

Projet de synthèse 2013

Interrupteur tactile

Réalisé par Mathieu Grenard
Jordane Malleret et julien Roquette

Sommaire

Introduction

I) Entrées/Sorties

- 1) Touche capacitive
- 2) CAN
- 3) Afficheur LCD

II) Commande de la lampe

- 1) Présentation
- 2) Tests et Améliorations

Conclusion

Annexes

Introduction

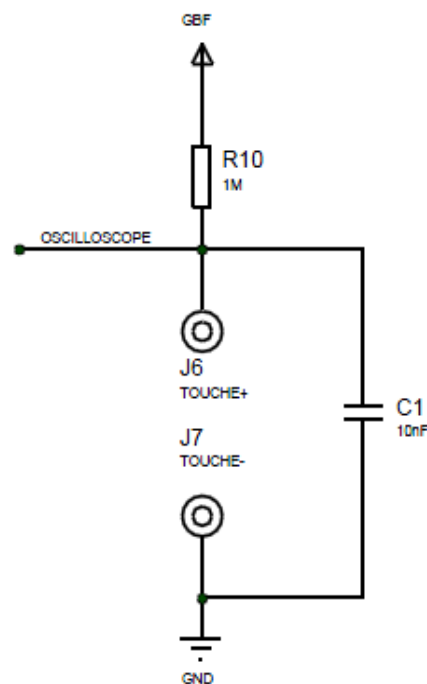
Au travers de ce projet, nous devons créer une commande à partir d'une touche capacitive. Pour cela, nous avons à disposition un micro-contrôleur PIC18F4550 HID associé à une plaque permettant de wrapper les composants supplémentaires et nécessaires au projet. Nous utiliserons également un afficheur LCD pour faciliter la visualisation des tests et des résultats. L'isolation galvanique se fera à l'aide de deux optocoupleurs.

I) Entrées/Sorties

1) Touche capacitive

Avant de nous lancer concrètement dans le projet, nous avons effectué des tests en simulant une touche capacitive pour bien comprendre son fonctionnement et déterminer la meilleure mise en œuvre possible. Une touche capacitive utilise le corps humain comme une capacité, lorsque l'on appui sur la touche, cela revient à remplacer cette touche par un condensateur. Si aucune appui n'est effectué, cela revient à laisser le circuit ouvert. Une touche tactile peut être facilement simulée via deux fils en l'air, dont un relié à la masse.

Tout d'abord, nous avons réalisé le test suivant en visualisant le signal de sortie sur un oscilloscope :

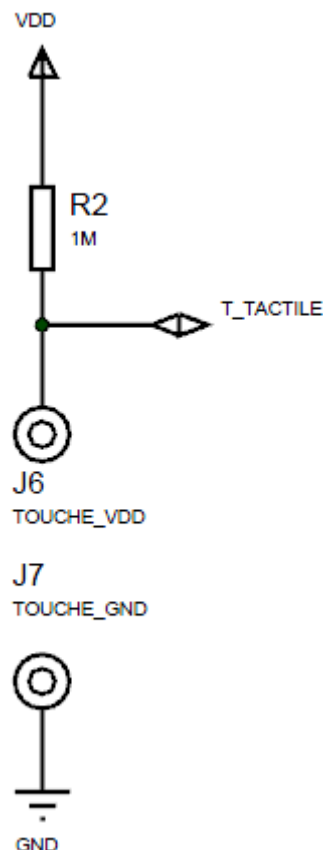


Il est nécessaire dans ce cas de charger puis décharger le condensateur de 10nF. Nous avons pour cela utilisé un GBF (signal rectangulaire) à la place d'une alimentation continue. Cela revient à passer la touche tactile en entrée puis en sortie afin de décharger les condensateurs.

Suite à cette expérience, il s'est avéré logiquement que le temps de charge des condensateurs augmente lors d'un appui sur la touche capacitive vu que l'on rajoute un second condensateur en parallèle du premier.

Cette solution fonctionne correctement mais nous avons cherché à améliorer ce système afin de rendre sa conception plus facile et plus efficace.

Nous avons donc effectué un nouveau test avec un autre montage :



Nous avons retiré le premier condensateur et gardé uniquement la touche capacitive. L'avantage avec ce montage : il n'est plus nécessaire de passer la broche en entrée puis en sortie durant un certain nombre de cycle car il n'y a plus de condensateur à décharger.

Finalement, nous avons donc choisi de mettre en œuvre le second montage car il est plus simple de conception et les résultats demandés par le cahier des charges sont quand même respectés.

Une fois la solution choisie, nous avons utilisé une touche tactile créée par les étudiants des années précédentes que nous avons wrappé sur la plaque au même titre que l'afficheur LCD.

2) CAN

Le module de conversion analogique numérique est intégré à l'intérieur du PIC18F4550.

On se sert de ce module pour détecter un appui sur la touche tactile. Il possède 13 entrées sur le PIC18F4550 et fourni un résultat sur 10 bits. Seule l'entrée AN0 est utilisée dans ce projet.

Ce module possède 5 registres :

- 3 registres de contrôle (ADCON0 à ADCON2)
- 2 registres pour stocker le résultat de la conversion (ADRESH et ADRESL)

ADCON0

b7	b6	b5	b4	b3	b2	b1	b0
-	-	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

Ce registre permet de sélectionner l'entrée sur laquelle la conversion sera réalisée (bits CHS0 à CHS3), d'autoriser (ADON) et de lancer la conversion (GO/DONE). L'entrée AN0 est sélectionnée et la conversion doit être autorisée. Ce registre sera donc initialisé à 0x01.

ADCON1

b7	b6	b5	b4	b3	b2	b1	b0
-	-	VCFG0	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0

Ce registre permet de configurer les broches AN0 à AN12 en entrées analogiques ou en entrées/sorties numériques (bits PCFG0 à PCFG3). Il permet aussi de configurer les tensions de référence du convertisseur V_{REF+} et V_{REF-} (bits VCFG0). L'entrée AN0 est utilisée et les tensions de référence sont : $V_{ref+} = V_{dd}$ et $V_{ref-} = V_{ss}$. Il faut donc initialiser ce registre à 0x0E.

ADCON2

b7	b6	b5	b4	b3	b2	b1	b0
ADFM	-	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0

Pour ce registre, seul le bit 7 est mis à 1 pour que le résultat de la conversion soit justifié à droite. Ainsi, les 2 bits de poids fort seront dans ADRESH et les 8 autres dans ADRESL. On a donc $ADCON2 = 0x80$.

Algorithmes

Fonction d'initialisation du convertisseur : init_can

Début

ADCON1 = 0x0E

ADCON0 = 0x01

ADCON2 = 0x80

Fin

Fonction de conversion : convert

Début

Temporisation (5ms)

ADCON0bits.GO = 1

! Départ conversion

Tant que (ADCON0bits.GO = 1)

Ne rien faire

ftq

renvoi ADRES

! registres ADRESH et ADRESL

Fin

Fonction permettant de détecter un appui sur la touche tactile et de calculer le rapport cyclique :

Début

Variables : caractère non signé *evolution*, *i*

entier non signé *CAN_val*

i = 0

CAN_val = convert ()

si (*CAN_val* < 767) ! valeur numérique correspondant à 3,75V

si (*Rap_cyc* >= 50)

Rap_cyc = 0

evolution = 1

sinon

Rap_cyc = 100

evolution = 0

fsi

aff_rap_cyc ()

tant que (*CAN_val* < 767)

CAN_val = convert ()

Temporisation (100ms)

i = *i* + 1

si (*i* = 10)

i = 0

si (*evolution* = 1)

Rap_cyc = *Rap_cyc* + 20

si (*Rap_cyc* = 100)

evolution = 0

fsi

sinon

Rap_cyc = *Rap_cyc* - 20

si (*Rap_cyc* = 0)

evolution = 1

fsi

fsi

fsi

ftq

fsi

Fin

3) Afficheur LCD

L'afficheur est utilisé en mode 4 bits. Il est connecté au port D du micro-contrôleur, il faut donc l'initialiser en sortie en mettant tous les bits du registre TRISD à 0. Dans toutes les fonctions, les broches RS, R/W et E de l'afficheur sont connectées aux broches RD0, RD1 et RD2 du micro-contrôleur. Les broches de donnée D4 à D7 de l'afficheur sont connectées aux broches RD4 à RD7 du micro-contrôleur.

Algorithmes

Fonction permettant d'envoyer un caractère : lcd_car

Début

Entrée : caractère non signé *car*

Temporisation (100µs)

RS = 1

R/W = 0

E = 0

$DATA = ((car \text{ ET } 0xF0) \text{ OU } (DATA \text{ ET } 0x0F))$! Stockage des bits de poids fort de *car*

Temporisation (10µs)

E = 1

Temporisation (10µs)

E = 0

$car = car * 16$

$DATA = ((car \text{ ET } 0xF0) \text{ OU } (DATA \text{ ET } 0x0F))$! Stockage des bits de poids faible de *car*

Temporisation (10µs)

E = 1

Temporisation (10µs)

E = 0

Fin

La fonction permettant d'envoyer une commande (lcd_com) est identique sauf que RS est mis à 0.

Fonction permettant d'envoyer une commande lors de l'initialisation : lcd_com_init

Début

Entrée : caractère non signé *commande*

RS = 1

R/W = 0

E = 0

$DATA = ((commande \text{ ET } 0xF0) \text{ OU } (DATA \text{ ET } 0x0F))$! Stockage des bits de poids fort
!de commande

Temporisation (10µs)

```

E = 1
Temporisation (10µs)
E = 0

```

Fin*Fonction d'initialisation : lcd_init***Début**

```

Temporisation (50 ms)
lcd_com_init (0x30)
Temporisation (5 ms)
lcd_com_init (0x30)
Temporisation (150 µs)
lcd_com_init (0x30)
Temporisation (150µs)
lcd_com_init (0x20)
lcd_com (0x28)           ! Mode 4 bits, affichage de caractères 5x8 sur 4 lignes
lcd_com (0x0C)
lcd_com (0x01)
lcd_com (0x06)
Temporisation (5 ms)

```

Fin*Fonction permettant d'afficher une chaîne de caractères : lcd_str***Début**

Entrée : chaîne de caractères *str*
 Variable : caractère non signé *i*

```

Tant que (str(i) != '\0')
    lcd_car (str(i))
    i = i + 1

```

ftq**Fin**

Intéressons nous maintenant aux fonctions de temporisation, elles seront réalisées à l'aide du timer0.

*Fonction de temporisation en milliseconde : tpo_ms***Début**

Entrée : entier non signé *trm_val*
Variable : caractère non signé *donnée*

```

T0CON = 0x04           ! Mode 16 bits, pré-diviseur 32
tmr_val = 65535 - (375 * tmr_val)
donnée = tmr_val / 256
TMR0H = donnée
TMR0L = tmr_val

```


TMR0ON = 1

tant que (TMR0IF = 0)
 ne rien faire

ftq

TMR0ON = 0

TMR0IF = 0

Fin

Fonction de temporisation en microseconde : tpo_us

Début

Entrée : entier non signé *trm_val*

Variable : caractère non signé *donnée*

T0CON = 0x08

! Mode 16 bits, sans pré-diviseur

tmr_val = 65535 – (12 * *tmr_val*)

donnée = *tmr_val* / 256

TMR0H = *donnée*

TMR0L = *tmr_val*

TMR0ON = 1

tant que (TMR0IF = 0)
 ne rien faire

ftq

TMR0ON = 0

TMR0IF = 0

Fin

Le code complet du programme est visible en annexes 1, 2 et 3.

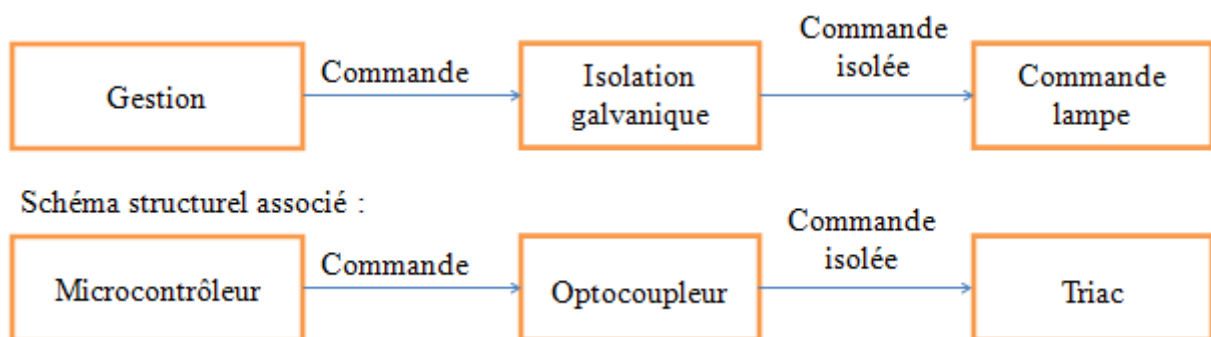
II) Commande de la lampe

1) Présentation

Objectif :

La partie commande fonctionnera sur une charge 12V matérialisée par une lampe de voiture et sera complètement isolée galvaniquement du système de commande.
L'alimentation de la charge se fera au travers d'un transformateur 220V -> 12V non redressé.

1^{er} schéma fonctionnel :



Principe de fonctionnement :

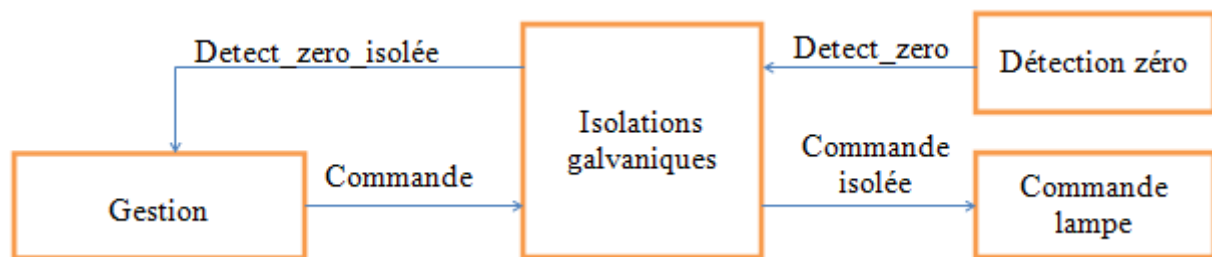
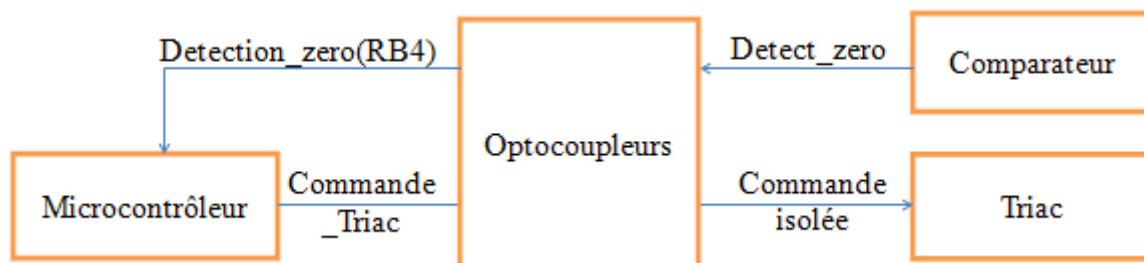
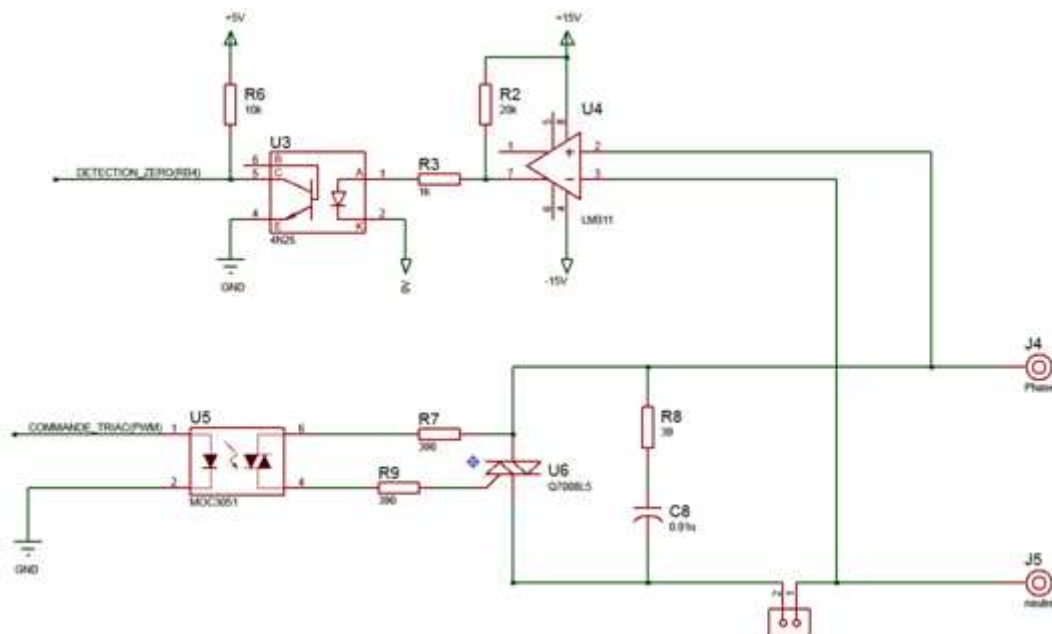
Le microcontrôleur envoie un signal de commande qui est isolé galvaniquement par l'optocoupleur. La commande isolée vient commander la gâchette du triac qui permet le pilotage de la lampe. Pour faire varier l'intensité de la lampe il suffit par conséquent de faire varier le rapport cyclique du signal de commande (gâchette triac) qui change les temps de conduction du triac, un rapport de 0% si l'on veut que la lampe soit éteinte un rapport de 100% si l'on veut que celle-ci soit allumée au maximum.

Problème :

Le triac s'amorce lorsque l'on envoie une tension de gâchette assez élevée pour l'amorcer, puis celui-ci se désamorcera uniquement au prochain passage par zéro de la tension à ses bornes. Par conséquent il faut synchroniser la commande avec les passages par zéro pour obtenir une variation d'intensité correspondante au rapport cyclique.

Solution :

Il faut rajouter une fonction permettant au microcontrôleur de connaître quand le signal 12V alternatif passe par 0, et ainsi se synchroniser avec pour envoyer la commande.

Nouveau schéma Fonctionnel :**Schéma structurel :****Schéma :**

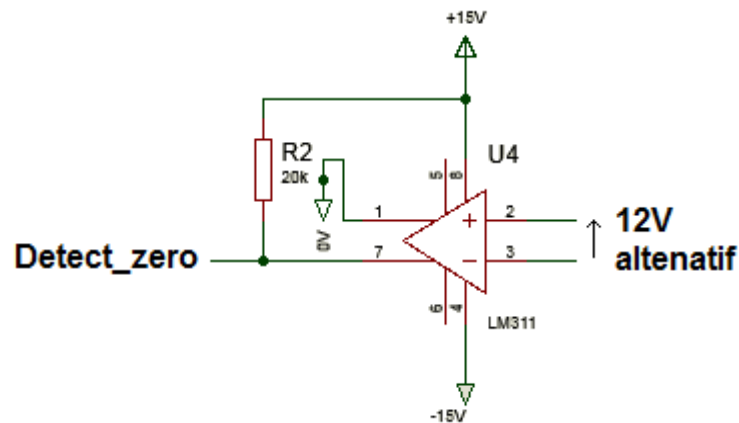
2) Tests et améliorations.

a) Fonction Détection de zéro :

Entrée : Signal alternatif 12V,

Sortie : Detect_zero, signal variant de 0 à 15V ou vice versa à chaque passage par zéro du signal 12V

Schéma :



Chronogramme :

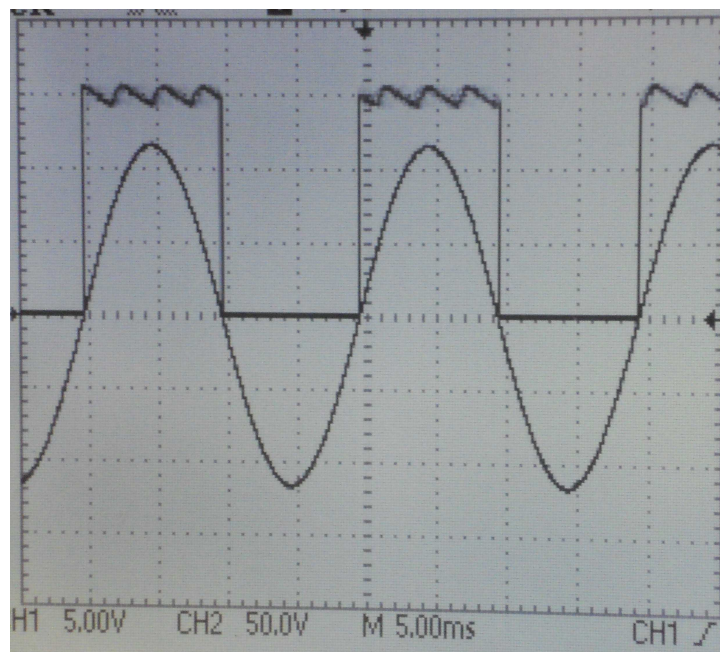


Illustration 1: CH1 : tension sinusoïdale 12V, CH2 : Detect_zero (5V/DIV)

Analyse du chronogramme : La fonction est correcte, Detect_zero change de niveau logique à chaque passage par zéro de la tension.

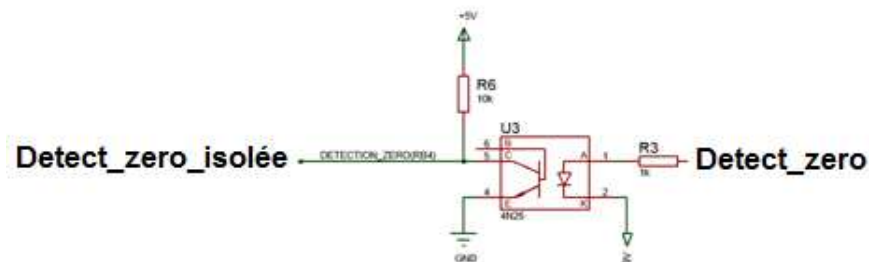
b) Fonction isolation galvanique

1) Détection zéro

Entrée : Detect_zero,

Sortie : Detect_zero_isolée, détection du zéro isolée galvaniquement.

Schéma :



Chronogramme :

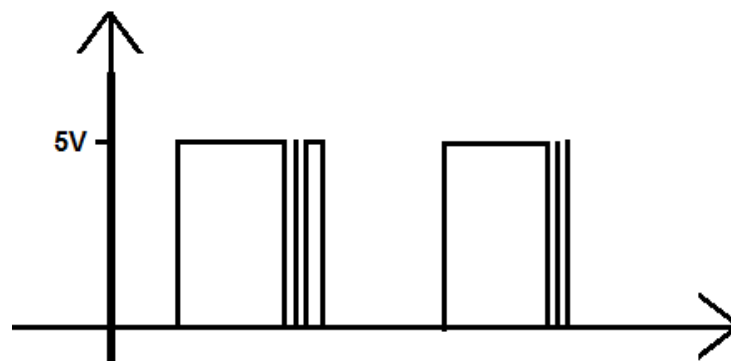


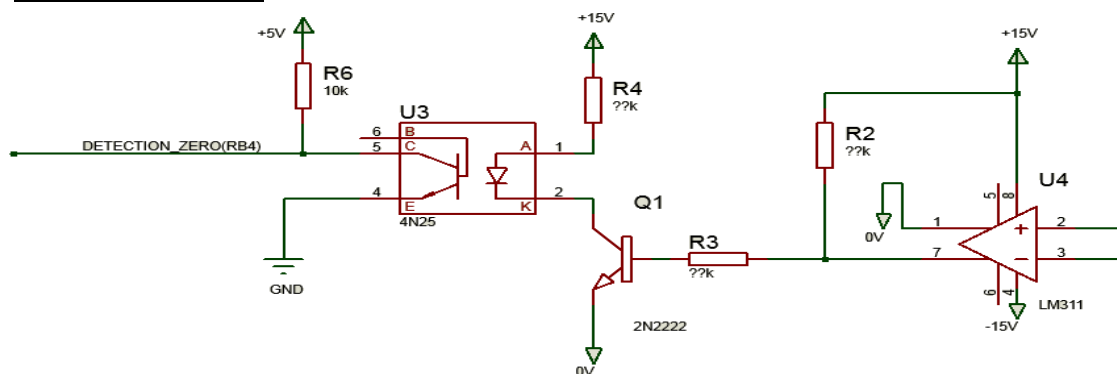
Illustration 2: detect_zero_isolée

Analyse du chronogramme : le chronogramme obtenu ne correspond plus à la détection du zéro souhaitée, on constate une déformation du signal carré et l'apparition de pic de tension parasite.

Origine du problème envisagée : Courant fourni par la sortie de l'ampli trop faible pour commander la diode de l'optocoupleur.

Solution : Faire une amplification en courant à l'aide d'un transistor 2n2222.

Nouveau schéma :



Calculs des valeurs des résistances R2, R3, et R4.

On fixe I_c à 15 mA, soit $R_4=1K\Omega$ en négligeant la tension de seuil de la diode
D'après la documentation technique du 2n2222 (annexe 4), $\beta=35$.

$$I_{bsatmin} = I_c / \beta = 15/35 = 0,429 \text{ mA soit } 429 \mu\text{A}$$

$$i_b = (15 - 0,7) / (R_3 + R_2) \Rightarrow (R_3 + R_2) = (15 - 0,7) / 0,000429 = 33,333k\Omega$$

Pour être certain de saturer le transistor, on prendra $R_3 = 1k\Omega$ et $R_2=10k\Omega$

Chronogramme :

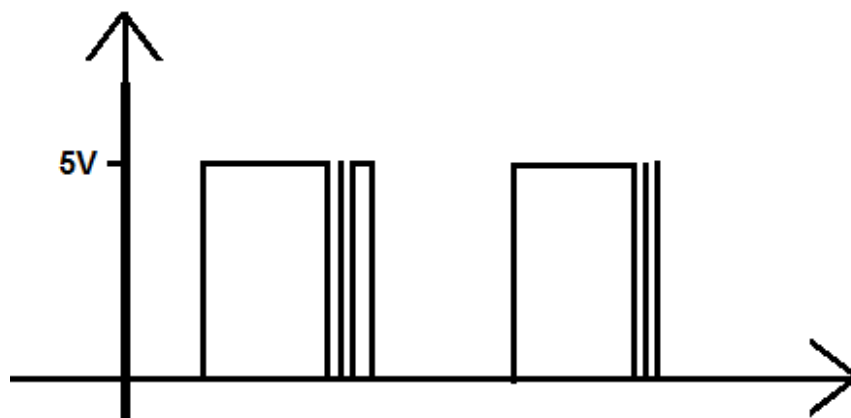


Illustration 3: detect_zero_isolée

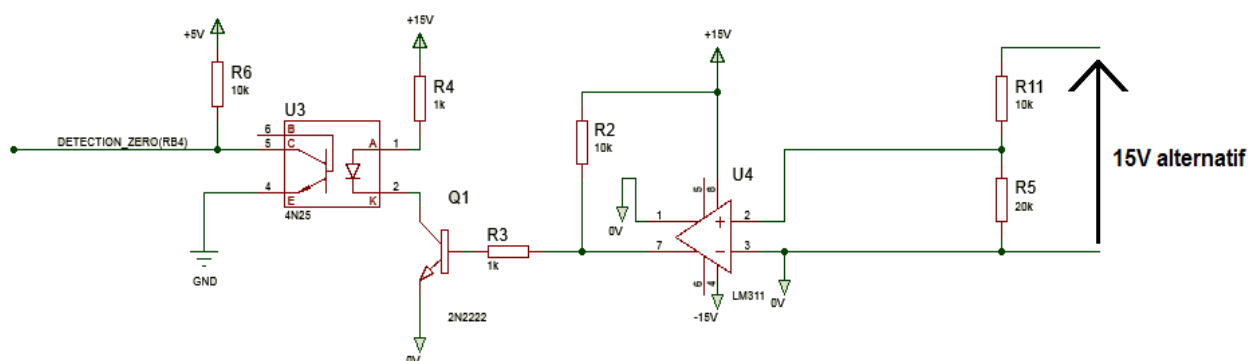
Analyse du chronogramme : On constate que rien ne change, le problème ne vient donc pas de l'optocoupleur.

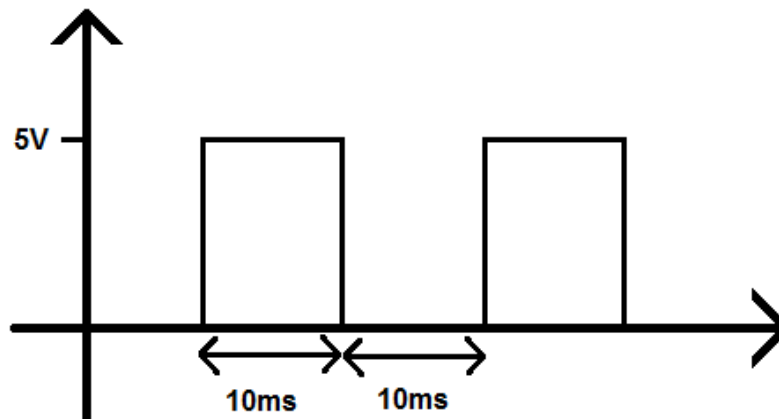
En effectuant différentes mesures, on observe que le signal devient correct lorsque l'on connecte les sondes à certaines masses.

Nouvelle origine du problème envisagée : Problème de masse.

Solution : On décide donc de ne plus utiliser le GBF qui est relié à la terre par le secteur, on le remplace par un transformateur 9V théorique (15V en pratique), ce qui permet d'isoler la tension alternative du secteur, et de relier les masses entre la tension alternative et le comparateur. On appliquera également un diviseur de tension en entrée du comparateur afin d'obtenir les 12V voulu.

Nouveau Schéma :



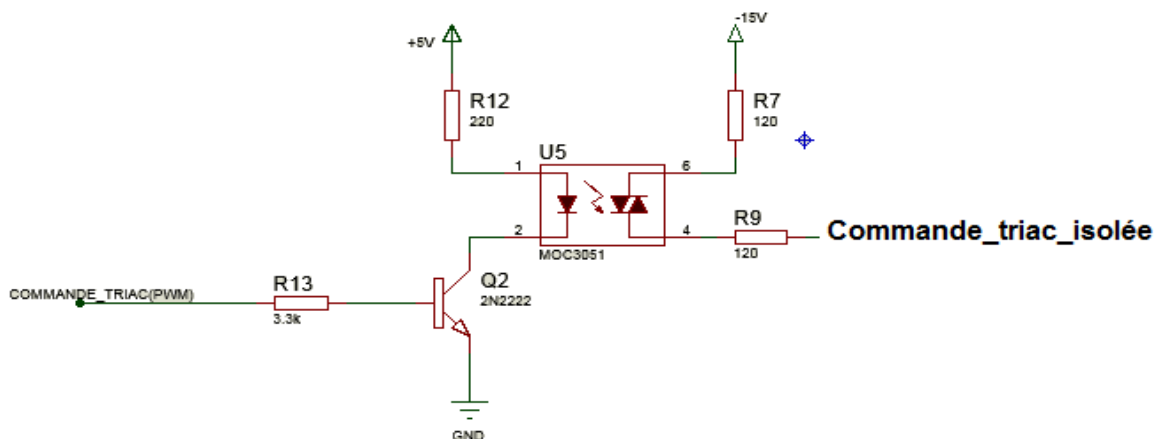
Chronogramme :*Illustration 4: Detect_zero_isolée*

Analyse du Chronogramme : on obtient le signal voulu, un signal carré image de celui obtenu en sortie du comparateur.

2) Commande du Triac.

Entrée : Commande_Triac, signal carré de période 10ms de rapport cyclique variable.

Sortie : Commande_Triac_isolée, commande isolée galvaniquement.

Schéma :**Justification des composants :**

D'après la documentation du constructeur, la tension de la diode du m0c3041, $V_{fmax}=1,5V$. (Annexe 5)

$i_c = 5 - 1,5 / R_{12} = 3,5 / R_{12} \Rightarrow$ on veut un I_c de 15mA, $3,5 / 0,015 = 233,33$. On prendra une résistance de 220 Ω .

$$i_b = I_c / \beta = 0,015 / 35 = 429 \mu A$$

$R_{13} = 5 / 0,000429 = 11635 \Omega$, on prendra $R_{13} = 3,3k \Omega$ pour s'assurer que le transistor est saturé.

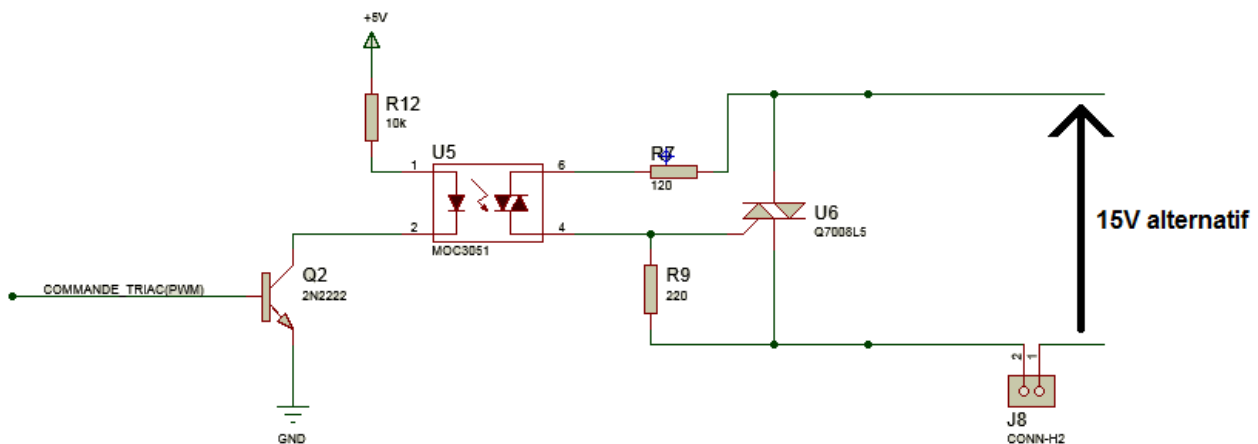
D'après la documentation technique du BTA 700B (annexe 6), le courant d'amorçage IGT est de 50 mA maximum pour les quadrants 1, 2 et 3. Par conséquent on s'assurera de travailler dans le quadrant 2 et 3 en alimentant la gâchette en -15V,

De plus il faut $I_g > 50\text{mA}$ pour assurer l'amorçage, $15/R = 50\text{mA} \Rightarrow R = 15/0,05 = 300$, on prendra $R = 240 \Omega$

Observation : Tension de gâchette mauvaise, mauvais pilotage du triac

Solution : Modification de la commande de la gâchette, ajout d'une résistance de pull-down, et alimentation par le réseau atténué.

Nouveau schéma :



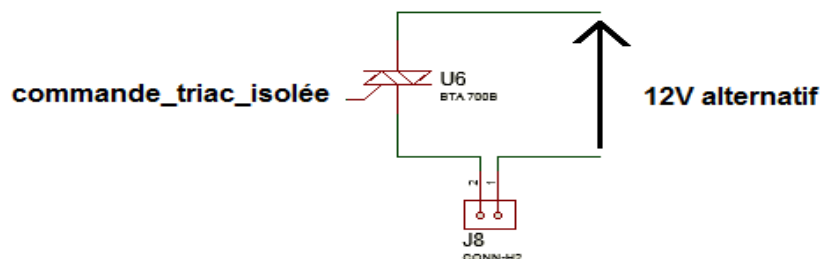
Observation : le triac se commande correctement.

c) Commande lampe

Entrée : Commande_Triac_isolée, signal carré de période 10ms de rapport cyclique variable isolé galvaniquement.

Sortie : Éclairage de la lampe

Schéma :



N'ayant pas de lampe, on remplacera celle-ci par une simple résistance.

Chronogramme : Test effectué et validé.

d) Fonction Gestion :

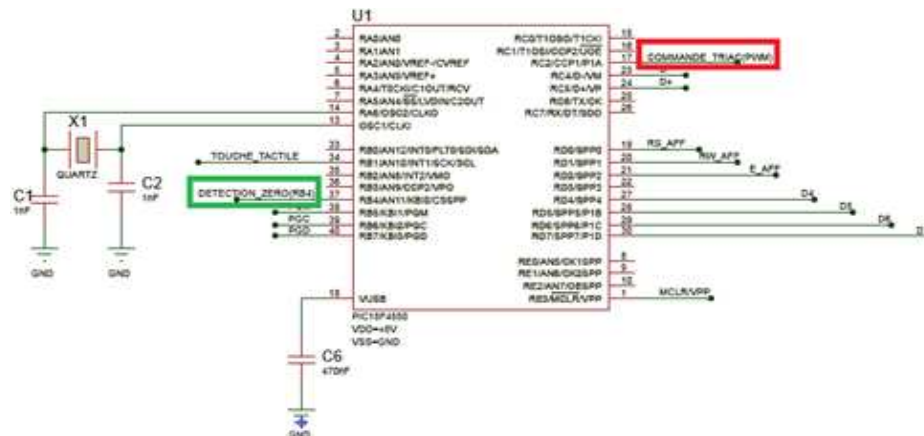
Entrée : Détection_zero(RB4), niveau logique 0-5V.

- Signal permettant la synchronisation avec le passage par zéro de la source alternative.

Sortie : Commande_triac(PWM), niveau logique 0-5V.

- Commande gérant la variation d'intensité de la lampe.

Schéma :



Objectif : Cette fonction a pour but de gérer la commande du triac à chaque détection de zéro, pour cela on utilisera l'interruption sur changement de RB4, qui déclenche une interruption à chaque changement de niveau logique sur la broche RB4. Dans cette interruption on gèrera la MLI.

Remarque: Rapidement on remarque que l'on ne pourra gérer la MLI avec le module du pic, celui-ci utilisant le timer 2, compteur 8 bits (annexe 7), il ne peut pas assurer la MLI d'une période 10 ms avec une fréquence d'oscillation de 48Mhz.

Démonstration : La fréquence d'oscillation est divisée par 4 => 12MHz

Prescaler maximum est de 16 => 0,75 MHz.

$0,01/(1/750000)=7500$, il faut donc compter 7500 cycles pour pouvoir réaliser la MLI, sachant que le timer 2 est un compteur 8 bits, soit 255 cycles, la fonction n'est donc pas réalisable.

Solution : On réalisera une MLI à l'aide de l'interruption du timer 1.

Initialisation Timer 1 :

REGISTER 12-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYN	TMR1CS	TMR1ON
bit 7							bit 0

On initialise le registre T1CON à 0b10011000, un prescaler de 2 (T1CKPS1..T1CKPS0), sur la fréquence interne divisée par 4 (TMR1CS), soit une fréquence de fonctionnement de 6Mhz.

Il faut également autoriser l'interruption du timer 1.

TABLE 12-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	53
PIR1	SPPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	56
PIE1	SPPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	56
IPR1	SPPIP ⁽¹⁾	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	56
TMR1L	Timer1 Register Low Byte								54
TMR1H	Timer1 Register High Byte								54
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNCR	TMR1CS	TMR1ON	54

Il faut donc mettre TMR1IE = 1, TMR1IF = 0, TMR1IP = 0 ;

Initialisation Interruption on change

Pour initialiser l'interruption il faut mettre : RBIE=1, RBIP=0, RBIF=0.

Algorithmes :

Fonction d'interruption sur changement

Début

Si (RBIF=1)

variable 16 bits RapCyc_timer;

variable 8 bits valeur_RB4;

variable 8 bits data;

valeur_RB4 <= RB4;

! lecture du port RB4

RC2 <= 0;

! mise a zéro du port RC2

RapCyc_timer <= 65535 - ((60000 / 100) * Rap_cyc);

TMR1ON <= 0;

! arrêt du timer1

data <= (RapCyc_timer >> 8);

TMR1H <= data;

! chargement de la valeur initiale du timer 1

TMR1L <= RapCyc_timer;

TMR1ON <= 1;

! lancement timer1

RBIF <= 0;

! remise a zéro du flag

finSi

Fin

Fonction d'interruption du timer1

Debut

Si (TMR1IF=1)

TMR1ON = 0;

! arrête le timer 1

SI(Rap_cyc=0)

RC2 = 1;

! met la sortie C2 au niveau 1

FinSi

TMR1IF = 0;

! remet a zéro le flag

FinSi

Fin

Fonctionnement : La gestion de la commande du triac se fait donc uniquement par interruption.

L'interruption sur changement d'état permet de savoir quand le 12V passe par zéro, celle-ci pilote le timer 1 en fonction du rapport cyclique voulu. En fonction du rapport cyclique l'interruption du timer 1 met plus ou moins de temps à arriver (de 0 à 10ms), dès qu'elle est déclenchée l'interruption du timer 1 met la sortie C2 au niveau 1 et stop le timer 1.

La sortie C2 reste à 1 jusqu'à la prochaine interruption déclenché par le passage à zéro.

Conclusion

L'ensemble du projet fonctionne, mais la touche tactile ne correspond pas à celle attendue par le projet car celle-ci pourrait ne pas fonctionner dans d'autres environnements.

Annexes

Annexe 1 : touche tactile.c

```
/** INCLUDES *****/
#include <p18cxxx.h>
#include "bootloader.h"
#include "Rap_cyc.h"
#include "lcd_4b.h"

/** VARIABLES *****/
unsigned char Rap_cyc=0;

#pragma udata

void init_Timer1 (void);
void init_portB (void);
void init_portC (void);
void int_ta1 (void);
void int_on_change (void);

#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode()
{
}

#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode()
{
    int_on_change();
    int_ta1();
}

void main (void)
{
    init_port_lcd();
    init_lcd();
    init_aff();

    adc_init();

    init_Timer1();
    init_portB();
    init_portC();

    INTCONbits.PEIE=1;           // enables all unmasked peripheral interrupts

    while(1)
    {
        RapCyc();
    }
}
```

```

    }
}

// Initialize PORTC
void init_portC(void)
{
    TRISCbits.TRISC2=0;        // pin 2 is an output
    LATCbits.LATC2=0;
}

// Initialize PORTB
void init_portB(void)
{
    TRISBbits.TRISB4=1;        // pin 4 is an input
    INTCON2bits.RBIP=0;        // low priority interrupt
    INTCONbits.RBIF=0;
    INTCONbits.RBIE=1;         // enables the RB port change interrupt
    RCONbits.IPEN=1;           // enables priority levels on interrupts
    INTCONbits.GIE=1;          // enables all unmasked interrupts
}

void init_Timer1(void)
{
    T1CON=0b10011000;          // internal clock, 1:2 prescale value
    PIE1bits.TMR1IE=1;          // enables the TMR1 overflow interrupt
    IPR1bits.TMR1IP=0;          // Low priority interrupt
    PIR1bits.TMR1IF=0;          // TMR1 register did not overflow
}

void int_ta1(void)
{
    if (PIR1bits.TMR1IF)
    {
        T1CONbits.TMR1ON=0;    // stops timer1
        LATCbits.LATC2=1;
        PIR1bits.TMR1IF=0;      // overflow interrupt flag bit reset
    }
}

void int_on_change(void)
{
    if (INTCONbits.RBIF)
    {
        short int RapCyc_timer;
        unsigned char valeur_RB4;
        unsigned char data;

        valeur_RB4=PORTBbits.RB4;
        LATCbits.LATC2=0;
        RapCyc_timer=65535-((60000/100)*Rap_cyc));
        T1CONbits.TMR1ON=0;

        data = (RapCyc_timer >> 8);
        TMR1H = data;
        TMR1L = RapCyc_timer;

        T1CONbits.TMR1ON=1;
        INTCONbits.RBIF=0;
    }
}

```

Annexe 2 : lcd_4b.c

```

#include <p18cxxx.h>
#include "lcd_4b.h"

#define RS          LATDbits.LATD0
#define R_W        LATDbits.LATD1
#define E          LATDbits.LATD2
#define DATA      LATD

// Initialize PORTD
void init_port_lcd (void)
{
    TRISD = 0;           // PORTD is an output
    LATD = 0x00;
}

// Write a character at current cursor position
void lcd_car (unsigned char car)
{
    tpo_us (200);
    RS = 1;
    R_W = 0;
    E = 0;

    DATA = ((car & 0xf0) | (DATA & 0x0f));    // storing higher bits in DATA

    tpo_us (10);
    E = 1;
    tpo_us (10);
    E = 0;
    car <<= 4;

    DATA = ((car & 0xf0) | (DATA & 0x0f));    // storing lower bits in DATA

    tpo_us (10);
    E = 1;
    tpo_us (10);
    E = 0;
}

// Write data to CGRAM/DDRAM
void lcd_com (unsigned char commande)
{
    tpo_us (100);
    RS = 0;
    R_W = 0;
    E = 0;

    DATA = ((commande & 0xf0) | (DATA & 0x0f));

    tpo_us (10);
    E = 1;
    tpo_us (10);
    E = 0;
    commande <<= 4;

    DATA = ((commande & 0xf0) | (DATA & 0x0f));
}

```

```

    tpo_us (10);
    E = 1;
    tpo_us (10);
    E = 0;
}
// Write data to CGRAM/DDRAM during initialization
void lcd_com_init (unsigned char commande)
{
    RS = 0;
    R_W = 0;
    E = 0;

    DATA = ((commande & 0xf0) | (DATA & 0x0f));

    tpo_us (10);
    E = 1;
    tpo_us (10);
    E = 0;
}

// 4-bit initialization
void init_lcd (void)
{
    tpo_ms (50);
    lcd_com_init (0x30); // Function set
    tpo_ms (5);
    lcd_com_init (0x30); // Function set
    tpo_us (150);
    lcd_com_init (0x30); // Function set
    tpo_us (150);
    lcd_com_init (0x20); // Function set
    lcd_com (0x28); // 4-bit interface, 4 display lines, 5x8 characters
    lcd_com (0x0C); // Cursor off and display on
    lcd_com (0x01); // Display clear
    lcd_com (0x06); // Entry mode set
    tpo_ms (5);
}

// Write a string at current cursor position
void lcd_str (unsigned char *str)
{
    unsigned char i=0;
    while(str[i] != '\0')
    {
        lcd_car (str[i]);
        i++;
    }
}

// Fonctions de gestion du timer 0
void tpo_ms (unsigned short tmr_val)
{
    unsigned char data;

    T0CON = 0b00000100; // run in 16-bit timer, 1:32 prescale value
    tmr_val = 65535 - (375 * tmr_val);

    data = (tmr_val >> 8);
}

```

```
TMR0H = data;
TMR0L = tmr_val;
T0CONbits.TMR0ON = 1;                                // starts the timer

while (INTCONbits.TMR0IF == 0);

T0CONbits.TMR0ON = 0;                                // stops the timer
INTCONbits.TMR0IF = 0;
}
void tpo_us (unsigned short tmr_val)
{
    unsigned char data;

    T0CON = 0b00001000;                                // run in 16-bit timer, no prescaler
    tmr_val = 65535 - ( 12 * tmr_val);

    data = (tmr_val >> 8);

    TMR0H = data;
    TMR0L = tmr_val;
    T0CONbits.TMR0ON = 1;                                // starts the timer

    while (INTCONbits.TMR0IF == 0);

    T0CONbits.TMR0ON = 0;                                // stops the timer
    INTCONbits.TMR0IF = 0;
}
```


Annexe 3 : rap_cyc.c

```

#include <p18cxxx.h>
#include "lcd_4b.h"

extern unsigned char Rap_cyc;

// ADC initialization
void adc_init (void)
{
    ADCON1 = 0x0E;           // AN0 : analog input, Vref+ = Vdd, Vref- = Vss
    ADCON0 = 0x01;           // An0 selected, enables ADC module
    ADCON2 = 0x80;           // result right justified
    TRISAbits.TRISA0 = 1;
}

unsigned int adc_convert (void)
{
    tpo_ms (5);
    ADCON0bits.GO = 1;       // starts conversion
    while(ADCON0bits.GO);
    return (ADRES);
}

void init_aff (void)
{
    unsigned char str1 [18] = "Rapport cyclique";
    lcd_com(0x80);
    lcd_str (str1);
    lcd_com(0xC0);
    lcd_car(0xE0);
    lcd_com(0xC2);
    lcd_car('=');
    lcd_com(0xC7);
    lcd_car('%');
}

void aff_rap_cyc (unsigned char Rap_cyc)
{
    lcd_com (0xC4);
    lcd_car (' ');
    lcd_car (' ');
    lcd_car (' ');

    if(Rap_cyc==0){
        lcd_com (0xC6);
        lcd_car ('0');
    }else
    if(Rap_cyc==20){
        lcd_com (0xC5);
        lcd_car ('2');
        lcd_car ('0');
    }else
    if(Rap_cyc==40){

```

```

        lcd_com (0xC5);
        lcd_car ('4');
        lcd_car ('0');
    }else
    if(Rap_cyc==60){
        lcd_com (0xC5);
        lcd_car ('6');
        lcd_car ('0');
    }else
    if(Rap_cyc==80){
        lcd_com (0xC5);
        lcd_car ('8');
        lcd_car ('0');
    }else
    if(Rap_cyc==100){
        lcd_com (0xC4);
        lcd_car ('1');
        lcd_car ('0');
        lcd_car ('0');
    }
}

void RapCyc (void)
{
    unsigned char evolution, i=0;
    unsigned int CAN_val;

    CAN_val = adc_convert ();

    if (CAN_val < 767)
    {
        if (Rap_cyc >= 50)
        {
            Rap_cyc = 0;
            evolution = 1;
        }
        else
        {
            Rap_cyc = 100;
            evolution = 0;
        }
        aff_rap_cyc (Rap_cyc);

        while (CAN_val < 767)
        {
            CAN_val = adc_convert ();
            tpo_ms (100);
            i++;

            if (i == 10)
            {
                i = 0;
                if (evolution)
                {
                    Rap_cyc += 20;
                    if (Rap_cyc == 100)
                        evolution = 0;
                }
            }
            else
            {

```

```
        Rap_cyc -= 20;
        if (Rap_cyc == 0)
            evolution = 1;
    }
    aff_rap_cyc (Rap_cyc);
}
}
```

Annexe 4**NPN switching transistors****2N2222; 2N2222A****CHARACTERISTICS** $T_J = 25\text{ }^{\circ}\text{C}$ unless otherwise specified.

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
I_{CBO}	collector cut-off current 2N2222	$I_E = 0; V_{CB} = 50\text{ V}$	–	10	nA
		$I_E = 0; V_{CB} = 50\text{ V}; T_{amb} = 150\text{ }^{\circ}\text{C}$	–	10	μA
I_{CBO}	collector cut-off current 2N2222A	$I_E = 0; V_{CB} = 60\text{ V}$	–	10	nA
		$I_E = 0; V_{CB} = 60\text{ V}; T_{amb} = 150\text{ }^{\circ}\text{C}$	–	10	μA
I_{EBO}	emitter cut-off current	$I_C = 0; V_{EB} = 3\text{ V}$	–	10	nA
h_{FE}	DC current gain	$I_C = 0.1\text{ mA}; V_{CE} = 10\text{ V}$	35	–	
		$I_C = 1\text{ mA}; V_{CE} = 10\text{ V}$	50	–	
		$I_C = 10\text{ mA}; V_{CE} = 10\text{ V}$	75	–	
		$I_C = 150\text{ mA}; V_{CE} = 1\text{ V}; \text{note 1}$	50	–	
		$I_C = 150\text{ mA}; V_{CE} = 10\text{ V}; \text{note 1}$	100	300	
h_{FE}	DC current gain 2N2222A	$I_C = 10\text{ mA}; V_{CE} = 10\text{ V}; T_{amb} = -55\text{ }^{\circ}\text{C}$	35	–	
h_{FE}	DC current gain 2N2222 2N2222A	$I_C = 500\text{ mA}; V_{CE} = 10\text{ V}; \text{note 1}$	30 40	– –	
V_{CEsat}	collector-emitter saturation voltage 2N2222	$I_C = 150\text{ mA}; I_B = 15\text{ mA}; \text{note 1}$	–	400	mV
		$I_C = 500\text{ mA}; I_B = 50\text{ mA}; \text{note 1}$	–	1.6	V
V_{CEsat}	collector-emitter saturation voltage 2N2222A	$I_C = 150\text{ mA}; I_B = 15\text{ mA}; \text{note 1}$	–	300	mV
		$I_C = 500\text{ mA}; I_B = 50\text{ mA}; \text{note 1}$	–	1	V
V_{BEsat}	base-emitter saturation voltage 2N2222	$I_C = 150\text{ mA}; I_B = 15\text{ mA}; \text{note 1}$	–	1.3	V
		$I_C = 500\text{ mA}; I_B = 50\text{ mA}; \text{note 1}$	–	2.6	V
V_{BEsat}	base-emitter saturation voltage 2N2222A	$I_C = 150\text{ mA}; I_B = 15\text{ mA}; \text{note 1}$	0.6	1.2	V
		$I_C = 500\text{ mA}; I_B = 50\text{ mA}; \text{note 1}$	–	2	V
C_c	collector capacitance	$I_E = I_B = 0; V_{CB} = 10\text{ V}; f = 1\text{ MHz}$	–	8	pF
C_e	emitter capacitance 2N2222A	$I_C = I_E = 0; V_{EB} = 500\text{ mV}; f = 1\text{ MHz}$	–	25	pF
f_T	transition frequency 2N2222 2N2222A	$I_C = 20\text{ mA}; V_{CE} = 20\text{ V}; f = 100\text{ MHz}$	250 300	– –	MHz MHz
F	noise figure 2N2222A	$I_C = 200\text{ }\mu\text{A}; V_{CE} = 5\text{ V}; R_S = 2\text{ k}\Omega;$ $f = 1\text{ kHz}; B = 200\text{ Hz}$	–	4	dB

Annexe 5**MOC3041 MOC3042 MOC3043****ELECTRICAL CHARACTERISTICS** ($T_A = 25^\circ\text{C}$ unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
INPUT LED					
Reverse Leakage Current ($V_R = 6\text{ V}$)	I_R	—	0.05	100	μA
Forward Voltage ($I_F = 30\text{ mA}$)	V_F	—	1.3	1.5	Volts
OUTPUT DETECTOR ($I_F = 0$ unless otherwise noted)					
Leakage with LED Off, Either Direction (Rated $V_{DRM}^{(1)}$)	I_{DRM1}	—	2	100	nA
Peak On-State Voltage, Either Direction ($I_{TM} = 100\text{ mA Peak}$)	V_{TM}	—	1.8	3	Volts
Critical Rate of Rise of Off-State Voltage ⁽³⁾	dv/dt	1000	2000	—	V/ μs
COUPLED					
LED Trigger Current, Current Required to Latch Output (Main Terminal Voltage = $3\text{ V}^{(2)}$)	I_{FT}	—	—	15 10 5	mA
Holding Current, Either Direction	I_H	—	250	—	μA
Isolation Voltage ($f = 60\text{ Hz}$, $t = 1\text{ sec}$)	V_{ISO}	7500	—	—	Vac(pk)
ZERO CROSSING					
Inhibit Voltage ($I_F = \text{Rated } I_{FT}$, MT1–MT2 Voltage above which device will not trigger.)	V_{IH}	—	5	20	Volts
Leakage in Inhibited State ($I_F = \text{Rated } I_{FT}$, Rated V_{DRM} , Off State)	I_{DRM2}	—	—	500	μA

1. Test voltage must be applied within dv/dt rating.
2. All devices are guaranteed to trigger at an I_F value less than or equal to max I_{FT} . Therefore, recommended operating I_F lies between I_{FT} (15 mA for MOC3041, 10 mA for MOC3042, 5 mA for MOC3043) and absolute max I_F (60 mA).
3. This is static dv/dt . See Figure 7 for test circuit. Commutating dv/dt is a function of the load-driving thyristor(s) only.

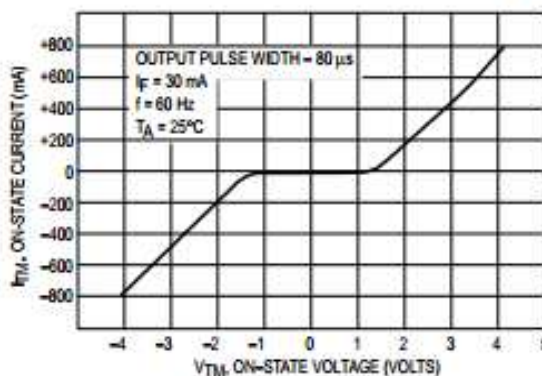
TYPICAL ELECTRICAL CHARACTERISTICS $T_A = 25^\circ\text{C}$ 

Figure 1. On-State Characteristics

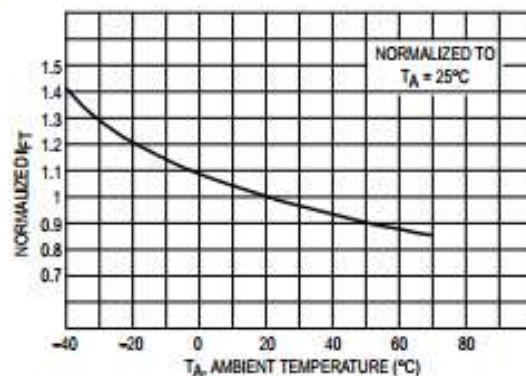


Figure 2. Trigger Current versus Temperature

Annexe 6**BTA/BTB12 and T12 Series****ELECTRICAL CHARACTERISTICS** ($T_j = 25^\circ\text{C}$, unless otherwise specified)**■ SNUBBERLESS™ and LOGIC LEVEL (3 Quadrants)**

Symbol	Test Conditions	Quadrant		T12	BTA/BTB12			Unit
				T1235	SW	CW	BW	
I_{GT} (1)	$V_D = 12\text{ V}$ $R_L = 30\ \Omega$	I - II - III	MAX.	35	10	35	50	mA
V_{GT}		I - II - III	MAX.	1.3				V
V_{GD}	$V_D = V_{DRM}$ $R_L = 3.3\text{ k}\Omega$ $T_j = 125^\circ\text{C}$	I - II - III	MIN.	0.2				V
I_H (2)	$I_T = 100\text{ mA}$		MAX.	35	15	35	50	mA
I_L	$I_G = 1.2\ I_{GT}$	I - III	MAX.	50	25	50	70	mA
		II		60	30	60	80	
dV/dt (2)	$V_D = 67\ \%V_{DRM}$ gate open $T_j = 125^\circ\text{C}$		MIN.	500	40	500	1000	V/ μs
$(dI/dt)_c$ (2)	$(dV/dt)_c = 0.1\text{ V}/\mu\text{s}$ $T_j = 125^\circ\text{C}$		MIN.	-	6.5	-	-	A/ms
	$(dV/dt)_c = 10\text{ V}/\mu\text{s}$ $T_j = 125^\circ\text{C}$			-	2.9	-	-	
	Without snubber $T_j = 125^\circ\text{C}$			6.5	-	6.5	12	

■ STANDARD (4 Quadrants)

Symbol	Test Conditions	Quadrant		BTA/BTB12		Unit
				C	B	
I_{GT} (1)	$V_D = 12\text{ V}$ $R_L = 30\ \Omega$	I - II - III IV	MAX.	25 50	50 100	mA
V_{GT}		ALL	MAX.	1.3		V
V_{GD}	$V_D = V_{DRM}$ $R_L = 3.3\text{ k}\Omega$ $T_j = 125^\circ\text{C}$	ALL	MIN.	0.2		V
I_H (2)	$I_T = 500\text{ mA}$		MAX.	25	50	mA
I_L	$I_G = 1.2\ I_{GT}$	I - III - IV	MAX.	40	50	mA
		II		80	100	
dV/dt (2)	$V_D = 67\ \%V_{DRM}$ gate open $T_j = 125^\circ\text{C}$		MIN.	200	400	V/ μs
$(dV/dt)_c$ (2)	$(dI/dt)_c = 5.3\text{ A/ms}$ $T_j = 125^\circ\text{C}$		MIN.	5	10	V/ μs

STATIC CHARACTERISTICS

Symbol	Test Conditions			Value	Unit
V_T (2)	$I_{TM} = 17\text{ A}$ $t_p = 380\text{ }\mu\text{s}$	$T_j = 25^\circ\text{C}$	MAX.	1.55	V
V_{to} (2)	Threshold voltage	$T_j = 125^\circ\text{C}$	MAX.	0.85	V
R_d (2)	Dynamic resistance	$T_j = 125^\circ\text{C}$	MAX.	35	m Ω
I_{DRM}	$V_{DRM} = V_{RRM}$	$T_j = 25^\circ\text{C}$	MAX.	5	μA
I_{RRM}		$T_j = 125^\circ\text{C}$		1	mA

Note 1: minimum I_{GT} is guaranteed at 5% of I_{GT} max.

Note 2: for both polarities of A2 referenced to A1

Annexe 7**PIC18F2455/2550/4455/4550****13.0 TIMER2 MODULE**

The Timer2 module timer incorporates the following features:

- 8-bit Timer and Period registers (TMR2 and PR2, respectively)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4 and 1:16)
- Software programmable postscaler (1:1 through 1:16)
- Interrupt on TMR2 to PR2 match
- Optional use as the shift clock for the MSSP module

The module is controlled through the T2CON register (Register 13-1) which enables or disables the timer and configures the prescaler and postscaler. Timer2 can be shut off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

A simplified block diagram of the module is shown in Figure 13-1.

13.1 Timer2 Operation

In normal operation, TMR2 is incremented from 00h on each clock ($F_{osc}/4$). A 2-bit counter/prescaler on the clock input gives direct input, divide-by-4 and divide-by-16 prescale options. These are selected by the prescaler control bits, T2CKPS1:T2CKPS0 (T2CON<1:0>). The value of TMR2 is compared to that of the Period register, PR2, on each clock cycle. When the two values match, the comparator generates a match signal as the timer output. This signal also resets the value of TMR2 to 00h on the next cycle and drives the output counter/postscaler (see Section 13.2 "Timer2 Interrupt").

The TMR2 and PR2 registers are both directly readable and writable. The TMR2 register is cleared on any device Reset, while the PR2 register initializes at FFh. Both the prescaler and postscaler counters are cleared on the following events:

- a write to the TMR2 register
- a write to the T2CON register
- any device Reset (Power-on Reset, MCLR Reset, Watchdog Timer Reset or Brown-out Reset)

TMR2 is not cleared when T2CON is written.

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

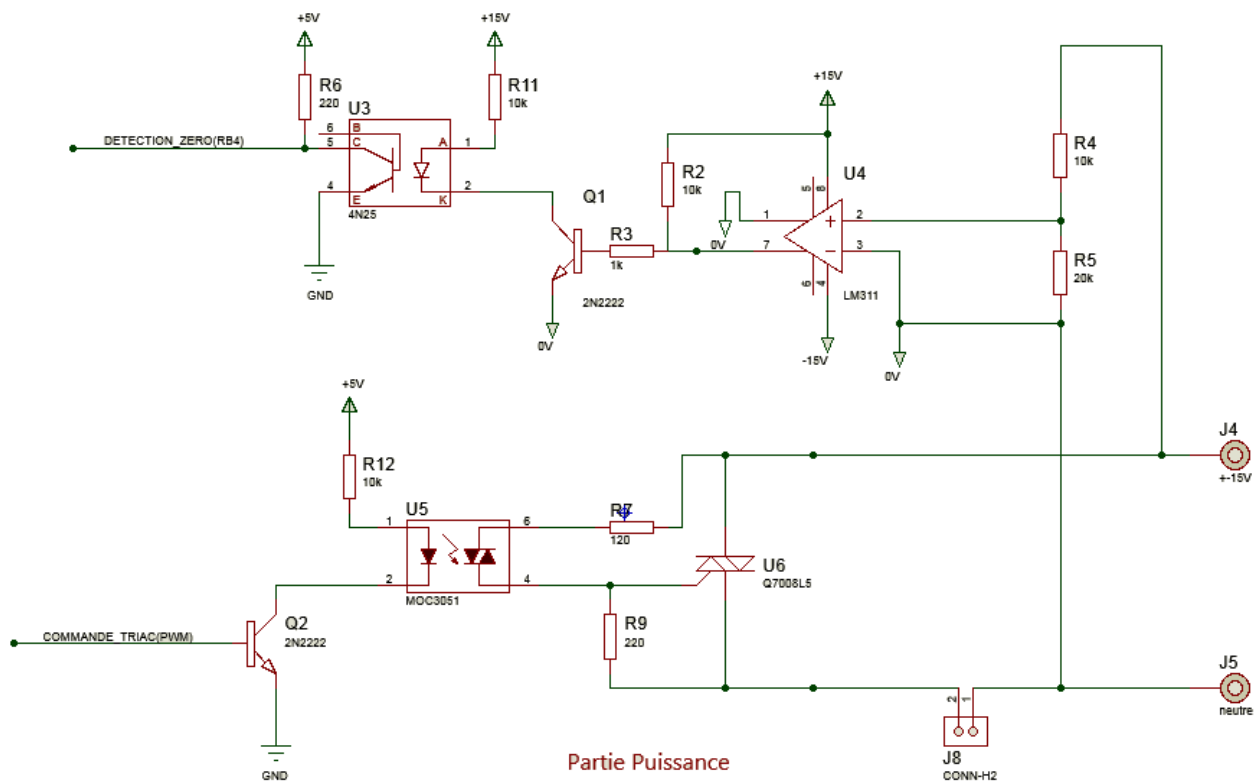
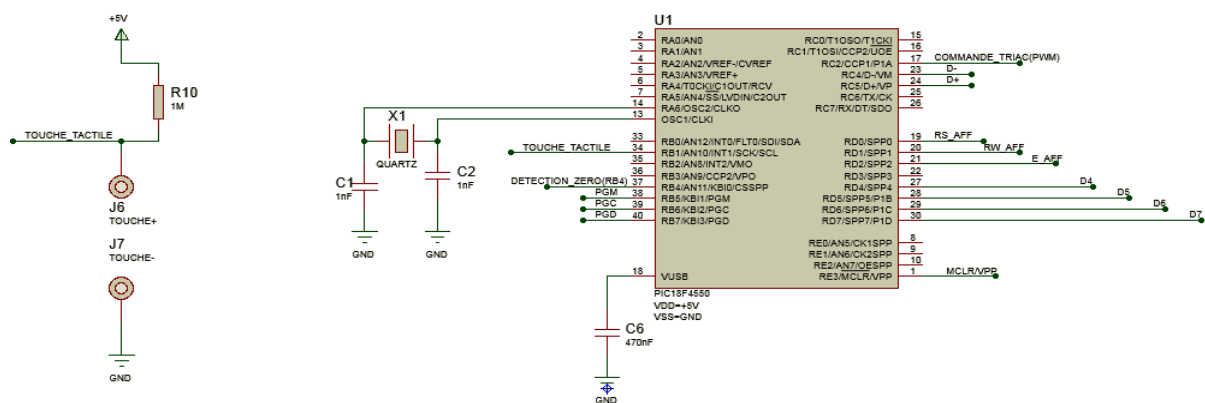
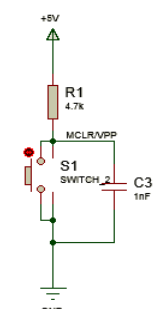
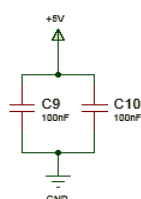
Legend:

R = Readable bit
-n = Value at POR

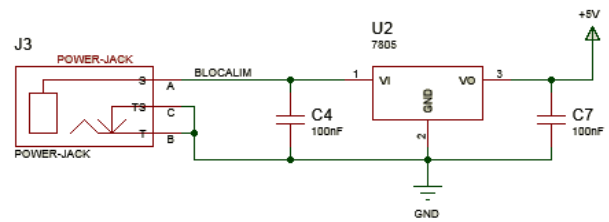
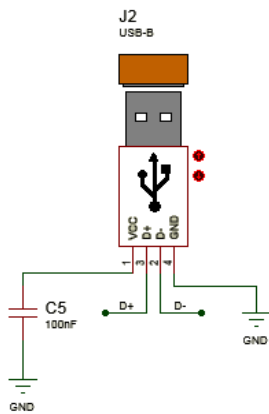
W = Writable bit
'1' = Bit is set

U = Unimplemented bit, read as '0'
'0' = Bit is cleared
x = Bit is unknown

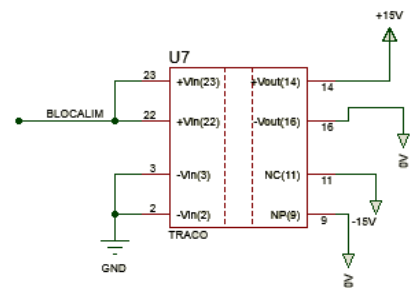
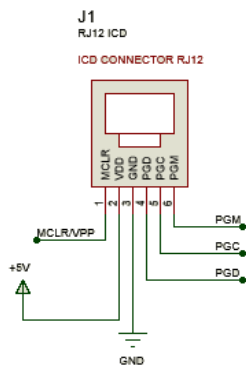
bit 7	Unimplemented: Read as '0'
bit 6-3	T2OUTPS3:T2OUTPS0: Timer2 Output Postscale Select bits 0000 = 1:1 Postscale 0001 = 1:2 Postscale • • • 1111 = 1:16 Postscale
bit 2	TMR2ON: Timer2 On bit 1 = Timer2 is on 0 = Timer2 is off
bit 1-0	T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits 00 = Prescaler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

Annexe 8 Schéma complet.**Commande Lampe & détection du zéro****Microcontrôleur****TOUCHE TACTILE (MICROPROCESSEUR)**

Alimentation et connecteurs



Connecteur et alimentation



Afficheur

