```
// Aff_mode_compt_conv_trans_mem.c      AFF_Mux, Affiche, Choix_mode, Compte, Convertit_temps,  Transmet, Memorise,
// Pour PIC16F628A
// Robert Denoyel
// Le PIC est en horloge interne sans Clock Out( 4 MHz)          #FUSES INTRC_IO   //Internal RC Osc, no CLKOUT
// La patte MCLR est disponible pour le reset                    #FUSES MCLR       //Master Clear pin enabled
// Le multiplexage de l'affichage est géré par timer0 (extinction et allumage pendant 8,2ms)
// Le BP MODE/TRANSMET est connecté sur PA7
// Le choix du mode se fait par positionnement de SW1 ( MODE ) durant les 2 secondes suivant le reset
// Timer2 ( regle a 10ms )permet le comptage des secondes ( temps-sport)
// Valid_sport est définie par ILS ( SW2 en PB0 ) ( =0--> pas de comptage )
// temps_sport est affichee sur AFFG et AFFD
// En mode memorise, la transmission se fait par appui sur SW1 (TRANSMET)
// On transmet le caractere "A" suivi des 4 octets de temps_sport
// On memorise la meme chose
// En mode memorise,temps_sport prend la derniere valeur memorisee durant les 2 premieres secondes du programme

/* Inclusions et équivalences */
#include "EmetSportTV 16F628.h"
#define selaff_gauche pin_B5              // Sélection de l'afficheur gauche
#define selaff_droit pin_B4               // Sélection de l'afficheur droit
#define SW1 pin_A7                        // Entrée de sélection du mode direct ou mémorise
#define ILS pin_B0                        // Entrée de l'ILS

/* Déclaration des variables globales */
int cathodes_gauche=0b00000000;           // Mot de commandes des cathodes de l'afficheur gauche initialisé à éteint (1 --> allumé)
int cathodes_droit=0b00000000;            // Mot de commandes des cathodes de l'afficheur droit initialisé à éteint (1 --> allumé)
int temp=0;                               // variable temporaire
int temp1=0;                              // variable temporaire
int choix_affich=0;                       // Variable de choix d'afficheur (0 -->gauche )
int mode=0;                               // variable d'indication du mode de fonctionnement (0 --> direct)
int valid_sport=1;                        // variable de validation du temps de calcul
int Nbre_fois=0;                          // variable de calcul du nombre d'interruption de timer2 (100-->1s)
int32 temps_sport=0;                      // variable de calcul du temps de sport en secondes (9h50mn max-->35400s max)
int heures=0;                             //nombres d' heures de sport
int minutes=0;                            //nombres de minutes de sport
int secondes=0;                           //nombres de secondes de sport
int AffG=0;                               // chiffre gauche du temps compte ( commandes 7seg pour le 0,1....9 )
int AffD=0;                               // chiffre droit du temps compte  ( commandes 7seg pour le 0,1....9 )
int G=0;                                  //chiffre gauche du temps compte à afficher en binaire
int D=0;                                  //chiffre droit du temps compte à afficher en binaire
int
tab[10]={0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111100,0b00000111,0b01111111,0b01100111};
//table de conversion
int temps_10s=0;                          // Temps de 10s atteint
char trame_tx[5]={0,0,0,0,0};             //tableau de trame d'envoi
int nboctet=0;                            // variable d'envoi de trame
int16 address_base_eeprom=0;              //Adresse de base de l'EEPROM de donnees

/* Déclaration des prototypes */
void affiche(void);                       // Pas de variable d'entrée et de  sortie
void choixmode(void);                     // Pas de variable d'entrée et de  sortie
void convertit_temps(void);               // Pas de variable d'entrée et de  sortie
void transmet(void);                      //pas de variable d'entrée et de  sortie
void memorise(void);                      //pas de variable d'entrée et de  sortie

/* Déclaration des fonctions */

/* Fonction affiche: affiche le contenu de cathodes_gauche ou cathodes_droit
sur l'afficheur gauche ou droit en fonction de choix_affich */
void affiche(void)
{
  if (choix_affich==1)
    {
      temp=cathodes_gauche;
      temp=temp & 0b00000001;
      if(temp!=0)output_low(Pin_A0); else output_high(Pin_A0);
      temp=cathodes_gauche;
      temp=temp & 0b00000010;
      if(temp!=0)output_low(Pin_A1); else output_high(Pin_A1);
      temp=cathodes_gauche;
      temp=temp & 0b00000100;
      if(temp!=0)output_low(Pin_A2); else output_high(Pin_A2);
      temp=cathodes_gauche;
      temp=temp & 0b00001000;
      if(temp!=0)output_low(Pin_A3); else output_high(Pin_A3);
      temp=cathodes_gauche;
      temp=temp & 0b00010000;
      if(temp!=0)output_low(Pin_A4); else output_high(Pin_A4);
```

```
      temp=cathodes_gauche;
      temp=temp & 0b00100000;
      if(temp!=0)output_low(Pin_A6); else output_high(Pin_A6);
      temp=cathodes_gauche;
      temp=temp & 0b01000000;
      if(temp!=0)output_low(Pin_B6); else output_high(Pin_B6);
      temp=cathodes_gauche;
      temp=temp & 0b10000000;
      if(temp!=0)output_low(Pin_B7); else output_high(Pin_B7);
      output_low(selaff_gauche);              // validation de l'afficheur gauche
      output_high(selaff_droit);              // Inhibition de l'afficheur droit
    }
  else
    {
      temp=cathodes_droit;
      temp=temp & 0b00000001;
      if(temp!=0)output_low(Pin_A0); else output_high(Pin_A0);
      temp=cathodes_droit;
      temp=temp & 0b00000010;
      if(temp!=0)output_low(Pin_A1); else output_high(Pin_A1);
      temp=cathodes_droit;
      temp=temp & 0b00000100;
      if(temp!=0)output_low(Pin_A2); else output_high(Pin_A2);
      temp=cathodes_droit;
      temp=temp & 0b00001000;
      if(temp!=0)output_low(Pin_A3); else output_high(Pin_A3);
      temp=cathodes_droit;
      temp=temp & 0b00010000;
      if(temp!=0)output_low(Pin_A4); else output_high(Pin_A4);
      temp=cathodes_droit;
      temp=temp & 0b00100000;
      if(temp!=0)output_low(Pin_A6); else output_high(Pin_A6);
      temp=cathodes_droit;
      temp=temp & 0b01000000;
      if(temp!=0)output_low(Pin_B6); else output_high(Pin_B6);
      temp=cathodes_droit;
      temp=temp & 0b10000000;
      if(temp!=0)output_low(Pin_B7); else output_high(Pin_B7);
      output_high(selaff_gauche);             // Inhibition  de l'afficheur gauche
      output_low(selaff_droit);               //  validation de l'afficheur droit
    }
}

/* Fonction choixmode: Au démarrage, en fonction de l'état de SW1, détermine le choix direct ou mémorisé
avec visualisation sur afficheur droit */
void choixmode(void)
{
  disable_interrupts(INT_TIMER1);
  temp=0;
  while((temp==0) && (input(SW1)==1))         // Attente d'appui sur SW1 pendant 2s
    {
      mode=0;                             // Passage en mode direct
      cathodes_droit=0b01011110;              // Mot de commandes des cathodes de l'afficheur droit initialisé à d (1 --> allumé)
      cathodes_gauche=0b00000000;             // Mot de commandes des cathodes de l'afficheur gauche initialisé à éteint (1 --> allumé)
      delay_ms(10);
      temp1=temp1+1;                      // incrementation de temp1 equivalent a 1ms
      if (temp1==200)                     // temps supérieur a 2 secondes ?
        temp=1;
      else;
    }
  if (temp==0)
    {
      mode=1;                             // Passage en mode mémorisé
      cathodes_droit=0b00110111;              // Mot de commandes des cathodes de l'afficheur droit initialisé à M (1 --> allumé)
      cathodes_gauche=0b00000000;             // Mot de commandes des cathodes de l'afficheur gauche initialisé à éteint (1 --> allumé)
      delay_ms(2000);
   // recuperation de la valeur memorisee de temps_sport
      For (nboctet=0; nboctet<=4; nboctet++ )                 // 4 octets lus
        {
          trame_tx[nboctet]=read_eeprom( address_base_eeprom+nboctet ); //ad0:TRX1----->ad3:TRX4
          If ((trame_tx[4]!=0xFF)&&(trame_tx[3]!=0xFF)&&(trame_tx[2]!=0xFF)&&(trame_tx[1]!=0xFF)) // memoire differente de $FF
             temps_sport=trame_tx[4]+trame_tx[3]*256+trame_tx[2]*62536+trame_tx[1]*16777216;      // temps_sport <---memoire
          Else;                 // temps de sport non change ( ==0 )
        }
      nboctet=0;
    }
  else;
  enable_interrupts(INT_TIMER1);
```

```
}

/* Fonction convertit_temps: convertit le temps (en secondes) en heures, minutes, secondes
Gere l'affichage en unité d'heure et dizaines de minutes ou en dizaines et unites de minutes ou en dizaines et unites de secondes ( G, D )
Gere les commandes des leds correspondantes (AFFG, AFFD )                    */
void convertit_temps(void)
{
  heures=temps_sport/3600;
  minutes=temps_sport/60-heures*60;
  secondes=(temps_sport-heures*3600-minutes*60);
  If ((heures==0) && (minutes==0))
    {G=secondes/10;D=secondes-G*10;AffD=tab[D]+128;AffG=tab[G]+128;}  //affichage secondes --> 2 points
  else
    {
      If ((heures==0) && (minutes!=0))
        {G=minutes/10;D=minutes-G*10;AffD=tab[D]+128;AffG=tab[G];}  //affichage minutes --> 1 points
      else
        {
          If (heures<10)
            {G=heures;D=minutes/10;AffD=tab[D];AffG=tab[G];}         //affichage heures et dizaines de minutes-->pas de points
          else
            {AffG=0b11111111;AffD=0b11111111;}                    //temps max = 9H50
        }
    }
}

/* Fonction transmet: transmet sur Tx dans l'ordre A puis 4e octet, 3e octet, 2e octet et 1er octet de temps_sport */
void transmet(void)
{
  cathodes_gauche=0b01110000;
  DELAY_ms(1000);
  trame_tx[0] = 'A';          // Code début de trame
  trame_tx[1] = (temps_sport/16777216);                    //4e octet de temps_sport dans  trame_tx[1]
  trame_tx[2] = (temps_sport - trame_tx[1] * 16777216)/65536;    //3e octet de temps_sport dans  trame_tx[2]
  trame_tx[3] = (temps_sport - trame_tx[1] * 16777216- trame_tx[2] * 65536)/256;    //2e octet de temps_sport dans  trame_tx[3]
  trame_tx[4] = (temps_sport - trame_tx[1] * 16777216- trame_tx[2] * 65536- trame_tx[3] * 256);    //1er octet de temps_sport dans
trame_tx[4]
  for (nboctet = 0 ; nboctet < 5 ; nboctet++)    // on envoie 5 octets
    putc(trame_tx[nboctet]);   // écriture sur liaison série
}

/* Fonction memorise: memorise temps_sport dans la EEPROM de donnee
write_eeprom (address, value)                                    */
void memorise(void)
{
  trame_tx[0] = 'A';          // Code début de trame
  trame_tx[1] = (temps_sport/16777216);                    //4e octet de temps_sport dans  trame_tx[1]
  trame_tx[2] = (temps_sport - trame_tx[1] * 16777216)/65536;    //3e octet de temps_sport dans  trame_tx[2]
  trame_tx[3] = (temps_sport - trame_tx[1] * 16777216- trame_tx[2] * 65536)/256;    //2e octet de temps_sport dans  trame_tx[3]
  trame_tx[4] = (temps_sport - trame_tx[1] * 16777216- trame_tx[2] * 65536- trame_tx[3] * 256);    //1er octet de temps_sport dans
trame_tx[4]
  For (nboctet=0; nboctet<=4;nboctet++)
    write_eeprom( address_base_eeprom+nboctet, trame_tx[nboctet] ); //ad0:TRX1----->ad3:TRX4
  nboctet=0;
}

/* Fonction init_eeprom: reinitialise l'eeprom donnees a $FF   */
void init_eeprom (void)
  {
      For (nboctet=0; nboctet<=4;nboctet++)
          write_eeprom( address_base_eeprom+nboctet, 0xFF ); //ad0:FF----->ad3:FF
          nboctet=0;
  }

/* déclaration du programme d'interruption de débordement du timer0 (muxliplexage de l'affichage=8,2ms) */
#int_RTCC
RTCC_isr()
{
  disable_interrupts(INT_RTCC);                  // inhibition des interruptions du timer0
    if (choix_affich==0)
      choix_affich=1;
    else
      choix_affich=0;
  affiche();
  enable_interrupts(INT_RTCC);                  // validation des interruptions du timer0
}

/* déclaration du programme d'interruption de débordement du timer2 (interruption toutes les 10ms)
```

```
On incrémente  temps_sport toutes les secondes     On transmet ou memorise toutes les 10 secondes */
#int_TIMER2
TIMER2_isr()
{
  disable_interrupts(INT_TIMER2);                    // inhibition des interruptions du timer2
  if (input(ILS)==0)
    valid_sport=0;
  else
    valid_sport=1;
  if (Nbre_fois<=100)                                // temps écoulé<=1s?
    Nbre_fois=Nbre_fois+1;
  else
    {
      if (valid_sport==1)
        temps_sport=temps_sport+1;
      else;
      convertit_temps();
      cathodes_gauche=AffG;              // affichage du temps compté sur AFFG
      cathodes_droit=AffD;              //  AFFD eteint
      Nbre_fois=0;
      if (temps_10s<=10)                 // 10s ?
        temps_10s=temps_10s+1;
      else
      {  if (mode==0)                    // Si 10s et mode direct =>transmet
           {
             transmet();
           }
         else
           memorise();                  // Si 10s et mode memorisz =>memorise
      temps_10s=0;
      }
    }
  enable_interrupts(INT_TIMER2);                     // validation des interruptions du timer2
}

/* Programme Principal */
void main()
{
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32);    //    timer0 interruption au bout de 8,2ms
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DIV_BY_4,156,16);               // Timer2 regle à 10ms
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  enable_interrupts(INT_RTCC);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(GLOBAL);
  SET_TRIS_B(0b00001011);                          //E/S de PB
  SET_TRIS_A(0b10100000);                          //E/S de PA
  choixmode();
  do
  {
  while (mode==1)                     // programme en mode mémorisé
    {
      if (input(SW1)==0)
        {
          transmet();
          init_eeprom();
        }
      else;                           // programme en mode direct
    }
  }
  While( true);
}
```