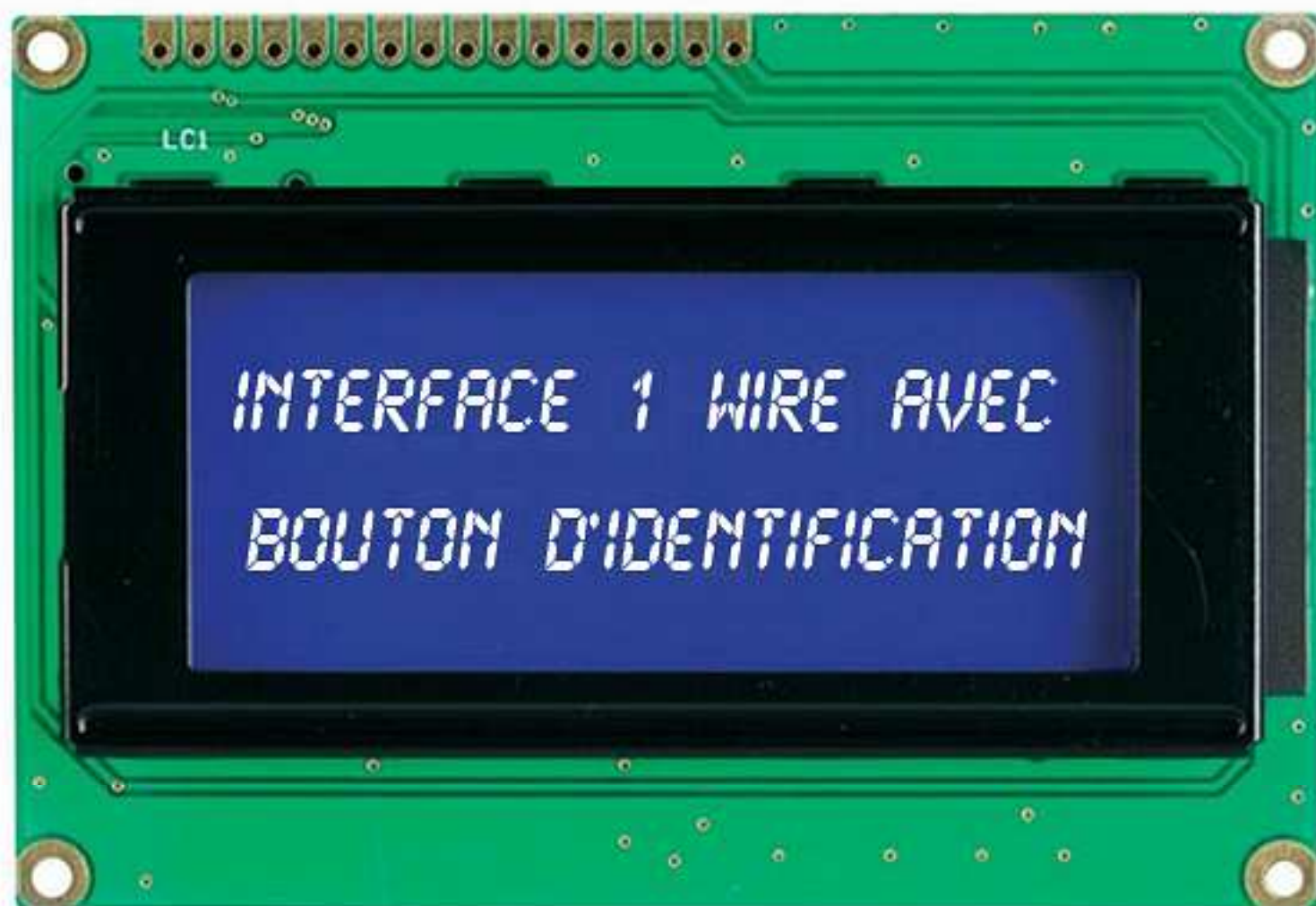


TP SYNTHÈSE
GE1



POLYTECH[®]
CLERMONT-FERRAND



MALRIC ALEXANDRE

LEYSSENE CLEMENT

PLANTIN ANTHONY

Table des matières

I. Introduction.....	2
II. Objectif du projet.....	2
III. Schéma fonctionnel.....	2
IV. Schéma structurel.....	3
V. Décomposition des fonctions principales :.....	4
1. FP1 traitement des données	4
2. FP2 alimentation du système :	5
3. FP3 : Capter numéro indentification	5
4. FP4 : Interface homme machine	6
5. FP5 : Commander organe de puissance	6
6. FP6 : Reprogrammation	7
VI. Timer du pic18F4550.....	8
Configuration des registres du timer0 :	8
Problèmes rencontrés :	9
VII. iButton.....	10
VIII. BUS 1WIRE.....	11
1. Pulse de reset.....	11
2. Emission d'un bit du maître vers l'esclave:	12
3. Réception d'un bit par le maître:	13
4. Envoie de la commande « Read Rom »	13
5. Détection d'une clé ibutton avec le protocole 1 wire.....	14
7. Calcul du CRC	15
IX. Algorithme du programme.....	15
Initialisation de l'afficheur	15
Affiche caractère (car)	15
Affiche commande (car).....	16
Pulse de reset	16
Lecture d'un bit sur le bus 1wire	17

Ecriture d'un bit sur le bus 1wire	17
Envoie d'un octet de données sur le bus	18
Lecture d'un octet sur le bus	18
Calcul du CRC (numero_de_serie)	18
Compare (numero_serie , T)	18
Programme principal	19
X. Tests	20
XI. Conclusion	21
XII. Tables des illustrations	22
Annexe Table d'aide au calcul du CRC	23
Annexe 2 : programme en C	24
1wire.h	24
1wire.c	24
Lcd_4B.h	28
Lcd_4B.c	29
Commande_triac.h	31
Commande_triac.c	31
Main.c	32

I. Introduction

Dans le cadre de notre formation, un projet de synthèse nous a été imposé, mettant en œuvre les compétences acquises durant l'année. Parmi les différents projets nous avons choisi l'interface 1 wire avec bouton d'identification. Le groupe de travail est composé de Malric Alexandre, Leyssene clément et Plantin Anthony.

II. Objectif du projet

Notre projet doit assurer la commande d'une charge par un Triac à partir d'un bouton d'identification 1 Wire. Le microcontrôleur devra être capable de lire le numéro de série inscrit dans un iButton en utilisant le protocole 1 Wire. Il affichera sur l'afficheur LCD le code d'identification lu. Si le code correspond à un code enregistré, un triac opto-isolé sera commandé afin d'allumer ou d'éteindre une charge.

III. Schéma fonctionnel

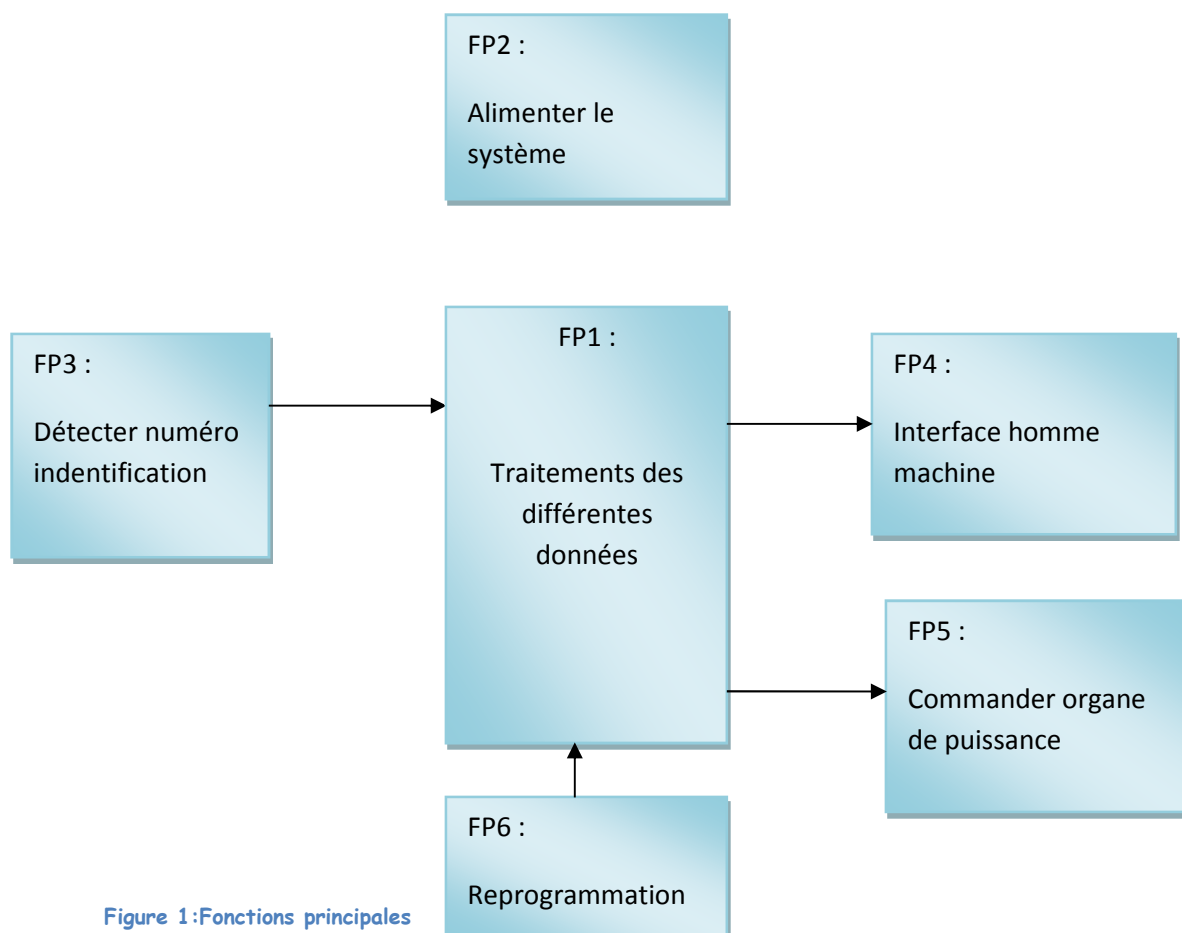
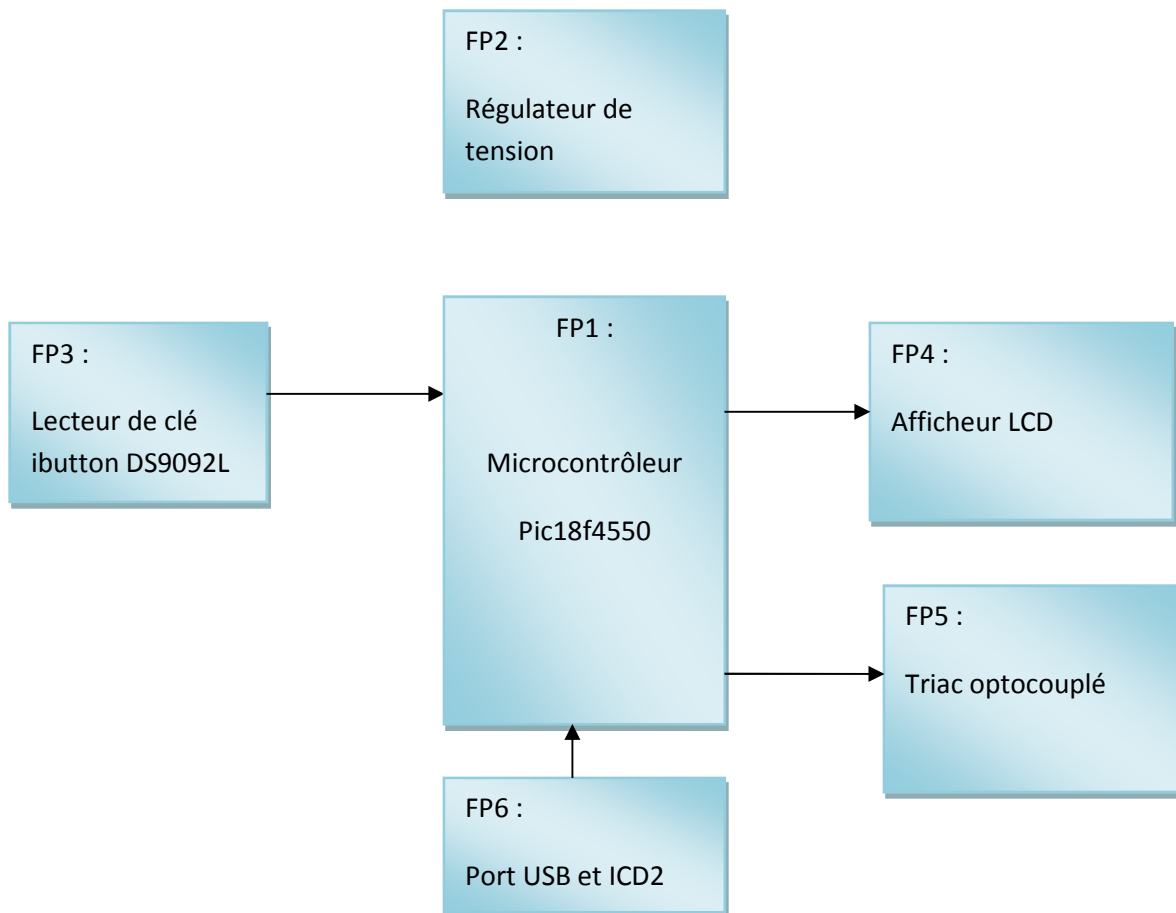


Figure 1: Fonctions principales

IV. Schéma structurel



V. Décomposition des fonctions principales :

1. FP1 traitement des données

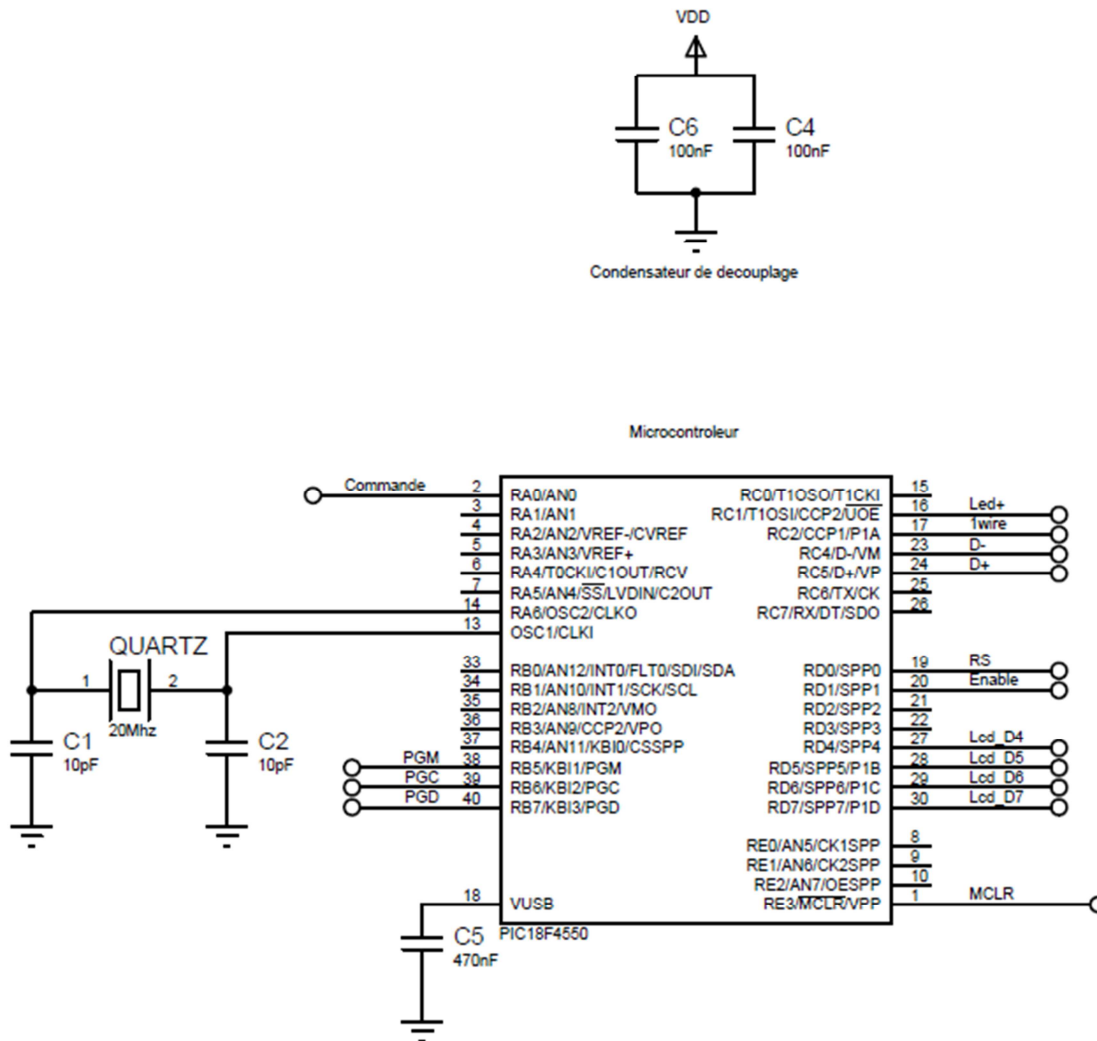


Figure 2: Schéma du pic 18f450

Le cahier des charges nous impose de travailler avec un pic18f450. On se sert du pic pour l'interface homme-machine, pour la commande du triac et pour l'acquisition du numéro d'identification.

On utilise des capacités de découplages pour limiter l'amplitude des perturbations et un quartz de 20Mhz pour cadencer le microcontrôleur à une fréquence de travail.

2. FP2 alimentation du système :

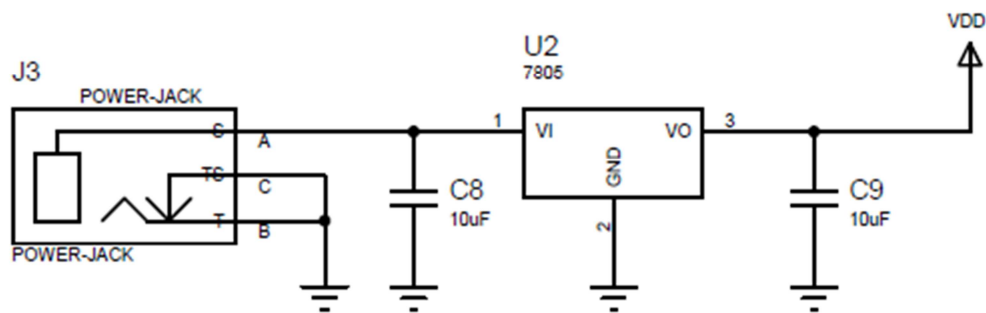


Figure 3: Schéma alimentation du système

On utilise un régulateur de tension (LM7805) pour avoir une tension fixe en sortie.

3. FP3 : Capter numéro indentation

Pour cette fonction technique on utilise le protocole one-wire de Dallas :

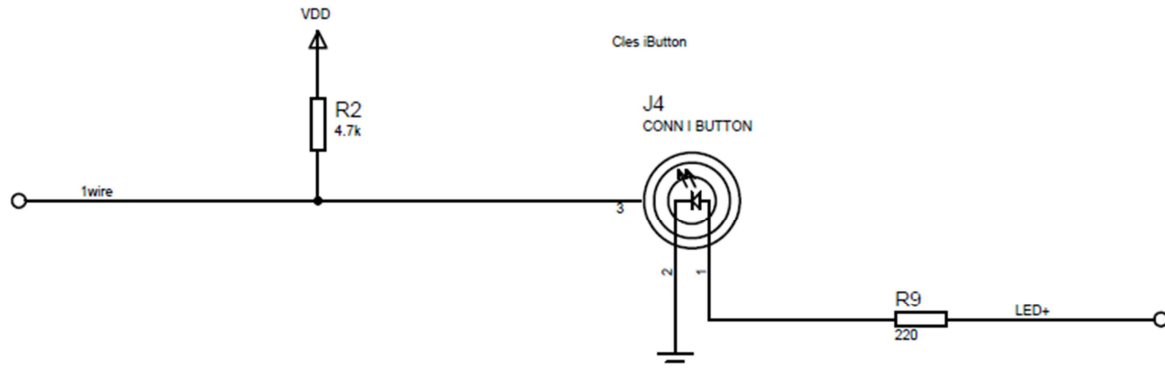


Figure 4: schéma protocole one-wire

Le protocole one-wire impose une résistance de pull up sur ce dernier. Pour la récupération du numéro d'indentification on utilise un détecteur de clé ibutton. Les données sont envoyées sur le microcontrôleur.

4. FP4 : Interface homme machine

Pour l'interface homme-machine on utilise un afficheur LCD avec son contrôleur Hitachi HD44780. On utilisera le LCD en mode 4 bits. Un potentiomètre est utilisé pour régler le contraste de l'afficheur. On envoie uniquement des données sur le LCD donc la pin R/W est mise à la masse. L'afficheur est placé sur le port D du microcontrôleur.

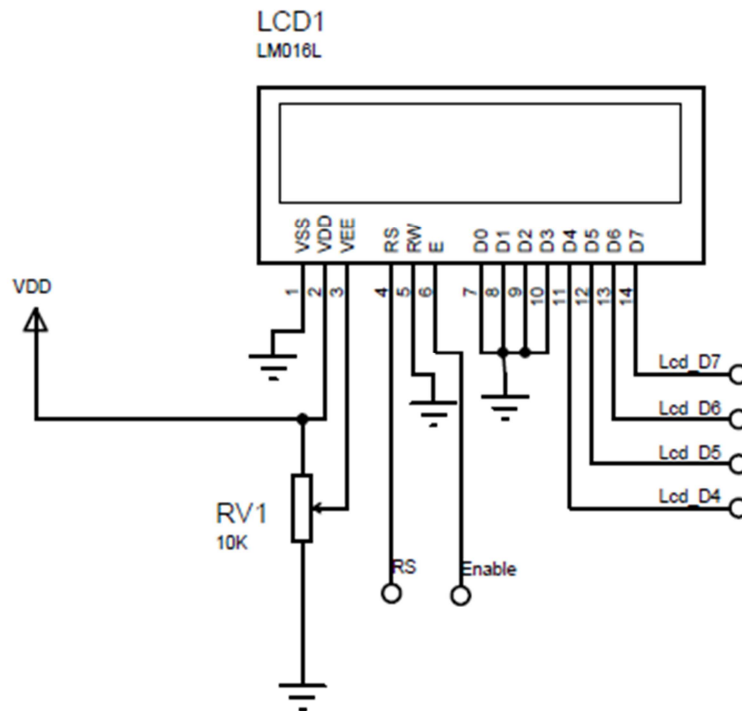


Figure 5: Schéma afficheur LCD

5. FP5 : Commander organe de puissance

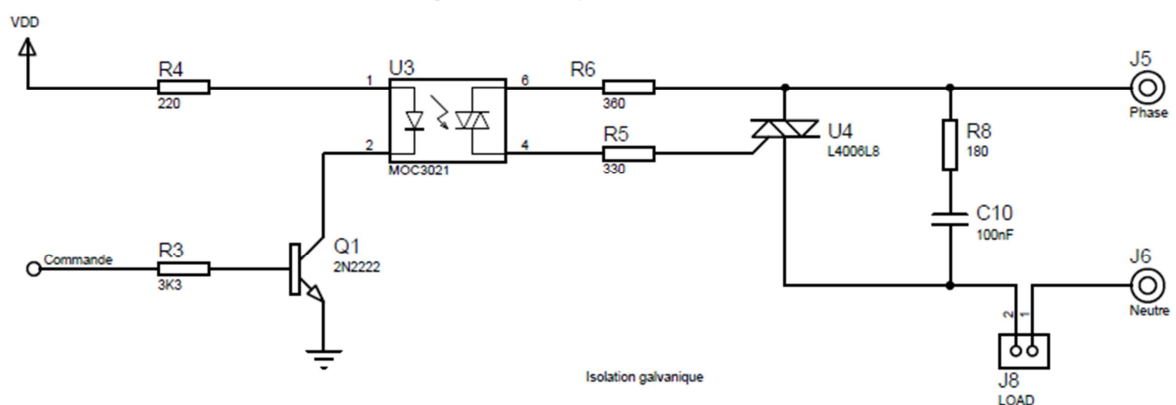


Figure 6: Schéma commande du triac

On utilise un triac (BTA/BTB12) pour commander une charge et un optocoupleur pour l'isolation galvanique.

On utilise une résistance de 3300Ω pour limiter un courant dans le transistor et pour qu'il soit saturé (il faut $I_{bmax} = 10mA$ et on a $I_b = 2mA$). On utilise la résistance $R4 = 220\Omega$ en entrée de la diode de l'optocoupleur pour protéger cette dernière.

La répartition du courant de gâchette du triac dans deux résistances permet de répartir la dissipation thermique occasionnée par les surintensités au déclenchement. Le snubber (circuit RC) limite le temps de montée de la tension appliquée au triac et contribue par la même à réduire la production de parasites sur le réseau d'alimentation. Lorsque on applique un '1' logique sur la base du transistor, celui-ci se sature et le courant circule dans la diode électroluminescente de l'optocoupleur. Deux photodiodes sont montées tête-bêche, elles deviennent passante grâce au photon émis par la LED et active la gâchette du triac. Celui-ci devient passant et commande la charge. Lorsque l'on ne commande plus le transistor, la diode électroluminescente ne s'allume plus les photodiodes ne sont plus passante le triac sera bloqué lorsque l'onde passera à 0.

6. FP6 : Reprogrammation

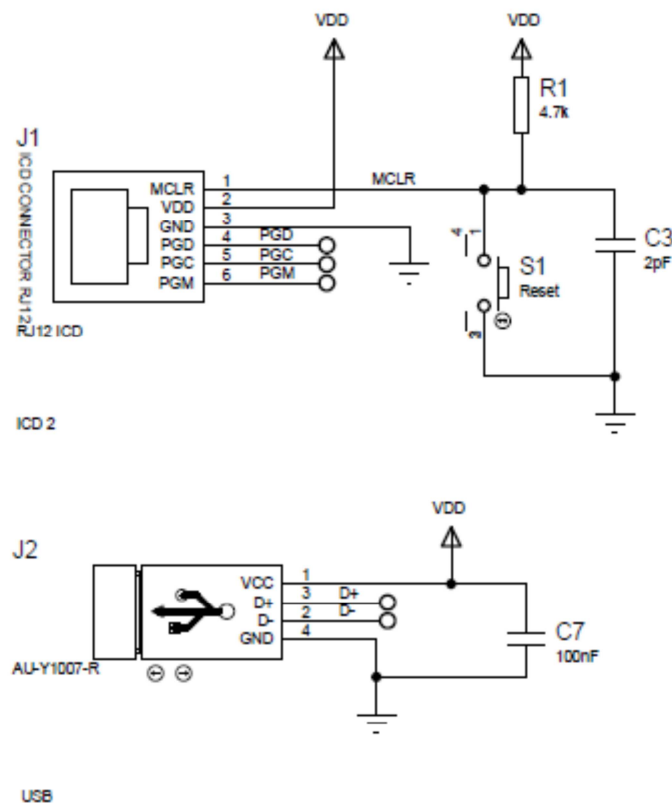


Figure 7: Schéma des connecteurs de programmation

Pour la reprogrammation du système on utilise un port ICD2 et un port USB.

VI. Timer du pic18F4550

Le microcontrôleur utilise un quartz de 20Mhz. Or on utilise un bootloader avec une connexion USB cadencée à 48 Mhz. On a utilisé le Timer 0 en mode 16 bits. Pour charger la valeur de comptage il faut charger séparément la partie basse dans TMR0L et la partie haute dans TMR0H.

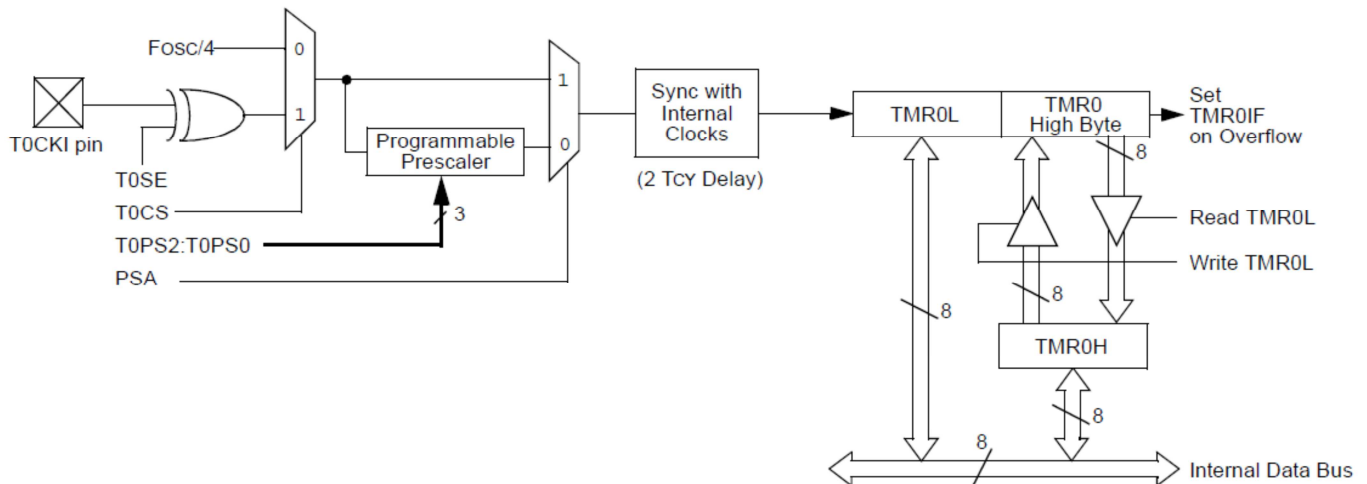


Figure 8: Schéma interne du Timer0

Configuration des registres du timer0 :

Nous avons pris le parti de faire des fonctions de temporisation variable selon l'argument pris pour des millisecondes et pour des microsecondes. Le timer 0 est utilisé en scrutation. Le registre TOcon est initialisé à la valeur 0x08 pour la temporisation de la microseconde. On charge ensuite la partie haute sur TMR0H puis ensuite la partie basse sur TMR0L. On teste ensuite le bit TMR0IF, pour une microseconde :

$$N = \frac{1 \mu}{83.3333n} = 12 \quad \text{Le timer des pics compte uniquement donc pour initialiser la valeur des registres il faut faire } 65535 - (12 * \text{tempo}).$$

Pour une temporisation d'une milliseconde :

$$N = \frac{1 ms}{83.3333n} = 12000$$

Or pour des temporisations supérieures à 5 ms il y a un dépassement de capacité. Il faut donc diviser la fréquence de l'horloge grâce à un prescaler. On initialise donc TOCON à 0x04 (Fréquence diviser par 32).

$$N = \frac{1 ms}{2.666666667 \mu s} = 375$$

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 (CLKI)
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

On divise F par 32 pour la
temporisation en millisecondes

Figure 9: Registre de contrôle du timer 0

Problèmes rencontrés :

Lors de la mise en œuvre du Timer nous avons rencontré deux problèmes majeurs :

Le premier était pour faire une temporisation variable selon l'argument pris. En effet la valeur ne se chargeait pas correctement dans les registres TMR0H et TMR0L. Après avoir lu la documentation, la valeur de TMR0H est mise à jour après une lecture de TMR0L. Il fallait simplement affecter le registre TMR0H avant TMR0L.

Le deuxième problème était sur la fréquence : en simulation, avec une fréquence de 20Mhz les temporisations étaient correctes. Mais une fois le programme chargé dans le microcontrôleur, les temps étaient différents. Le problème venait du réglage de la fréquence. En effet nous utilisons un bootloader avec une liaison usb. Pour cadencer la transmission usb, une fréquence d'horloge de 48 Mhz est nécessaire. L'horloge du quartz est divisée puis multipliée grâce à une PLL (Boucle à verrouillage de phase) pour obtenir cette fréquence de 48 Mhz. Cette fréquence est utilisée également pour les périphériques. Pour le timer 0 : $F = F_{osc}/4$.

VII. iButton

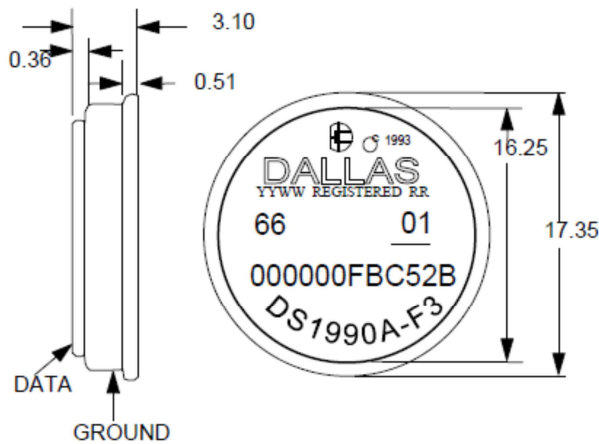


Figure 10 : iButton

Le DS1990A iButton est un support de données robuste qui agit comme une clé électronique avec un numéro d'identification automatique gravé au laser. Il comprend un numéro de série unique de 48 bits, un CRC de 8 bits et un code de famille de 8 bits (01h). Les données sont transférées via le protocole 1-Wire qui ne nécessite qu'une seule ligne de données et une mise à la masse. Le protocole 1-Wire impose une résistance de pull-up car les composants connectés ont une sortie avec un collecteur ouvert. La valeur de la résistance de pull-up devrait être d'environ 5 k Ω pour des longueurs de lignes courtes. Le bus 1-Wire a un débit de données maximal de 16.3 Kbits/seconde. Le iButton tire son énergie lors du contact avec le détecteur de clé.

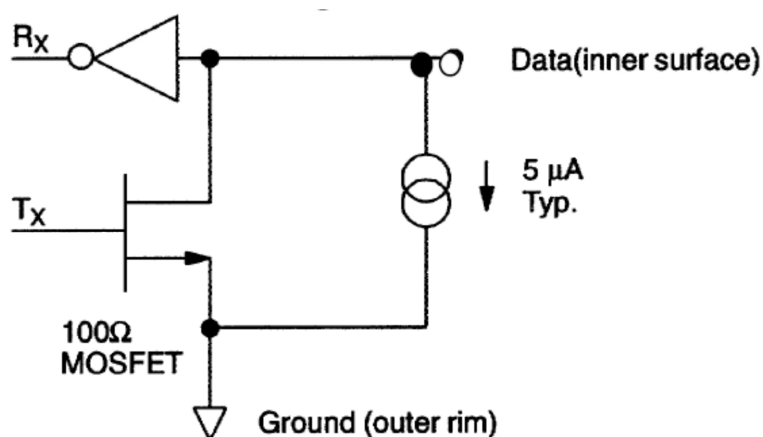


Figure 11: Schéma interne d'un ibutton

VIII. BUS 1WIRE

Le bus 1 WIRE de DALLAS, permet de connecter et de faire dialoguer entre eux des circuits sur un seul fil. Ce système de bus utilise un seul maître, qui pourra dialoguer avec un ou plusieurs esclaves. Toutes les commandes et données sont envoyées avec le bit LSB en tête. Le fil unique du bus doit être tiré au +Vcc par une résistance de 4,7K Ω . L'état repos du bus est donc un état haut.

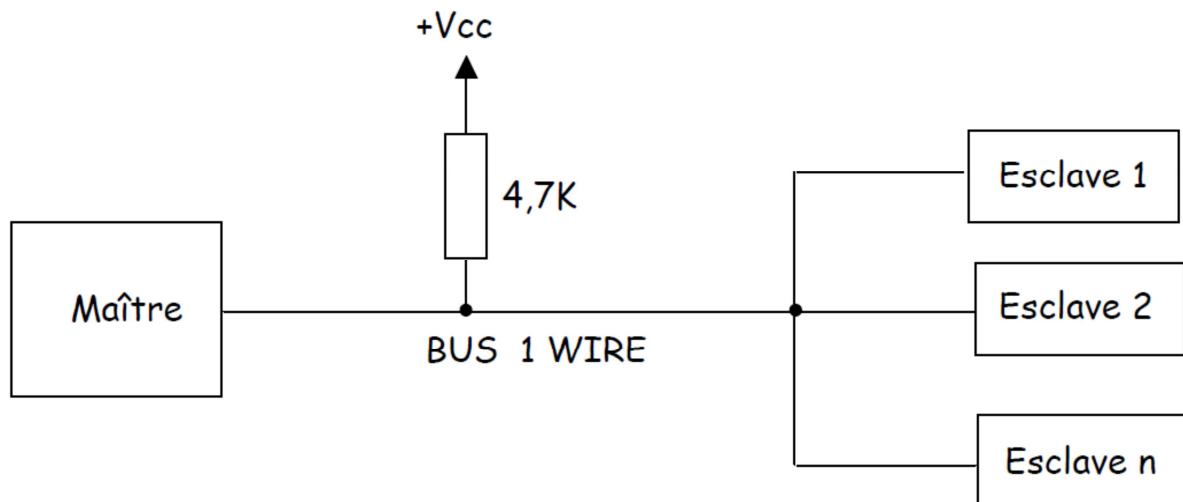
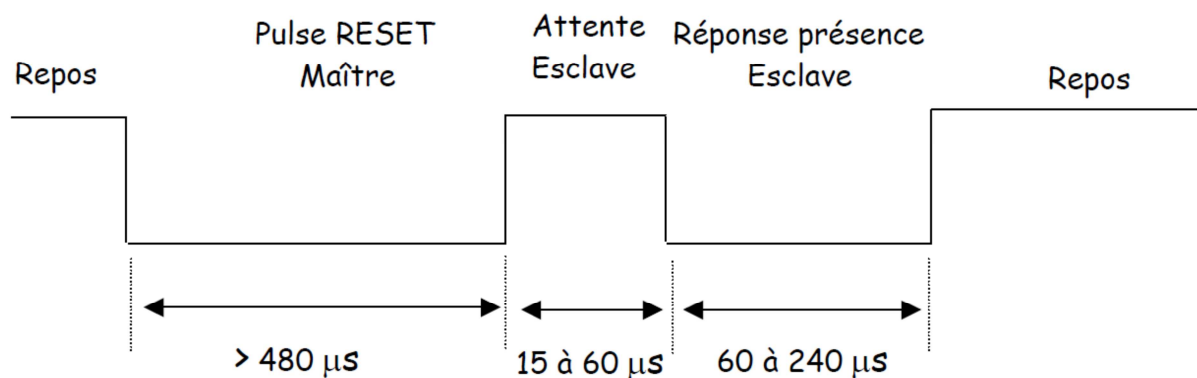


Figure 12 : Schéma de principe du bus 1wire

Dans notre projet nous n'avons qu'un seul esclave ce qui est plus simple à mettre en œuvre.

1. Pulse de reset

Si le bus est maintenu à l'état bas plus de 480 μ s par le maître, tous les composants sur le bus sont remis à zéro. C'est le pulse d'initialisation ou de Reset. Après un délai de 15 à 60 μ s, l'esclave raccordé, force le bus à l'état bas pendant 60 à 240 μ s pour signaler sa présence.



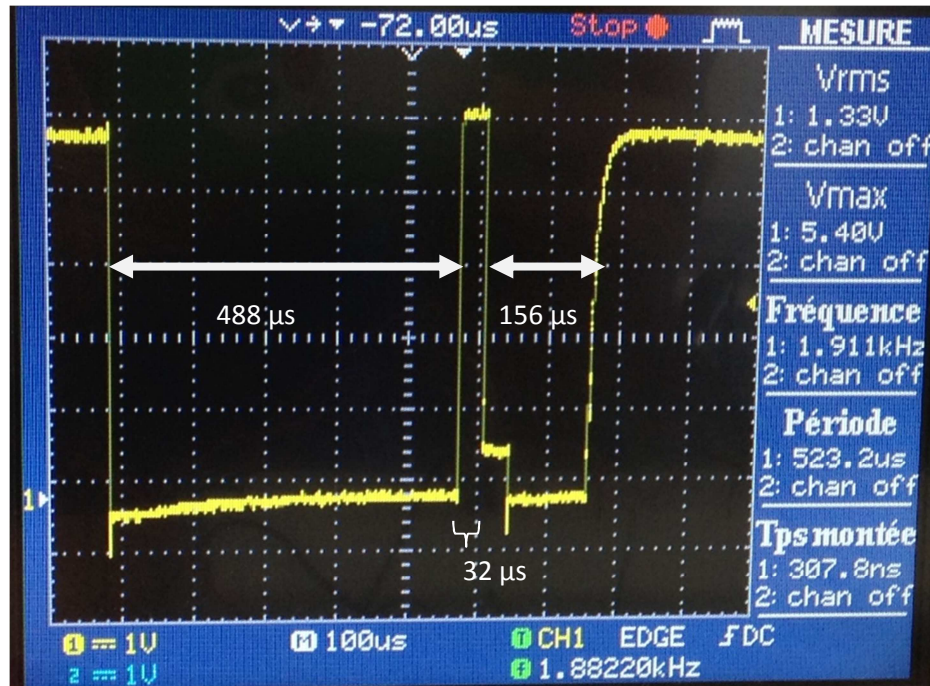
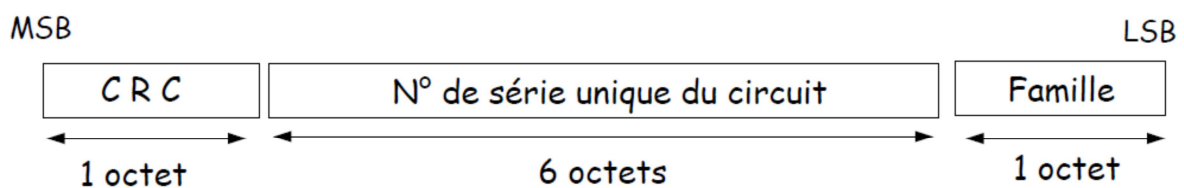


Figure 13: Chronogramme du pulse de reset

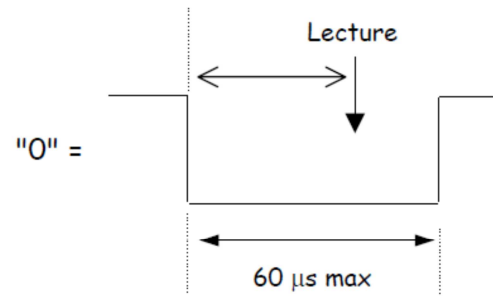
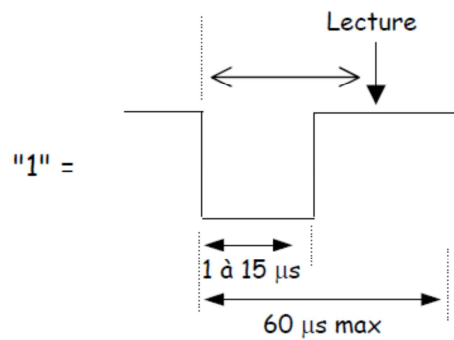
Pour le pulse de reset on respecte les temps d'établissements des niveaux logiques.

Chaque circuit possède une adresse physique unique, gravée dans la puce à la fabrication. Cette adresse est constituée de 64 bits soit 8 octets. Le premier octet détermine le type de famille auquel appartient le circuit. Les 6 octets suivants, constituent le code propre du circuit. Le dernier octet est le CRC. C'est un octet de contrôle calculé à partir des 56 bits précédents.



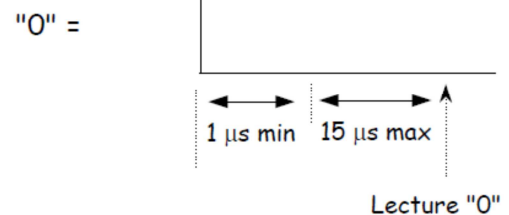
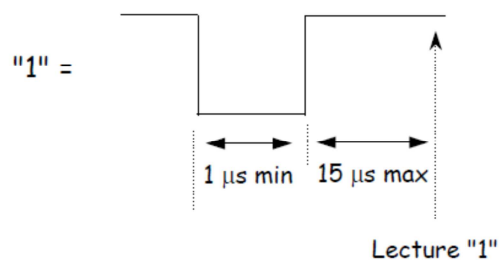
2. Emission d'un bit du maître vers l'esclave:

Le maître force le bus à "0" pendant 1 à 15 μs. L'esclave va lire le bus entre 15 et 45 μs après le front descendant (valeur typique 30 μs). Si on veut émettre un "1", il faut repasser le bus à "1" immédiatement, et ne plus rien faire jusqu'à $t = 60 \mu s$. Pour émettre un "0", il faut laisser le bus à "0" jusqu'à $t = 60 \mu s$, puis repasser le bus à "1". La durée du bit est donc de 60 μs, ce qui donne un débit de 16 kbits/sec.



3. Réception d'un bit par le maître:

Le maître force le bus à "0" pendant au moins 1 μ s. Si l'esclave veut émettre un "1", il laisse le bus libre donc tiré à "1". Pour émettre un "0", l'esclave doit tirer le bus à "0" pendant 15 μ s au minimum. Le maître devra donc dans tous les cas lire le bus 15 μ s maximum après avoir tiré le bus à "0" pendant 1 μ s. L'état du bus donnera alors le bit transmis par l'esclave.



4. Envoie de la commande « Read Rom »

Cette commande ne peut être utilisée que s'il n'y a qu'un seul esclave sur le bus. Celui-ci répond alors ses 64 bits de code.

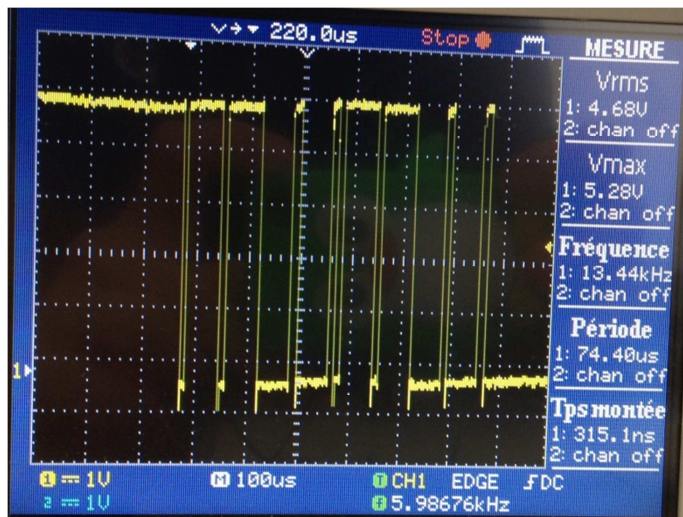


Figure 14:Envoie d'une commande READ ROM

Envoie de la commande read rom soit 33 en hexadécimal. Les bits de poids faibles sont envoyés en premier.

5. Détection d'une clé ibutton avec le protocole 1 wire.

Pour détecter une clé et avoir son numéro de série le maitre devra en permanence chercher un esclave sur le bus. Lorsque l'on aura détecté l'esclave on devra envoyer la commande read rom et lire ensuite les 64 octets de données qui correspondent au numéro de série. Le maitre pourra comparer ces 64 bits à ceux contenus dans une table, et commander ou non un organe de puissance.

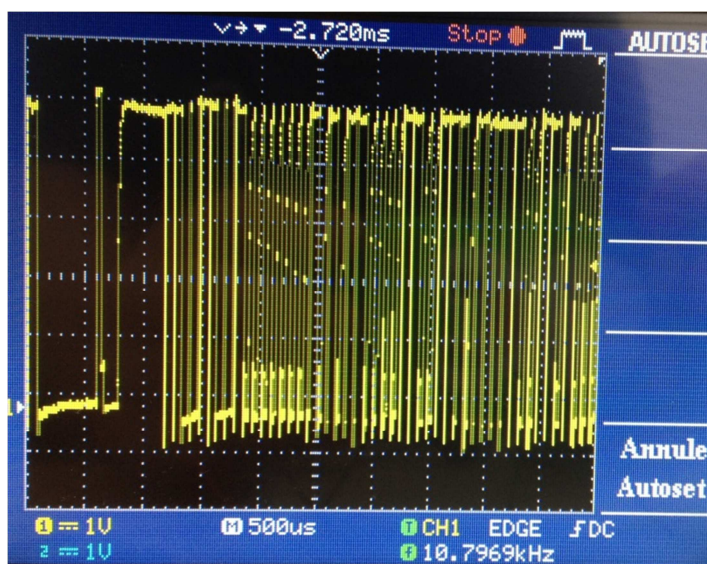


Figure 15:Reception du numéro de série

7. Calcul du CRC

Le CRC (Cyclic Redundancy Code) est un code permettant de vérifier l'intégrité d'un mot formé de plusieurs bits.

Quand on aura reçu les 56 premiers bits du circuit, soit 7 octets, on devra calculer l'octet de CRC pour le comparer à celui que l'on va recevoir avec les 8 bits restants. Pour éviter les calculs complexes du polynôme, on va utiliser une table indexée de 256 valeurs décimales. (Voir en annexe)

Lors de la compilation, la table d'index prenant trop de place en RAM, nous avons dû la mettre dans la mémoire flash du microcontrôleur.

IX. Algorithme du programme

Initialisation de l'afficheur

Début :

Tempo afficheur (5ms).
Affiche commande (0x38).
Tempo afficheur (15ms).

Affiche commande (0x38)
Tempo afficheur (100µs).
Affiche commande (0x38).
Tempo afficheur (100µs).
Affiche commande (0x06).
Affiche commande (0x0E).
Affiche commande (0x01).

Tempo afficheur (5ms).

Fin

Affiche caractère (car)

Variable

Caractère non signé data

Début :

tempo afficheur(100µs)

Mettre RS à 1

Mettre E à 0

Data <- ((car&0xf0)|(data&0x0f))

```
Mettre E à 1
tempo afficheur (5µs);
Mettre E à 0;

car <- car decaler de 4bits
data <- ((car&0xf0)|(data&0x0f))
tempo afficheur(5µs)
Mettre E à 1
Tempo afficheur(5µs)
Mettre E a 0
Tempo afficheur(5ms)
```

Fin

Affiche commande (car)

Début :

```
tempo afficheur(100µs)

Mettre RS à 0
Mettre E à 0
Data <- ((car&0xf0)|(data&0x0f))

Mettre E à 1
tempo afficheur (5µs);
Mettre E à 0;

car <- car decaler de 4bits
data <- ((car&0xf0)|(data&0x0f))
tempo afficheur(5µs)
Mettre E à 1
Tempo afficheur(5µs)
Mettre E à 0
Tempo afficheur(5ms)
```

Fin

Pulse de reset

Début:

```
One wire etat bas      // force le bus à l'état bas
Tempo 480 us
```

```
OW_etat_haut();          // force le bus à l'état haut et libère le bus
Tempo 70us

presence_esclave <- lecture one wire

tempo 500us
retourner presence_esclave    // 0 si il est présent et 1 si il est absent
```

Fin

Lecture d'un bit sur le bus 1wire

Début :

```
One wire en entrée
Si (one wire)
    Retourner 1
Fsi
Sinon
    Retourner 0
Fsinon
```

Fin

Ecriture d'un bit sur le bus 1wire

Début

```
Si (Bit)
{
    //Ecriture d'un 1
    One wire état bas
    tempo(10µs)
    One wire état haut
    tempo (50µs)
}
Fin Si
Sinon
{
    //Ecriture d'un 0
    One wire état bas
    tempo(60µs)
    One wire état haut
    tempo (10µs)
}
Fin sinon
}
```

FIN

Envoie d'un octet de données sur le bus

Début :

```
Pour i =0 à i= 8
    Ecriture du 1er bit sur le bus          // Envoie du premier bit
    data >>= 1;                            // décalage des données d'un bit
Fpour
```

Fin

Lecture d'un octet sur le bus

Début :

```
Pour i =0 à i< 8
    data >>= 1;
    réception du bit de poids faible sur le bus

Fpour
```

Fin

Calcul du CRC (numero_de_serie)

Début :

```
Pour i= 0 à i<7
    index_table <- crc ^ numero_de_serie [i]    // utilise la fonction XOR
    crc <- table_crc[index_table]                //on compare par rapport à une table
```

```
Retourner crc;
fin
```

Compare (numero_serie , T)

Début

```
Pour i=0 à i<8

    Si (numero_serie[i] != T[i] )
        ok=0;
    Fsi
Fpour
Retourner ok;
```

Fin

Programme principal

Début

```
Si ( detect_esclave)
    Faire
        Envoie d'un octet sur le bus (ReadRom)
        Reception de 8 octet (numero_serie)
        Calcule CRC (numero_serie)
        Tant que Calcule CRC (numero_serie) != numero_serie[8]

        Conversion_hexa_ascii(numero_serie)
        Affichage

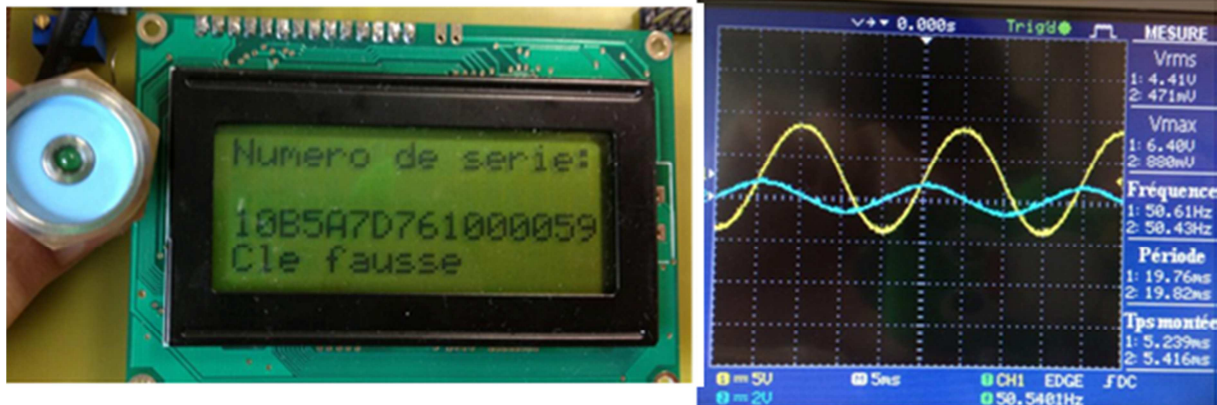
        Si (numero_serie=numero_identification)
            Triac=1                // on active la commande du triac
            Led=1                   // on active la diode
        FinSi
    FinSi
```

Fin

X. Tests

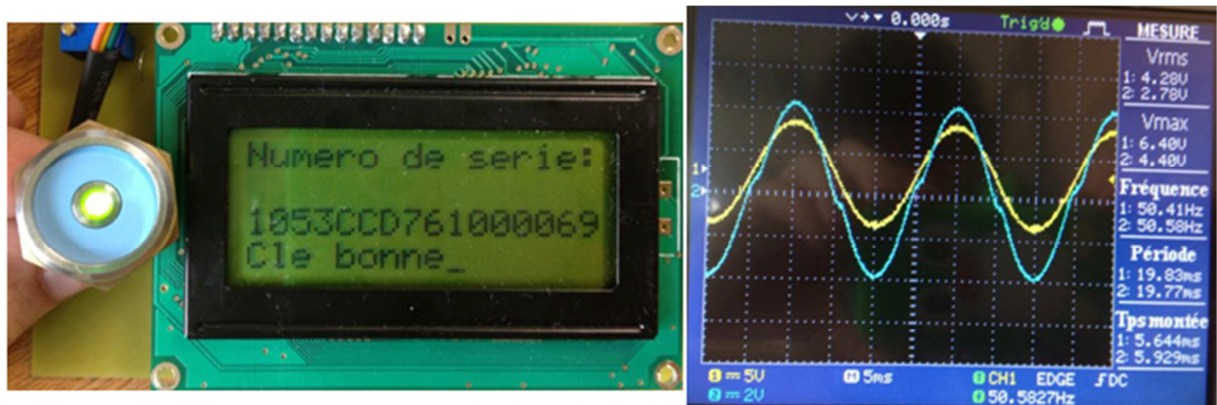
Le programme fini et les montages terminés, nous avons pu tester si le programme fonctionnait. Nous avons choisi 3 iButtons dont on veut reconnaître le numéro de série pour la commande du triac et deux autres dont on ne veut pas qu'ils commandent.

On affiche pour une clé fausse le numéro de série de la clé et « clé fausse ».



La voie 2 de l'oscilloscope (bleue) est à l'état bas, le triac n'est pas commandé.

Pour une des 3 clés bonnes on affiche à l'écran le numéro de série ainsi que « clé bonne ». On allume également la diode et on commande le triac :



XI. Conclusion

De manière général ce projet nous a permis de mettre en applications les différentes compétences acquises durant l'année comme la programmation de microcontrôleur mais aussi de l'électronique de puissance avec la commande d'un triac. Pendant ce projet nous étions en totale autonomie. Nous nous sommes confrontés à des problèmes ce qui nous a amené à chercher les solutions par nous-mêmes.

Nous avons réussi à faire fonctionner le iButton avec le protocole One-Wire et à commander le triac pour certaine valeur du numéro de série.

XII. Tables des illustrations

<i>Figure 1:Fonctions principales</i>	<i>2</i>
<i>Figure 2: Schéma du pic 18f450.....</i>	<i>4</i>
<i>Figure 3: Schéma alimentation du système.....</i>	<i>5</i>
<i>Figure 4: schéma protocole one-wire</i>	<i>5</i>
<i>Figure 5: Schéma afficheur LCD.....</i>	<i>6</i>
<i>Figure 6: Schéma commande du triac</i>	<i>6</i>
<i>Figure 7:Schéma des connecteurs de programmation</i>	<i>7</i>
<i>Figure 8:Schéma interne du Timer0.....</i>	<i>8</i>
<i>Figure 9 : iButton.....</i>	<i>10</i>
<i>Figure 10: Schéma interne d'un ibutton.....</i>	<i>10</i>
<i>Figure 11 : Schéma de principe du bus 1wire.....</i>	<i>11</i>
<i>Figure 12:Chronogramme du pulse de reset.....</i>	<i>12</i>
<i>Figure 13:Envoie d'une commande READ ROM.....</i>	<i>14</i>
<i>Figure 14:Reception du numéro de série</i>	<i>14</i>

Annexe Table d'aide au calcul du CRC

index																
0	0	94	188	226	97	63	221	131	194	156	126	32	163	253	31	65
16	157	195	33	127	252	162	64	30	95	1	227	189	62	96	130	220
32	35	125	159	193	66	28	254	160	225	191	93	3	128	222	60	98
48	190	224	4	92	223	129	99	61	124	34	192	158	29	67	161	255
64	70	24	250	164	39	121	155	197	132	218	56	102	229	187	89	7
80	219	133	103	57	186	228	6	88	25	71	165	251	120	38	196	154
96	101	59	217	135	4	90	184	230	167	249	27	69	198	152	122	36
112	248	166	68	26	153	199	37	123	58	100	134	216	91	5	231	185
128	140	210	48	110	237	179	81	15	78	16	242	172	47	113	147	205
144	17	79	173	243	112	46	204	146	211	141	111	49	178	236	14	80
160	175	241	19	77	206	144	114	44	109	51	209	143	12	82	176	238
176	50	108	142	208	83	13	239	177	240	174	76	18	145	207	45	115
192	202	148	118	40	171	245	23	73	8	86	180	234	105	55	213	139
208	87	9	235	181	54	104	138	212	149	203	41	119	244	170	72	22
224	233	183	85	11	136	214	52	106	43	117	151	201	74	20	246	168
240	116	42	200	150	21	75	169	247	182	232	10	84	215	137	107	53

Annexe 2 : programme en C

1wire.h

```
// Prototypes des fonctions

void tempo_us(unsigned short tempo);
void tempo_ms(unsigned short tempo);

void OW_etat_bas (void);
void OW_etat_haut (void);

unsigned char reset_pulse(void);

void OW_write_bit (unsigned char write_bit);
void OW_write_byte (unsigned char data);

unsigned char read_OW (void);
unsigned char OW_read_bit (void);
unsigned char OW_read_byte (void);

unsigned char detect_esclave(void);
// Bus 1wire
#define OW_WRITE_PIN          LATCbits.LATC1
#define OW_DIRECTION_PIN      TRISCbits.TRISC1
#define OW_READ_PIN           PORTCbits.RC1
```

1wire.c

```
/* ***** */
/*          Fonction d'attente          */
/* ***** */
void tempo_us(unsigned short tempo)
{
    unsigned char data;

    TOCON=0b00001000; // Initialisation en mode 16 bits sans prescaler
    tempo=65595 - (12*tempo);           // Fclck/4 = 12MHz soit 83.3333ns=>
                                         // 12*n = n microseconde

    data = tempo >> 8;

    TMROH=data;
    TMROL=(tempo);
    TOCONbits.TMROON = 1;                // Start the timer
    while(INTCONbits.TMROIF==0);
```

```

    TOCONbits.TMR0ON = 0;    // Stop le timer
    INTCONbits.TMR0IF=0;    //Remise a 0 du bit de debordement

}

/*****
/*          Fonction d'attente          */
*****/
void tempo_ms(unsigned short tempo)
{
    unsigned char data;

    TOCON=0b000000100;        // Initialisation en mode 16 bits avec F= f/32
    tempo=65535 - (375*tempo);
    data = tempo >> 8;

    TMROH=data;
    TMROL=(tempo);
    TOCONbits.TMR0ON = 1;    // Start the timer

    while(INTCONbits.TMR0IF==0);

    TOCONbits.TMR0ON = 0;    // Stop the timer
    INTCONbits.TMR0IF=0;    //Remise a 0 du bit de débordement

}

/*****
/*          Force le bus a l'etat bas    */
*****/
void OW_etat_bas(void)
{
    OW_DIRECTION_PIN=0;      //A0 en configurer en sortie
    OW_WRITE_PIN=0;          //Mise a 0 sur A0
}

/*****
/*          Force le bus a l'etat haut    */
*****/
void OW_etat_haut(void)
{
    OW_DIRECTION_PIN=0;      //A0 en configurer en sortie
    OW_WRITE_PIN=1;          //Mise a 1
}

```

```

/*****/
/*          Pulse de reset du bus 1wire          */
/*          et detection d'un esclave sur le bus  */
/*          Renvoie un 1 si il y a presence d'un esclave */
/*          Sinon renvoie un 0                      */
/*****/
unsigned char reset_pulse(void)
{
    unsigned char presence_esclave;

    OW_etat_bas();
    tempo_us(tempo_480us);

    OW_etat_haut();
    tempo_us(tempo_70us);

    presence_esclave = read_OW();

    tempo_us(500);
    return presence_esclave;
}

/*****/
/*          Lecture du bus 1wire          */
/*****/
unsigned char read_OW (void)
{
    unsigned char read_data=0;

    OW_DIRECTION_PIN=1;          //C0 en configurer en entree

    if (OW_READ_PIN==1 )
        read_data = 1;
    else
        read_data = 0;

    return read_data;
}

/*****/
/*          Ecriture d'un bit sur le bus          */
/*          Prend en argument le bit a envoyer    */
/*****/
void OW_write_bit (unsigned char write_bit)
{
    if (write_bit)
    {
        //Ecriture d'un 1

```

```

        OW_etat_bas();
        tempo_us(tempo_6us);          //10us
        OW_etat_haut();
        tempo_us(56);                  // attente 50us
    }
    else
    {
        //Ecriture d'un 0
        OW_etat_bas();
        tempo_us(56);                  // 60us
        OW_etat_haut();
        tempo_us(tempo_6us);
    }
}

/*****
/*          Lecture d'un bit sur le bus          */
/*          Retourne la valeur lue              */
*****/
unsigned char OW_read_bit (void)
{
    unsigned char read_data;
    // Lecture d'un bit
    unsigned char i;

    OW_etat_bas();

    tempo_us(tempo_6us);
    OW_etat_haut();

    tempo_us(6);
    read_data = read_OW();

    tempo_us(tempo_55us);
    return read_data;
}

/*****
/*          Envoie d'un octet de donnees sur le bus          */
/*          Retourne la valeur lue              */
*****/
void OW_write_byte (unsigned char data)
{
    unsigned char i;
    unsigned char masque = 0x01 ;

    for (i = 0; i < 8; i++)
    {
        OW_write_bit(data & masque);          //Envoie du premier bit

```

```

        data >>= 1;                                //decalage des donnees d'un bit
    }
}

```

```

/*****
/*          Renvoie 1 si un esclave est detectee          */
/*          Renvoie 0 si il n'y a aucun esclave          */
*****/
unsigned char detect_esclave(void)
{
    if (!reset_pulse())
        return 1 ;
    else
        return 0;
}

```

```

/*****
/*          Lecture d'un octet sur le bus                  */
/*          Renvoie la valeur de l'octet                  */
*****/
unsigned char OW_read_byte (void)
{
    unsigned char i, data=0;

    for (i= 0; i< 8; i++)
    {

        data >>= 1;
        if (OW_read_bit())
            data = data|0x80;
    }
    return data;
}

```

Lcd_4B.h

```

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

```

// Prototypes des fonction de l'afficheur en mode 4 bits

```
void tpo_ms(unsigned short duree);
```

```
void tpo_us(unsigned short duree);

void init_port_lcd(void);

void lcd_car(unsigned char car);

void lcd_cmde(unsigned char commande);

void lcd_cmde_init(unsigned char init);

void lcd_init(void);

void lcd_str(unsigned char *str);
```

Lcd_4B.c

```
//Fonctions de l'afficheur lcd
#include<p18f4550.h>
#include"1wire.h"

void init_port_lcd(void)
{
    TRISD=0;
    LATD=0;
}

void lcd_car(unsigned char car)
{
    tempo_us(100);
    RS=1;

    E=0;
    data= ((car&0xf0)|(data&0x0f));

    E=1;
    tempo_us(5);
    E=0;

    car=car<<4;
    data= ((car&0xf0)|(data&0x0f));
    tempo_us(5);
    E=1;
    tempo_us(5);
    E=0;
```

```
        tempo_ms(5);
    }

void lcd_cmde(unsigned char commande)
{
    tempo_us(100);
    RS=0;

    E=0;
    data= ((commande&0xf0)|(data&0x0f));
    tempo_us(5);
    E=1;
    tempo_us(5);
    E=0;

    commande=commande<<4;
    data= ((commande&0xf0)|(data&0x0f));
    tempo_us(5);
    E=1;
    tempo_us(5);
    E=0;
    tempo_ms(5);
}

void lcd_cmde_init(unsigned char init)
{
    tempo_us(100);
    RS=0;

    E=0;
    data= ((init&0xf0)|(data&0x0f));
    tempo_us(5);
    E=1;
    tempo_us(5);
    E=0;
}

void lcd_init(void)
{
    tempo_ms(15);
    lcd_cmde_init(0x30);
    tempo_ms(5);
    lcd_cmde_init(0x30);
    tempo_ms(100);
    lcd_cmde_init(0x30);
    tempo_ms(100);

    lcd_cmde_init(0x20); //Function set
}
```



```
    lcd_cmde(0x28);           //Display On
    lcd_cmde(0x0E);
    lcd_cmde(0x06);           //Display clear
    lcd_cmde(0x01);           //Entry mode set

    tempo_ms(5);

}

void lcd_str(unsigned char *str)
{
    unsigned char i=0;

    while(*str)
    {
        lcd_car(*str);
        str++;
    }
}
```

Commande_triac.h

```
unsigned char compare(unsigned char * numero_serie , unsigned char * T);

#define Commande_triac          LATAbits.LATA0
#define Commande_triac_direction TRISAbits.TRISA0
#define diode_allume            LATCbits.LATC2
#define diode_allume_dir        TRISCbits.TRISC2
```

Commande_triac.c

```
unsigned char compare(unsigned char * numero_serie , unsigned char * T)
{
    unsigned char ok=1;
    int i;
    for(i=0; i<8; i++)
    {
        if(numero_serie[i] != T[i] )
        {
            ok=0;
        }
    }
    return ok; }
```

Main.c

```

/** INCLUDE **/
#include<p18f4550.h>
#include "bootloader.h"
#include "1wire.h"
#include "lcd_4b.h"
#include "commande_triac.h"

/** INTERRUPT **/
#pragma udata
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode(){}
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode(){}

/** DEFINE **/
#define READ_ROM 0x33

/** PROTOTYPE **/
void hex2ascii(unsigned char ascii);
unsigned char crc_calcul(unsigned char *serie);

/** VARIABLES **/
rom unsigned char table_crc[] = { 0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83, 0xc2, 0x9c,
0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41, 0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e, 0x5f, 0x01,
0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc, 0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0, 0xe1,
0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62, 0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63,
0x3d, 0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff, 0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79,
0x9b, 0xc5, 0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07, 0xdb, 0x85, 0x67, 0x39, 0xba,
0xe4, 0x06, 0x58, 0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a, 0x65, 0x3b, 0xd9, 0x87,
0x04, 0x5a, 0xb8, 0xe6, 0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24, 0xf8, 0xa6, 0x44,
0x1a, 0x99, 0xc7, 0x25, 0x7b, 0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9, 0x8c, 0xd2,
0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f, 0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd, 0x11, 0x4f,
0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92, 0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50, 0xaf,
0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c, 0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1, 0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d,
0x73, 0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49, 0x08, 0x56, 0xb4, 0xea, 0x69, 0x37,
0xd5, 0x8b, 0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4, 0x95, 0xcb, 0x29, 0x77, 0xf4,
0xaa, 0x48, 0x16, 0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a, 0x2b, 0x75, 0x97, 0xc9,
0x4a, 0x14, 0xf6, 0xa8, 0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7, 0xb6, 0xe8, 0x0a,
0x54, 0xd7, 0x89, 0x6b, 0x35, };

unsigned char modele_numero_serie1[8]={0x01,0x85,0x57,0x7D,0x16,0x00,0x00,0x56};
unsigned char modele_numero_serie2[8]={0x01,0x54,0xB4,0x7D,0x16,0x00,0x00,0x31};
unsigned char modele_numero_serie3[8]={0x01,0x35,0xCC,0x7D,0x16,0x00,0x00,0x96};

```

```
void main(void)
{
    unsigned char i =0;
    unsigned char message[]="Attente ibutton";
    unsigned char serie[]="Numero de serie:";
    unsigned char cle_fausse[]="          Cle fausse";
    unsigned char cle_ok[]="          Cle bonne";
    unsigned char numero_serie[8];
    unsigned char c1, c2, c3;
    unsigned char crc=0;

    init_port_lcd();
    lcd_init();
    lcd_str(message);
    Commande_triac_direction =0;
    diode_allume_dir = 0;

    while(1)
    {

        if(detect_esclave()){
            do
            {

                OW_write_byte(READ_ROM);

                for(i = 0;i<8; i++)
                    numero_serie[i] = OW_read_byte();

                crc=crc_calcul(numero_serie);

            }while(crc!=numero_serie[7]);

            tempo_us(5);
            lcd_cmde(0x01);
            tempo_us(5);

            lcd_str(serie);

            for(i=0;i<8;i++)
                hex2ascii(numero_serie[i]);

            c1=compare(numero_serie,modele_numero_serie1);
            c2=compare(numero_serie,modele_numero_serie2);
            c3=compare(numero_serie,modele_numero_serie3);

            if ((c1 ==1)|| (c2 ==1 )|| (c3 == 1 ))
```

```
        {
            lcd_str(cle_ok);
            Commande_triac=1;
            diode_allume = 1;
        }
        else
        {
            lcd_str(cle_fausse);
            Commande_triac=0;
            diode_allume = 0;
        }
    }
}

void hex2ascii(unsigned char ascii)                // conversion hexadecimal vers ascii
                                                    // et affichage
{
    unsigned char temp;

    temp = ((ascii & 0x0F));

    if (temp <= 0x09)
        lcd_car(temp+'0');
    else
        lcd_car(temp+'0'+0x07);

    temp=( ascii & 0xF0)>>4;
    if (temp <= 0x09)
        lcd_car(temp+'0');
    else
        lcd_car(temp+'0'+0x07);
}

unsigned char crc_calcul(unsigned char *serie)
{
    unsigned char crc = 0;
    unsigned char i=0;
    unsigned char index_table=0;

    for(i=0; i<7;i++)
    {
        index_table=(crc^serie[i]);
        crc= table_crc[index_table];
    }

    return crc;
}
```