Microprocesseur

TP2 : Timers / PWM

Cette séance permet de maitriser le temps de façon précise et de produire une sortie modulée en PWM (Pulse Width Modulation)

Les étapes de cette séance seront :
- utilisation du timer2 pour maitriser une durée d'attente
- utilisation du module de comparaison pour produire un signal modulé en PWM
- application simple du principe PWM à la LED tricolore

Note : Par abus de langage, nous appellerons *LED rouge*, la composante rouge de la LED tricolore, *LED verte*, la composante verte de la LED tricolore et *LED bleue*, la composante bleue de la LED tricolore.

1) Clignotement à 1Hz (heartbeat)

La première étape du TP est de repartir d'un projet vide pour faire clignoter la LED0 à la fréquence précise de 1 Hz. Cette fonctionnalité est appelée *heartbeat*, car elle permet d'avoir un bon indicateur qu'un programme tourne correctement. Il serait techniquement possible de faire plusieurs tentatives pour trouver empiriquement une valeur à passer à la fonction `wait()` du TP1 afin qu'elle dure approximativement 0.5 s. Nous verrons plus tard en quoi, cette approche est totalement inapropriée techniquement et méthodiquement.

Pour obtenir une mesure précise du temps qui s'écoule, nous allons utiliser un compteur matériel ou *timer*. Un *timer* fonctionne directement à partir de l'horloge et ne peut donc pas être perturbé/influencé par ce qu'il se passe au niveau du processeur, ce qui explique sa précision bien supérieure. Le chapitre 14 (p. 173) de la datasheet du microcontrôleur décrit le fonctionnement des timers 2 à 5 (un fonctionnement plus détaillé est expliqué dans la datasheet spécifique aux timers (le timer 1 est un timer de type A, les timers 2 à 5 sont des timers de type B). Nous allons utiliser le timer 2 en mode 16 bits.

Grâce à l'utilisation du timer, l'algorithme heartbeat devient :
- initialiser les entrées sorties (pour controler les LEDs)
- initialiser le TIMER2 à une période de 0,5 seconde
- dans une boucle infinie
  - allumer la LED0
  - attendre la fin de la période en cours
  - éteindre la LED0
  - attendre la fin de la période

La liste des registres (p. 175) montre que seuls trois registres sont impliqués dans l'utilisation de chaque timer : TxCON est explicité p. 176, TMRx est la valeur en cours du compteur et PRx est la valeur dite *de période*, lorsque TMRx atteind cette valeur, il est remis à zero au lieu d'être incrémenté, puis recommence à compter. Chaque timer utilise l'horloge des périphériques comme

base de temps. Dans notre cas, cette horloge est cadencée à 120 MHz.
Les fonctionnalités suivantes sont disponibles :

- Il est possible d'ajouter un circuit *prescaler*, qui divise la fréquence de comptage du timer.
- Pour *x* de 1 à 5, lorsque TMR*x* est remis à zero en fin de période, le bit T*x*IF du registre IFS0 passe à 1 (bit numéro *5x-1*), il est cependant possible d'écrire dans IFS0 pour forcer un retour de ce bit à 0. Cette fonctionnalité est très utile pour repérer simplement si la période en cours s'est terminée.

Pour utiliser le timer dans les conditions voulues, il faut également définir les réglages suivants :

- l'horloge source à utiliser est l'horloge des périphériques internes.
- Le mode *Gated Timer* doit être désactivé. Ce mode permet d'inhiber le comptage en fonction d'une broche externe, ce n'est pas le comportement recherché.

Créez un nouveau projet (vous aurez toujours besoin d'utiliser config_bits.h), et écrivez un programme produisant l'effet heartbeat. L'intérêt est d'obtenir un clignotement à 10 Hz garanti par conception.

Modifiez ce programme pour obtenir le clignottement de la LED à 1 Hz.

De l'intérêt de l'usage d'un timer :
A supposer qu'il soit possible de garantir l'exactitude de la durée d'une fonction `wait()` telle que donnée au TP1, l'usage d'une telle fonction n'est pas souhaitable car le temps de manipulation des LEDs s'ajoute au temps d'attente de la fonction. La fréquence n'est donc ni précise, ni stable (selon les opérations à effectuer pour manipuler les LEDs). Le timer étant indépendant, en calant les évènements sur sa fin de période, on obtient une fréquence finale aussi précise que le permet l'oscillateur de la carte, quelles que soient les opérations effectuées entre les phases d'attente. Le temps de réaction du processeur (variable) ne se traduira que par un léger bruit de phase.

2) Output compare

Il est possible d'adjoindre des modules de comparaison aux timers 2 et 3. Ces modules donnent la possibilité de générer des evènements en fonction de valeurs intermédiaires du *timer*. Ces evènements vont de la mise à '1' d'un bit, au changement d'état d'une sortie. Le PIC32MX370 contient cinq modules de comparaison.

L'étape suivante est de produire un flash sur la LED rouge lorsque le timer 2 commence ou termine sa période (selon votre choix), mais de le faire sans solliciter le processeur. Le chapitre décrivant les ports du microcontrôleur indique que certaines broches peuvent être directement contrôlées par les modules matériels présents sur microcontrôleur (paragraphe 12.3.5, p. 144). Les pages 145 et 146 donnent :

- La liste de ces broches (RPxx, où xx désigne le port et le bit concerné),
- Le registre permettant de définir quelle fonction attribuer à chaque broche (RpxxR),
- Pour chaque groupe de broches, quelles valeurs codent chaque fonction possible.

La LED rouge est sur le port D2, la LED verte sur le port D12 et la LED bleue sur le port D3. Un esprit aiguisé remarquera qu'il est possible d'attribuer le module *Output Compare 3* (OC3) sur D2, *Output Compare 5* (OC5) sur D12 et *Output Compare 4* (OC4) sur D3 (ça alors ! Quelle coincidence !).

L'usage du module *Output Compare* est précisé dans le chapitre 17 (p. 187) de façon succincte mais suffisante. Comme d'habitude, plus de détails sont disponibles dans la datasheet dédiée. Le mode qui nout intéresse est le mode qui produit des pulses en continu après initialisation. Dans ce mode, la sortie passe à '1' quand le timer vaut OCxR et retourne à '0' quand le timer vaut OCxRS.

Modifiez votre programme pour produire un flash rouge à chaque fin de période du timer 2 (éclairage bref). Pour cela, il faut :

- préciser que le module Output compare 3 doit sortir sur la LED rouge
- configurer quelles valeurs du timer 2 vont provoquer l'allumage ou l'extinction de la LED
- configurer le fonctionnement du module Output Compare lui-même.

3) La PWM sur les LEDs

Si vous accélérez le timer 2 en ne jouant que sur le prescaler, vous remarquerez que les LEDs apparaissent comme fixes, mais de luminosité plus faible que la normale. Ce comportement est normal : l'oeil effectue un filtrage passe-bas et ne perçoit donc que la luminosité moyenne émise par la LED.

En jouant sur les seuils de comparaison, il est donc possible de produire un signal modulé en PWM. Modifiez votre programme pour que :

- le clignotement de la LED rouge soit imperceptible (fréquence > 100Hz)
- la luminosité apparente de la LED rouge augmente puis baisse lentement (2 à 3 secondes pour passer du minimum au maximum)

Les LEDs RGB sont très puissantes, évitez de les allumer à plus de 30 ou 50 % de leur puissance pour la longévité de vos rétines...
A ce stade, vous pouvez simplement retirer le code relatif au fonctionnement de la LED0.

Feu d'artifice final :
En combinant les trois couleurs de LED, produisez l'animation suivante :

1. au départ, le rouge est allumé
2. le vert s'allume progressivement
3. le rouge s'éteint progressivement
4. le bleu s'allume progressivement
5. le vert s'éteint progressivement
6. le rouge s'allume progressivement
7. le bleu s'éteint progressivement
8. retour à l'étape 2

Si en plus la variation progressive des intensités lumineuses est calée temporellement sur un autre timer, le code atteint un certain niveau de classe que seul le TP3 pourra dépasser...

De l'inutilité de la fonction PWM pour générer un *heartbeat* :
A méditer si vous avez le temps, il est techniquement possible d'utiliser une fonction PWM (timer + output compare) pour produire un heartbeat sans solliciter le processeur (et donc en économisant des ressources de calcul). Mais cette fonction devient alors inutile, car sont intérêt est précisément de témoigner du bon fonctionnement du programme, et donc de la disponibilité du processeur pour le clignotement.