TP1 - Bus I2C

Objectif : Interfacer un STM32 avec des capteurs I2C

Eléments pour mettre en place le protocole I2C entre le capteur de pression et de température BMP280 et le STM32:

- Son adresses I²C est 1110111 (0x77) car le port SD0 est connecté au V_{DDIO}
- Le registre d'identification du composant est à l'adresse 0xD0 et vaut 0x58
- Le registre mode[1:0] permet de changer le mode du composant. Son adresse est 0xF4 et sa valeur est 11 en mode normal.
- L'étalonnage en pression et en température du composant sont contenu dans le registre 0xF4. Les bits 2, 3 et 4 correspondent à la pression tandis que les bits 5, 6 et 7 correspondent à la température
- Le registre 0xF7 contient les 8 bits de poids fort de la pression mesurée. 0xF8 contient les 8 bits de poids moyen. 0xF9 contient les 4 bits de poids faible (bit 7, 6, 5 et 4). La pression est formatée sur 20 bits.
- Le registre 0xFA contient les 8 bits de poids fort de la température mesurée. 0xFB contient les 8 bits de poids moyen. 0xFC contient les 4 bits de poids faible (bit 7, 6, 5 et 4). La température est formatée sur 20 bits.

Interfaçage du capteur avec le STM32 :
on choisit l'I2C1:

- Le port SDA du capteur est connecté au PB9 du STM32
- Le port SCL du capteur est connecté au PB8 du STM32
- Les masses du capteur et du STM32 sont reliées entre elles

Contenu du programme STM32:

- La fonction printf est redéfini dans le fichier stm32f4xx_hal_msp.c afin de renvoyer les caractères sur la liaison UART sur USB. Il est ainsi possible d'afficher les printf du STM32 sur l'ordinateur via un terminal série tel que putty
- Les fonctions BMP280_Write_Reg et BMP280_Read_Reg contenu dans le fichier fonctions.c permettent d'envoyer au capteur les requêtes d'écriture/de lecture des registres en protocole I2C. Ses fonctions gèrent également la réception des réponses du capteur en I2C.
- La fonction Get_Id permet de faire une lecture de la valeur du registre d'identification
- La fonction Set_config permet de placer le capteur en mode normal en modifiant la valeur du registre de configuration.

- La fonction Get_Calib permet la lecture de toutes les constantes de calibration stockées dans le capteur.
- Les fonctions Get_T et Get_P permettent la lecture de température et de pression leurs registres respectifs
- Les fonctions bmp280_compensate_T_int32 et bmp280_compensate_P_int64 utilisent les constantes de calibration et les mesures de T et de P afin de retrouver les valeurs compensées de T et de P.

TP2 - Interfaçage STM32 - Raspberry

Objectif: Permettre l'interrogation du STM32 via un Raspberry Pi Zero Wifi

Mise en route du Raspberry PI Zéro :

- Le raspberry Pi OS a été mis en place sur la carte SD de la raspberry. Nous avons modifié les partitions rootfs et boot de cette carte de façon à rendre la raspberry accessible en ssh sur notre réseau. Les modifications permettent aussi de libérer le port UART du raspberry. Nous nous connectons en ssh à la raspberry en utilisant un terminal ssh tel que putty. Nous nous servons des données contenues dans le DNS du routeur pour obtenir l'adresse IP du raspberry. Pour finaliser la connexion, nous utilisons l'identifiant "pi" et le mot de passe "raspberry".

Interfaçage du STM32 avec la raspberry :

- Le port PA0 (UART4_TX) du STM32 est connecté au port xx (RX) de la raspberry
- Le port PA1 (UART4_RX) du STM32 est connecté au port xx (TX) de la raspberry
- Les masses du STM32 et de la raspberry sont reliées entre elles

Contenu du programme STM32:

- L'UART du STM32 génère des interruptions dès qu'un caractère est reçu de la part du clavier de l'ordinateur. Ces interruptions entraînent l'appel de la fonction HAL_UART_RxCpltCallback dans le fichier main. Les passages dans cette fonction sont détectés dans le while(1) du main grâce à une variable globale (uart_it)
- Le while(1) du main contient le code permettant la mise en place d'une console. Cette console gère l'appui sur les touches DELETE et ENTER en plus de renvoyer tous les caractères reçus en écho vers l'ordinateur.
- La fonction Commande_exec du fichier fonctions.c permet d'exécuter les commandes écrites dans la console (si elles sont implémentées) après appui sur la touche ENTER.
- La fonction Commande_exec permet d'afficher l'entête au début des lignes de la console

Contenu du fichier Python comSTM32 permettant à la raspberry de communiquer avec la STM32:

- La fonction UART_WR permet d'écrire sur le port série une commande sous la forme d'une chaîne de caractère. Cette fonction attend et affiche la réponse du STM32
- Les fonctions GET_T et GET_P font partis des ordres du protocoles et utilisent simplement la fonction UART_WR avec la bonne commande en argument
- Le corps principal permet de tester les fonctions GET_T et GET_P

TP3 - Interface REST

Objectif : Développement d'une interface REST sur le Raspberry

Installation d'un serveur Python et premier fichier WEB

- Nous avons créé notre propre session sur la raspberry et avons réinstaller pip pour python3. Nous avons créé un répertoire "serveur" dans lequel nous avons placé un fichier requirement.txt
- Le premier fichier python hello.py contient le code de notre serveur

Décryptage du fichier hello.py

- Le décorateur @app.route permet de lier un URL à une fonction
- Le fragment <int:index> permet de passer un argument de type int à la fonction routé grâce à l'URL
- Nous utilisons les fonctions json.dumps ou jsonify afin de nous assurer que notre serveur répond sous forme JSON
- Nous avons implémenté la gestion des mauvaises URL avec le décorateur @app.errorhandler(404). En cas d'erreur, la page web est redirigée vers page_not_found.html. Il s'agit d'une ressource stockée dans le répertoire templates du serveur.
- La route /api/request/ nous a servi d'exemple pour comprendre la gestion des méthodes POST et GET dans le code du serveur.
- Nous avons testé la méthode POST de la fonction /api/request avec la commande curl suivante:
    - curl -X POST -H 'Content-Type: application/json' http://192.168.1.24:5000/api/request/?name="Bob" -d '{"name":"Alice","age":3}'
    - Le champ args correspond à : ?name="Bob"
    - Le champ data correspond à: -d '{"name":"Alice","age":3}'

Création d'une API CRUD. Nous avons ajouté différentes fonctions CRUD à la route api_welcome_index. Toutes les données sont fournies dans le champ args ! (pas dans data):

- Méthode POST sans index. Nous changeons la phrase stockée dans la variable welcome du serveur. La nouvelle phrase est donnée en argument de la requête URL. Nous retournons en sortie la nouvelle phrase en format JSON
- Méthode GET sans index. Nous retournons en sortie la phrase en format JSON
- Méthode GET avec l'index entier i . Nous retournons en sortie le ième caractère de la phrase en format JSON

- Méthode PUT avec l’index entier i. Nous changeons la phrase en introduisant un mot à la ième position. Le mot est donné en argument de la requête URL. Nous retournons en sortie la nouvelle phrase en format JSON
- Méthode PATCH avec l’index entier i. Nous changeons la phrase en remplaçant la ième lettre de la phrase. La nouvelle lettre donnée en argument de la requête URL. Nous retournons en sortie la nouvelle phrase en format JSON
- Méthode DELETE avec l’index entier i. Nous changeons la phrase en supprimant la ième lettre de la phrase. Nous retournons en sortie la nouvelle phrase en format JSON
- Méthode DELETE sans index. Nous supprimons la phrase. Nous retournons en sortie la chaîne de caractère vide en format JSON.

TP4 - Bus CAN

Objectif : Interfacer le STM32 avec le moteur pas à pas via un Tranceiver CAN

Afin de faciliter l'insertion du composant dans le montage, on peut l'insérer sur la carte nucléo64. Cependant, les broches du CAN se retrouvent sur celle de l'I2C. On va alors changer l'I2C de place. On va choisir l'I2C2 au lieu de l'I2C1 (SDA -->PB11, SCL-->PB10).
On modifie donc le code dans fonction.h (extern I2C_HandleTypeDef hi2c**1**; -->extern I2C_HandleTypeDef hi2c**2**;) .
Puis dans tout le code on remplace hi2c1 par hi2c2.

Interfaçage du STM32 avec la carte STM32L476:

- Le port PB9 (CAN1_TX) du STM32 est connecté au port RX de la carte STM32L476
- Le port PB8(CAN1_RX) du STM32 est connecté au port TX de la carte STM32L476

On vient connecter le moteur pas à pas via une carte fille que l'on insère sur la nucléo64. On alimente le moteur en 12V via une alimentation externe.

Eléments pour mettre en place le protocole CAN entre la carte STM32L476 et le STM32:

- Il faut configurer une structure qui permet de configurer le header du CAN:

```
CAN_TxHeaderTypeDef headerCAN;   //on définit le nom de la structure
 headerCAN.StdId = 0x61;                 //adresse pour n'envoyer qu'un angle et le sens
 headerCAN.ExtId =0;                     //on n'utilise pas le mode étendu
 headerCAN.IDE =CAN_ID_STD;      //la trame est standard
 headerCAN.RTR =CAN_RTR_DATA; //la trame est standard
 headerCAN.DLC =2;                       //on envoi 2 octets de données
 headerCAN.TransmitGlobalTime =DISABLE;
```

TP5 - Integration 12C - Serial - REST - CAN

Création d'une API CRUD dans le fichier API_REST.c. . Nous avons implémenté différentes fonctions CRUD. Toutes les données sont fournies dans le champ args ! (pas dans data):

- Méthode POST sans index dans temp/ : Nous interrogeons le STM32 via le port série pour obtenir la mesure instantanée de T. Cette méthode ne fonctionne pas car nous n'avons pas réussi à faire fonctionner le port série dans le code du serveur.
- Méthode POST sans index dans pres/ : Nous interrogeons le STM32 via le port série pour obtenir la mesure instantanée de P. Cette méthode ne fonctionne pas non plus.
- Méthode GET sans index dans temp/. Nous lisons toutes les valeurs précédentes de T. Elles sont stockées dans le serveur. En ajoutant un index, la méthode retourne une seul valeur
- Méthode GET sans index dans temp/. Nous lisons toutes les valeurs précédentes de P. Elles sont stockées dans le serveur. En ajoutant un index, la méthode retourne une seul valeur
- Méthode GET sans index dans scale/. Nous lisons la valeur de K stockée dans le serveur.
- Méthode GET sans index dans angle/. Nous lisons la valeur de K stockée dans le serveur ainsi que la valeur de T précédente et nous les multiplions pour obtenir l'angle.
- Méthode POST dans scale/. Nous envoyons la nouvelle valeur de K vers la STM32. K est passé en index de l'URL. Cette méthode ne fonctionne pas car nous n'avons pas réussi à faire fonctionner le port série dans le code du serveur.
- Méthode DELETE dans temp/. Nous supprimons la température contenu à la position index du tableau du serveur.
- Méthode DELETE dans pres/. Nous supprimons la pression contenu à la position index du tableau du serveur.