CEG 3536 – Architecture d'ordinateur II
Labo 3 – L'interface au matériel – afficheur à 7 segments/LCD

Objectifs:

Introduire l'interface du matériel au Motorola 9S12DG256 en réalisation une application d'un dispositif d'affichage à 7 segments et l'interface à un afficheur à cristaux liquides (LCD).

Préparation:

- Des notes sont affectées à votre travail de préparation.
    - Préparez une ébauche du rapport de laboratoire selon les directives à la fin de ce document. L'ébauche devra être un document Word complet avec page couverture et sections du rapport.
    - Concevez votre système (matériel et logiciel) selon les directives à la fin de ce document, assurez-vous de refléter votre conception dans votre ébauche du rapport.
    - Codez et corrigez vos modules et intégrez-les dans le projet CodeWarrior du système d'alarme. Apportez le projet mis à jour sur une clef USB ou votre portable.
- Montrez votre travail de préparation (ébauche du rapport et le projet CodeWarrior compilé) à l'AE du lab dès votre arrivée à la session du labo.

Équipement utilisé:

- Windows PC (avec CodeWarrior installé)
- Carte d'entrainement Dragon-12

Description:

Dans ce laboratoire vous intégrez les afficheurs de 7 segments et le LCD de la carte Dragon-12 dans le projet de système d'alarme (un projet C CodeWarrior). En plus, vous allez intégrer les modules assembleur *Keypad* et *Délai* du labo 2 dans le projet CodeWarrior. Les afficheurs à 7-segments sont utilisés pour montrer de décompte lorsque le système est armé et désarmé; il sert aussi à montre un déclenchement d'alarme. L'afficheur LCD permet d'afficher les mêmes chaînes de caractères qui apparaissaient dans la fenêtre terminale du MiniIDE. Le clavier sert à capter l'entrée de l'utilisateur. Pour les détails de l'organisation des afficheurs à 7-segments et de l'afficheur LCD de la carte Dragon-12, consulter la documentation Dragon-12, et les sections 4.3 et 4.5 du texte Cady. Un document PDF donne un aperçu de l'interface avec le LCD. La matière du tutorat 4 et du tutorat 5 peut aider aussi. Dans de labo, toute interaction avec l'utilisateur se fait avec le matériel de la carte Dragon-12.

Le logiciel du labo

Introduction

Le projet CodeWarrior est fourni pour le système d'alarme. Il intègre les afficheurs à 7-segment et l'afficheur LCD. Les modules du projet logiciel sont donnés ci-dessous.

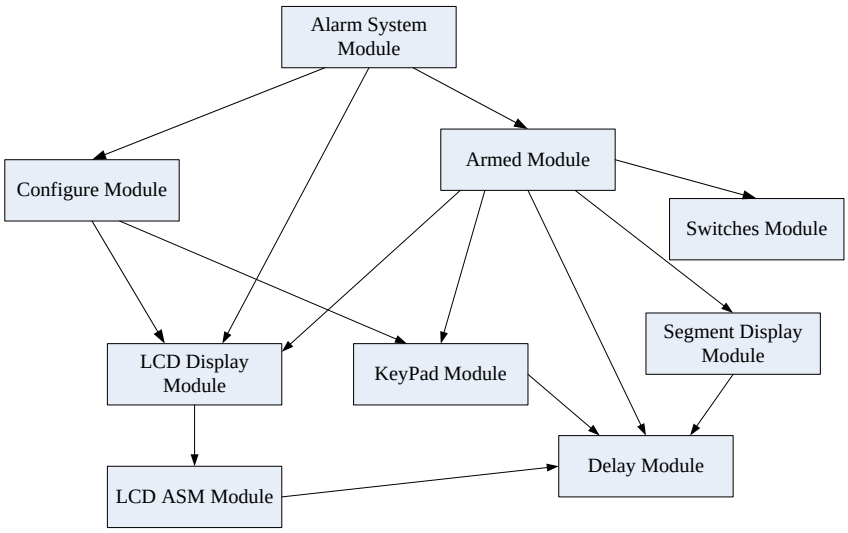

Figure 2 – Modules de logiciel du système d'alarme

Vous devez être déjà familier avec les modules utilisés dans les labos antérieurs:

- *Module Alarm System* (fichiers: alarm.c/alarm.h/alarmextern.h): Ce module C donnent les mêmes fonctions que le module assembleur équivalent des labos antérieurs. La nouveauté dans ce module est les appels au module *LCD Display* pour afficher les messages sur l'afficheur LCD au lieu des appels aux routines Debug-12.
- *Module Config* (fichiers: config.c/config.h): Ce module donnent C les mêmes fonctions que le module assembleur équivalent des labos antérieurs, mais il appels le module *LCD Display* pour afficher des messages sur l'afficheur LCD.
- *Module Armed* (fichiers: armed.c/armed.h): La majorité de la logique de ce module C est pareil à celle du module assembleur des labos antérieurs. Les messages sont affichés sur l'afficheur LCD. Le module utilise aussi les afficheurs à 7-segment pour afficher un compte à rebours (10, 9, 8, …) lorsque le système est armé et lorsque le système est désarmé (à l'ouverture de la porte avant). L'afficheur à 7-segment est aussi utilisé pour afficher un déclenchement d'alarme (clignote un A sur l'afficheur le plus à la gauche). Notez que ces dernières fonctions sont réalisées avec des appels au module *Segment Display*.
- *Module Switches* (fichiers: switches.c/switches.h): Ce module est la version C du module assembleur *Switches* offert dans le labo 2.
- Module LCD ASM (files: lcd.asm/lcd_asm.h): Ce module assembleur offre des fonctions de bas-niveau pour manipuler l'afficheur LCD. Ce module vous est fourni. Voyez la section de conception pour les détails des fonctions offertes par ce module.

Vous devez fournir les modules suivants. Les deux premiers sont des modules assembleurs apportés du labo 2, tandis que les deux autres sont des modules C que vous devez développer. La dernière page de ce document vous donne des consignes pour intégrer les modules assembleur dans un projet C.

- *Module Keypad* (fichiers: keypad.asm/keypad_asm.h): Ce module est le module Keypad du labo 2. Le fichier en-tête *keypad_asm.h* donne les prototypes de fonctions pour le module (ce fichier est fourni) et le fichier *keypad.asm* du labo 2 doit être ajouté au projet.

- *Module Delay* (fichiers: delay.asm, delay_asm.h): Ce module est le module *delay* du labo 2. Le fichier en-tête *delay_asm.h* donne les prototypes de fonctions pour le module (ce fichier est fourni) et le fichier *delay.asm* du labo 2 doit être ajouté au projet.
- *Module LCD Display* (fichiers: lcdDisp.c/lcdDisp.h): Ce module C offre les fonctions pour afficher des messages sur l'afficheur LCD. Il utlise des fonctions de bas-niveau disponibles dans le module *LCD ASM*. Le fichier d'en-tête est fourni et le gabarit *lcdDisp.c* doit être complété.
- *Module Segment Display* (fichiers: segDisp.c/segDisp.h): Ce module C offre des fonctions pour afficher des caractères sur les afficheurs à 7-segment. Le fichier d'en-tête est fourni et le gabarit *segDisp.c* doit être complété.

Conception de logiciel

1) Module assembleurs

Les modules assembleurs du laboratoire 2 sont intégrés dans le projet C CodeWarrior en créant un fichier d'en-tête qui donne les prototypes de fonctions C au modules (ceux-ci sont fournis) et ensuite modifier les fichiers assembleurs tel que décrit à la dernière page de ce document.

Module KeyPad : Les sous-programmes de ce module assembleur sont appelés comme des fonctions C. Rappelez-vous que le module du laboratoire 2 contient un sous-programme pour initialiser le matériel (c.-à-d. le Port A) branché au clavier et un sous-programme qui permet de détecter une clef pressée (il retourne la valeur ASCII de la clef pressée) et un sous-programme qui permet d'interroger le clavier. Voyez le fichier *keypad_asm.h*.

Module Delay : Contient les sous-programmes assembleurs pour la création de délais (les sous-programmes sont appelés comme des fonctions C). Les sous-programmes seront appelés par divers modules. Vous devez possiblement modifier ce module pour répondre aux besoins du module *Segment Display*. Voir le fichier *delay_asm.h*.

2) Modules C

Module Segment Display*:* Le fichier *SegDisp.c* contient le code du module *Segment Display*. Ce module devra offrir les fonctions suivantes :

`void initDisp(void)`*:* Initialise le matériel (port B et port P) branché aux afficheurs à 7-segments. Il devra aussi initialiser les afficheurs en blanc (aucun segment allumé).

`void setCharDisplay(char, byte )`: Une fonction qui ajoute un caractère (identifier avec son code ASCII) à afficher. Réservez 4 octets en mémoire (un tableau) pour contenir soit des caractères ou codes pour afficher les caractères sur les afficheurs correspondants. Lorsque la fonction est appelée, deux arguments sont fournies, le caractère à afficher (premier argument) et un numéro d'afficheur (débute à 0) pour indiquer sur lequel des afficheurs le caractère doit apparaitre.

`void segDisp(void)`: Une fonction qui met à jour les afficheurs pour une période de temps (utilisez 100 ms). Ceci permet de la fonction appelante de regagner le contrôle périodiquement pour lui permettre de compléter d'autres tâches tel que la vérification du clavier.

`void clearDisp(void)`: Cette fonction devra mettre les afficheurs en blanc (éteindre tous les segments).

Module LCD Display**:** Ce module fait l'utilisation des fonctions fournies par le module *LCD ASM* (voir ci-dessous). Il offre les fonctions suivantes aux autres modules pour l'affichage des chaines de caractères (messages) sur l'afficheur LCD.

`void initLCD(void)`: Initialise l'afficheur LCD (cette fonction fait un appel à la fonction `lcd_init()` fournie par le module *LCD ASM*).

`void printLCDStr(char *, byte)`: Cette fonction affiche une chaine sur une ou deux lignes de l'afficheur LCD. L'adresse de la chaîne à être afficher est passée dans le premier argument tandis que le deuxième argument est soit 0, soit 1 pour identifier la première ou la deuxième ligne respectivement.

Dans la conception des modules, définissez d'autres fonctions au besoin. Soyez modulaire dans la conception du logiciel. Dans les laboratoires subséquents, le module *Segment Display* sera révisé puisque l'interrogation ne sera pas utilisée. Si vous êtes modulaires, vous pourrez conserver plus de fonctions.

Module LCD ASM

Le module *LCD ASM* fournit des fonctions de bas niveaux pour manipuler l'afficheur à cristaux liquide (LCD). Voir le document « Interfacing to a Liquid Crystal Display (LCD) » pour un survol de l'interface à un LCD. Un contrôleur/pilote simplifie la manipulation du LCD. Dans la carte Dragon-12, le LCD est un afficheur avec 2 lignes ayant chacune 16 caractères. Chaque position de caractère possède un adresse selon la table suivante (adresses sont en hex); chaque rangée représente une ligne du LCD.

00	01	02	03	04	05	06	07	08	09	10	0A	0B	0C	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

La mise à jour de l'afficheur comprend d'abord le déplacement du « curseur » à une position de caractère pour ensuite écrire soit un caractère à cette position, soit une chaine de caractères à partir de cette position. Le module *LCD ASM* contient plusieurs fonctions de bas niveau (traduit en assembleur) :

`void lcd_init(void)`: Initilalise l'afficheur LCD (et efface l'afficheur).
`void data8(char)`: Affiche un caractère ASCII à la position courante du curseur.
`void clear_lcd(void)`: Efface l'afficheur.
`void type_lcd(char*)`: Affiche la chaine de caractère à partir de la position courante du curseur.
`void set_lcd_addr(char)`: Change l'adresse du curseur (voir la table ci-dessus).
`void instr8(char)`: Écrit une instruction dans le registre de contrôle du contrôleur.

Voici quelques indices pour vous aider dans votre développement:

1. Pour tester l'opération du matériel, vous pouvez modifier les registres des ports parallèles. Lorsque vous vous trouvez dans le débogueur CodeWarrior, examiner et modifier le contenu de la mémoire avec la fenêtre « Debugger Memory »; initialisez les registres de données appropriées des ports qui vous intéressent. Vous pouvez aussi utiliser la même approche pour modifier les registres de données des ports parallèles qui affectent les afficheurs à 7-segment.
2. Partager la préparation en divisant les modules entre les deux membres de votre équipe. Par exemple, un membre peut être responsable de la conception et l'implémentation du module *Segment Display* tandis que l'autre membre peut s'occuper du module *LCD Display* et l'intégration des modules *Keypad* et *Delay* dans le projet CodeWarrior. Assurez-vous que vous présentez votre développement à votre partenaire par la suite.
3. Un projet CodeWarrior additionnel est fourni pour tester le module *LDC Display* et *Segment Display* (il utilise aussi les modules *KeyPad* et *Delay*). Ce programme affiche les caractères tapés sur le clavier en commençant par l'afficheur le plus à la gauche vers la droite jusqu'au 4^{ième} afficheur pour ensuite faire une rotation à l'afficheur le plus à la gauche. Utilisez le projet pour déboguer le module *Segment Display* (et l'intégration des modules *KeyPad* et *Delay*). Lorsque vous avez fini de tester les modules, copiez les fichiers des modules dans le projet de système d'alarme.

À faire

Préparation (avant la session du labo):

- Préparez l'ébauche de votre rapport de labo; ceci devra être un document Word (ou autre traitement de texte) qui contient :
    - La page couverture
    - Les objectifs du laboratoire
    - Le matériel et les composantes utilisés
    - La conception matériel/logiciel
        i. Présentez un circuit qui montre comment le matériel est branché aux ports du microcontrôleur. Montrez le clavier et les afficheurs.
        ii. La conception de logiciel (selon les lignes directrices données) des modules *LCD Display* et *Segment Display*. Soyez le plus complet possible. Il n'est pas nécessaire d'inclure du pseudo-code, mais décrivez toutes les fonctions incluses dans vos modules. Divisez les fonctions en deux groupes : les « Points d'entrée » qui sont les fonctions C appelées par d'autres modules et les « Fonctions locales » qui sont les fonctions internes du module. Il N'est PAS nécessaire d'inclure la conception des modules *KeyPad* et *Delay*.
- Préparez votre code :
    - Créez les fichiers d'en-tête et les fichiers C pour les modules *LCD Display* et *Segment Display*. Ils devront être intégrés dans le projet CodeWarrior du système d'alarme.
    - Créez les fichiers d'en-tête et modifier les fichiers assembleurs du labo 2 des modules *KeyPad* et *Delay*. Intégrez les fichiers dans le projet CodeWarriour du système d'alarme.
    - À votre arrivé à la session de labo, votre projet devra compiler sans erreurs.
- Vous devez montrer votre ébauche de rapport et votre projet CodeWarrior compilé à l'AE du labo dès votre arrivée (des notes sont affectées à ce travail). Il est important de compléter votre préparation car la construction de votre circuit prendra du temps. Il n'est pas possible de concevoir et créer le logiciel durant la session.

Au laboratoire:

a. Chargez vos programmes dans la carte Dragon-12 avec l'aide de CodeWarrior. Roulez votre programme et testez toutes les fonctions du système d'alarme pour assurer le bon fonctionnement des deux afficheurs.
b. Montrez à votre AE l'opération de votre système. Il ou elle notera votre succès.

Dans votre rapport:

a. Donnez la conception finale de vos modules. Inclure une bonne documentation de la conception de votre logiciel. Fournissez votre projet CodeWarrior source dans un fichier zip séparé.

L'ajout d'un module ASM à un projet CodeWarrior

1) Créez un fichier d'en-tête ayant les prototypes C et autres définitions. Voici l'exemple du *keypad_asm.h.*

```
/*----------------------
File: KeyPad_asm.h
Description:  Header file to use the KeyPad Module
-----------------------*/

#ifndef _KEYPAD_ASM_H
#define _KEYPAD_ASM_H

//C Prototypes to assembler subroutines
void initKeyPad(void);
Byte pollReadKey(void);
Byte readKey(void);

// Some Definitions
#define NOKEY 0  // See KeyPad.asm

#endif /* _KEYPAD_ASM_H */
```

2) Changez le fichier d'en-tête pour les definitions des registres HCS12 (assurez-vous d'enlever l'inclusion des fichiers `sections.inc` et `reg9s12.inc`)

```
; Include header files
 NOLIST ; a space exists at the start of this line and the next 2 lines
 include "mc9s12dg256.inc"  ; Defines EQU's for Peripheral Ports
 LIST
```

3) Changez les définitions de sections (enlevez l'inclusion de du fichier `sections.inc` du labo 2). Ces sections sont déjà définies dans le projet CodeWarrior. Ceci implique que vous remplacez toutes les instructions SWITCH dans le fichier assembleur avec une instruction SECTION. Par exemple « `SWITCH global` » devient « `.rodata SECTION` ». Notez qu'il n'existe aucune espace au début des nouvelles instructions, par exemple, `.rodata` est placer dans le champ d'étiquette.
   a. Données contantes globales:
   ```
   .rodata SECTION ; place in constant data section (EEPROM)
   ```
   b. Code:
   ```
   .text SECTION   ; place in code section
   ```
   c. Variables:
   ```
   .data SECTION    ; place in data section (RAM)
   ```

   Notez que la directive assembleur `SECTION` dans CodeWarrior est différente de la directive assembleur `SWITCH` dans le MiniIDE. Elle remplace la directive MiniIDE (elle définit aussi les sections) et est utilisée pour placer le code et/ou les données suivantes dans les diverses sections du projet. Voir la documentation CodeWarrior pour plus de détails au sujet de la directive assembleur `SECTION`.
4) Avec la directive `XDEF`, définissez les noms des sous-programmes dans les modules comme étant des symboles externes à être référé par les autres modules :

```
; Define External Symbols
 XDEF initKeyPad, pollReadKey, readKey
```

5) Avec la directive `XREF`, définissez les noms de sous-programme des autres modules qui sont référé par le module courant.

```
; External Symbols Referenced
 XREF delayms
```

6) L'assembleur du CodeWarrior est sensible à la case. Lorsque vous compilez votre projet, toutes erreurs dans les étiquettes (c.-à-d. les étiquettes ayant les mêmes caractères mais avec différentes cases) apparaitront.
7) Certaines instructions reconnues par le MiniIDE ne sont pas reconnues par l'assembleur CodeWarrior. Par exemple, CodeWarrior ne reconnait pas `ldb`, mais plutôt `ldab`.