7 Voltmètre avec la sortie série RS232

Description générale

- **Date du TP :** 14ième semaine ;
- **Durée du TP :** 3 h (2 semaines) ;
- **Matériels nécessaires :** Carte d'expérimentation, microcontrôleur PIC 18f, programmateur, des composants électroniques, plaque d'essai ;
- **Logiciels utilisés :** RTOS, Complilateur C PCW, Simulateur ISIS, IC Writer ;
- **Public cible :** 4ième niveau d'études de technicien supérieur en informatique industrielle ;
- **Partie du cours en rapport avec le TP :** Classification logicielle des systèmes temps réel, systèmes multitâches.

Objectifs

- Assimiler les systèmes d'exploitation temps réel RTOS ;
- Manipuler les semaphores ;
- Approfondir les acquis en programmation C des microcontrôleurs PIC ;
- Maîtrise des architectures des microcontrôleurs Microchip PIC18.

Pré requis

Notions générales sur :
- Système d'exploitation ;
- L'électronique ;
- Algorithmique ;
- Programmation en C ;
- Les microcontrôleurs.

7.1 présentation du projet

Dans ce projet de RTOS, qui est plus complexe que le précédent, la tension est lu à l'aide d'un convertisseur ANALOGIQUE-NUMÉRIQUE A/D et envoyé via le port série à un PC.
Le projet se compose de trois tâches : Live, Get_voltage et To_RS232.
- La tâche Live s'exécute chaque 200ms et elle sert à clignoter la diode LED connectée au pin RD7 du microcontrôleur pour indiquer que le système fonctionne.
- La tâche Get_voltage lit le canal 0 du convertisseur A/D où la tension à mesuré est connecté. La valeur lue est formatée et puis enregistrée dans une variable. Cette tâche s'exécute toutes les deux secondes.
- La tâche To_RS232 permet la lecture de la tension formatée et l'envoie de cette donnée via la liaison RS232 à un PC chaque seconde.

La figure 7.1 montre le schéma fonctionnel du projet. Le schéma de circuit est donné dans La figure 7.2.
Un microcontrôleur de type PIC18F8520 avec un cristal 10MHz est utilisé dans ce projet (bien que n'importe quel microcontrôleur de la série PIC18F peut être utilisé). La tension à mesurée est relié au port analogique AN0 du microcontrôleur. La sortie de RS232 TX du microcontrôleur (RC6) est relié à un circuit convertisseur de niveau : MAX232 , et puis à l'entrée DB9 (port série) d'un PC (par exemple, COM1). La pin RD7 est relié à une LED pour indiquer si le projet fonctionne.

FIG. 7.1 – le schéma fonctionnel du projet

La programme du projet est donnée dans le listing 8.
Au début du programme, l'A/D est configurée à 10 bits, l'horloge est définie à 10MHz, et la vitesse de la liaison RS232 est définie en tant que 2400 bauds. Le temporisateur de RTOS et le minor_cycle sont alors définit par la commande de préprocesseur #use rtos.
Dans le main du programme, le PORTD est configuré comme sortie et tous les pin du PORTD sont l'état logique bas. Alors que, le PORTA est configuré comme entrée (RA0 est

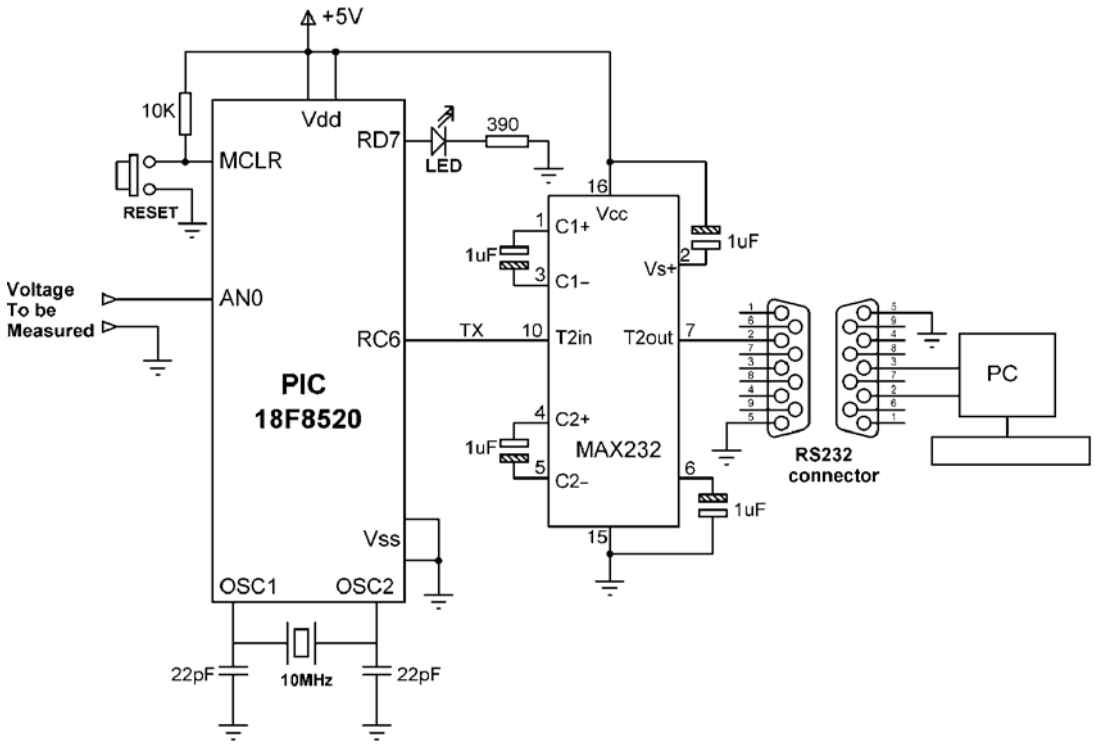

FIG. 7.2 – Le circuit du projet

l'entrée analogique), les entrées analogiques du microcontrôleur sont configurées, l'horloge de l'A/D est réglée, et le canal 0 de l'A/D (AN0) est choisi. Le RTOS est alors exécutée en appelant la fonction rtos_run ().

Le programme se compose de trois tâches :
La tâche LIVE qui s'exécute chaque 200ms et clignote la LED relier au pin RD7 du microcontrôleur pour indiquer que le projet fonctionne.
La tâche Get_voltage lit la tension analogique du canal 0 (pin RA0 ou AN0) du microcontrôleur. La valeur est alors convertie en millivolts en se multipliant par 5000 et la division par 1024 (dans un A/D de 10-bit il y a 1024 niveaux de quantification, et en travaillant avec une tension de référence de 5V, chaque niveau de quantification correspond à 5000/1024mV). La tension est stockée dans une variable globale appelé Volts.
la tâche To_RS232 lit la tension mesurée de la variable Volts et puis elle l'envoie vers le port de la liaison RS232 en utilisant la fonction printf de C. Le résultat est envoyé sous le format suivant :

$$Measured\ voltage = nnnn\ mV$$

Le programme de HyperTerminal est exécuté sur le PC pour obtenir le résultat du programme. Un résultat sur l'écran de HyperTerminal est montré sur la figure 7.3.

```
//*****************************listing8*****************************//
#include <18F8520.h>
#device adc=10
#use delay (clock=10000000)
#use rs232(baud=2400,xmit=PIN_C6,rcv=PIN_C7)
unsigned int16 adc_value;
unsigned int32 Volts;
// Define which timer to use and minor_cycle for RTOS
#use rtos(timer=0, minor_cycle=100ms)
// Declare TASK "Live" - called every 200ms
// This task flashes the LED on port RD7
#task(rate=200ms, max=1ms)
void Live()
{
output_toggle(PIN_D7); // Toggle RD7 LED
}
//
// Declare TASK "Get_voltage" - called every 10ms
#task(rate=2s, max=100ms)
void Get_voltage()
{
adc_value = read_adc(); // Read A/D value
Volts = (unsigned int32)adc_value*5000;
Volts = Volts / 1024; // Voltage in mV
}
//
// Declare TASK "To_RS232" - called every millisecond
#task(rate=2s, max=100ms)
void To_RS232()
{
printf("Measured Voltage = %LumV\n\r",Volts); // send to RS232
}
//
// Start of MAIN program
void main()
{
set_tris_d(0); // PORTD all outputs
output_d(0); // Clear PORTD
set_tris_a(0xFF); // PORTA all inputs
setup_adc_ports(ALL_ANALOG); // A/D ports
setup_adc(ADC_CLOCK_DIV_32); // A/D clock
set_adc_channel(0); // Select channel 0 (AN0)
delay_us(10);
rtos_run(); // Start RTOS
}
//*****************************************************************************//
```

7.2 Utilisation d'un sémaphore

Le programme donné le listing 8 fonctionne et montre la tension mesurée sur l'écran de PC. Ce programme peut être amélioré légèrement en utilisant un sémaphore pour synchroniser

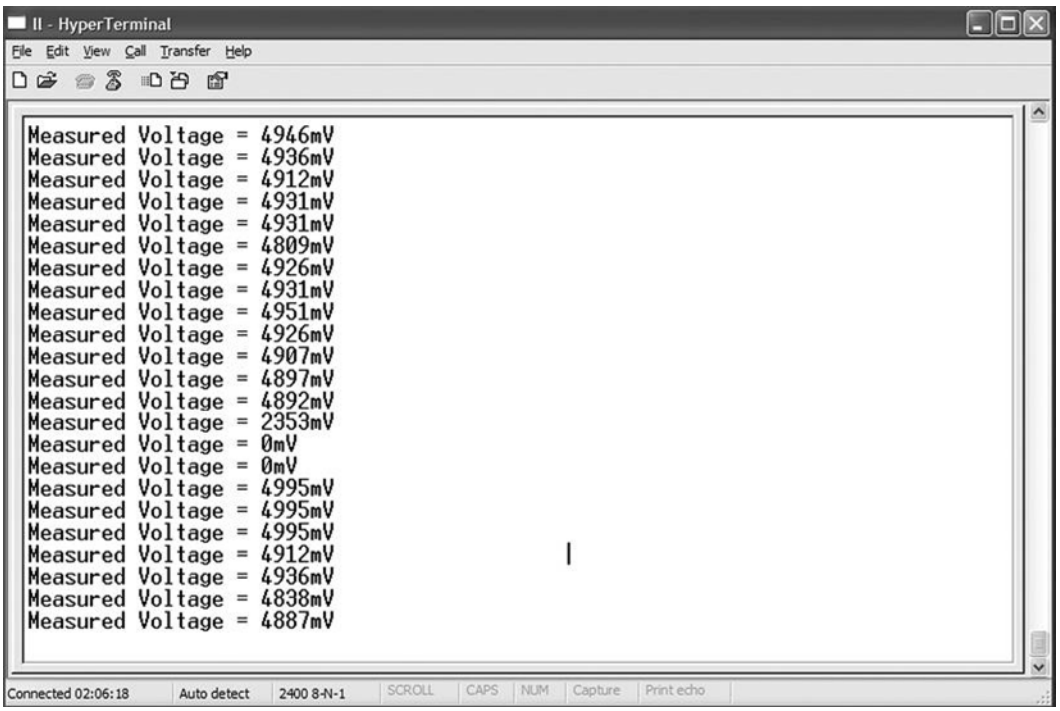

FIG. 7.3 – La sortie du programme sur l'HyperTerminal

l'affichage de la tension mesurée avec les échantillons de l'A/D. le programme est le listing 9.
L'exécution du nouveau programme est comme suit :
– la variable de sémaphore (sem) est placé à 1 au début du programme.
– la tâche Get_voltage décrémente la variable de sémaphore (l'appel de rtos_wait) ainsi la tâche To_RS232 est bloquée (la variable de sémaphore sem=0) et il ne peut pas envoyer les données au PC. Quand un nouvel échantillon de l'A/D est prêt, la variable de sémaphore est incrémenté (l'appel de rtos_signal) et la tâche To_RS232 peut continuer. La Tâche To_RS232 envoie alors la tension mesurée au PC et incrémente la variable de sémaphore pour indiquer qu'elle a eu accès aux données. Et la tâche Get_voltage peut obtenir alors un nouvel échantillon. Ce processus est répété pour toujours.

7.3 Travail demandé

1. .
2. Écrire puis compiler les programmes avec PCW par la suite vérifier son bon fonctionnement à l'aide de la simulation par ISIS.
3. expliquer puis commenter les programme présenté dans les listing.
4. Charger le programme dans le microcontrôleur et réaliser le schema sur une plaque d'essai.
5. Qu'elle est l'apport du semaphore dans le deuxième cas.

```
//*****************************Listing 9*********************//
#include <18F8520.h>
#device adc=10
#use delay (clock=10000000)
#use rs232(baud=2400,xmit=PIN_C6,rcv=PIN_C7)
unsigned int16 adc_value;
unsigned int32 Volts;
int8 sem;
// Define which timer to use and minor_cycle for RTOS
#use rtos(timer=0, minor_cycle=100ms)
// Declare TASK "Live" - called every 200ms
// This task flashes the LED on port RD7
#task(rate=200ms, max=1ms)
void Live()
{
output_toggle(PIN_D7); // Toggle RD7 LED
}
// Declare TASK "Get_voltage" - called every 10ms
#task(rate=2s, max=100ms)
void Get_voltage()
{
rtos_wait(sem); // decrement semaphore
adc_value = read_adc(); // Read A/D value
Volts = (unsigned int32)adc_value*5000;
Volts = Volts / 1024; // Voltage in mV
rtos_signal(sem); // increment semaphore
}
// Declare TASK "To_RS232" - called every millisecond
#task(rate=2s, max=100ms)
void To_RS232()
{
rtos_wait(sem); // Decrement semaphore
printf("Measured Voltage = %LumV\n\r",Volts); // Send to RS232
rtos_signal(sem); // Increment semaphore
}
// Start of MAIN program
void main()
{
set_tris_d(0); // PORTD all outputs
output_d(0); // Clear PORTD
set_tris_a(0xFF); // PORTA all inputs
setup_adc_ports(ALL_ANALOG); // A/D ports
setup_adc(ADC_CLOCK_DIV_32); // A/D clock
set_adc_channel(0); // Select channel 0 (AN0)
delay_us(10);
sem = 1; // Semaphore is 1
rtos_run(); // Start RTOS
}
//*************************************************************//
```