TP4 PIC32MX
Générateur de signal piloté via USB

CADRE DE LA MANIPULATION

Cette manipulation a pour objectif de permettre le réglage des paramètres du générateur (réalisé au TP3) via une liaison USB.

Le KIT PIC32MX sera vu comme un USB device de type cdc (Communication Device Class), ce qui permettra de gérer l'USB facilement côté PC comme un port de communication virtuel. Les paramètres du générateur seront réglés à partir d'une application C# utilisant un port de communication série.

PRINCIPE DU COMPORTEMENT DU PROGRAMME

Le principe voulu est de disposer du générateur et du réglage local lorsque l'USB n'est pas connecté.
Lorsque l'USB est actif, le menu local n'agit plus. Pour que l'observateur puisse faire la différence au niveau du menu principal, au lieu d'avoir un * au début de la ligne active, il est demandé d'afficher # au début des 4 lignes.

Au démarrage on affichera :
(pendant 3 s.)

```
TP4 UsbGen 2016-17
Nom1
Nom2
```

Les valeurs des paramètres sont récupérées de l'EEPROM I2C externe (MCP79411), pour autant que l'on retrouve la valeur du champ "Magic". Ceci en remplacement de la sauvegarde dans la mémoire flash-programme interne.
A faire dès que l'exercice 9-1 est terminé.

ORGANISATION DU PROJET PIC32MX

Pour obtenir un USB qui fonctionne, nous allons utiliser l'exemple qui est sous :

C:\microchip\harmony\v<n>\apps\usb\device\cdc_com_port_single.

Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire **cdc_com_port_single** au même niveau et renommons la copie en **TP4_UsbGen**.

Le chapitre des travaux pratiques traitant de l'USB sert de référence pour la modification du projet.

Ensuite, on ajoutera à ce projet les fichiers permettant de faire fonctionner le générateur. Il sera encore nécessaire d'adapter les parties systèmes pour retrouver les interruptions nécessaires au générateur et à la gestion locale du menu.

Il faudra donc ajouter au niveau du MHC les drivers des timers (static avec interruption) :

- Timer 1, période 1 ms, priorité 3.
- Timer 3, période initiale 250 us, priorité 7.

Pour le reste il faut reprendre tous les fichiers du TP3, mais ne pas copier les fichiers comme app.c et autres fichiers systèmes.

MODIFICATION POUR LA GESTION DES MENUS

Il est nécessaire d'apporter une légère modification aux fonctions des menus.

Voici les nouveaux prototypes des fonctions :

```
// pas de changement
void MENU_Initialize(S_ParamGen *pParam);

void MENU_Execute(S_ParamGen *pParam, bool local);
```

Le rôle de bool local est d'effectuer un test : Si local = true affichage standard du menu, si false affichage en remote.

```
// nouveau demande sauvegarde
void MENU_DemandeSave(void);
```

Permet d'obtenir la sauvegarde avec un affichage minimal (exploiter la fin de la séquence de sauvegarde du menu).

FONCTIONS DE COMMUNICATION

Pour permettre la réception des messages et l'envoi des paramètres, voici deux fonctions proches de celles utilisées pour le TP2 :

Contenu du fichier Mc32gest_SerComm.h (fourni) :

```
// prototypes des fonctions
bool GetMessage(int8_t *USBReadBuffer, S_ParamGen *pParam,
                bool *SaveTodo);
void SendMessage(int8_t *USBSendBuffer,
                 S_ParamGen *pParam, bool Saved);
```

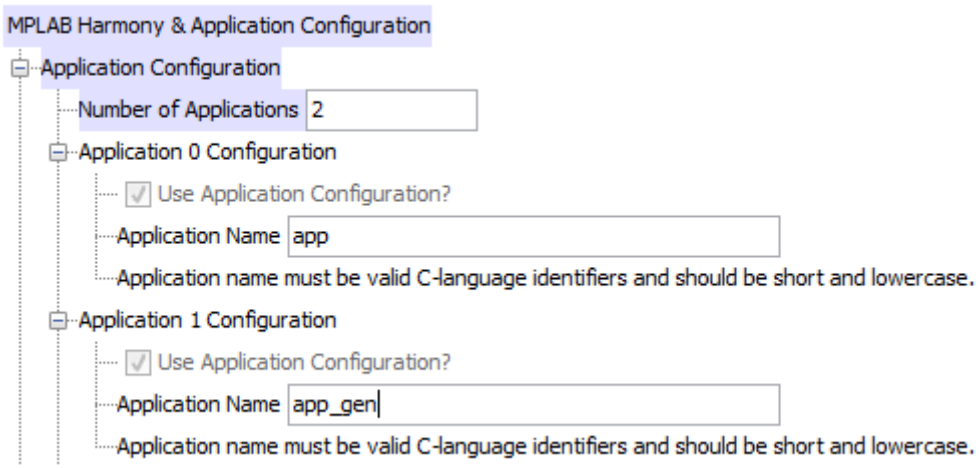
Remarque : Lorsque la sauvegarde des paramètres est effectuée avec succès, il faut transmettre une fois un message avec Saved = true pour permettre d'éventuellement afficher une quittance au niveau de l'application.

FICHIERS FOURNIS EN COMPLEMENT

Vous trouvez les fichiers Mc32gest_SerComm.h et Mc32gest_SerComm.c dans le répertoire :
...\Maitres-Eleves\SLO\Modules\SL229_MINF\TP\TP4_UsbGen\Fichiers_TP4

ADAPTATION DE L'APPLICATION

La machine d'état de l'application issue de l'exemple USB a des états spécifiques au fonctionnement USB. Pour permettre le traitement habituel avec les 3 états (INIT, WAIT, SERVICE_TASKS), la solution consiste à ajouter une deuxième application distincte avec sa machine d'état. Cela se fait dans le MHC de la manière suivante :



Cela va permettre de reprendre les éléments de l'application générateur et menu du TP3.

Dans l'initialisation de l'application, on commence par synchroniser les paramètres remote avec les locaux :

```
// Synchronise les paramètres
RemoteParamGen = LocalParamGen;
```

Puis, par exemple dans le case APP_GEN_STATE_SERVICE_TASKS :

```
    // Execution du menu
    if ( usbStat) {
         MENU_Execute(&RemoteParamGen, false);
    } else {
         MENU_Execute(&LocalParamGen, true);
    }
```

Ce fonctionnement à 2 machines d'état implique :

- Il faut trouver une solution pour mettre à jour bool usbStat, en fonction de la situation de communication.
- Plus généralement, il faudra prévoir des variables ou fonctions pour se passer les informations entre les 2 machines d'état.

CONCEPT DE COMMUNICATION AVEC L'USB

Au niveau de la réception, on obtient les données dans un tampon déclaré comme suit :

```
uint8_t APP_MAKE_BUFFER_DMA_READY readBuffer[APP_READ_BUFFER_SIZE];
```

Avec

```
#define APP_READ_BUFFER_SIZE 64
```

Ce qui nous permet un message de max 64 octets. Comme le système ne présente pas de fonction permettant de connaitre la taille des données reçues et que le message est traité comme une chaine de caractères, nous allons choisir un message affichable comportant une indication du paramètre et de sa valeur :

```
!S=TF=2000A=10000O=+5000W=0#
   S                -      1#
   C
   D
```

Pour la forme du signal, T=triangle, S=sinus, C=carré et D=dent-de-scie. Suivent les valeurs de la fréquence, de l'amplitude et de l'offset. Pour demander une sauvegarde, on envoie W=1.

AIDE POUR LE PARSING EN RECEPTION

Il y a certaines fonctions de <string.h> qui peuvent être utiles, comme **strstr**, qui permet de chercher l'occurrence d'une chaine dans une autre.

De <stdlib.h>, on dispose de strtol, atoi, atol etc.

De <stdio.h>, on dispose de sscanf, équivalent du scanf utilisant une chaine de caractères en input.

EMISSION

Pour tester l'émission, nous allons envoyer un message faisant écho aux valeurs reçues avec un complément comme :

```
!S=TF=2000A=10000O=+5000WP=0#
!S=TF=2000A=10000O=+5000WP=1#     // ack sauvegarde
```

☝ La fonction sprintf convient bien pour cela !

APPLICATION C#

En vous inspirant des applications fournies pour le test de la communication RS232, il est demandé de réaliser une application permettant de choisir la forme du signal (utilisation liste ou ComboBox) ainsi que la fréquence, l'amplitude et l'offset (utilisation de NumericUpDown ou de TrackBar). Utilisation d'un Button pour déclencher la sauvegarde.
Prévoir des zones d'affichage pour les messages reçus et envoyés ainsi que le choix du port de communication.

TRAVAIL ET DELIVRABLES

Le travail s'effectue par groupe de deux en partageant la réalisation, ce qui permet d'acquérir des compétences en collaboration dans la réalisation d'un programme.

☝ Cependant, il est demandé que chaque étudiant réalise toutes les étapes de mise en place du projet USB et de son adaptation.

Les délivrables consisteront en :

- Listings des programmes en C :
  - app.c (uniquement les modifications par rapport à l'exemple Harmony)
  - appgen.c
  - menuGen.c
  - Mc32gest_SerComm.c
- Explications concernant :
  - Eléments clés de l'adaptation du projet USB de base, comment se greffe le générateur, etc,
  - Modifications apportées dans appgen.c,
  - Fonctionnement et principes de communication USB :
    - Prise en compte du branchement de l'USB et impact sur le fonctionnement du générateur,
    - Réception et émission,
    - Décodage des messages reçus et mise en forme des messages à envoyer,
  - Sauvegarde en EEPROM I2C externe.
- Listing et vue face avant de l'application C#.
- Une démonstration du fonctionnement en relation avec la fiche de contrôle :
  - Affichage et signal généré,
  - Comportement au branchement/débranchement de la liaison USB,
  - Comportement du générateur lors de communication USB,
  - Sauvegarde .

Votre travail sera évalué sur la base de :

- Qualité, facilité de réutilisation/modification et taux d'aboutissement du code.
- Fonctionnement et taux d'aboutissement.

DUREE DE LA REALISATION

A réaliser en 5 séances.