

**École de Génie Électrique et Informatique**  
**Université d'Ottawa**



uOttawa

CEG 3536– Architecture des ordinateurs II  
Automne 2023

Laboratoire 3 :  
**L'interface au matériel – afficheur à 7 segments/LCD**  
Soumis par :  
Elam Olame **Mugabo** 300239792  
Decaho **Gbegbe** 300094197

Professeur : Mohamed Ali **Ibrahim**, PhD.ing  
Assistant à l'Enseignement : Joel **Simweray**

Date de soumission : 18 novembre 2023

## **1. Introduction**

Dans ce laboratoire, plusieurs concepts clés de la programmation des microprocesseurs sont mis en lumière. Plus précisément, l'expérience consistait à intégrer les afficheurs de 7 segments et l'écran LCD de la carte Dragon-12 dans un projet CodeWarrior. L'objectif était de les utiliser pour afficher le décompte du système lorsqu'il est armé ou désarmé, ainsi que pour signaler le déclenchement de l'alarme. En résumé, la tâche impliquait l'achèvement de plusieurs fonctions, notamment l'initialisation des ports appropriés pour l'affichage, et la mise en œuvre de la logique permettant d'écrire sur l'affichage en fonction de l'entrée fournie.

## **2. Objectifs**

Les objectifs de ce laboratoire sont les suivants:

- Familiariser avec l'interface matérielle du Motorola 9S12DG256 en développant une application pour un dispositif d'affichage à 7 segments.
- Explorer l'interface avec un afficheur à cristaux liquides (LCD) et mettre en œuvre son utilisation dans le contexte du microcontrôleur.
- Acquérir une compréhension pratique de la façon dont ces dispositifs d'affichage peuvent être intégrés et exploités dans le cadre du Motorola 9S12DG256.

En d'autres termes, l'objectif principal est de fournir une expérience pratique dans l'utilisation de l'interface matérielle du microcontrôleur pour créer une application fonctionnelle avec un affichage à 7 segments et un écran LCD.

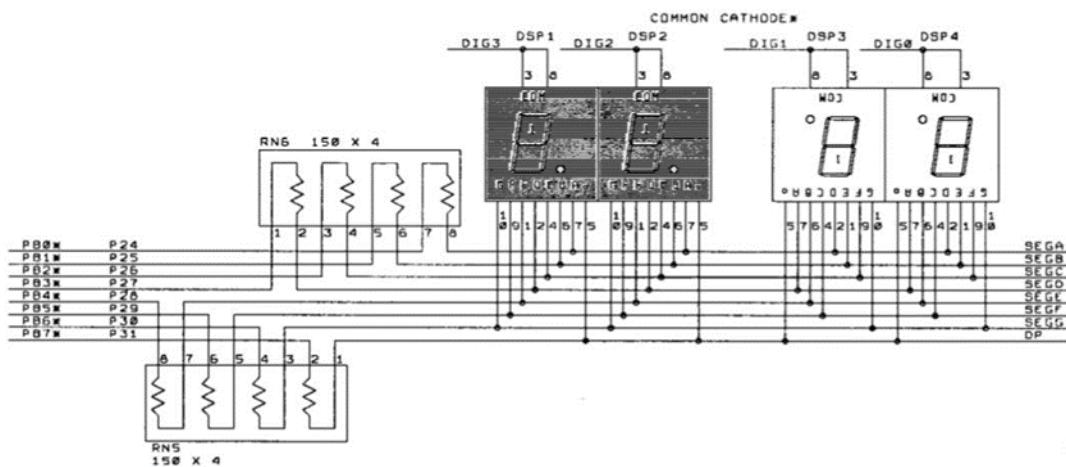
## **3. Équipements & Composants nécessaires pour ce laboratoire:**

- Un PC Windows équipé du logiciel CodeWarrior.
- La carte entraîneur Dragon 12 Plus.

Ces outils serviront de plateforme pour le développement et le test de l'application, permettant ainsi d'explorer et de mettre en œuvre les interfaces d'affichage à 7 segments et d'afficheur à cristaux liquides (LCD) sur la carte Dragon 12 Plus à l'aide de CodeWarrior.

## Préparation de l'expérience

Dans la phase de préparation de l'expérience, l'objectif était de présenter le circuit détaillant la connexion du matériel aux ports du microcontrôleur, y compris le clavier et les afficheurs. Chaque segment d'affichage est spécifiquement connecté à une broche du microcontrôleur, représentée par sa connexion respective aux pins du PORTB. Ces connexions seront initialisées chaque fois que des informations doivent être affichées, contrôlant ainsi les valeurs affichées sur le dispositif à 7 segments. Cette approche méthodique garantit une intégration précise et une communication efficace entre le microcontrôleur et les composants d'affichage.



**Figure 1: Circuit qui montre comment le matériel est branché aux ports du microcontrôleur pour les afficheurs**

Dans la deuxième phase, la conception logicielle des modules LCD Display et Segment Display était au cœur de l'expérience. Voici les étapes et les fonctions clés utilisées dans la réalisation de ces modules:

### Module C Segment Display:

Ce module offre des fonctions spécifiques pour afficher des caractères sur les afficheurs 7-segments. Ces derniers sont utilisés pour visualiser le décompte lorsque le système est armé ou désarmé. Ils jouent également un rôle dans l'affichage du déclenchement de l'alarme. Les fonctions essentielles de ce module comprennent l'initialisation des ports appropriés pour

l'affichage, ainsi que la mise en œuvre de la logique permettant l'écriture sur les afficheurs 7-segments en fonction des données d'entrée.

Cette approche structurée garantit un contrôle précis de l'affichage, contribuant ainsi à la fonctionnalité globale du système.

### **La conception de logiciel du module Segment Display**

```
#define LCD_DISP_ROW 2  
#define LCD_DISP_COL 16  
#define ASCII_SPACE 32  
  
#define ZERO 0x3F  
#define ONE 0x06  
#define TWO 0x5b  
#define THREE 0x4F  
#define FOUR 0x6D  
#define FIVE 0x66  
#define SIX 0x7D  
#define SEVEN 0x07  
#define EIGHT 0xFF  
#define NINE 0x6F  
  
char arr[4] = {0x00, 0x00, 0x00, 0x00};
```

*/\*La fonction initDisp() initialise le matériel (port B et port P) branché aux afficheurs à 7-segments et les afficheurs en blanc (aucun segment allumé).\*/*

**Pseudo code C pour initDisp():**

```
void initDisp(void)
{
    DDRB = 0xFF; //Mettre tous les pins du portB en tant que sortie
    DDRP = 0x0F; //Mettre tous les pins du portP de 0 à 3 en tant que sortie
    clearDisp(); //appeler la fonction clearDisp() qui a pour but d'effacer les afficheurs
}
```

*/\*La fonction clearDisp() mets les afficheurs en blanc (éteindre tous les segments)\*/*

**Pseudo code C pour clearDisp():**

```
void clearDisp(void)
{
    int i = 0; //initialise la variable i à 0
    PORTB = 0x00; //initialise le port B à 0
    PTP = 0x11; // initialise registre I/O Port P
```

while(i<4) //while loop qui sert à mettre à 0 tous les cases de l'afficheur

```
{  
    arr[i] = 0x00; //met la case i à 0  
    i++; // incrémenter le i pour aller à la case suivante
```

```
    }  
}  
  
}
```

*/\*La fonction setCharDisplay() ajoute un caractère à afficher, réserve 4 octets en mémoire pour contenir soit des caractères ou codes pour afficher les caractères sur les afficheurs correspondants. Lorsque la fonction est appelée, deux arguments sont fournis, le caractère à afficher et un numéro d'afficheur pour indiquer sur lequel des afficheurs le caractère doit apparaître. La valeur hexadécimal des chiffres 0 à 9 et des lettres a,b,c et d a été déterminée selon le tableau dans l'annexe 1.\*/*

### Pseudo code C pour setCharDisplay():

```
void setCharDisplay(char ch, byte dispNum)
```

```
{
```

switch(ch){ //la variable ch dans l'instruction switch est évaluée. Si ch correspond à un des cas, alors toutes les instructions spécifiées dans ce cas sont exécutées. Sinon le cas par défaut est exécuté.

//pour chaque chiffre (de 0 à 9) et lettre (a,b,c,d) une valeur hexadécimale reliée aux segment lumineux qu'il faut allumé pour former la lettre ou le chiffre est associée au tableau d'affichage

```
case '0' :
```

```
    arr[(int)dispNum]=0x3F;
```

```
    break;
```

```
case '1' :
```

```
    arr[(int)dispNum]=0x06;
```

```
    break;
```

```
case '2' :
```

```
    arr[(int)dispNum]=0x5B;
```

```
        break;

case '3' :

    arr[(int)dispNum]=0x4F;

    break;

case '4' :

    arr[(int)dispNum]=0x66;

    break;

case '5' :

    arr[(int)dispNum]=0x6D;

    break;

case '6' :

    arr[(int)dispNum]=0x7D;

    break;

case '7' :

    arr[(int)dispNum]=0x07;

    break;

case '8' :

    arr[(int)dispNum]=0x7F;

    break;

case '9' :

    arr[(int)dispNum]=0x67;

    break;

case 'a' :
```

```

arr[(int)dispNum]=0x5F;
break;

case 'b' :
arr[(int)dispNum]=0x7C;
break;

case 'c' :
arr[(int)dispNum]=0x39;
break;

case 'd' :
arr[(int)dispNum]=0x5E;
break;

default:
arr[(int)dispNum]= 0x00;
break;
}

}

```

*/\*La fonction segDisp() met à jour les afficheurs pour une période de temps. Ceci permet à la fonction appelante de regagner le contrôle périodiquement pour lui permettre de compléter d'autres tâches telles que la vérification du clavier.\*/*

### Pseudo code C pour segDisp():

```

void segDisp(void)
{

```

```

int x;

for(x=0;x<5;x++) {

PORTB= arr[0];

PTP=0xE;

delayms(5);

PORTB= arr[1];

PTP=0xD;

delayms(5);

PORTB= arr[2];

PTP=0xB;

delayms(5);

PORTB= arr[3];

PTP=0x7;

delayms(5);

}

}

```

Le module LCD Display fait l'utilisation des fonctions fournies par le module LCD ASM . Il offre les fonctions suivantes aux autres modules pour l'affichage des chaînes de caractères (messages) sur l'afficheur LCD.

## La conception de logiciel du module LCD Display

*/\*La fonction initLCD initialise l'afficheur LCD (cette fonction fait un appel à la fonction lcd\_init() fournie par le module LCD ASM).\*/*

### Pseudo code C pour initLCD(void)

```
void initLCD(void)
{
    lcd_init(); //Appeler la fonction d'initialisation fourni
}
```

*/\*La fonction printLCDStr(char \*, byte) affiche une chaîne sur une ou deux lignes de l'afficheur LCD.\*/*

### Pseudo code C pour printLCDStr(char \*, byte)

```
void printLCDStr(char *str, byte lineno)
{
    if(lineno == 0)//si lineno est égale à 0, imprimer le string sur la première ligne
    {
        lineno = 0x00;
        set_lcd_addr(lineno); //fixé le write address à la première ligne
    } else if(lineno == 1) //si lineno est égale à 1, imprimer le string sur la deuxième ligne
    {
        lineno = 0x40;
        set_lcd_addr(0x40); //fixé le write address à la deuxième ligne
    }
}
```

```
    }  
  
    type_lcd(str); //imprimer le string sur le display  
  
}
```

## Résultats de l'expérience

Il semble que vous ayez apporté des modifications importantes pour adapter les fichiers assembleurs KeyPad.asm et Delay.asm au contexte du laboratoire précédent. La création des fichiers d'en-tête (keypad\_asm.h et delay\_asm.h) avec les prototypes C et autres définitions est une pratique courante pour assurer une bonne organisation du code et faciliter la réutilisation des modules.

De plus, l'ajustement du fichier d'en-tête pour les définitions des registres HCS12, en enlevant l'inclusion des fichiers sections.inc et reg9s12.inc, ainsi que la modification des définitions de sections, démontre une adaptation efficace au projet CodeWarrior et à ses propres définitions.

L'utilisation des directives XDEF et XREF pour définir les noms des sous-programmes comme symboles externes et référencer les sous-programmes entre les modules est une approche logique pour assurer une interconnexion correcte des modules.

Enfin, la modification des instructions, telles que le changement de "ldb" à "ldab" pour s'aligner avec l'assembleur CodeWarrior, montre une attention aux détails pour assurer la compatibilité avec l'environnement de développement utilisé.

```
; Include header files
NOLIST
include "mc9s12dg256.inc" ; Defines EQU's for Peripheral Ports
LIST

; Define External Symbols
XDEF initKeyPad, pollReadKey, readKey

; External Symbols Referenced
XREF delayms

*****EQUATES*****

; codes for scanning keyboard

ROW1 EQU %11101111
ROW2 EQU %11011111
ROW3 EQU %10111111
ROW4 EQU %01111111

;-----Conversion table
NUMKEYS EQU 16 ; Number of keys on the keypad
BADCODE EQU $FF ; returned if translation is unsuccessful
NOKEY EQU $00 ; No key pressed during poll period
POLLCOUNT EQU 1 ; Number of loops to create 1 ms poll time

.rodata SECTION ; Constant data

; definitions for structure cnvTbl_struct
```

```

OFFSET 0
cnvTbl_code ds.b 1
cnvTbl_ascii ds.b 1
cnvTbl_struct_len EQU *

; Conversion Table
cnvTbl dc.b %11101110,'1'
        dc.b %11101101,'2'
        dc.b %11101011,'3'
        dc.b %11100111,'a'
        dc.b %11011110,'4'
        dc.b %11011101,'5'
        dc.b %11011011,'6'
        dc.b %11010111,'b'
        dc.b %10111110,'7'
        dc.b %10111101,'8'
        dc.b %10111011,'9'
        dc.b %10110111,'c'
        dc.b %01111110,'*'
        dc.b %01111101,'0'
        dc.b %01111011,'#'
        dc.b %01110111,'d'

text SECTION ; place in code section

; Subroutine: initKeyPad
; Description:
;   Initiliases PORT A
;   Bits 0-3 as inputs
;   Bits 4-7 as ouputs
;   Enable pullups
-----
initKeyPad:
    movb #,%11110000,DDRA ; Data Direction Register
    bset PUCR,%00000001 ; Enable pullups
    rts

; -----
; Subroutine: ch <- pollReadKey
; Parameters: none
; Local variable: code - in accumulator B
;   ch - in accumulator B - char returned
;   count - index reg X - count to create
;           10 ms delay
; Returns
;   ch: NOKEY when no key pressed,
;       otherwise, ASCII Code in accumulator B
;
; Description:
;   Loops for a period of 2ms, checking to see if
;   key is pressed. Calls readKey to read key if keypress
;   detected (and debounced) on Port A.
;   When no key pressed, loop takes 11 cycles.

```

```

; 1 loop takes 11 X 41 2/3 nano-seconds = 458 1/3 ns
; 1 ms delay requires 2182 loops - set to POLLCOUNT
-----
; Stack Usage
OFFSET 0 ; to setup offset into stack
PRK_CH DS.B 1 ; return value, ASCII code
PRK_VARSIZE
PRK_PR_X DS.W 1 ; preserve X
PRK_RA DS.W 1 ; return address

pollReadKey: pshx ; preserve register
    leas -PRK_VARSIZE,SP
    movb #NOKEY,PRK_CH,SP ; char ch = NOKEY;
    ldx #POLLCOUNT ; int count = POLLCOUNT;
    movb #$0f,PORTA ; PORTA = 0x0f; /*set outputs to low
prk_loop: ; do {
prk_if1:
    ldab PORTA ; 3 cycles if(PORTA != 0x0f)
    cmpb #$0f ; 1 cycle {
    beq prk_endif1 ; 3 cycles
    ldd #1 ; delayms(1)
    jsr delayms ;
prk_if2:
    ldab PORTA ; if(PORTA != 0x0f)
    cmpb #$0f ; {
    beq prk_endif2 ; ch = readKey();
    jsr readKey
    stab PRK_CH,SP
    bra prk_endloop ; break;
prk_endif2:
prk_endif1:
    dex ; 1 cycle count--;
    bne prk_loop ; 3 cycles } while(count!=0);
prk_endloop:
    ldab PRK_CH,SP ; return(ch);
; Restore stack and registers
    leas PRK_VARSIZE,SP
    pulx
    rts

-----
; Subroutine: ch <- readKey
; Arguments: none
; Local variable: code - on stack
;           ch - accumulator B
; Returns
; ch - ASCII Code in accumulator B

; Description:
; Main subroutine that reads a code from the
; keyboard using the subroutine readKeyCode. The
; code is then translated with the subroutine
; translate to get the corresponding ASCII code.

```

```

;-----  

; Stack Usage  

; OFFSET 0 ; to setup offset into stack  

; RDK_CODE DS.B 1 ; code variable  

; RDK_VARSIZE DS.B 1 ; Preseserve A  

; RDK_RA DS.W 1 ; return address  

;  

readKey:psha      ; byte code;  

; byte ch;  

leas -RDK_VARSIZE,SP  

RDK_DO          ; do  

; {  

    movb $0F,PORTA      ; PORTA = 0x0F; // set all output pins to 0  

RDK_LP1 ldab PORTA      ; while(PORTA==0x0F) /* wait */;  

    cmpb #$0F  

    beq RDK_LP1  

    movb PORTA,RDK_CODE,SP ; code = PORTA  

    ldd #10  

    jsr delayms           ; delayms(10);  

    ldab PORTA            ; } while(code != PORTA); // start again if PORTA has changed  

    cmpb RDK_CODE,SP  

    bne RDK_DO  

    jsr readKeyCode        ; code = readKeyCode(); // get the keycode  

    stab RDK_CODE,SP  

    movb #$0F,PORTA      ; PORTA = 0x0F; // set all output pins to 0  

RDK_LP2 ldab PORTA      ; while(PORTA!=0F) /* wait */;  

    cmpb #$0F  

    bne RDK_LP2  

    ldd #10  

    jsr delayms           ; delayms(10); // Debouncing release of the key  

    ldab RDK_CODE,SP  

    jsr translate           ; ch = translate(code);  

    leas RDK_VARSIZE,SP  

    pula  

    rts       ; return(ch);  

;  

;-----  

; Subroutine: key <- readKeyCode  

; Arguments: none  

; Local variables: key: Accumulator B  

; Returns: key - in Accumulator B - code corresponding to key pressed  

;  

; Description: Assume key is pressed. Set 0 on each output pin  

; to find row and hence code for the key.  

;-----  

; Stack Usage: none  

;  

readKeyCode:  

    movb #ROW1,PORTA      ; PORTA = ROW1;  

RKY_IF1 ldab PORTA      ; if(PORTA == ROW1) // input pin is 0 if key in row 1  

    cmpb #ROW1  

    bne RKY_ENDIF1 ; {  

    movb #ROW2,PORTA      ; PORTA = ROW2

```

```

RKY_IF2 ldab PORTA      ; if(PORTA == ROW2)
        cmpb #ROW2
        bne RKY_ENDIF2
        movb #ROW3,PORTA
RKY_IF3 ldab PORTA      ; if(PORTA == ROW3)
        cmpb #ROW3
        bne RKY_ENDIF3
        movb #ROW4,PORTA
RKY_ENDIF3
RKY_ENDIF2
RKY_ENDIF1
        ldab PORTA      ; key = PORTA;
        rts             ; return(key);

;-----;
; Subroutine: ch <- translate(code)
; Arguments
; code - in Acc B - code read from keypad port
; Returns
; ch - saved on stack but returned in Acc B - ASCII code
; Local Variables
;     ptr - in register X - pointer to the table
; count - counter for loop in accumulator A
; Description:
;     Translates the code by using the conversion table
; searching for the code.  If not found, then BADCODE
; is returned.
;-----;
; Stack Usage:
; OFFSET 0
TR_CH DS.B 1 ; for ch
TR_PR_A DS.B 1 ; preserved registers A
TR_PR_X DS.B 1 ; preserved registers X
TR_RA DS.W 1 ; return address

translate: psha
        pshx ; preserve registers
        leas -1,SP          ; byte chascii;
        ldx #cnvTbl         ; ptr = cnvTbl;
        clra               ; ix = 0;
        movb #BADCODE,TR_CH,SP ; ch = BADCODE;

TR_loop           ; do {
        ; IF code = [ptr]
        cmpb cnvTbl_code,X      ; if(code == ptr->code)
        bne TR_endif
        ; {
        movb cnvTbl_ascii,X,TR_CH,SP ; ch <- [ptr+1]
        bra TR_endwh            ; break;
TR_endif          ; }
        ; else {
        leax cnvTbl_struct_len,X    ; ptr++;
        inca ; increment count      ; ix++;

```

```
        ;} WHILE count < NUMKEYS
        ;}
        cmpa #NUMKEYS
        blo TR_loop
TR_endwh ; ENDWHILE

        pulb ; move ch to Acc B
        ; restore registres
        pulx
        pula
        rts
```

Figure 2: Images du code dans le fichier KeyPad.asm modifié pour fonctionner dans le projet CodeWarrior

```
;-----  
; Alarm System Assembler Program  
; File: delay.asm  
; Description: The Delay Module  
; Author: Gilbert Arbez  
; Date: Fall 2010  
;  
TRUE      equ    1  
FALSE     equ    0  
  
|  
  
;---- Define External Symbols  
XDEF delayms, setDelay, polldelay  
  
; Some definitions  
MSCOUNT equ 3000  
  
.text: SECTION  
  
;  
; Subroutine delayms  
; Parameters: num - number of milliseconds to delay - in D  
; Returns: nothing  
; Description: Delays for num ms.  
;  
delayms: pshd  
dlms_while:          ; do  
    jsr delay1ms      ; delay1ms()  
    dbne d,dlms_while ; num--; while(num !=0)  
dlms_endwhile:  
    puls                ; restores register  
    rts  
  
;  
; Subroutine setDelay  
; Parameters: cnt - accumulator D  
; Returns: nothing  
; Global Variables: delayCount  
; Description: Initialises the delayCount  
;               variable.  
;  
setDelay:  
    std delayCount      ; delayCount = cnt;  
    rts  
  
;  
; Subroutine: polldelay  
; Parameters: none  
; Returns: TRUE when delay counter reaches 0 - in accumulator A  
; Local Variables  
;     retval - acc A cntr - X register  
; Global Variables:  
;     delayCount
```

```

; Description: The subroutine delays for 1 ms, decrements delayCount.
; If delayCount is zero, return TRUE; FALSE otherwise.
-----
; Stack Usage:
; OFFSET 0 ; to setup offset into stack
PDLY_VARSIZE:
PDLY_PR_X    DS.W 1 ; preserve X
PDLY_RA      DS.W 1 ; return address

polldelay: pshx
    ldaa #FALSE      ; byte retval=FALSE; // return value
    jsr delay1ms     ; delay1ms();
    ldx delayCount   ; delayCount--;
    dex
    stx delayCount
pld_if:
    bne pld_endif    ; if(delayCount==0)
    ldaa #TRUE        ; retval=TRUE;
pld_endif:
    ; restore registers and stack
    pulx
    rts

-----
; Subroutine: delay1ms
; Parameters: none
; Returns: nothing
; Description: The subroutine delays for 1 ms and returns.
; Core Clock is set to 24 MHz, so 1 cycle is 41 2/3 ns
; NOP takes up 1 cycle, thus 41 2/3 ns
; Need 24 cycles to create 1 microsecond delay
; 8 cycles creates a 333 1/3 nano delay
; DEX - 1 cycle
; BNE - 3 cycles - when branch is taken
; Need 4 NOP
; Run Loop 3000 times to create a 1 ms delay
-----
; Stack Usage:
; OFFSET 0 ; to setup offset into stack
DLY1_VARSIZE:
DLY1_PR_X    DS.W 1 ; preserve X
DLY1_RA      DS.W 1 ; return address

delay1ms: pshx
    ldx #MSCOUNT      ; byte cntr = MSCOUNT;
dly1_while:
    beq dly1_endwhile ; while(cntr != 0)
                        ; {
        nop            ;     asm { nop; nop; nop; nop; }
        nop
        nop
        nop
        dex            ;     cntr--;

```

```

bra dly1_while    ; }
dly1_endwhile:
    pulx
    rts

;-----[ Global variables ]-----;
.rodata SECTION
delayCount ds.w 1    ; 2 byte delay counter

```

**Figure 3: Image du code dans le fichier Delay.asm modifié pour fonctionner dans le projet CodeWarrior**

```

/*-----[ File: SegDisp.c ]-----;
File: SegDisp.c
Description: Segment Display Module
-----*/;

#include <stdtypes.h>
#include "mc9s12dg256.h"
#include "SegDisp.h"
#include "Delay_asm.h"

// Prototypes for internal functions

#define LCD_DISP_ROW 2
#define LCD_DISP_COL 16
#define ASCII_SPACE 32

#define ZERO 0x3F
#define ONE 0x06
#define TWO 0x5b
#define THREE 0x4F
#define FOUR 0x6D
#define FIVE 0x66
#define SIX 0x7D
#define SEVEN 0x07
#define EIGHT 0xFF
#define NINE 0x6F

char arr[4] = {0x00, 0x00, 0x00, 0x00};

/*-----[ Function: initDisp ]-----;
Function: initDisp
Description: initializes hardware for the
7-segment displays.
-----*/;
void initDisp(void)
{
    // Complete this function
    DDRB = 0xFF;
    DDRP = 0x0F;
    clearDisp();
}

```

Figure 4: Code C pour la fonction initDisp()

```
/*
Function: clearDisp
Description: Clears all displays.
*/
void clearDisp(void)
{
    // Complete this function
    int i = 0;
    PORTB = 0x00;
    PTP = 0x11;

    while(i<4)
    {
        arr[i] = 0x00;
        i++;
    }
}
```

Figure 5: Code C pour la fonction clearDisp()

```
void setCharDisplay(char ch, byte dispNum)
{
    // Complete this function
    switch(ch){
        case '0':
            arr[(int)dispNum]=0x3F;
            break;
        case '1':
            arr[(int)dispNum]=0x06;
            break;
        case '2':
            arr[(int)dispNum]=0x5B;
            break;
        case '3':
            arr[(int)dispNum]=0x4F;
            break;
        case '4':
            arr[(int)dispNum]=0x66;
            break;
        case '5':
            arr[(int)dispNum]=0x6D;
            break;
        case '6':
            arr[(int)dispNum]=0x7D;
            break;
        case '7':
            arr[(int)dispNum]=0x07;
            break;
        case '8':
            arr[(int)dispNum]=0x7F;
            break;
        case '9':
            arr[(int)dispNum]=0x67;
            break;
        case 'a':
            arr[(int)dispNum]=0x5F;
            break;
        case 'b':
            arr[(int)dispNum]=0x7C;
            break;
        case 'c':
            arr[(int)dispNum]=0x39;
            break;
        case 'd':
            arr[(int)dispNum]=0x5E;
            break;
        default:
            arr[(int)dispNum]= 0x00;
            break;
    }
}
```

Figure 6: Code C pour la fonction setCharDisplay()

```

-----*/
void segDisp(void)
{
    // Complete this function
    int x;
    for(x=0;x<5;x++) {
        PORTB= arr[0];
        PTP=0xE;
        delayms(5);

        PORTB= arr[1];
        PTP=0xD;
        delayms(5);

        PORTB= arr[2];
        PTP=0xB;
        delayms(5);

        PORTB= arr[3];
        PTP=0x7;
        delayms(5);
    }
}

```

Figure 7: Code C pour la fonction segDisp()

```

/*
File: lcdDisp.c  (LCD Display Module)

Description: C Module that provides
            display functions on the
            LCD. It makes use of the LCD ASM
            Module developed in assembler.
-----*/
#include <mc9s12dg256.h>
#include "lcd_asm.h"

/*
Function: initLCD
Parameters: None.
Returns: nothing
Description: Initialised the LCD hardware by
            calling the assembler subroutine.
-----*/

void initLCD(void)
{
    // complete this function
    lcd_init();
}

```

Figure 8: Code C pour la fonction initLCD(void)

```

void printLCDStr(char *str, byte lineno)
{
    // Complete this function
    if(lineno == 0)//if lineno is 0, print string on first line
    {
        lineno = 0x00;
        set_lcd_addr(lineno); //set write address to first line
    } else if(lineno == 1) //if lineno is 1, print string on second line
    {
        lineno = 0x40;
        set_lcd_addr(0x40); //set write address to second line
    }
    type_lcd(str); //print the string to the display
}

```

**Figure 9: Code C pour la fonction printLCDStr(char \*, byte)**

## Discussion & Conclusion

L'objectif de cette expérience était d'intégrer les afficheurs de 7 segments et le LCD de la carte Dragon-12 dans le projet de système d'alarme en combinant les modules assembleurs Keypad et Délai (issus de l'expérience précédente) dans le projet CodeWarrior. Le logiciel CodeWarrior et la carte Dragon 12 Plus ont facilité la construction du code C en permettant sa compilation et la vérification du résultat final à l'aide des touches présentes sur la carte Dragon 12 Plus.

Lors des tests du code, bien que la compilation se soit déroulée sans erreur, un problème est survenu lors de l'impression de caractères sur le LCD display. Plus précisément, l'ancienne écriture n'était pas effacée avant que la nouvelle écriture ne soit affichée. Cette problématique a été résolue en ajustant quelques lignes de code dans la fonction printLCDStr(char \*, byte). Pour remédier à ces erreurs, il a été nécessaire de renommer la fonction 'main' en 'main2' dans le fichier main.c. Une fois cette correction apportée, le projet a fonctionné de manière optimale.

Cette expérience a offert l'opportunité d'approfondir la compréhension de la programmation sur le Dragon 12 Plus en langage C. Plus précisément, elle a permis d'acquérir des connaissances sur le fonctionnement et le hardware du LCD display et du 7 Segment Display, ainsi que sur la création de fichiers d'en-tête permettant de définir des prototypes de fonctions tirées des modules assembleurs.

En conclusion, le résultat obtenu correspondait exactement aux attentes, ce qui permet de considérer cette expérience comme un succès complet. Les membres de l'équipe sont désormais plus familiers et à l'aise avec le codage en C sur la carte Dragon 12 Plus.