Institut Universitaire de Technologie Saint-Étienne

télécom saint-étienne
école d'ingénieurs nouvelles technologies

Université Jean Monnet Saint-Étienne

Compte Rendu de TP d'Info2

Séance 1

Réalisé par : Noam Hache

Encadré par : Laetitia Belguermi

Chapitre 1

La carte de développement

1 Expliquez quelle est la différence entre le registre PTBD et le registre PTBDD

Le registre `PTBDD` est un registre qui contrôle les pins du port B. Il décrit les pins du port B qui seront en lecture ou en écriture (en mode entrée ou sortie). Un bit à 0 dans ce registre indique un pin d'entrée, un bit à 1 indique un pin de sortie.

Le registre `PTBD` quant à lui est le registre de données du port B. Pour les pins en mode sortie, la dernière valeur du registre (au pin correspondant) sera envoyée sur le pin en question. Pour les pins en mode entrée, la valeur du registre (au pin correspondant) sera mise à jour avec la valeur lue du pin en question.

Le registre `PTBD` est donc un registre de donnée alors que le registre `PTBDD` est un registre de contrôle.

2 Vérifiez que les capacités annoncées sont correctes en utilisant les valeurs des plages d'adresses pour la RAM et la FLASH.

La mémoire `RAM` se trouve entre les adresses `0x0060` et `0x025F`. On peut alors calculer la différence des adresses pour trouver la capacité mémoire :

$$025F_{(16)} - 0060_{(16)} + 1 = 200_{(16)} = 512_{(10)}$$

La taille d'adresse étant de 8 bits, on a 512 octets de mémoire `RAM`.

La mémoire `FLASH` se trouve entre les adresses `0xE000` et `0xFFFF`. Selon le même raisonnement :

$$FFFF_{(16)} - E000_{(16)} + 1 = 2000_{(16)} = 8192_{(10)}$$

On a 8192 octets de mémoire `FLASH`.

3 Expliquez pourquoi sur cette carte, le port B doit être configuré en sortie et le port A en entrée.

Selon le schéma de câblage de la carte, on voit que le port B est relié aux différentes diodes de la carte ; on ne peut alors pas recevoir d'informations de celles-ci, elles servent uniquement à afficher une information. Ainsi le port B doit être en mode sortie sur tout ses pins.

Le port A quant à lui est relié au potentiomètre, au bouton poussoir 1, aux deux interrupteurs et au connecteur 6 broches. Aucun de ces composants ne servent à afficher une information mais à en envoyer. Le port A doit donc être un port d'entrée qui reçoit les informations de ces composants. De plus le port A est relié au connecteur 6 broches qui sert au transfert d'informations entre le microcontrôleur et un ordinateur, c'est donc d'autant plus logique que ce port soit en entrée.

Chapitre 2

CodeWarrior en mode Full Chip Simulation

4

L'instruction `__RESET_WATCHDOG` réinitialise le Watchdog afin que son comptage reprenne à 0.
Il est présent en début de la boucle `for(;;)` car on veut réinitialiser le Watchdog au démarrage du programme. Si il n'était pas réinitialisé, il aurait potentiellement une valeur depuis une execution précédente ; si il avait bien une valeur, le watchdog pourrait potentiellement réinitialiser le programme principal pendant son execution même si aucun problème n'est présent, uniquement parce-que le compteur du watchdog avait déjà progressé et n'avait pas été réinitialisé.

5

Les instruction `#include<stdio.h>` et `printf` et `scanf` ne sont pas présentes dans le programme car elles servent à interagir avec l'utilisateur via le terminal (l'invite de commande / la console). Or dans nôtre cas, nous n'utilisons pas le terminal pour gérer les entrées/sorties, mais les composants de la carte. Ainsi nous n'avons pas besoin des fonctions citées plus haut.

6 Déclaration des variables représentant les ports du microcontrôleur

6.1

La structure de donnée utilisée pour représenter les registres des ports du microcontrôleur est un `struct` (Type composé).

6.2

Ces registres sont déclarés avec une union de `structs` dans lesquels chaque bit du registre correspond à un byte, nommé `PTAD` de 0 jusqu'a 5.

```
/*** PTAD - Port A Data Register; 0x00000000 ***/
typedef union {
    byte Byte;
    struct {
        byte PTAD0       :1;                /* Port A Data Register Bit 0 */
        byte PTAD1       :1;                /* Port A Data Register Bit 1 */
        byte PTAD2       :1;                /* Port A Data Register Bit 2 */
        byte PTAD3       :1;                /* Port A Data Register Bit 3 */
        byte PTAD4       :1;                /* Port A Data Register Bit 4 */
        byte PTAD5       :1;                /* Port A Data Register Bit 5 */
        byte             :1;
        byte             :1;
    } Bits;
    struct {
        byte grpPTAD :6;
        byte         :1;
        byte         :1;
    } MergedBits;
} PTADSTR;
```

FIGURE 1 – extrait de la déclaration du registre PTDA

6.3

Le mot clé `extern` sert à définir une variable accessible par toutes les fonctions du code.

> « On peut également définir des variables *externes* à toutes les fonctions, c'est-à-dire des variables auxquelles toutes les fonctions peuvent accéder par leur nom. »
>
> — Brian W. Kernighan & Dennis M. Ritchie, *Le Langage C Norme ANSI 2e édition*, trad. Jean-François Grotf, Éric Mottier et Étienne Alard, 2023, Dunod, p. 31

Le mot clé `volatile` quant à lui sert à définir une variable qui peut changer à tout moment sans l'action du code lui même.

6.4

Le lien entre l'adresse des registres du port B et les variables représentant son registre de données et de direction est fait grâce à ces instructions

```
extern volatile PTBDSTR _PTBD @0x00000002;
extern volatile PTBDDSTR _PTBDD @0x00000003;
```

6.5

Comme vu dans la figure 1, les bits 0 à 5 sont nommés et donc accessibles dans le code, alors que les bits 6 et 7 ne sont pas nommés et donc inaccessibles.

7

7.1

Pour tester si le bit d'indice 4 du registre `PTBD` est à 1, il faudrait utiliser le masque suivant sur le registre :

```
PTBD & 0x20
```

7.2

L'instruction `#define PTBD_PTBD0 _PTBD.Bits.PTBD0` sert à simplifier l'accès au bit d'indice 0 du registre et à ne pas avoir à écrire toute l'instruction de droite à chaque fois que l'on souhaite accéder à ce bit.

7.3

Alternativement à la question 7.1 on peut utiliser l'instruction suivante :

```
if (PTBD_PTBD4 == 0b1) {
    ...
}
```