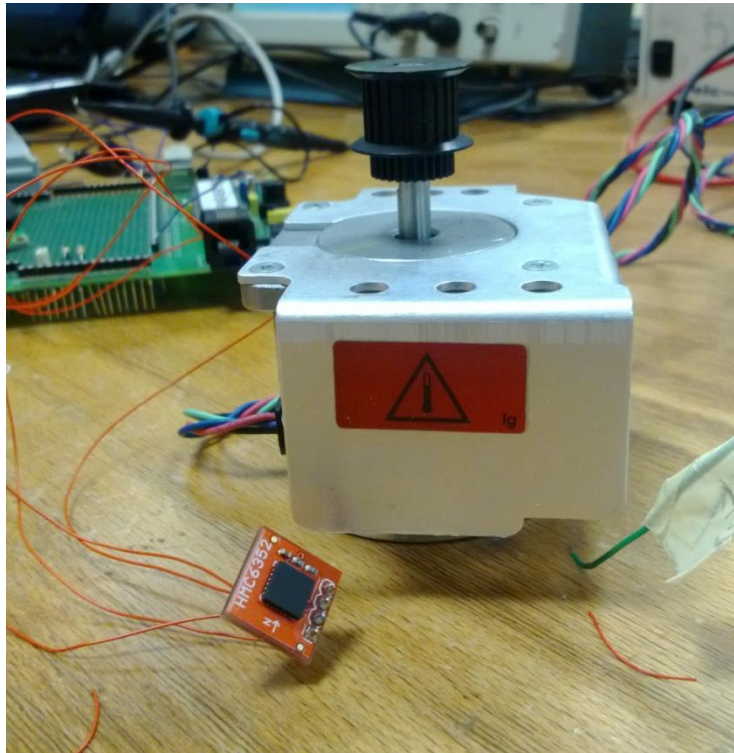


ANNEXE



25/06/2013

Projet de Synthèse

Il s'agit ici d'orienter l'axe du moteur pas à pas dans une direction voulue par rapport au Nord magnétique.



BACHIR MEGHNINE
LUDOVIC REYNOUARD

NICOLAS MENDES
YOANN DUTEL

SOMMAIRE

SOMMAIRE.....	1
1. PROGRAMATION.....	2
1.1. Fichier main.c.....	2
1.2. Fichier setup.c.....	3
1.3. Fichier boussole.c.....	4
1.4. Fichier lcd_4bits.c.....	7
1.5. Fichier can.c.....	8
1.6. Fichier commande_hacheur.c.....	9
1.7. Fichier setup.h.....	11
1.8. Fichier can.h.....	11
1.9. Fichier boussole.h.....	11
1.10. Fichier lcd_4bits.h.....	11
1.11. Fichier commande_hacheur.h.....	11
2. DATASHEETS.....	12
2.1. Registre du timer 0.....	12
2.2. Registre timer 2.....	12
2.3. Registre Convertisseur analogique numérique(CAN).....	13
2.3.1. Registre ADCON0.....	13
2.3.2. Registre ADCON1.....	13
2.3.3. Registre ADCON2.....	14
3. SCHEMA PROTEUS.....	15
3.1. Microcontrôleur.....	15
3.2. Alimentation.....	15
3.3. Input.....	16
3.4. Output.....	16
3.5. Hacheurs.....	16
3.6. Isolation galvanique / Amplification.....	17

1. PROGRAMATION

1.1. Fichier main.c

```
#include <p18cxxx.h>
#include "bootloader.h"
#include "setup.h"
#include "lcd_4bits.h"
#include "commande_hacheur.h"
#include "boussole.h"
#include "can.h"

/*****PROTOTYPES*****/
void init(char mode);

/*****VARIABLES_GLOBALES*****/
unsigned char char_boussole[] = {"-DIREC BOUSSOLE-"};
unsigned char char_choix[] = {"DIRECTION VOULUE"};
unsigned char char_calibrage[] = {" CALIBRAGE EN COURS "};
unsigned long angle_boussole;
unsigned long angle_pot;

/*****CODE_BOOTLOADER*****/
#pragma udata
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode(){}
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode(){}
/*-----*/
void main(void)
{
    init(0);          // 1:calibrage / 0:pas de calibrage
    while(1)
    {
        angle_boussole = retour_angle_boussole();
        angle_pot = retour_angle_can();
        tourner_avec_pot(angle_boussole,angle_pot);
    }
}
/*-----*/
```

```
void init(char mode)
{
    init_port();
    init_boussole();
    init_lcd();
    init_can();
    init_moteur();

    if(mode)
    {
        calibrage();
        lcd_str(char_calibrage);
    }

    lcd_str(char_boussole);
    lcd_str(char_choix);

    LED0 = 1;
}
```

1.2. Fichier setup.c

```

/* Declaration des entree/sortie, des timers, ... */
#include <p18cxxx.h>
#include "lcd_4bits.h"
#include "setup.h"
/*-----*/
void tempo_ms(unsigned short duree)    // ne pas dépasser 348ms
{
    unsigned short buffer;

    duree = 0xFFFF - 188*duree;
    buffer = duree;
    buffer >>= 8;
    TOCON = 0b00010101;    // configuration du timer0 : 16bits, pré-
    // diviseur à 64 (en mettant T0cs à 0, on a une prévision de la fréquence par 4 puis
    // on divise ensuite par 64 la fréquence, on a alors une division par 256)
    TMROH = buffer;    // initialisation du registre de
    // comptage, partie haute de la durée dans le registre partie basse
    TMROL = duree;    // on met la partie haute de la durée dans le
    // registre partie basse
    TOCONbits.TMR0ON = 1;    // on démarre le timer avec le bit qui le fait
    INTCONbits.TMR0IF = 0;    // initialisation du drapeau lié au timer0
    while (INTCONbits.TMR0IF == 0 );    // attente de la levée du drapeau
    TOCONbits.TMR0ON = 0;    // on stop le timer avec le bit qui le fait
}
/*-----*/
void tempo_us(unsigned char duree)    // ne pas dépasser 50 us
{
    duree = 0xFF - 5*duree;
    TOCON = 0b01010000;    // configuration du timer0 : 8bits, pré-
    // diviseur à 2 plus celui initial à 4 on divise au final par 8
    TMROL = duree;
    TOCONbits.TMR0ON = 1;
    INTCONbits.TMR0IF = 0;    // initialisation du drapeau lié au timer0
    while (INTCONbits.TMR0IF == 0 );    // attente de la levée du drapeau
    TOCONbits.TMR0ON = 0;
}
/*-----*/

```

```

void init_port(void)
{
    TRISD = 0;    // LCD en sortie
    LATD = 0;    // LCD à 0
    TRISB = 0;    // Hacheur en sortie
    LATB = 0;    // Transistor à 0
    TRISCbits.TRISC0 = 0;    // SCL en sortie
    TRISCbits.TRISC1 = 0;    // SDA en sortie
    TRISE = 0;    // LEDS en sortie
    LATE = 0;    // LEDS à 0
    TRISAbits.RA0 = 1;    // Potentiometre en entree du CAN
}
/*-----*/
void afficher_donnees(unsigned char position_curseur, unsigned long angle)
{
    unsigned char st[] = {" deg NORD"};
    short int buffer;

    lcd_com(position_curseur);

    buffer = angle/100;    // exemple 320° : on recupere le 3
    st[0] = buffer+'0';    // on le transforme en '3'

    angle = angle - buffer*100;    // on soustrait 300
    buffer = angle/10;    // on recupere le 2
    st[1] = buffer+'0';    // on le transforme en '2'

    angle = angle - buffer*10;    // on soustrait 20 et on recupere le 0
    st[2] = angle+'0';    // on transforme en '0'

    lcd_str(st);
}

```

1.3. Fichier boussole.c

```
#include <p18cxxx.h>
#include "setup.h"
#include "boussole.h"
#include "lcd_4bits.h"
/*-----*/
/******VARIABLES_GLOBALES******/
static unsigned char ak=1,i=0;
/*-----*/
//on utilisera le timer 2 pour réaliser la temporisation
//on veut une temporisation de 50us,sachant que
//la fréquence du microcontrôleur est de 48MHZ,on aura donc N=250, 8 bits sont donc
suffisant donc pas de diviseur
void tpo_i2c(void)
{
    T2CON = 0b00000000;    //on arrête le timer
    PIR1bits.TMR2IF=0;    //on remet à 0 le flag du timer
    PR2 = 250;
    TMR2 = 0;
    T2CONbits.TMR2ON = 1;    //on démarre le timer
    while(PIR1bits.TMR2IF == 0);    //on attend le débordement
    T2CONbits.TMR2ON = 0;    //on remet tout à 0
}
/*-----*/
void start(void)    //programme réalisant le start
{
    SDA=1;
    SCL=1;
    tpo_i2c();
    SDA=0;
    tpo_i2c();
    SCL=0;
    tpo_i2c();
}
/*-----*/
void stop(void)    //programme réalisant le stop
{
    SDA=0;
    SCL=0;
    tpo_i2c();
}
```

```
SCL=1;
tpo_i2c();
SDA=1;
tpo_i2c();
}
/*-----*/
//envoi d'un mot de 8 bit sur le bus i2c
unsigned char send_i2c(unsigned char mot){
    unsigned char i,mask,a;
    mask=0x80;    //masque servant à envoyer le bit de poids
    fort en 1er,d'ou le 0x80(=0b10000000)

    for(i=0;i<8;i++)
    {
        if((mot&mask)==0)    //on change la valeur de chaque bit
            SDA=0;    //envoi bit i du mot envoyé
        else
            SDA=1;
        tpo_i2c();
        SCL=1;    //generation impulsion horloge
        tpo_i2c();
        SCL=0;
        mask>>=1;    //decalage masquage bit (MSB->LSB)
    }
    tpo_i2c();
    SDA_IO = 1;    //SDA en entrée
    SCL=1;    //generation 9eme impulsion horloge
    if(SDA_return == 0)    //test ACK envoyé par esclave
        a=1;
    else
        a=0;
    tpo_i2c();
    SDA_IO = 0;    //SDA en sortie cf doc et p18f4550.h
    SCL=0;
    SDA=0;
    tpo_i2c();
    return(a);
}
/*-----*/
```

```

unsigned char reception_i2c(char ack)
{
    unsigned char i,a=0x00;

    SDA_IO = 1;

    for(i=0;i<8;i++)
    {
        tpo_i2c();          //on change la valeur de chaque bit
        SCL = 1;            //generation impulsion horloge
        tpo_i2c();
        a = a | SDA_return;
        a <<= 1;            //on decale a car on recoit du MSB vers le LSB
        SCL=0;
    }
    tpo_i2c();
    SDA_IO = 0;             //SDA en sortie
    if (ack == 1)
        SDA = 0;            //test ACK envoyé par esclave
    else
        SDA = 1;
    SCL=1;                  //generation 9eme impulsion horloge
    tpo_i2c();
    SCL=0;
    SDA=0;
    tpo_i2c();
    return(a);
}
/*-----*/
void calibrage(void)
{
    unsigned char i;
begin:
    start();
    ak = send_i2c(0x42);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak = send_i2c('C');

```

```

    if(ak==0)
    {
        stop();
        goto begin;
    }
    stop();
    for(i=0;i<150;i++)
        tempo_ms(300);

    start();
    ak = send_i2c(0x42);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak = send_i2c('E');
    if(ak==0)
    {
        stop();
        goto begin;
    }
    stop();

    tempo_ms(15);
}
/*-----*/
void init_boussole(void)
{
begin:
    start();
    ak = send_i2c(0x42);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak = send_i2c('W');
    if(ak==0)
    {

```

```

        stop();
        goto begin;
    }
stop();

tempo_us(50);
tempo_us(50);
tempo_us(50);

start();
    ak=send_i2c(0x42);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c('G');
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c(0x74);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c(0x72); // Rate:20hz, Set/Reset=on, Continuous Mode
    if(ak==0)
    {
        stop();
        goto begin;
    }
}

stop();

tempo_us(50);
tempo_us(50);
tempo_us(50);

```

```

// ecriture dans l'octet Output Data Mode
start();

```

```

    ak=send_i2c(0x42);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c('G');
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c(0x4e);
    if(ak==0)
    {
        stop();
        goto begin;
    }
    ak=send_i2c(0x00); // heading mode
    if(ak==0)
    {
        stop();
        goto begin;
    }
}

stop();

tempo_us(50);
tempo_us(50);
tempo_us(50);
}
/*-----*/
unsigned long retour_angle_boussole(void)
{
    short int MSB,LSB,angle;

begin:
    start();
        ak = send_i2c(0x43);
        if(ak==0)
        {
            stop();
            goto begin;
        }
    }
}

```

```

    MSB = reception_i2c(1);          // reception de MSB
avec ACK
    LSB = reception_i2c(0);          // reception de LSB avec
NACK
    stop();

    MSB <<= 8;
    angle = (MSB | LSB)/20;

    afficher_donnees(0xC2,angle);

    return angle;
}

```

1.4. Fichier lcd_4bits.c

```

#include <p18cxxx.h>
#include "setup.h"
/*-----*/
void lcd_car(unsigned char caractere) // envoi caractere
{
    unsigned char buffer;

    buffer = caractere;
    buffer <<= 2;
    caractere >>= 2;

    tempo_us(50);
    tempo_us(50);
    RS = 1;          // mode caractere
    E = 0;
    DATA = (caractere&0b00111100)|(DATA&0b00000011);
// selectionne 4 bits poids fort de caractere et compare avec 4 bits poids faible de data
pour ecrire sur DB7 ... DB4
    tempo_us(5);
    E = 1;          // démarre l'envoi
    tempo_us(5);
    E = 0;          // arrete l'envoi
    DATA = (buffer&0b00111100)|(DATA&0b00000011);

```

```

    tempo_us(5);
    E = 1;
    tempo_us(5);
    E = 0;
    tempo_ms(5);
}
/*-----*/
void lcd_com(unsigned char commande) // envoi commande
{
    unsigned char buffer;

    buffer = commande;
    buffer <<= 2;
    commande >>= 2;

    tempo_us(50);
    tempo_us(50);
    RS = 0;          // mode commande
    E = 0;
    DATA = (commande&0b00111100)|(DATA&0b00000011);
// selectionne 4 bits poids fort de commande et compare avec 4 bits poids faible de data
pour ecrire sur DB7 ... DB4
    tempo_us(5);
    E = 1;          // démarre l'envoi
    tempo_us(5);
    E = 0;          // arrete l'envoi
    DATA = (buffer&0b00111100)|(DATA&0b00000011);
    tempo_us(5);
    E = 1;
    tempo_us(5);
    E = 0;
    tempo_ms(5);
}
/*-----*/
void lcd_com_init(unsigned char commande) // envoi 4 bits uniquement de commande
{
    commande >>= 2;
    RS = 0;          // mode commande
    E = 0;
    DATA = (commande&0b00111100)|(DATA&0b00000011);
    tempo_us(5);

```

```

    E = 1;                // démarre l'envoi
    tempo_us(5);
    E = 0;                // arrete l'envoi
}
/*-----*/
void init_lcd(void)      // initialise l'afficheur
{
    tempo_ms(30);
    lcd_com_init(0x30);
    //0x38 pour dire qu'on envoi sur 4bits
    tempo_ms(5);
    lcd_com_init(0x30);
    tempo_us(50);
    tempo_us(50);
    lcd_com_init(0x30);
    lcd_com_init(0x20);
    lcd_com(0x28);
    lcd_com(0x0C);
    //0x0E pour dire qu'on eteint l'ecran et qu'on affiche le curseur
    lcd_com(0x06);      //pour decaler le curseur d'un cran vers la droite
    lcd_com(0x01);      //pour effacer l'ecran
    tempo_ms(5);
}
/*-----*/
void lcd_str(unsigned char *str) //ecrire une chaine
{
    unsigned char i=0;

    while(str[i] != '\0')    // tant qu'on est pas a la fin de la chaine
    {
        lcd_car(str[i]);
        i++;
    }
}

```

1.5. Fichier can.c

```

#include <p18cxxx.h>
#include "setup.h"
#include "can.h"
/*-----*/
void init_can(void)
{
    ADCON1 = 0b00001110;
    ADCON0 = 0b01000001;
    ADCON2 = 0b00100010;
}
/*-----*/
unsigned char convertir(void)
{
    tempo_us(5);
    ADCON0bits.GO = 1;      /* lancement conversion */
    while(ADCON0bits.GO);  /* attente fin de conversion */
    return (ADRESH);        /* retour du resultat */
}
/*-----*/
unsigned long retour_angle_can(void)
{
    unsigned long angle;
    unsigned char valeur_can;

    valeur_can = convertir();

    angle = valeur_can;
    angle = angle*360/255;

    afficher_donnees(0xD2,angle);

    return angle;
}

```

1.6. Fichier commande_hacheur.c

```
#include <p18cxxx.h>
#include "setup.h"

/*****VARIABLES_GLOBALES*****/
static short etat = 0;
/*-----*/
/*fonction qui permet d'être sur pour la suite que le dernier transistor utilisé est bien le T1*/
/*la mettre juste avant le while(1) du int main()*/
void init_moteur(void)
{
    T4 = 1;
    tempo_ms(vitesse_moteur);
    T4 = 0;
    tempo_ms(vitesse_moteur);
    T2 = 1;
    tempo_ms(vitesse_moteur);
    T2 = 0;
    tempo_ms(vitesse_moteur);
    T3 = 1;
    tempo_ms(vitesse_moteur);
    T3 = 0;
    tempo_ms(vitesse_moteur);
    T1 = 1;
    tempo_ms(vitesse_moteur);
    T1 = 0;
    tempo_ms(vitesse_moteur);
}
/*-----*/
void sens_horaire(unsigned long pas)
{
    unsigned long j;
    LED1 = 1;
    for(j=0; j<pas; j++)
    {
        switch(etat)
        {
            case 0 : T4 = 1;
                    tempo_ms(vitesse_moteur);
                    T4 = 0;
```

```

                    etat = 1;
                    tempo_ms(vitesse_moteur);
                    break;

            case 1 : T2 = 1;
                    tempo_ms(vitesse_moteur);
                    T2 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 2;
                    break;

            case 2 : T3 = 1;
                    tempo_ms(vitesse_moteur);
                    T3 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 3;
                    break;

            case 3 : T1 = 1;
                    tempo_ms(vitesse_moteur);
                    T1 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 0;
                    break;

            default : break;
        }
    }

    LED1 = 0;
}

/*-----*/
```

```

void sens_anti_horaire(unsigned long pas)
{
    unsigned long i,j;
    LED2 = 1;
    for(i=0; i<pas; i++)
    {
        switch(etat)
        {
            case 0 : T3 = 1;
                    tempo_ms(vitesse_moteur);
                    T3 = 0;
                    etat = 1;
                    tempo_ms(vitesse_moteur);
                    break;

            case 1 : T2 = 1;
                    tempo_ms(vitesse_moteur);
                    T2 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 2;
                    break;

            case 2 : T4 = 1;
                    tempo_ms(vitesse_moteur);
                    T4 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 3;
                    break;

            case 3 : T1 = 1;
                    tempo_ms(vitesse_moteur);
                    T1 = 0;
                    tempo_ms(vitesse_moteur);
                    etat = 0;
                    break;

            default : break;
        }
    }
    LED2 = 0;
}
/*-----*/

```

```

void tourner_vers_nord(unsigned long degre) // coordonnees boussole
{
    unsigned long pas;

    if(degre>180)
    {
        pas = (360 - degre)*200/360;
        if(pas>5)
            sens_horaire(pas);
    }
    else
    {
        pas = (degre*200)/360;
        if(pas>5)
            sens_anti_horaire(pas);
    }
}
/*-----*/
void tourner_avec_pot(unsigned long degre_bous, unsigned long degre_pot)
// coordonnees boussole et potentiometre
{
    unsigned long pas;

    if(degre_bous<degre_pot)
    {
        pas = (degre_pot - degre_bous)*200/360;
        if(pas>5)
            sens_horaire(pas);
    }
    else
    {
        pas = (degre_bous - degre_pot)*200/360;
        if(pas>5)
            sens_anti_horaire(pas);
    }
}

```

1.7. Fichier setup.h

```

/*****INPUT_OUTPUT*****/
#define LED0      LATEbits.LATE0
#define LED1      LATEbits.LATE1
#define LED2      LATEbits.LATE2

#define T1        LATBbits.LATB1
#define T2        LATBbits.LATB2
#define T3        LATBbits.LATB3
#define T4        LATBbits.LATB4

#define RS        LATDbits.LATD0
#define E         LATDbits.LATD1
#define DATA     LATD

#define SCL        LATCbits.LATC0
#define SDA        LATCbits.LATC1
#define SDA_return PORTCbits.RC1
#define SDA_IO     TRISCbits.TRISC1

/*****PROTOTYPES*****/
void init_port(void);
void tempo_ms(unsigned short duree);
void tempo_us(unsigned char duree);
void afficher_donnees(unsigned char position curseur, unsigned long angle);

/*****VARIABLES_GLOBALES*****/

#define vitesse_moteur 40

```

1.8. Fichier can.h

```

/*****PROTOTYPES*****/
void init_can(void);
unsigned char convertir(void);
unsigned long retour_angle_can(void);

```

1.9. Fichier boussole.h

```

/*****PROTOTYPES*****/
void tpo_i2c(void);
void start(void);
void stop(void);
unsigned char send_i2c(unsigned char mot);
unsigned char reception_i2c(char ack);
void init_boussole(void);
void calibrage(void);
unsigned long retour_angle_boussole(void);

```

1.10. Fichier lcd_4bits.h

```

/*****PROTOTYPES*****/
void init_port_lcd(void);           //init port
void lcd_car(unsigned char caractere);
void lcd_com(unsigned char commande);
void lcd_com_init(unsigned char commande);
void init_lcd(void);
void lcd_str(unsigned char *str);

```

1.11. Fichier commande_hacheur.h

```

/*****PROTOTYPES*****/
void sens_horaire(unsigned long n);           // x>10000 & n>1
void sens_anti_horaire(unsigned long n);     // x>10000 & n>1
void init_moteur(void);
void tourner_avec_pot(unsigned long degre_bous, unsigned long degre_pot);
void tourner_vers_nord(unsigned long degre);

```

2. DATASHEETS

2.1. Registre du timer 0

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7	TMR0ON: Timer0 On/Off Control bit 1 = Enables Timer0 0 = Stops Timer0
bit 6	T08BIT: Timer0 8-Bit/16-Bit Control bit 1 = Timer0 is configured as an 8-bit timer/counter 0 = Timer0 is configured as a 16-bit timer/counter
bit 5	T0CS: Timer0 Clock Source Select bit 1 = Transition on T0CKI pin 0 = Internal instruction cycle clock (CLKO)
bit 4	T0SE: Timer0 Source Edge Select bit 1 = Increment on high-to-low transition on T0CKI pin 0 = Increment on low-to-high transition on T0CKI pin
bit 3	PSA: Timer0 Prescaler Assignment bit 1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler. 0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.
bit 2-0	T0PS2:T0PS0: Timer0 Prescaler Select bits 111 = 1:256 Prescale value 110 = 1:128 Prescale value 101 = 1:64 Prescale value 100 = 1:32 Prescale value 011 = 1:16 Prescale value 010 = 1:8 Prescale value 001 = 1:4 Prescale value 000 = 1:2 Prescale value

2.2. Registre timer 2

REGISTER 13-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7	Unimplemented: Read as '0'
bit 6-3	T2OUTPS3:T2OUTPS0: Timer2 Output Postscale Select bits 0000 = 1:1 Postscale 0001 = 1:2 Postscale • • • 1111 = 1:16 Postscale
bit 2	TMR2ON: Timer2 On bit 1 = Timer2 is on 0 = Timer2 is off
bit 1-0	T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits 00 = Prescaler is 1 01 = Prescaler is 4 1x = Prescaler is 16

2.3. Registre Convertisseur analogique numérique(CAN)

2.3.1. Registre ADCON0

REGISTER 21-1: ADCON0: A/D CONTROL REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON
bit 7							bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

- bit 7-6 **Unimplemented:** Read as '0'
- bit 5-2 **CHS3:CHS0:** Analog Channel Select bits
0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)^(1,2)
0110 = Channel 6 (AN6)^(1,2)
0111 = Channel 7 (AN7)^(1,2)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented⁽²⁾
1110 = Unimplemented⁽²⁾
1111 = Unimplemented⁽²⁾
- bit 1 **GO/DONE:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress
0 = A/D Idle
- bit 0 **ADON:** A/D On bit
1 = A/D converter module is enabled
0 = A/D converter module is disabled

- Note 1:** These channels are not implemented on 28-pin devices.
Note 2: Performing a conversion on unimplemented channels will return a floating input measurement.

2.3.2. Registre ADCON1

U-0	U-0	R/W-0	R/W-0	R/W-0 ⁽¹⁾	R/W ⁽¹⁾	R/W ⁽¹⁾	R/W ⁽¹⁾
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
-n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

- bit 7-6 **Unimplemented:** Read as '0'
- bit 5 **VCFG1:** Voltage Reference Configuration bit (VREF- source)
1 = VREF- (AN2)
0 = VSS
- bit 4 **VCFG0:** Voltage Reference Configuration bit (VREF+ source)
1 = VREF+ (AN3)
0 = VDD
- bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7 ⁽²⁾	AN6 ⁽²⁾	AN5 ⁽²⁾	AN4	AN3	AN2	AN1	AN0
0000 ⁽¹⁾	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111 ⁽¹⁾	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input

D = Digital I/O

2.3.3. Registre ADCON2

REGISTER 21-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Legend:

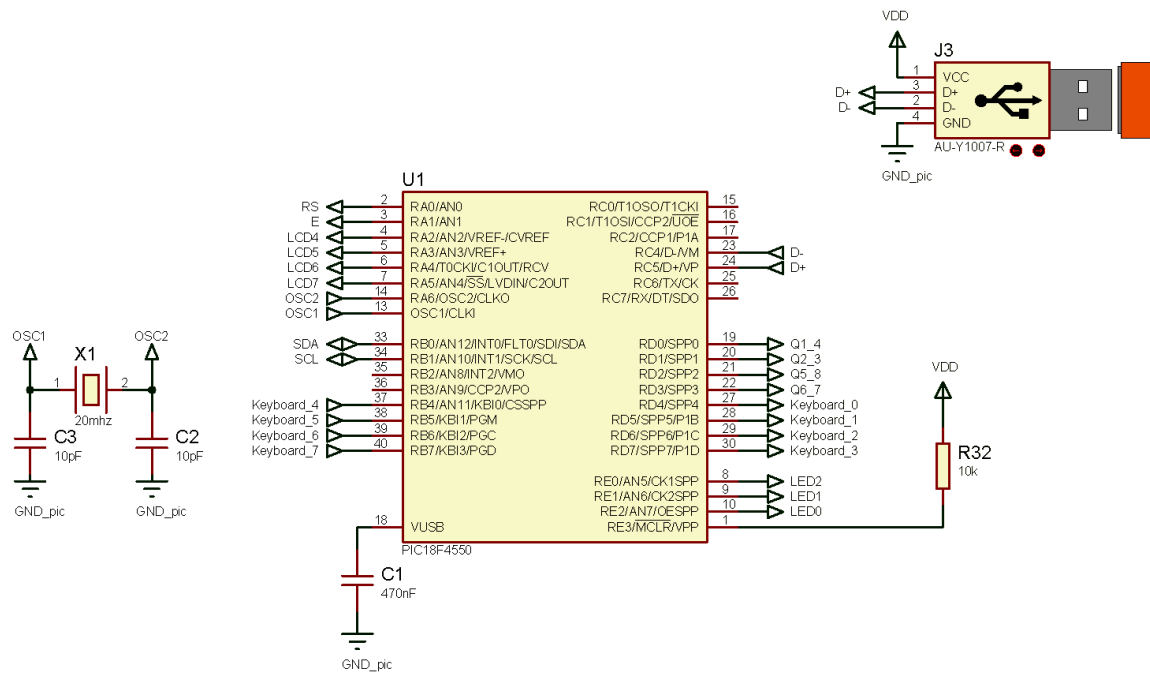
R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
 -n = Value at POR '1' = Bit is set '0' = Bit is cleared x = Bit is unknown

- bit 7 **ADFM:** A/D Result Format Select bit
 1 = Right justified
 0 = Left justified
- bit 6 **Unimplemented:** Read as '0'
- bit 5-3 **ACQT2:ACQT0:** A/D Acquisition Time Select bits
 111 = 20 TAD
 110 = 16 TAD
 101 = 12 TAD
 100 = 8 TAD
 011 = 6 TAD
 010 = 4 TAD
 001 = 2 TAD
 000 = 0 TAD⁽¹⁾
- bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits
 111 = FRC (clock derived from A/D RC oscillator)⁽¹⁾
 110 = FOSC/64
 101 = FOSC/16
 100 = FOSC/4
 011 = FRC (clock derived from A/D RC oscillator)⁽¹⁾
 010 = FOSC/32
 001 = FOSC/8
 000 = FOSC/2

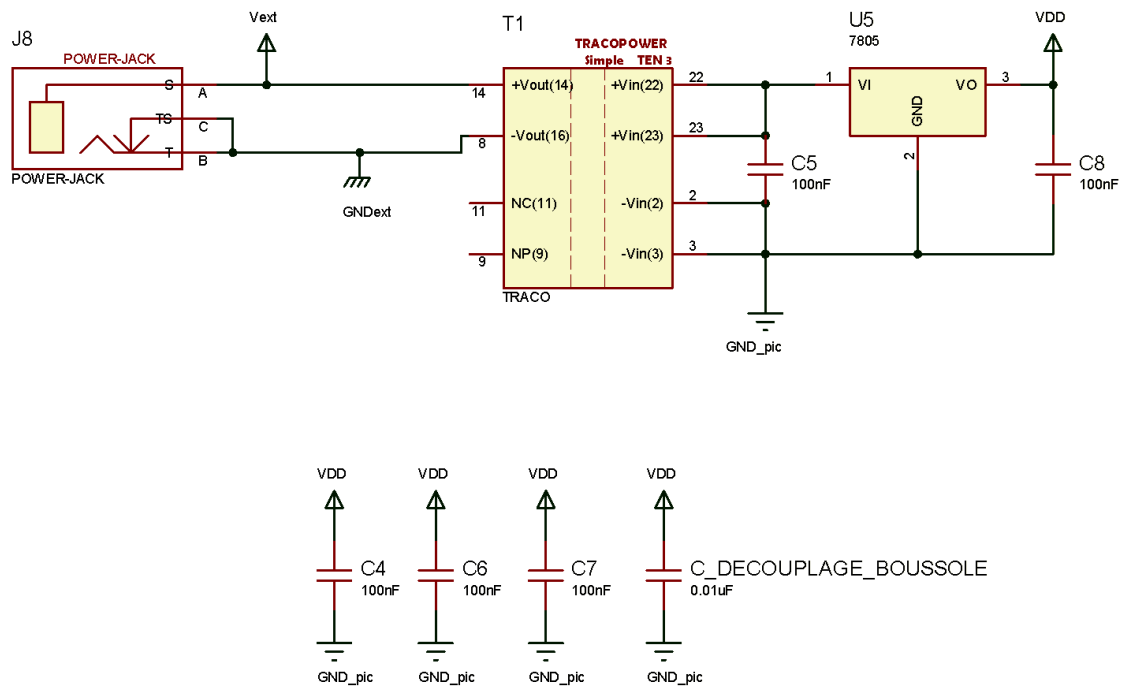
Note 1: If the A/D FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the A/D clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

3. SCHEMA PROTEUS

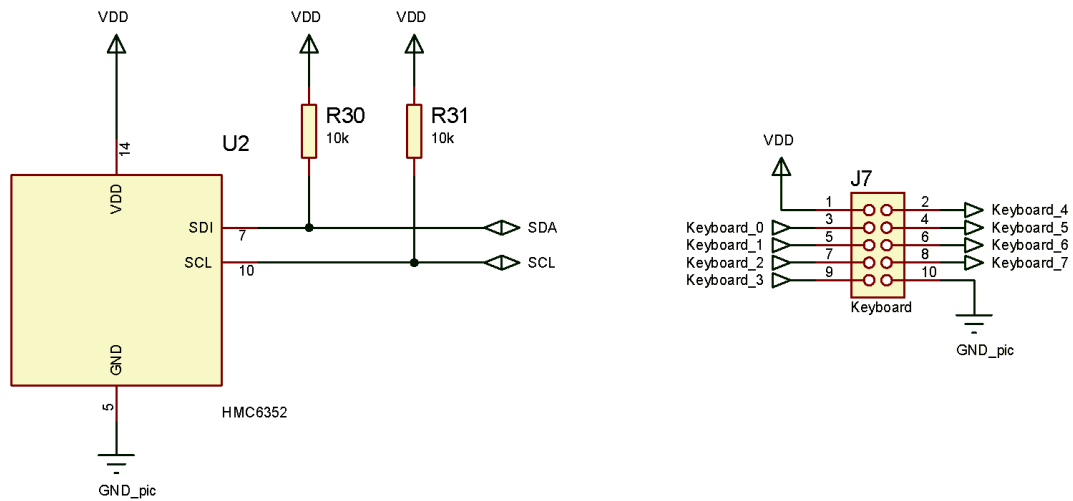
3.1. Microcontrôleur



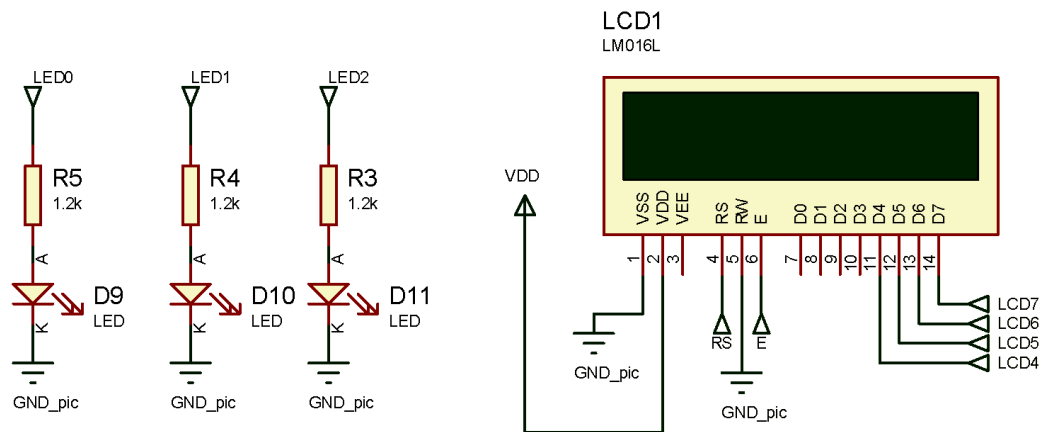
3.2. Alimentation



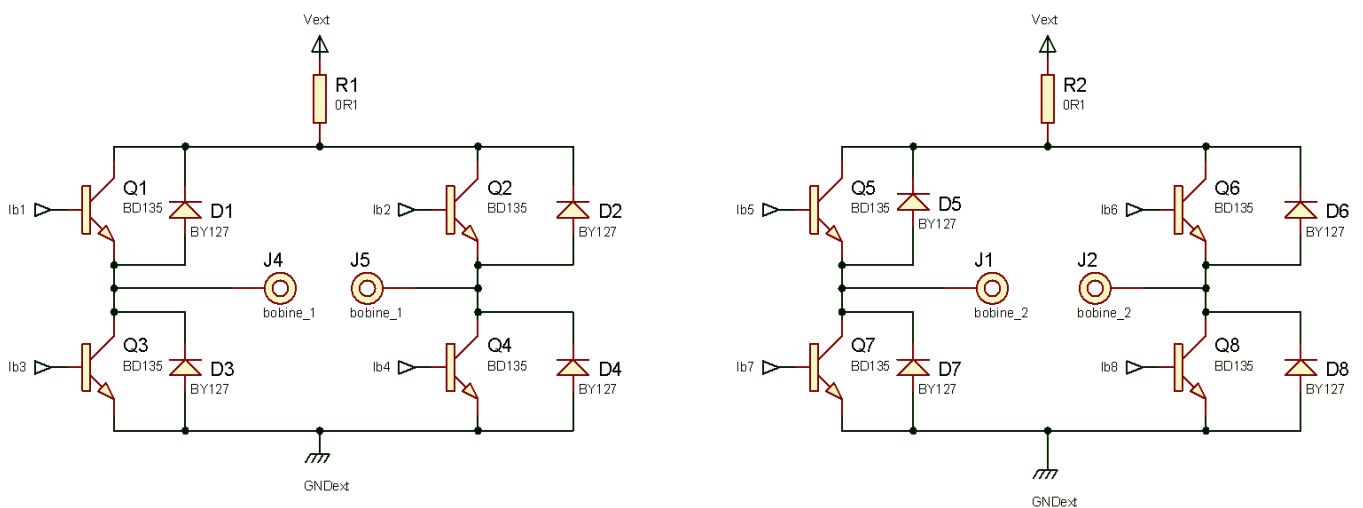
3.3. Input



3.4. Output



3.5. Hacheurs



3.6. Isolation galvanique / Amplification

