Université Djilali BOUNAAMA, Khemis Miliana
مليانة خميس بونعامة جيلالي جامعة
Faculté des Sciences et de la Technologie
والتكنولوجيا العلوم كلية

# Série d'exercices 6 : Interfaces du Microcontrôleur

**Exercice 1 : Lecture et écriture en EEPROM interne**

Un microcontrôleur PIC16F877A possède 256 octets d'EEPROM interne. Écrire une fonction void ecrire_EEPROM(unsigned char adresse, unsigned char donnee) et une fonction unsigned char lire_EEPROM(unsigned char adresse) en respectant la procédure de configuration des registres EECON1, EECON2, EEADR et EEDATA.

**Exercice 2 : Configuration USART pour liaison PC**

On veut configurer l'USART du PIC en mode asynchrone, 9600 bauds, 8 bits de données, 1 stop bit, sans parité. Fréquence quartz = 4 MHz. Calculer la valeur à charger dans SPBRG, puis écrire la routine d'initialisation et une fonction d'envoi d'un caractère.

**Exercice 3 : Réception USART avec interruption**

À partir de la configuration précédente, écrire le code d'interruption de réception USART qui stocke le caractère reçu dans un buffer circulaire de 16 cases. Indiquer les initialisations nécessaires.

**Exercice 4 : Communication SPI – Maître vers esclave**

On configure le PIC en maître SPI pour envoyer un octet vers un périphérique esclave. Fréquence d'horloge SPI = Fosc/4. Écrire l'initialisation (broches, mode, registre SSPCON, SSPSTAT) et la fonction void spi_ecrire(unsigned char data).

**Exercice 5 : Communication I2C – Écriture dans une EEPROM externe 24LC02**

**Énoncé :**
Le PIC est maître I2C. Écrire une fonction void i2c_ecrire_eeprom(unsigned char adresse_esclave, unsigned char mot_memoire, unsigned char donnee) qui écrit un octet dans l'EEPROM 24LC02 (adresse esclave 0xA0). Utiliser les registres SSPCON2, SSPSTAT, SSPBUF.

**Exercice 6 : Application mixte – Sauvegarde USART vers EEPROM**

On reçoit en USART des chaînes de caractères se terminant par \r. Chaque caractère reçu est sauvegardé en EEPROM à partir de l'adresse 0. La réception utilise l'interruption. Écrire le code complet de l'interruption qui gère la réception et l'écriture en EEPROM (sans attendre le temps d'écriture à chaque fois).

**Solution Série 5**

**Exercice 1 :**

```
void ecrire_EEPROM(unsigned char adresse, unsigned char donnee) {
    while (WR == 1);          // Attente fin écriture précédente
    EEADR = adresse;          // Adresse cible
```

```
    EEDATA = donnee;        // Donnée à écrire
    EEPGD = 0;              // Accès EEPROM (pas mémoire programme)
    WREN = 1;               // Autorise l'écriture
    GIE = 0;                // Interdiction des interruptions
    EECON2 = 0x55;
    EECON2 = 0xAA;
    WR = 1;                 // Lance l'écriture
    GIE = 1;                // Réautorise interruptions
    while (WR == 1);        // Attente fin
    WREN = 0;               // Désactive écriture
}

unsigned char lire_EEPROM(unsigned char adresse) {
    while (WR == 1);
    EEADR = adresse;
    EEPGD = 0;
    RD = 1;                 // Lance la lecture
    return EEDATA;
}
```

**Exercice 2 :**

Formule pour asynchrone haute vitesse (BRGH = 1) :

$BAUD=FOSC/(16\times(SPBRG+1))$
$9600=4\,000\,000/(16\times(SPBRG+1)) \Longrightarrow$
$SPBRG+1=4\,000\,000/(16\times9600\ ) \approx 26{,}04 \Longrightarrow\ SPBRG=25$

```
void init_USART(void) {
    TRISC6 = 0;  // TX = sortie
    TRISC7 = 1;  // RX = entrée
    SPBRG = 25;
    TXSTA = 0x24;  // BRGH=1, TXEN=1, asynchrone
    RCSTA = 0x90;  // SPEN=1, CREN=1
}
```

```
void envoyer_char(unsigned char c) {
   while (TXIF == 0);
   TXREG = c;
}
```

**Exercice 3 : Réception USART avec interruption**

```
void init_USART(void) {
   TRISC6 = 0;  // TX = sortie
   TRISC7 = 1;  // RX = entrée
   SPBRG = 25;
   TXSTA = 0x24;  // BRGH=1, TXEN=1, asynchrone
   RCSTA = 0x90;  // SPEN=1, CREN=1
}

void envoyer_char(unsigned char c) {
   while (TXIF == 0);
   TXREG = c;
}
```

**Exercice 3 :**

```
#define TAILLE 16
volatile unsigned char buffer[TAILLE];
volatile unsigned char tete = 0, queue = 0;

void init_interruption_USART(void) {
   PIE1bits.RCIE = 1;   // Interruption réception
   INTCONbits.PEIE = 1; // Interruptions périphériques
```

```
    INTCONbits.GIE = 1;
}

void __interrupt() isr(void) {
    if (PIR1bits.RCIF) {
        unsigned char c = RCREG;
        unsigned char prochaine = (tete + 1) % TAILLE;
        if (prochaine != queue) { // buffer non plein
            buffer[tete] = c;
            tete = prochaine;
        }
    }
}
```

**Exercice 4 :**

**Code :**

```
void init_SPI_maitre(void) {
    TRISC3 = 0;   // SCK sortie
    TRISC5 = 0;   // SDO sortie
    TRISC4 = 1;   // SDI entrée
    TRISA0 = 0;   // SS (broche esclave) gérée manuellement

    SSPSTAT = 0x00;      // Entrée échantillonnée milieu, bord montant
    SSPCON = 0x20;       // SPI maître, Fosc/4, mode 0,0
}

void spi_ecrire(unsigned char data) {
    SSPBUF = data;
    while (SSPSTATbits.BF == 0); // Attente fin envoi
}
```

**Exercice 5 :**

**Code :**

```
void i2c_attendre(void) {
   while ((SSPCON2 & 0x1F) || (SSPSTATbits.R_W));
}

void i2c_start(void) {
   i2c_attendre();
   SEN = 1;
   i2c_attendre();
}

void i2c_stop(void) {
   PEN = 1;
   i2c_attendre();
}

void i2c_ecrire_octet(unsigned char octet) {
   SSPBUF = octet;
   i2c_attendre();
}

void i2c_ecrire_eeprom(unsigned char adresse_esclave, unsigned char mot_memoire, unsigned char
donnee) {
   i2c_start();
   i2c_ecrire_octet(adresse_esclave);   // 0xA0 + écriture
   i2c_ecrire_octet(mot_memoire);       // adresse interne EEPROM
   i2c_ecrire_octet(donnee);            // donnée
   i2c_stop();
   __delay_ms(10);                      // temps d'écriture EEPROM
}
```

**Exercice 6 :**

```
unsigned char adr_eeprom = 0;

void __interrupt() isr(void) {
   if (PIR1bits.RCIF) {
      unsigned char c = RCREG;
      if (c == '\r') {
         // Fin de ligne, on pourrait signaler
         return;
      }
      if (adr_eeprom < 256) {
         // Écriture EEPROM sans attendre (à vérifier WR avant chaque écriture)
```

```
            while (WR == 1);
            EEADR = adr_eeprom++;
            EEDATA = c;
            EEPGD = 0;
            WREN = 1;
            GIE = 0;
            EECON2 = 0x55;
            EECON2 = 0xAA;
            WR = 1;
            GIE = 1;
            WREN = 0;
        }
    }
}
```

***Remarque :*** En pratique, il faut un buffer pour ne pas bloquer l'interruption, car l'écriture EEPROM prend ~4 ms. La solution ci-dessus simplifie mais bloque l'interruption pendant l'écriture. Une meilleure approche utiliserait un buffer FIFO et une tâche principale écrivant en EEPROM.