fonction. Afin d'être en conformité avec les anciennes appellations en vigueur dans le monde des PIC, la fonction get_rtcc est équivalente à la fonction get_timer0.

– **GETC(), GETCH(), GETCHAR(), FGETC()**
Ces fonctions sont classiques en langage C mais nécessitent une petite précision dans le cas des compilateurs C pour PIC. Les trois premières formes attendent un caractère depuis la liaison série RS 232 du PIC ne comportant pas d'identifiant de flux, telle qu'elle a été définie par la directive #use RS232 étudiée précédemment. La forme fgetc quant à elle attend un caractère mais depuis le flux qui y est spécifié. Ce flux fait alors référence à une liaison série RS 232 du PIC telle que définie au moyen de la directive #use RS232 étudiée précédemment mais specifiant cette fois-ci le même flux. La syntaxe est la suivante :
valeur = getc() ou getch() ou getcharO dans le premier cas et valeur = fgetc(flux) dans le deuxième cas ; où valeur est une variable contenant le caractère reçu et où flux est une constante spécifiant le flux à utiliser.

– **GOTO_ADRESS()**
Cette fonction force le programme à poursuivre son exécution à l'adresse spécifiée. Elle s'utilise de la façon suivante :
goto_adress(adresse)
où adresse est une adresse valide en mémoire de programme codée sur 16 ou 32 bits selon le type de PIC utilisé. Cette fonction n'a en principe pas a être utilisée dans une application normale bit construite.

– **I2C_POLL()**
Cette fonction ne doit être utilisée que lorsque l'I2C est validé en mode matériel c'est-à-dire en utilisant la ressource SSP (port série synchrone) interne. Elle retourne une valeur vraie (1) si un octet a été reçu sur le bus I2C et une valeur fausse (0) aucun octet n'a été reçu. Sa syntaxe est la suivante :
i2c_poll()

– **I2C_READ()**
Cette fonction est utilisable sans restriction en mode I2C et lit un octet reçu sur bus I2C. En mode maître, elle assure la génération de l'horloge sur la ligne SC depuis le PIC alors qu'en mode esclave elle assure l'attente de l'horloge en provenance du maître du bus. Cette fonction n'intègre aucune vérification de délai d'attente maximum. Il est donc souhaitable d'utiliser i2C_poll si un tel risque est inacceptable. Elle s'utilise de la façon suivante :
donnée = i2c_read()
ou donnée = i2c_read(ack)
où donnée contient la donnée lue sur le bus I2C et où le paramètre optionnel a peut être égal à 1 ou à 0. Dans le premier cas, un acquittement est généré sur le h I2C ce qui est également le cas par défaut ; dans le deuxième cas aucun acquittement n'est généré.

– **I2C_START()**
Cette fonction ne doit être utilisée que lorsque l'I2C est valide en mode maître Elle permet de générer une condition de départ dans la terminologie I2C. Si elle appelée une nouvelle fois avant que la fonction I2C_STOP ait été utilisée, une condition de départ spéciale est alors générée. Sa syntaxe est la suivante :
i2c_start()

– **I2C_STOP()**
Cette fonction ne doit être utilisée que lorsque I'I2C est valide en mode maître Elle permet de génerer une condition d'arrêt dans la terminologie I2C. Sa syntaxe la suivante :
i2c_stop()

– **I2C_WRITE()**
Cette fonction est utilisable sans restriction en mode I2C et écrit un octet sur le bus I2C. En mode maître, elle assure la génération de l'horloge sur la ligne SCL depuis le PIC alors qu'en mode esclave elle assure l'attente de l'horloge en provenance du maître du bus. Cette fonction n'intègre aucune vérification de délai datte maximum. Il est donc souhaitable d'utiliser i2C_poll si un tel risque est inacceptable. Cette fonction fournit également en retour la valeur du bit d'acquittement qui est égale à 0 s'il y eut acquittement du destinataire et à 1 dans le cas contraire. Elle s'utilise de la façon suivante :
i2c_write (donnée)
où donnée contient la donnée (au sens large du terme) à écrire sur le bus I2C.

– **INPUT()**
Cette fonction indique l'état logique de la ligne de port spécifiée dans une variable de type bit qu'elle rend égale à 0 ou à 1. Elle respecte le mode de traitement des ports parallèles imposé au préalable par la directive #use xxx_io et peut donc être amenée à forcer la ligne correspondante en entrée avant de lire son état (voir ci-dessus la description des directives #use xxx_io si nécessaire). Elle s'utilise de la façon suivante :
valeur = input(ligne de port)
où ligne de port correspond à la définition d'une ligne de port d'entrée/sortie telle qu'elle est faite dans le fichier .h du PIC utilisé. Ainsi par exemple :
valeur = input(PIN_A1)
lit le niveau logique présent sur la ligne d'entrée/sortie 1 du port A.

– **INPUT_STATE()**
Cette fonction indique l'état logique de la ligne de port spécifiée dans une variable de type bit qu'elle rend égale à 0 ou à 1. Contrairement à input, elle ne respecte pas le mode de traitement des ports parallèles imposé au préalable par la directive #use xxx_io et ne modifie donc pas le sens de travail défini préalablement pour cette ligne. Elle s'utilise de la façon suivante : valeur = input_state(ligne de port) où ligne de port correspond à la définition d'une ligne de port d'entrée/sortie telle qu'elle est faite dans le fichier .h du PIC utilisé. Ainsi par exemple : valeur = input_state(PIN_A1) lit le niveau logique présent sur la ligne d'entrée/sortie 1 du port A sans en modifier le sens

de fonctionnement.

- **INPUT_x()**
  Cette fonction lit d'un seul coup l'état des huit lignes du port spécifié. Elle respecte le mode de traitement des ports parallèles imposé au préalable par la directive #use xxx_io et peut donc être amenée à forcer le port correspondant en entrée avant de lire son état (voir ci-dessus la description des directives #use xxx_io si nécessaire). Elle s'utilise de la façon suivante : valeur = input_x() où x est la lettre de désignation du port choisi, comprise entre A et G selon le PIC utilisé. Ainsi, par exemple : valeur = input_b Lit les données présentes sur les 8 lignes du port B (BO à B7).

- **OUTPUT_x()**
  Cette fonction écrit un octet sur les huit lignes du port spécifié. Elle respecte le mode de traitement des ports parallèles imposé au préalable par la directive #use xxx_io et peut donc être amenée à forcer le port correspondant en sortie avant d'y écrire l'octet désiré (voir ci-dessus la description des directives #use xxx_io si nécessaire). Elle s'utilise de la façon suivante : output_x(donnée) où donnée est un entier sur 8 bits qui est écrit sur le port désigné par x ; x étant compris entre A et G selon le PIC utilisé. Ainsi par exemple : output_b(0x0F) a pour effet de mettre au niveau logique haut les quatre bits de poids faibles du port B (B0,B1,B2etB3).

- **OUTPUT_BIT()**
  Cette fonction met au niveau logique haut ou bas la ligne de port spécifiée. Elle respecte le mode de traitement des ports parallèles imposé au préalable par la directive #use xxx_io et peut donc être amenée à forcer la ligne correspondante en sortie avant d'y écrire la valeur désirée (voir ci-dessus la description des directive #use xxx_io si nécessaire). Elle s'utilise de la façon suivante : output_bit(ligne de port, valeur) où ligne de port correspond à la définition d'une ligne de port d'entrée/sortie telle qu'elle est faite dans le fichier .h du PIC utilisé et où valeur est égale à 0 ou à 1 Ainsi par exemple : output_bit(PIN_A1,0) met au niveau logique bas la ligne d'entrée/sortie 1 du port A.

- **OUTPUT_FLOAT()**
  Cette fonction met en entrée la ligne de port spécifiée. Cela lui permet de flotter ou d'être dans l'état haute impédance. Elle s'utilise de la façon suivante : output_float(ligne de port)

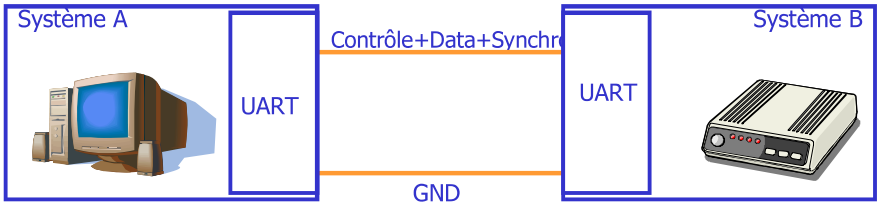
.3 Liaison série asynchrone

.3.1 Objectifs

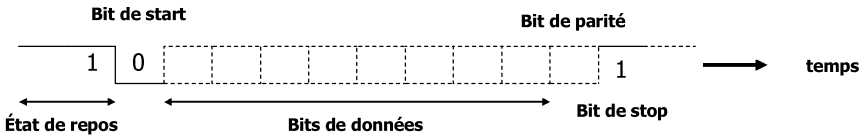
- Augmenter la distance de communication (Km) ;
- Connecter des ordinateurs entre eux, à des périphériques (clavier, souris,...), ou à des machines ;
- Facile à mettre en oeuvre ;

– Traitement logiciel simple.

.3.2 Principe

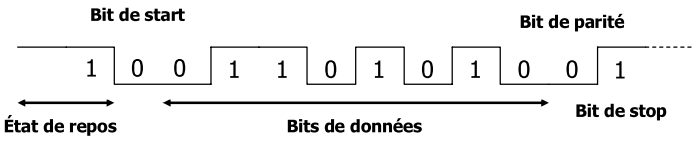


Format des données : caractère

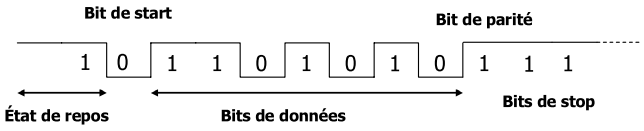


Contenu du caractère:
1 bit de start
7 ou 8 bits de données
1 bit de parité optionnel
1 ou 2 bits de stop

Exemples
0x6A, 8bits de données, parité paire, 1 bit de stop

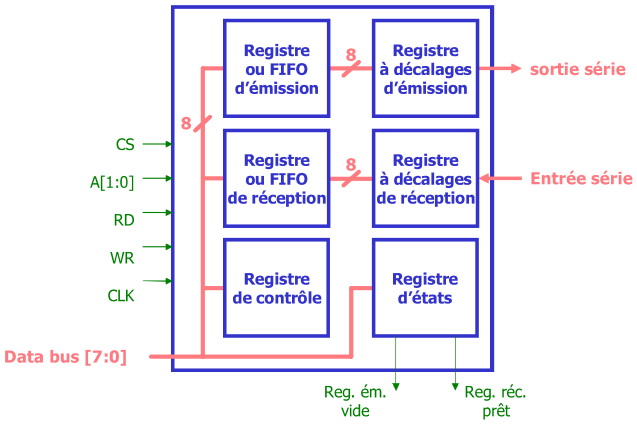


0x6A, 7bits de données, parité impaire, 2 bits de stop
contrôle de flux :
L'émission des bits peut se faire à une cadence plus rapide que celle de leur traitement, alors une mémoire de type FIFO (First In First Out) stocke les caractères à leur arrivée dans le récepteur, en attendant leur traitement. Dans certains systèmes, la FIFO peut se remplir complètement et déborder d'où la nécessité de prévenir l'émetteur afin qu'il suspende l'envoi de nouvelles données .
Il existe deux solution techniques pour contrôler le flux :
– Contrôle matériel (RTS/CTS) ;
– Controle logiciel (Xon/Xoff).



.3.3 UART

UART : Universal Asynchronous Receiver Transmitter



.3.4 les points forts et faibles

- Point forts : Nombre de fils réduit, bonne immunité au bruit sur de très longues distances (paire différentielle), matériel très simple, logiciel réduit (read ou write sous interruption) et protocoles de complexité croissante.
- Points faibles, débit très faible (200 Kbits/s) et faible rendement (65

.4 afficheurs à cristaux liquides

.4.1 Généralités

.4.1.1 Description

Les afficheurs à cristaux liquides sont des modules compacts intelligents et nécessitent peu de composants externes pour un bon fonctionnement. Ils sont relativement bons marchés et s'utilisent avec beaucoup de facilité. Un exceptionnel microprocesseur "pilote" de la famille C-MOS diminue considérablement leur consommation (inférieur à 0.1 mW). Ils

sont pratiquement les seuls a être utilisés sur les appareils à alimentation par piles.
Plusieurs afficheurs sont disponibles sur le marche et ne diffèrent les uns des autres, non seulement par leurs dimensions, (de 1 à 4 lignes de 6 à 80 caractères), mais aussi par leurs caractéristiques techniques et leurs tension de service. Certains sont dotés d'un rétro eclairage de l'affichage. Cette fonction fait appel à des LED montées derrière l'écran du module, cependant, cet éclairage est gourmand en intensité (250 mA max.).

.4.1.2 Principe de fonctionnement

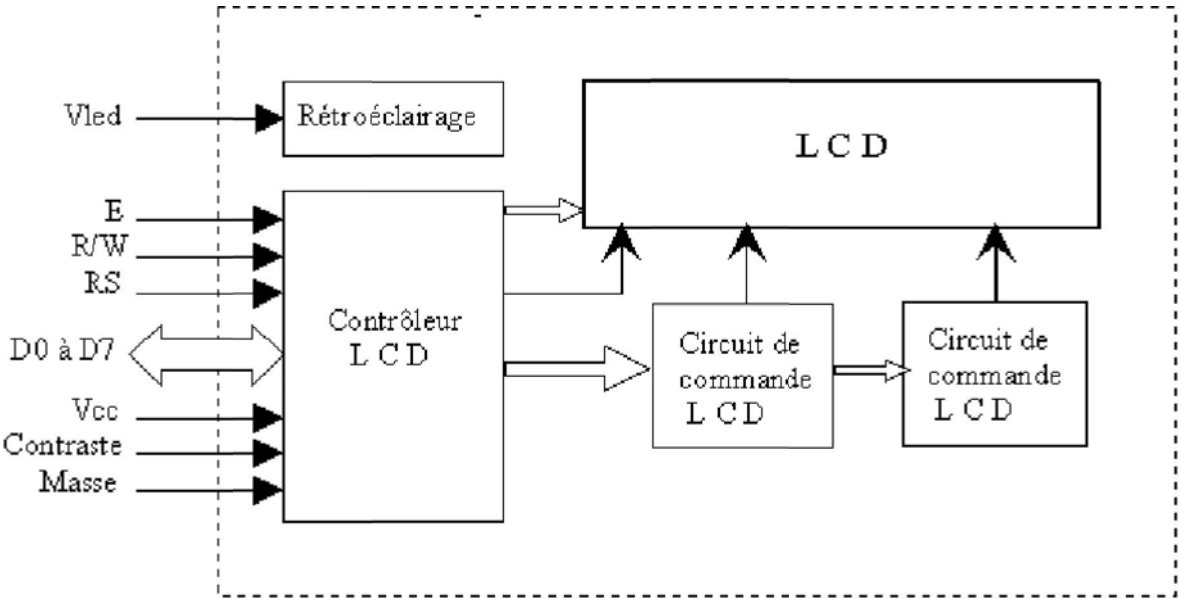


FIG. 4 – Schéma fonctionnel

Comme le montre le schema fonctionnel, l'affichage comporte d'autres composants que l'afficheur à cristaux liquides (LCD) seul. Un circuit intégré de commande spécialisé, le LCD-controller, est charge de la gestion du module. Le "contrôleur" remplit une double fonction : d'une part il commande l'affichage et de l'autre se charge de la communication avec l'extérieur.

.4.1.3 Connexions

Les connexions à realiser sont simples puisque l'afficheur LCD dispose de peu de broches. Il faut, évidement, l'alimenter, le connecter à un bus de donnée (4 ou 8 bits) d'un microprocesseur, et connecter les broches Enable (validation), Read/Write (écriture/lecture) et Register Select (instruction/commande).

.4.2 Principe des cristaux liquides

L'afficheur est constitué de deux lames de verre, distantes de 20 micrométres environ, sur lesquelles sont dessinées les mantisses formant les caractères. L'espace entre elles est rempli de cristal liquide normalement réfléchissant (pour les modèles réflexifs). L'application entre les deux faces d'une tension alternative basse fréquence de quelques volts (3 à 5 V) le rend absorbant.
Les caractères apparaissent sombres sur fond clair. N'émettant pas de lumiere, un afficheur à cristaux liquides reflexif ne peut être utilisé qu'avec un bon éclairage ambiant. Sa lisibilité augmente avec l'éclairage. Les modeles transmissifs fonctionnent différemment : normalement opaque au repos, le cristal liquide devient transparent lorsqu'il est excité ; pour rendre un tel afficheur lisible, il est nécessaire de l'éclaire par l'arrière.

.5 Le Timer TMR0

Le timer (l'horloge) intégré dans microcontroleur (dénommé TMR0), est constitué d'un prescaler (prediviseur) de 8 bits et du timer lui-même, caractérisé également par 8 bits. Le timer peut fonctionner en deux modes distincts, qui sont déterminés par la valeur du bit D5 dans le registre OPTION, bit que l'on appelle T0CS.

- Mode TIMER : On le sélectionne en mettant à 0 le bit T0CS. Dans ce type de fonctionnement, le timer est alimenté par son horloge interne dont la fréquence est égale à celle de l'horloge du microcontrôleur divisée par 4.
- Mode COUNTER : On le sélectionne en mettant à 1 le bit T0CS. Dans ce mode, le timer incrémente son propre comptage à chaque front (de montée ou de descente) présent sur la patte RA4. Pour déterminer si cette incrémentation doit se produire sur le front positif ou sur le négatif, il faut agir sur le bit D4, toujours dans le registre OPTION, bit que l'on appelle également T0SE. Si ce bit est mis à 0, l'incrementation est effectuée sur le front positif, et vice versa, s'il est à 1, l'incrémentation est opérée sur le front negatif. Le timer peut être lu et modifié a tout moment. Il est en effet situé à l'adresse 1 parmi les registres d'utilisation spéciale.

.5.1 Le prescaler

Le prescaler est un dispositif qui sert à diviser la fréquence qui va piloter le véritable compteur et permet donc d'obtenir des intervalles de temps relativement longs. Le prescaler peut être connecté, aussi bien au timer TM0, qu'au watchdog (circuit de surveillance). Pour relier le prescaler à l'un ou a l'autre, il vous suffira d'agir sur le bit D3 du registre OPTION, que l'on nomme PSA. Si ce bit est mis à 0, le prescaler est relié au TMR0, s'il est mis à 1, il sera relié au watchdog. Le rapport de division du prescaler est déterminé à travers trois bits, nommés PS0, PS1 et PS2 du registre OPTION, suivant la figrure . Rappelez-vous qu'il n'est pas possible de lire ni d'écrire la valeur du prescaler (c'est-à-dire du comptage qu'il est en train d'effectuer) et que ce registre est remis à 0 à chaque fois que vous effectuez une opération d'écriture dans le registre TMR0.

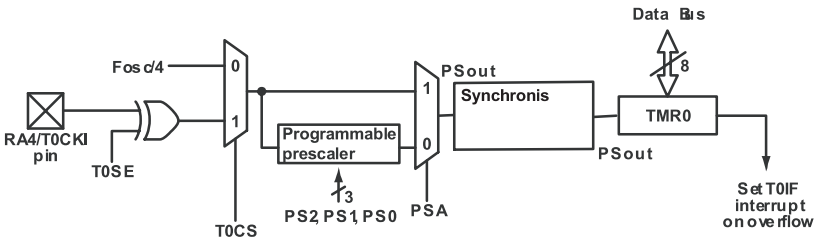


FIG. 5 – Schéma synoptique du timer TMR0

PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

FIG. 6 – Détermination du rapport de division du prescaler

.5.2 L'interruption générée par le TMR0

Le timer TMR0 est donc un compteur qui incrémente sa propre valeur. Il est piloté par l'horloge, qui commande egalement le microcontrôleur, ou par des fronts montant ou descendant présents sur l'entrée RA4. Le timer est très souvent utilisé pour générer des intervalles de temps précis, en jouant justement sur la fréquence de l'horloge et sur le rapport de division introduit par le prescaler. Pour utiliser le timer de cette façon, nous vous conseillons de travailler avec le signal que le timer génere lui-même quand la valeur du registre TMR0 passe de FFh à 00h. En fait, quand le timer arrive à la fin de son comptage, c'est-à-dire à FFh, dans l'incrémentation suivante, le registre TMR0 est mis à 00h et le bit D2 du registre INITCON, nommé T0IF, est mis à 1. Ceci détermine également une demande d'interruption au microcontrôleur, qui ira exécuter une routine donnée en réponse à un tel événement.

Supposons, par exemple, que vous vouliez allumer ou éteindre une LED à inter valles réguliers. Vous devrez charger le prescaler de façon à obtenir l'intervalle de clignotement demandé et faire en sorte qu'à chaque fois qu'une interruption est générée par la fin du comptage effectué par le timer, la routine qui répond à cette interruption aille inverser l'état logique présent sur la patte à laquelle la LED est reliée.

.6 Le Watchdog Timer WDT (Chien de garde)

Le Watchdog pourrait se traduire "chien de garde" ou plus simplement "surveillant". Le Watchdog est un Timer (compteur) qui est normalement utilisé dans les systèmes à microcontrôleurs comme système de sécurité afin d'éviter qu'une cause accidentelle et non prévue par le programmateur.
C'est un compteur 8 bits incrémente en permanence (même si le microcontroleur est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le microcontroleur est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester plante en cas de blocage du microcontrôleur par un processus indesirable non contrôlé.
- Si le le microcontroleur est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là ou elle s'est arrêtée, avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations L'horloge du WDT a une période voisine de 70 μs ce donne un Time-Out toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passer le signal Time-Out dans un prédiviseur programmable (partagé avec le timer TMR0).

PS2	PS1	PS0	Div
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

L'affectation se fait à l'aide du bit PSA du registre OPTION_REG

- PSA = 1 alors on utilise le prédiviseur
- PSA = 0 pas de prediviseur (affecté à TMR0)

Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (voir tableau ci-contre) L'utilisation du WDT doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un WDT timeOut lors de l'exécution d'un programme, on a deux possibilités :

- Inhiber le WDT d'une façon permanente en mettant à 0 le bit WDTE dans l'EEPROM de configuration ;
- Remettre le WDT à 0 périodiquement dans le programme à l'aide de l'instruction CLRWDT pour éviter qu'il ne déborde

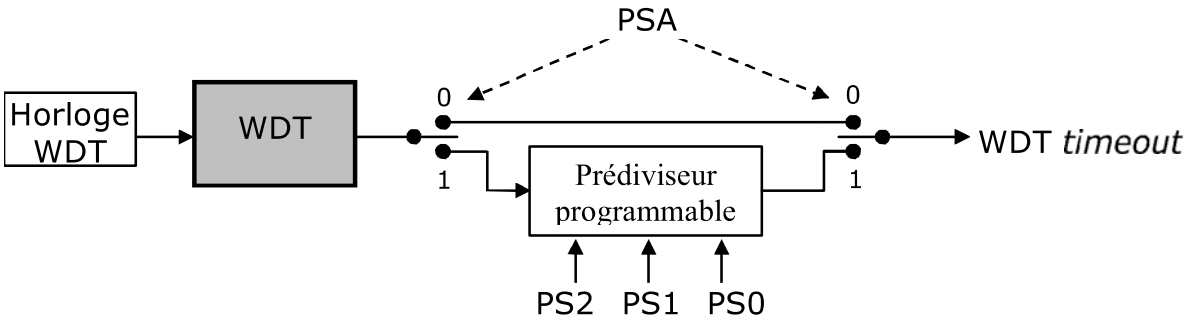


FIG. 7 – Schéma synoptique du timer WDT

Bibliographie

[1] Dogan Ibrahim, *Advanced PIC Microcontroller Projects in C From USB to RTOS with the PIC18F Series,* Elsevier Ltd, USA, 2008.

[2] Cristian Tavernier, *Programmation en C des PIC,* DUNOD, Paris, 2005.

[3] Cristian Tavernier, *Microcontrôleurs PIC 10, 12, 16 Description et mise en oeuvre* DUNOD, 3e édition, Paris, 2007.

[4] Pascal MAYEUX, *Apprendre la programmation des PIC par l'expérimentation et la simulation* , DUNOD, Paris, 2005.

[5] G.C. Buttazzo , *Hard real time computing systems, predictable scheduling, algorithms and applications,* Kluwer Academic Publishers, 2004.

[6] F. Cottet , J. Delacroix, C. Kaiser et Z. Mammeri , *Ordonnancement temps reel,* Hermes, 2000.

[7] K. Kopetz, *Real Time Systems, Design Principles for Distributed Embedded Applications,* Kluwer Academic Publishers, 1997.

[8] J.W.S. Liu , *Real time systems,* Prentice Hall, 2000.

[9] K. Curtis *Embedded Multitasking with small microcontroller,*, Elsevier Inc, USA, 2006.