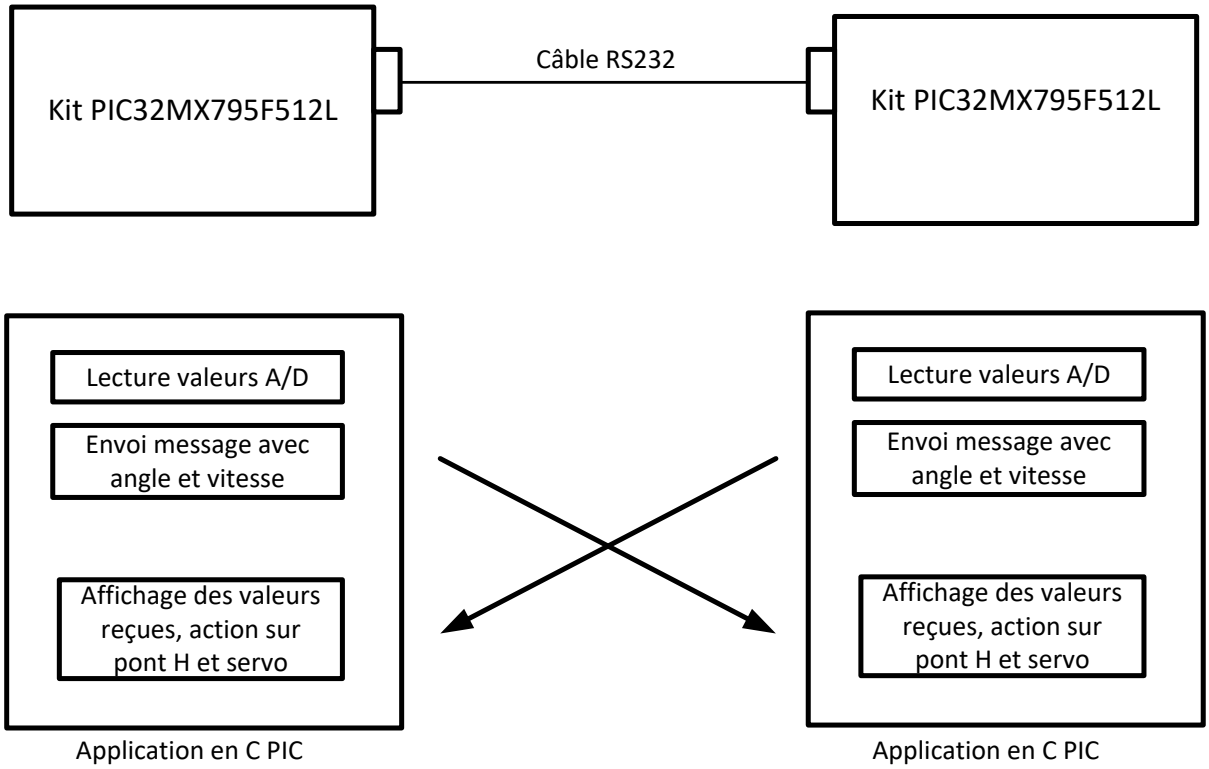
TP2 PIC32MX
PWM et RS232

OBJECTIFS

Ce 2ème travail pratique a pour objectif d'ajouter la possibilité de varier la vitesse du moteur ainsi que l'angle du servomoteur à partir de la liaison RS232.

La même application sera chargée dans deux kits communiquant entre eux par le biais d'un câble croisé ou par la suite entre un kit et une application C# sur PC.

VUE D'ENSEMBLE DE L'APPLICATION



- L'application microcontrôleur effectue la lecture des 2 canaux A/D et les utilise pour transmettre une valeur d'angle et de vitesse.
- De même, elle reçoit les messages, affiche les valeurs reçues et agit sur le PWM du pont en H et un servomoteur.
- Lorsque la liaison est établie, l'application affiche les valeurs de vitesse et d'angle reçues et agit en conséquence sur le moteur et le servomoteur. Les valeurs obtenues des potentiomètres sont transmises.
- Lorsque la liaison n'est pas établie ou coupée, on retourne au réglage local. Cela implique un mécanisme de surveillance de la communication.

STRUCTURE DU MESSAGE BINAIRE

Start	Vitesse	Angle	CRC16	
0xAA	1 byte	1 byte	MSB CRC	LSB CRC

Angle est une valeur 8 bits signée variant de -90 à +90.
Vitesse est une valeur 8 bits signée variant de -99 à +99.

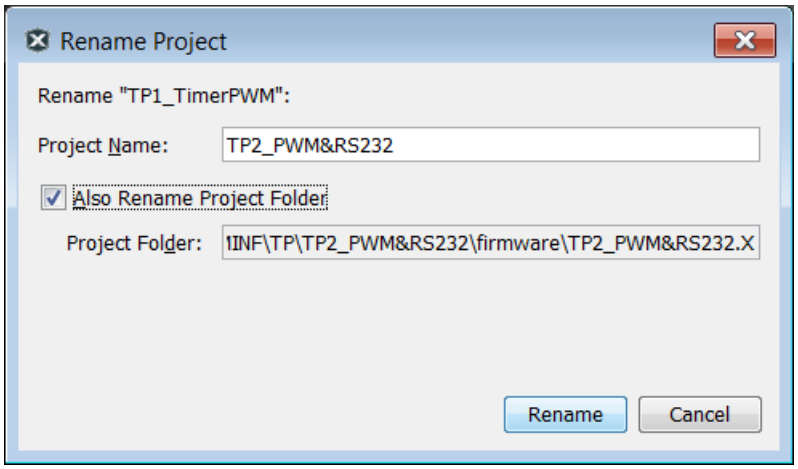
- Les valeurs sont transmises sous forme binaire, donc un octet suffit par valeur.
- A l'émission, le CRC est calculé sur les 3 premiers bytes (0xAA, Vitesse, Angle). La valeur obtenue est placée à la suite du message.
- A la réception, on calcule le CRC sur tout le message. Si tout est correct, on obtient 0.

Pour calculer le CRC, on utilise la fonction updateCRC16.
En voici un exemple d'utilisation :

```
uint16 ValCrc16 = 0xFFFF;
ValCrc16 = updateCRC16(ValCrc16, 0xAA);
ValCrc16 = updateCRC16(ValCrc16, Vitesse);
ValCrc16 = updateCRC16(ValCrc16, Angle);
```

PRINCIPE DE REALISATION DU PROJET

Il faut partir d'une copie du TP1 que l'on renomme TP2. Selon le principe suivant :



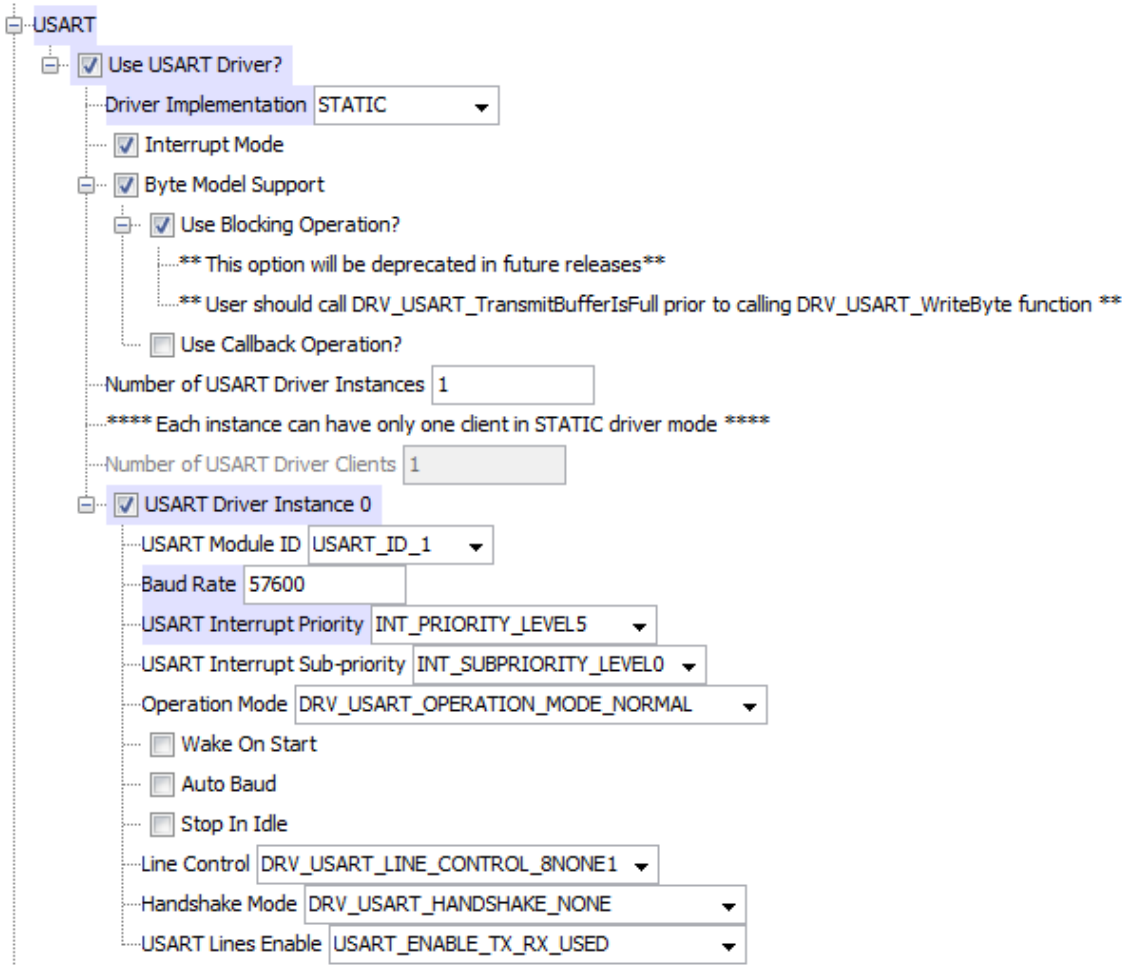
Ne pas oublier le "Set as Main Project" !

AJOUT AU NIVEAU MHC

En relançant le MHC, il est possible de modifier la configuration !

a) Supprimez le timer4.

b) Il faut introduire un driver USART avec les choix suivants :



Cette configuration correspond à utiliser l'USART 1 avec un baudrate de 57600 Bd en prévoyant d'utiliser les interruptions d'émission et de réception.

Lors de la génération, il ne faut pas écraser les fichiers existants et réaliser la fusion en activant successivement les ajouts. Exception pour le timer 4.

MODIFICATION NECESSAIRE

Pour obtenir un comportement adapté au niveau du traitement des interruptions de réception et d'émission de l'USART, il faut effectuer manuellement la modification suivante au niveau du fichier drv_usart_static.c.

Situation après génération :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_IDLE,
                               USART_ENABLE_TX_RX_USED);
```

Situation après modification :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_EMPTY,
                               USART_ENABLE_TX_RX_USED);
```

Liste des modes pour RX interrupt :

```
typedef enum {
    USART_RECEIVE_FIFO_3B4FULL = 0x02,
    USART_RECEIVE_FIFO_HALF_FULL = 0x01,
    USART_RECEIVE_FIFO_ONE_CHAR = 0x00
} USART_RECEIVE_INTR_MODE;
```

Liste des modes pour TX interrupt :

```
typedef enum {
    USART_TRANSMIT_FIFO_NOT_FULL = 0x00,
    USART_TRANSMIT_FIFO_IDLE = 0x01,
    USART_TRANSMIT_FIFO_EMPTY = 0x02
} USART_TRANSMIT_INTR_MODE;
```

COMPLEMENT DU PROJET EXISTANT

Sous ...\Maitres-Eleves\SLO\Modules\SL229_MINF\TP\TP2_PwmRs232\Fichiers_TP2, vous trouverez 6 fichiers :

- Mc32CalCrc16.c et Mc32CalCrc16.h — Librairie calcul CRC16
- GesFifoTh32.c et GesFifoTh32.h — Librairie gestion des FIFO
- Mc32gest_RS232.c et Mc32gest_RS232.h — Librairie gestion RS232

Ces fichiers sont à placer au même niveau que app.c et app.h

☝ Le fichier Mc32gest_RS232.c contient l'ISR pour l'UART, il faut supprimer l'ISR générée par le MHC.

MODIFICATIONS ET COMPLEMENT

AU NIVEAU DE APP.C

Dans le case APP_STATE_INIT, ajout de l'appel à la fonction InitFifoComm (prototype dans Mc32gest_RS232.h).

Déplacement des appels des 3 fonctions GPWM_GetSettings, GPWM_DispSettings et GPWM_ExecPWM dans le case APP_STATE_SERVICE_TASK.

Complément du traitement dans APP_STATE_SERVICE_TASK en ajoutant les appels aux fonctions GetMessage et SendMessage.

Lorsque l'on est connecté (CommStatus = true), on obtient les PWMDatas de la fonction GetMessage et on obtient les entrées locales avec GPWM_GetSettings pour les émettre avec SendMessage.

Traitement proposé :

```
case APP_STATE_SERVICE_TASK:
{
    // Réception param. remote
    CommStatus = GetMessage(&PWMData);

    // Lecture pot.
    if (CommStatus == false)   // local ?
        GPWM_GetSettings(&PWMData); // local
    else
        GPWM_GetSettings(&PWMDataToSend); // remote

    // Affichage
    GPWM_DispSettings(&PWMData, CommStatus);

    // Execution PWM et gestion moteur
    GPWM_ExecPWM(&PWMData);

    // Envoi valeurs
    if (CommStatus == false)   // local ?
        SendMessage(&PWMData); // local
    else
        SendMessage(&PWMDataToSend); // remote

    appData.state = APP_STATE_WAIT;
    break;
}
```

MODIFICATION AU NIVEAU DU FICHIER GESTPWM.C ET .H

Il faut modifier la fonction GPWM_DispSettings pour qu'elle gère un paramètre supplémentaire.

```
void GPWM_DispSettings(S_pwmSettings *pData, bool Remote);
```

Si Remote = true, il faut afficher sur la 1ère ligne :

```
** Remote Settings
```

Autrement :

```
   Local Settings
```

LA FONCTION GETMESSAGE

Voici le prototype de la fonction **GetMessage** :

```
bool GetMessage(S_pwmSettings *pData);
```

La fonction **GetMessage** extrait le message à partir du FIFO de réception. **Cette fonction ne doit pas être bloquante**. A chaque appel, la fonction vérifie si un message complet est présent dans le FIFO. La fonction n'effectue le traitement que si le message est complet. Lorsqu'un message est complet et valide au niveau de son CRC, la fonction met à jour tous les champs de la structure. En plus **la fonction retourne false si pas de réception de message durant 10 cycles et true dès réception d'une trame**.

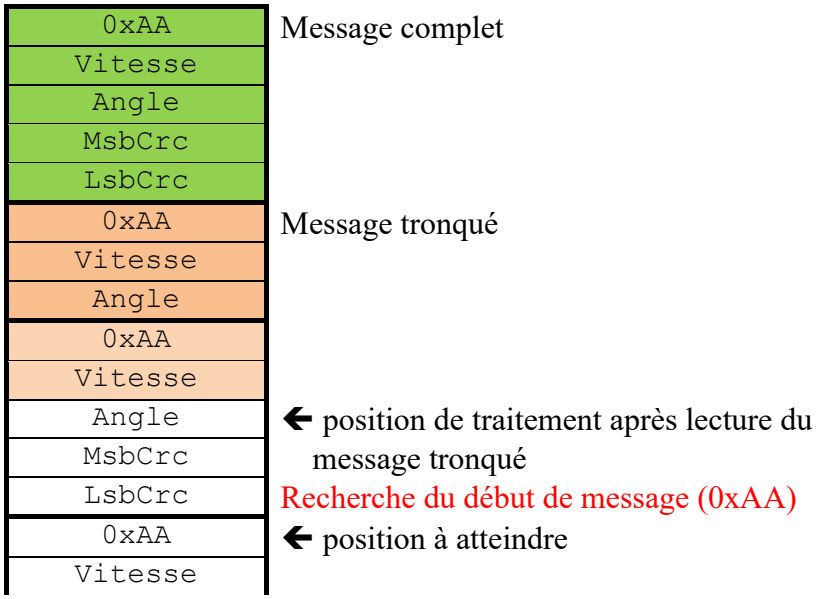
Principe de réalisation :

Il s'agit de déterminer s'il y a déjà au moins un message présent dans le FIFO de réception. S'obtient par :

```
NbCharToRead = GetReadSize (&descrFifoRX);
// Si >= taille message alors traite
if  (NbCharToRead >= MESS_SIZE)  {

    // analyse du contenu du message
}
```

L'analyse du contenu du message consiste à lire (extraire) un à un les caractères présents dans le FIFO avec **GetCharFromFifo** (&descrFifoRX, &RxC); tout en vérifiant si le début du message est bien 0xAA et si le CRC est valide.



Il est nécessaire de mettre en place un mécanisme pour se positionner au début du message suivant (on ignore le message tronqué et on perd le suivant).

LA FONCTION SENDMESSAGE

Voici le prototype de la fonction SendMessage :

```
void SendMessage(S_pwmSettings *pData);
```

La fonction **SendMessage** doit préparer la trame et la déposer dans le FIFO d'émission pour autant qu'il y ait assez de place pour un message complet. Transmission des champs SpeedSetting et AngleSetting.

La fonction **SendMessage** est appelée **tous les 5 cycles** (pour ménager la carte ou application distante).

☝ Modification à introduire par rapport à la suggestion du APP_STATE_SERVICE_TASK.

Principe de réalisation :

Il s'agit de déterminer s'il y a assez de place pour écrire un message complet dans le FIFO d'émission. S'obtient par :

```
// Test si place Pour écrire 1 message
FreeSize = GetWriteSpace (&descrFifoTX);
if (FreeSize >= MESS_SIZE ) {

   // Compose le message

   // Dépose le message dans le fifo
     PutCharInFifo ( &descrFifoTX, TxMess.Start);
   ... idem pour les 4 bytes suivants
}
```

La composition du message peut être réalisée en utilisant :

```
// Struct pour émission des messages
StruMess TxMess;
```

La composition du message comprend également le calcul du CRC sur les 3 premières valeurs.

GESTION DU TIMER4

Le timer 4 n'est plus nécessaire, il faut au minimum le maintenir stoppé (ne pas appeler la fonction de Start).

AU NIVEAU DE SYSTEM_INTERRUPT.C

☝ Le fichier Mc32gest_RS232.c contient l'ISR pour l'UART. Au niveau du fichier system_interrupt.c, il faut supprimer l'ISR générée par le MHC.

La réponse à l'interruption de l'UART fournie est à compléter afin de gérer notamment les FIFO d'émission et de réception ainsi que le handshaking hardware.

Elle comporte une action sur led 3 afin d'observer la présence et la durée de l'interruption. En plus, led 4 toggle dans int. RX et led 5 toggle dans int. TX.

Il faut modifier la réponse à l'interruption du timer 1 (cycle de 20 ms) pour uniquement établir l'état de l'application à APP_STATE_SERVICE_TASK à chaque cycle, une fois que 3 secondes se sont écoulées (150 cycles).

AU NIVEAU DE MC32GEST_RS232.C

Il faut réaliser la fonction GetMessage et la fonction SendMessage.

ASPECT AFFICHAGE

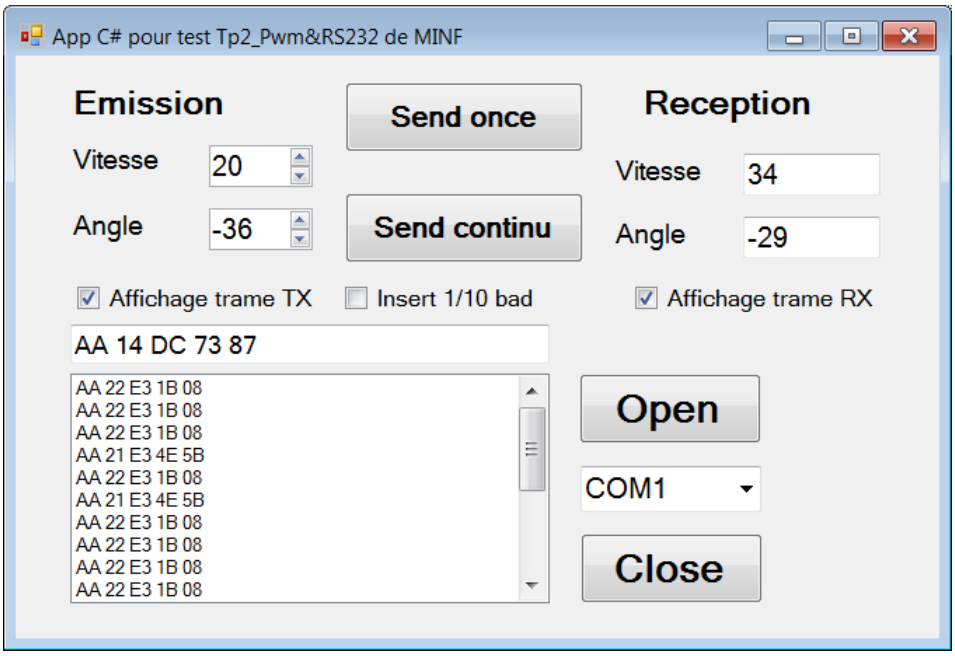
Après toutes les initialisations nécessaires, le programme affiche durant 3 secondes :

```
Local Settings
TP2 PWM&RS232 <année>
Nom 1
Nom 2
```

Ensuite, c'est la fonction GPWM_DispSettings qui prend en charge les affichages en fonction de la situation.

APPLICATION C# DE TEST

Sous ...\Maitres-Eleves\SLO\Modules\SL229_MINF\TP\TP2_PwmRs232\Fichiers_TP2, vous trouverez le répertoire de l'application C# contenant l'exécutable du projet.
L'application permet l'affichage des trames émises et reçues. Elle effectue le contrôle du CRC selon le même principe que le programme PIC32. Elle permet la génération de mauvais messages.



TRAVAIL ET ETABLISSEMENT DU RAPPORT

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.

A rendre au plus tard à la fin de la 1ère séance :

- L'organigramme (diagramme de flux) ou structogramme GNS de principe de GetMessage.

Le rapport final doit contenir au minimum les éléments suivants :

- Listing de app.c avec mise en évidence des modifications liées à l'ajout de GetMessage et SendMessage.
- Le listing de Mc32gest_RS232.c
- Des observations à l'oscilloscope montrant la relation entre l'interruption de réception (led 3 et led 4) et le signal RX.
- Des observations à l'oscilloscope montrant la relation entre l'interruption d'émission (led 3 et led 5) et le signal TX. Pour cela, il faut mettre l'application C# en mode d'envoi unique.
- En utilisant la possibilité de perturber les trames de l'application C# et en ajoutant un toggle de la led 6 lorsque l'on a un mauvais CRC, montrez la situation du signal sur led 6 en relation avec le signal RX et l'interruption de réception (led 4).
- Le fonctionnement doit être démontré avec l'application C# de test (en relation avec la fiche de contrôle).

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 MANIPULATION

A réaliser en 3 séances.