1) *Comment sait-on le bit sur lequel il faut écrire/tester?. Par exemple, pour savoir si le bouton poussoir est allumé, on teste le bit 1 tandis que pour allumer la Led rouge, on effectue une opération Set bit sur le bit 0. Est-ce que ce genre de définitions serait fourni dans un examen ?*

#define Pous1On (!(P2IN&(1<<1)))
#define Pous2On (!(P1IN&(1<<1)))
#define LedRougeOn P1OUT |= (1<<0)
#define LedRougeOff P1OUT &=~(1<<0)

Réponse : C'est le liaisons électriques qui déterminent les bits à utiliser. Dans notre cas, le chemin est un peu tortueux :

- Un ingénieur de chez Texas Instrument a décidé quelle broches du microcontrôleur seraient reliées sur le connecteur du LauchPad (les deux doubles rangées de broches dorées). L'information se trouve dans les schémas de la carte, disponible sur leur site.

- J'ai réalisé quelques années un raccord entre un ancien LauchPad MSP430G et j'ai noté sur les petits bois qui tiennent les prises Logidules. Malheureusement, le nom des pin n'est pas compatible sur la version MSP430F5529 utilisée cette année. J'ai du faire la translation.

- Des fils Logidules relient ces fiches aux fiches correspondantes du moteur Logidule.

- Non, une telle question ne sera pas posée dans un examen !

2) *Et que veux dire «Toggle» ? et l'opérateur ^= ? Cette définition n'est pas utilisée dans le code, est-ce normal ?*

#define LedRougeToggle P1OUT ^= (1<<0)

Réponse : Le mot Toggle veut dire en anglais basculer. Il est utilisé pour indiquer un changement d'une valeur : si elle est à 0, elle passe à 1, si elle est à 1, elle passe à 0.

L'opérateur logique OU-exclusif permet de réaliser cette fonction. Souvenez-vous de sa table de vérité :

A | B | A xor B
0 | 0 | 0
0 | 1 | 1
1 | 0 | 1
1 | 1 | 0

En C, le symbole ^ correspond au OU-exclusif, en mode « bit-wise », donc sur un champ de bits (comme les opérateurs &, | et ~).
Si on souhaite, par exemple, changer l'état de la LED sur P1.0 sans toucher les autres bits de P1, l'opération suivante convient :

P1OUT ^= (1<<0);

En effet, le résultat pour le bit 0 sortira des deux dernières lignes de la table de vérité (A = 1) : si B vaut 1, il passera à 0, s'il vut 0, il passera à 1.

Mais les autres bits ne seront par changés, comme le montre les deux premières lignes de la table de vérité (A = 0) : si B vaut 0, il restera à 0, s'il vaut 1, il restera à 1.

J'ai placé cette définition pour qu'elle soit à disposition, même si elle n'est pas utilisée dans chaque pogramme.

3) *Dans le programme test, je ne comprends pas la fonction AttenteMs().*
Qu'est-ce qui va définir une durée de temps dans ce cas? D'où vient la valeur de 105 pour i ?

Réponse : Vous trouverez la réponse dans les notes de cours « Gestion du temps, sorties ». Cette valeur est obtenue par calibrage : on essaie une valeur, on produit un délai par exemple de 10 secondes, on ajuste ensuite la valeur. Notez que cette solution semble un peu bricolée. On en

voit aussi les limites pour des attentes courtes (voir questions suivante). On est donc bientôt prêt à aborder le bonne solution pour gérer le temps : les Timers des microcontrôleurs.

4) *Dans votre prémière vidéo "PWM: Modulation de largeur d'impulsion" vous utilisez une fonction "delayMicrosecond". Est-celle l'équivalent de AttenteMs() ?*

Réponse : Pas vraiment... Il ne faut pas confondre les millisecondes et les microsecondes !

Pour attendre 39 us, vous pouvez vous inspirer de la procédure AttenteMs(), dont la boucle **for(i=105;i>0;i--){}** attend environ 1 ms = 1000 us.
Avec une règle de 4, vous pouvez calculer qu'il faut mettre le nombre d'itération à 4 pour avoir 39 us.

5) *Concernant l'exercice du va-et-vient de la semaine 4, vous donnez les définitions pour le moteur sur la carte rouge. Or possédant la carte blanche, rien ne semble fonctionner !*

Réponse : En effet. Vous ne disposez pas de moteur Logidule chez vous, il vous sera donc bien difficile de test mon programme ! Par contre, vous pouvez essayer de le comprendre.

C'est la raison pour la quelle j'ai noté « TP des années précédentes ». Je prépare une vidéo explicative et une démonstration pour diminuer un peu votre frustration !

6) *Aucune réaction se produit sur le microcontrôleur après avoir cliqué sur le marteau, par exemple en lançant votre programme de test. (Pour ma part en tout cas).*

Réponse : Juste ! Il ne suffit pas en effet ce compiler le programme (cliquer sur le marteau). Il faut encore transférer le binaire produit dans le microcontrôleur.

Deux manière de procéder :

- lancer le Debug (l'icône avec un petit insecte). Après le chargement, le curseur de place à la première ligne du programme. Il faut alors lancer le programme en continu (icône flèche verte), ou exécuter du pas à pas dans le programme.
- lancer le changement su binaire (icône Flash, avec deux accolades). Le binaire est simplement transféré dans la mémoire Flash et le programme est lancé, sans possibilité de l'interrompre pour le Debug.

Une vidéo montrant ces opérations est maintenant disponible sur Moodle.

7) *J'ai regardé votre vidéo explicative du code sur les encodeurs à Led. Mais je n'ai pas compris la ligne malgré vos explications : #define encodeurX (P1IN&(1<<7))>>7*

Pouvez vous expliquer a quoi correspond l'expression ...>>7 ?

Réponse : encodeurX P1IN&(1<<7) rend soit la valeur binaire 00000000, soit la valeur 10000000.

encodeurY P2IN&(1<<2)) rend soit la valeur binaire 00000000, soit la valeur 00000100.

C'est gênant pour notre algorithme : en comparant X et ancienY par : if (X == ancienY) on va parfois comparer 1000000 et 00000100, qui sont différents. Or ils correspondent à X qui est "vrai" et ancienY qui est aussi "vrai".

L'opérateur >> est le décalage à droite en C. On a souvent utilisé son frère-jumeau << qui décale vers la gauche.

encodeurX (P1IN&(1<<7))>>7 rend soit la valeur binaire 00000000, soit la valeur 00000001.

encodeurY (P2IN&(1<<2))>>2 rend soit la valeur binaire 00000000, soit la valeur 00000001.

On peut alors comparer sans erreur X et ancienY.