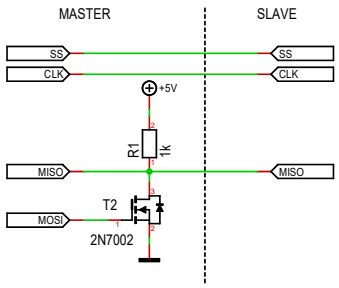
Mise en œuvre des codeurs SPI

1.Matériel

Le bus SPI standard nécessite 4 fils : Une horloge (CLK), une ligne de données sortante du maître (MOSI ou SDO), une ligne de données entrante dans le maître (MISO ou SDI) et un signal de sélection du périphérique en cours de communication (SS ou CS)

Dans le cas particulier des codeurs SPI, les signaux MOSI et MISO sont connectés entre eux dans le codeur et ne se raccordent pas directement à un bus standard. Une électronique spéciale (collecteur ouvert) est nécessaire, telle que :



T2 formant un inverseur, il faut inverser les bits transmis

La polarité du SPI est :

CPHA=1 : Le codeur lit la donnée sur le front montant de CLK, le master lit la donnée sur le front descendant.

CPOL=0 : CLK = 0 au repos.

Le codeur est sélectionné si SS=0

Période mini de CLK : 7µs

Durée mini entre bytes : 38µs

Durée mini entre le dernier CLK et la remontée de SS : 7µs

Durée mini entre le StartByte et Byte0 : 50µs

Durée mini de SS=1 provoquant une resynchronisation : 1500µs

Durée mini entre SS=0 et le premier front montant d'horloge : 7 µs

Durée mini d'initialisation du codeur après une mise sous tension : 20ms

Le codeur calcule l'angle courant en permanence, ce qui prend 1,5ms mini. L'angle lu par SPI est le dernier angle calculé.

2.Communication

Un trame de données est constituée de 10 bytes.

Le master initie la transmission par 0xAA 0xFF (StartByte)

Le codeur répond avec 2 bytes 0xFF puis B0 et B1 (B0=MSB) contenant la valeur de l'angle, puis 2 autres bytes B2 et B3 qui sont B0 et B1 complémentés, afin de faire une vérification d'intégrité des données.

Le codeur termine la transmission par 4 bytes 0xFF

La transmission SPI se faisant sur les fronts montants et descendants de l'horloge, le master a donc écrit 0xAA suivi de 9 bytes 0xFF et a simultanément lu 2 bytes 0xFF, B0 B1 B2 B3 et 4 bytes 0xFF

[B0, B1] contient l'angle sur 14 bits, cadré sur 16 bits côté MSB.

Les 2 bits LSB de B1 distinguent une donnée d'angle [b1,b0]=[0,1] d'un message d'erreur [b1,b0] = [1,0] interprété comme suit :

Most Significant Byte								Least Significant Byte							
MSB							LSB	MSB							LSB
A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	0	1

DATA16: Error

Most Significant Byte								Least Significant Byte							
MSB							LSB	MSB							LSB
E15	E14	E13	E12	E11	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0

BIT	NAME	Description
E0	0	
E1	1	
E2	F_ADCMONITOR	ADC Failure
E3	F_ADCSATURA	ADC Saturation (Electrical failure or field too strong)
E4	F_RGTOOLOW	Analog Gain Below Trimmed Threshold (Likely reason: field too weak)
E5	F_MAGTOOLOW	Magnetic Field Too Weak
E6	F_MAGTOOHIGH	Magnetic Field Too Strong
E7	F_RGTOOHIGH	Analog Gain Above Trimmed Threshold (Likely reason: field too strong)
E8	F_FGCLAMP	Never occurring in serial protocol
E9	F_ROCLAMP	Analog Chain Rough Offset Compensation: Clipping
E10	F_MT7V	Device Supply VDD Greater than 7V
E11	-	
E12	-	
E13	-	
E14	F_DACMONITOR	Never occurring in serial protocol
E15	-	

3.Exemple de code

Adapté à un PIC18F2431, compilateur XC8

Ce code n'utilise pas le module SPI du PIC mais une émulation du SPI par 1 seul port utilisé alternativement comme MOSI en sortie collecteur ouvert avec pull-up externe, puis en MISO, ce qui permet de s'affranchir du montage spécial avec un MOSFET.

```
#define USE_OR_MASKS
#include <xc.h>
#include <plib\timers.h>
#define _XTAL_FREQ 40000000UL

//    RA4     RA3     RA2
//    Data    sclk    ss
//    pull-up

#define mosi0 {LATAbits.LATA4=0; TRISAbits.RA4 =0;}
#define mosi1 TRISAbits.RA4 =1;

#define miso PORTAbits.RA4
#define sclk LATAbits.LATA3
#define ss LATAbits.LATA2

// Les temps et délais dépendent fortement du
câblage
#define t1_2 9
#define t2 120
#define t4 24
#define t5_6 250
#define t6 24
```

```
#define t7 150

typedef union {
    struct {
        unsigned char low, high;
    };
    unsigned int val_int;
} value_int;

struct _coder {
    value_int data;
    value_int invData;
};


struct _coder coder;

void coderInit(void) {
    ANSEL0bits.ANS2 = 0;
    ANSEL0bits.ANS3 = 0;
    ANSEL0bits.ANS4 = 0;

    TRISAbits.RA2 = 0;
    TRISAbits.RA3 = 0;
    TRISAbits.RA4 = 1;

    ss = 1;
    sclk = 0;
    mosi1

}

void writeByteOnSPI(char v) {
    for (char i = 0; i < 8; i++) {

        if ((v & 0x80) == 0)
            mosi0
        else
            mosi1
            sclk = 1;
        __delay_us(t1_2);
        v = v << 1;
        sclk = 0;
        __delay_us(t1_2);
    }
}

char readByteOnSPI(void) {
    char c = 0;
    mosi1
    __delay_us(t2);
    for (char i = 0; i < 8; i++) {
        sclk = 1;
        __delay_us(t1_2);
        sclk = 0;
        c = c << 1;
        c = c + miso;
        __delay_us(t1_2);
    }
    return c;
}

struct _coder readcoder(void) {
    struct _coder u;
    u.data.val_int = 0;
    u.invData.val_int = 0;

    sclk = 0;
    ss = 0;
    __delay_us(t6);

    writeByteOnSPI(0xAA);
    writeByteOnSPI(0xFF);
    __delay_us(t7);
    u.data.high = readByteOnSPI();
    u.data.low = readByteOnSPI();
    u.invData.high = readByteOnSPI();
    u.invData.low = readByteOnSPI();
    writeByteOnSPI(0xFF);
    writeByteOnSPI(0xFF);
    writeByteOnSPI(0xFF);
    writeByteOnSPI(0xFF);
    __delay_us(t4);
    ss = 1;
    return u;
}

void main(void) {
    coderInit();
    struct _coder u;

    while (1) {
        u = readcoder();
        __delay_us(t5_6);
        __delay_us(t5_6);
        __delay_us(t5_6);
        __delay_us(t5_6);
        __delay_us(t5_6);
        __delay_us(t5_6);
        __delay_us(t5_6);

        if (u.data.val_int != ~(u.invData.val_int))
{
               // c'est une erreur
            Nop();
        }
    }
    while (1);
    return;
}
```