Tâche 5 – Configuration du Timing

- Question 1 :
  - La conversion du signal analogue doit être exécutée par un timer hardware
- Question 2 :
  - Pour exécuter le rafraichissement de l'écran le XF est suffisant. Mais le RTOS permettrait de moins ralentir le reste du code
- Question 3 :
  - Le timer hardware est priorisé car il s'occupe de tâches qu'on souhaite le plus « synchrone » possible. Sinon on utiliserait des timeout

Tâche 7 – Configuration de l'ADC

- Question 1 :
  - La fréquence de sampling doit être au minimum le double de celle du signal. Donc minimum 2[kHz]
- Question 2 :
  - Oui, un filtre passe-base de 1[kHz]
- Question 3 :
  - C'est la fréquence de Nyquist, pour éjecter les fréquences « mirroir » dues à la convolution
- Question 4 :
  - Le signal IN0 doit être utilisé pour mesurer sur PA0
- Question 5 :
  - Oui en mode continuous. Mais pour être vraiment régulier, il faudrait utiliser un timer hardware pour activer la conversion.

Tâche 13 – Sample-Rate Tuning

- Question 1 :
  - 200[kHz]
- Question 2 :
  - L'ADC limite le système, on ne peut pas échantillonner plus vite que l'ADC arrive à convertir les données.

Tâche 14 – RTOS Integration

- Question 1 :
  - Pour garder une réaction fluide du TouchScreen, 130[kHz]
- Question 2 :
  - FreeRTOS permet d'améliorer la fluidité du UI du TouchScreen, car il sépare les tâches.
- Question 3 :
  - N'importe quel programme multi-tâches comprenant des méthodes bloquantes.

FA1 : Fonction Trigger

Deux méthodes ont été testée pour le Trigger :

- La première dans l'interruption, cela fonctionnait mais ralentissait visiblement l'exécution. De plus le temps d'interruption long occasionner des crashs aléatoirement durant l'exécution du programme
- La deuxième dans la méthode d'affiche du *OscilloscopeController*, cependant je n'ai jamais réussi à faire fonctionner celle-ci.

Problèmes rencontrés

Au départ, je n'arrivais qu'à sampler à 10[kHz] dans un tableau de 256 valeurs.
Ma méthode de trigger était la suivante :

```
    if(triggerOn()) {
        uint32_t shift = 0;
        for(int i = 0; i < ADC_VALUES_BUFFER_SIZE; i++) {
            if(_adcValuesBuffer[i] >= 2000 && _adcValuesBuffer[i] <= 2100) {
// Around than 2^11, which is ~middle
                shift = i;
                break;
            }
        }
        gui().drawGraphPoints(_adcValuesBuffer+shift, ADC_VALUES_BUFFER_SIZE-
shift);
    } else {
        gui().drawGraphPoints(_adcValuesBuffer, ADC_VALUES_BUFFER_SIZE);
    }
```

Lorsqu'on « trig », on va donc changer le pointeur de début du tableau, et diminuer sa taille pour la méthode drawGraphPoints().
Cependant, l'appel de cette méthode réalise en réalité 3 choses :

1. Clear du scope
2. Redimensionnement du scope selon le nombre de points à afficher
3. Affichage des points

Le point 2 engendrait donc un redimensionnement. Ce qui créé un affichage « tremblant » lorsque je triggait.

En souhaitant augmenter la taille de mon tableau de sample, j'ai fait face à de grands problèmes de ralentissement. J'ai essayé d'optimiser plusieurs méthodes, de vérifier que la version de *TouchGFX* était la bonne, etc…
En réalité je n'avais pas activé les *Data and Instruction Caches*.

Une fois activés, j'ai pu passer à fsamp=100[kHz] et un tableau de 8000 valeurs

L'implémentation du trigger a été supprimée de mon code actuel, causant trop de bug d'affichage. Cette fonction n'est donc pas implémentée.

FA2 : Display – Axe du temps

Pour implémenter l'axe du temps deux choses ont été ajoutées :

Un grand switch case :

```
    float scale = 1.0;
    switch(_tdivValue) {
    case TDIV_500us:
        scale = 20.0;
        break;
    case TDIV_1ms:
        scale = 10.0;
        break;
    case TDIV_2ms:
        scale = 4.0;
        break;
    case TDIV_5ms:
        scale = 2.0;
        break;
    case TDIV_10ms:
        scale = 1.0;
        break;
    default:
        scale = 1.0;
        break;
    }
```

Et une modification de la méthode setGraphPoints de Model.cpp :

```
void Model::setGraphPoints(uint16_t * values, uint16_t count, float
xScale){
    this->values=values;
    this->count=count/xScale;
#if (TOUCHGFX_BAREMETAL != 0)
    this->flagGraph=true;
#endif // TOUCHGFX_BAREMETAL
#if (TOUCHGFX_FREERTOS != 0)
    GUI_EVENT msg = GRAPH_EVENT;
    if(uxQueueMessagesWaiting(gui_msg_q)==0){
        xQueueSend(gui_msg_q, &msg, 0);
    }
#endif // TOUCHGFX_FREERTOS
}
```

L'ajout de count=count/xScale ; et l'ajout d'un switch case pour le scale permet d'adapter l'affichage rapidement en jouant sur le fait que la méthode d'affichage prends toujours au maximum 400 points du tableau, qu'elle espace équitablement. (par exemple : un tableau de 8000 valeur verra 1 valeur sur 20 être affichée).

Notre tableau faisant 8000 et notre fsamp = 100[kHz], cela veut dire que pour afficher 500[us/div] avec 8[div] ou 4[ms] il faut afficher les 400 premières cases du tableau.
Donc si on divise count par 20 -> 8000/20 = 400 -> La scale sera de 500[us/div]

FA3 : Enregistrer les valeurs avec la DMA

Par manque de temps cette option n'a pas été explorée.

Conclusion

Toutes les tâches ont été implémentées avec succès.
Les tâches optionnelles FA1 et FA3 ne sont pas implémentées.

Sion, le 24.01.2022 Samy Francelet