Projet
conception d'un robot mobile

Professeur : Valentin Gies

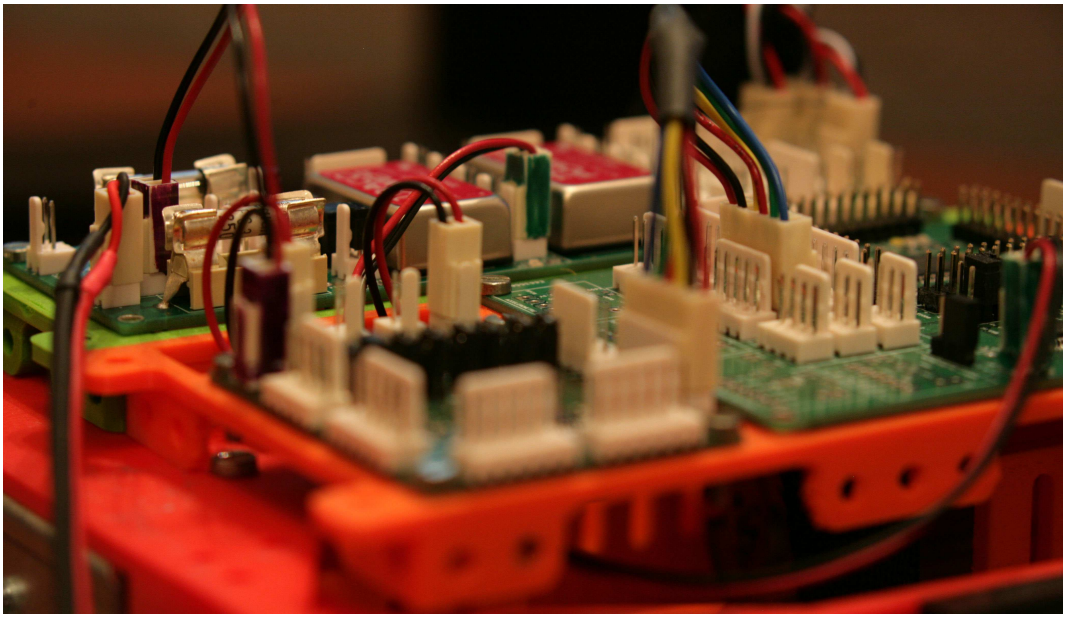

Table des matières

1 A la découverte des bus terrains dans les microcontrôleurs

1.1 Le bus SPI

La carte de contrôle des robots permet de raccorder des capteurs à un bus SPI de manière simple, via deux connecteurs dénommés SPI1 et SPI2 (fig.**??**).

1.1.1 Interfaçage de capteurs SPI

La mesure des paramètres d'un système embarqué est une fonctionnalité qui se retrouve dans la grande majorité des systèmes actuels. En particulier les mesures de position, vitesse et accélération sont très communes.

Les capteurs permettant ces mesures sont le plus souvent interfacés avec un microcontrôleur via un bus terrain de type I2C ou SPI. Nous proposons dans cette partie de réaliser cet interfaçage sur une centrale inertielle comprenant un accéléromètre 3 axes, un gyroscope 3 axes et un magnétomètre 3 axes.
Le bus SPI du DSPIC permet des communications série à haute vitesse avec des composants externes. La communication est de type *Master-Slave*, le *Master* étant le DSPIC.

Le $dsPIC33FJ512GM306$ utilisé possède deux bus SPI. Nous utiliserons le premier. **Attention, une erreur s'est glissée dans la sérigraphie de la carte. SPI1 est en SPI2 et vice-versa.** Les pinsdu module *SPI1* ne sont pas remappables, et il n'est donc pas nécessaire de faire d'autre déclaration que leur état entrée ou sortie. Ces pins sont les suivantes :
- $SCLK$: Sortie *SPI Clock* : pin $C3$ (sortie)
- $MOSI$: Sortie *Master Out Slave In* : pin $A4$ (sortie)
- $MISO$: Entrée *Master In Slave Out* : pin $A9$ (entrée)

Le *Chip Select* (CS) est une pin simple configurée en sortie. On le placera sur la pin $B7$ correspondant sur la carte à *SPI SS1*.

La centrale inertielle utilisée est une *MPU9250* de chez InventSense, que l'on retrouve dans de nombreux produits grand public.

$\Longrightarrow$ Afin de comprendre comment fonctionne cette centrale inertielle et surtout d'avoir une référence complète sur le sujet, vous pouvez télécharger sa documentation technique à l'aide du lien suivant : Page de présentation de la $MPU9250$.

$\Longrightarrow$ Configurez les entrées-sorties du $dsPIC$ pour que le SPI puisse fonctionner, ainsi qu'une macro permettant d'accéder facilement au *Chip Select* à l'aide du code suivant, placé dans *IO.h* :

```
#define SS_MPU9250 _LATB7
```

$\Longrightarrow$ Configurez la pin remappable $RP18$ ($B5$), afin qu'elle gère l'interruption *INT1*. Cette interruption permettra de savoir quand les données du gyroscope sont disponibles.

$\Longrightarrow$ Insérez dans le projet les fichiers SPI.c, SPI.h, SPI_MPU9250.c et SPI_MPU9250.h.

$\Longrightarrow$ Examinez le code des fichiers fournis précédemment. Expliquer dans le détail dans votre rapport le fonctionnement du SPI, codé dans la fonction *WriteMultipleCommandMultipleReadSpi1* située dans le fichier *"SPI.c"*. Cette fonction est le coeur de la liaison SPI.

$\Longrightarrow$ Insérez dans le code du fichier *"main.c"* les initialisations du SPI, du driver du gyroscope et de l'interruption nécessaire au fonctionnement du gyroscope

$\Longrightarrow$ Insérez dans la boucle infinie du "main.c" le code à compléter suivant, qui permet la récupération des données du gyroscope :

```
if(MPU9250IsAccelDataReady())
{
    ...
}
```

⟹ Testez votre code. En inclinant successivement la carte sur les 3 axes, vous devriez avoir un résultat semblable à celui-ci dans l'interface de visualisation.

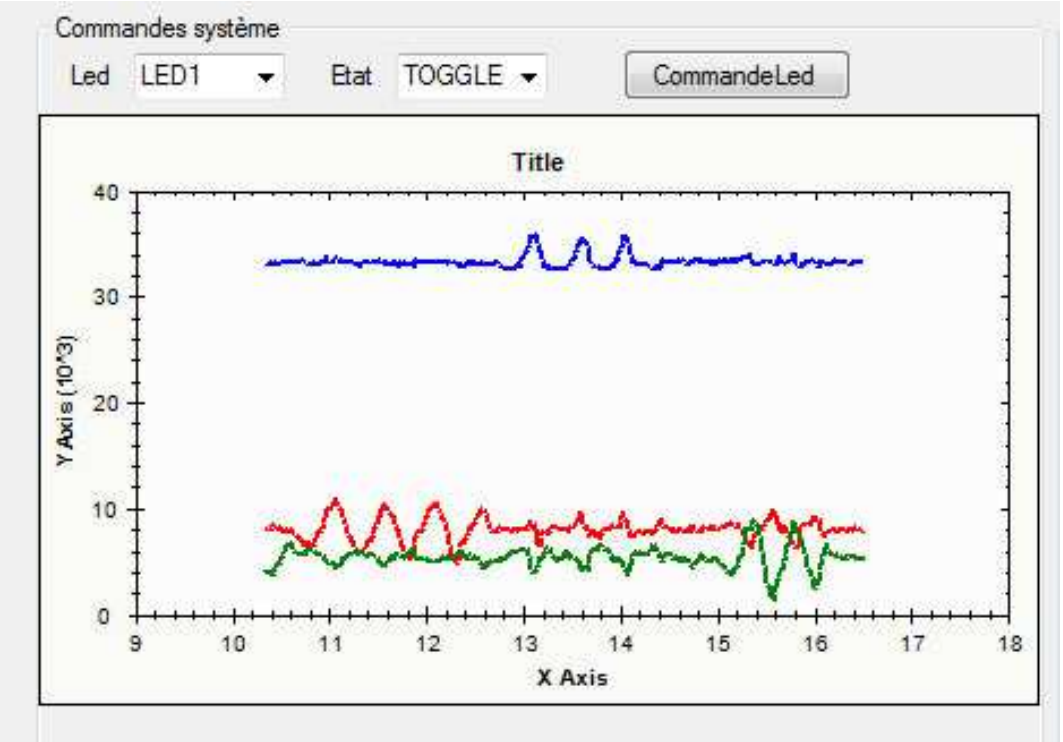

FIGURE 1 – Signaux du gyroscope 3 axes

⟹ Visualisez à l'aide de l'oscilloscope les signaux SPI du gyroscope. Vous devriez obtenir en cycle normal des signaux de la forme suivante :

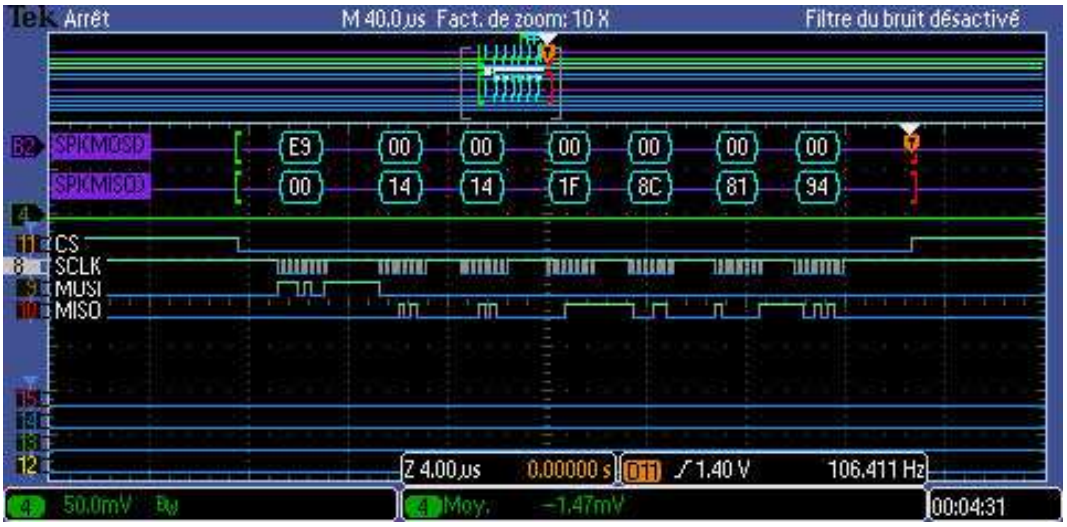

FIGURE 2 – Chronogrammes d'une récupération de données sur le gyroscope via la liaison SPI

⟹ En zoomant sur le signal *SCLK*, mesurez la fréquence d'horloge SPI. Est-ce conforme à ce que vous pouvez voir dans le code (regardez les pre et port scaler de la liaison SPI déclarés dans la fonction *L3G4200SetSPIFreq*) ?

⟹ Expliquez le fonctionnement de l'interruption du gyroscope en vous aidant de la datasheet et du code fourni. En particulier, indiquez à quelle condition l'interruption est levée et à quelle condition elle revient à 0. Décrivez le mécanisme de récupération des données associé à cette interruption.

⟹ Visualisez simultanément les signaux SPI et UART. Vous devriez avoir des signaux de la forme suivante :

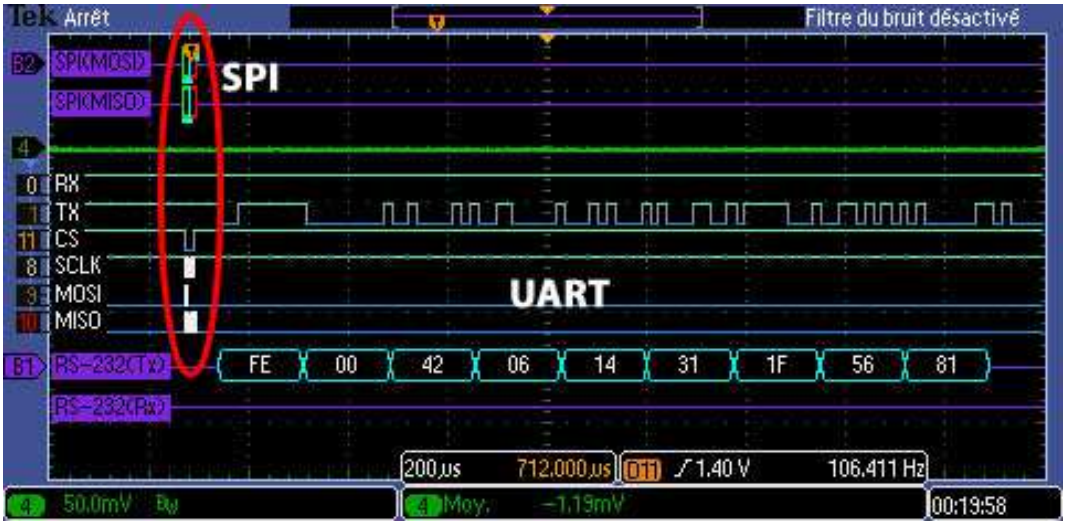

FIGURE 3 – Chronogrammes d'une récupération de données sur le gyroscope via la liaison SPI suivie de l'envoi UART

⟹ Que pouvez-vous dire des vitesses relatives de deux types de bus terrain étudiés ?
A présent que vous avez fait fonctionner l'accéléromètre de la $MPU9250$, faites de même avec le gyroscope.

⟹ En vous inspirant du driver de l'accéléromètre de la *MPU9250*, créez un driver pour le gyroscope de la

MPU9250, et les fonctions permettant de visualiser dans l'interface *C#* les données issues du capteur.

1.2 Le bus I2C

Le bus I2C est un bus terrain utilisé dans nombre de capteurs fonctionnant avec une fréquence d'acquisition modérée. Nous proposons dans cette partie de réaliser un interfaçage sur bus I2C pour un capteur de type télémètre à ultra-sons assez perfectionné.

$\Longrightarrow$ Sa référence est le *SFR08*. Afin de comprendre son fonctionnement, vous pouvez télécharger sa documentation technique à l'aide du lien suivant : Lien vers la page du SFR08.

$\Longrightarrow$ Afin de vous gagner un peu de temps un driver générique pour l'I2C vous est fourni. Il est téléchargeable ici : I2C.c et I2C.h. Attention : l'I2C est un bus typiquement non fiable, qui se fige de manière aléatoire. Dans ce cas, des procédures doivent être mise en oeuvre une fois le blocage détecté. Cette détection est prévue dans le driver ci-dessus, elle nécessite pour fonctionner l'appel de la fonction IncrementI2CAntiBlockCounter, depuis un timer à une fréquence de 1kHz.

$\Longrightarrow$ Le code suivant est appelé périodiquement sur une interruption timer qui lance un event lu dans le main. Expliquez sommairement ce qu'il fait à partir de la datasheet (la dernière ligne est très importante), et déterminer la fréquence maximale à laquelle doit fonctionner le timer. En quoi est-ce intéressant d'utiliser des télémètre à ultrasons ayant un mode broadcast ?

```
if(IsEventActive(EVENT_ULTRASONIC_SENSOR))
{
    ClearEvent(EVENT_ULTRASONIC_SENSOR);
    unsigned char trame[8];
    unsigned char data[10];
    unsigned int telemetreValue;

    //Cas Lecture et écriture synchrone
    I2C1ReadN(TELEMETRE_GAUCHE, 0x00, data, 10);
    telemetreValue = ((unsigned int)data[2])*256 + ((unsigned int)data[3]);
    if(telemetreValue > 0)
    {
        SetDistanceTelemetreFaceGauche(telemetreValue);
        telemetreValue = GetDistanceTelemetreFaceGauche();
        trame[0] = (unsigned char)(telemetreValue);
        trame[1] = (unsigned char)(telemetreValue>>8);
    }
    I2C1ReadN(TELEMETRE_DROIT, 0x00, data, 10);
    telemetreValue = ((unsigned int)data[2])*256 + ((unsigned int)data[3]);
    if(telemetreValue > 0)
    {
        SetDistanceTelemetreFaceDroit(telemetreValue);
        telemetreValue = GetDistanceTelemetreFaceDroit();
        trame[2] = (unsigned char)(telemetreValue);
        trame[3] = (unsigned char)(telemetreValue>>8);
    }
    I2C1ReadN(TELEMETRE_CENTRE, 0x00, data, 10);
    telemetreValue = ((unsigned int)data[2])*256 + ((unsigned int)data[3]);
    if(telemetreValue > 0)
    {
        SetDistanceTelemetreFaceCentre(telemetreValue);
        telemetreValue = GetDistanceTelemetreFaceCentre();
        trame[4] = (unsigned char)(telemetreValue);
        trame[5] = (unsigned char)(telemetreValue>>8);
    }
    #ifdef DEBUG_MODE_TELEMETRE
        UartEncodeAndSendMessage(0xAD, trame, 8);
    #endif

    I2C1Write1(ALL_I2C, 0x00, 0x51);  //Mesure en broadcast sur tous les SFR08 en êmme temps
```

```
}
```

$\Longrightarrow$ A partir de l'analyse de la documentation technique, déterminer un appel de fonction *I2C* d'initialisation possible pour les télémètres dans le cas où ils sont placés horizontalement et qu'il sont donc susceptibles de recevoir un premier écho en provenance du sol.