

TP4

**Application pratique des files d'attente
avec les périphériques du PIC24F****1. Objectif :**

Ce TP a comme objectifs :

- Développer une application basée sur freertos pour gérer des entrées analogiques, un clavier, une communication série asynchrone et un affichage sur LCD.
- mettre en œuvre les concepts de la communication des tâches à travers les files (Queues).

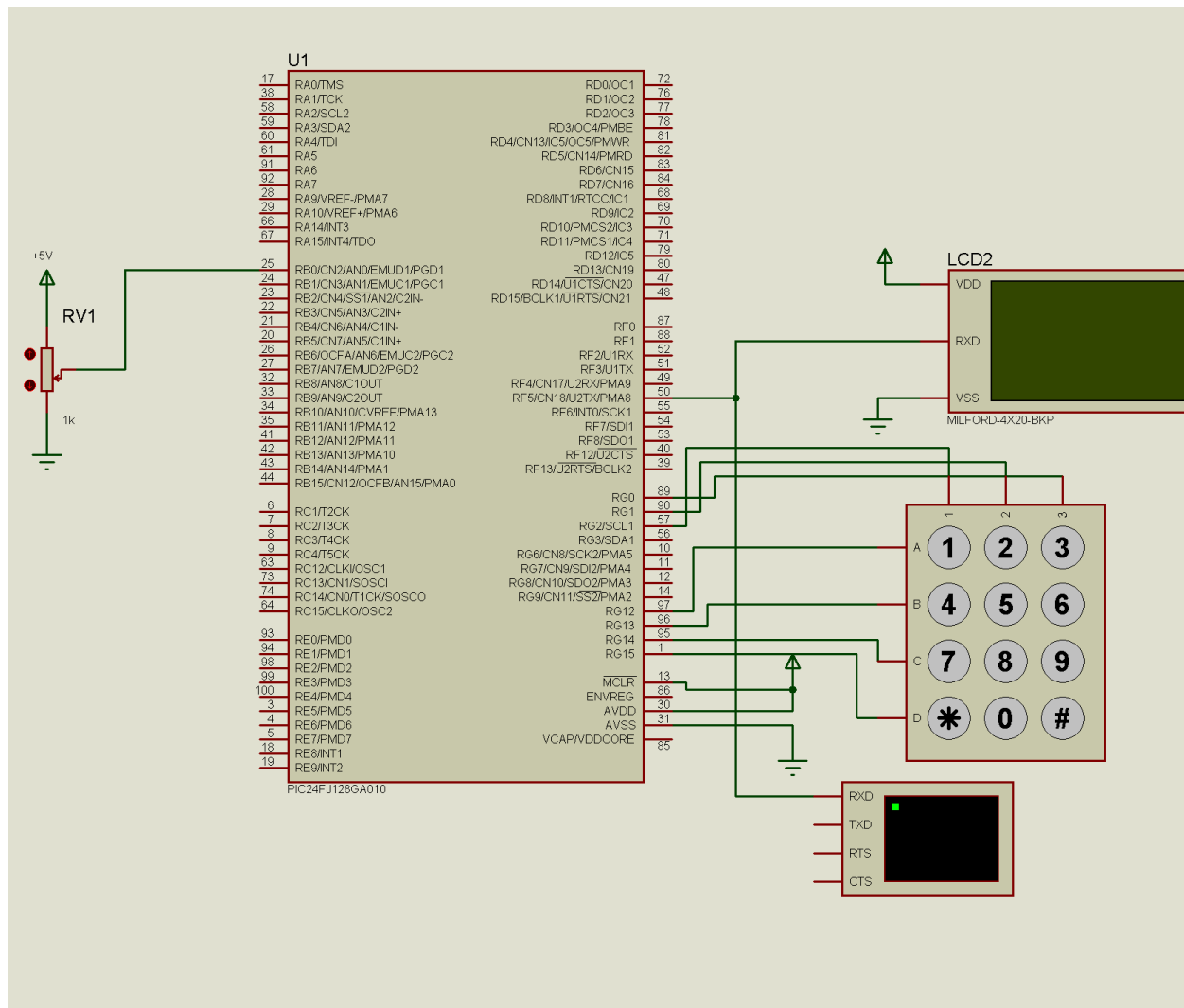
2. Mode opératoire :**2.1. Outils de travail :**

Dans ce TP vous aurez besoin, en plus de freertos que vous avez déjà téléchargé pour les premiers TP de :

- un environnement de développement intégré IDE(Integrated Development Environment), dans ce TP on utilisera le dernier IDE de Microchip Mplab X.
- Un compilateur pour les pic 16 bits, on utilisera le compilateur xc16 fourni par Microchip.
- Isis Proteus (si vous avez la licence !) ou une carte didactique avec un pic 24F comme par exemple **The Explorer 16 Development Board** conçue par microchip.
- Datasheet du microcontrôleur PIC24FJ128GA010 (à télécharger).
- la librairie de périphérique pour PIC24F déjà installée (ceci a été fait en TP3).

2.2. Réalisation sur ISIS de la plateforme de travail :**1. Créez sur ISIS le design suivant**

Dans ce design on utilise un afficheur LCD 4 lignes 20 colonnes (référence MILFORD-4X20-BKP, caractéristique en annexe). L'HyperTerminal ajouté en parallèle avec cet afficheur est utilisé juste pour le débogage : pour vérifier si le microcontrôleur envoie les données correctement à l'afficheur.



2.3. Application multitâche utilisant une file d'attente:

Le but est de développer une application basée sur freertos, contenant trois tâches :

- Une tâche, **vLCDTask** qui permet de récupérer des items à partir d'une file **xLCDQueue** et afficher les messages textuels qu'ils contiennent sur l'une des 4 lignes de l'afficheur LCD.
- Une tâche, **vCapteurTask**, qui convertie le signal de l'entrée analogique RB0 (AN0), venant du potentiomètre (simulant un capteur), en numérique sur 10 bits, puis ramène la valeur mesurée à un pourcentage de sa valeur maximale et envoie un message de type « Capteur 30% » vers la file **xLCDQueue** pour qu'il soit récupéré et affiché sur la ligne 1 par la tâche **vLCDTask**.
- Une tâche, **vClavierTask**, scrute le clavier et envoie un item contenant le message de type « Touche X » (X est numéro de la touche appuyée 0,1,..., *, #), vers la file **xLCDQueue** pour qu'il soit récupéré et affiché sur la ligne 4 par la tâche **vLCDTask**.

2. Faites une copie du projet template créée dans le TP1, et la renommez TP4.
3. Ajoutez au projet les fichiers **Lcd.c** et **Lcd.h** qui vous sont fournis : Des informations complémentaires sur le fonctionnement du LCD (4 lignes) sont données en Annexe1, en plus des commentaires dans le code source des fichiers **lcd.h** et **lcd.c** qui vous sont fournis.

4. Ajoutez dans le fichier **main.c** les directives :

```
#include "queue.h" //pour pouvoir utiliser les queues
```

```
#include "lcd.h" //pour pouvoir utiliser les macros/fonctions de commandes du LCD
```

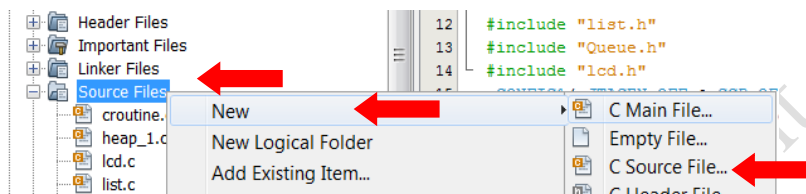
5. les trois tâches **vClavierTask**, **vLCDTask**, **vCapteurTask** manipuleront des messages à travers la file **xLCDQueue** : ajoutez au projet un fichier d'entête **LCD_UART.h** dont le contenu sera le suivant :

```

1  /*
2  * File:    LCD_UART.h
3  * Author:  Y.ROCHDI
4  * Created: Mars 2019,
5  */
6  /**** type des items echanges via xLCDQueue *****/
7  typedef struct {
8      unsigned short xRow; // numero de la ligne ou il faut afficher
9      char *pc;           // adresse du message a afficher sur la ligne xRow
10 } xLCDMessage;
11

```

6. Ajoutez un nouveau fichier **lcdtask.c** (click droit sur le dossier **Sources Files**, puis **New**, puis **C Source File**),



Dans ce fichier on ajoute les fichiers d'entête nécessaire et on définit une fonction **init_UART2()** qui configure le port UART2 connecté au LCD comme suit :

```

1  #define USE_AND_OR //pour pouvoir utiliser la librairie de peripheriques
2  #include <uart.h> // pour pouvoir utiliser la librairie UART
3  #include "lcd.h"
4  #include "FreeRTOS.h"
5  #include "task.h" //pour pouvoir créer ici la tâche vLCDTask()
6  #include "queue.h" //pour pouvoir créer la file de handle xLCDQueue
7  #include "lcd_uart.h" //pour pouvoir travailler avec le type xLCDMessage
8  QueueHandle_t xLCDQueue; //declaration du handle de la file
9
10 /* On suppose que le LCD est connecte a l'UART2 */
11 void init_UART2(void) {
12     CloseUART2(); /*desactive le port UART2 et desactive les interruptions pour cet UART*/
13     ConfigIntUART2(UART_RX_INT_DIS | UART_TX_INT_DIS); /*desactive les interruptions sur UART*/
14     /*UART initialized to 9600 baudrate @BRGH=0, 8bit,no parity and 1 stopbit*/
15     OpenUART2(UART_EN, UART_TX_ENABLE, 25); /*voir daasheet : 25 --> 9600 baud*/
16 }

```

7. Ajoutez à ce fichier la fonction **vLCDTask()** qui permet d'afficher en boucle sur les 4 lignes des messages, de type "Line 1..... !" sur la ligne 1, puis "Line 2 : !" sur la ligne 2 ...etc :

```

void vLCDTask(void *p) {
    xLCDMessage M; //un message
    char genericMessage[] = "Line X .....!";
    M.pc = genericMessage;
    unsigned short i = 0;

```

```

for (;;) {
    M.xRow = i + 1;           //numero de la ligne 1,2,3 ou 4
    *(M.pc + 5) = '1' + i;    //remplace X dans genericMessage by char '1', '2', '3' or '4',
    i = (i + 1) % 4;          //modulo --> i=0,1,2,3,0,1,2,3 ..etc
    LCD_set_address(M.xRow, 0); //aller a la position 0 dans ligne i+1
    putsUART2(M.pc);          // envoie le message pointé par M.pc via le port UART2
    vTaskDelay(1000);         //se bloque pendant 1000 ticks
    LCD_clear();              //efface le LCD
}
}

```

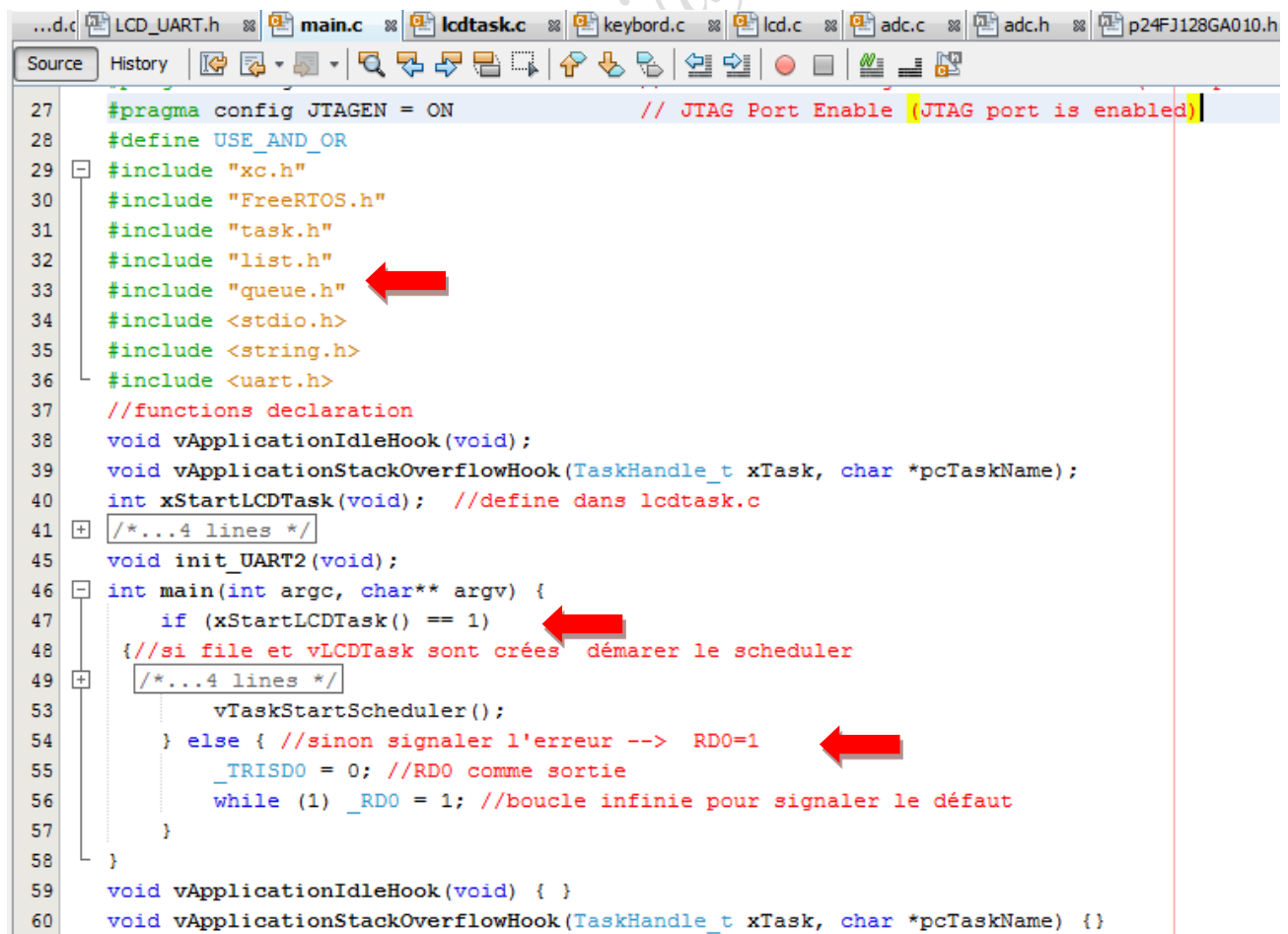
8. Complétez le fichier **lcdtask.c** par une fonction **xstartLCDTask()** permettant de créer la file de handle **xLCDQueue**, puis crée la tâche **vLCDTask()**, cette fonction est définie comme suit :

```

int xStartLCDTask(void) {
    init_UART2();    //initialiser le port UART1
    LCD_init();      // attente que le LCD soit initialisé
    xLCDQueue = xQueueCreate(4, sizeof(xLCDMessage)); //création de la file
    if ((xLCDQueue != NULL) && (xTaskCreate(vLCDTask, "LCD", 200, NULL, 2, NULL) == pdPASS))
        return (1); //retourne 1 si la file et la tâche sont créées
    else return (0); //retourne 0 sinon
}

```

9. Modifiez le contenu de **main.c** de telle sorte à ce qu'il ressemble au code suivant :



```

...d.c LCD_UART.h main.c lcdtask.c keyboard.c lcd.c adc.c adc.h p24FJ128GA010.h
Source History
27 #pragma config JTAGEN = ON // JTAG Port Enable (JTAG port is enabled)
28 #define USE_AND_OR
29 #include "xc.h"
30 #include "FreeRTOS.h"
31 #include "task.h"
32 #include "list.h"
33 #include "queue.h"
34 #include <stdio.h>
35 #include <string.h>
36 #include <uart.h>
37 //functions declaration
38 void vApplicationIdleHook(void);
39 void vApplicationStackOverflowHook(TaskHandle_t xTask, char *pcTaskName);
40 int xStartLCDTask(void); //define dans lcdtask.c
41 /*...4 lines */
45 void init_UART2(void);
46 int main(int argc, char** argv) {
47     if (xStartLCDTask() == 1)
48     { //si file et vLCDTask sont créées démarrer le scheduler
49     /*...4 lines */
53         vTaskStartScheduler();
54     } else { //sinon signaler l'erreur --> RD0=1
55         _TRISD0 = 0; //RD0 comme sortie
56         while (1) _RD0 = 1; //boucle infinie pour signaler le défaut
57     }
58 }
59 void vApplicationIdleHook(void) { }
60 void vApplicationStackOverflowHook(TaskHandle_t xTask, char *pcTaskName) {}

```



```

220 PORTG = 0x0004; //Keyboard's column 1 is on
221 switch (PORTG & 0xF000) {
222     case 0x1000:
223     {
224         while ((PORTG & 0x1000));
225         return '1';
226     }
227     case 0x2000:
228     {
229         while ((PORTG & 0x2000));
230         return '4';
231     }
232     case 0x4000:
233     {
234         while ((PORTG & 0x4000));
235         return '7';
236     }
237     case 0x8000:
238     {
239         while ((PORTG & 0x8000));
240         return '*';
241     }
242 }
243 return 'E'; //error more than one key are pressed or any key pressed
244 }

```

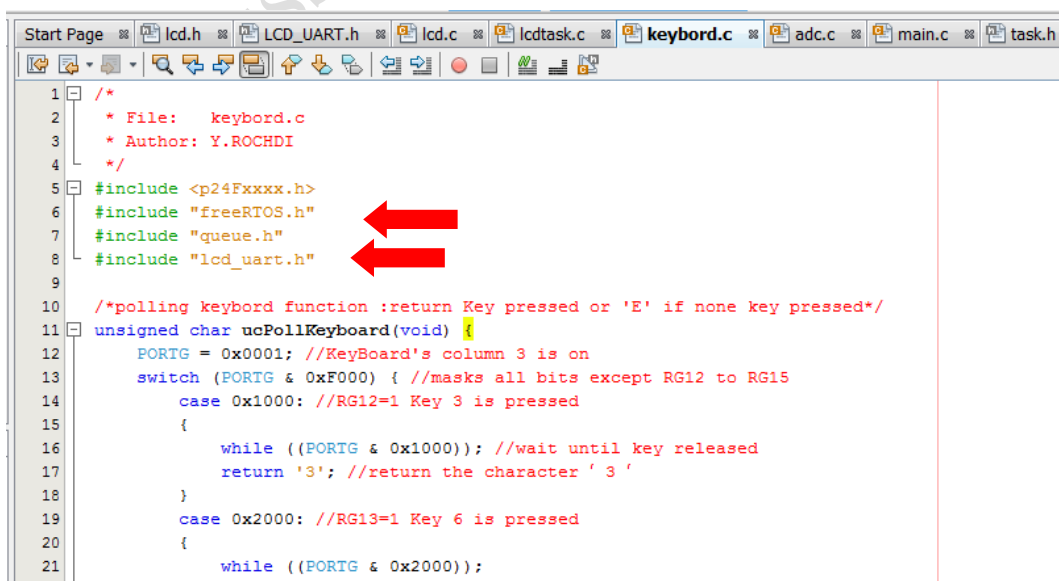
12. Ajoutez dans ce fichier **keyboard.c** une fonction tâche de gestion du clavier **vClavierTask()**

```

83
84 void vClavierTask(void *pvParameters) {
85     unsigned char C;
86     char pcString[10] = "Touche: "; //Message générique
87     xLCDMessage xMessage;
88     xMessage.pc=pcString;
89     xMessage.xRow=4; //ligne 4
90     TRISG = 0xFFFF8; //Config PORTG
91     while (1) {
92         C = ucPollKeyboard(); //scruter le clavier pour detecter la touche appuyée
93         if (C != 'E') { //une touche appuyée
94             xMessage.pc[8]=C; // insérer le caractère de cette touche dans le message générique
95             xQueueSendToBack( xLCDQueue, &xMessage, portMAX_DELAY ); //envoyer ce message à la file
96         } else; //nothing to do
97     }
98 }

```

NB : cette fonction utilise le handle **xLCDQueue** et le type **xMessage** qui sont définis dans **lcd_uart.h**, il faut donc ajouter à **keyboard.c**, **#include " lcd_uart.h "**



```

1 /*
2  * File:   keyboard.c
3  * Author: Y.ROCHDI
4  */
5 #include <p24Fxxx.h>
6 #include "freeRTOS.h"
7 #include "queue.h"
8 #include "lcd_uart.h"
9
10 /*polling keyboard function :return Key pressed or 'E' if none key pressed*/
11 unsigned char ucPollKeyboard(void) {
12     PORTG = 0x0001; //Keyboard's column 3 is on
13     switch (PORTG & 0xF000) { //masks all bits except RG12 to RG15
14         case 0x1000: //RG12=1 Key 3 is pressed
15         {
16             while ((PORTG & 0x1000)); //wait until key released
17             return '3'; //return the character ' 3 '
18         }
19         case 0x2000: //RG13=1 Key 6 is pressed
20         {
21             while ((PORTG & 0x2000));

```

13. Modifiez le code de la fonction tâche **vLCDTask()** comme suit, pour qu'elle puisse recevoir les messages à partir de la file **xLCDQueue** et les afficher sur la ligne du LCD indiquée par le champ **xRow** dans le message:

```

18 void vLCDTask(void *p) {
19     xLCDMessage M; //un message
20     /*...11 lines */
31     while(1){
32         xQueueReceive(xLCDQueue, &M, portMAX_DELAY); //attendre l'arrivee d'un message
33         LCD_set_address(M.xRow , 0); //aller a la position 0 dans la ligne M.xRow
34         putsUART2(M.pc); //afficher le message
35     }
36 }

```

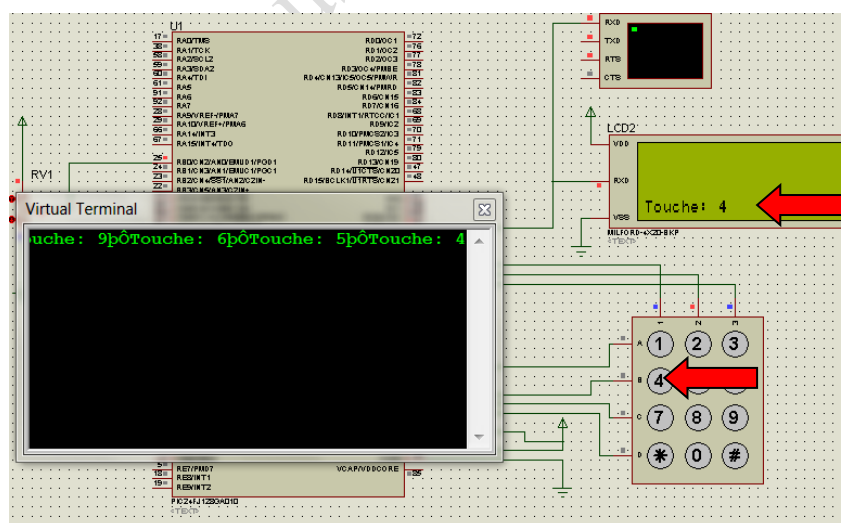
14. Complétez le fichier main par la création de la tâche **vClavierTask** de priorité 1.

```

17 void vApplicationIdleHook(void); //to execute some code when idle task is running
18 int xStartLCDTask(void );
19 /*...*/
26 /*...*/
27 void vClavierTask(void *pvParameters);
30 int main(int argc, char** argv) {
31     if (xStartLCDTask() == 1) //si file et vLCDTask sont créés démarrer le scheduler
32     /*...*/
39     { xTaskCreate(vClavierTask, "Kbd", 150, NULL, 1, NULL); //créer la tâche clavier
40         vTaskStartScheduler();
41     }
42     else { //sinon signaler l'erreur
43         _TRISD0 = 0; //RD0 comme sortie
44         while (1) _RD0 = 1; //boucle infinie pour signaler le défaut
45     }
46 }

```

15. Construisez le projet et tester le résultat qui doit être comme suit si on appuie sur la touche 4:



16. On définit maintenant la fonction tâche **vCapteurTask()** qui utilise le convertisseur analogique numérique du PIC pour convertir la tension analogique entre 0 et 5V au niveau de AN0 (soit RB0) en numérique et l'afficher concaténée à la chaîne "Capteur : ". Au fait, le convertisseur fait 16 conversions successives et prend leur moyenne (pour minimiser l'effet d'un bruit sur la mesure). Pour plus d'informations sur l'utilisation de ce convertisseur analogique numérique consultez le fichier :

Microchip\xc16\v1.22\docs\periph_libs\Microchip PIC24F Peripheral Library.chm

Pour l'utilisation de `sprintf` déclarée dans `stdio.h`, consultez le fichier

C:\Program Files (x86)\Microchip\xc16\v1.22\docs\16-Bit_Language_Tools_Libraries_Manual.pdf

Ajoutez un nouveau fichier source à votre projet et le nommez **adc.c**, et y insérez le code suivant par copiez-collez:

```
#define USE_AND_OR /* To enable AND_OR mask setting */
#define USE_AND_OR /* To enable AND_OR mask setting */
#include <adc.h> //pour pouvoir utiliser les fonctions de lib de pÃ©riph du PIC24 relatives Ã  ADC
#include <stdio.h>
#include <xc.h>
#include "freertos.h"
#include "task.h"
#include "queue.h"
#include "lcd_uart.h" //pour inclure la dÃ©finition le type xLCDMessage et inclure la dÃ©claration externe de xLCDQueue
extern QueueHandle_t xLCDQueue;
void vCapteurTask(void *p) {
    UINT i = 0; //UINT8 type definie par macro, equivalent a unsigned char , dans GenericTypeDefs.h
    UINT channel, config1, config2, config3, configport, configscan; //registres de config de l'ADC voir datasheet
    UINT ADCResult[16], mesure, oldmesure, cumul;
    xLCDMessage xMessage1; //le type xLCDMessage est declaree dans lcd_uart.h inclu precedemment
    xMessage1.xRow = 1; //afficher dans la premiere ligne du LCD
    oldmesure = 0; //ancienne mesure
    mesure = 0; //nouvelle mesure
    char sbuf[20], s[] = "Capteur:", blanc[21] = " "; //20 blancs pour effacer une ligne du LCD
    char strvalue[5]; //chaîne XYZ%
    unsigned long int temp; //pour faire la conversion de la mesure en %
    CloseADC10(); //turns off the ADC module and disables the ADC interrupts
    //configuration ADC pour:
    // conversion multiple du canal AN0 (16 conversions successives)
    //Liaisons du canal0 de l'ADC (S/H positif et négatif)
    channel = ADC_CH0_POS_SAMPLEA_AN0|ADC_CH0_NEG_SAMPLEA_VREFN; //registre AD1CHS datasheet
    SetChanADC10(channel);
    AD1PCFG=0; TRISB=0x01;
    config1 = ADC_MODULE_OFF | ADC_IDLE_STOP | ADC_FORMAT_INTG | ADC_CLK_AUTO |
    ADC_AUTO_SAMPLING_OFF|ADC_SAMP_OFF ;
    config2 = ADC_VREF_AVDD_AVSS| ADC_SCAN_OFF | ADC_INTR_16_CONV |ADC_ALT_BUF_OFF
|ADC_ALT_INPUT_OFF ;
    config3 = ADC_CONV_CLK_SYSTEM|ADC_SAMPLE_TIME_31 | ADC_CONV_CLK_20Tcy;

    configport=ENABLE_AN0_ANA|ENABLE_AN1_DIG|ENABLE_AN2_DIG|ENABLE_AN3_DIG|ENABLE_AN4_DIG|ENABLE
_AN5_DIG|ENABLE_AN6_DIG;
    configscan = 0; //ignoré puisque ADC_SCAN_OFF dans config2
    //application de la config
    OpenADC10(config1, config2, config3, configport, configscan); //configurer ADC avec les paramètres choisis
    EnableADC1; //activer l'ADC
    while (1) {
        oldmesure = mesure; //la nouvelle mesure devienne l'ancienne mesure
        cumul = (UINT) 0; //le cumul est initialise a 0
        IFS0bits.AD1IF=0; //mettre le flag d'interruption à 0
```



```

AD1CON1bits.ASAM=1; //lancer auto sampling
//conversion sera lancée auto après 31Tad
while (IFS0bits.AD1IF==0){}; //attendre la fin des 16 sampling/conversions successives
AD1CON1bits.ASAM=0; //arreter le sampling auto pour récupérer les 16 mesures
i = 0;
while (i < 16) { //calcul du cumul
    ADCResult[i] = ReadADC10(i); //lire la ieme mesure dans ADCResult[i]
    cumul = cumul + ADCResult[i]; //ajouter cette mesure au cumul
    i++;
}
//obtenir la moyenne des mesures par division par 16
mesure = cumul >> 4; // division by 16=2^4 <=> decalage gauche de 4 bits: mesure=cumul/16

//conversion en % ; l'ADC ayant 10bits donc la mesure doit etre comprise entre 0 et 1023
if ((mesure <= (UINT) 1023) && (mesure >= (UINT) 0)) {
    temp = ((unsigned long int) mesure * (unsigned long int) 100);
    //1023*100=100 023 depasse la capacitee d'un unsigned int d'ou la conversion en long int
    //en principe on doit apres diviser par 1023, mais il est plus facile et rapide de diviser par 1024
    //on commet alors une erreur qui reste negligeable
    //au lieu de faire temp=temp/((long int) 1023); on fait
    temp = (UINT32) (temp >> (UINT) 10); //division by 1024=2^10 --> decalage de 10 bit a gauche
    //temp=temp/((long int) 1023);
    if ((temp <= (UINT32) 100) && (temp >= (UINT32) 0))
        mesure = (UINT) temp;
}
if (oldmesure != mesure) { //si la mesure precedente est differente de la nouvelle--> changer l'affichage

    xMessage1.pc = blanc; // message blanc pour effacer l'ancien message
    xQueueSendToBack(xLCDQueue, &xMessage1, portMAX_DELAY); //envoyer message
    vTaskDelay(50); /*attendre que le premier message soit completement efface*/
    sprintf((char *) sbuf, "%s%3d%s", s, mesure, "%"); //concatener "capteur:" avec mesure et "%" dans sbuf
    xMessage1.pc = sbuf; //message contenant la nouvelle mesure
    xQueueSendToBack(xLCDQueue, &xMessage1, portMAX_DELAY);
    vTaskDelay(50); /*attendre que le nouveau message soit completement affiche*/
}
}
CloseADC10();
}

```

17. Ajoutez à la fonction main() du projet la déclaration externe et la création de la tâche vCapteurTask() comme suit :

```

26  /*...*/
29  void vClavierTask(void *pvParameters);
30  void vCapteurTask(void *pvParameters); ←
31  int main(int argc, char** argv) {
32      if (xStartLCDTask() == 1) //si file et vLCDTask sont créés démarrer le scheduler
33          /*...*/
40      { xTaskCreate(vClavierTask, "Kbd", 150, NULL, 1, NULL); //créer la tâche clavier
41        xTaskCreate(vCapteurTask, "Capt", 150, NULL, 1, NULL); ←
42        vTaskStartScheduler();
43      }
44      else { //sinon signaler l'erreur
45          _TRISD0 = 0; //RD0 comme sortie
46          while (1) RD0 = 1; //boucle infinie pour signaler le défaut
47      }
48  }

```

18. Construisez et testez votre projet sur ISIS tout en faisant varier le potentiomètre.
 19. L'affichage sur la première ligne du lcd donne la valeur numérique sur 10bits correspondante à la tension allant de 0 à 5V en pourcent: soit une valeur numérique allant de 0% à 99%. Modifiez le code pour afficher

à la place du pourcentage par rapport à sa valeur maximale, la température sous forme 56°C sachant que 0V correspond à 0°C et 5V correspond à 200°C.

Exercice d'application :

En vous inspirant du code précédent et du TP3 ajoutez au projet précédant deux tâches une vUART1RxTask qui reçoit des caractères sur la ligne série (U1RX) du PIC venant d'un hyperterminal et les envoie à une file d'attente UARTQueue, et la deuxième vUART1TxTask, qui lie les caractères à partir de cette file et les transmet sur la ligne série (U1Tx) du PIC vers un autre hyperterminal. Testez le projet sur ISIS.

Annexe1 : Afficheur LCD

La plus part des afficheurs LCD (2 ou 4 lignes) utilisent une matrice d'affichage, une mémoire RAM pour stocker les codes des caractères à afficher, et une mémoire ROM contenant les matrices de points pour dessiner chaque caractère ou symbole, et un Microcontrôleur HITACHI 44780 (datasheet à télécharger à partir de <https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>).

Par-dessus ce système minimal, certains afficheurs LCD ajoutent une interface parallèle (Bus de données + bus de contrôle) ou une interface série de type UART, ou SPI, ou I2C,....

L'afficheur est contrôlé à travers deux registres du HITACHI 44780 , qui sont le registre d'instruction IR et le registre de données DR. Pour exécuter une instruction comme décaler l'affichage vers la gauche ou la droite, déplacer le curseur vers l'avant ou vers l'arrière, écrire/lire en/à partir de la RAM, ...il faut écrire son code (de l'instruction) dans le registre IR. Le registre de données DR reçoit par exemple le code du caractère à écrire vers, ou lu à partir de, la RAM.

NB : l'exécution de chaque instruction par HITACHI 44780 nécessite un certain temps spécifié dans le datasheet, qu'il faut respecter.

Table des instructions

Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)	
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.		
Return home	0	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s
Display on/off control	0	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μ s
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μ s
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μ s
Write data to CG or DDRAM	1	0	Write data									Writes data into DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu s^*$
Read data from CG or DDRAM	1	1	Read data									Reads data from DDRAM or CGRAM.	37 μ s $t_{ADD} = 4 \mu s^*$
<div>I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 \times 10 dots, F = 0: 5 \times 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable</div>												DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{osc} is 250 kHz, $37 \mu s \times \frac{270}{250} = 40 \mu s$

Note: — indicates no effect.

- * After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.