Protocoles de Communication dans le Véhicule

TP3: Tableau de bord d'un véhicule a base de communication I2C

Réalisé par:

- **ELMADI Choaib**
- **ELHAZMIRI Ayoub**

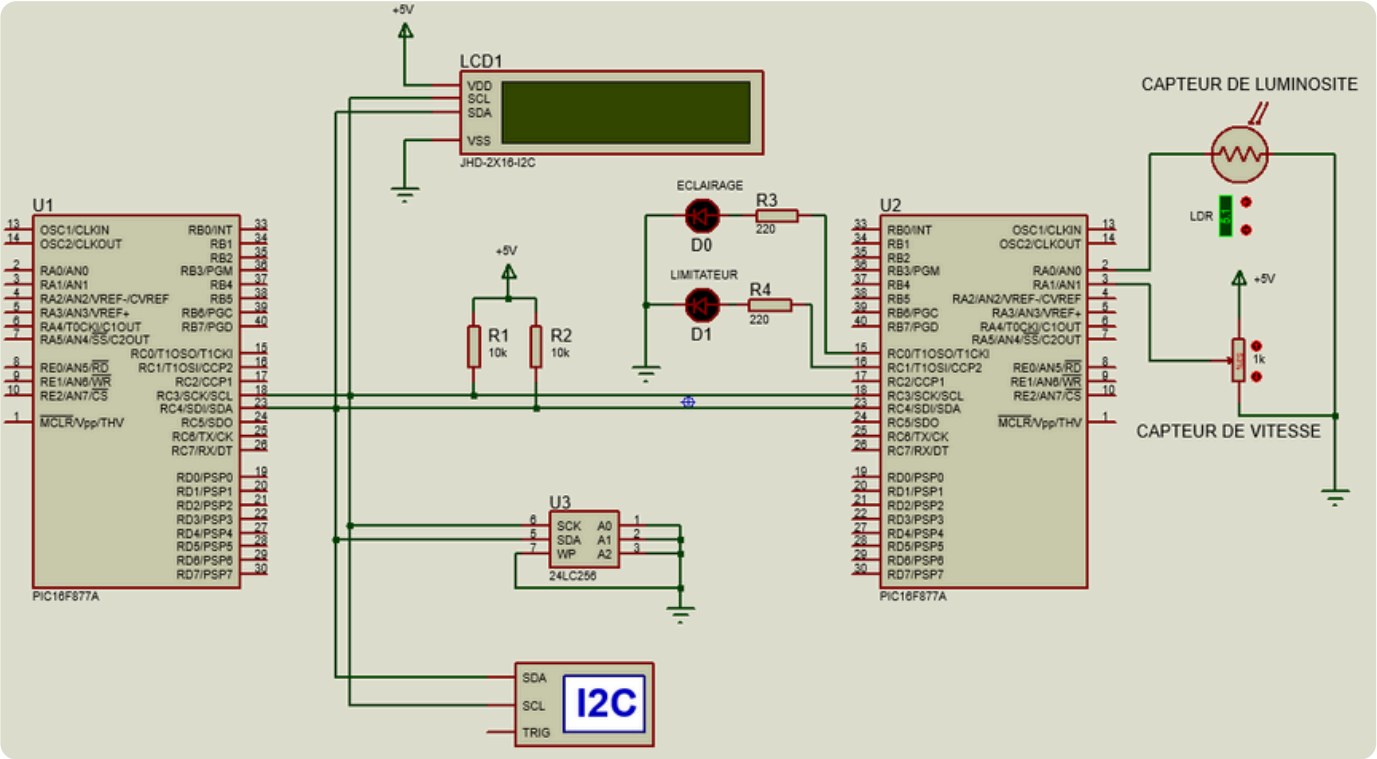
Encadré par:

- **M. Anas HATIM**

Introduction:

Dans ce troisième TP des protocoles de communication embarqués, l'objectif est de mettre en œuvre une communication I2C entre deux microcontrôleurs PIC16F877A configurés en maître et esclave. Le maître est également connecté à une mémoire EEPROM 24LC256 et à un écran LCD via le bus I2C.
Le travail consiste à programmer les échanges I2C entre les deux PIC, à envoyer des commandes, recevoir des données de capteurs simulés, contrôler des actionneurs, enregistrer les données dans l'EEPROM, et afficher les résultats sur le LCD.

Proteus setup:



Le programme principal de l'esclave:

```c
#include <xc.h>
#include <stdint.h>

#define _XTAL_FREQ 20000000

uint8_t Vitesse = 45;
uint8_t Lum = 78;

void I2C_Slave_Init(uint8_t address) {
    SSPSTAT = 0x00;
    SSPCON = 0x36;
    SSPADD = address;
    TRISC3 = 1;
    TRISC4 = 1;
    PIR1bits.SSPIF = 0;
    PIE1bits.SSPIE = 1;
    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;
}

void main() {
    TRISC0 = 0;
    TRISC1 = 0;
    PORTCbits.RC0 = 0;
    PORTCbits.RC1 = 0;
    I2C_Slave_Init(0x0F);
    while (1) {}
}

```

Interrupt Service Routine de l’esclave:

```
void __interrupt() ISR() {
    uint8_t rec;
    if (PIR1bits.SSPIF == 1) {
        if (!SSPSTATbits.D_nA && !SSPSTATbits.R_nW) {
            rec = SSPBUF;
            while (!BF);
            rec = SSPBUF;
            PIR1bits.SSPIF = 0;

            switch (rec) {
                case 0xA0:
                    break;
                case 0xA1:
                    break;
                case 0xA2:
                    PORTCbits.RC0 = 1;
                    break;
                case 0xA3:
                    PORTCbits.RC0 = 0;
                    break;
                case 0xA4:
                    PORTCbits.RC1 = 1;
                    break;
                case 0xA5:
                    PORTCbits.RC1 = 0;
                    break;
            }
        }

        if (!SSPSTATbits.D_nA && SSPSTATbits.R_nW) {
            rec = SSPBUF;
            while (!BF);
            SSPCONbits.CKP = 0;

            if (rec == 0xA0)
                SSPBUF = Lum;
            else if (rec == 0xA1)
                SSPBUF = Vitesse;

            SSPCONbits.CKP = 1;
            while (SSPSTATbits.BF);
        }
        PIR1bits.SSPIF = 0;
    }
}
```

Les définitions des fonctions maître:

```
void I2C_Master_Init(int clock_speed) {
    SSPSTAT = 0×00;
    SSPCON = 0×28;
    SSPADD = (_XTAL_FREQ / (4 * clock_speed)) - 1;
    TRISC3 = 1;
    TRISC4 = 1;
}

void Start_Bit() {
    SSPCON2bits.SEN = 1;
    while (SSPCON2bits.SEN);
    PIR1bits.SSPIF = 0;
}

void Repeated_Start() {
    SSPCON2bits.RSEN = 1;
    while (SSPCON2bits.RSEN);
    PIR1bits.SSPIF = 0;
}

void Send_Byte_Data(uint8_t data) {
    SSPBUF = data;
    while (!PIR1bits.SSPIF);
    PIR1bits.SSPIF = 0;
    if (SSPCON2bits.ACKSTAT) Stop_Bit();
}

uint8_t Receive_Byte_Data() {
    SSPCON2bits.RCEN = 1;
    while (!SSPSTATbits.BF);
    return SSPBUF;
}

void Send_ACK_Bit() {
    SSPCON2bits.ACKDT = 0;
    SSPCON2bits.ACKEN = 1;
    while (SSPCON2bits.ACKEN);
}

void Send_NACK_Bit() {
    SSPCON2bits.ACKDT = 1;
    SSPCON2bits.ACKEN = 1;
    while (SSPCON2bits.ACKEN);
}

void Stop_Bit() {
    SSPCON2bits.PEN = 1;
    while (SSPCON2bits.PEN);
    PIR1bits.SSPIF = 0;
}

void EEPROM_Write(uint8_t addr, uint8_t data) {
    Start_Bit();
    Send_Byte_Data(0×A0);
    Send_Byte_Data(addr);
    Send_Byte_Data(data);
    Stop_Bit();
    __delay_ms(5);
}

void LCD_Send_String(const char* str) {
    while (*str) {
        Start_Bit();
        Send_Byte_Data(0×4E);
        Send_Byte_Data(*str);
        Stop_Bit();
        str++;
    }
}

```

Le programme principal du maître:

```c
#include <xc.h>
#include <stdint.h>
#include "utils.c"

void main(void) {
    uint8_t lum, vitesse;

    I2C_Master_Init(CLOCK_SPEED);

    while (1) {
        Start_Bit();
        Send_Byte_Data((SLAVE_ADDRESS << 1) | 0);
        Send_Byte_Data(0xA0);
        Stop_Bit();

        __delay_ms(10);

        Start_Bit();
        Send_Byte_Data((SLAVE_ADDRESS << 1) | 1);
        lum = Receive_Byte_Data();
        Send_NACK_Bit();
        Stop_Bit();

        Start_Bit();
        Send_Byte_Data((SLAVE_ADDRESS << 1) | 0);
        Send_Byte_Data(0xA1);
        Stop_Bit();

        __delay_ms(10);

        Start_Bit();
        Send_Byte_Data((SLAVE_ADDRESS << 1) | 1);
        vitesse = Receive_Byte_Data();
        Send_NACK_Bit();
        Stop_Bit();

        EEPROM_Write(0x10, lum);
        EEPROM_Write(0x11, vitesse);

        LCD_Send_String("L=");
        LCD_Send_String((char[]){lum / 10 + '0', lum % 10 + '0', ' ', '\0'});
        LCD_Send_String("V=");
        LCD_Send_String((char[]){vitesse / 10 + '0', vitesse % 10 + '0', '\0'});

        __delay_ms(1000);
    }
}
```

Les prototypes des fonctions maître:

```c
#define _XTAL_FREQ 20000000
#define CLOCK_SPEED 400000
#define SLAVE_ADDRESS 0x0F

void I2C_Master_Init(int);
void Start_Bit(void);
void Repeated_Start(void);
void Send_Byte_Data(uint8_t);
uint8_t Receive_Byte_Data(void);
void Send_ACK_Bit(void);
void Send_NACK_Bit(void);
void Stop_Bit(void);

void EEPROM_Write(uint8_t addr, uint8_t data);
void LCD_Send_String(const char* str);
```

Conclusion:

En conclusion, la communication I2C entre les deux microcontrôleurs PIC16F877A, ainsi qu'avec les périphériques EEPROM 24LC256 et LCD, a été correctement mise en œuvre. Les échanges de données entre maître et esclave se sont déroulés conformément aux commandes prévues, et les valeurs simulées de vitesse et de luminosité ont été enregistrées et affichées avec succès.