UQÀM **EMB7000**
Département d'informatique **Introduction aux systèmes embarqués**

Lab. #2 : Clavier et Afficheur LCD

I. But du laboratoire et matériel requis :

Le but du laboratoire est d'encoder un clavier matriciel afin de générer un code ASCII correspondant à chaque touche du clavier sur l'écran LCD.

Matériel requis:
- Carte d'extension
    - Clavier matriciel 4x3
    - Afficheur LCD Lumex LCM-S01602DTR
- Système de développement
    - MSP-EXP430FR6989 **ou**
    - CY8CKIT-059 PSoC 5LP
- Plaquette de montage, fils, etc.

II. Partie #1 : Encodeurs de claviers

Introduction

Une des applications classiques des microcontrôleurs touche aux claviers d'ordinateurs. L'usage des microcontrôleurs permet non seulement d'identifier les touches pesées dans un clavier générique, mais aussi de régler le sens à leur donner par logiciel.

Identification des touches pesées sur un clavier

Il existe deux types fondamentaux d'encodage de claviers, dépendant de la structure mécanique du clavier utilisé.

<u>Encodage linéaire :</u>

Dans ce cas, le clavier est organisé pour que les touches partagent une borne commune. L'identification de la touche enfoncée se fait en scrutant chaque touche jusqu'à la détection d'un changement d'état dans l'une d'elles.

Cette technique est relativement simple, mais le nombre de lignes d'e/s requis pour le microcontrôleur est égal au nombre de touches, ce qui peut excéder la capacité du microcontrôleur.

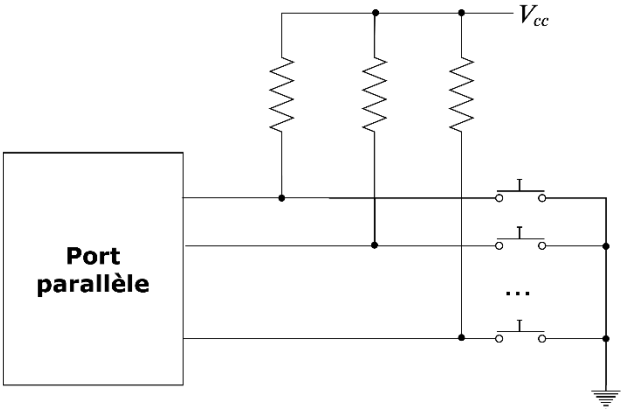


Schéma d'encodage linéaire

Encodage matriciel :

Dans ce cas, les touches du clavier sont disposées en matrice, et c'est l'intersection d'une ligne avec une colonne qui permet d'identifier la touche enfoncée. L'identification de la touche peut se faire de deux façons :

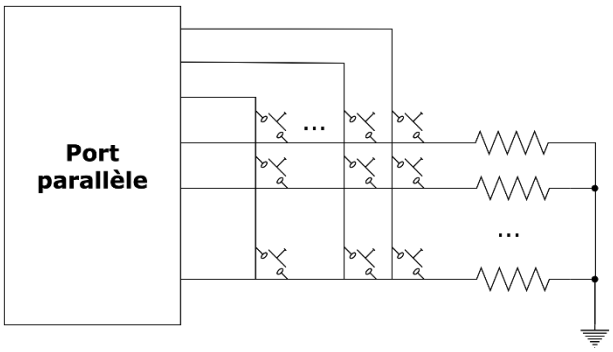


Schéma d'encodage matriciel

a) Scrutation séquentielle : On génère un « 1 » sur chaque colonne en succession jusqu'à ce que la lecture d'une des lignes le retourne. Le numéro de la ligne et celui de la colonne qui a généré le « 1 » donnent les coordonnées de la touche enfoncée. Des résistances de tirage sont requises pour les lignes afin de lire « 0 » sur les lignes inactives.

b) Renversement de lignes : On envoie un « 1 » sur toutes les colonnes simultanément et on fait une lecture des lignes. Si une touche est enfoncée, un « 1 » apparaîtra sur la ligne correspondante. On renvoie alors le code lu des lignes vers les colonnes, ce qui fera apparaître un « 1 » sur l'une d'elles. Les positions des deux « 1 » lus donnent les coordonnées de la touche recherchée. Cette technique est potentiellement plus rapide que la précédente (2 étapes au lieu de n, le nombre de colonnes, pour la précédente), mais elle demande des résistances de tirage aussi bien pour les lignes que pour les colonnes.

Dans les deux cas, si un clavier possède m rangées de n clés, l'identification des clés pesées requière l'utilisation de $m + n$ lignes de ports parallèles, au lieu de $m \times n$ pour un clavier linéaire.

Problèmes connexes

Après avoir enfoncé ou relâché une touche d'un clavier, un certain délai (jusqu'à 10 ms) s'écoule durant lequel le ressort de rappel situé sous la touche subit des rebondissements, créant ainsi de faux contacts. Cette situation impose d'attendre la fin du délai de rebondissement avant d'identifier la touche pesée ou relâchée, menant au pseudo-code suivant pour le programme pilote du clavier :

```
lire_clavier()
{
        Attendre que le clavier devienne actif
        Identifier touche
        Attendre la fin du délai de rebondissement
        Vérifier que la même touche est toujours enfoncée

        Identifier les coordonnées de la touche

        Attendre que le clavier devienne inactif
        Attendre la fin du délai de rebondissement

        Retourner les coordonnées de la touche
}
```

L'approche précédente fonctionne bien, mais n'est pas efficace; le temps passé dans les boucles d'attente pourrait être mis à profit pour exécuter d'autres tâches. Une meilleure solution consiste

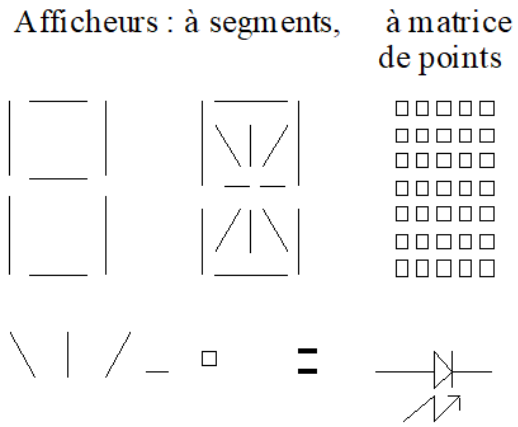
à mettre en œuvre le pilote du clavier suivant le modèle d'un automate à états finis. L'idée est de générer des interruptions périodiques et, à chaque interruption, de vérifier une des étapes précédentes et mettre à jour un compteur d'étapes s'il y a lieu. Les délais d'attente précédents deviennent alors implicites dans la chaîne de traitement. Le pseudo-code est comme suit :

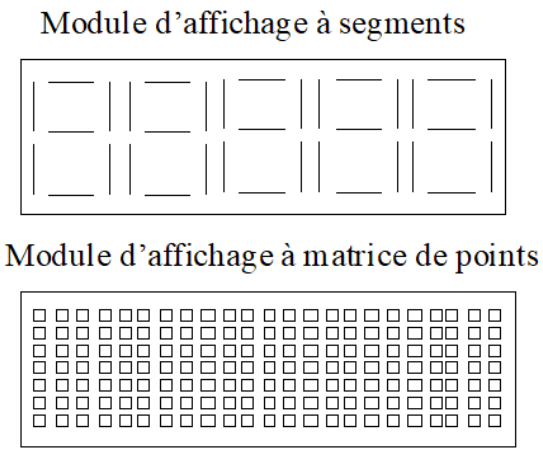
```
main()
{
    ...
    compteur d'étapes = 0
    indicateur de touche pesée = 0
    Programmer un temporisateur (timer) avec une valeur de compte récurrente
    Autoriser les interruptions du temporisateur
    Autoriser les interruptions du microcontrôleur

    ... autres actions ...

    if(indicateur de touche pesée = 1)
    {
              Récupérer le code
              indicateur de touche pesée = 0

              ... utiliser le code récupéré ...
    }

    ... autres actions ...
}

ISR()
{
    switch(compteur d'étapes)
    {
              case 0:
                  if(détection de touche enfoncée)
                      Incrémenter le compteur d'étapes
                  break;
              case 1:
                  if(touche toujours enfoncée)
                  {
                      code = Décoder les coordonnées de la touche
                      Incrémenter le compteur d'étapes
                  }
                 else
                  {
                      compteur d'étapes = 0
                 }
                  break;
             case 2:
                  if(touche relâchée)
                      Incrémenter le compteur d'étapes
                  break;
              case 3:
                  if(touche toujours relâchée)
                  {
                    compteur d'étapes = 0
                    indicateur de touche pesée = 1
                  }
                  break;
    }
}
```

III. Partie #2 : Circuits d'affichage

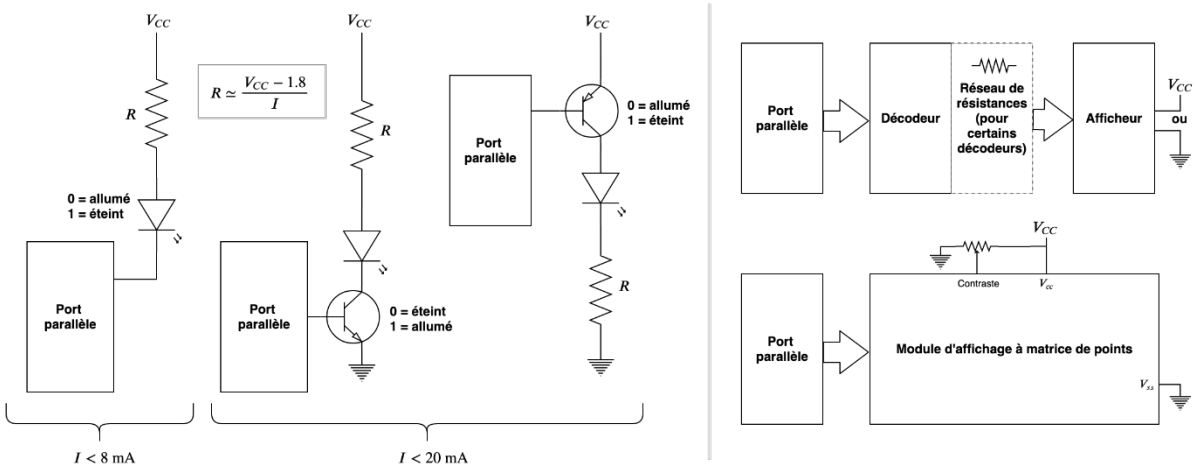
Introduction

Les microcontrôleurs utilisent deux technologies de base pour afficher l'information : les diodes émettrices de lumière (DEL) et les cristaux liquides. Dans le premier cas, les DEL sont organisées en ensembles de segments dont les sous-ensembles, lorsqu'allumés, affichent l'information voulue. Quant aux cristaux liquides, ils sont généralement organisés en matrices de points ou pixels. Dans tous les cas, l'information est affichée en allumant les segments ou pixels concernés. Pour cela, on peut soit relier les lignes d'un port parallèle aux différents segments dans des cas simples, ou bien utiliser une interface avec contrôleur dédié.





Concernant les afficheurs à segments, on peut les piloter en utilisant des décodeurs matériels qui convertissent un code binaire en un code de segments à allumer (e.g. 7447, 7448), ou encore des décodeurs logiciels qui génèrent les codes de segment par programmation. Quant aux affichages à matrice de pixels, ils sont généralement pilotés à l'aide de contrôleurs spécialisés que l'on commande à l'aide d'un port parallèle ou sériel. Un modèle populaire utilise un bus de données de 4 ou 8 bits (programmable lors de l'initialisation) et des lignes R/W, E et RS pour programmer le type d'opération voulu.



Cinq différents types de circuit d'affichage

IV. Procédure à effectuer

1. Étudier le code fourni en annexe à la lumière des fichiers de spécification de l'afficheur LCD fourni et vérifier que le code fonctionne comme prévu en le compilant et en l'exécutant sur le kit de développement du MSP430 (ou PSoC 5LP). Notez que le module LCD est programmé en mode 4 bits pour le transfert des données, ce qui implique que chaque mot de 8 bit envoyé par le microcontrôleur le sera en deux moitiés successives de 4 bits.

2. Écrire un programme qui décode un clavier téléphonique de 3x4 et qui montre sur l'afficheur LCD les codes hexadécimaux et ASCII de toute clé pesée et décodée. Afin de réduire le nombre de lignes d'e/s du microcontrôleur à utiliser, veuillez noter que les 4 lignes de données du module LCD sont mises en commun avec les 4 lignes du clavier (voir le schéma électrique à l'annexe C – ce mode de connexion est possible puisque le module LCD ignore l'état de ses lignes de données lorsque sa ligne E est inactive). La détection de touche **doit** se faire en utilisant la méthode « renversement de lignes », soit par interruption (préférable) ou par attente active. L'exemple de lecture de clavier utilise la méthode « scrutation séquentielle ». Notez que la 4e colonne avec les lettres A-D ne sera pas utilisée dans ce laboratoire.

3. Écrire un programme qui fait clignoter une DEL à une période d'une seconde via une routine de service d'interruption tel que montré en annexe.

4. Écrire un programme qui réalise une horloge électronique et qui affiche le jour et l'heure (ex: Mercredi 08:35 pm) sur le module LCD. L'entrée initiale de la date se fera par le biais du clavier téléphonique 3x4, et le clavier et afficheur LCD seront reliés au microcontrôleur par des ports parallèles. Veuillez utiliser du code modulaire pour simplifier la structure du projet.

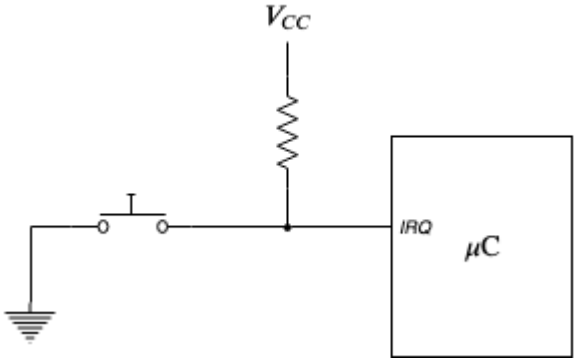
V. Questions

1) Que se passe-t-il si plus d'une touche est enfoncée à la fois ? Comparez les deux méthodes de décodage de claviers matriciels présentées plus haut face à ce problème et suggérez une solution.

2) Supposez que le montage ci-contre soit relié à l'entrée IRQ d'un microcontrôleur :

a) Décrivez les conséquences de l'effet de rebondissement sur le microcontrôleur quand on ferme le commutateur.

b) Suggérez une solution au problème.



Annexe A : MSP430

<u>Exemple de programmation d'un afficheur à cristaux liquides</u>

```
/**********************************************************************/
/* EMB7000 - Introduction aux systèmes embarqués                      */
/* Lab2 - Clavier et Afficheur LCD                                    */
/*                                                                    */
/* Exemple de programmation d'un afficheur à cristaux liquides        */
/*                                                                    */
/* Pour le MSP430FR6989                                               */
/**********************************************************************/
/*             |
*   MPS430    |
*             |
*         P3.0 |-------------------------------------+ db4
*         P3.1 |-------------------------------------|-+ db7
*         P3.2 |-------------------------------------|-|-+ db6
*         P3.3 |-------------------------------------|-|-|-+ db7
*         P4.0 |-----------------------+RS           | | | |
*         P4.1 |-----------------------|---+E        | | | |
*              |                       |   |         | | | |
*              |    RW=Vss             |   |         | | | |
*              |                       |   |         | | | |
*              |            Vdd        |   |         | | | |
*              |             |         |   |         | | | |
*______________|             +-----+   |   |         | | | |
*                            |     |   |   |         | | | |
*                          +---+   |   |   |         | | | |
*                          |10K|---|-+ |   |         | | | |
*                          +---+   | | |   |         | | | |
*                            |     | | |   |         | | | |
*                      +-----+---+-!-|-|-+ |         | | | |
*                      |       1 | | | | | | | | | | | | | 14
*                      |       +-------------------------------------+
*                     ---      | o o o o o o o o o o o o o o o      |
*                      -       | V V V R R E D D D D D D D D     LCD |
*                              | s d o S W   b b b b b b b b  (LM015) |
*                              | s d         0 1 2 3 4 5 6 7        |
*                              +-------------------------------------+
*/
/**********************************************************************/
#include <msp430fr6989.h>

#define LCD_RS (0x01) // Register select bit (P4.0)
#define LCD_E (0x02) // Clock bit (P4.1)

void initLCD(void);
void wLCDdat(unsigned char data);
void wLCDctrl (unsigned char ctrl);
void wLCDctrl4(unsigned char ctrl);
void putcharLCD(unsigned char car);
void cputsLCD(char *putstr);
```

```
/*******************************************************
* LCD display peripheral initialization *
*******************************************************/
void initLCD(void) {
    P3DIR |= 0x0f; /* P3.0-3 = output DATA */
    P4DIR |= 0x03; /* P4.0-1 = output E RS */

    P3OUT = 0x00; /* Initialize output port to 0x00 */

     /* In 8-bit mode */
     wLCDctrl (0x02); /* 4-bit mode */
     __delay_cycles(624); // 39 us

     /* In 4-bit mode */
     wLCDctrl4(0x24); /* Function Set- 4-bit, 1-line, 5X7 */

     wLCDctrl4(0x0f); /* Display on, Cursor on, Blink on */

     wLCDctrl4(0x01); /* Clear Display */
     __delay_cycles(24000); // 1.5 ms -- probably not necessary

     wLCDctrl4(0x06); /* Entry mode - Increment address, no shift */

     __delay_cycles(4000); // 250 us
 }

/*******************************************************
* wLCDdat - Write data word to LCD peripheral *
* Enter with data word in accumulator *
*******************************************************/
void wLCDdat(unsigned char data) {
     P3OUT = (data >> 4) & 0x0F; /* Write HI nibble word to LCD */
     P4OUT |= LCD_RS ; /* RS->1 */
     P4OUT |= LCD_E ; /* E->1 */
     P4OUT &= ~LCD_E ; /* E->0 */
     P4OUT &= ~LCD_RS ; /* RS->0 */

     // Note: no delay needed between the two nibbles

     P3OUT = data & 0x0F; /* Write LO nibble word to LCD */
     P4OUT |= LCD_RS ; /* RS->1 */
     P4OUT |= LCD_E ; /* E->1 */
     P4OUT &= ~LCD_E ; /* E->0 */
     P4OUT &= ~LCD_RS ; /* RS->0 */

     __delay_cycles(624); // 39 us
 }

/*******************************************************
* wLCDctrl - Write control word to LCD peripheral *
* Enter with control word in accumulator *
*******************************************************/
void wLCDctrl (unsigned char ctrl) {
     P3OUT = ctrl ;
```

```
     P4OUT |= LCD_E ; /* E->1 */
     P4OUT &= ~LCD_E ; /* E->0 Pulse the enable line */

     __delay_cycles(624); // 39 us
}

 /*********************************************************
 * wLCDctrl4 - Write control word to LCD peripheral *
 * configured as a 4 bit interface *
 * Enter with control word in accumulator *
 *********************************************************/
void wLCDctrl4(unsigned char ctrl) {
    P3OUT = (ctrl >> 4) & 0x0F; /* Write HI nibble word to LCD */;
    P4OUT |= LCD_E ; /* E->1 */
    P4OUT &= ~LCD_E ; /* E->0 */

    P3OUT = ctrl & 0x0f; /* Write LO nibble word to LCD */
    P4OUT |= LCD_E ; /* E->1 */
    P4OUT &= ~LCD_E ; /* E->0 */

    __delay_cycles(624); // 39 us
}

 /*********************************************************
 * Send a character to the LCD display *
 *********************************************************/
void putcharLCD(unsigned char car) {
    if ((car == 0x0a) || (car == 0x0d)) /* Look for a LF or CR */
    {
        wLCDctrl(0x02) ; /* Home */
        wLCDctrl4(0x01); /* Clear Display */
    }
    else
        wLCDdat(car);
}

 /*********************************************************
 * Write a string to the LCD display *
 *********************************************************/
void cputsLCD(char *putstr) {
    while(*putstr != 0)
        putcharLCD(*putstr++);
}

void main (void) {
    // Stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    initLCD();
    cputsLCD ("EMB7000 - Lab 2");
}
```

Exemple de programmation d'un clavier

```
/*************************************************************************/
/* EMB7000 - Introduction aux systèmes embarqués */
/* Lab2 - Clavier et Afficheur LCD */
/**/
/* Exemple de programmation d'un clavier */
/**/
/* Pour le MSP430FR6989 */
/*************************************************************************/

#include <msp430fr6989.h>

// On suppose que :

// La disposition des touches est comme suit :
//
// COL1 COL2 COL3
//   1    2    3   ROW 1
//   4    5    6   ROW 2
//   7    8    9   ROW 3
//   *    0    #   ROW 4

// L'arrangement des broches est comme suit :
// P2.1-P2.4 = L1-L4
// P9.3-P9.5 = C1-C3 (on n'utilise pas C4)

#define ROW_PORT  P2OUT
#define COL_PORT  P9IN

#define ROW_MASK 0x1E
#define COL_MASK 0x38

#define ROW_1 0x02 // P2.1 00000010
#define ROW_4 0x10 // P2.4 00010000

#define COL_1 0x08 // P9.3 00001000

unsigned char keyPadMatrix[] =
{
    '1','2','3',
    '4','5','6',
    '7','8','9',
    '*','0','#', 0xFF
};

/*******************************************************
 * scanCalvierInit: Initialise les ports pour la lecture *
 *******************************************************/
void scanClavierInit() {
    P2OUT = 0x00; /* Initialize output (rows) to 0x00 */
    P2DIR |= ROW_MASK; /* P2.1-4 = output pour les lignes  */

    P9DIR &= ~COL_MASK; /* P9.3-5 = input pour les colonnes */
    P9REN |= COL_MASK; /* P9.3-5 Activer R pullup/down Col 1-3*/
```

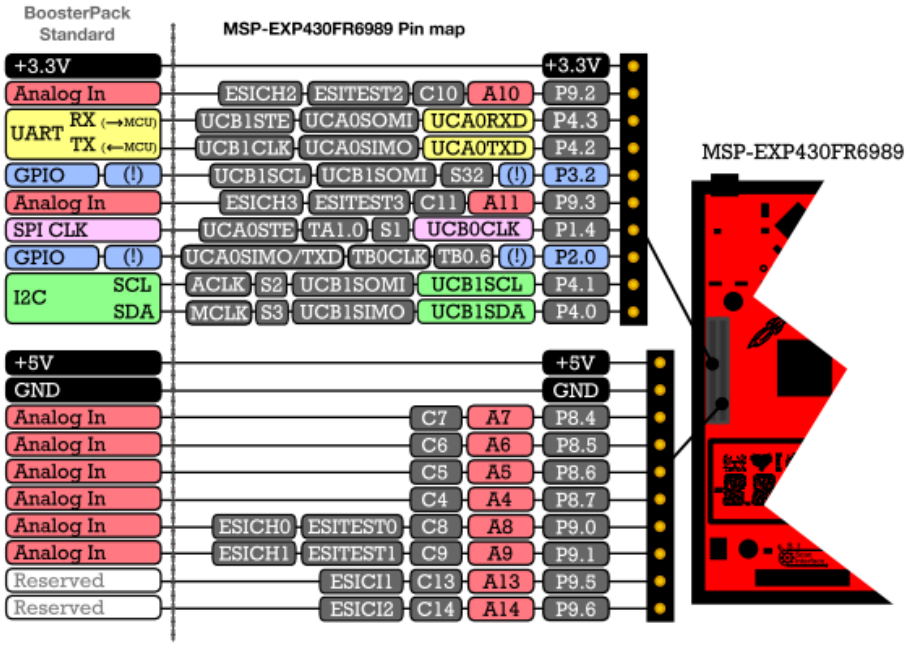
```
    P9OUT &= ~COL_MASK; /* P9.3-5 Set pulldown  */
}

/*********************************************************
 * scanClavier: Scan et retourne la première touche pesée *
 *  du clavier pendant le scan, sinon 0xff                *
 *********************************************************/
unsigned char scanClavier() {
    unsigned char key = 0, row;

    for(row = ROW_1; row <= ROW_4; row <<= 1) // Suppose que ROW_4 est à un bit plus
significatif que ROW_1
    {
        ROW_PORT &= ~ROW_MASK; // Mettre toutes les lignes à 0
        ROW_PORT |= row;  // Propager un 1 sur une ligne

        // Lire les colonnes et arrêter si un 1 détecté
        if(COL_PORT & COL_1)
            break; // Colonne 1
        key++;

        if(COL_PORT & (COL_1 << 1)) // COL_2 = COL_1 << 1
            break; // Colonne 2
        key++;

        if(COL_PORT & (COL_1 << 2)) // COL_3 = COL_1 << 2
            break; // Colonne 3
        key++;
    }

    return keyPadMatrix[key];
}

/*********************************************************
 * scanTouche: Attend la première touche pesée du clavier *
 *  et retourne sa valeur après l'écoulement du délai     *
 *  de rebondissement.                                    *
 *********************************************************/
unsigned char scanTouche() {
    unsigned char touche=0xff;

    while (touche==0xff) {
        touche = scanClavier();

        if (touche != 0xff) // Touche pesée
        {
            _delay_cycles(160000); // Délai de rebondissement (~10 ms)

            if (touche != scanClavier())
                touche = 0xff; // La touche a changée pendant le délai
        }
    }

    return touche ;
}
```

```
/*******************************************************
 * Programme de test                                   *
 *******************************************************/
void main() {
    // Stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    scanClavierInit();
    unsigned char touche = scanTouche(); // La variable 'touche' contient maintenant
le code ASCII de la touche appuyée
}
```

Exemple de base de temps

```
/*************************************************************************/
/* EMB7000 - Introduction aux systèmes embarqués                         */
/* Lab2 - Clavier et Afficheur LCD                                       */
/*                                                                       */
/* Exemple de base de temps                                              */
/*                                                                       */
/* Pour le MSP430FR6989                                                  */
/*************************************************************************/

#include <msp430fr6989.h>

#define COMPARE_VALUE 1000

void main(void) {
    // Stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;

    // Disable the GPIO power-on default high-impedance mode to activate
    // previously configured port settings
    PM5CTL0 &= ~LOCKLPM5;

    P1DIR |= 0x01; // Set P1.0 to output direction

    TA0CCTL0 = CCIE;                            // TACCR0 interrupt enabled
    TA0CTL = TASSEL_2 | MC_2;                   // SMCLKx1, continuous up
    TA0CCR0 = COMPARE_VALUE;

    __bis_SR_register(LPM0_bits | GIE);         // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR                 // set vector for interrupt
__interrupt void Timer_A (void)                 // routine declaration
{
    P1OUT ^= 0x01;                              // Toggle P1.0
```

```
    TA0CCR0 += COMPARE_VALUE;                    // Add Offset to TACCR0
}
```



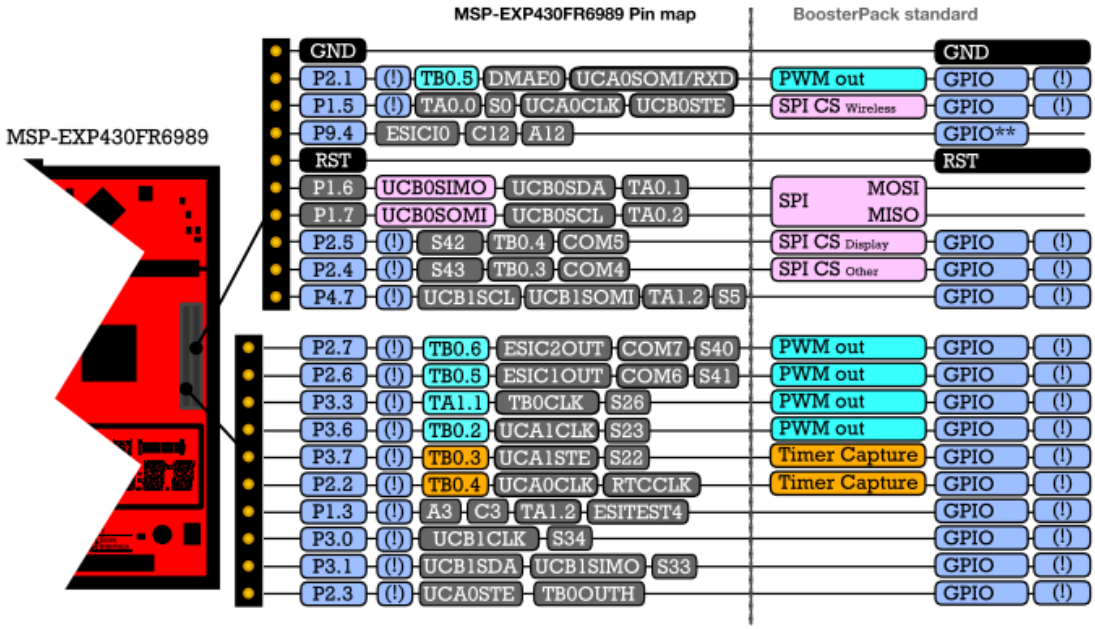
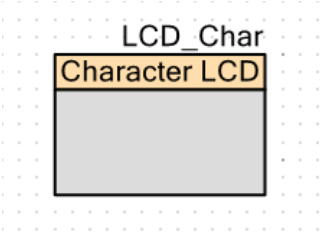


Schéma de raccordement du Kit de développement MSP-EXP430FR6989
(http://www.ti.com/lit/ug/slau627a/slau627a.pdf)

Annexe B : PSoC 5LP

Comme la carte d'extension a été spécialement conçue pour le kit de développement MSP-EXP430FR6989, il va falloir apporter quelques modifications aux laboratoires. En premier lieu, veuillez porter une **attention particulière aux schémas électriques** présentés dans les énoncés. Ils sont d'une très grande importance pour que les laboratoires fonctionnent sur le PSoC 5LP. Aussi, veuillez vous assurer que le PSoC 5LP et la carte d'extension sont **connectés à la même masse** : ceci est primordial.

<u>Exemple de programmation d'un afficheur à cristaux liquides</u>



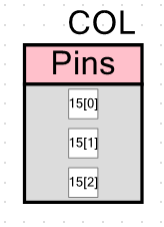
La première étape consiste à créer un projet et utiliser l'outil de génération automatique de code. Vous n'avez tout simplement qu'à créer une instance de *Character LCD*. Ensuite, vous utilisez l'option *Pins* dans le menu de gauche pour sélectionner le port à utiliser **Attention : Certaines broches contiennent des condensateurs de dérivation qui affecteront la communication, vous devrez donc changer de port/broches si ça ne fonctionne pas.** Cela fait, vous pouvez compiler le projet pour générer le pilote de l'afficheur LCD. Ensuite, vous devez modifier la fonction `void LCD_Char_IsReady(void)` se trouvant dans le fichier LCD_Char.c. Remplacer son contenu par le suivant :

```
void LCD_Char_IsReady(void)
{
    CyDelay(1);
    return;
}
```

Brièvement, le pilote généré utilise une communication bidirectionnelle entre le PSoC et l'afficheur LCD pour s'assurer que l'afficheur est prêt avant de lui envoyer des données; ceci nécessite le contrôle de la ligne R/$\overline{W}$ pour recevoir des données de l'afficheur LCD. Cependant, comme vous pouvez le voir dans la prochaine annexe, la ligne R/$\overline{W}$ est tout simplement relié à la masse, donc il est impossible de savoir quand l'afficheur est prêt à recevoir des données. Pour remédier à cela, il suffit d'attendre 1ms, ce qui est assez pour s'assurer du bon fonctionnement du pilote. Finalement, le code principal est très simple :

```
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    LCD_Char_Start();
    LCD_Char_ClearDisplay();
    LCD_Char_Position(0, 0);
    LCD_Char_PrintString("EMB7000 - Lab 2");

    for(;;) ;
}
```

Exemple de programmation d'un clavier



Vous devez premièrement créer une instance de broches d'entrées (pour les colonnes) et de sorties (pour les lignes). Cela peut se faire en créant un objet *Digital Input Pin* (ou *Output*) et en ajustant le nombre de broches adéquatement. Par exemple, on voit à la gauche l'objet *COL* qui sera utilisé pour lire les colonnes. Ensuite, les broches logiques peuvent être associées à des broches physiques en utilisant l'option *Pins* (notez que le port 15 est utilisé ici).

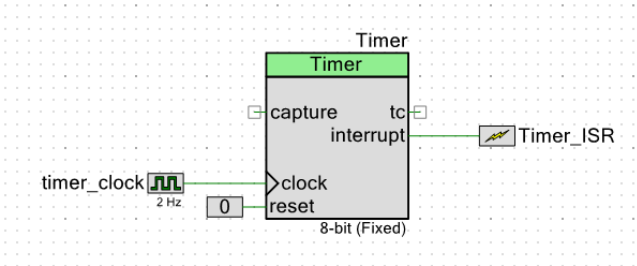
Ensuite, le code est comme suite :

```
/*************************************************************************/
/* EMB7000 - Introduction aux systèmes embarqués */
/* Lab2 - Clavier et Afficheur LCD */
/**/
/* Exemple de programmation d'un clavier */
/**/
/* Pour le CY8CKIT-059 PSoC 5LP */
/*************************************************************************/

#include "project.h"

/*
 *  Partage des lignes avec le LCD ---
 *  dans ce cas, l'objet LCD se nomme LCD_Char
*/
#define ROWS_OUT LCD_Char_PORT_DR_REG

#define ROWS_MASK 0x0f
#define ROW_1 0x01 // LCDPort[0] 0001
#define ROW_4 0x08 // LCDPort[3] 1000

#define COL1 0x01

unsigned char keyPadMatrix[] =
{
    '1','2','3',
    '4','5','6',
    '7','8','9',
    '*','0','#', 0xFF
};

/*********************************************************
 * scanClavier: Scan et retourne la première touche pesée *
 *  du clavier pendant le scan, sinon 0xff               *
 *********************************************************/
unsigned char scanClavier() {
    unsigned char key = 0, row, col;

    for(row = ROW_1; row <= ROW_4; row <<= 1) // Suppose que ROW_4 est à un
bit plus significatif que ROW_1
    {
        ROWS_OUT &= ~ROWS_MASK;; // Mettre toutes les lignes à 0
        ROWS_OUT |= row;  // Propager un 1 sur une ligne

        // Lire les colonnes et arrêter si un 1 détecté
```

```
        col = COL_Read();
        if(col & COL1)
            break; // Colonne 1
        key++;

        if(col & (COL1 << 1)) // COL2 = COL1 << 1
            break; // Colonne 2
        key++;

        if(col & (COL1 << 2)) // COL3 = COL1 << 2
            break; // Colonne 3
        key++;
    }

    return keyPadMatrix[key];
}


/*********************************************************
 * scanTouche: Attend la première touche pesée du clavier *
 *  et retourne sa valeur après l'écoulement du délai     *
 *  de rebondissement.                                    *
 *********************************************************/
unsigned char scanTouche() {
    unsigned char touche=0xff;

    while (touche==0xff) {
        touche = scanClavier();

        if (touche != 0xff) // Touche pesée
        {
            CyDelay(10); // Délai de rebondissement (~10 ms)

            if (touche != scanClavier())
                touche = 0xff; // La touche a changée pendant le délai
        }
    }

    return touche ;
}


int main(void) {
    CyGlobalIntEnable; /* Enable global interrupts. */

    unsigned char touche = scanTouche(); // La variable 'touche' contient
maintenant le code ASCII de la touche appuyée
}
```

Exemple de base de temps

Le bloc à utiliser est le *Timer* et il faut lui fournir une horloge appropriée avec une période associée pour générer une interruption chaque seconde. Également, les interruptions doivent se faire *On TC* puis raccorder une routine de service d'interruption, comme illustré à la figure de droite. Ensuite, le code pour faire clignoter une DEL reste assez simple :




```
/*************************************************************************/
/* EMB7000 - Introduction aux systèmes embarqués */
/* Lab2 - Clavier et Afficheur LCD */
/**/
/* Exemple de base de temps */
/**/
/* Pour le CY8CKIT-059 PSoC 5LP */
/*************************************************************************/

#include "project.h"

CY_ISR(InterruptHandler)
{
   /* Read Status register in order to clear the sticky Terminal Count (TC)
bit
    * in the status register. Note that the function is not called, but
rather
    * the status is read directly. */
      Timer_STATUS;

   LED_Pin_Write(~LED_Pin_Read()); // Toggle LED
}

int main(void) {
    CyGlobalIntEnable; /* Enable global interrupts. */

    Timer_ISR_StartEx(InterruptHandler);
    Timer_Start();

    for(;;) ;
}
```

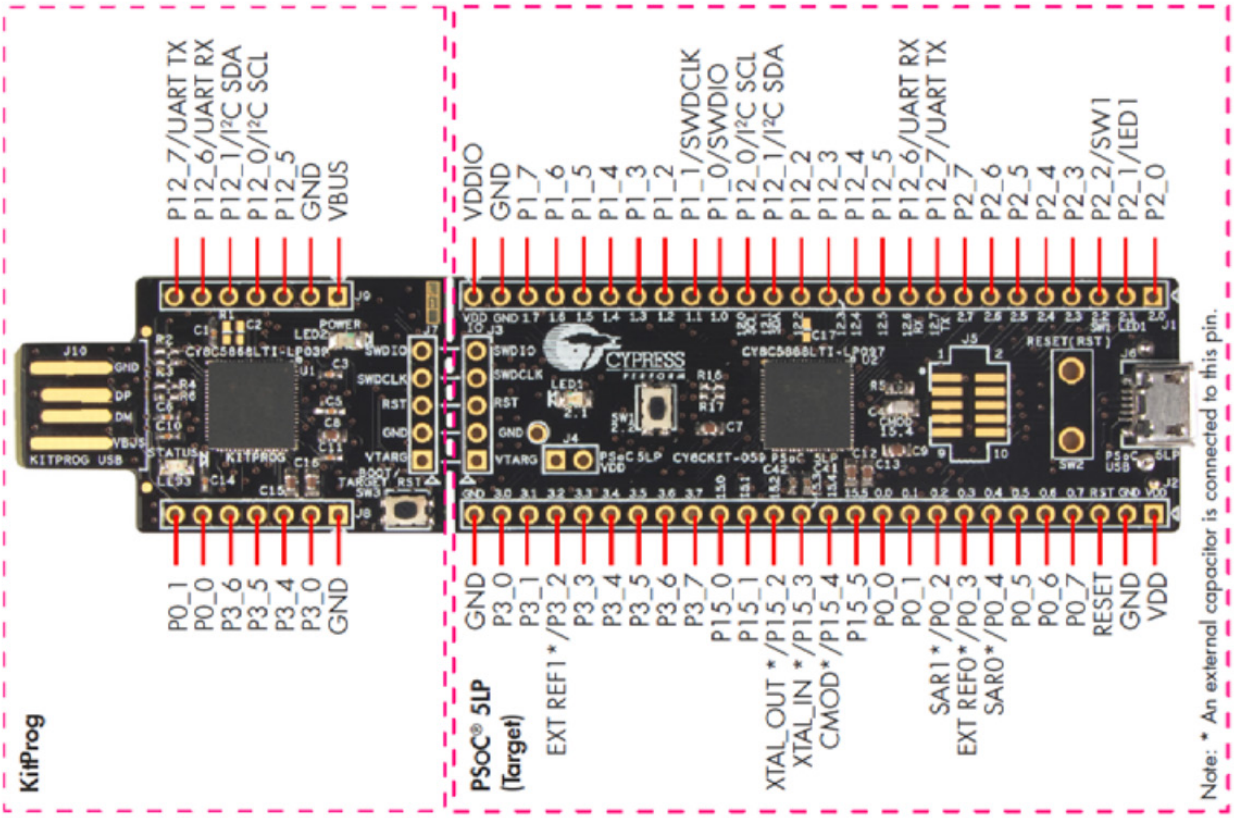
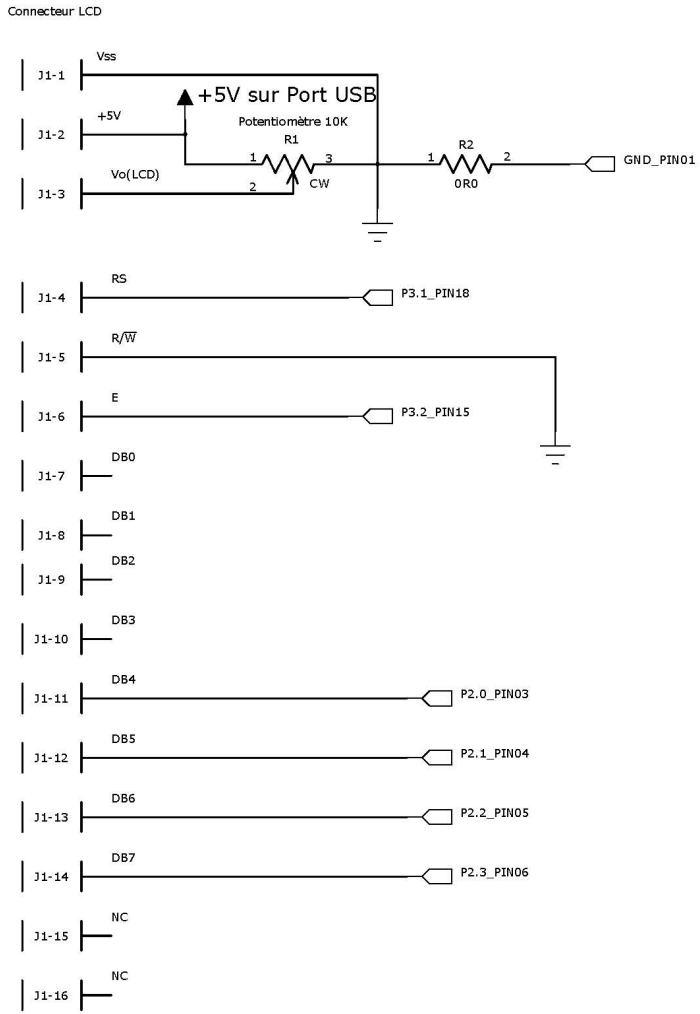
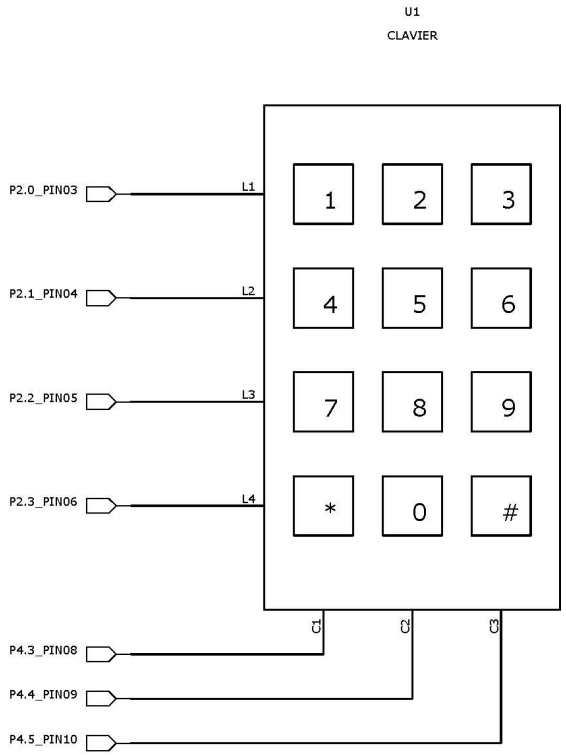


Schéma de raccordement du Kit de développement CY8CKIT-059 PSoC 5LP
(https://www.cypress.com/file/157971/download)

Annexe C : Schéma Électrique





Vue côté Clavier 4x3

NC L3 L4 C1 L1 C2 L2 C3