# TP3 & DM : MICROCONTROLEUR 2 : COMMUNICATION SPI AVEC UN ACCELOROMETRE ET VISUALISATION DES DONNEES

## RENDU

- codes C
- rapport :
    - réponse aux questions
    - captures d'écrans des mesures réalisées analog discovery 2 (prendre des captures avec le numéro du boitier visible)
    - interprétation des mesures

## OBJECTIFS DU TP

- Configurer la communication SPI entre un microcontrôleur STM32 et le capteur ADXL345.
- Lire et traiter les données d'accélération du capteur (axes X, Y, Z).
- Transmettre les données via l'USART.
- Visualiser les données en temps réel dans Processing.
- Utiliser l'Analog Discovery 2 pour analyser les signaux SPI et USART.

## MATERIEL

- Carte Nucleo STM32F103RB.
- Capteur ADXL345.
- Boîtier Analog Discovery 2 pour la capture des signaux.
- Logiciel Processing pour la visualisation des données.

# PARTIE 1 : ANALYSE FONCTIONNELLE

**Objectif**

L'objectif de cette analyse fonctionnelle est d'identifier et comprendre les aspects techniques et matériels du capteur ADXL345, nécessaires pour établir une communication fiable avec le microcontrôleur STM32F103RB via le protocole SPI. Cela inclut l'étude des connexions physiques, des registres, ainsi que des paramètres de configuration SPI spécifiques.

## TABLEAU DES ENTREES/SORTIES

Le tableau ci-dessous présente les connexions physiques entre le capteur ADXL345 et le STM32. Chaque broche du capteur doit être correctement connectée à une broche spécifique du STM32 pour permettre une communication SPI correcte.

Compléter le tableau avec la **Correspondance STM32**

| **Broche ADXL345** | **Fonction** | **Correspondance STM32** |
|---|---|---|
| CS | Chip Select (SPI Enable) | |
| SCL/SCLK | Horloge SPI | |
| SDA/SDI/SDIO | Données SPI (Entrée/Sortie) | |
| SDO/ALT_ADDRESS | Données SPI (Sortie) | |
| INT1 | Interruption 1 | |
| INT2 | Interruption 2 | |
| VS | Alimentation | 3.3V |
| GND | Masse | GND |

## TABLEAU DES REGISTRES ESSENTIELS DU CAPTEUR ADXL345

Complétez le tableau avec : **adresse** et **Valeur de Réinitialisation**

| **Adresse** | **Nom du Registre** | **Fonction** | **Valeur de Réinitialisation** |
|---|---|---|---|
| | DEVID | Identifiant unique du périphérique | |
| | POWER_CTL | Contrôle l'activation du capteur (mode mesure/veille) | |
| | DATA_FORMAT | Configuration du format des données et de la plage d'accélération | |
| | DATAX0 - DATAZ1 | Données d'accélération pour les axes X, Y, Z | |
| | BW_RATE | Contrôle la bande passante et le taux d'échantillonnage | |
| | FIFO_CTL | Contrôle le fonctionnement du FIFO (Bypass, Stream, etc.) | |
| | INT_SOURCE | Identifie la source des interruptions | |

## TABLEAU DES PARAMETRES SPI

Compléter le tableau avec les valeurs recommandée

| **Paramètre** | **Valeur recommandée** | **Description** |
|---|---|---|
| Mode SPI | | L'horloge est inactive à l'état haut, échantillonnage sur le front montant |
| Fréquence d'horloge | | Fréquence maximale pour la communication SPI avec le capteur |
| Durée de CS à SCK | | Délai minimum entre la mise à zéro de CS et l'activation de l'horloge SPI |

## PARTIE 2 : CONFIGURATION DU SPI SUR LE STM32

**Objectif**

Configurer le STM32 pour jouer le rôle de maître SPI et initier la communication avec le capteur ADXL345. L'objectif est de configurer le bus SPI pour assurer une bonne communication entre le microcontrôleur et le capteur.

### BROCHES DU SPI2 SUR STM32

Dans cette partie, vous allez configurer les broches associées au **SPI2**. Chaque broche SPI doit être configurée correctement pour que la communication soit possible entre le STM32 et le capteur ADXL345.

**Tâche 1** : Complétez le tableau suivant en identifiant les broches utilisées par le **SPI2** sur le STM32, ainsi que leurs fonctionnalités et configurations.

| **Broche STM32 (SPI2)** | **Nom correspondant dans le protocole SPI** | **Fonctionnalité SPI** | **Configuration GPIO** |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

**Exercice** :

- À partir des informations fournies, complétez le tableau avec les détails sur la configuration des GPIOs (ex : type de sortie, mode alternatif, etc.).
- Vous devez configurer ces broches pour correspondre aux fonctionnalités SPI et activer les bons registres GPIO.

## CONFIGURATION DES REGISTRES SPI

Dans cette partie, vous allez analyser et configurer les registres du **SPI2** sur le STM32 en fonction de l'analyse fonctionnelle effectuée auparavant. Vous devez identifier les registres essentiels à la configuration du **SPI2** pour assurer une communication correcte avec le capteur ADXL345.

**Exercice : Analyse et Configuration des Registres SPI2**

**Registre SPI_CR1 (Control Register 1)** : Ce registre contrôle des aspects clés du fonctionnement du SPI, comme le rôle du périphérique (maître ou esclave), la vitesse de l'horloge, la polarité et la phase de l'horloge.

Complétez le tableau suivant en indiquant les rôles des différents bits du SPI_CR1 et la configuration spécifique requise pour le capteur ADXL345 :

| **Bits** | **Nom** | **Description** | **Valeur à Configurer** |
|---|---|---|---|
| | MSTR | Sélection du mode maître ou esclave | |
| | BR[2:0] | Vitesse de l'horloge SPI | |
| | CPOL | Polarité de l'horloge (Idle High ou Low) | |
| | CPHA | Phase de l'horloge (Front montant/descendant) | |
| | SSM | Gestion logicielle du CS | |
| | SSI | Permet de forcer le signal CS | |

**Registre SPI_CR2 (Control Register 2)** : Ce registre contrôle certains aspects supplémentaires de la configuration SPI, notamment la gestion du signal CS (Slave Select Output Enable).

| Bits | Nom | Description | Valeur à Configurer |
|---|---|---|---|
| 2 | SSOE | Activation automatique du signal CS | |

**Instructions pour l'Exercice**

1. Analysez le rôle de chaque bit dans les registres SPI_CR1 et SPI_CR2.
2. Complétez les colonnes manquantes dans les tableaux en vous basant sur l'analyse fonctionnelle du capteur ADXL345 et du protocole SPI.
3. Justifiez vos choix de valeurs en fonction des spécifications du capteur et de la configuration du STM32 en tant que maître SPI.

## PARTIE 3 : IMPLEMENTATION DU CODE SPI2 AVEC VALIDATION PAR MESURES

**Objectif**

Écrire et tester le code de configuration du **SPI2** sur le STM32, puis valider cette configuration en mesurant et analysant les signaux SPI échangés avec le capteur ADXL345 à l'aide de l'**Analog Discovery 2**.

### ÉTAPE 1 : IMPLEMENTATION DU CODE SPI2

1. **Configurer les Broches GPIO pour le SPI2** :

2. **Initialisation du SPI2 en Mode Maître** :

    - Utilisez les paramètres de configuration définis dans la **Partie B** pour initialiser le **SPI2**. Cela inclut le mode maître, la fréquence d'horloge, et les réglages de phase et polarité (CPOL, CPHA).

### ÉTAPE 2 : TEST DE TRANSMISSION AVEC L'ANALOG DISCOVERY 2

Avant de connecter le capteur ADXL345, vous allez utiliser l'**Analog Discovery 2** pour simuler un périphérique SPI et tester la communication SPI.

1. Dans **WaveForms**, ouvrez l'outil **Protocol Analyzer** et sélectionnez **SPI**.

    - **SCK (DIO0)** : Horloge SPI.
    - **MOSI (DIO1)** : Données envoyées par le maître (STM32).
    - **MISO (DIO2)** : Données renvoyées de l'esclave (Analog Discovery 2) au maître (STM32).
    - **CS (DIO3)** : Sélection de l'esclave.

2. **Envoyer une Commande Test via SPI2** :

    - Créer un code pour envoyer une commande simple via **SPI2** et lire la réponse de l'Analog Discovery 2.

3. **Mesurer les Signaux avec l'Analog Discovery 2** :

    - Lancez la capture des signaux SPI dans **WaveForms** et observez les trames échangées entre le STM32 et l'Analog Discovery 2.
    - Vérifiez que la commande envoyée par le STM32 via **MOSI** est correcte

## ÉTAPE 3 : INITIALISATION DU CAPTEUR ADXL345

**Objectif**

Initialiser le capteur ADXL345 en configurant les registres nécessaires pour commencer à lire les données d'accélération sur les trois axes (X, Y, Z).

1. **Configurer les registres du capteur ADXL345** :
    - Utilisez les résultats obtenus lors de l'**analyse fonctionnelle** (Étape 0) pour configurer correctement les registres du capteur. Par exemple,
        - Le registre **POWER_CTL** (adresse 0x2D) doit être configuré pour activer le mode de mesure.
        - Le registre **DATA_FORMAT** (adresse 0x31) permet de configurer la plage d'accélération (±2g) et le mode SPI (3 ou 4 fils).
2. **Récupérer l'ID du Capteur** :
    - Implémentez une fonction pour lire le registre **DEVID** du capteur ADXL345 (adresse 0x00), qui contient l'ID du capteur. L'ID attendu est **0xE5**. Utiliser l'Analog Discovery pour valider les essais.

## ÉTAPE 4 : VERIFICATION DE L'ACQUISITION DES DONNEES D'ACCELERATION

**Objectif**

L'objectif de cet exercice est de lire les données d'accélération des trois axes (X, Y, Z) du capteur **ADXL345** et de vérifier que les valeurs brutes sont correctement acquises.

1. **Configurer le capteur ADXL345** :
    - Utilisez les fonctions précédemment créées pour initialiser le capteur (**POWER_CTL**, **DATA_FORMAT**) afin d'activer le mode de mesure et configurer la plage d'accélération (±2g).
    - Assurez-vous que le capteur est en mode de fonctionnement normal.
2. **Lire les données d'accélération** :
    - Créer et utilisez une fonction de lecture des registres d'accélération pour récupérer les valeurs brutes sur les axes X, Y, Z (registres **DATAX0 à DATAZ1**).
    - Affichez les valeurs brutes sur un terminal (USART2) ou utilisez des LEDs pour indiquer si la lecture est correcte.

# PARTIE 4 : CHAINE COMPLETE DE TRANSMISSION ET VISUALISATION DES DONNEES D'ACCELERATION

**Objectif**

Implémenter la chaîne complète d'acquisition, transmission, réception, traitement et visualisation des données d'accélération provenant du capteur **ADXL345**. Le microcontrôleur STM32 envoie les données brutes au PC via l'USART, où elles sont reçues, traitées et affichées avec **Processing**.

**Prérequis : S'appuyer sur le TP2**

Dans ce TP, vous allez réutiliser les concepts et les implémentations de **communication série** (USART) et de **gestion de buffer circulaire** (FIFO) étudiés dans le **TP2**. Assurez-vous de bien comprendre comment configurer l'USART et utiliser une FIFO pour stocker les données à transmettre.

## ÉTAPE 1 : ACQUISITION ET LECTURE DES DONNEES DU CAPTEUR ADXL345

1. **Initialisation du Capteur ADXL345** :
    - Utilisez le code d'initialisation du **ADXL345** (activant le mode de mesure et configurant la plage ±2g) que vous avez développé.
2. **Lecture des Données d'Accélération** :
    - Utilisez la fonction **ADXL345_ReadXYZ** pour lire les données d'accélération des trois axes (X, Y, Z) sous forme brute.

## ÉTAPE 2 : IMPLEMENTATION D'UN PROTOCOLE DE TRANSFERT DES DONNEES

**Objectif**

L'objectif de cet exercice est de permettre aux étudiants de mettre en œuvre un **protocole de transmission personnalisé** pour les données d'accélération, en utilisant un **ID étudiant unique**, un **code de session** spécifique et un **checksum** pour valider l'intégrité des données. Chaque étudiant aura un protocole distinct pour envoyer les données du capteur **ADXL345** via l'interface série **USART** du STM32 vers un PC.

### PREPARATION DU PAQUET DE DONNEES

1. **Personnalisation** : Chaque étudiant reçoit un **ID unique** et un **code de session** (fournis par l'enseignant, voir tableau excel).
2. **Champs à inclure** :
    - **ID Étudiant** : Votre identifiant unique.
    - **Code de Session** : Le code de session fourni pour cette séance.
    - **ID Capteur** : Identifiant du capteur ADXL345 (0x01).
    - **Données X, Y, Z** : Les valeurs brutes d'accélération des trois axes (X, Y, Z) lues à partir du capteur ADXL345.
    - **Checksum** : Calculé sur tous les champs sauf le checksum et l'end byte.

- **Start Byte** et **End Byte** : Indiquant le début et la fin du paquet.

3. **Exemple de Paquet** : Supposons que votre ID étudiant soit **0x03**, le code de session **0xA5**, et les données d'accélération :
    - X = 0x1234, Y = 0x5678, Z = 0x9ABC

Le paquet à envoyer sera le suivant (en hexadécimal) :

**[0x7E, 0x03, 0xA5, 0x01, 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xD5, 0x7F]**

## CALCULER LE CHECKSUM

Le **checksum** est calculé en additionnant tous les octets du paquet (sauf le checksum lui-même et l'end byte), puis en prenant le résultat **modulo 256**.

1. **Addition des octets** :
    - Start Byte : 0x7E
    - ID Étudiant : 0x03
    - Code de Session : 0xA5
    - ID Capteur : 0x01
    - Données X : 0x12 + 0x34
    - Données Y : 0x56 + 0x78
    - Données Z : 0x9A + 0xBC
2. Calcul :
    - **0x7E + 0x03 + 0xA5 + 0x01 + 0x12 + 0x34 + 0x56 + 0x78 + 0x9A + 0xBC = 0x3D6**
3. **Appliquer le Modulo 256** :
    - **0x3D6 % 256 = 0xD5**
4. **Checksum final** : **0xD5**. Ce checksum est ajouté au paquet juste avant l'end byte.

## IMPLEMENTATION SUR STM32

**Écrire le code pour assembler et envoyer le paquet de données personnalisé** :

- Utilisez votre **ID étudiant** et le **code de session** fourni par l'enseignant.
- Lisez les valeurs d'accélération (X, Y, Z) à partir du capteur **ADXL345**.
- Calculez le **checksum** sur les données du paquet

## RECEPTION ET VERIFICATION SUR PC (PROCESSING)

### OBJECTIF

L'objectif de cette partie est d'implémenter un programme **Processing** qui reçoit, vérifie et affiche les données d'accélération envoyées par le STM32 via l'USART. Chaque paquet de données contient des informations sur l'**ID étudiant**, le **code de session**, les **données d'accélération X, Y, Z**, et un **checksum** pour la validation.

### ÉTAPE 1 : CONFIGURATION DE PROCESSING POUR LA RECEPTION SERIE

1. **Installation et Configuration de Processing** :

   - **Processing** est un environnement de programmation simple pour les visuels et les applications interactives. Il possède une bibliothèque série (Serial) qui permet de lire les données provenant du port série.
   - Vous devez installer **Processing** depuis le site officiel.

2. **Ouvrir la Communication Série** :

   - Vous devez sélectionner le port série sur lequel est connecté votre STM32. Processing possède une fonction **Serial.list()** qui retourne une liste des ports disponibles.

```
3. import processing.serial.*;
4.
5. Serial myPort;
6.
7. void setup() {
8.   // Lister tous les ports série disponibles
9.   printArray(Serial.list());
10.
11.     // Sélectionner le bon port (par exemple, le premier de la liste)
12.     String portName = Serial.list()[0];
13.
14.     // Ouvrir la communication série à 9600 bps (à ajuster selon la
   configuration STM32)
15.     myPort = new Serial(this, portName, 9600);
16. }
```

## ÉTAPE 2 : RECEPTION DES DONNEES

1. **Réception des Paquets de Données** :
    - Le STM32 envoie les paquets de données contenant les champs personnalisés (ID Étudiant, Code de Session, Données X, Y, Z, Checksum).
    - Dans Processing, vous devez recevoir chaque octet un à un et le traiter au fur et à mesure.
2. **Structure du Paquet à Recevoir** : Le programme doit attendre un **Start Byte** (0x7E) pour commencer à traiter un paquet. Une fois tous les octets reçus, le programme doit calculer un **checksum** pour vérifier l'intégrité du paquet, puis afficher les données d'accélération.

## ÉTAPE 3 : TRAITER ET VERIFIER LES DONNEES

Voici le processus détaillé pour traiter les données dans Processing :

- **Start Byte (0x7E)** : Indique le début du paquet.
- **ID Étudiant, Code de Session** : Identifie l'étudiant et la session.
- **Données X, Y, Z** : Données brutes d'accélération à afficher.
- **Checksum** : Permet de valider que le paquet n'a pas été corrompu.
- **End Byte (0x7F)** : Indique la fin du paquet.

```
import processing.serial.*;

Serial myPort;
int[] accelData = new int[3];  // Pour stocker les valeurs X, Y, Z
int checksum = 0;  // Checksum calculé pendant la réception
int receivedChecksum = 0;
boolean packetStarted = false;  // Indique si on est en train de recevoir un
paquet
int index = 0;  // Indice pour les données reçues

void setup() {
  // Ouvrir la communication série (choisir le port correct)
  String portName = Serial.list()[0];
  myPort = new Serial(this, portName, 9600);
}
void draw() {
  // Vérifier s'il y a des données disponibles sur le port série
  while (myPort.available() > 0) {
    int inByte = myPort.read();  // Lire l'octet reçu

    if (inByte == 0x7E) {  // Début du paquet (Start Byte)
      packetStarted = true;
      index = 0;  // Réinitialiser l'indice pour les données
      checksum = 0;  // Réinitialiser le checksum
    } else if (inByte == 0x7F && packetStarted) {  // Fin du paquet (End Byte)
      // Comparer le checksum calculé et reçu
      if (checksum == receivedChecksum) {
        // Afficher les données si elles sont valides
        println("Données valides. X: " + accelData[0] + " Y: " + accelData[1] + "
Z: " + accelData[2]);
      } else {
        println("Erreur de checksum. Données corrompues.");
      }
      packetStarted = false;  // Réinitialiser pour le prochain paquet
    } else if (packetStarted) {
      // Traiter les autres octets du paquet
      if (index == 0) {
        // ID Étudiant (on peut choisir de l'afficher ou non)
        println("ID Étudiant : " + inByte);
      } else if (index == 1) {
        // Code de Session (on peut vérifier sa validité ici)
        println("Code de Session : " + inByte);
      } else if (index >= 2 && index <= 7) {
        // Données d'accélération X, Y, Z (16 bits par axe, donc 6 octets en
tout)
        int shift = (index % 2 == 0) ? 8 : 0;
        accelData[(index - 2) / 2] |= (inByte << shift);  // Combiner les octets
faibles et forts
      } else if (index == 8) {
        // Checksum reçu
        receivedChecksum = inByte;
      }
      // Calculer le checksum à chaque réception d'octet (sauf le checksum lui-
même)
      if (index < 8) {
        checksum += inByte;
      }
      index++;  // Passer à l'octet suivant
    }
  }
}
```

## ÉTAPE 4 : AFFICHAGE DES RESULTATS

- **Affichage des Données** : Après avoir vérifié l'intégrité des données, les résultats sont affichés dans la console **Processing** sous forme de texte. Les valeurs d'accélération X, Y, Z seront imprimées si le checksum est correct.

- **Affichage Graphique** (Optionnel) : Vous pouvez également représenter graphiquement les valeurs d'accélération en temps réel, sous forme de courbes ou de barres.

**Exemple simple d'affichage graphique** (barres représentant les valeurs X, Y, Z) :

```
void draw() {
  // Dessiner un fond blanc
  background(255);

  // Lire les valeurs d'accélération et afficher des barres
  fill(0);
  rect(50, height / 2, accelData[0] / 10, 20);  // Axe X
  rect(150, height / 2, accelData[1] / 10, 20);  // Axe Y
  rect(250, height / 2, accelData[2] / 10, 20);  // Axe Z
}
```