école
normale
supérieure
paris–saclay
UNIVERSITÉ PARIS-SACLAY

TP 5 : Liaisons I2C - Utilisation du MCP23008

— UE 441b : Informatique industrielle —

Andréa Lequin Alexandre D'Hooge

26 novembre 2018

Table des matières

1 Chenillard sur 8 leds

La liaison I2C du LPC804 est gérée à l'aide d'un circuit MCP23008. Le connecteur des broches pour la liaison présente en amont une série de leds (une par broche). On peut donc commencer par tester la connexion I2C en observant sur les leds le message que l'on souhaite envoyer. Le code suivant permet de réaliser un chenillard sur 8 leds avec une transition toutes les 250 ms.

```
/*
===============================================================================
 Name        : TP_i2c
 Description : source de base pour MCP23008 (I2C GPIO)
===============================================================================
*/

// Directives de préprocesseur

#include <cr_section_macros.h>
#include <stdio.h>
#include "LPC8xx.h"
//#include "fro.h"
#include "rom_api.h"
#include "syscon.h"
#include "swm.h"
#include "i2c.h"
#include "ctimer.h"
//#include "core_cm0plus.h"

#include "lib_UE441_lcd.h"


// Définitions des registres du MCP23008

#define MCP23_I2C_AD  0x40  // Adresse du MCP23
#define MCP23_IODIR_REG   0
#define MCP23_IPOL_REG    1
#define MCP23_GPINTEN_REG 2
#define MCP23_DEFVAL_REG  3
#define MCP23_INTCON_REG  4
#define MCP23_IOCON_REG   5
#define MCP23_GPPU_REG    6
#define MCP23_INTF_REG    7
#define MCP23_INTCAP_REG  8
#define MCP23_GPIO_REG    9
#define MCP23_OLAT_REG    10


#define SYSTICK_TIME 15000 // 1 ms pour clk 15 MHz

// Fonction d'écriture dans les registres du MCP23008

void MCP23_write_reg(uint8_t reg_num,uint8_t valeur){
  uint8_t I2CMasterBuffer[3]; // ad, #reg, valeur
  uint8_t I2CWriteLength=2;
  I2CMasterBuffer[0]=MCP23_I2C_AD;
  I2CMasterBuffer[1]=reg_num;
  I2CMasterBuffer[2]=valeur;
  I2CmasterWrite(I2CMasterBuffer, I2CWriteLength );
}
```

```
52
53 // Fonction de lecture des registres du MCP23008
54
55 uint8_t MCP23_read_reg(uint8_t reg_num){
56   uint8_t I2CMasterBuffer[2]; // ad, #reg
57   uint8_t I2CSlaveBuffer[1];
58   uint8_t I2CWriteLength=1;
59   uint8_t I2CReadLength=1;
60   I2CMasterBuffer[0]=MCP23_I2C_AD;
61   I2CMasterBuffer[1]=reg_num;
62   I2CmasterWriteRead( I2CMasterBuffer, I2CSlaveBuffer, I2CWriteLength, I2CReadLength );
63   return I2CSlaveBuffer[0];
64 }
65
66 void MCP23_init_OD(){
67   MCP23_write_reg(MCP23_IOCON_REG,0xFF);  // On configure les inputs outputs
68   MCP23_write_reg(MCP23_GPPU_REG,0xFF);   // On active les pull-ups sur toutes les
       broches
69   MCP23_write_reg(MCP23_IODIR_REG,0x00); // On définit le MPC23 comme le maître (donc
       sorties sur toutes les broches)
70 }
71 void MCP23_write_OD(uint8_t valeur){
72   MCP23_write_reg(MCP23_GPIO_REG,valeur);  // On écrit sur les broches pour changer leur
       état
73 }
74
75 volatile uint32_t millis;
76
77 void init_cpt_millis() {
78   millis=0;
79   // mise en place du compteur de millisecondes cadencé par SysTick
80   SysTick->LOAD = SYSTICK_TIME;  // période = valeur de rechargement
81   // clock = system_clock, tick interrupt enabled, count enabled
82   SysTick->CTRL = (1<<SysTick_CTRL_CLKSOURCE_Pos) |
83                   (1<<SysTick_CTRL_TICKINT_Pos) |
84              (1<<SysTick_CTRL_ENABLE_Pos);
85   // Clear the counter and the countflag bit by writing any value to SysTick_VAL
86   SysTick->VAL = 0;
87   // Enable the SYSTICK interrupt in the NVIC
88   NVIC_EnableIRQ(SysTick_IRQn);
89 }
90
91 void SysTick_Handler(void) {
92   // Clear the interrupt flag by reading the SysTick_CTRL register
93   uint32_t temp = SysTick->CTRL;
94   millis++;
95   return;
96 }
97
98 int main(void) {
99
100   char text[17]; // une ligne + \0
101   uint32_t ech_chenillard=250;
102   uint32_t ech_secondes=0;
103   uint32_t secondes=0;
104   uint8_t touche,c_touche;
105   //Configuration de l'horloge à 15 MHz
106   LPC_PWRD_API->set_fro_frequency(30000);
107
108   // Peripheral reset to the GPIO0 and pin interrupt modules. '0' asserts, '1' deasserts
       reset.
109   LPC_SYSCON->PRESETCTRL0 &=  (GPIO0_RST_N & GPIOINT_RST_N);
110   LPC_SYSCON->PRESETCTRL0 |= ~(GPIO0_RST_N & GPIOINT_RST_N);
111
112   //Mise en fonctionnement des périphériques utilisés
113   LPC_SYSCON->SYSAHBCLKCTRL0 |= (IOCON | GPIO0 | SWM | CTIMER0 | GPIO_INT);
114
115
116   //initialisation de l'afficheur lcd avec un affichage
117   init_lcd();
118     lcd_gohome();
119     lcd_puts("UE441 - M1 - ENS");
120     init_cpt_millis();
121     MCP23_init_OD(); // après init_lcd !!
```

```
122
123     int leds_chenill=0x01;
124     MCP23_write_OD(leds_chenill);
125     while(1) {
126       if (millis>ech_secondes){
127         sprintf(text,"%02d",(secondes++)%100);
128       lcd_position(1,1);
129       lcd_puts(text);
130       ech_secondes+=1000;
131       }
132       if(millis>ech_chenillard){
133         leds_chenill*=2;
134           ech_chenillard+=250;
135           if (leds_chenill==256){
136             leds_chenill=0x01;
137           }
138           MCP23_write_OD(~(leds_chenill));
139       }
140     }
141 }
```

Nous aurions pu réaliser cette fonction avec une machine à états, nous avons préféré utilisé une méthode « mathématique », en déterminant l'opération à réaliser pour envoyer le bon nombre dans la liaison I2C pour allumer les leds successivement.

Il est important d'initialiser l'écran LCD avant le MCP23008 car les commandes d'initialisation de la liaison I2C pour la carte LPC804 sont réalisées dans la fonction `init_lcd()`.

2 Utilisation du clavier matriciel

On désactive les leds à l'aide du jumper.

2.1 Routines d'émulation de sorties « drain ouvert »

Nous avons déjà utilisé les routines demandées pour le code du chenillard.

2.2 Lecture du clavier

On branche sur les broches de la liaison I2C un clavier matriciel. On définit donc des fonctions de lecture du clavier et on complète le `main` pour afficher le caractère enfoncé.

```
1
2  uint8_t Ftouche() {
3    // détection d'un front d'appui sur touche
4    static uint8_t etat=0;
5    uint8_t lecture;
6    MCP23_write_OD(0x0f);
7    lecture=MCP23_read_reg(MCP23_GPIO_REG);
8    MCP23_write_OD(0xf0);
9    lecture|=MCP23_read_reg(MCP23_GPIO_REG);
10
11
12   // Si aucune touche n'est pressée ou si la pression est maintenue, la fonction doit
       renvoyer 0xFF
13
14   if (lecture==0){
15     lecture=0xFF;
16   }
17
18   uint8_t lecture2;
19   MCP23_write_OD(0x0f);
20   lecture2=MCP23_read_reg(MCP23_GPIO_REG);
21   MCP23_write_OD(0xf0);
22   lecture2|=MCP23_read_reg(MCP23_GPIO_REG);
23
24   if (lecture2==lecture){
25     lecture=0xff;
26   }
27
28   return lecture;
29 }
```

```
uint8_t decode_touche(uint8_t code) {
  const uint8_t Tcodes[]={0x77,0x7b,0x7d,0x7e,
              0xb7,0xbb,0xbd,0xbe,
              0xd7,0xdb,0xdd,0xde,
              0xe7,0xeb,0xed,0xee};
  const uint8_t Tascii[]={'1','2','3','A',
              '4','5','6','B',
              '7','8','9','C',
              '*','0','#','D'};
  uint8_t i=0;
  while ((i<16)&&(Tcodes[i]!=code)) {
    i++;
  }
  return (i<16)?Tascii[i]:'?';  // Si on sort du tableau ie on a pas trouvé le caractère,
     on renvoie '?'
}

int main(void) {


    while(1) {
      touche=Ftouche();
    if (touche!=0xff){
      c_touche=decode_touche(touche);
      lcd_position(1,4);
      sprintf(text,"%02x %c",touche,c_touche);
      lcd_puts(text);
    }
    }
}
```

Nous avons pu vérifier le bon fonctionnement du code précédent.

3 Construction d'un mini terminal

On utilise maintenant la liaison série en plus de la liaison I2C. On reprend pour ça une partie du code utilisé au TP précédent et on modifie en plus l'affichage sur l'écran LCD pour que les caractères se succèdent. On a repassé l'horloge à 12 MHz pour pouvoir reprendre la génération du baud rate du TP précédent. Nous avons également modifié le code de lecture de touche pour vérifier qu'une touche reste enfoncée car celui-ci ne semblait pas fonctionner comme attendu.

```

#define WaitForUART0txRdy  while(((LPC_USART0->STAT) & (1<<2)) == 0)
#define WaitForUART0rxRdy  while(((LPC_USART0->STAT) & (1<<0)) == 0)

#define SYSTICK_TIME 12000 // 1 ms pour clk 12 MHz

#define DELAY 500 // on considère que l'appui est maintenu s'il dure plus de 500ms

volatile uint32_t reading_time=0;
volatile uint8_t old_touche;

uint8_t Ftouche() {
  // détection d'un front d'appui sur touche
  uint8_t lecture;
  MCP23_write_OD(0x0f);
  lecture=MCP23_read_reg(MCP23_GPIO_REG);
  MCP23_write_OD(0xf0);
  lecture|=MCP23_read_reg(MCP23_GPIO_REG);
  // Si aucune touche n'est pressée ou si la pression est maintenue, la fonction doit
    renvoyer 0xFF

  if (lecture==0){
    lecture=0xFF;
  }
  if ((lecture==old_touche)&&(millis-reading_time<=DELAY)){
    lecture=0xFF;
  }
  return lecture;
}
int main(void) {
```

```
  //Configuration de l'horloge à 12 MHz
  LPC_PWRD_API->set_fro_frequency(24000);

  // Clock configuration for UART

  LPC_SYSCON->FRG0CLKSEL=FRGCLKSEL_MAIN_CLK;
  LPC_SYSCON->UART0CLKSEL=FCLKSEL_FRG0CLK;

  //Mise en fonctionnement des périphériques utilisés

  LPC_SYSCON->SYSAHBCLKCTRL0 |= (IOCON | GPIO0 | SWM | CTIMER0 | GPIO_INT|UART0);

  // Connect UART0 TXD, RXD signals to port pins

  ConfigSWM(U0_TXD,DBGTXPIN);
  ConfigSWM(U0_RXD,DBGRXPIN);


  // Clear USART peripheral resets

  LPC_SYSCON->PRESETCTRL0 &=  (UART0_RST_N & UART1_RST_N);
  LPC_SYSCON->PRESETCTRL0 |= ~(UART0_RST_N & UART1_RST_N);

  // Baud rate configuration

   LPC_USART0->OSR = 4;
   LPC_USART0->BRG = 20;

  // Configure USART0CFG register

  LPC_USART0->CFG=DATA_LENG_8|PARITY_NONE|STOP_BIT_1;

  // Clear pending flags

  LPC_USART0->STAT=0xFFFF;

  // Enable USART0

  LPC_USART0->CFG|=UART_EN;

    int position=0;

    while(1) {
      if (millis>ech_secondes){
        sprintf(text,"%02d",(secondes++)%100);
      lcd_position(1,1);
      lcd_puts(text);
      lcd_position(0,position);
      ech_secondes+=1000;
      }
    touche=Ftouche();
    reading_time=millis;
    c_touche=decode_touche(touche);
    if(position==16){
      position=0;
      lcd_position(0,0);
      lcd_puts("                ");
      lcd_position(0,0);
    }
    if ((touche!=0xff)&&((old_touche!=touche)||(millis-reading_time>=DELAY))){
      old_touche=touche;
      lcd_position(0,position);
      lcd_putc(c_touche);
      position++;
      WaitForUART0txRdy;
      LPC_USART0->TXDAT=c_touche;
    }
    }
}
```

4 Consultation périodique du clavier par interruption

Pour périodiser la lecture du clavier et ne pas surcharger le `main`, on souhaite lire périodiquement l'état du clavier en réalisant une interruption avec le Multi-Rate Timer (MRT). Le code suivant réalise cette interruption, affiche sur l'écran LCD le caractère lu et l'envoie à l'ordinateur via la liaison série.

```
volatile uint32_t reading_time=0;
volatile uint8_t old_touche=0;

uint8_t Ftouche() {
  // code précédent
}

uint8_t decode_touche(uint8_t code) {
  // code précédent
}

// Définition des variables globales nécessaires

volatile uint8_t touche;
volatile uint8_t c_touche;
volatile int flag=0;

int main(void) {
  // Clear MRT resets

  LPC_SYSCON->PRESETCTRL0 &=  (MRT_RST_N);
  LPC_SYSCON->PRESETCTRL0 |= ~(MRT_RST_N);


  // Enable MRT interrupt

  NVIC->ISER[0]|=1<<10;

  // Configure MRT interrupt

  LPC_MRT->Channel[0].INTVAL=360000;  // interrupt every 30 ms
  LPC_MRT->Channel[0].CTRL=0b001; // Enable interrupt on repeat interrupt mode

    while(1) {
      if (flag==1){
        old_touche=touche;
        lcd_position(0,position);
        lcd_putc(c_touche);
        position++;
        WaitForUART0txRdy;
        LPC_USART0->TXDAT=c_touche;
        flag=0;
    }
    }
}

// Fonction d'interruption du MRT

void MRT_IRQHandler(void){
  touche=Ftouche();
  if ((touche!=0xFF)&&(flag==0)){
    c_touche=decode_touche(touche);
    reading_time=millis;      // Le bouton sera souvent enfoncé plus de 30 ms
    flag=1;
  }
  LPC_MRT->Channel[0].STAT=1<<0;  // Clear the interrupt flag
  return;
}
```