraiment devoir utiliser des ports et des broches communs (comme l'UART par exemple), il vous suffira alors par exemple de sauvegarder simplement la valeur « TRUE » à l'adresse 00000000h de la mémoire RAM XHEAP (partageable entre toutes les tâches) lorsqu'une tâche utilise l'UART et les autres taches pourront être facilement programmées pour attendre que cette dernière « passe » à « FALSE » pour pouvoir à leur tours utiliser l'UART.

Gardez également à l'esprit que tout puissant soit-il, le module ROVIN™ pourra être ralenti si plusieurs tâches nécessitant de nombreux cycles « machines » sont exécutés en même temps. Dès lors il vous faudra peut être modifier certains paramètre de temporisation de certaines tâches afin d'optimiser leur fonctionnement en mode multitâches.

COMFILE
TECHNOLOGY

Précautions d’usages

Tous les modules ROVIN™ sont testés avec soins avant leur expédition afin de vous procurer un fonctionnement immédiat lors de vos utilisations. Correctement utilisés, les modules ROVIN™ vous permettrons de réaliser d'innombrables applications dont vous ne pouviez même pas envisager l'existence. Toutefois il vous faut impérativement garder à l'esprit que le ROVIN™ n'est rien d'autre qu'un microcontrôleur et au même titre qu'avec tout autre microcontrôleur il vous faut respecter certaines règles de bases afin d'éviter qu'il ne rende l'âme ! **Aussi vérifiez de ne jamais :**

1) Alimentez ce dernier sous une tension supérieur à +5 Vcc.

2) Si vous appliquez des tensions issues de capteurs ou dispositif extérieur:

   - Vérifiez toujours que ces tensions soient inférieures à + 5V.
   - Coupez en PRIORITÉ l'alimentation des capteurs externes AVANT de couper celle du ROVIN™ afin d'éviter qu'une tension soit toujours présente sur les entrées du ROVIN™ alors que ce dernier n'est plus alimenté.
   - Pour rejoindre le point ci-avant vérifiez que vous ne disposez pas de gros condensateurs sur les entrées des ROVIN™ pouvant stocker une tension qui viendra alors se décharger dans le ROVIN™ lorsque vous couperez les alimentations.

3) Ne jamais inverser la polarité d'alimentation du ROVIN™.

4) Lorsque vous utilisez les ports du ROVIN™ en entrées, n'utilisez jamais de grand fils pour y raccorder des boutons-poussoir et autres capteurs sans avoir recours à un circuit de mise en forme et de protection (circuit RC avec zener de protection ou optocoupleur). Si pour vos tests, vous n'utilisez pas de protection de ce type, limitez la longueur de vos fils à 3 - 4 cm afin d'éviter les phénomènes de "latch-up" ou de destruction par électricité statique.

5) Utilisez impérativement des diodes de protection lorsque vous pilotez des charges inductives (moteurs par exemple) et éviter de placer le câble de téléchargement à côté de cette source.

6) Découplez rigoureusement l'alimentation du ROVIN™.

7) Avant d'appliquer une quelconque tension (+ 5V ou masse) sur une des broches du ROVIN™ , vérifiez IMPERATIVEMENT que cette broche ai bien été configurée en ENTREE.

8) N'appliquez pas de tension ou ne mettez pas des broches du ROVIN™ à la masse si elles ont été configurées en sortie – Evitez les conflits de gestion des ports ES entre les différentes tâches.

9) Passez toujours par un montage à transistor ou buffeur ou optocoupleur pour alimenter un dispositif consommant plus d'une dizaine de milli-ampère.

10) Ne réalisez JAMAIS de boucle "sans fin" sur une instruction écrivant en mémoire EEprom (Le nombre de cycles d'écriture en EEPROM étant limité).

11) Ne coupez jamais l'alimentation du ROVIN™ (de même ne déconnectez JAMAIS le câble USB) lorsque ce cernier est en phase de programmation ou de DEBUG !

12) Si certaines broches du ROVIN™ ne sont pas utilisées pour les besoins de votre application, configurez tout de même impérativement ces dernières en SORTIE et placez celles-ci au niveau logique « 0 ». Remettez à jour l'état de toutes les broches des ROVIN™ régulièrement (même celles non utilisées) au sein de la « boucle » principale de votre programme (ne vous contentez pas d'une simple configuration au début du programme).

13) Comme TOUT microcontrôleur, les ROVIN™ sont sensibles à l'électricité statique. Ces derniers devront donc être manipulés (et soudés) avec les précautions qui s'imposent afin d'éviter leurs destruction ou leur fragilisation.

En cas de non respect des limites et des conditions d'utilisations indiquées dans ce manuel, la fiabilité et la durée de vie des " ROVIN™" sera remise en cause (et l'échange du "ROVIN™" ne pourra pas être pris en charge au titre de la garantie).

Problèmes et solutions

P : Je n'arrive pas (ou plus) à « dialoguer » avec le module ROVIN ?

S : Vérifiez que le voyant rouge USB en haut de l'écran soit bien allumé.

Si ce n'est pas le cas, vérifiez que le câble USB du ROVIN™ soit bien relié au port USB du PC (utilisez de préférence la prise arrière du PC).

Essayez dans un premier temps de cliquer sur le bouton de 'reconnexion' du câble en haut de l'écran de l'environnement « ROVIN-IDE ».

Si rien n'y fait, déconnectez le câble USB du PC… attendez quelques secondes et reconnectez le à nouveau. Si le voyant rouge USB ne s'allume toujours pas, sortez du programme « ROVIN-IDE », puis sous le bureau de Windows™, déconnectez à nouveau le câble USB du PC… attendez quelques secondes et reconnectez le. Exécutez alors le programme « ROVIN-IDE » : le voyant doit alors s'allumer.

Si le voyant ne s'allume toujours pas et qu'il s'agit de la première fois que vous utilisez le « ROVIN-IDE », il y a de forte chance que vous n'ayez pas installé le bon driver USB (désinstallez le driver et installez impérativement celui présent dans le répertoire du ROVIN).

S : Vérifiez que le module ROVIN™ soit correctement alimenté.

S : Vérifiez que le connecteur au bout du câble USB soit bien relié au module ROVIN™.

S : Si la connexion a été involontairement perturbée, suite au fait que vous ayez par exemple « coupé » l'alimentation du ROVIN™ et que vous ayez oublié d'alimenter à nouveau ce dernier avant de procéder à sa programmation, ou si une erreur de téléchargement est survenue dans la fenêtre « Téléchargment & Manageur de tâches » (mauvaise sélection de fichiers par exemple), en absence de réponse de la part du ROVIN™, l'interface de développement « ROVIN-IDE » coupera automatiquement la communication USB (ceci est matérialisé par l'enfoncement du bouton avec la 'croix jaune'). Il vous faut alors impérativement cliquer à nouveau sur ce même bouton (pour qu'il ne soit plus enfoncé et être à nouveau à même de pouvoir envoyer des commandes vers le module ROVIN™ via son câble).

P : Je n'arrive pas à ajouter une variable dans la fenêtre « Watch » ?

S : L'ajout ne peut se faire qu'après une compilation.
S : Il faut impérativement surligner correctement TOUTE la variable pour l'ajouter à la Fenêtre « watch » (par exemple pour la variable iTIME, il ne faut pas juste surligner TIME, mais bien iTIME).

P : Lorsque je suis en mode DEBUG et que j'utilise les boutons « Pas-à-pas » et « Pas-à-pas animés », rien ne se passe et le cadre qui entour les instructions de mon programme ne passe plus d'une instruction à l'autre ?

S : Ceci peut être le cas si vous passez sur une instruction nécessitant normalement un nombre important de « cycles machines » avant d'être exécutée : comme par exemple l'instruction while(pTIME--) ; Pour vous en assurer ajoutez la variable pTIME dans la fenêtre Watch et vous verrez effectivement que l'instruction while(pTIME--) est bien en train de s'exécuter à chaque sollicitation des boutons « pas-à-pas » ou « pas-à-pas automatique » (la variable pTIME sera bien decomptée) , même si rien ne semble se passer dans l'évolution de la fenêtre de DEBUG.

P : La fenêtre « Mémoire » n'affiche rien malgré le fait que la plage mémoire ait été indiquée ?
S : La fenêtre « Mémoire » n'est opérationnelle qu'en mode « **DEBUG** ».

P : Je n'arrive pas à ouvrir la fenêtre de sélection de fontes ?
S : L'ajout ne peut se faire qu'après avoir ouvert un fichier source.

P : L'environnement « ROVIN-IDE » semble être bloqué en mode DEBUG et je ne peux plus réaliser aucune action ?
S : Vérifiez que le bouton « Pas-à-Pas Automatique » n'est pas resté enfoncé.

P : Le fichier PROJET que je viens de créer est vide et ne comprend plus rien ?
S : Consultez la rubrique « Méthode de travail » où ce cas de figure est exposé ».

P : Mon module ROVIN fait n'importe quoi au niveau de ses ports d'E/S ?
S : Plusieurs de vos tâches doivent utiliser les mêmes ports d'E/S sans concertation – EVITEZ les conflits lors de la gestion de ports communs.

COMFILE
TECHNOLOGY

Fin