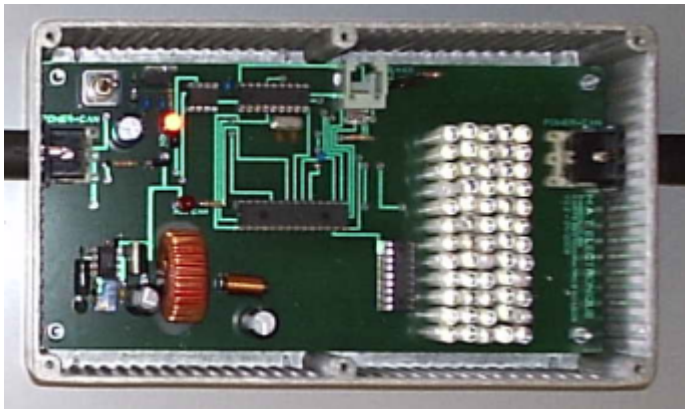
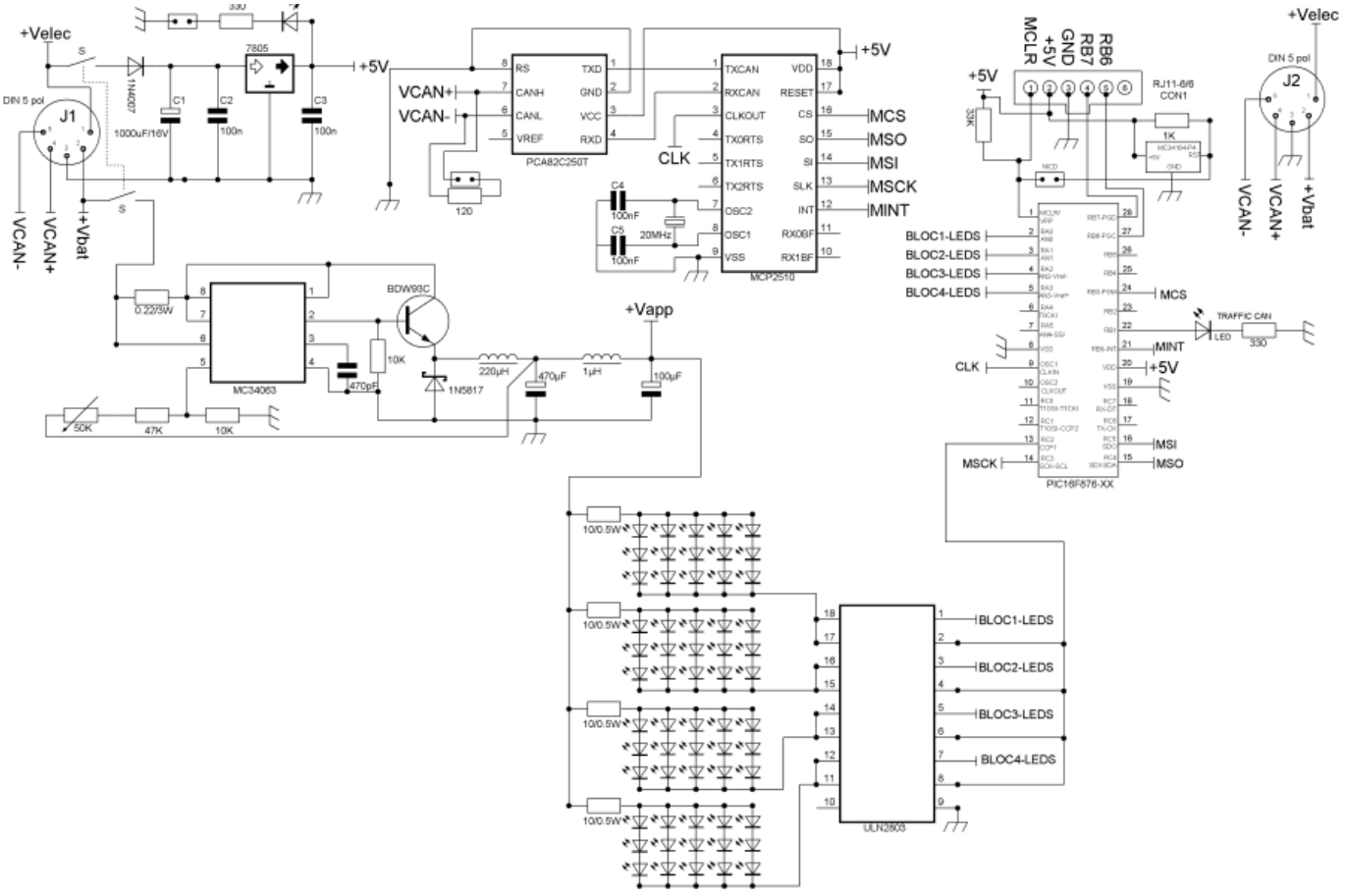
Présentation du module éclairage.

Le module éclairage permet la commande d'une matrice de 64 LEDS. Dans le système étudié, ce pavé de LEDS est destiné à servir de source d'éclairage dont on peut moduler la luminosité. Cette source est destinée à servir d'éclairage dans un bloc de secours. D'autres applications utilisent la même structure matérielle afin d'alimenter des signaux lumineux de type messages d'alertes ; ces messages, dont l'importance est liée à la sécurité, nécessitent d'avoir une alimentation ininterrompue. Dans ce dernier cas, les messages sont obtenus par mise en place d'un cache ajouré devant les blocs de LEDS regroupées par 15 (bloc 1 à 4 pour notre matrice).

Le module d'éclairage est commandé par le module de contrôle au travers du bus CAN.



On donne le schéma structurel de ce module :

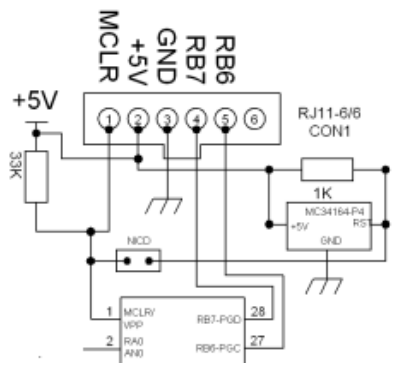


Gestion de la carte.

Le pilotage de la carte est confié à un microcontrôleur PIC16F876. La programmation in situ du PIC est assurée depuis le connecteur RJ11. Elle utilise les broches RB6 et RB7 du PIC. La broche MCLR permet, en mode programmation d'appliquer la tension de programmation.

Présentation du module éclairage.

L'horloge du processeur est fournie par la sortie SLK du circuit MCP2510 qui fournit un signal de fréquence 2,5 MHz (la fréquence du quartz divisée par 8).



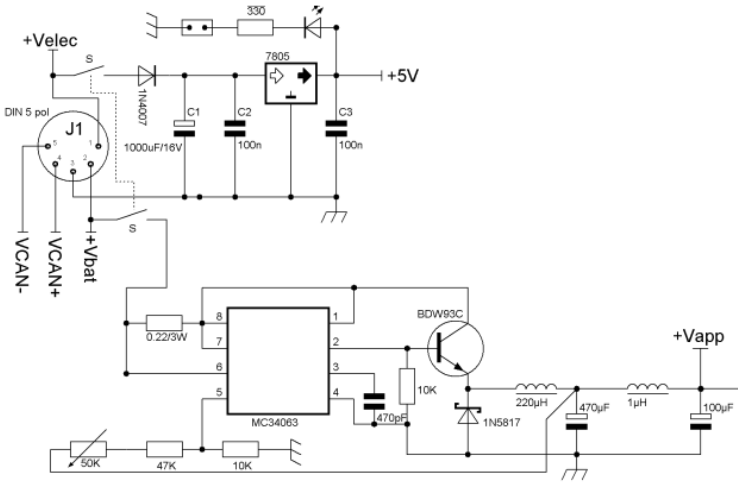
La broche MCLR permet également d'assurer un RESET matériel.

Le superviseur d'alimentation MC34164 est activé en mode normal par la mise en place d'un cavalier sur le connecteur NICD. Il permet de produire un RESET matériel dans le cas ou la tension d'alimentation devient trop faible.

Lors de la programmation ou la mise au point d'un programme, ce cavalier est à enlevé. Le microcontrôleur fonctionne à une fréquence de 4 MHz obtenue grâce à un quartz.

Alimentation de la carte.

L'alimentation de la carte est obtenue depuis le bus système.



La tension *+Velec* de 8V générée par le module énergie permet grâce au régulateur 7805 de fournir la tension de 5V nécessaire aux différents circuits de la carte.

La matrice de LEDS est alimentée par la tension *+Vapp* obtenue par abaissement de la tension *+Vbat* du bus système et générée par le module énergie. On utilise un régulateur à découpage construit autour du circuit MC34063 afin d'améliorer le rendement de la conversion.

La LED indicateur de présence tension peut être désactivée par le cavalier prévu à cet effet afin de diminuer la consommation.

Un interrupteur S permet de mettre la carte hors tension. Cette interrupteur peut être omis sur certaines versions (on a alors des « straps » et le module et continuellement sous tension).

Commande de la matrice à LEDS.

La matrice, alimentée par la tension *+Vapp*, est commandée par le microcontrôleur au moyen de 5 signaux.

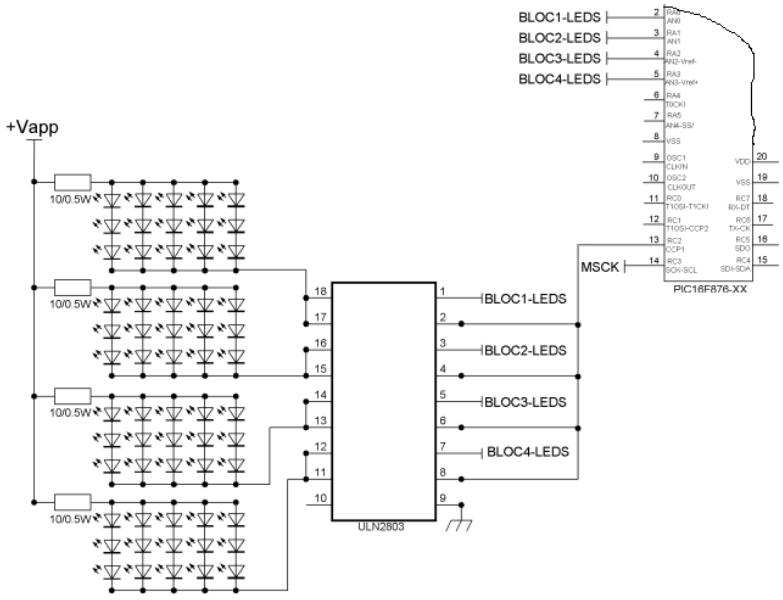
Un *buffer* 8 voies ULN2803, permet de commander indépendamment chacun des 4 pavés de 16 LEDS grâce aux 4 signaux *BLOCx-LEDS*.

La sortie CPP1 du PIC permet de produire des signaux périodiques à rapport cyclique variable. Cette sortie peut, au travers du *buffer* commander l'ensemble des 4 blocs donc toute la matrice.

Le réglage du rapport cyclique permettant alors de faire varier le niveau de luminosité des LEDS.

Le circuit ULN2803 dispose de sortie à collecteur ouvert ce qui permet de mettre en parallèle un couple de 2 sorties.

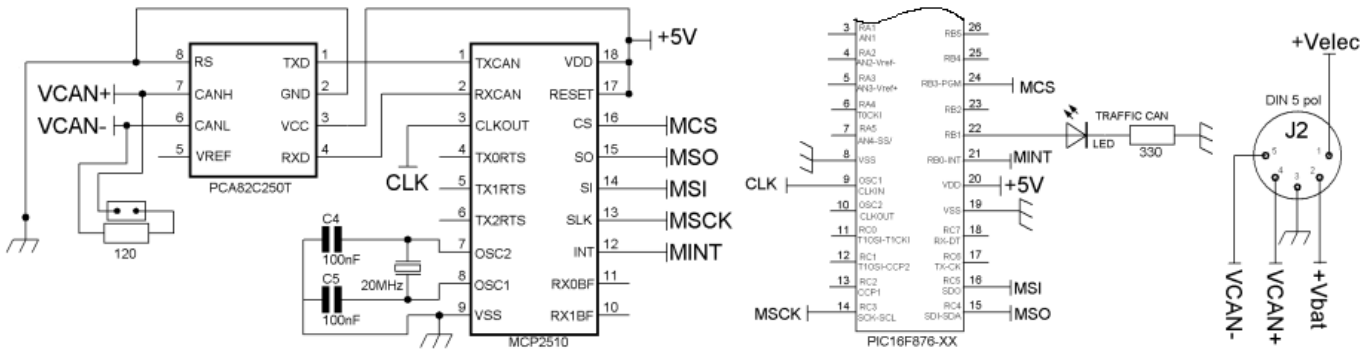
On pourra, avec la même structure, soit commander un des blocs de 16 LEDS en tout ou rien, soit moduler la luminosité de l'ensemble de la matrice.



Le niveau d'intensité dans les LEDS est réglable par la tension *+Vapp* qu'on peut ajuster. Les résistances de 10 ohms limitent le courant.

Accès au bus CAN.

Pour l'accès au bus CAN, on utilise les éléments suivants :



Le circuit MCP2510 dialogue avec le microprocesseur de la carte à travers les liaisons suivantes :

- Un bus série constitué de MSO, MSI et l'horloge MSCK.
- Un fils de sélection de boîtier MCS.
- La sortie MINT peut être utilisé pour générer des interruptions pour le microprocesseur (non activé dans notre cas).

La liaison du circuit MCP2510 avec le bus CAN s'effectue grâce aux broches TXCAN et RXCAN (niveau TTL) et le circuit d'interface PCA82C250T qui produit les signaux du bus. Le bus utilise les signaux VCAN+, VCAN- et GND.

Le circuit MCP2510 nécessite pour rythmer les signaux du bus CAN d'un quartz fixé ici à 20 MHz. La broche CLKOUT produit un signal de 2,5 MHz (division par 8 de la fréquence du quartz) qui est utilisé pour fournir une horloge au circuit PIC.

On notera la présence de la résistance RT de 120 ohms qui permet d'assurer, si nécessaire, la terminaison du bus CAN si l'on met en place le cavalier prévu.
La LED « TRAFFIC CAN » sera commandé par le PIC pour signaler une activité sur le bus CAN.

Etude du logiciel embarqué dans le PIC.

En ce qui concerne le logiciel, on trouve tout d'abord les indications relatives à la compilation du programme pour le PIC.

```
#include <16F876.h>
#device ICD=TRUE
#device *=16
#use delay(clock=2500000)
#fuses XT, NOPROTECT, BROWNOUT, NOWDT
#zero_ram                                   //remet la ram a 0 (initialise les variables a 0)
```

Gestion du bus CAN.

Le driver logiciel du circuit MCP2510 est pris en compte par la ligne :

```
#include "can.c"                            //driver can a consulter pour bits utilises
```

L'identifiant utilisé pour ce module est déclaré :

```
/* identifiants bus can */
#define i_rmain  0x200                      //pour reception depuis module de controle
```

La commande de la LED d'activité est également déclarée :

```
#define actcan_on output_high(PIN_B1)
#define actcan_off output_low(PIN_B1)
```

Le bus CAN n'est utilisé qu'en réception par la fonction *gest_can* .

La réception est opérée par scrutation au moyen de la commande `if ( can_kbhit() )`

Si une réception est détectée, on examine l'identifiant et, suivant le cas, on recueille les données adéquates.

```
void gest_can(){
   if ( can_kbhit() ){                                  //y a t il des donnees dans le buffer ?...
      if(can_getd(crx_id, &crxbuf[0], crx_len, rxstat)){    //...si oui lecture des donnees
         if (crx_id == i_rmain) {
            actcan_on;                                  //change la LED activite can
            ec_com=crxbuf[0];                           //commande
            ec_int=crxbuf[1];                           //intensite
            flag=1;
         }
      }
   }
}
```

Gestion de la matrice de LEDS.

On trouve la définition des commandes de la matrice en tout ou rien.

```
//IO broches connectees aux blocs de la matrice
#define BLOC1_OFF output_low(PIN_A0)
#define BLOC2_OFF output_low(PIN_A1)
#define BLOC3_OFF output_low(PIN_A2)
#define BLOC4_OFF output_low(PIN_A3)
#define BLOC1_ON  output_high(PIN_A0)
#define BLOC2_ON  output_high(PIN_A1)
#define BLOC3_ON  output_high(PIN_A2)
#define BLOC4_ON  output_high(PIN_A3)
```

La matrice de LEDS est commandée en fonction de la variable *ec_com* qui indique la commande à effectuer et de la variable *ec_int* qui permet de moduler le niveau d'éclairage de la matrice.

La fonction *gest_mat* est lancée cycliquement par le programme principal.

S'il y a eu réception d'une commande depuis le bus CAN, flag est alors à 1, on exécute cette commande.

```
void gest_mat(){
   if(flag==1){
      if (ec_com==0){
         BLOC1_OFF;
         BLOC2_OFF;
         BLOC3_OFF;
         BLOC4_OFF;
         set_pwm1_duty(0);
      }
      if (ec_com==1){
         BLOC1_OFF;
         BLOC2_OFF;
         BLOC3_OFF;
         BLOC4_OFF;
         set_pwm1_duty(ec_int*17);
      }
      if (ec_com==2){
         BLOC1_ON;
         BLOC2_OFF;
         BLOC3_OFF;
         BLOC4_OFF;
         set_pwm1_duty(0);
      }
      if (ec_com==3){
         BLOC1_OFF;
         BLOC2_ON;
         BLOC3_OFF;
         BLOC4_OFF;
         set_pwm1_duty(0);
      }
      if (ec_com==4){
         BLOC1_OFF;
         BLOC2_OFF;
         BLOC3_ON;
         BLOC4_OFF;
         set_pwm1_duty(0);
      }
      if (ec_com==5){
         BLOC1_OFF;
         BLOC2_OFF;
         BLOC3_OFF;
         BLOC4_ON;
         set_pwm1_duty(0);
      }
      flag=0;
   }
}
```

Le programme principal.

La déclaration des variables :

```
/* variables du systeme */
int ec_com;                          //indicateur pour activation eclairage
int ec_int;                          //niveau eclairage
int1 flag;
```

On déclare les fonctions :

```
/* declaration des fonctions */
void init();
void gest_mat();
void gest_can();
```

Reste le programme principal qui appelle le programme d'initialisation :

```
/* programme principal */
void main()
{
   init();
   while (1)
   {
      actcan_off;
      gest_can();
      gest_mat();
   }
}
```

Le programme d'initialisation initialise le bus CAN et le *timer*.

```
void init(){
   can_init();
   flag=1;;
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DIV_BY_1,255,1);
   setup_ccp1(CCP_PWM);
}
```