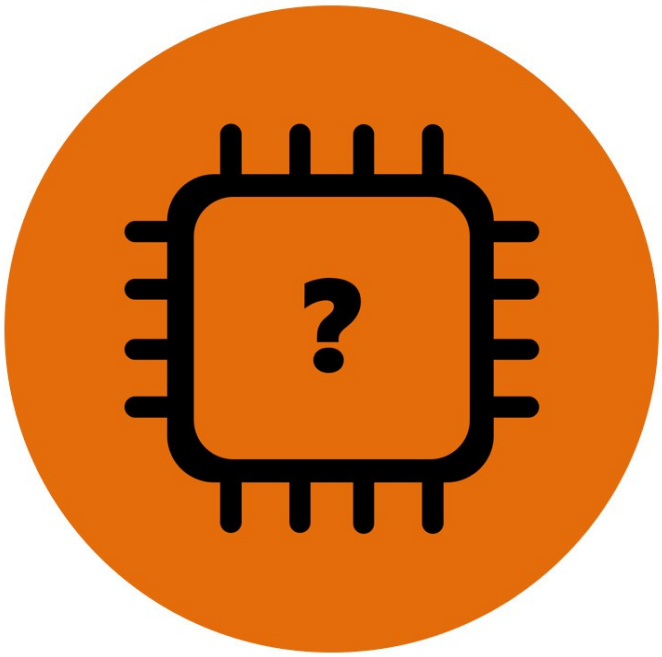
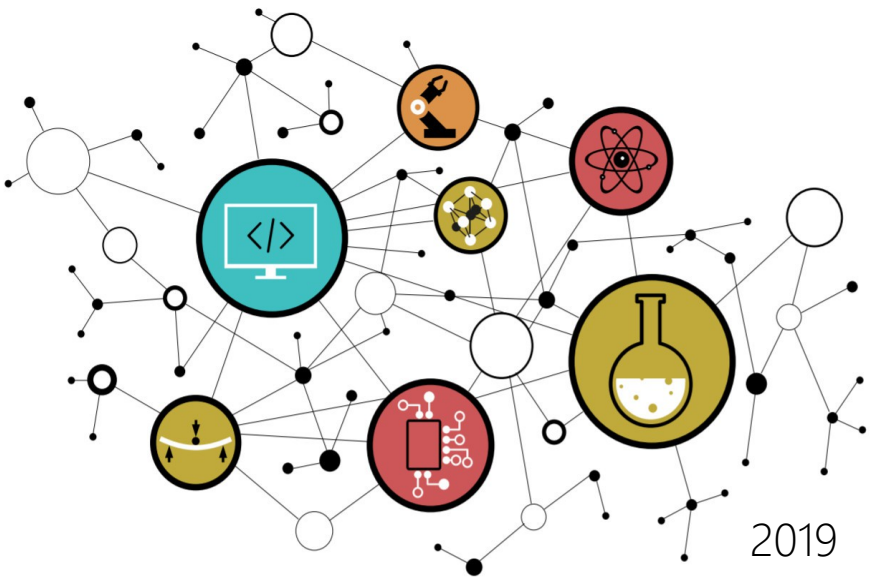
FONCTIONS PERIPHERIQUES



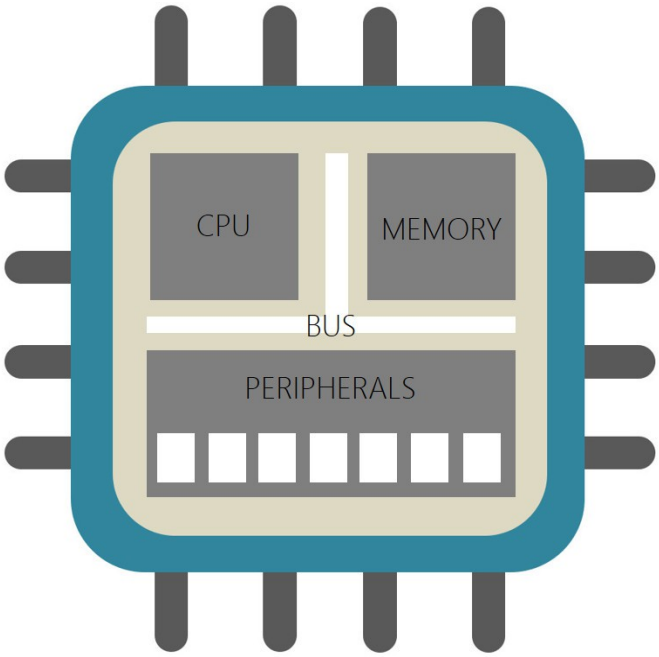


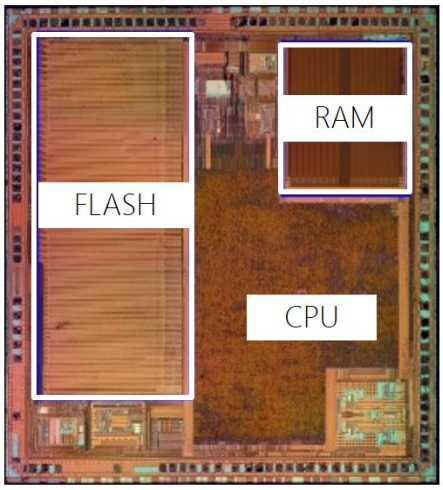
2019

SOMMAIRE

1. PREAMBULE

1. PREAMBULE





Die MCU 32bits Microchip
MCU PIC32MX340F512
MIPS32 M4K Core 5-stage Pipeline
Memory 512Ko Flash / 32Ko SRAM

Ce document a été pensé pour présenter les architectures génériques de périphériques majeurs (GPIO, Timer et UART) présents sur la grande majorité des MCU du marché (Micro Controller Unit ou Microcontrôleur). Pour chaque périphérique, une illustration technologique sur PIC18 (MCU 8bits propriétaire Microchip) sera également proposée ainsi qu'un exemple de configuration bas niveau en assembleur.

Rappelons que sur processeur MCU, la mémoire et le CPU (Central Processing Unit) sont respectivement chargés de stocker l'information et de la traiter. La Flash non-volatile mémorise de façon persistante le programme (ou code) et la RAM volatile mémorise à l'exécution seulement les données en cours de manipulation par le CPU. D'une implémentation technologique à une autre, l'empreinte silicium de ces deux éléments fondamentaux peuvent prendre plus ou moins de place sur le die (puce silicium). L'exemple ci-dessus montre les contraintes d'intégration d'un CPU 32 bits ''performant'' (CPU MIPS32 M4K) ainsi l'emport de ressources importantes de mémoire. Les services proposés par un processeur seront toujours le fruit de compromis technologiques liés à l'intégration. A titre indicatif, d'un point de vu littéral, l'ensemble CPU/Mémoire représente déjà à lui seul un processeur. Une fois l'information traitée et stockée, nombreuses sont les applications chargées de les échanger vers l'extérieur du système. Les périphériques entrent alors en jeu.

Les fonctions matérielles spécialisées périphériques, plus communément nommées périphériques (à l'ensemble CPU/Mémoire), jouent généralement 3 rôles primordiaux :

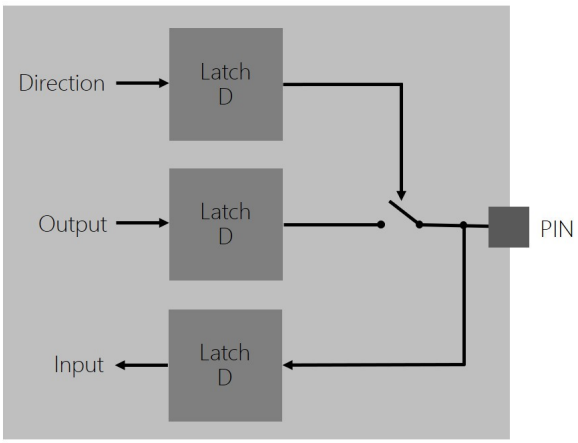
- *Communiquer :* fonctions spécialisées d'échange d'information implémentant un protocole de communication souvent normalisé voire standard (UART, SPI, I2C, USB, Ethernet, CAN, etc)
- *Convertir :* Ponts entre domaines de l'analogique et du numérique (GPIO, ADC, PWM, DAC, etc)
- *Traiter :* fonctions spécialisées de traitement (comptage, calcul, copie, etc). Permet au CPU de se dédier à la supervision du système (application) voire au calcul (Timer, DMA, Crypto, FFT, etc)

2. GPIO

GENERAL PURPOSE INPUT OUTPUT

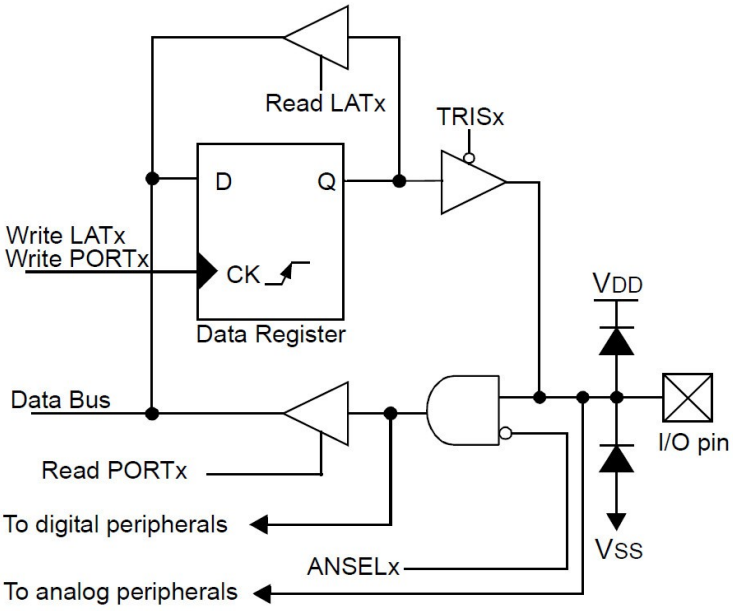
2. GPIO – GENERAL PURPOSE INPUT OUTPUT

2.1. Présentation

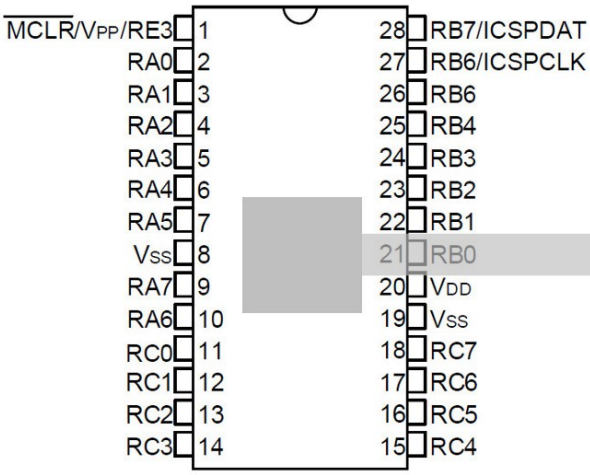


Une broche d'entrée sortie généraliste ou GPIO (General Purpose Input Output) offrira toujours un service de configuration de la direction (entrée ou sortie) de type TOR (Tout Ou Rien) puis un service d'utilisation en lecture ou en écriture. Chacun des états logiques de configuration et d'utilisation sera sauvegardé dans des bascules assurant la mémorisation logique de ces mêmes états. Nous nommons un ensemble de bascules un registre. Il peut s'agir de registres de configuration, d'utilisation voire d'états permettant de connaître l'état (traitement réalisé ou en cours) d'un élément du système, par exemple un périphérique. D'une implémentation technologique à une autre (MCU Microchip PIC18, MCU STMicroelectronics STM32, MCU Texas Instruments MSP430, etc) ces registres auront des fonctions, noms, et tailles différentes.

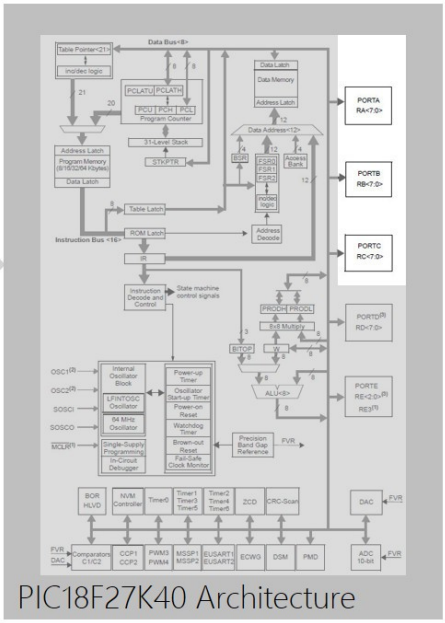
2.2. GPIO sur MCU PIC18



Le schéma précédent est extrait de la documentation technique ou datasheet d'un MCU PIC18F27K40 proposé par Microchip. Il présente plus en détail l'architecture matérielle réelle cachée derrière chaque GPIO. Un port d'entrée sortie (PORTA, PORTB et PORTC sur PIC18F27K40 avec boîtier 28 broches) représente un ensemble de 8 broches ou pins de type GPIO. Les ports sont manipulables par utilisation de registres 8bits (regroupement de 8 GPIO). Néanmoins, ces broches peuvent être configurées et utilisées pour d'autres usage en fonction de l'application (ADC, Timer, UART, SPI, I2C, etc).



MCU PIC18F27K40
Packages SPDIP, SSOP, SOIC



PIC18F27K40 Architecture

Configuration sur MCU PIC18

Sur PIC18, les registres de configuration en entrée sortie des ports sont nommés TRISx (x=A,B, C sur PIC17F27K40) pour Tri-state (haut ou 1, bas ou 0, et haute impédance ou Z). Les registres d'écriture sont nommés LATx (pour Latch) et les registres de lecture PORTx. *Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre relatif aux GPIO (I/O Ports) afin d'observer la configuration des registres*. La séquence assembleur PIC18 ci-dessous présente des exemples de configuration et de gestion des ports.

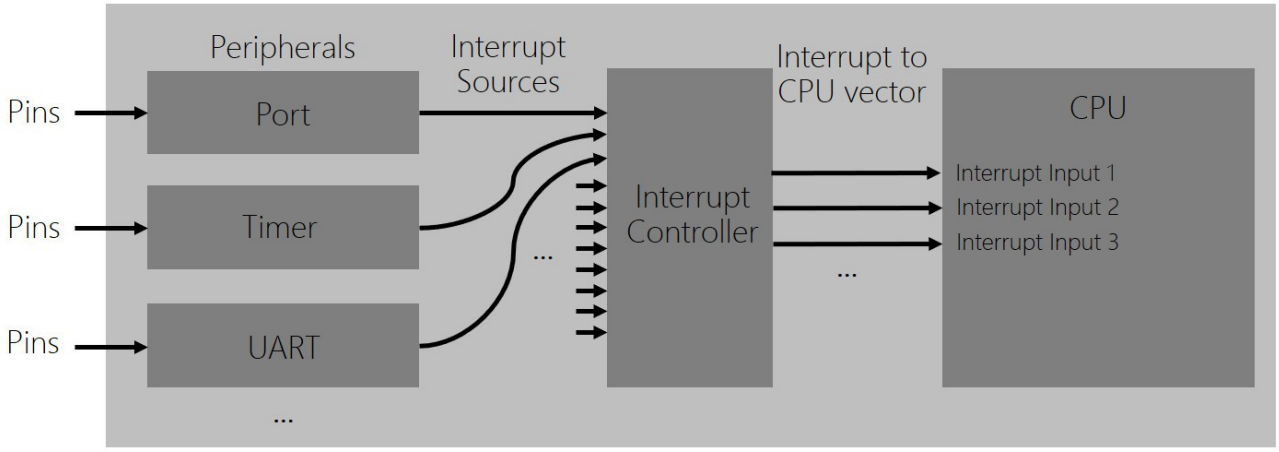
```
; all PORTA pins in inputs
MOVLW       0xFF
MOVWF       TRISA
; all PORTB pins in outputs
MOVLW       0x00
MOVWF       TRISB
; pin RC3 in output
BCF       TRISC, 3
; pin RC0 in input
BSF       TRISC, 0
```

```
; read all PORTA pin and save value in W (Work) CPU register
MOVF       PORTA, 0
; pins RB0-RB3 are set to low level and RB4-RB7 are set to high
MOVLW       0xF0
MOVWF       LATB
; set RC3 to high level
BSF       LATC, 3
; test if input RC0 level is low and perform action
BTFSC       PORTC, 0
<do_this_if_high>
<do_this_if_low>
```

3. INTERRUPTION

3. INTERRUPTION

3.1. Présentation

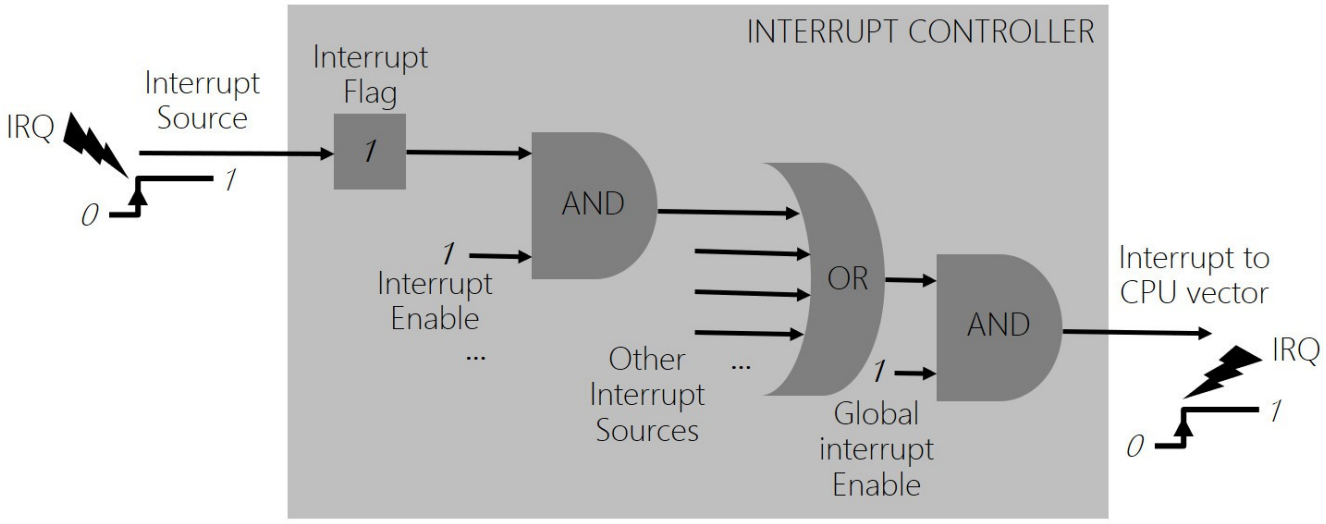


Le concept d'interruption matérielle est primordial sur tout processeur conçu autour d'un CPU. Une interruption matérielle sera toujours rattachée à un périphérique. Une fois configuré, un périphérique est un composant indépendant et autonome dans le processeur lui-même. En fonction de son rôle (communiquer, convertir ou traiter), et si il a été configuré dans ce but (contrôleur d'interruption), lorsqu'un événement physique se produit au sein d'un périphérique, celui-ci va tenter d'interrompre le CPU en cours d'exécution d'instructions ou en veille. Une interruption est un signal physique partant d'un périphérique et arrivant (sous conditions) jusqu'au CPU. Les entrées d'interruption d'un CPU sont le plus souvent classées par niveau de priorité. Il est à noter qu'une ''Belle'' application, ne réalisera notamment des actions qu'au moment opportun, lorsqu'un traitement est strictement nécessaire. Le reste du temps, le système de supervision devra rester au repos (veille). La mise en veille CPU correspond à sa capacité d'inaction (Pipeline inactif). Alors, seuls les périphériques restent en attente d'informations et servent d'interfaces entre le système et son environnement physique extérieur.

Source et requête d'interruption

Dans la majorité des cas, il existe au moins autant de sources d'interruption que de périphériques. Une source d'interruption est un chemin physique unidirectionnel (conducteur sur le die) allant d'un périphérique au contrôleur d'interruption. Une interruption (par abus de langage) ou requête d'interruption ou IRQ (Interrupt Request) correspond au passage d'un état logique inactif à actif sur une source d'interruption. Un périphérique envoie alors une requête au CPU et demande à interrompre son traitement en cours. Cependant, encore faut-il que le CPU y soit sensible.

Logique de démasquage d'interruption

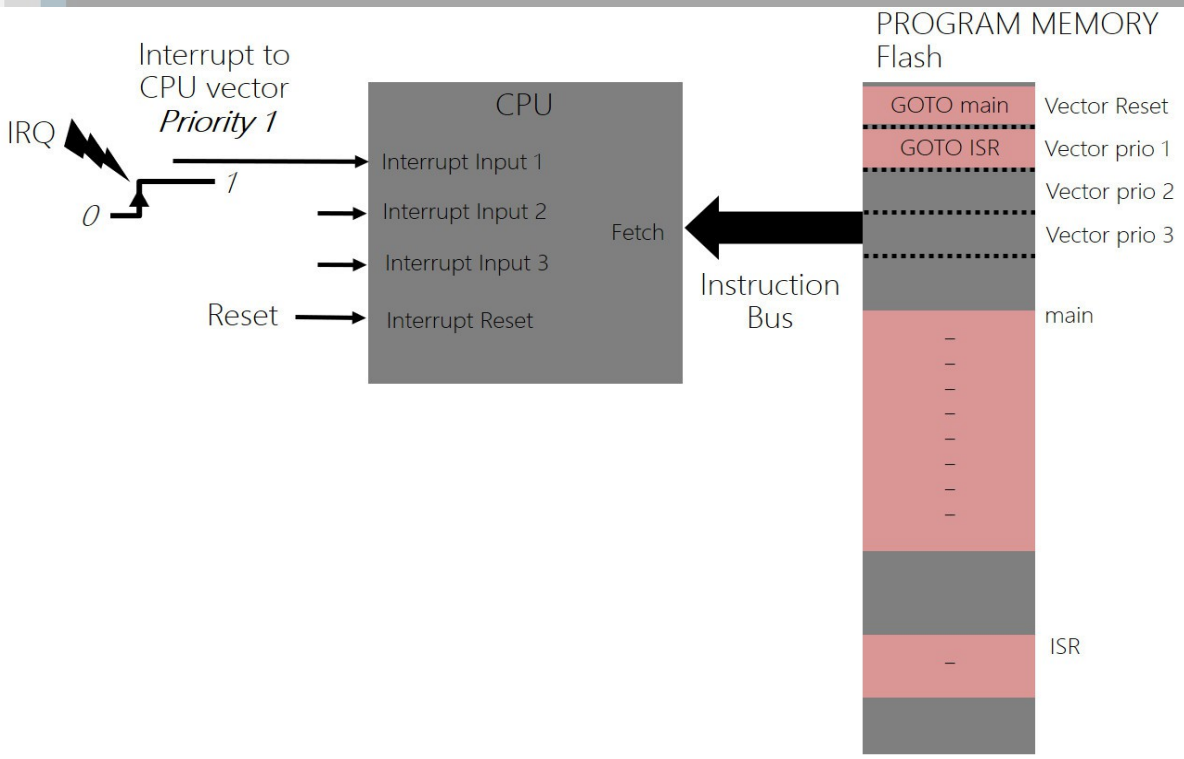


Il est à noter qu'à la mise sous tension, généralement le CPU n'est sensible à aucune source d'interruption, hormis celle du reset (bouton poussoir externe voire reset logiciel). Même si des périphériques devaient envoyer des IRQ, ils ne pourraient interrompre l'exécution du programme en cours. En effet, l'ingénieur en charge du développement devra démasquer pour le CPU les sources d'interruption une à une en fonction des besoins de l'application. Il s'agit d'une logique combinatoire de démasquage à réaliser afin de configurer le contrôleur d'interruption. Comme pour tous les périphériques, cette configuration se fera par écriture dans des registres. Il y aura en général à minima la validation de la source d'interruption et la validation globale des interruptions pour l'ensemble du processeur. De même, les requêtes d'interruption ou IRQ seront mémorisées (interrupt flag est une bascule D) et à acquitter explicitement par mise à zéro dans l'application. Ces acquittements se feront dans les fonctions d'interruption et seront présentés par la suite.

Si une IRQ parvient à traverser le contrôleur d'interruption, elle forcera le CPU à arrêter les traitements en cours et à exécuter un code spécifique (vecteur d'interruption). La commutation est matériellement câblée dans le CPU. Il est souvent associé un niveau de priorité (voire de sous priorité) à un vecteur d'interruption. Ceci permet à l'ingénieur de rendre plus au mois prioritaires des périphériques (Timer, UART, Ethernet, etc) et donc les traitements à réaliser par la suite dans l'application. Le problème se produira lorsque plusieurs périphériques enverront des requêtes simultanément.

Vecteur d'interruption

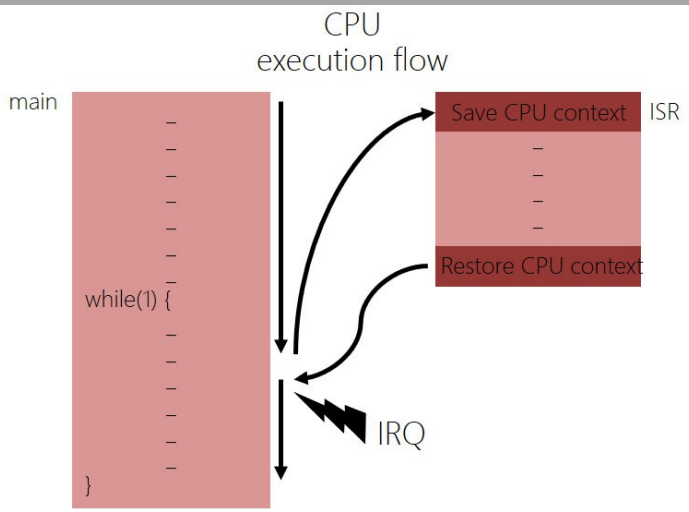
Un vecteur d'interruption est une ''petite'' zone en mémoire programme. Il est donc chargé de mémoriser ''quelques'' instructions. Ces instructions permettent une redirection vers la fonction d'interruption ou ISR (Interrupt Service Routine). Les ISR sont quant à elles accessibles depuis l'application. Il est potentiellement possible d'avoir autant d'ISR que de vecteurs d'interruption dans une application. Rappelons que généralement il est associé un niveau de priorité à un vecteur d'interruption et donc aux ISR liées. Ces priorités sont également configurables par registres.



Attention, le schéma ci-dessous représente deux concepts radicalement différents. Le matériel ou hardware (blocs gris et flèches noires) et le micrologiciel ou firmware (rose, cf. version numérique du document). Le firmware est quant à lui mémorisé en mémoire programme non-volatile, souvent nommée mémoire flash. Il s'agit de l'application logicielle embarquée. Sans système d'exploitation (scheduler), si aucun périphérique ne cherche à interrompre le CPU, alors celui-ci n'exécutera que du code de fonctions appelées depuis la fonction main (hors mise en veille). Dans l'exemple ci-dessus, dès que le CPU capte l'IRQ sur son entrée d'interruption de priorité 1, il commence à aller chercher puis exécuter le code présent dans le vecteur d'interruption de priorité 1, puis par indirection celui de l'ISR associée. Dès que l'ISR est terminée, il reprend l'exécution de l'application exactement à l'instruction à laquelle il avait été interrompu. Nous nommons ce phénomène commutation de contexte (sauvegarde et restauration). Il sera illustré sur PIC18 par la suite dans cet enseignement. De même, les vecteurs d'interruption sont généralement situés aux adresses les plus basses de la mémoire programme et sont sur certaines architectures translatables à d'autres adresses mémoire. Ce n'est pas le cas des PIC18 de Microchip.

ISR ou fonction d'interruption

Une ISR (Interrupt Service/Software Routine) ou fonction d'interruption est un fonction logicielle telle que vous avez pu en rencontrer en langage C. A ceci près, qu'elle ne doit jamais être appelée de façon explicite depuis l'application. Il s'agit de programmation événementielle. Les ISR se réveilleront de façon asynchrone (non prédictible lorsqu'il s'agit de périphériques de communication) sur événement matériel en suivant la logique précédemment présentée. Une ISR possédant le plus haut niveau de privilège d'exécution sur un processeur à CPU (MCU, AP, GPP, DSP, etc), il faut toujours penser à y passer le moins de temps possible à l'intérieur en déportant les traitements longs dans les fonctions appelées depuis le main (code applicatif). Utiliser alors des variables globales pour les échanges.



En respectant le code couleur précédemment utilisé (cf. version numérique pdf), les concepts maintenant présentés sont représentatif du logiciel développé par l'ingénieur. Sans système d'exploitation, les développements sont alors nommés Baremetal, soit nu directement sur le processeur sans système logiciel exploitant la machine (scheduler pour le CPU, gestionnaire mémoire pour la MMU/MPU, drivers ou pilotes pour les périphériques, etc). L'application doit alors impérativement implémenter un while(1). Nous verrons en travaux pratiques comment développer une application en implémentant un scheduler offline.

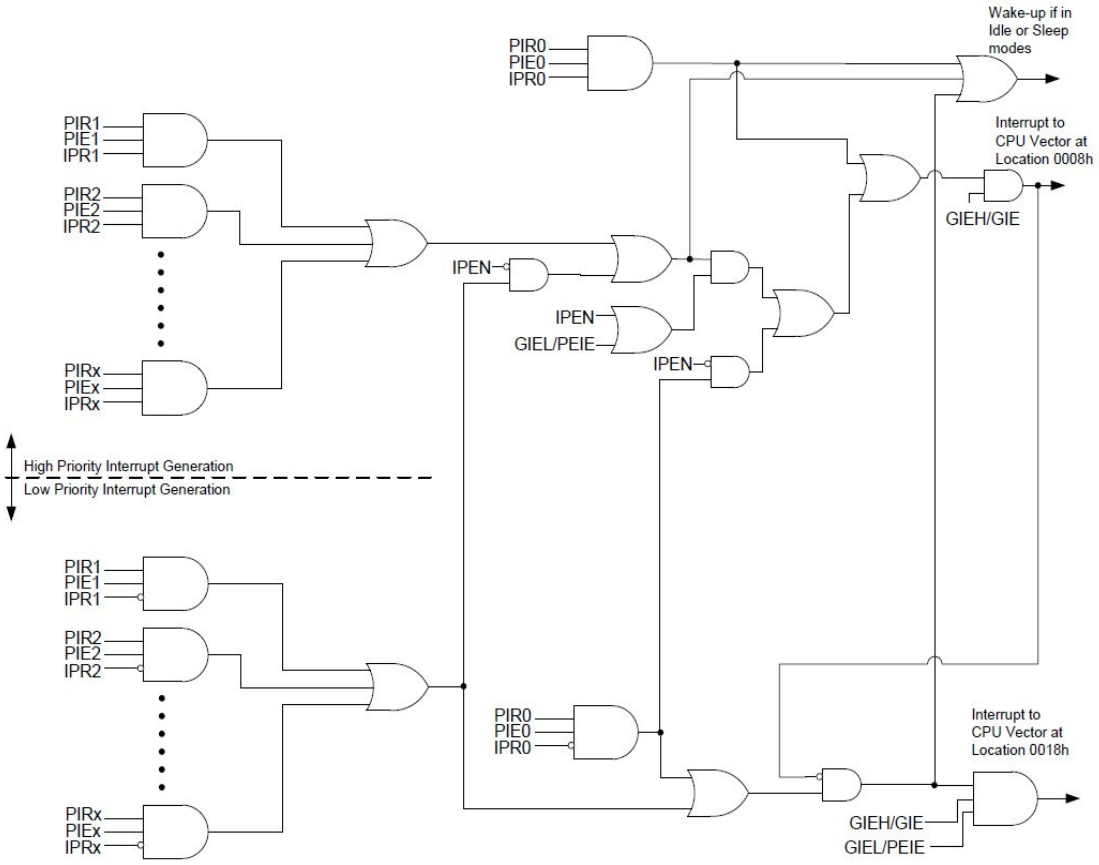
Les ISR étant particulière, d'un processeur à un autre et donc d'un chaîne de compilation à une autre, leur déclaration sera différente. Le plus souvent, un qualificateur de fonction lui est associé. Le compilateur a besoin de ce qualificateur afin d'implémenter les sauvegardes et restaurations de contexte en en-tête et pied de fonction. L'exemple ci-dessous illustre une déclaration d'ISR de priorité haute sur PIC18 en langage C. Pour information, il y a seulement deux niveaux de priorité (haute et basse) et donc deux vecteurs d'interruption sur PIC18 (hors celui du reset).

```
void main (void)
{
   _
   _
   _
   while(1) {
      _
      _
      _
   }
}
```

```
void __interrupt(high_priority) ISR (void)
{
   _
   _
   _
}
```

Vous constaterez que le code C ainsi que le schéma présentés ci-dessus illustrent deux méthodes différentes pour présenter un même concept (texte vs graphique). L'une est une implémentation technologique et l'autre une représentation conceptuelle. La majorité des exercices demandés dans la trame de travaux pratiques seront représentés graphiquement de façon générique. Il sera à votre charge de réaliser les implémentations technologiques C ou assembleur sur PIC18 sous environnement de développement MPLABX.

3.2. Interruption sur MCU PIC18



Les MCU 8bits PIC18 de Microchip ont tous en commun le même CPU, et donc le même jeu d'instructions assembleur ainsi que la même chaîne de compilation (XC8). Il existe néanmoins un très large choix de PIC18 différents au catalogue de Microchip. Il sont tous différenciés par leurs ressources mémoire (Flash et SRAM) ainsi que par leur jeu de périphériques. Un PIC18 intègre en général un large jeu de périphériques et chaque périphérique possède une voire plusieurs sources d'interruption. Il existe donc un certain nombre de registres de configuration pour les interruptions. Nous pouvons classer tous ces registres en 4 familles :

- *INTCON :* configuration globale pour l'ensemble du système (bits GIEH, GIEL, IPEN, etc)
- *PIEx (x=0 à 7) :* configuration des bits de validation (interrupt enable)
- *PIRx (x=0 à 7) :* lecture et acquittement des bits d'interruption (interrupt flag)
- *IPRx (x=0 à 7) :* configuration des bits de priorité (priorité basse ou haute)

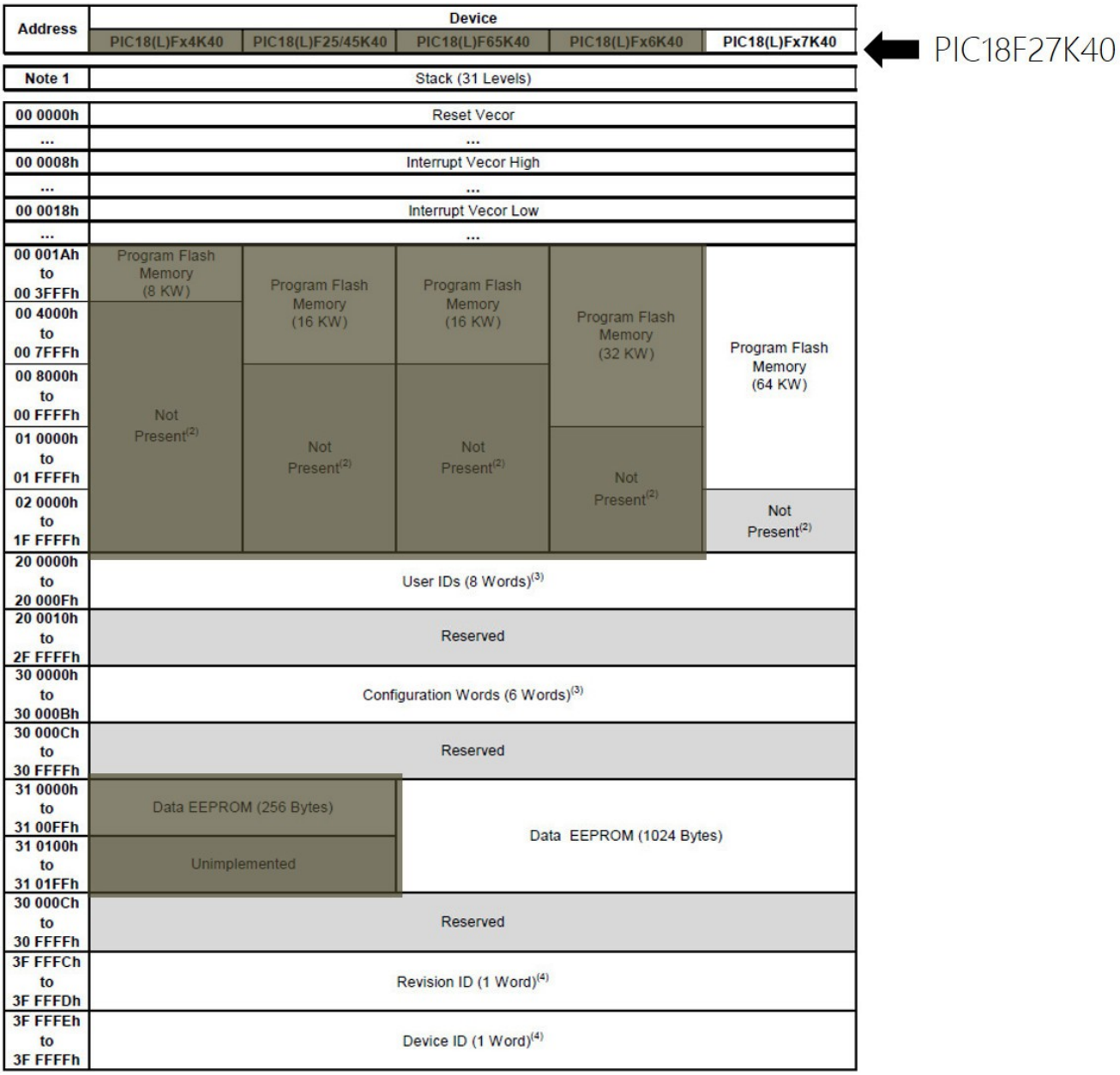
De même, pour chaque périphérique, 3 bits seront alors à configurer (xxx dépend du périphérique à configurer). Ces bits correspondent à des champs dans les registres précédemment cités.

- *xxxIE :* Interrupt Enable
- *xxxIF :* Interrupt Flag
- *xxxIP :* Interrupt Priority

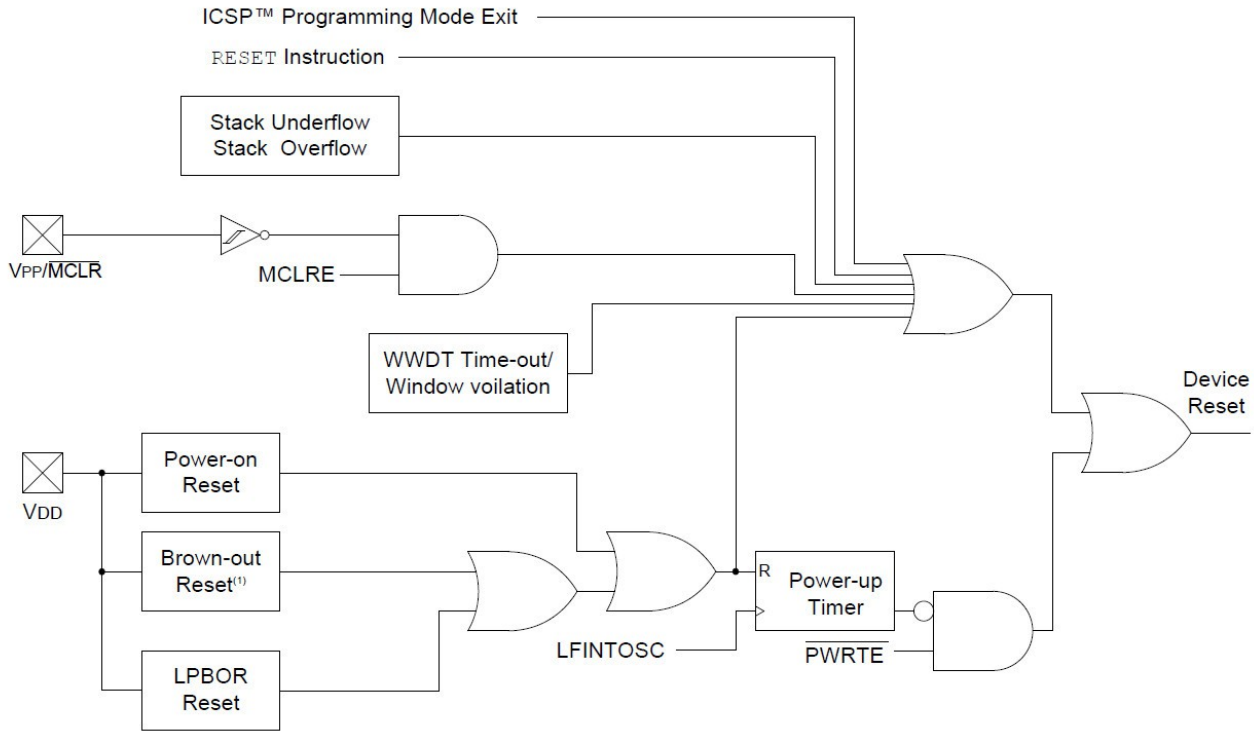
L'exemple suivant présente une configuration d'interruption en assembleur pour le Timer0 :

```
; clear flag, enable and set low priority interrupt
BCF     PIR0, TMR0IF
BSF     PIE0, TMR0IE
BCF     IPR0, TMR0IP
; global interrupt enable
BSF     INTCON, IPEN
BSF     INTCON, GIEL
BSF     INTCON, GIEH
```

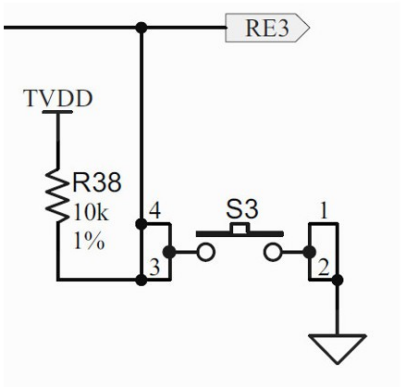
Observons ci-dessous un extrait de datasheet présentant l'organisation de la mémoire programme d'un PIC18F27K40, le MCU utilisé en TP. Nous pouvons également constater l'emplacement des vecteurs d'interruption (adresses 0x000000 reset, 0x000008 priorité haute et 0x000018 priorité basse). Ces emplacements sont les mêmes pour tous les PIC18 du marché. Chaque vecteur d'interruption ne propose que 8o de stockage, donc 4 instructions au maximum peuvent être placées à l'intérieur. Notre MCU possède 128ko de Flash (64KWords avec 1Word=2octets sur notre MCU) et 1ko de mémoire donnée non-volatile EEPROM.

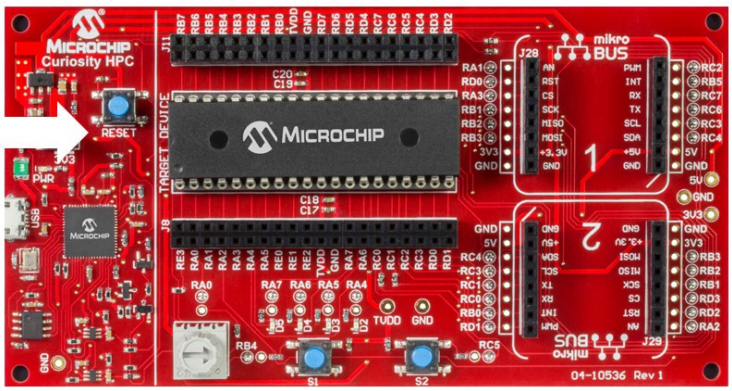


3.3. Reset sur MCU PIC18



Vous pouvez observer ci-dessus la logique d'occurrence d'un reset processeur. Par exemple, si un niveau logique bas est appliqué sur la broche Vpp/$\overline{\text{MCLR}}$ (Master Clear) et si le bit de configuration MCLRE (MCLR Enable) est actif (fait par défaut à la mise sous tension), alors un reset processeur est forcé. Le CPU exécute alors le code présent dans le vecteur d'interruption du reset. En général, celui-ci rappelle la fonction main ou la fonction de startup utilisée par défaut par la chaîne de compilation. Observons par exemple ci-dessous comment est câblé le bouton poussoir du reset sur la carte Curiosity HPC de Microchip utilisée en TP. La broche Vpp/$\overline{\text{MCLR}}$ est également la broche RE3 du processeur.

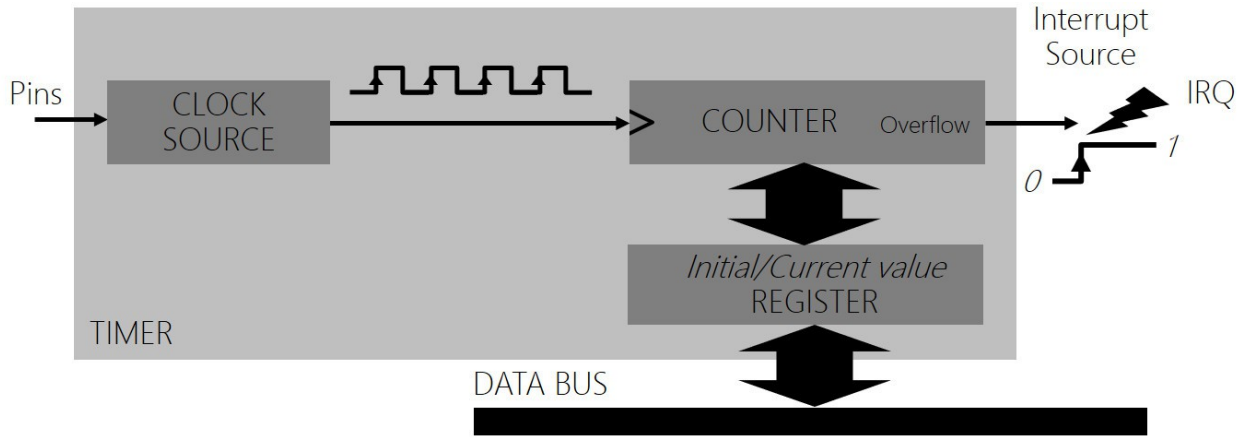




4. TIMER

4. TIMER

4.1. Présentation



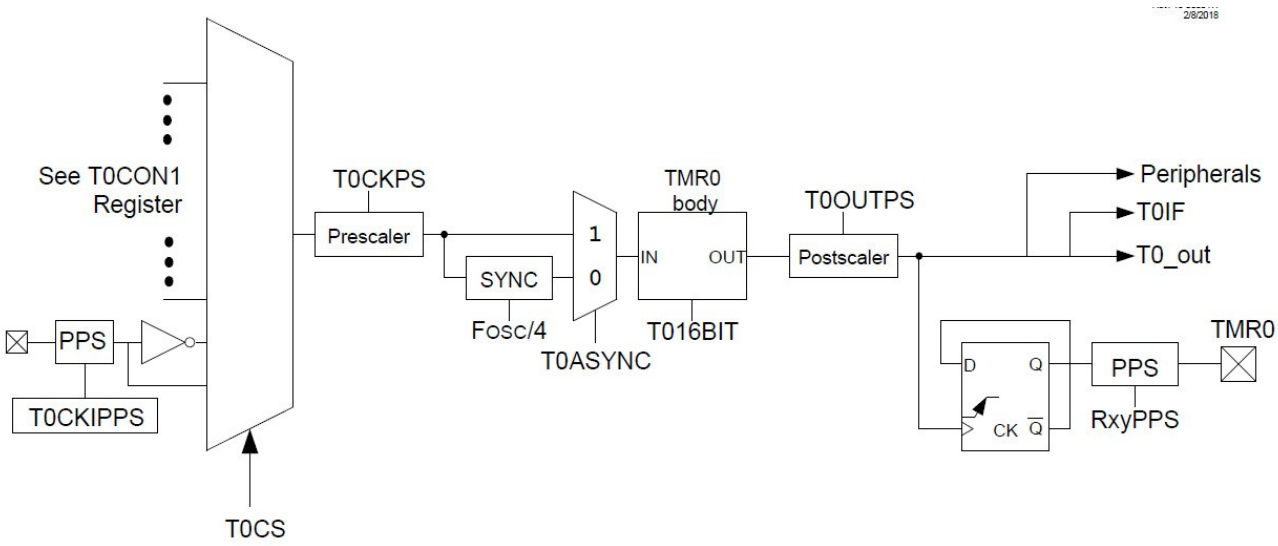
Un Timer sera toujours conçu autour d'un compteur numérique. Il est donc dédié aux opérations de comptage, et le plus souvent à la gestion de bases de temps. Afin de générer une référence temporelle à l'aide d'un compteur numérique, nous pouvons jouer sur 3 éléments. La période de comptage (fréquence de l'horloge de référence), la valeur initiale de comptage et le nombre de bits utilisés par le compteur (8-16-32-64 bits) avant débordement. L'ensemble de ces trois éléments constituent à minima un Timer. Les Timers les plus élémentaires rencontrés sur processeur ne possèdent même pas de référence initiale de comptage ni d'horloge de référence configurable. Ils démarrent en partant de 0 à la mise sous tension et comptent à la fréquence de travail du CPU. Il sont nommés TSC (Time Stamp Counter) et sont présents dans chaque CPU d'un grand nombre de processeurs du marché (MCU, AP, GPP, DSP, etc). Ils sont souvent utilisés pour réaliser de la mesure de temps d'exécution de programme. Néanmoins, un Timer applicatif est souvent plus riche en services matériels et fonctionnalités (comparateur numérique, rechargement automatique, module PWM, etc). Leur configuration peut sur certaines architectures être même ardue.

Prenons l'exemple d'un Timer 8bits et calculons une valeur initiale de comptage à recharger après débordement afin d'obtenir une base de temps. Par défaut, après débordement (par exemple sur 8bits, $255_{10} + 1_{10} = 1111\ 1111_2 + 1_2 = 1\ 0000\ 0000_2 = 256_{10}$ soit un résultat sur 9bits), le Timer reprend le comptage à 0 et le bit de débordement devient actif (overflow) :

- Comptage de 0 à 255 soit de 0x00 à 0xFF (compteur 8bits)
- Horloge de référence de période 1ms (après configuration)
- Quelle serait la valeur initiale de comptage à charger afin d'obtenir une base de temps de 100ms ?
- Réponse : *155 ou 0x9B*
- Pourquoi : *nous devons compter 100 cycles, le compteur déborde au 256^{ième} cycle. Nous devons donc débuter le comptage à la valeur 155*

Sur un Timer, le signal à la source de l'interruption ou IRQ n'est autre que le bit de débordement (overflow flag). Ce signal électrique est alors envoyé au CPU (cf. chapitre Interruption).

4.2. Timer0 sur MCU PIC18



Le MCU PIC18F27K40 utilisé en TP intègre 4 Timers 16bits (Timer0/1/3/5) ainsi que 3 Timers 8bits (Timers 2/4/6). Le schéma bloc ci-dessus ne présente que la structure interne du Timer0. Ce Timer est configurable en mode 16bits ou 8bits. Seul le mode 16bits est présenté ci-dessus et sera utilisé en TP. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes : *compteur 16bits*, *ensemble assurant la référence d'horloge* et *le signal d'interruption ?*

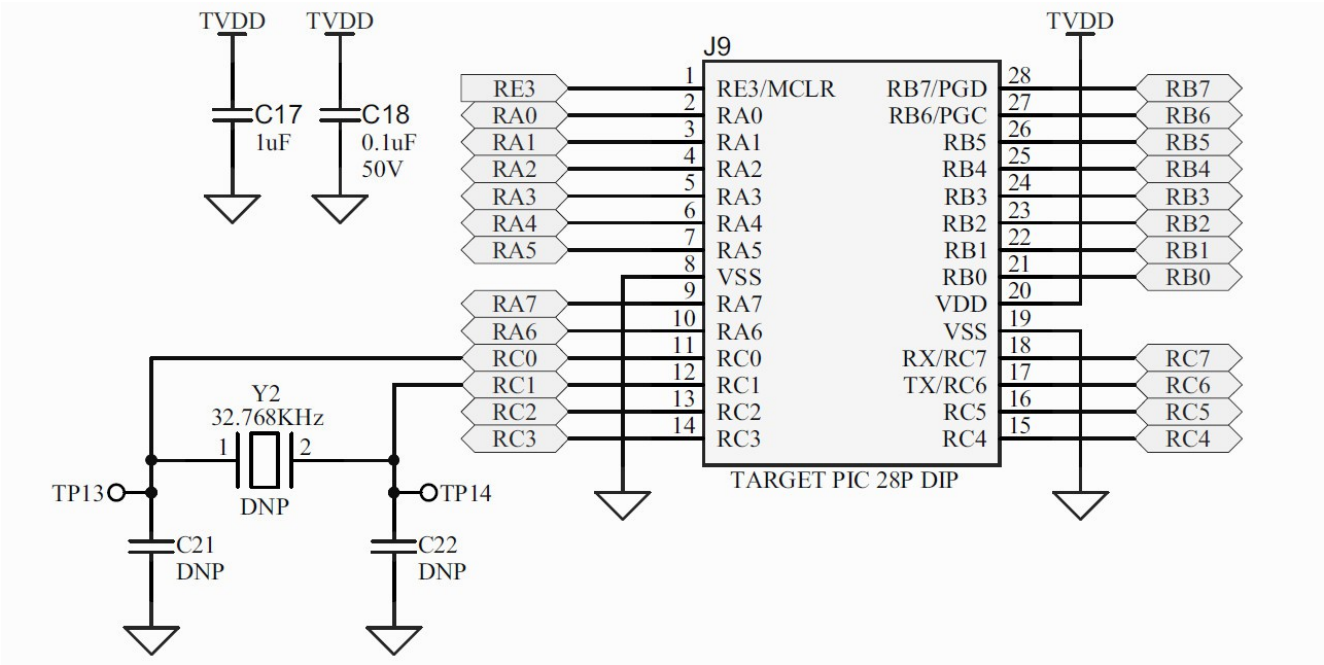
Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre relatif au Timer0 (Timer0 Module) afin d'observer la configuration des registres. Le Timer0 possède deux registres 8bits de configuration (T0CON0 et T0CON1) et deux registres 8bits de chargement de la valeur initiale de comptage (TMR0L et TMR0H). Analysons le registre 8bits T0CON1 chargé de configurer l'horloge de référence. Nous pouvons observer sur le schéma ci-dessus que les champs T0CS (Timer0 Clock Source Select) et T0ASYNC (Timer0 Input Synchronization) permettent de piloter deux multiplexeurs d'aiguillage chargés de router la référence interne ou externe d'horloge. De même, le champ T0CKPS (Timer0 Clock Prescaler Select) permet de configurer un diviseur de fréquence (1:1, 1:2, 1:4, etc, 1:32768) avant attaque du compteur 16bits.

Name: T0CON1
Offset: 0xFD6

Timer0 Control Register 1

Bit	7	6	5	4	3	2	1	0
	T0CS[2:0]			T0ASYNC	T0CKPS[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Configuration sur MCU PIC18



Attention, la configuration proposée n'est pas celle demandée dans les TP. Elle sera donc à adapter. L'exemple suivant présente comment réaliser une base de temps très précise de 1 seconde (cf. montre à Quartz, dérive du quartz externe de +/-10ppm ou Particules Par Millions soit +/-0,00001%). Cet exercice pourrait d'ailleurs être réalisé sur la carte Curiosity HPC utilisée en TP à l'image de la capture du schéma électrique ci-dessus mais en utilisant néanmoins le Timer3.

```
; timer disable, 16bits mode, post-scaler 1:1
MOVLW   0b00010000
MOVWF   T0CON0
; pin select with extern 32.768KHz resonator
; no CPU synchro, pre-scaler 1:1
MOVLW   0b00110000
MOVWF   T0CON1
; initial decimal value 32767 (0x7FFF)
; Always write TMR0H before TMR0L
MOVLW   0x7F
MOVWF   TMR0H
MOVLW   0xFF
MOVWF   TMR0L
; timer enable
BSF     T0CON0, T0EN
```

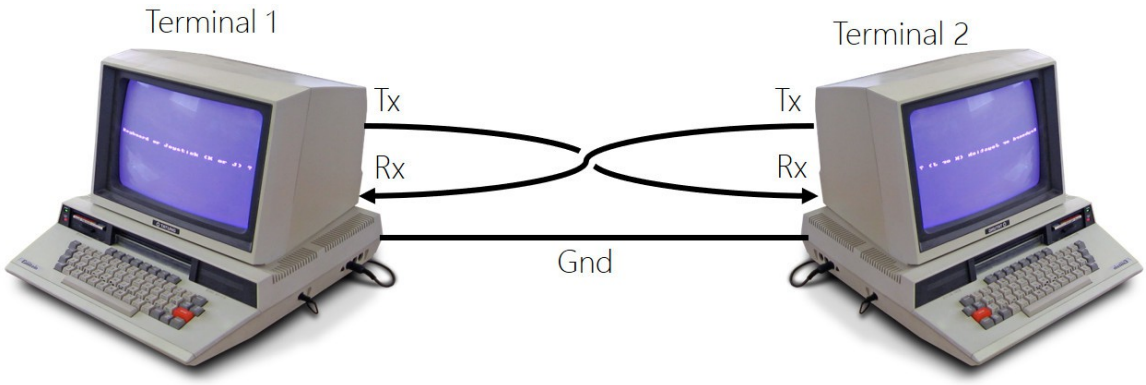
```
; clear flag, enable and set low priority interrupt
BCF     PIR0, TMR0IF
BSF     PIE0, TMR0IE
BCF     IPR0, TMR0IP

; global interrupt enable
BSF     INTCON, IPEN
BSF     INTCON, GIEL
BSF     INTCON, GIEH

; this program generate an interruption after
1s !
```

5. UART

UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

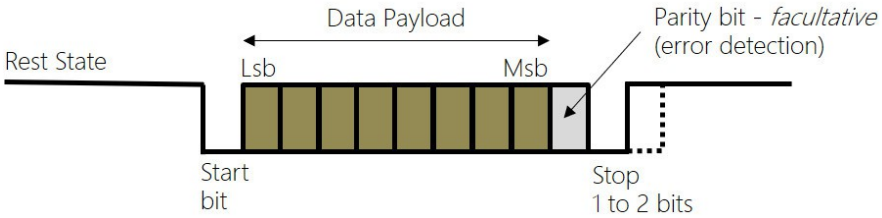
5. UART – UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

5.1. Présentation



Les périphériques UART sont rencontrés depuis maintenant longtemps sur processeur numérique et ordinateur. A titre indicatif, la norme RS232 à été spécifiée en 1981 et n'a progressivement disparu sur ordinateur qu'à partir des années 2000 suite à l'arrivée de l'USB (norme USB1.0 Universal Serial Bus en 1996). L'UART reste néanmoins encore très rencontré en Systèmes Embarqués, et elle le sera encore probablement très longtemps. Ceci s'explique par le fait qu'elle répond pleinement à un besoin, donc nulle nécessité de développer un nouveau protocole. Elle est de plus bien connue des ingénieurs du domaine. L'UART est spécialisé dans l'échange de caractères (données sur 8bits de façon générique) en topologie point à point entre 2 systèmes. Le tout avec des débits plutôt faibles à notre époque (contrôle de procédé, test et prototypage, mesure, contrôle de module de communication, etc). Étant spécialisé dans l'échange de caractères, si un Homme cherche à interagir directement avec un périphérique UART depuis un ordinateur, celle-ci se configurera et s'utilisera depuis un terminal asynchrone de communication (minicom, kermit, PuTTY etc sous GNU\Linux et TeraTerm, PuTTY, etc sous Windows).

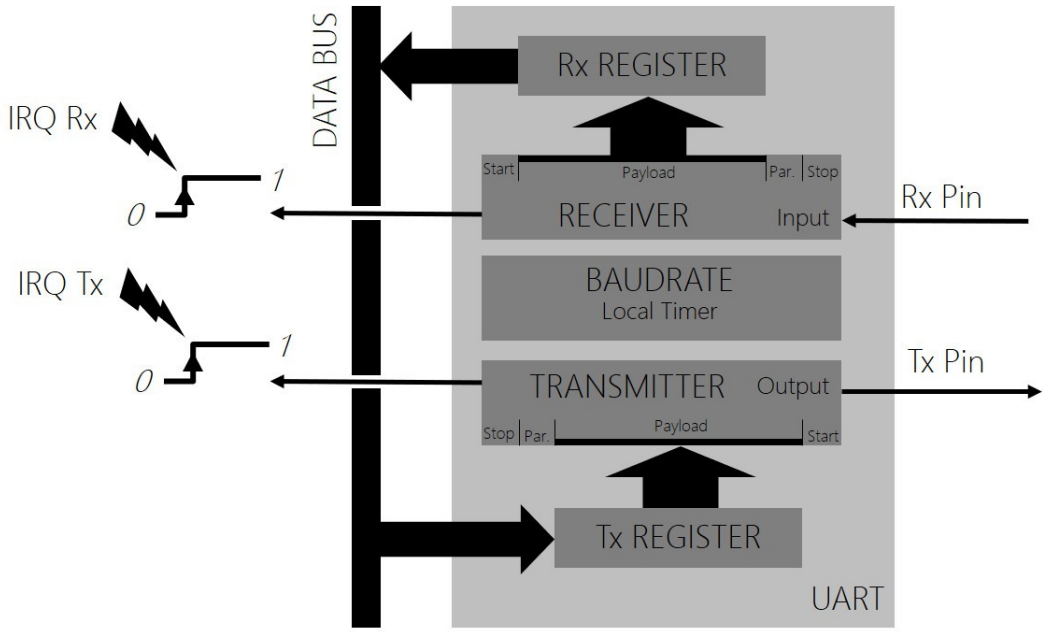
Protocole de communication



Ne pas confondre l'UART (le périphérique) et le protocole de communication (stratégie d'échange). Celui-ci est le plus souvent nommé *liaison série asynchrone*. Il s'agit de l'un des rares derniers protocoles asynchrones de communication. Cela signifie que l'émetteur n'envoie aucune information concernant le débit de transfert, à l'instar des SPI et I2C (conducteur d'horloge dédié) ou encore de l'USB et l'Ethernet (champs d'horloge en en-tête de trame assurant la synchronisation de la PLL de réception). Ne jamais oublier de bien configurer récepteur et émetteur au même débit.

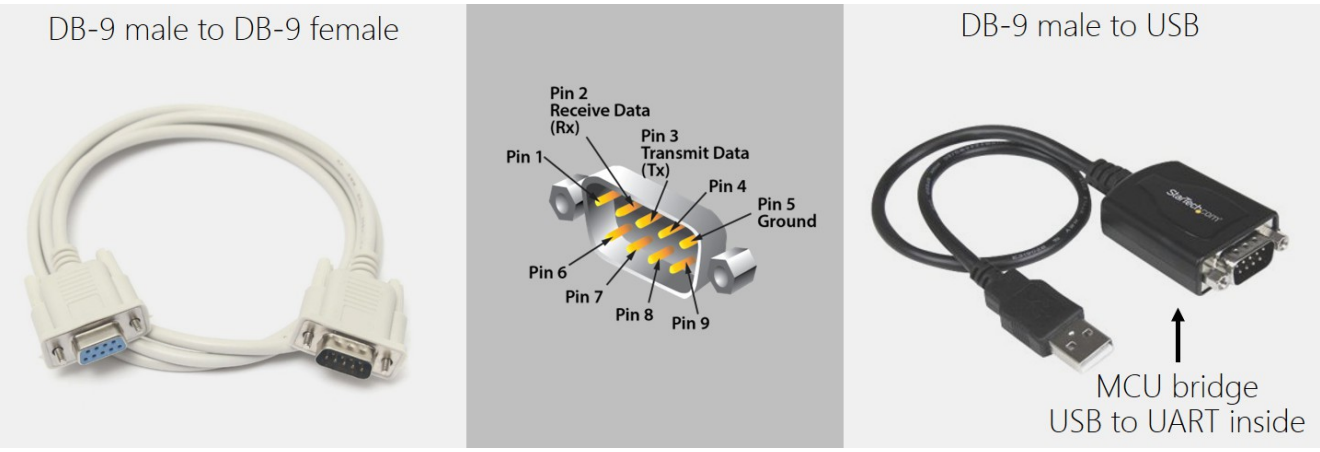
L'état au repos de la ligne de communication est l'état logique haut. Une trame débutera toujours par 1 bit de start (état logique bas). Suivent 7 ou 8bits de données utiles ou payload, 1 bits de parité facultatif permettant une gestion rustique de détection d'erreur. Si ce bit est utilisé (parité paire ou impaire) par modification de l'état de ce bit, l'émetteur doit impérativement garantir un nombre pair ou impair de 1 pour les données en addition de ce bit (à configurer à l'émission et la réception). Une trame se termine par 1 à 2 bits de stop. Néanmoins, la liaison série asynchrone est le plus souvent utilisée avec la configuration suivante : 1 bits de start, 8 bits de payload et 1 bit de stop (solution minimale). Une trame série complète d'informations est alors constituée de 10bits dont 8 bits utiles (différence entre débit réel et utile). Par exemple, avec un débit configuré à 9600Bd/s (Baud/s ou symbole/s ou bits/s pour une liaison série asynchrone), l'envoie d'un bit dure environ 100us et donc l'envoie d'un caractère environ 1ms.

Périphérique UART



Un périphérique UART peut être vu comme deux périphériques dissociés. Un récepteurr et un transmetteur. Néanmoins, les deux implémentent le protocole d'une liaison série asynchrone et leur configuration est similaire (protocole et débit). Un Timer local (Baurate generator) est souvent présent dans une UART afin de configurer le débit de communication. Comme tout périphérique série de communication, le transmetteur et le récepteur sont conçus autour de registres à décalage. Le récepteur est chargé de récupérer les trames bit à bit dans son registre interne à décalage, de détecter d'éventuelles erreurs de transmission et d'enlever l'enveloppe protocolaire de la trame pour ne récupérer que les données utiles. A chaque nouvelle trame, la donnée utile est chargée dans un registre de travail (Rx register ci-dessus) et une requête d'interruption est envoyée au CPU (IRQ Rx). Le transmetteur fait quant à lui le travail inverse. L'application demande à envoyer une donnée par écriture dans le registre de transmission (Tx register ci-dessus, opération atomique de qq cycles CPU). Néanmoins, cela ne signifie pas que la donnée ait été transmise. Le transmetteur est alors responsable de charger la donnée utile dans son registre interne à décalage, de rajouter l'enveloppe protocolaire (start, stop voire parité), puis d'envoyer bit à bit les données en respectant de débit configuré. Une fois la donnée envoyée, une requête d'interruption est envoyée au CPU (IRQ Tx).

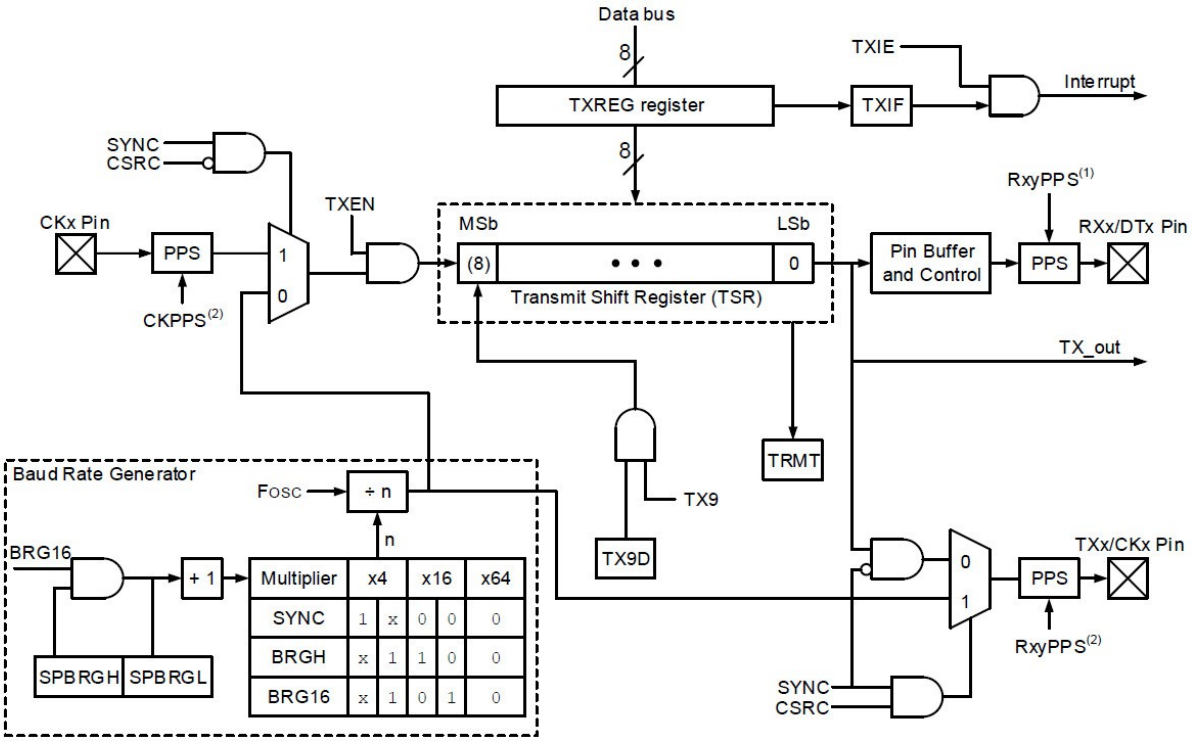
Norme RS232



Ne pas confondre la norme RS-232 (nommée aussi EIA 232) avec le protocole d'une liaison série asynchrone et un périphérique UART. Cette norme est apparue en 1981 et a donnée naissance aux interfaces port COM sur ordinateur sous Windows (remplacés par l'USB depuis les années 2000). Cette norme de communication de machine à machine utilise des UART et implémente une liaison série asynchrone, mais tend à standardiser les débits de communication, les connecteurs et câbles associés, les longueurs de câbles, les niveaux de tension sur les conducteurs physiques, etc. Cette norme standardise donc les contraintes et les limites permettant de faciliter l'interfaçage et la communication de machines entre elles. Observons quelques une de ces limites :

- *Longueurs de câble :* 60m (2,4KBd/s), 15m (9,6KBd/s) et 2,6m (56KBd/s)
- *Niveaux de tensions (logique inversée) :* niveau logique 0 (de +3V à +25V) niveau logique 1 (de -3V à -25V)
- *Codage bit :* NRZ (Non Return to Zero)
- *Connecteurs :* DB9 (9 broches)

5.1. UART sur MCU PIC18

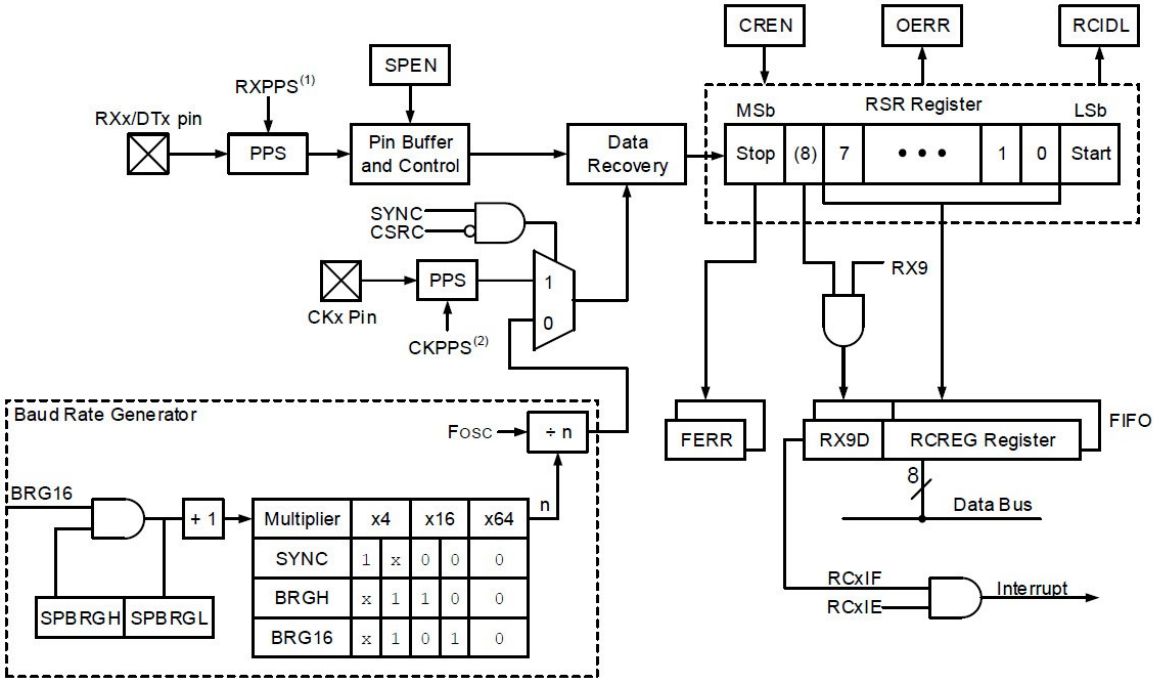


Le MCU PIC18F27K40 utilisé en TP intègre 2 UART (UART1/2). Les deux seront configurés et utilisées durant cet enseignement. Le schéma bloc ci-dessus présente la structure du transmetteur. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes : *registre interne à décalage de transmission*, *registre de travail de transmission (pour écriture de la payload par l'application)*, *ensemble assurant la référence d'horloge (débit), broche Tx de sortie* et *le signal d'interruption ?*

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Prenons l'exemple de l'UART1, sachant que l'UART2 possède une stratégie de configuration et d'utilisation similaire. L'UART 1 possède 2 registres de configuration récepteur/transmetteur (RC1STA et TX1STA), 3 registres de configuration du débit à l'image de la configuration d'un Timer (BAUDCON1, SP1BRGH et SP1B1GL) et deux registres de travail pour la gestion des payload (TX1REG et RC1REG). Analysons une partie du registre 8bits TX1STA chargé de configurer le transmetteur. Nous pouvons observer sur le schéma ci-dessus que le champ TXEN (Transmitter Enable) permet d'appliquer la référence d'horloge au registre à décalage de transmission (TSR ou Transmit Shift Register) et autorise donc une transmission. Le champ SYNC (Synchronous) permet potentiellement d'utiliser une référence d'horloge externe, comme de sortir sur broche cette même référence et de transformer la communication asynchrone en communication synchrone.

Transmit Status and Control Register

Bit	7	6	5	4	3	2	1	0
	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
Access	R/W	R/W	R/W	R/W	R/W	R/W	RO	R/W
Reset	0	0	0	0	0	0	1	0



Le schéma bloc ci-dessus présente la structure du récepteur. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes : *registre interne à décalage de réception*, *registres FIFO (First In First Out) de travail pour la réception (pour lecture de la payload par l'application)*, *ensemble assurant la référence d'horloge (débit), broche Rx d'entrée* et *le signal d'interruption ?*

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Analysons une partie du registre 8bits TX1STA chargé de configurer le transmetteur. Nous pouvons observer sur le schéma ci-dessus que le champ SPEN (Serial Port Enable) permet de valider ou pas la connexion physique de la broche au registre à décalage de réception. Nous pouvons également constater que le récepteur est chargé de détecter les erreurs. Si une erreur de communication est détectée, alors l'un des champs FERR (Frame Error) et OERR (Overrun Error) est activé par le récepteur. Dans tous les cas, si une erreur de communication est détectée ou constatée dans l'application, la meilleure stratégie reste de demander un renvoi à l'émetteur.

Receive Status and Control Register

Bit	7	6	5	4	3	2	1	0
	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
Access	R/W	R/W	R/W	R/W	R/W	RO	R/HC	R/HC
Reset	0	0	0	0	0	0	0	0

Configuration sur MCU PIC18

Attention, le configuration présentée ci-dessous n'est pas celle demandée dans les TP. Elle sera donc à adapter. L'exemple suivant présente une séquence assembleur de code configurant le transmetteur de l'UART1 avec un débit de 115200Bd/s pour un horloge système de 64MHz. Une fois configuré, le caractère 'D' est transmis en respectant le protocole d'une liaison série asynchrone sur la broche TX1/RC6 du processeur. Le bit ou flag TRMT (TSR Register is Empty) est mis à jour par l'UART1 est permet de savoir si des bits restent présents dans le registre à décalage de transmission TSR. A notre époque, les ordinateurs modernes ne disposent plus de connecteur DB-9 pour liaison série. Nous utilisons généralement des modules passerelles USB vers UART assurant une bascule de protocole mais ne modifiant pas la donnée (bridge). Un bridge UART1 vers UART2 sera développé en TP. Sur le marché des modules ''USB to UART'', la société FTDI s'est spécialisée sur ce type de solution. Microchip propose également en catalogue des modules équivalents.

```
; uart1 enable, asynchronous mode
; 8bits data, no parity, high baudrate
MOVLW       0b00100100
MOVWF       TX1STA
; 16bits baudrate mode
; baudrate 115200KBd/s (64MHz CPU clock)
MOVLW       0b00001000
MOVWF       BAUDCON1
MOVLW       0x8B
MOVWF       SP1BRG
MOVLW       0x00
MOVWF       SP1BRGH
```

```
    ; send 'D' ASCII code ('D' = 68 = 0x44)
    MOLW      0x44
    MOWF      TX1REG
     ; wait the end of transmission
LF1 :
    BTFSS     TX1STA, TRMT
    GOTO      LF1
    NOP

; this program send 'D' caracter by UART1
```