Projet Calculette

1. Introduction

Le projet proposé a pour but de porter une calculatrice vectorielle à la mode xxxLab fonctionnant sur PC sur un microcontrôleur NXP LPC55S69 pour en faire une calculatrice autonome. L'environnement de développement MCUXPressoIDE développé par NXP sera utilisé pour le projet. Des ressources sont disponibles sur gitlab à l'adresse : **https://git.enib.fr/bouchare/proj_cm33** .

Sur PC, le programme imlémente un environnement permettant d'entrer des calculs (réels ou comples, scalaire ou matriciels) à la main sur une ligne de commande, ou de les stocker dans un fichier et de charger le fichier pour exécuter les séquences de calcul. Le programme fournit également une interface de tracé de courbes. Finalement il fournit la possibilité de stocker des variables, ainsi qu'un langage de programmation simple pour réaliser des algorithmes de calcul numériques.

Porter le programme sur microcontrôleur implique :

– recréer l'interface de commande en ligne pour éditer des calculs à la main : on propose d'utiliser la liaison série qui passe par le câble USB (partiellement réalisé).

– gérer le stockage de fichiers de calculs sur un périphérique de stockage : la carte propose un slot pour une carte micro-sd.

– gérer l'affichage des courbes sur un écran : la carte dispose d'un bornier arduino sur lequel on peut venir connecter un afficheur LCD 240x320 pixels, avec la couleur codée sur 16 bits.

– étudier la possibilité d'accélérer certaines fonctionnalités grâce au DSP intégré au microcontrôleur.

Le microcontrôleur est construit autour de deux cœurs à architecture de Harvard Cortex-M33 à 150 MHz, 640 kB de mémoire Flash, et 320 kB de RAM. Les deux cœurs ne sont pas tout à fait symétriques. Le premier (Core 0) dispose d'une unité de calcul en virgule flottante (FPU) et d'un coprocesseur (DSP PowerQuad) capable de réaliser des calculs accélérés par du matériel (fonctions mathématiques standard, FFT, filtrage/convolution, …). On envisage d'éxécuter la calculatrice sur ce cœur. Le second (Core 1) ne peut faire que du calcul entier. On envisage de l'associer à l'écran LCD (communication SPI) et de l'utiliser pour exécuter les routines de tracés

basiques (ligne, rectangle, texte).

La carte de développement dispose également d'un accéléromètre qui pourra être utilisé pour faire basculer l'affichage des graphiques lorsque la carte est tournée.

Le programme utilise une pile en RAM pour exécuter les calculs, stocker les variables et les fonctions. Pour permettre une utilisabilité maximale de la calculatrice, on a intérêt à maximiser la taille de la pile en RAM. Le développement effectué doit essayer d'utiliser aussi peu de RAM que possible pour maximiser la taille de la pile de calcul. Par ailleurs, la FPU et le DSP PowerQuad sont optimisés pour la virgule flottante en simple précision. Tous les nombres réels utiliseront le format simple précision (au lieu d'un format double précision pour le programme PC). Cela a deux conséquences :

– on divise par 2 la taille nécessaire pour stocker un réel sur la pile de calcul (par rapport à la double précision), on peut donc en mettre 2 fois plus dans une quantité de mémoire donnée (good!).

– on perd en précision sur les valeurs : 7 décimales de précision contre 15 décimales pour le format double précision (pas good …).

La figure 1 montre l'architecture du microcontrôleur, avec

– les deux cœurs CPU0 et CPU1, la mémoire Flash et RAM (répartie en plusieurs banques SRAMX, SRAM0 à SRAM4 et USB SRAM),

– les différentes interconnexions possibles entre les éléments qui peuvent être maître du bus AHB (Multilayer AHB Matrix), ceux en haut de la figure (les CPUs, contrôleurs DMA), et les éléments esclaves sur le bus (mémoires, GPIO, interfaces série Flexcomm – RS232, I2C, SPI),

– les deux bus périphériques APB0 et APB1 (en bas) aves les coupleurs périphériques rattachés.

Le mapping mémoire adopté est le suivant :

start address	end address	Memory
`0x00000000`	`0x0009FFFF`	Flash (640kB)
`0x04000000`	`0x04007FFF`	SRAMX (32kB), CPU1 code (not used)
`0x20000000`	`0x20017FFF`	CPU0 SRAM (96kB), data, stack and heap
`0x20018000`	`0x2002FFFF`	calc SRAM stack (96kB)
`0x20030000`	`0x2003BFFF`	CPU1 : code+data+stack (48kB)
`0x2003C000`	`0x2003FFFF`	shmem shared memory buffer between CPU0 and CPU1 (16kB)
`0x20040000`	`0x20043FFF`	PowerQuad SRAM (16kB)
`0x40100000`	`0x40103FFF`	USB SRAM (16kB)

Le projet est géré (génération du makefile, du script de l'éditeur de lien, des fichiers à compiler) par l'IDE MCUXpresso de NXP (un clône d'Eclipse). Le code de la calculatrice en elle-même se trouve dans le répertoire `source`. Ce code est portable (et donc pas à modifier!) à l'exception du fichier `source/sysdep.c` qui contient les fonctions de bas niveau permettant d'intégrer la calculatrice dans des environnements différents.

Le code «sysdep» en l'état gère une interface basique permettant l'entrée de commandes à la main, et l'affichage des résultats via une console série (`COMxx` sous windows et `/dev/ttyACM0` sous Linux, 115200 bauds, pas de parité, 1 bit de stop) qui fonctionne avec les interruptions + des buffers circulaires en émission et réception.

Le répertoire du projet est organisé de la manière suivante :

```
/répertoire racine du projet/
 +- board/                          : config du microcontrôleur effectuée avant l'entrée
 |                                    dans le «main»
```

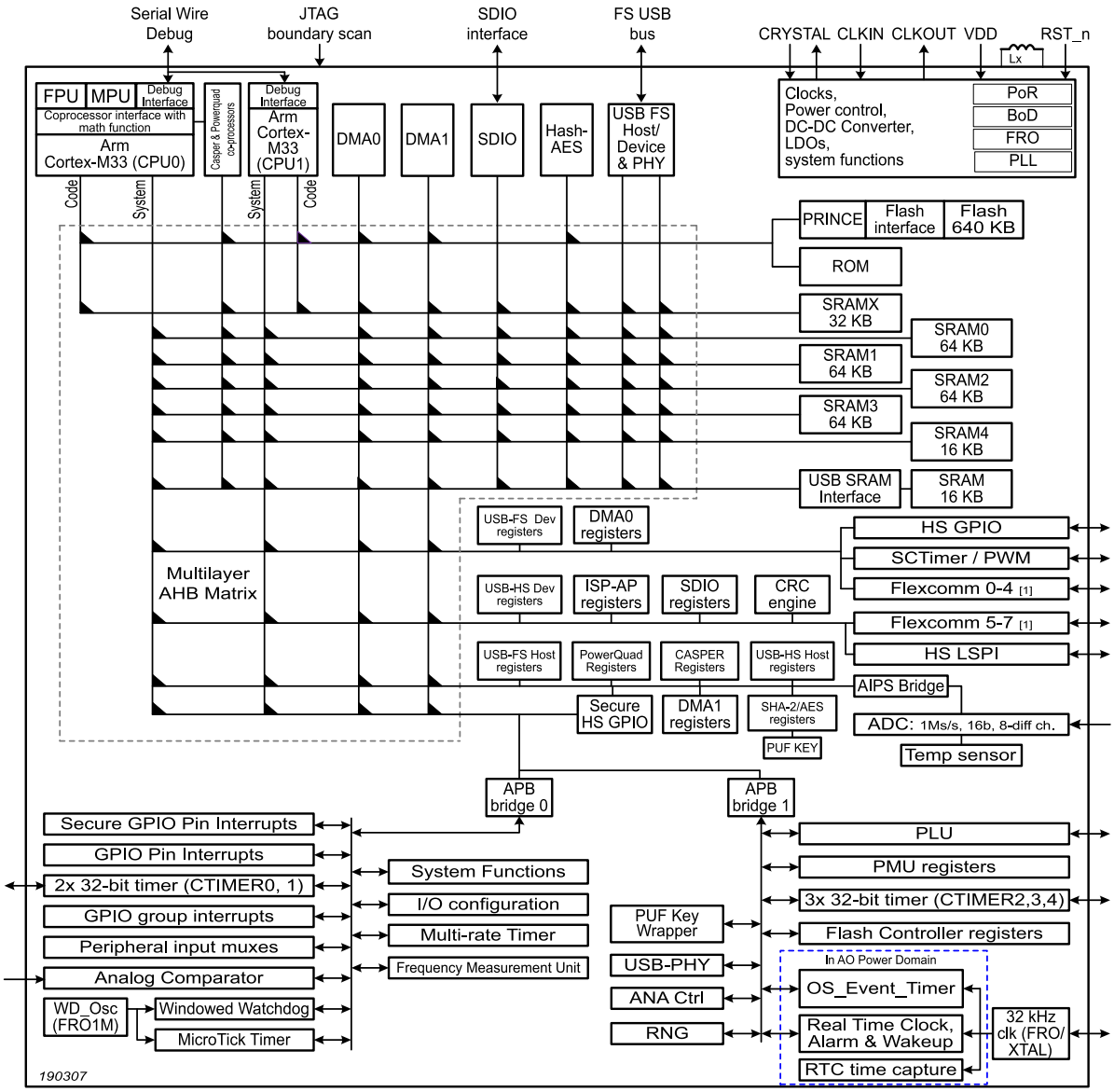

Figure 1. Synoptique microcontrôleur LPC55S69

```
+- CMSIS/                          : couche d'abstraction du processeur [source : ARM]
+- device/
|  +- LPC55S69_cm33_core0.h        : définition des IRQn, structures des périphériques
+- drivers/                        : drivers de bas niveau du microcontrôleur
+- lcd/
|  +- *.[ch]                       : driver de l'écran lcd/touchscreen
|  +- lcd_private.[ch]             : driver communication SPI
+- source/
|  +- *.[ch]                       : bibliothèque d'appels systèmes
|  +- sysdep.c                     : code applicatif
+- startup/
|  +- startup_lpc55s69_cm33_core0.c : code de startup, table des vecteurs
+- fichier mex                     : fichier de config des horloges et des broches
```

2. Organisation du projet et travail attendu

Le projet est découpé en tâches qui sont détaillées dans les paragraphe suivants. Chaque étudiant du groupe (3 étudiants) doit se saisir de la tâche qui lui a été attribuée et la développer de manière globalement autonome. Vers la fin du projet, l'ensemble du travail effectué doit être mis en commun (intégration). A l'occasion de cette mise en commun, chaque étudiant doit pouvoir expliquer aux autres membres du groupe les éléments importants de ce qu'il a développé de manière à ce que chacun ait une vue globale et puisse répondre aux questions de l'enseignant lors de la démonstration finale du projet devant celui-ci.

semaine	travail étudiant 1	travail étudiant 2	travail étudiant 3
1	prise en main MCUXpresso + calculatrice		
2 à 5	Gestion des séquences VT100 Initialisation et gestion de la carte SD et du système de fichiers FAT Lien des fonctions de bas niveau de la libc avec les fonctions du système de fichiers	Démarrage du second cœur et test d'un dispositif de communication et synchronisation entre les deux cœurs. Mise en place d'une interface de communication synchrone, puis asynchrone permettant l'exécution de fonctions sur l'autre cœur.	Mise en œuvre de l'accéléromètre, et intégration dans une fonction de la calculette Gestion du temps Utilisation de PowerQuad : calcul du cos d'un vecteur Utilisation de PowerQuad : FFT
6	Mise en commun + démonstration		

3. Tests préliminaires (semaine 1)

Pour démarrer

- Importer le projet (Quick Panel > Import from filesystem), puis aller chercher l'**archive** zip. Le projet doit apparaître dans l'onglet Project Explorer.
- Compiler (Quick Panel > Build) et télécharger le projet sur la carte (Quick Pane > Debug). Le système commence par détecter la sonde programmation intégrée à la carte, puis le cœur du processeur (Core 0) sur lequel le code va s'exécuter. Valider les choix par défaut. Le programme s'arrête à l'entrée de la fonction `main`.
- La calculatrice est démarrée par la fonction `mainloop` appelée à la fin de la fonction `main`, en bas du fichier `sysdep.c`. Cette fonction appelle indéfiniment la fonction `parse`.

  `parse` permet de :

  – récupérer une ligne à évaluer, que ce soit en provenance d'un fichier ou de l'entrée clavier standard via les fonctions `lineinput` et `edit` du fichier `source/edit.c` et finalement de la fonction `sys_wait_key` du fichier `source/sysdep.c`.

  – d'évaluer la ligne soit en exécutant une commande ou un mot clé du langage de programmation, soit en évaluant une expression via la fonction `parse_expr`. Cette dernière fonction est capable d'évaluer une expression en décomposant la ligne en «token» (nombre, opérateur, …) qui sont ensuite combinés conformément à une table de précédence qui définit les priorités entre opérateurs.

  – d'afficher un résultat via les fonctions `output*` et finalement la fonction `sys_print` de `source/sysdep.c`.

  Mettre des points d'arrêts à l'entrée des fonctions `parse` et `parse_expr`. Lancer le code (touche du clavier F8). Le programme doit s'arrêter au début de `parse`. Continuer (F8).

  Entrer un calcul simple : `2+3` puis valider. Le programme doit s'arrêter au début de `parse_expr`. Aller en pas à pas (F6). La fonction `scan` est chargée d'analyser la ligne et de fournir le prochain token rencontré. Ici, c'est un `T_REAL`. On peut vérifier la valeur détectée dans `cc->val` en plaçant le curseur de la souris dessus (on doit voir la valeur '2'). Continuer en pas à pas (F6). On récupère ensuite l'opérateur `T_PLUS`, puis

la deuxième valeur (T_REAL, avec la valeur '3'). L'opération est ensuite évaluée. Continuer (F8). Le résultat s'affiche dans la console série.

Si le programme est mis en pause, on se rend compte qu'à chaque fois, on est dans la fonction `uart_getc` qui attend un nouveau caractère en provenance du terminal série. En fait le programme passe la plus grande partie de son temps à attendre …

- Le programme permet de définir des variables représentant des scalaires ou des vecteurs réels ou complexes : essayer les commandes de l'exemple 1.

Exemple 1

```
a=2
b=2*a+1
a=1:4
b=2*a+1
z=4+3i
abs(z)
arg(z)
N=8;z=exp(1i*2*pi*(0:N-1)/N)
```

et des fonctions : essayer les commandes de l'exemple 2.

Exemple 2

```
function f(x)
  return 2*x+1
endfunction
f(3)
f(1:5)
```

- Arrêter le debug. L'environnement MCUXpresso offre des outils graphiques pour
  - ajouter des bibliothèques : drivers de carte SD, Codec, OS temps réel, … Sélectionner le nom du projet dans le «Project explorer» puis cliquer sur l'icone  . Il est alors possible de sélectionner les éléments à ajouter.
  - définir les horloges du systèmes : sélectionner le nom du projet dans le «project explorer» et utiliser l'icone  pour accéder au menu OpenClocks qui permet d'afficher l'arbre de l'horloge, faisant figurer les deux PLL, ainsi que les horloges des périphériques

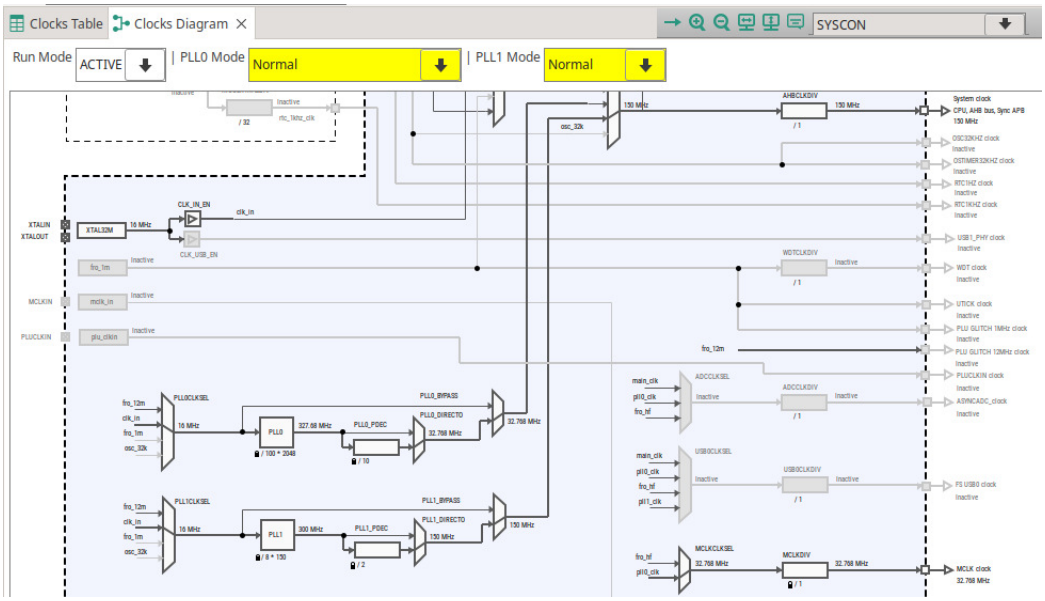

Figure 2. Configuration des horloges

La sortie du mode se fait par le bouton Update Code qui réalise la génération du code reflétant la configuration graphique dans `board/clock_config.c`.

– définir graphiquement l'utilisation des broches du microcontrôleur : on choisit l'item «Open Pins» dans le menu précédent.

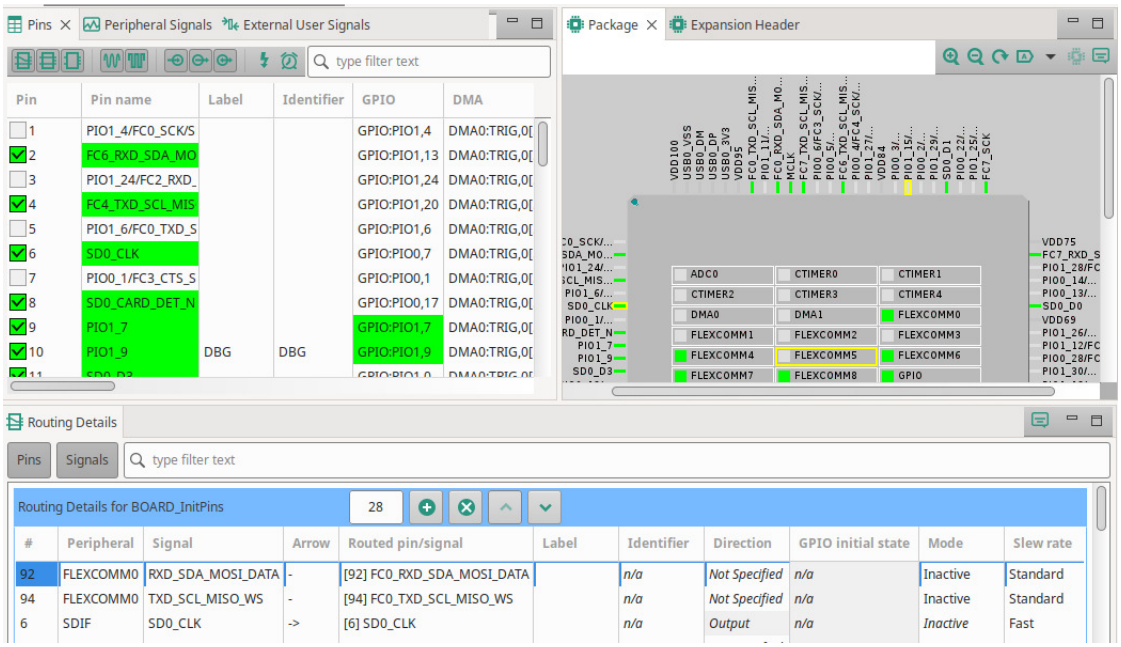

Figure 3. Configuration des broches

La sortie du mode se fait par le bouton Update Code qui réalise la génération du code reflétant la configuration graphique dans `board/pin_config.c`.

Voilà pour la partie prise en main de l'environnement et pour le contexte du projet. Chaque étudiant du groupe doit travailler en autonomie à partir de là sur les tâches qui lui sont allouées.

4. Travail étudiant 1

4.1. Gestion des séquences VT100

En l'état actuel, la fonction `sys_wait_key` dans le fichier `source/sysdep.c` se contente de détecter si le caractère récupéré par la fonction *uart_getc* correspond à '\r' ou '\n' auquel cas, on lui associe le code spécial `enter` qui permet de lancer l'interprétation de la ligne, sinon le code ASCII du caractère est simplement renvoyé.

Afin de pouvoir éditer la ligne (effacer des caractères, se déplacer avec les flèches, insérer des caractères), on utilise le fait que le terminal série associe aux touches de flèches, backspace, … du clavier les séquences d'octets détaillées ci-dessous.

code VT100	commentaire	code VT100	commentaire	code VT100	commentaire
`"\x1b[A"`	Cursor Up	`"\x1b[B"`	Cursor Down	`"\x1b[C"`	Cursor Forward
`"\x1b[D"`	Cursor Back	`"\x1b[F"`	End of line	`"\x1b[H"`	Line start
`"\x1b[5~"`	Page Up	`"\x1b[6~"`	Page Down	`"\x1b[2~"`	Insert/Help
`"\x1b[3~"`	Delete	`"\x08"`	BackSpace	`"\x04"`	eot - Ctrl D
`"\x1bOP"`	F1	`"\x1bOQ"`	F2	`"\x1bOR"`	F3
`"\x1bOS"`	F4	`"\x1b[15~"`	F5	`"\x1b[17~"`	F6
`"\x1b[18~"`	F7	`"\x1b[19~"`	F8	`"\x1b[20~"`	F9
`"\x1b[21~"`	F10	`"\x1b[23~"`	F11	`"\x1b[24~"`	F12
`"\x1b"`	ESC				

Modifier la fonction `sys_wait_key` pour qu'elle associe aux séquences ci-dessus les «codes de scan» de type `scan_t` définis dans `source/sysdep.h`. Dans le cas des caractères standard, la fonction renvoie le code `key_normal`.

Ces codes sont utilisés par la fonction `edit` de `source/edit.c` pour maintenir la position d'un curseur d'insertion en fonction des déplacements des flèches, rajout et suppression. Le terminal série, lui aussi maintient la position d'un curseur. Pour synchroniser ce curseur avec les différentes actions opérées par la fonction *edit*, il est nécessaire d'envoyer des commandes de déplacement du curseur au terminal série. Ces commandes sont envoyées sous la forme de codes VT100 et sont fournies ci-dessous. A noter qu'il existe également des commandes pour effacer l'écran du terminal série ou une partie de la ligne sur laquelle est positionné le curseur.

code VT100	commentaire	code VT100	commentaire
`"\x1b[2J"`	effacer l'écran	`"\x1b[K"`	effacer jusqu'à la fin de la ligne
`"\x1b[1C"`	déplacer le curseur d'1 position vers la droite	`"\x1b[1D"`	déplacer le curseur d'1 position vers la gauche
`"\x1b[?25h"`	cursor on	`"\x1b[?25l"`	cursor off
`"\x1b[H"`	curseur en haut à gauche		

Compléter les fonctions `sys_clear`, `move_cl_cb`, `move_cr_cb`, `cursor_on_cb`, `cursor_off_cb` et `clear_eol` en envoyant les codes adéquats avec la fonction `uart_puts`.

Vérifier que la modification interactive de la ligne de commande est maintenant possible et que l'historique des commandes activé avec les touches flèches haute et basse est fonctionnel.

La fonction `sys_out_mode` est appelée à chaque fois que le mode de fonctionnement du programme change : edition d'une ligne → édition d'une fonction → affichage de résultat → affichage d'erreur et d'avertissement. Les différents modes correspondent aux différents cas traités par le switch … case. C'est également dans cette fonction qu'on définit les «prompts» (> et $) utilisés en début de ligne d'édition de commande et de fonction. On peut profiter de ces fonctions pour définir des couleurs associées à chaque mode sous la forme de code VT100. L'encodage des couleurs est donné ci-dessous.

code VT100	commentaire	code VT100	commentaire	code VT100	commentaire
`"\x1b[31m"`	rouge	`"\x1b[32m"`	vert	`"\x1b[33m"`	jaune
`"\x1b[34m"`	bleu	`"\x1b[35m"`	magenta	`"\x1b[36m"`	cyan
`"\x1b[37m"`	blanc	`"\x1b[1;31m"`	rouge vif (rajout de 1;)	`"\x1b[2;31m"`	rouge atténué (rajout de 2;)
`"\x1b[0m"`	couleur par défaut				

Modifier, en fonction du paramètre de mode de la fonction `sys_out_mode`, la couleur d'affichage (ex: rouge pour les erreurs). Vérifier le fonctionnement.

A ce stade, l'interface de communication série est complète, et permet d'éditer une ligne de commande facilement, de rappeler les anciennes lignes dans l'historique, de compléter les noms des fonctions et même d'activer un petit débugger intégré : tester les commandes de l'exemple 3.

Exemple 3

```
function tanh(x)
  a=exp(x); b=exp(-x);
  return (a-b)/(a+b)
endfunction
trace tanh
c=tanh(1:3)
<flèche basse>
<inser>
> Expression? > a
<flèche basse>
c
trace tanh
```

4.2. Carte SD et système de fichiers FAT

Une carte SD est un périphérique permettant le stockage d'information de manière non volatile et sa récupération.

La carte de développement dispose d'un slot permettant d'accéder à une carte *microsd*. Du point de vue matériel, Le microcontrôleur dispose d'un coupleur SDIO permettant la mise en œuvre du protocole spécifique à la carte microsd. Du point de vue logiciel, un driver est fourni dans l'environnement MCUXPressoIDE :

* `drivers/sdif.[ch]` [dépendant du coupleur périphérique matériel] gestion de l'accès physique à la carte SD (détection, transfert de données en lecture/écriture),
* `sdmmc/fsl_sdmmchost.[ch]` [indépendant] couche d'abstraction du périphérique matériel,
* `sdmmc/fsl_sd.[ch]` [indépendant] abstraction de la carte SD.

A la base, une carte SD permet simplement d'accéder à des paquets d'octets référencés par des adresses. En raison des effets de vieillissement des mémoires flash utilisées pour réaliser le stockage d'information et du fait que l'écriture des informations se fait par secteur (typiquement des blocs de 512 octets), mais l'effacement par blocs de secteur (dépendant du circuit flash NAND), on n'utilise pas directement les adresses au niveau «utilisateur».

A la place, on structure le stockage des informations de manière à manipuler des **fichiers**/**répertoires** repérés par un **chemin**. Cette organisation logique rajoutée à la carte SD s'appelle un **système de fichier**. Ici, on implémente un système de fichier FAT32 hérité de Microsoft, et très couramment utilisé sur les cartes SD. Son implémentation est fournie par le driver FatFS (**http://elm-chan.org/fsw/ff/00index_e.html**).

FatFS permet de gérer une structure de système de fichier FAT sur différents support matériel. le symbole SD_DISK_ENABLE (défini dans `source/ffconf.h`) permet d'adapter la couche d'entrée sortie de bas niveau à l'utilisation de la cartes SD (voir `fatfs/source/diskio.c`).

A. Détecter, initialiser et accéder à quelques informations de la carte SD

   1. Il est nécessaire de configurer les broches : récupérer la configuration de l'exemple `sdcard_interrupt`.
   2. La carte SD est décrite par la structure `sd_card_t` dans `sdmmc/fsl_sd.h`. On a défini une variable globale

      ---

      sd_card_t card;

      Il est nécessaire de réaliser une initialisation minimale de la structure. C'est réalisé par la fonction

`sd_init` à appeler dans la fonction `main`.

```
status_t sd_init(sd_card_t *card, sd_cd_t detectCb, void *userData)
{
    SDK_ALIGN(static uint32_t s_sdmmcHostDmaBuffer[64],
        SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE);
    static sd_detect_card_t s_cd;
    static sdmmchost_t s_host;

    /* attach main clock to SDIF */
    CLOCK_AttachClk(kMAIN_CLK_to_SDIO_CLK);
    /* need call this function to clear the halt bit in clock divider register */
    CLOCK_SetClkDiv(kCLOCK_DivSdioClk,
        (uint32_t)(SystemCoreClock / FSL_FEATURE_SDIF_MAX_SOURCE_CLOCK + 1U), true);

    memset(card, 0U, sizeof(sd_card_t));
    card->host = &s_host;
    card->host->dmaDesBuffer            = s_sdmmcHostDmaBuffer;
    card->host->dmaDesBufferWordsNum= 64;
    card->host->hostController.base     = SDIF;
    card->host->hostController.sourceClock_Hz= CLOCK_GetSdioClkFreq();

    /* install card detect callback */
    s_cd.cdDebounce_ms= 100u;
    s_cd.type         = kSD_DetectCardByHostCD;
    s_cd.callback     = detectCb;
    s_cd.userData     = userData;
    card->usrParam.cd= &s_cd;

    NVIC_SetPriority(SDIO_IRQn, 5);

    return SD_HostInit(card);
}
```

Ajouter l'appel à sd_init dans le `main`. Le paramètre `userData` sera positionné à `NULL`. La détection de la carte SD (connexion/déconnexion) déclenche une interruption auprès du coupleur, puis l'appel de la callback passée en paramètre, dans laquelle on met à jour le flag global `updateFS`. Ce flag permet de déclencher l'appel de la fonction `update_sd_state` dans la routine d'attente de caractère. C'est cette fonction qui réalise la prise en compte de la carte SD et met à jour le flag `card_ready` qui pourra être testé par les fonctions susceptibles d'accéder à la carte SD (gestion du système de fichier, lecture/écriture de fichiers) pour savoir si une carte SD est présente ou pas.

Lorsqu'une carte est connectée, il faut l'initaliser avec la fonction 'SD_CardInit'. En cas de déconnexion, on ne fait rien.

Pour tester cette partie, on pourra appeler la fonction card_info qui affiche des informations relatives à la carte SD. Vérifier qu'on passe bien dans la callback à l'insertion et au retrait de la carte SD.

B. Accéder au système de fichiers FAT

1. La prise en compte du système de fichiers nécessite de réaliser dans la callback, à l'insertion de la carte SD, les actions suivantes :

   – monter le système de fichier (`f_mount`),

   – sélectionner la partition (`f_chdrive`),

   – récupérer le répertoire racine (`f_getcwd`) à copier dans la variable `cur_path`.

Lors du retrait de la carte SD du slot, il faut démonter la carte (voir la fonction `f_mount` également).

A ce stade, on doit pouvoir afficher le contenu de la carte SD à la racine lorsqu'une carte SD est présente (commande `ls`), et rien lorsqu'il n'y a pas de carte SD. Si on remet la carte SD, on doit pouvoir à nouveau afficher le répertoire racine. Un contenu standard vous est proposé dans les ressources (fichier `carte_sd.zip`).

2. Compléter la fonction **char** **fs_cd* (**char** **dir*) qui renvoie le répertoire courant si `dir==NULL`, et se déplace dans le répertoire `dir` et met à jour le répertoire courant (`f_chdir`, `f_getcwd`) sinon.

   Cette fonction est utilisée par la commande 'cd' de calc. Essayer de se déplacer dans l'arborescence et afficher le contenu du répertoire.

3. Compléter et tester les fonctions **int** *fs_mkdir*(**char*** *dirname*) et **int** *fs_rm*(**char*** *filename*).

Malgré l'initialisation du système de fichiers, le programme n'est toujours pas capable d'accéder au contenu d'un fichier (on a toujours le message indiquant que le fichier `first.e` n'a pas pu être trouvé).

4.3. Raccordement de la libc à FatFS

La libc offre des fonctions de manipulation de fichiers : open, read, write, close. Sur PC, ces fonctions sont liées aux drivers du système d'exploitation permettant d'accéder au support de stockage physique.

Ici, il faut donc pouvoir lier ces fonctions aux fonctions de FatFS qui permettent de faire l'accès au périphérique physique (la carte SD).

A bas niveau, les fonctions de la libc utilisent les fonctions suivantes

---

int *_open*(**const char** **name*, **int** *flags*, **int** *mode*);
int *_close*(**int** *fd*);
int *_read*(**int** *fd*, **char** **ptr*, **int** *len*);
int *_write*(**int** *fd*, **char** **ptr*, **int** *len*);

---

avec `name` le nom du fichier, `flags` un masque indiquant le type d'ouverture (voir par exemple **https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/** pour le détail de flags), `mode` n'est utilisé que pour les créations de fichier (droits sur le fichier). La fonction `open` renvoie un descripteur de fichier (entier) qui est utilisé par les autres fonctions pour faire référence au fichier ouvert. Le premier fichier ouvert a comme descripteur de fichier la valeur 3. Les valeurs 0 à 2 sont réservées : 0=stdin, 1=stdout, 2=stderr (pour nous, c'est le terminal série). Ressource de la libc : **https://sourceware.org/newlib/libc.html#Syscalls**.

Il sera nécessaire de prévoir une structure de données pour stocker les références FATFS aux fichiers ouverts. On pourra tirer profit de la remarque suivante : en lecture, et écriture, le dernier fichier ouvert est le seul à être accédé (on peut donc utiliser une pile).

Le code de `_open` devra anlyser le paramètre `flags` reçu par *_open* et les restranscrire en valeurs adéquates pour l'appel de f_open (open version FatFS). On pourra se servir des exemples ci-dessous pour définir les valeurs utiles de `flags`.

Créer les structures de données nécessaires et compléter les fonctions `_open`, `_read`, `_write` et `_close`.

A ce stade, vous avez la possibilité de charger des modules de fonctions présents sur la carte SD, de faire une sauvegarde des commandes entrées (commande `dump` – exemple 4), de sauvegarder et charger des matrices réelles ou complexes enformat texte ou binaire (exemple 5).

Exemple 4 (sauvegarde de session sur la carte sd)

```
dump test.txt
2+3
function f(x)
  return 2*x+1
endfunction
f(1:4)
dump test.txt
```

Exemple 5 (sauvegarde et lecture de matrices réelles ou complexes sur la carte sd)

```
N=10;x=(0:N-1)/N*2*pi;y=sin(x);
z=exp(1i*x);
mwrite(x_y,"mat.txt",0)
p=mread("mat.txt")
mwrite(x_y_z,"mat.txt",0)
p=mread("mat.txt")
mwrite(x_y_z,"mat.txt",2)
p=mread("mat.txt")

mwrite(x_y,"mat.bin",1)
p=mread("mat.bin")
mwrite(x_y_z,"mat.bin",1)
p=mread("mat.bin")
```

Aide en ligne : modifier dans le fichier `source/sysdep.h` le symbole HELPFILE, et s'assurer que le fichier `help.txt` est présent à la racine de la carte SD.

#define HELPFILE "2:/help.txt"

Vous devriez pouvoir utiliser l'aide en ligne (exemple 6).

Exemple 6

```
help cos
help polyval
```

5. Travail étudiant 2

5.1. Vue d'ensemble de la communication avec l'écran LCD

Le programme intègre des fonctions de tracé de courbe. Vous pouvez les tester avec l'exemple 7.

Exemple 7

```
x=-1:0.1:3;f="(x-1)^2-2";
subplot(111);
setplot([-1,3,-2,2]);
xgrid(-1:3,1,1,1,16); ygrid(-2:2,1,1,1,16);
plot(x,f(x),"Fc6")
xlabel("x");ylabel("y");title(f);
```

Ces fonctions font appel à bas niveau aux fonctions du fichier `source/sysdep.c` :

---

void *gsubplot*(**int** *r*, **int** *c*, **int** *i*);
void *gsetplot*(*real xmin*, *real xmax*, *real ymin*, *real ymax*, **unsigned long** *flags*, **unsigned long** *mask*);
void *ggetplot*(*real *xmin*, *real *xmax*, *real *ymin*, *real *ymax*, **unsigned long** **flags*);
void *gplot*(*Calc *cc*, *header *hdx*, *header *hdy*);
void *gsetxgrid*(*header *ticks*, *real factor*, **unsigned int** *color*);
void *gsetygrid*(*header *ticks*, *real factor*, **unsigned int** *color*);

void *gtext* (*real x*, *real y*, **char** **text*, **unsigned int** *align*, **int** *angle*, **unsigned int** *color*);
void *glabel*(**char** **text*, **unsigned int** *type*);

Ces fonctions ont déjà été adaptées pour la carte et finissent par utiliser les fonctions de l'API `lcd/lcd.h` qui permettent de dessiner des éléments graphiques simples sur l'écran LCD (ligne, rectangle, texte).

La communication avec l'écran LCD utilise un protocole série SPI avec une horloge à 50 MHz. Les fonctions de communication de bas niveau permettant d'envoyer des données à l'écran LCD sont dans le fichier `lcd/lcd_private.c`. Ce sont les fonctions :

SPI spi_master_init*(**int** *cfg*);
/* send data n 8bit data */
void *spi_write*(*SPI *spi*, *uint8_t *data*, *uint32_t n*);
/* send data 18bit data */
void *spi_write_byte*(*SPI *spi*, *uint8_t data*);
/* send data n 16bit data */
void *spi_write16*(*SPI *spi*, *uint16_t *data*, *uint32_t n*);
/* send data n times the same 16bit data */
void *spi_write16_n*(*SPI *spi*, *uint16_t data*, *uint32_t n*);

Le but de cette partie va être de démarrer le 2e cœur, de porter l'exécution de toutes les fonctions graphiques, tout en permettant leur appel depuis le code qui s'exécute sur le premier cœur.

5.2. Démarrage du cœur 1, mailbox et synchronisation

Le microcontrôleur dispose de deux cœurs CM33. Le cœur 0 démarre en automatique après un reset du microcontrôleur. L'autre doit être démarré explicitement. Le second cœur n'est pas tout à fait équivalent au premier : il ne dispose pas d'unité de calcul en virgule flottante. Les opérations sur des réels doivent être émulées à partir d'opérations sur des entiers et sont donc lentes. Il n'a pas non plus accès au DSP.

Ce second cœur est donc en réalité là pour décharger le premier pour réaliser des opérations qui ne nécessitent que la manipulation d'entier. Dans notre cas, l'affichage des lignes, rectangle, etc, réalisé par le module lcd ne nécessite que l'utilisation d'entiers. On se propose de basculer toutes les fonctions d'affichage sur le cœur 1, et de s'en servir comme «carte graphique» pour notre application.

Les deux processeurs partagent l'ensemble des ressources du microcontrôleur (périphériques et mémoire), mais ont une table des vecteurs, un NVIC, et des registres (dont le PC) qui leur sont propres. La solution proposée par NXP pour gérer le fait que chaque processeur est capable d'exécuter son propre flux d'instructions est de créer deux projets : un maître, qui s'exécutera sur le cœur 0, et l'autre esclave, qui s'exécutera sur le cœur 1. On réalise un partitionnement de la mémoire en affectant des zones mémoire distinctes pour chaque processeur. On répartit également les périphériques utilisés par chaque cœur. L'horloge du système est commune et initialisée par le cœur 0 après le reset.

1. Etudier le projet d'exemple disponible avec le SDK via `driver_examples > mailbox > mailbox_interrupt_cm33_core0` et `..._core1`. Ils fournissent deux projets maître et esclave simples. Les notes d'application `AN12358.pdf` et *AN12335.pdf* donnent les éléments essentiels à la mise en place d'un projet muticœur et commentent le projet d'exemple. On notera dans l'exemple que l'horloge (clock config) n'est pas reconfigurée au démarrage du cœur 1 ; les deux processeurs utilisent l'horloge configurée par le cœur 0.

2. **Démarrer le cœur 1**

   Le mapping mémoire adopté pour l'application est donné figure 4.

   Projet maître :

   * Modifier le projet calculette (*après en avoir fait une copie*) pour rendre le projet multicœur. Il faudra, entre autres choses, ajouter le symbole `MULTICORE_APP=1` qui permet d'intégrer le code de la mailbox au projet. Le code de la mailbox de l'exemple a été remplacé par notre propre driver (fonctions `mb_init`, `mb_pop_evt`, `mb_push_evt`). On fournit également la fonction `core1_startup` qui prend en paramètre

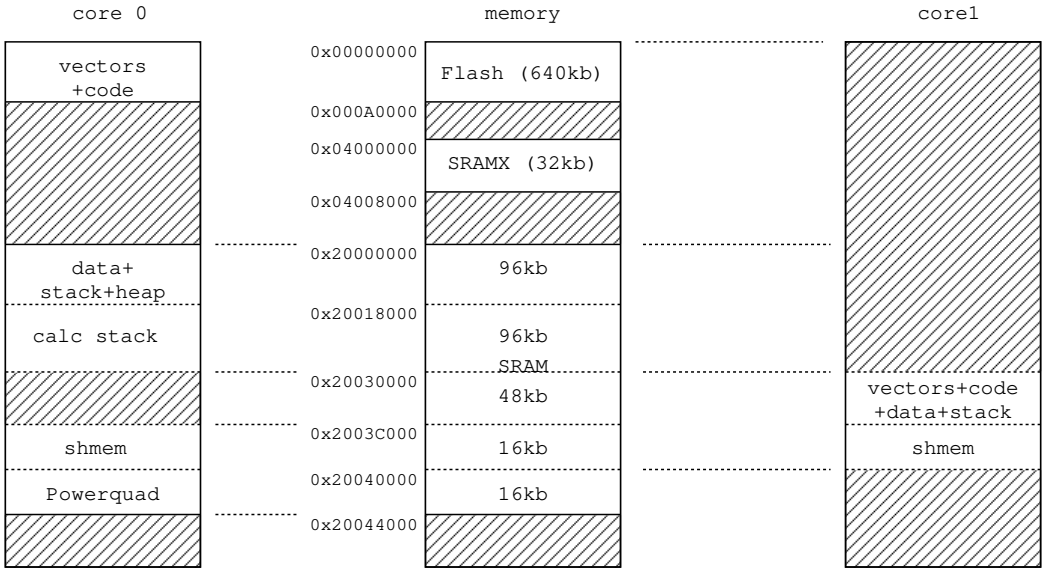

Figure 4. Multicœur : mapping mémoire

l'adresse de démarrage pour le cœur 1.

* Démarrer le cœur 1dans la fonction main. L'adresse de démarrage est $0x20030000$ (adresse de la table des vecteurs du cœur 1).

Projet esclave :

* créer un projet esclave ou dupliquer et modifier la configuration du projet esclave d'exemple.

* remplacer le fichier main du projet par le fichier `main_core1.c` disponible dans les ressources. Remplacer aussi le fichier `drivers/fsl_clock.c` par celui fourni dans les ressources. On y a modifié les fonctions `CLOCK_GetPll0OutFreq` `CLOCK_GetPll1OutFreq` pour qu'elles recalculent la fréquence d'horloge du processeur à partir de la configuration des registres des PLL réalisée par le cœur 0. C'est nécessaire pour pouvoir configurer et utiliser des périphériques, mais curieusement le driver se réfère à une variable interne au projet (non partagé entre les projet maître et esclave), ce qui amène à considérer, pour le projet esclave que la fréquence issue des PLL est de 0Hz …

* porter les fonctions `mb_pop_evt` et `mb_push_evt` à partir du code fourni dans le projet maître (lire également la section sur la mailbox dans le manuel de programmation du microcontrôleur `UM11126.pdf` chapitre 52, p 1171).

Vérifier que le mapping mémoire présenté figure 4 est bien respecté pour les deux parties (c'est une cause commune de non fonctionnement).

Tester les deux projets conjointement. La fonction `core1_startup` (exécutée par le cœur 0) attend une notification venant du cœur 1 via la mailbox (EVT_CORE_UP) que le cœur 1 a effectivement démarré. Vérifier que le second cœur est démarré (LED rouge clignote, possibilité d'interrompre le cœur 1), et que la calculette s'exécute sur le cœur 0 (prompt dans le terminal série, évaluation de calculs).

5.3. Exécution des fonctions LCD sur le cœur 1, de manière synchrone par rapport au cœur 0

Dans le code actuel, toutes les fonctions d'affichage sur l'écran LCD sont exécutées sur le cœur 0. L'affichage d'une courbe mathématique nécessite la conversion des coordonnées des points de la courbe en coordonnées en pixel pour l'écran. Cette conversion est réalisée par les fonctions `g_xgrid`, `g_ygrid`, `g_draw_plot`, `gtext`.

Ces fonctions utilisent ensuite les fonctions du module `lcd` pour réaliser la représentation effective sur l'écran. Dans les trois premières fonctions qui permettent respectivement de tracer les échelles (avec une grille) en x, en y, puis les points d'une courbe, le nombre de points à traiter peut devenir important (on impose une limite max de 2048 points). Les coordonnées en pixel calculées sont codées sur des entiers 16 bits, ce qui offre une

dynamique de représentation plus que suffisante (32 bits pour le codage d'un point M(x,y)).

Les coordonnées en pixel calculées sont stockées dans la zone `shmem` partagée par les deux cœurs et sont accessibles des deux côtés via la variable `shdata` déclarée à la fois dans `sysdep.c` (cœur0) et dans `main_core1.c` (cœur 1).

On se servira de la variable `shdata` pour «transférer» les points à afficher du cœur 0 vers le cœur 1 (le «transfert» a lieu sans recopie de donnée, ce qui permet de conserver de bonnes performances). On utilisera la mailbox pour envoyer des messages indiquant au cœur 1 les actions à réaliser sur les données de la zone partagées. Chaque message est formaté sur 32 bits et permet de définir un évènement relatif à une requète graphique (EVT_DRAWLINE, EVT_DRAWRECT, …), ainsi que des données spécifiques à l'évènement (nombre de points, style, …). La figure 5 présente un format général pour les messages, ainsi que deux exemples spécifiques.

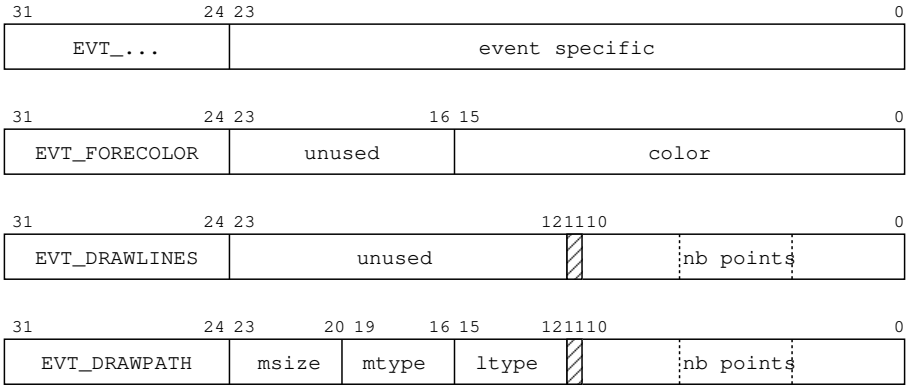

Figure 5. Exemples de structure de messages

Côté esclave, les messages reçus par l'ISR de la mailbox sont placés dans une file d'attente de type FIFO (buffer circulaire). La fonction `next_event` extrait un message dont il faut ensuite récupérer les différents éléments en fonction du type de message, puis exécuter la fonction graphique requise.

On propose la démarche suivante

* Copier le répertoire `lcd` du projet maître vers le projet esclave et réaliser l'initialisation du module `lcd` dans la fonction main du projet esclave.

---

lcd_init();
lcd_switch_to(*LCD_DPY*);

L'initialisation doit être effectuée avant d'envoyer la notification de démarrage du cœur 1.

* Dans le projet maître, supprimer l'initialisation du bloc `lcd`. Supprimer tout le code du répertoire `lcd` sauf le répertoire `fonts` qui contient les polices de caractères et le fichier `lcd.h`. Décommenter dans les fonctions `lcd_`… du fichier `sysdep.c`. Ces fonctions vont servir de tuyau (stub) pour appeler les implémentations côté projet esclave. Les fonctions `lcd_set_forecolor` et `lcd_drawlines` sont déjà écrites.

  La fonction `Color lcd_set_forecolor(Color color)` permet de spécifier la couleur du crayon pour dessiner sur l'écran. La fonction `void lcd_draw_lines(SPoint *p, int n)` permet d'envoyer la commande d'affichage des *n* points en coordonnées pixel du tableau passé en paramètre. Ce tableau est en réalité `shdata`. Cette dernière fonction attend une notification de type `EVT_RETVAL` de la part du cœur 1 qui met ainsi en place un protocole de type «poignée de main» (handshake) qui permet d'assurer que le tableau `shdata` n'est utilisé que par une seule fonction à un instant t (sorte d'exclusion mutuelle).

  Vous notez que les prototypes des fonctions peuvent avoir changé par rapport à ceux du bloc `lcd`. Effectuer les ajustements pour que le code soit compilable.

  Dans la fonction `g_draw_path2d`, dans le cas `L_SOLID`, on traçait la courbe en dessinant des seg-

ments de droite liant les différents points consécutifs. Remplacer la boucle d'affichage par un appel à `lcd_draw_lines`. Tester le projet conjointement au projet esclave avec le code de la calculette de l'exemple 8.

Exemple 8

```
x=-1:0.1:3;f="(x-1)^2-2";
subplot(111);
setplot([-1,3,-2,2]);
plot(x,f(x),"Fc6")
```

On doit voir l'affichage d'une parabole en rouge (sans axes ni graduation).

* Définir un format de message et compléter les fonctions `lcd_segments` (affichage de la grille), `lcd_draw_path2d` (affichage de courbe(s) et de points, avec tous les styles possibles), `lcd_draw_rect` (cadre autour des graphiques), `lcd_fill_rect` (effacer l'écran), `lcd_clip`, `lcd_unclip` (clipping), `lcd_set_font`, `lcd_set_alignment`, `lcd_set_direction`, `lcd_draw_string` (affichage de texte). Le message devra être encodé sur 32 bits. Les données supplémentaires qui ne pourraient pas être encodées dans le message (de type coordonnées pixel par exemple) seront passées par le tableau partagé `shdata`. Il faudra mettre en place le protocole de type «poignée de main» (handshake) pour permettre d'indiquer au cœur 0 (maître) que le message a été reçu et traité par le cœur 1 (esclave).

  Lors de l'exécution de la fonction `lcd_draw_path2d`, la première partie du tableau partagé `shdata` est occupée par les points de la courbes à tracer. Le codage adopté pour les messages suggère de limiter un nombre maximal de points à 2048 (imposé), ce qui laisse de la place au delà de cette limite pour encoder plusieurs buffers dans la zone `shdata` pour effectuer d'autres opérations graphiques (corrdonnées d'une chaîne de texte à afficher …).

  Effectuer le nécessaire pour avoir l'affichage des différentes courbes avec les styles différents proposés dans l'exemple 9.

Exemple 9

```
f0=2k;T0=1/f0;t=-T0:T0/12:T0;x=1.2*sin(2*pi*f0*t);
t1=-T0:T0/20:T0;x1=1.2*sin(2*pi*f0*t1);
subplot(211);setplot([-T0,T0,-1.5,1.5]);
xgrid((-0.5:0.25:0.5)*1m,1m,1,1,1);ygrid(-1.5:0.5:1.5,1,1,1,1);
plot(t,x,"F,lb#,c2+");plot(t1,x1,"l-");
subplot(212);setplot([-T0,T0,-2,2]);
xgrid((-0.5:0.25:0.5)*1m,1m,1,1,1);ygrid(-2:2,1,1,1,1);
plot(t,x,"F,ls,c1+");plot(t1,x1,"l-");

subplot(211);setplot([-T0,T0,-1.5,1.5]);
xgrid((-0.5:0.25:0.5)*1m,1m,1,1,1);ygrid(-1.5:0.5:1.5,1,1,1,1);
plot(t,x,"F,lb,c1+");plot(t1,x1,"l-");
subplot(212);setplot([-T0,T0,-2,2]);
xgrid((-0.5:0.25:0.5)*1m,1m,1,1,1);ygrid(-2:2,1,1,1,1);
plot(t,x,"F,ls#,c6+");plot(t1,x1,"l-");
```

A terme, pour des raisons d'efficacité, il serait sans doute utile d'envisager de transférer la totalité de la fonction `g_draw_path2d` du côté esclace.

5.4. Exécution des fonctions LCD sur le cœur 1, de manière asynchrone par rapport au cœur 0

L'utilisation du second cœur n'apporte pas de gain de performance au système étant donné que chaque fonction qui passe des données par la zone partagée `shdata` doit attendre que la fonctionnalité côté esclave soit exécutée pour permettre au code côté maître de continuer. Par ailleurs, le système est moins sobre en énergie qu'avec un seul cœur. Chaque cœur fonctionne en permanence, même lorsqu'il n'a rien à faire (on brûle des calories dans une boucle d'attente).

* **Rendre asynchrone la communication entre les deux cœur**

  L'idée est de supprimer le protocole «poignée de main», et de gérer la zone partagé `shdata` comme un

«pool» de mémoire dans lequel on allouera des buffers pour les différentes d'affichage (côté maître) qui n'auront alors qu'à écrire les données, sans besoin de rester attendre l'exécution côté esclave.

Du côté esclave, les messages reçus seront stockés dans la file d'attente en vue de leur exécution. Une fois exécuté le buffer associé sera libéré pour que la mémoire puisse être réutilisée par une opération d'affichage côté maître.

Mettre en place les structures nécessaires à la gestion de la zone partagée sous la forme de buffers allouables et supprimer le protocole «poignée de main». Aucune donnée venant du maître ne doit être perdue : si on ne peut pas allouer un buffer côté maître, il faut attendre.

* **Optimiser la consommation électrique**

  La fonction `__WFI(void)` (Wait For Interrupt) permet d'endormir le cœur qui l'exécute. Le réveil est réalisé lors d'une demande d'interruption. Modifier les projets maître et esclave pour économiser l'énergie.

5.5. Exécution autonome

En l'état, le code du cœur 1 est chargé automatiquement par MCUXpresso en RAM. On est dépendant de la connexion à MCUXpresso : si on fait un reset ou qu'on coupe l'alimentation, le code et les data du cœur 1 sont perdus !

On ne peut pas non plus avoir le code des deux cœurs en FLASH et les faire fonctionner à partir de la FLASH (ils se retrouveraient en compétition pour accéder à leurs instructions, ce qui nuirait aux performances de l'ensemble).

Pour rendre autonome le programme sur la carte, il faut que le code et les valeurs initiales des data pour les deux cœurs soit en FLASH (mémoire non volatile). Pour avoir de bonnes performances lors de l'exécution, on copie au démarrage le code du cœur 1 dans la SRAMX (le bus d'instruction du cœur 1 est connecté à la SRAMX – voir figure 1), et ses data à l'adresse `0x20030000` (comme avant). On démarre le cœur 1 sur le code dans la SRAMX.

Effectuer les modifications nécessaires au niveau du code du cœur 0, et de l'éditeur de liens. Tester. Quand les tests sont concluants, fermer MCUXPresso, faire un reset.

6. Travail étudiant 3

6.1. Utilisation de l'accéléromètre MMA8652

1. Test de l'accéléromètre : le projet `sen_cm33_lab3_i2c.zip` permet de tester l'accéléromètre MMA8652 présent sur la carte indépendamment de la calculette. La communication I2C avec l'accéléromètre est opérationnelle et une interface a été développée pour configurer et utiliser simplement l'accéléromètre. L'API (fichier `component/mma8652fc.[hc]`) comprend les fonctions :

   ---

   status_t mma8652_init(*I2C_Type *i2c*, *uint32_t cfg*);
   status_t mma8652_id(*uint32_t *id*);
   status_t mma8652_status(*uint8_t *st*);
   status_t mma8652_read_xyz(*int32_t* data*);

   La fonction *mma8652_read_xyz*(*int32_t* data*) prend en paramètre l'adresse d'un tableau de 3 entiers signés 32 bits et doit renvoyer les composantes xyz de l'accélération de pesanteur en mg. En l'état, la fonction ne réalise que la communication avec le capteur pour récupérer les données brutes. Il faut compléter la fonction de manière à transformer ces données brutes en valeurs signées et normalisées en mg. Le tableau 15 de la page 17 de la note d'application AN4083 donne des informations sur la manière de procéder. On pourra également consulter la datasheet du capteur pour obtenir le protocole de communication utilisé.

   Compléter le code et tester.

2. Intégration dans la calculette : la fonction `maccel` du fichier `source/dsp.c` du projet calculette est appelée lorqu'on entre la commande `accel()` dans l'interface de la calculatrice. Le lien est réalisé par le tableau `binfunc_list[]` du fichier `source/calc.c`. Classiquement, une fonction appelée par la calculatrice doit

utiliser les fonctions de manipulation de la pile de calcul (`source/stack.h`) pour :

- récupérer les paramètres passés à la fonction sur la ligne de commande (aucun ici) à partir de lapile de calcul (`next_param` et `getvalue`),
- réserver de l'espace sur la pile de calcul pour stocker le résultat qui sera fourni par la fonction (`new_real/complex/matric/cmatrix/...`),
- réaliser le traitement requis et remplir la variable résultat : on peut accéder à la zone de données par les macros `realof`, `matrixof` ou la fonction `getmatrix`,
- pousser le résultat à l'origine de la pile (on écrase la zone utilisée pour passer les paramètres) avec `pushresults`.

Le document **insider.md**, présent sur gitlab, fournit une description de l'API, ainsi qu'un exemple d'utilisation de cette API.

Intégrer l'accès à l'accéléromètre à la calculatrice (config broches et horloge pour le port FLEXCOMM4 = I2C4, driver accéléromètre), et compléter la fonction `maccel` pour qu'elle permette de renvoyer un vecteur [1x3] comportant une mesure de l'accélération de pesanteur en «g» (et pas «mg»). Tester la fonction.

Ecrire un script de test qui permet de relever l'accélération de pesanteur à intervalle de temps régulier et qui trace les 3 composantes sur une courbe.

6.2. Gestion du temps

Les fonctions

---

real sys_clock (**void**);
void *sys_wait* (*real time*, *scan_t *scan*);

---

doivent permettre respectivement d'obtenir un tick correspondant au temps écoulé depuis le démarrage de la carte en millisecondes, et de déclencher une temporisation qui sera donnée en millisecondes également. L'appui sur une touche permet d'interrompre la temporisation. Ces deux fonctions constituent les fonctions de bas niveau appelées par les fonctions `time()` et `wait(ms)` de la calculatrice.

Pour implémenter ces fonctionnalités, on utilise le timer SysTick associé au processeur (cœur 0) qui est incrémenté au rythme de la fréquence d'horloge du processeur donné par la variable globale *SystemCoreClock*.

uint32_t SysTick_Config(*uint32_t ticks*);

avec *freq_div* le nombre d'incrémentations du compteur entre deux interruptions. Compléter l'appel de *SysTick_Config* pour générer des demandes d'interruptions toutes les millisecondes.

La routine d'interruption nommée *SysTick_Handler* est définie au niveau des fonctions *sys_clock* et *sys_wait* et devra incrémenter la variable *sys_tick_cnt*. Cette incrémentation devra être réalisée de manière à pouvoir représenter exactement la variable *sys_tick_cnt* lorsqu'elle est convertie en nombre en virgule flottante.

La fonction *sys_clock* se contentera de renvoyer le nombre de millisecondes écoulées. La fonction *sys_wait* est bloquante et attendra, si le delai passé en paramètre (*time*) est positif, qu'il soit écoulé (renvoyer un code scan 0), ou que la fonction soit interrompue par l'appui d'une touche (renvoyer le code de scan `escape`). Si le délai *time* est négatif, on attendra simplement l'appui sur une touche (on pourra afficher un message du genre «press a key»), et la fonction renverra le code de scan 0.

Implémenter et tester les fonctions *sys_clock* et *sys_wait*, ainsi que la routine d'interruption du timer SysTick.

6.3. PowerQuad : cosinus d'un vecteur

Le «PowerQuad» est un coprocesseur associé au cœur 0 du microcontrôleur, et développé par NXP. Il permet d'accélérer l'exécution d'un certain nombre d'opération par rapport à un calcul réalisé en utilisant le FPU

(Floating Point Unit) du Cortex-M33.

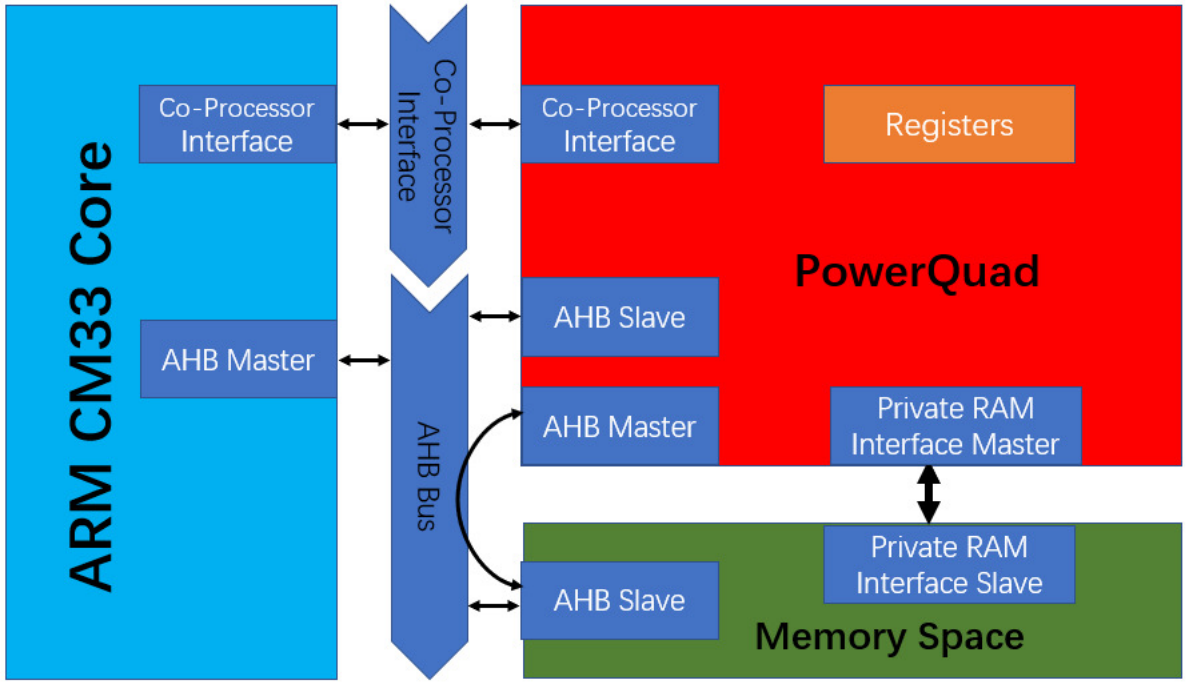

Figure 6. PowerQuad

La figure 6 montre les interfaces de communication possibles :

* Utilisation de l'interface coprocesseur du processeur Cortex-M33 : il est dès lors possible d'étendre le jeu d'instruction du processeur en ajoutant des instructions exécutées par le processeur PowerQuad, comme si elles étaient intégrées au jeu d'instruction du processeur lui-même. Les fonctions accélérées sont : (x1)/x2), sqrt(x), sin(x), cos(x), ln(x), exp(x), exp(-x), 1/(x), 1/sqrt(x), biquad(x).

  Les calculs sont réalisés en virgule flottante. Les données et résultats peuvent être donnés, soit en figule flottante (simple précision), soit en virgule fixe.

* Utilisation du PowerQuad en périphérique : le PowerQuad dispose de registres de configuration permettant de choisir les formats de données utilisés en entrée et en sortie et d'indiquer les adresses où chercher ces données. Les fonctions utilisées dans ce mode sont : filtrage FIR, corrélation, convolution, opérations matricielles, FFT/IFFT, DCT/IDCT.

* Le circuit dispose également d'une interface 128 bits permettant la communication avec la mémoire pour accéder aux données d'entrée et de sortie. Il peut utiliser la SRAM de 16kB démarrant à l'adresse `0x20040000` pour ses besoins de calculs intermédiaires.

Plusieurs documents sont intéressants à consulter pour comprendre comment mettre en œuvre ce composant : le manuel de programmation du LPC55S69 `UM11126.pdf`, les notes d'applications AN12282, AN12383, AN12387, les tutoriaux disponibles sur le site mcuoneclipse :

* Tutorial 1: **coprocessor operations**
* Tutorial 2: **matrix operations**
* Tutorial 3: **FFT**

Les vidéos présentes dans ces tutoriaux sont particulièrement instructives.

Compléter la fonction `mpqcos` du fichier `source/dsp.c` qui permettra de calculer le cosinus d'un nombre ou d'un vecteur réel. On pourra s'orienter vers les fonctions `PQ_CosF32` et `PQ_VectorCosF32` de l'API `drivers/fsl_powerquad.h`. Il sera nécessaire de vérifier le type des données fournies en entrée (champ *type* de la structure `header` dans `source/stack.h`) et de générer une erreur en cas de mauvais paramètre (voir la gestion des erreurs dans a fonction `mfft`).

Tester la fonction et valider les résultats obtenus. Ecrire un benchmark qui permettra de comparer la durée d'exécution de la fonction `cos` et `pqcos`. On pourra implémenter d'autres fonctions et réaliser des benchmarks.

6.4. PowerQuad : FFT

Etudier les exemples d'utilisation des fonctions `PQ_TransformRFFT`, `PQ_TransformCFFT` et `PQ_TransformIFFT` de l'API PowerQuad.

Interfacer ces fonctions avec les fonctions `pqfft` et `pqifft`. Il faudra permettre de passer en paramètre un vecteur réel ou complexe (et donc vérifier le type du paramètre) et réaliser les transtypages nécessaires.

Tester la validité des résultats et réaliser un benchmark pour comparer les temps d'exécution avec les fonctions `fft` et `ifft` de la calculatrice.