Lab 3
Arithmétique à virgule fixe / flottante

M. Briday

October 12, 2020

1 Objectif

L'objectif de ce TP est de comparer les performances de calcul d'un microcontrôleur lors de l'utilisation de nombres non entiers.

La première partie consiste à mesurer le temps nécessaire pour effectuer une simple addition et une multiplication, en utilisant des nombres entiers et des nombres flottants.

Dans un deuxième temps, des opérations utilisant le calcul en virgule fixe seront effectuées.

Dans une troisième étape, une table de trigonométrie sera mise en œuvre pour les calculs à virgule fixe. Les performances seront comparées à une solution à virgule flottante, tant en termes de vitesse que de précision.

2 Temps d'exécution des opérations élémentaires

Un *timer* est configuré pour fonctionner comme un chronomètre (32 bits). 3 fonctions sont fournies (fichiers `sw.h/c`):

```
/* Init stopwatch. Should be called first (during setup)
 * Works at CPU clock speed (64MHz) to count instruction cycles
 * */
void SWInit();
/* Reset stopwatch. It will restart counter from 0 */
void SWReset();
/* Get the 32 bit value of the stopwatch
 * Expire in ~67s
 * */
uint32_t SWGet();
```

Pour mesurer le nombre de cycles nécessaires pour exécuter une opération[1], on va en fait mesurer le temps pour en effectuer 1000 pour avoir des résultats plus facilement exploitables.

L'affichage des résultats utilisera l'afficheur de la carte[2].

Question 1 *Mesurer le temps nécessaire pour[3] :*

- *faire 1000 tours "à vide" (i.e. boucle vide);*
- *faire 1000 tours avec 1 addition. En déduire le nombre de cycle pour effectuer une addition entière.*
- *de la même manière, mesurer le temps pour faire une multiplication entière, une addition et une multiplication avec des* `float`*, une racine carrée[4] et un cosinus.*

Dans cette première partie, la FPU (*Floating Point Unit*) n'était pas déclarée au niveau du compilateur (et donc le compilateur utilise une bibliothèque logicielle d'émulation). Pour activer la FPU, il faut modifier le fichier `stm32f303.cmake` (accessible dans `CMake Modules...` dans le gestionnaire de projets de QtCreator), dans la définition des options du CPU (inverser les commentaires des lignes 2 et 3).

Question 2 *Refaire les mesures avec la FPU. Comparer les résultats.*

3 Utilisation de nombres à virgule fixe

Rappel: si `val` est un nombre sur 16 bits, codé en virgule fixe (10.6), celà signifie qu'il y a 10 bits pour la partie entière, et 6 bits pour la partie fractionnaire.

Pour retrouver la valeur réelle, il suffit alors de multiplier par 2^{-6}. Par exemple:

```
int16_t val = 0xabcd; //43981
```

Ici, la valeur réelle est: $43981 \times 2^{-6} = 687.203125$

Lors d'une multiplication, il sera nécessaire de faire une opération supplémentaire car l'exposant est modifié. Il faudra faire un décalage supplémentaire du résultat. De plus, pour garder une valeur correcte, il faudra réaliser l'opération sur une taille plus importante (le résultat d'un produit de 16bits x 16bits est sur 32 bits).

[1] cette approche est possible sur un micro-contrôleur, mais peut devenir rapidement très complexe pour des architectures complexes avec des pipelines profonds.

[2] Documentation sur `https://gitlab.univ-nantes.fr/briday-m/itii-mac-tp-etu/-/blob/master/doc/tft.md`

[3] il sera indispensable d'utiliser des variables déclarées `volatile`

[4] requiert l'inclusion de `math.h` en début de fichier

Question 3 *Proposer une fonction qui fait l'affichage (sur le tft) d'une valeur en virgule fixe (10.6). Cette fonction ne sera utilisée que pour de la validation et il est possible de transformer la valeur en* `float` *(par contre, pas de fonctions de* `math.h`*!!!).*

Question 4 *Réaliser l'addition de 2 nombres (10.6), tester avec des opérandes positifs et négatifs.*

Question 5 *Réaliser la multiplication de 2 nombres (10.6), tester avec des opérandes positifs et négatifs. Mesurer le temps nécessaire avec la même approche que la méthode précedente*[5][6]*.*

4 Table de sinus/cosinus

Il est quelquefois nécessaire de réaliser des opérations très rapidement et on utilise pour celà des tables pré-calculées à la place de l'évaluation d'une formule. C'est notamment le cas des tables de sinus/cosinus pour la commande des moteurs sans balais (*brushless*) par commande à champ tournant (*Field Oriented Control*). Dans ce cas, il y a besoin de faire un changement de repère par rapport à un repère tournant *via* une matrice de rotation. Ces opérations sont réalisées très souvent (fréquence en dizaines de KHz).

Question 6 *Proposer une table de sinus pré-calculée de 256 valeurs sur* $[0, \frac{\pi}{2}[$*. Les valeurs seront codées sur (1.15), en complément à 2. On pourra utiliser le script python fourni pour la génération du code.*

- *quelle est la plage utilisable sur cet intervalle?*
- *Quelle sera la précision angulaire?*

À partir de la table de sinus sur $[0, \frac{\pi}{2}[$, on peut déduire une fonction qui renvoie le sinus de n'importe quel angle. On considère que le paramètre d'entrée `step` correspond à un angle de $\frac{\pi}{512}$, soit 1024 valeurs par tour. Il suffit alors de déterminer dans quel *cadrant* (intervalle $[n \times \frac{\pi}{2}, (n+1) \times \frac{\pi}{2}[$) se situe l'angle et d'utiliser les propriétés de symétrie de la fonction sinus.

Question 7 *Implémenter la fonction* sinus *qui utilise le tableau pré-calculé:*

```
/* angle direcly in steps (1024 steps/round) */
int16_t sin(const int step);
```

Question 8 *Implémenter la fonction cosinus qui utilise le même tableau pré-calculé*[7]*:*

```
/* angle direcly in steps (1024 steps/round) */
int16_t cos(const int step);
```

[5] L'opération de multiplication sera réalisée sur 32 bits pour conserver la précision maximale.

[6] Attention au décalage à droite qui n'a pas le même comportement avec une valeur signée et une valeur non signée

[7] *cos* est la dérivée de *sin*: $sin(\theta + 90) = cos(\theta)$

On chercher maintenant à évaluer le temps mis dans un cas classique, l'évaluation de $V_c = 20.cos(\theta)$ et $V_s = 20.sin(\theta)$, pour θ de 0 (0 rad) à 1023 (2π rad).

Question 9 *Déterminer le type de V_c et V_s (en utilisant un résultat sur 16 bits, en complément à 2). On cherche maintenant à évaluer les performances:*

- *Combien de temps est-il nécessaire pour faire les 1024 calculs de V_c et V_s?*
- *Combien de temps est-il nécessaire avec un calcul à virgule flottante?*
- *Quelle est l'erreur maximale de calcul entre les versions à virgule fixe/flottante?*