V.DEROUET 2024 - IAI - PROJET µC

Objectif : Utiliser et concevoir un programme en C pour la plateforme Open1768

Sommaire

1. Configuration des broches du LPC1768
2. Configuration de l'I2C0
    - 2.1 Combien de mots contient cette mémoire et quelle est la taille des mots ?
    - 2.2 L'interface de la mémoire est un bus I2C
    - 2.3 Décrivez précisément les trames à envoyer pour écrire un mot en mémoire
3. Initilisation du contrôleur I2C
    - 3.1 Vérifiez si ce périphérique I2C0 est alimenté par défaut
    - 3.2 Décidez de la vitesse à laquelle on va faire fonctionner ce bus
    - 3.3 Trouvez également dans ce fichier la fonction qui permet d'autoriser des échanges I2C0
    - 3.4 Ecrivez une procédure init_i2c_eeprom() qui fait l'initialisation
4. Ecriture dans la mémoire
    - 4.1 Quels sont les paramètres à passer à cette fonction ?
    - 4.2 Quelles valeurs mettre sur ces paramètres pour faire l'écriture d'un mot dans notre mémoire ?
    - 4.3 Ecrivez une procédure i2c_eeprom_write
    - 4.4 Compilez maintenant pour tourner sur la carte
    - 4.5 Ecrire une fonction i2c_eeprom_read
5. Description du projet et des fonctionnalités
    - 5.1 Vue d'ensemble
    - 5.2 Fonctionnalités
    - 5.3 Actions Utilisateur
    - 5.4 Périphériques Utilisés
    - 5.5 Conclusion

1. Configuration des broches du LPC1768.

Voici le fichier `gestion_Pinsel.c`.
Cela permet d'initialiser les boutons, la mémoire i2c de type **FM24CL16**.

```c
#include "gestion_Pinsel.h"

#include "lpc17xx_pinsel.h"
#include "lpc17xx_gpio.h"

void pin_Configuration(void) {
    //La reset value des pinsels sont à 00, qui correspond à
l'initialisation
    //des pin au GPIO

    // Broches gestion de la mémoire


    PINSEL_CFG_Type configPinsel1;
    configPinsel1.Portnum = PINSEL_PORT_0;
    configPinsel1.Pinnum = PINSEL_PIN_27;
    configPinsel1.Funcnum = PINSEL_FUNC_1;
    configPinsel1.Pinmode = 1;
    configPinsel1.OpenDrain = PINSEL_PINMODE_OPENDRAIN;
    PINSEL_ConfigPin(&configPinsel1);

    configPinsel1.Portnum = PINSEL_PORT_0;
    configPinsel1.Pinnum = PINSEL_PIN_28;
    configPinsel1.Funcnum = PINSEL_FUNC_1;
    configPinsel1.Pinmode = 1; // deja une resistance de pull up dans les
circuits
    configPinsel1.OpenDrain = PINSEL_PINMODE_OPENDRAIN;
    PINSEL_ConfigPin(&configPinsel1);

    // Boutons

    GPIO_SetDir (2,(1<<10),0);  /* PORT2.10 defined as input        */
    GPIO_SetDir (2,(1<<11),0);  /* PORT2.11 defined as input        */

}
```

2. Configuration de l’I2C0.

2.1 Combien de mots contient cette mémoire et quelle est la taille des mots ? Déduisez-en combien de bits d’adresse sont donc nécessaires pour adresser un mot ?

La mémoire **FM24CL16** contient 2048 mots de 8 bits chacun. Pour adresser 2048 mots, nous avons besoin de 11 bits d'adresse.

Calcul:

$$2^{11} = 2048$$

Donc, **11** bits d'adresse sont nécessaires pour accéder à chaque mot de la mémoire.

2.2 L'interface de la mémoire est un bus I2C. Déduisez-en combien de signaux composent un bus I2C, comment se fait un échange sur ce bus. Qui sera le maitre du bus dans notre cas ? Qu'elle est l'adresse esclave I2C à utiliser pour notre mémoire ?

Un bus I2C se compose de deux signaux :

1. **SDA** (Serial Data Line): Ligne de données série.
2. **SCL** (Serial Clock Line): Ligne de l'horloge série.

- Échange sur le bus I2C

L'échange sur le bus I2C se fait via un protocole maître-esclave :

1. Le maître génère les signaux d'horloge et initie la communication.
2. L'esclave répond aux commandes du maître.

- Maître du bus dans notre cas

Dans notre application, le maître du bus sera le microcontrôleur LPC1768.

- Adresse esclave I2C pour notre mémoire

L'adresse esclave I2C de base pour le **FM24CL16** est $1010b$ (binaire) suivi de trois bits pour la sélection de page, puis un bit de lecture/écriture (R/W).

- **Adresse de base pour l'écriture :** $0xA0$ (pour 1010 0000 en binaire, où le dernier 0 indique l'écriture).
- **Adresse de base pour la lecture** : $0xA1$ (pour 1010 0001 en binaire, où le dernier 1 indique la lecture).

2.3 Décrivez précisément les trames à envoyer pour écrire un mot en mémoire, pour lire un mot en mémoire : combien d'octets vont être envoyés, quel est le contenu du 1er octet, du 2ème...

$\Rightarrow$ **Écriture d'un mot en mémoire**

Pour écrire un mot (1 octet) dans la mémoire FM24CL16, la trame I2C doit inclure l'adresse esclave, l'adresse interne de la mémoire, et la donnée à écrire.

$\rightarrow$ Trame d'écriture

1. **Condition de Start**: Initiée par le maître.
2. **Adresse esclave**:

- 7 bits pour le type d'appareil (`1010`), bits de sélection de page (`000` si nous utilisons la première page), et bit de lecture/écriture (`0` pour l'écriture).
- Adresse complète pour l'écriture : `0xA0` (`1010 0000` en binaire).

3. **Adresse interne**:
   - 1 octet pour l'adresse interne (8 bits pour spécifier une adresse dans une page de 256 octets).
4. **Donnée à écrire**:
   - 1 octet de donnée.

→ Contenu des octets

1. **Premier octet**: Adresse esclave (`0xA0`).
2. **Deuxième octet**: Adresse interne (8 bits, par exemple, `0x00` pour la première adresse).
3. **Troisième octet**: Donnée à écrire (par exemple, `0x5A`).

→ Exemple de trame d'écriture

Octet	Contenu	Description
1er octet	`0xA0`	Adresse esclave (écriture)
2ème octet	`0x00`	Adresse interne
3ème octet	`0x5A`	Donnée à écrire

⇒ Lecture d'un mot en mémoire

Pour lire un mot (1 octet) dans la mémoire FM24CL16, la trame I2C doit inclure l'adresse esclave, l'adresse interne de la mémoire pour la sélection, suivie d'une nouvelle condition de start, puis l'adresse esclave avec le bit de lecture.

⇒ **Trame de lecture**

1. **Condition de Start**: Initiée par le maître.
2. **Adresse esclave**:
   - 7 bits pour le type d'appareil (`1010`), bits de sélection de page (`000` si nous utilisons la première page), et bit de lecture/écriture (`0` pour l'écriture).
   - Adresse complète pour l'écriture : `0xA0` (`1010 0000` en binaire).
3. **Adresse interne**:
   - 1 octet pour l'adresse interne (8 bits pour spécifier une adresse dans une page de 256 octets).
4. **Condition de redémarrage (Repeated Start)**: Initiée par le maître.
5. **Adresse esclave**:

- 7 bits pour le type d'appareil (`1010`), bits de sélection de page (`000` si nous utilisons la première page), et bit de lecture/écriture (`1` pour la lecture).
- Adresse complète pour la lecture : `0xA1` (`1010 0001` en binaire).

6. **Donnée lue**:
   - 1 octet de donnée lue de la mémoire.

→ Contenu des octets

1. **Premier octet**: Adresse esclave (`0xA0`).
2. **Deuxième octet**: Adresse interne (8 bits, par exemple, `0x00` pour la première adresse).
3. **Repeated Start**: Condition de redémarrage.
4. **Troisième octet**: Adresse esclave (`0xA1`).
5. **Quatrième octet**: Donnée lue.

→ Exemple de trame de lecture

Octet	Contenu	Description
1er octet	`0xA0`	Adresse esclave (écriture)
2ème octet	`0x00`	Adresse interne
Repeated Start		Condition de redémarrage
3ème octet	`0xA1`	Adresse esclave (lecture)
4ème octet	`0x5A`	Donnée lue

3. Initilisation du contrôleur I2C

3.1 Vérifiez si ce périphérique I2C0 est alimenté par défaut (Power control PCONP registre). Si ce n'est pas le cas, trouvez comment l'alimenter.

- En mode debug -> Peripherals -> Clocking and power control -> power control
  - Pour l'alimenter : il faut dans le registre **ISER0** enable : **ISE_I2C0**

3.2 Décidez de la vitesse à laquelle on va faire fonctionner ce bus. Trouvez dans le fichier lpc17xx_i2c.c la fonction qui permet de programmer cela.

Nous avons choisi de travailler avec un bus va fonctionner à la vitesse de $100kHz$ (doc page 428)

- gestion de la vitesse de la memoire :

```
static void I2C_SetClock (LPC_I2C_TypeDef *I2Cx, uint32_t target_clock)
```

3.3 Trouvez également dans ce fichier la fonction qui permet d'autoriser des échanges I2C0.

- La fonction qui permet d'autoriser les echanges i2C :

```
    void I2C_Cmd(LPC_I2C_TypeDef* I2Cx, FunctionalState NewState)
```

3.4 Ecrivez une procédure init_i2c_eeprom() qui fait l'initialisation.

- Initialiser les transferts i2C : init -> setclock -> cmd

```
void init_i2c_eeprom() {
    I2C_Init(LPC_I2C0, 100000);
    I2C_Cmd(LPC_I2C0, ENABLE);
}
```

4. Ecriture dans la mémoire

4.1 Quels sont les paramètres à passer à cette fonction ?

```
Status I2C_MasterTransferData(LPC_I2C_TypeDef *I2Cx, I2C_M_SETUP_Type
*TransferCfg, I2C_TRANSFER_OPT_Type Opt);
```

4.2 Quelles valeurs mettre sur ces paramètres pour faire l'écriture d'un mot dans notre mémoire ? NB : attention le buffer contenant les données doit être déclaré en global.

- Valeurs à mettre :
  - LPC_I2C0 pour le periph
  - Une structure de type I2C_M_SETUP_Type
  - I2C_TRANSFER_POLLING pour indique si on est en polling mode
  - La structure I2C_M_SETUP_Type nous indiques "Définitions de structure de données de configuration de transfert d'esclave"

4.3 Ecrivez une procédure i2c_eeprom_write(uint16_t addr, void* data, int length) qui permet d'écrire dans la mémoire un tableau de donnée de taille length à partir de l'adresse mémoire addr. NB : le type « void » a été utilisé pour vous laisser décider du type le plus approprié ; à vous de le changer. Rappel : ne pas confondre adresse esclave I2C et adresse où on veut écrire dans la mémoire.

```
void i2c_eeprom_write(void *data, int length)
{
    I2C_M_SETUP_Type i2cSetup;

    i2cSetup.sl_addr7bit = EEPROM_I2C_ADDR;
    i2cSetup.tx_data = (uint8_t*)data;
  i2cSetup.tx_length = length; //longueur des données
  i2cSetup.rx_data = NULL;
  i2cSetup.rx_length = 0;
    i2cSetup.retransmissions_max = 10;
```

```
    i2cSetup.callback = NULL;

    I2C_MasterTransferData(LPC_I2C0, &i2cSetup, I2C_TRANSFER_POLLING);
}
```

Utilisation :

```
i2c_eeprom_write(data, sizeof(data));
```

4.4 Compilez maintenant pour tourner sur la carte (baguette magique/ onglet Debug et cocher « use jlink »). Recompilez et chargez le code sur la carte. Utilisez une carte Digilent Explorer pour visualiser le bus I2C.

Capture d'écan du logiciel une fois l'analyseur de signaux connectés à la mémoire.

Ecriture :

```
  uint8_t data[2] = {0x01,0x30};
  i2c_eeprom_write(&data, sizeof(data));
```

Lecture :

```
  i2c_eeprom_read(0x01, &dataRecupere, 1);
  sprintf(chaine, "Read: 0x%02X", dataRecupere); //pour l'afficher
```

4.5 Ecrire une fonction i2c_eeprom_read(uint16_t addr, void* data, int length) qui permet de lire dans la mémoire un bloc de donnée de taille length à l'adresse addr.

```
// read dans la mémoire i2c
void i2c_eeprom_read(uint8_t add_case, void *data_recup, int length)
{
    I2C_M_SETUP_Type setupi2c;

    setupi2c.sl_addr7bit = EEPROM_I2C_ADDR;
    setupi2c.tx_data = &add_case;
    setupi2c.tx_length = 1;
    setupi2c.rx_data = data_recup;
    setupi2c.rx_length = length;
    setupi2c.retransmissions_max = 10;
    setupi2c.retransmissions_count = 0;
    setupi2c.tx_count = 0;
    setupi2c.rx_count = 0;
```

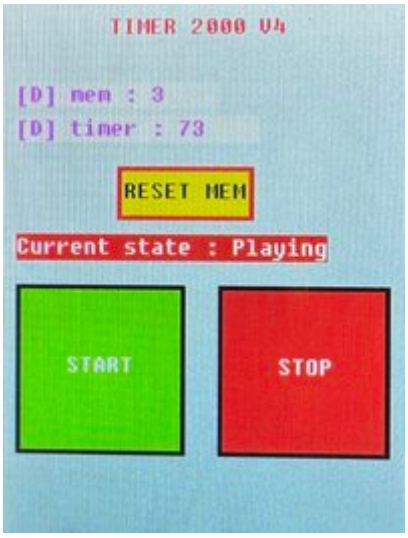
```
    setupi2c.callback = NULL;

    I2C_MasterTransferData(LPC_I2C0, &setupi2c, I2C_TRANSFER_POLLING);
//mode POLLING
}
```

Utilisation :

```
i2c_eeprom_read(addr_data[0], &data_recup, sizeof(data_recup)); // Lire à partir de l'adresse spécifiée
```

5. Description du projet et des fonctionnalités



5.1 Vue d'ensemble

Ce projet est développé pour la plateforme **Open1768**, qui intègre un microcontrôleur **LPC1768**.

⚠ Attention le programme n'est pas conçu pour stocker des valeurs au dessus de 255.

5.2 Fonctionnalités

1. **Gestion du Timer** :
    - Un compteur est géré par un timer matériel, configuré pour une précision de `2500 ticks` et une demi-période de `100 ms`.
    - Le compteur est incrémenté à chaque période définie par le timer, avec des actions spéciales lors de la gestion de l'interruption du timer.
2. **Interface Utilisateur Graphique** :
    - Un écran tactile LCD permet de démarrer, arrêter ou réinitialiser le compteur, ainsi que d'effacer la mémoire EEPROM.
    - Des zones spécifiques sur l'écran tactile sont définies pour chaque action : START, STOP et RESET MEM.

- Les actions de l'utilisateur sur l'écran tactile sont interprétées et déclenchent des changements dans l'état du compteur ou des opérations sur la mémoire.

3. **Stockage Persistant avec EEPROM** :
   - Les valeurs du compteur peuvent être sauvegardées dans une mémoire EEPROM externe via une interface I2C.
   - Le projet inclut des fonctions pour écrire et lire des données depuis cette mémoire EEPROM, permettant de préserver l'état du compteur entre les redémarrages.

5.3 Actions Utilisateur

1. **Démarrage du Compteur** :
   - L'utilisateur peut appuyer sur la zone "START" de l'écran tactile pour démarrer ou redémarrer le compteur.
   - Cette action réinitialise le processus et affiche l'état actuel sur l'écran.
2. **Arrêt du Compteur** :
   - En appuyant sur la zone "STOP" de l'écran tactile, l'utilisateur peut arrêter le compteur.
   - L'état du compteur est conservé et affiché à l'écran, et il peut être réinitialisé si nécessaire en rappuyant sur "STOP".
3. **Réinitialisation de la Mémoire** :
   - La zone "RESET MEM" sur l'écran tactile permet d'effacer le contenu de l'EEPROM.
   - Cette action remet à zéro les données stockées dans la mémoire, préparant ainsi le système pour une nouvelle session.
4. **Sauvegarde Automatique** :
   - Un bouton physique est utilisé pour sauvegarder manuellement la valeur actuelle du compteur dans l'EEPROM.

5.4 Périphériques Utilisés

1. **Écran Tactile LCD** :
   - L'écran tactile est utilisé pour l'affichage et l'interaction avec l'utilisateur.
   - Il fournit des boutons graphiques pour démarrer, arrêter et réinitialiser les actions.
2. **Bouton Physique** :
   - Un bouton externe est utilisé pour déclencher la sauvegarde manuelle de la valeur actuelle du compteur dans l'EEPROM. (KEY 2 sur la platine)
3. **EEPROM via I2C** :
   - La mémoire EEPROM externe est connectée via un bus I2C pour stocker et récupérer les valeurs du compteur.

- Les opérations de lecture et d'écriture sont gérées par le protocole I2C intégré dans le microcontrôleur.

4. **Timer Matériel** :
   - Un timer matériel intégré dans le LPC1768 est utilisé pour gérer l'incrémentation du compteur à des intervalles réguliers.

5.5 Conclusion

Ce projet combine plusieurs aspects de la programmation embarquée, y compris la gestion des périphériques, l'interfaçage avec des composants externes via I2C, et l'interaction avec les utilisateurs via un écran tactile.