

Annexe DC : programme PIC User.c

Annexe DC - Programme PIC User.C

```
/** INCLUDES *****/
#include "usb.h"
#include "usb_function_generic.h"
#include "HardwareProfile.h"
#include "user.h"
#include "timers.h"

/** V A R I A B L E S *****/
#pragma udata
BYTE old_sw2,old_sw3;
BYTE counter;
BYTE trf_state;
BYTE temp_mode;
int val_tmr1H = 0;
int val_tmr1L = 0;
int prescal_bit0 = 0;
int prescal_bit1 = 0;
int temp = 0;
int val_tmr3H_IR = 0;
int val_tmr3L_IR = 0;
int val_tmr3H_UV = 0;
int val_tmr3L_UV = 0;
int nb_tour = 0;
int cycle = 60;
int mode_tour = 1;
int num_servo = 0;
int choix_cam = 0;

#pragma udata USB_VARIABLES=0x500

DATA_PACKET INPacket;
DATA_PACKET OUTPacket;
#pragma udata

USB_HANDLE USBGenericOutHandle = 0;
USB_HANDLE USBGenericInHandle = 0;

BOOL blinkStatusValid = TRUE;

***** DEFINITION *****

//declaration ports servos moteurs
#define servo1 PORTCbits.RC1
#define servo1_tris TRISCbits.TRISC1
#define servo2 PORTCbits.RC2
#define servo2_tris TRISCbits.TRISC2

//declaration ports moteur Caméra
#define Mot_Cam_Horloge PORTDbits.RD0
#define Mot_Cam_Horloge_tris TRISDbits.TRISD0
#define Mot_Cam_pas_mode PORTCbits.RC7
#define Mot_Cam_pas_mode_tris TRISCbits.TRISC7
#define Mot_Cam_CS PORTDbits.RD2
#define Mot_Cam_CS_tris TRISDbits.TRISD2
#define Mot_Cam_Sens_Rotation PORTDbits.RD1
```

Intégration des bibliothèques

Déclaration des variables

Configuration du port C pour la commande des servomoteurs

Configuration du port C/D pour la commande des moteurs pas à pas

Annexe DC - Programme PIC User.C

```

#define Mot_Cam_Sens_Rotation_tris TRISDbits.TRISD1
#define Mot_Cam_RESET PORTCbits.RC6
#define Mot_Cam_RESET_tris TRISCbits.TRISC6

#define Mot_Mir_Horloge PORTDbits.RD6
#define Mot_Mir_Horloge_tris TRISDbits.TRISD6
#define Mot_Mir_pas_mode PORTDbits.RD5
#define Mot_Mir_pas_mode_tris TRISDbits.TRISD5
#define Mot_Mir_CS PORTDbits.RD3
#define Mot_Mir_CS_tris TRISDbits.TRISD3
#define Mot_Mir_Sens_Rotation PORTDbits.RD7
#define Mot_Mir_Sens_Rotation_tris TRISDbits.TRISD7
#define Mot_Mir_RESET PORTDbits.RD4
#define Mot_Mir_RESET_tris TRISDbits.TRISD4

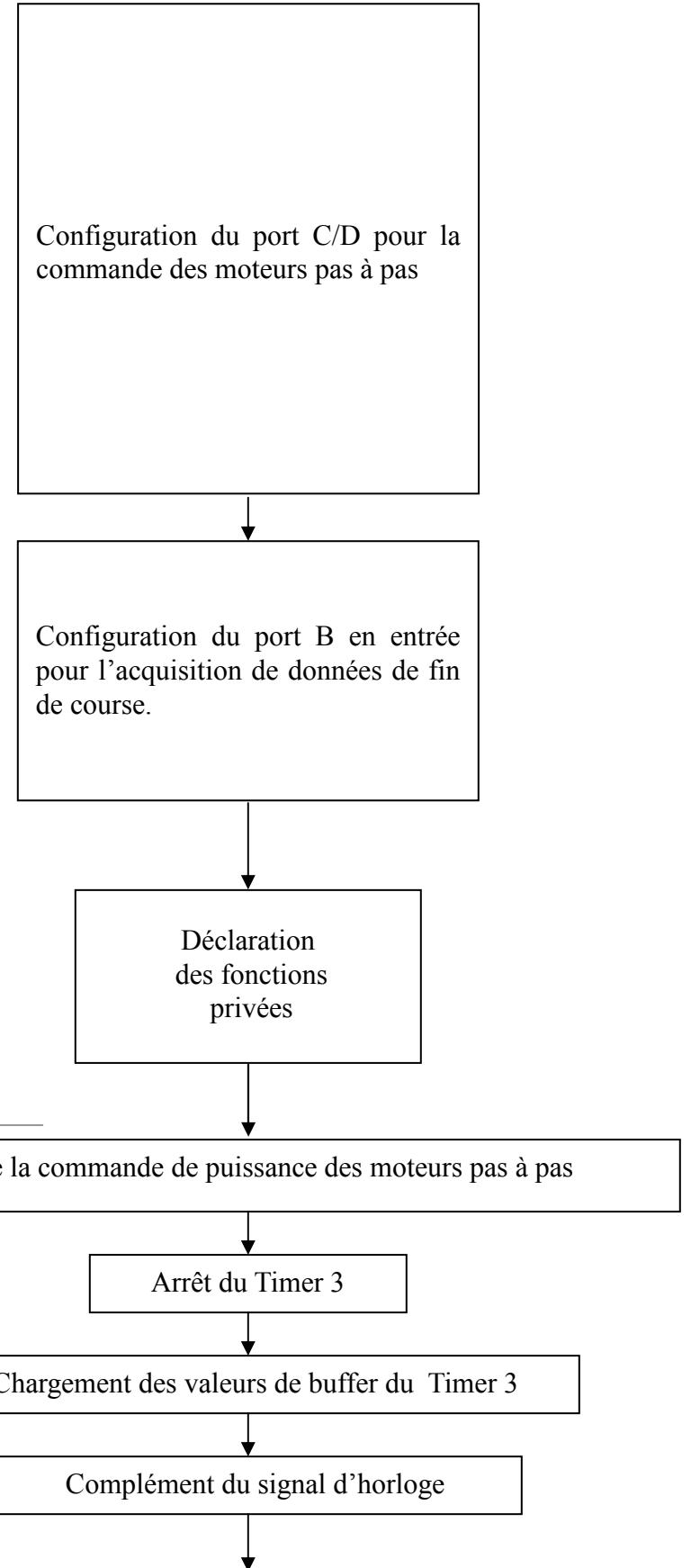
//déclaration des entrées fin de course
#define CFC_CAM_AV PORTBbits.RB6
#define CFC_CAM_AV_tris TRISBbits.TRISB6
#define CFC_CAM_AR PORTBbits.RB7
#define CFC_CAM_AR_tris TRISBbits.TRISB7
#define CFC_MIR_HT PORTBbits.RB5
#define CFC_MIR_HT_tris TRISBbits.TRISB5
#define CFC_MIR_Bas PORTBbits.RB4
#define CFC_MIR_Bas_tris TRISBbits.TRISB4

/** PRIVATE PROTOTYPES *****/
void ServiceRequests(void);

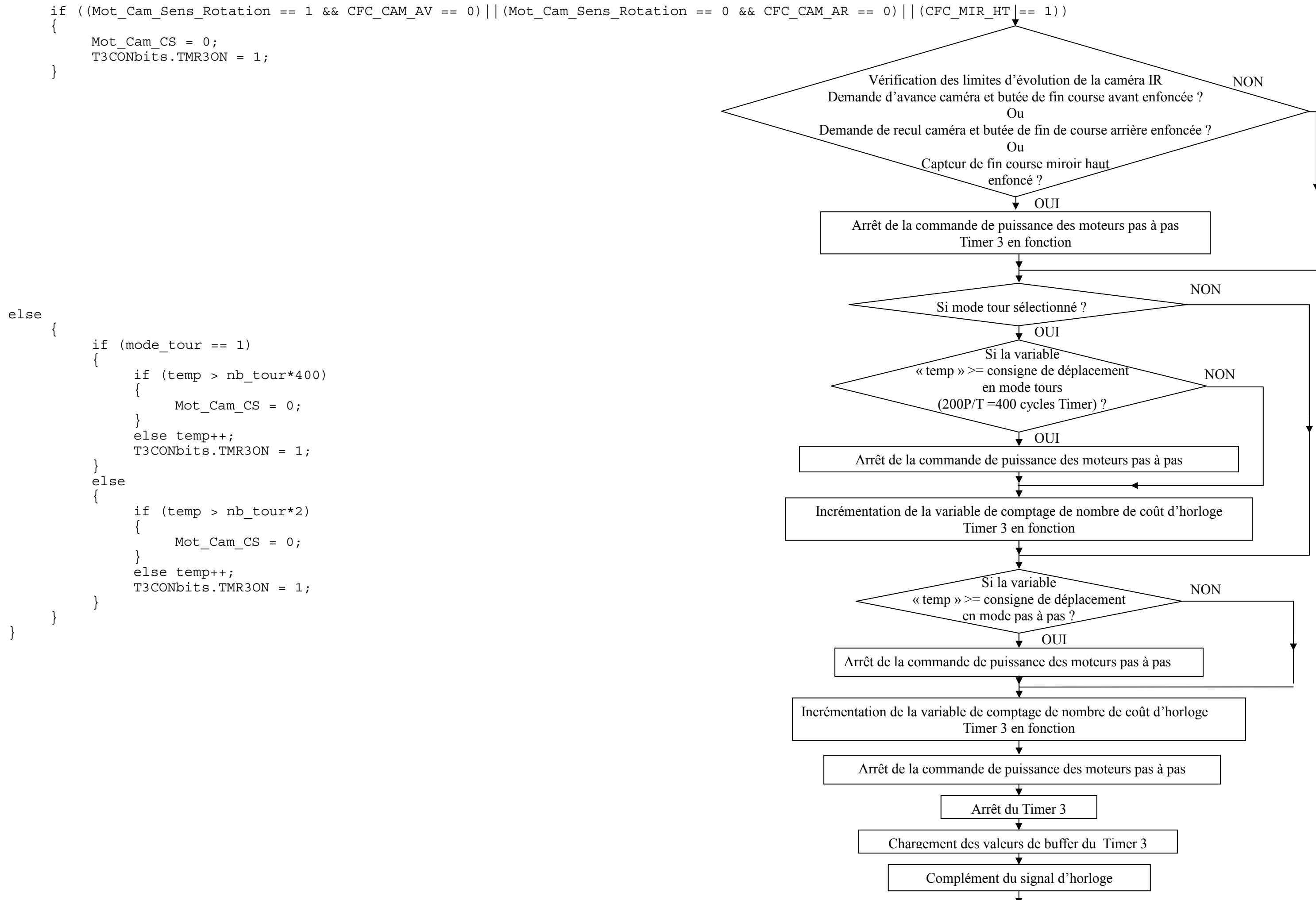
/** DECLARATIONS *****/
#pragma code

void Mot_CamIR()
{
    Mot_Mir_CS = 0;
    T3CONbits.TMR3ON = 0;
    TMR3L = val_tmr3L_IR;
    TMR3H = val_tmr3H_IR;
    Mot_Cam_Horloge=!Mot_Cam_Horloge;
    // reset le timer3
}

```



Annexe DC - Programme PIC User.C



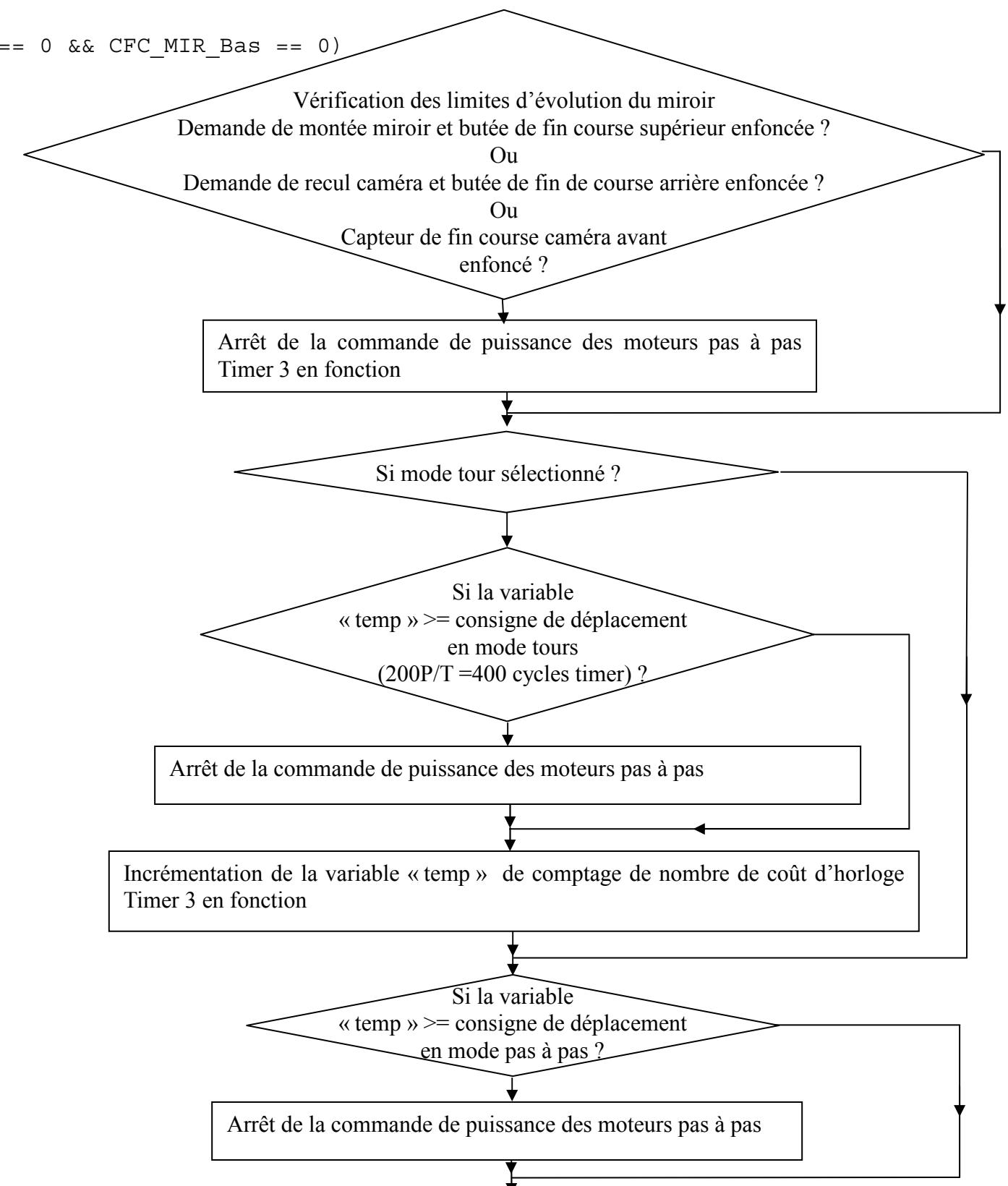
Annexe DC - Programme PIC User.C

```

void Mot_CamUV()
{
    Mot_Cam_CS = 0;
    T3CONbits.TMR3ON = 0;
    TMR3L = val_tmr3L_UV; // reset le timer3
    TMR3H = val_tmr3H_UV;
    Mot_Mir_Horloge=!Mot_Mir_Horloge;

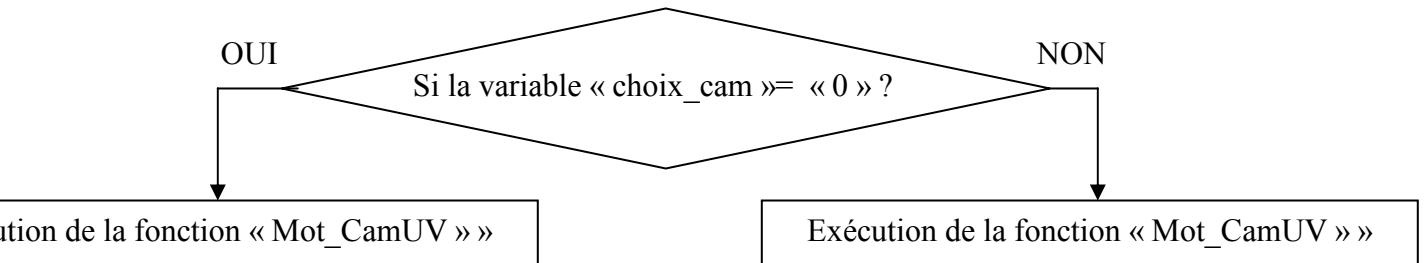
    if ((Mot_Mir_Sens_Rotation == 1 && CFC_MIR_HT == 0) || (Mot_Mir_Sens_Rotation == 0 && CFC_MIR_Bas == 0)
        || (CFCc_CAM_AV == 1))
    {
        Mot_Mir_CS = 0;
        T3CONbits.TMR3ON = 1;
    }
    else
    {
        if (mode_tour == 1)
        {
            if (temp > nb_tour*400)
            {
                Mot_Mir_CS = 0;
            }
            else temp++;
            T3CONbits.TMR3ON = 1;
        }
        else
        {
            if (temp > nb_tour*2)
            {
                Mot_Mir_CS = 0;
            }
            else temp++;
            T3CONbits.TMR3ON = 1;
        }
    }
}

```

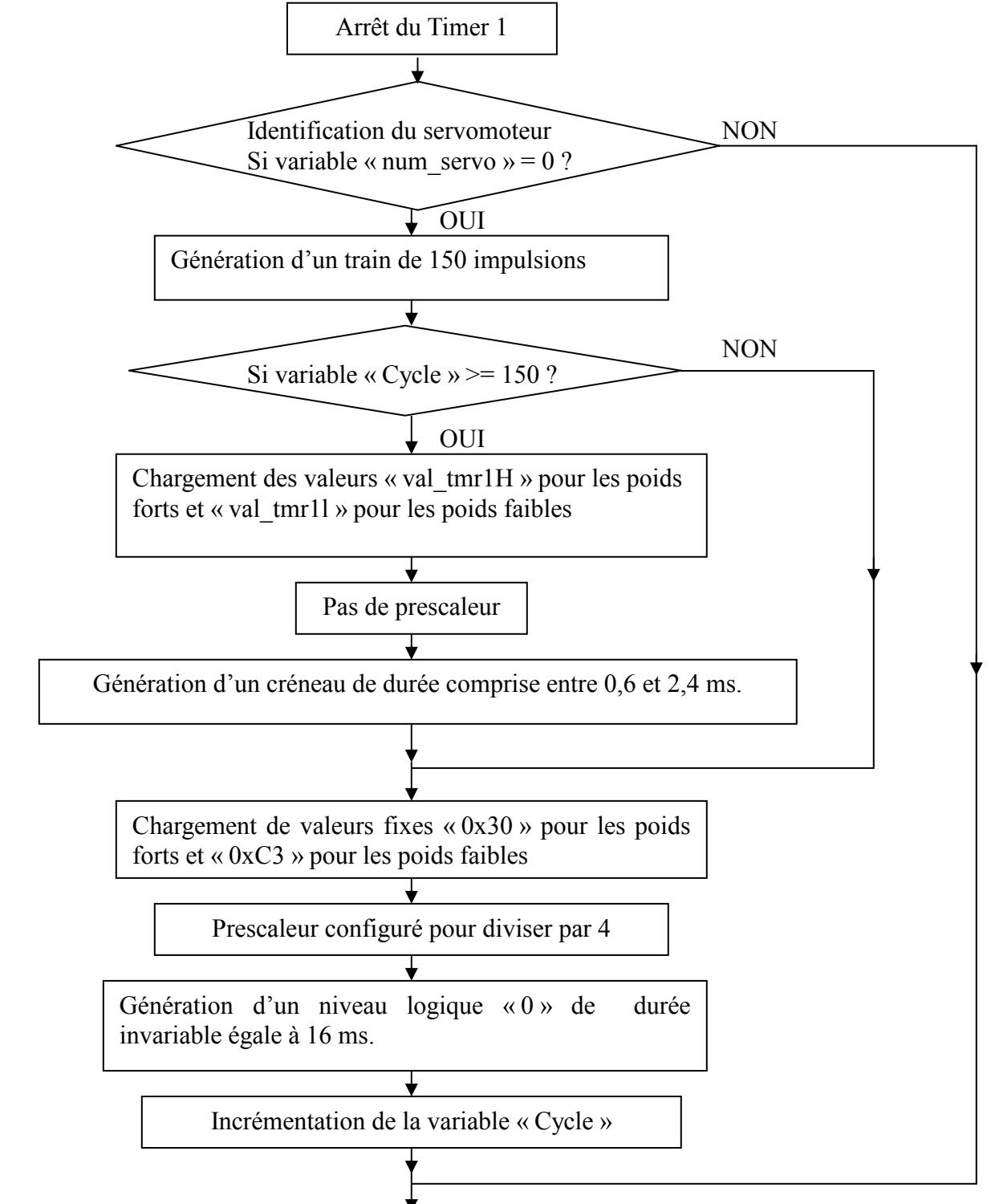


Annexe DC - Programme PIC User.C

```
void Mot_Cam()
{
    if (choix_cam == 0) Mot_CamUV();
    else Mot_CamIR();
}
```



```
void servo()
{
    T1CONbits.TMR1ON = 0;
    if (num_servo == 0)
    {
        if (cycle <= 150)
        {
            if (servo1 == 0)
            {
                TMR1H = val_tmr1H;
                TMR1L = val_tmr1L;
                T1CONbits.T1CKPS0 = 0; // prescaleur a 1:1
                T1CONbits.T1CKPS1 = 0;
                servo1 = 1;
            }
            else
            {
                TMR1H = 0x30;
                TMR1L = 0xC3;
                T1CONbits.T1CKPS0 = 0; // prescaleur a 1:4
                T1CONbits.T1CKPS1 = 1;
                servo1 = 0;
            }
            cycle++;
        }
    }
}
```

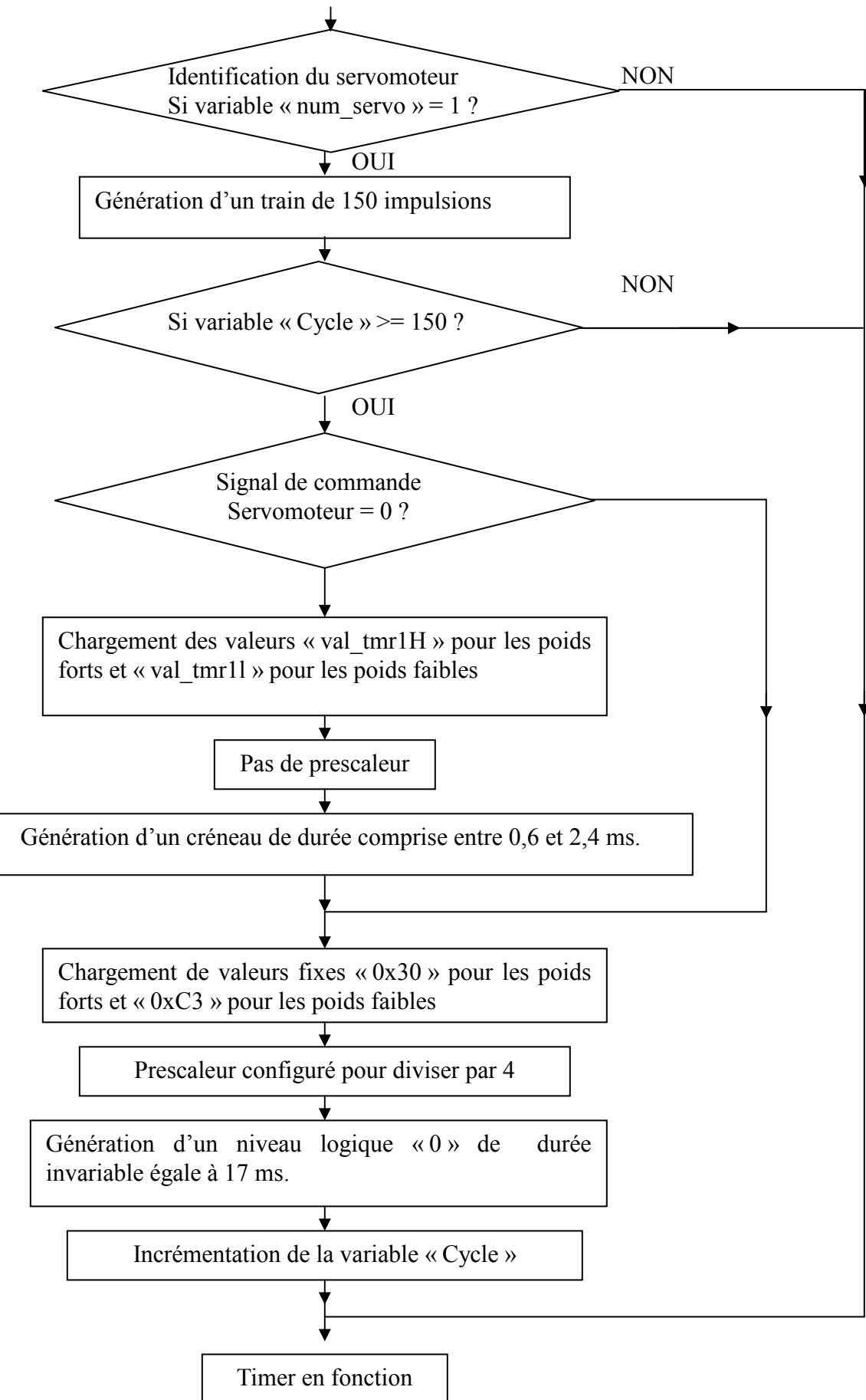


Annexe DC - Programme PIC User.C

```

else if (num_servo == 1)
{
    if (cycle <= 150)
    {
        if (servo2 == 0)
        {
            TMR1H = val_tmr1H;
            TMR1L = val_tmr1L;
            T1CONbits.T1CKPS0 = 0; // prescaleur a 1:1
            T1CONbits.T1CKPS1 = 0;
            servo2 = 1;
        }
        else
        {
            TMR1H = 0x30;
            TMR1L = 0xC3;
            T1CONbits.T1CKPS0 = 0; // prescaleur a 1:4 modif le 16/09/2010
            T1CONbits.T1CKPS1 = 1;
            servo2 = 0;
        }
        cycle++;
    }
    T1CONbits.TMR1ON = 1;
}

```



Annexe DC - Programme PIC User.C

```

***** Initialisation utilisateur *****

RCONbits.IPEN = 1;           // valide les interruptions prioritaire
INTCONbits.GIEH = 1;          // valide les interruption haute priorité
INTCONbits.GIEL = 1;          // valide les interruption basse priorité

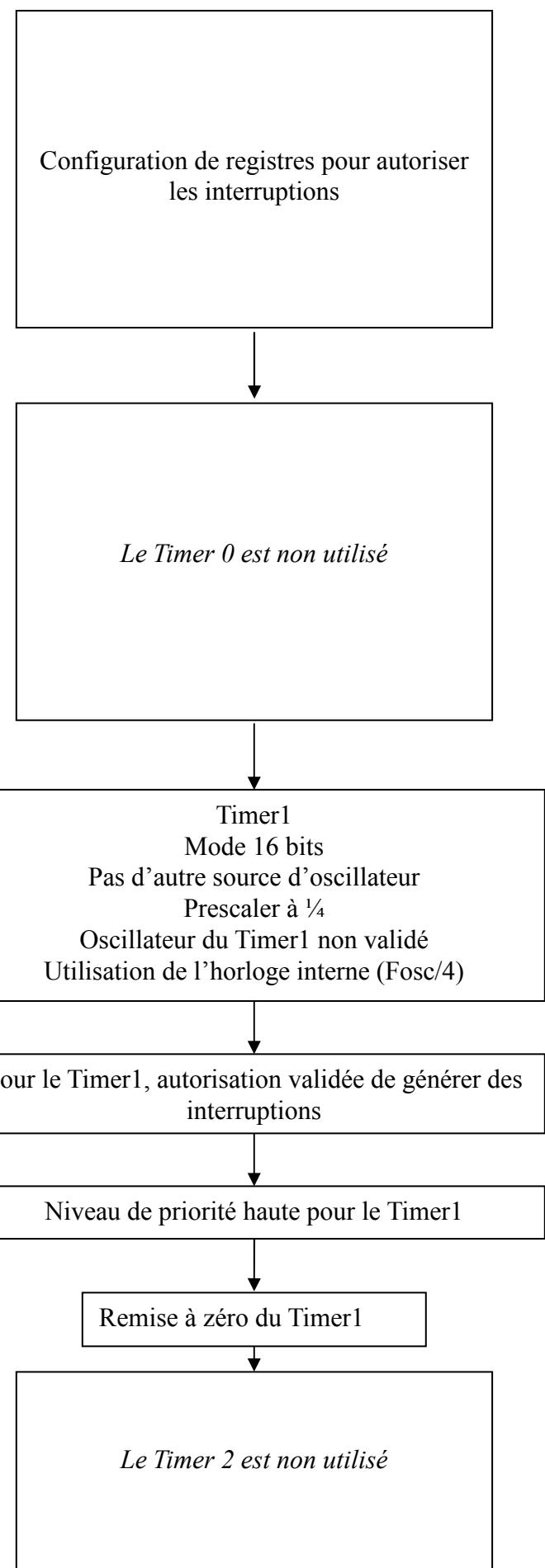
INTCON2bits.INTEDG0 = 1; // interruption sur front montant de INT0
INTCONbits.INT0IE = 1;        // Valide interruption sur INT0
// interruption INT2
INTCON2bits.INTEDG2 = 1; // interruption sur front montant de INT2
INTCON3bits.INT2IP = 1;       // INT2 priorité haute
INTCON3bits.INT2IE = 1;        // Valide interruption sur INT2

***** Init TIMER 0 *****
T0CONbits.T08BIT = 0;
T0CONbits.T0CS = 0;
T0CONbits.T0SE = 1;
T0CONbits.PSA = 1;           //Prescaler OFF/ON (1 OFF / 0 ON)
T0CONbits.TOPS2 = 0;
T0CONbits.TOPS1 = 0;          //Prescaler
T0CONbits.TOPS0 = 0;
INTCONbits.TMROIE = 0;        // autorise les interruptions avec le timer 0 (0:NON; 1:OUI)
TMROL = 0x0;                  // reset le timer0
TMROH = 0x0;
T0CONbits.TMROON = 0;

***** Init TIMER 1 *****
T1CONbits.RD16 = 1;           // mode R/W 16bits
T1CONbits.T1RUN = 0;          // autre source d'oscillateur
T1CONbits.T1CKPS0 = 0;         // prescaleur de 1 à 1/8
T1CONbits.T1CKPS1 = 1;
T1CONbits.T1OSCEN = 0;         // Timer1 oscilator off
T1CONbits.TMR1CS = 0;          // source du timer1 sur Fosc/4
// interruption sur TIMER1
PIE1bits.TMR1IE = 1;          // autorise les interruptions avec le timer 1 (0:NON; 1:OUI)
IPR1bits.TMR1IP = 1;           // Timer1 priorité
TMR1H = 0x00;                  // reset le timer1
TMR1L = 0x00;
T1CONbits.TMR1ON = 1;

***** Init TIMER 2 *****
T2CONbits.T2OUTPS3 = 0;
T2CONbits.T2OUTPS2 = 0;
T2CONbits.T2OUTPS1 = 0;
T2CONbits.T2OUTPS0 = 0;
T2CONbits.T2CKPS1 = 0;
T2CONbits.T2CKPS0 = 0;
PIE1bits.TMR2IE = 0;           // N'autorise pas les interruptions avec le timer 2
IPR1bits.TMR2IP = 1;

```



Annexe DC - Programme PIC User.C

```

TMR2 = 0x00;           // reset le timer2
PR2 = 0;
T2CONbits.TMR2ON = 0;

/***************** Init TIMER 3 *****/
/** But : obtenir un signal basse fréquence*/
T3CONbits.RD16 = 1;    // mode R/W 16bits
T3CONbits.T3CCP2 = 0;  // CCP module
T3CONbits.T3CCP1 = 0;  // CCP module
T3CONbits.T3CKPS1 = 0; // prescaleur a 1:1
T3CONbits.T3CKPS0 = 0;
T3CONbits.T3SYNC = 0;  // Timer3 external Clock (ignoré si TMR3CS=0)
T3CONbits.TMR3CS = 0;  // source du timer1 sur Fosc/4
// interuption sur TIMER1
PIE2bits.TMR3IE = 1;   // autorise les interruptions avec le timer 3 (0:NON; 1:OUI)
IPR2bits.TMR3IP = 1;   // Timer3 priorité
TMR3L = 0;             // reset le timer3
TMR3H = 0;
T3CONbits.TMR3ON = 1;

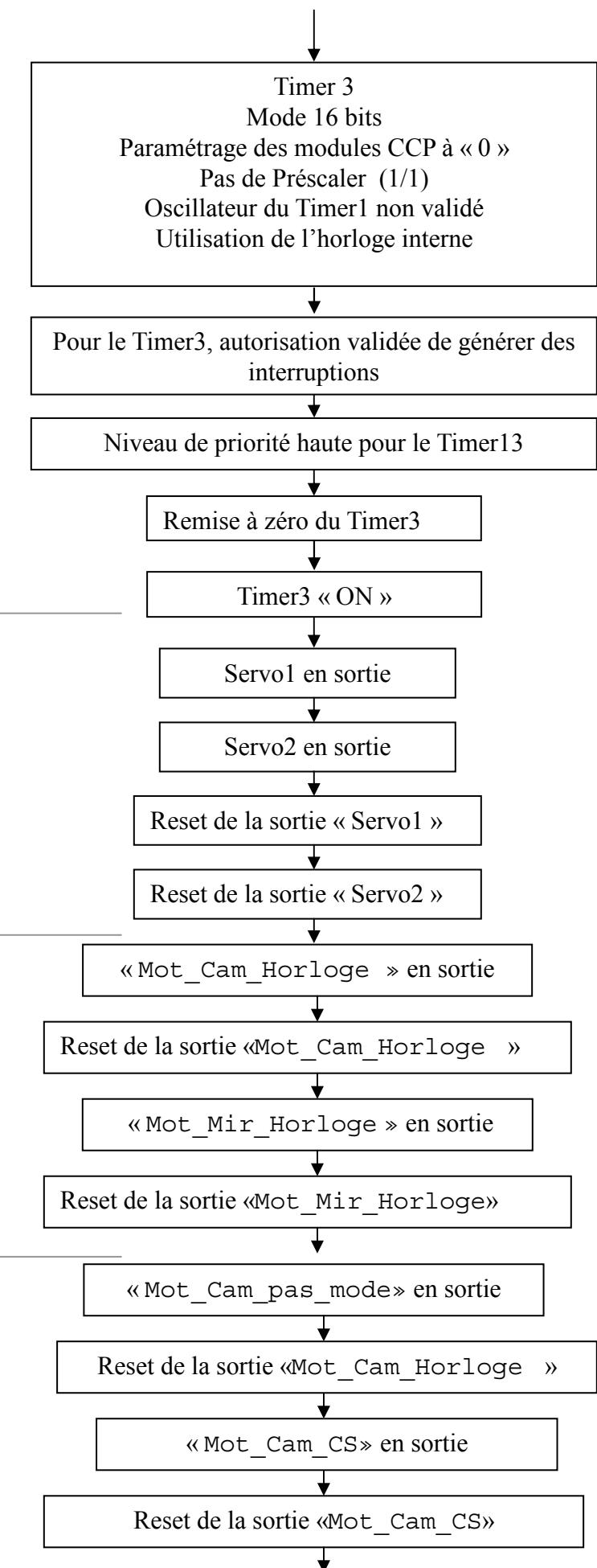
/***************** Init Servos *****/
servo1_tris = 0;
servo2_tris = 0;
servo1=0;
servo2=0;

/***************** Init Horloge Moteur *****/
Mot_Cam_Horloge_tris = 0;
Mot_Cam_Horloge = 0;

Mot_Mir_Horloge_tris = 0;
Mot_Mir_Horloge = 0;

/***************** Init Moteur Cam *****/
Mot_Cam_pas_mode = 0;
Mot_Cam_pas_mode_tris = 0;
Mot_Cam_CS = 0;
Mot_Cam_CS_tris = 0;

```



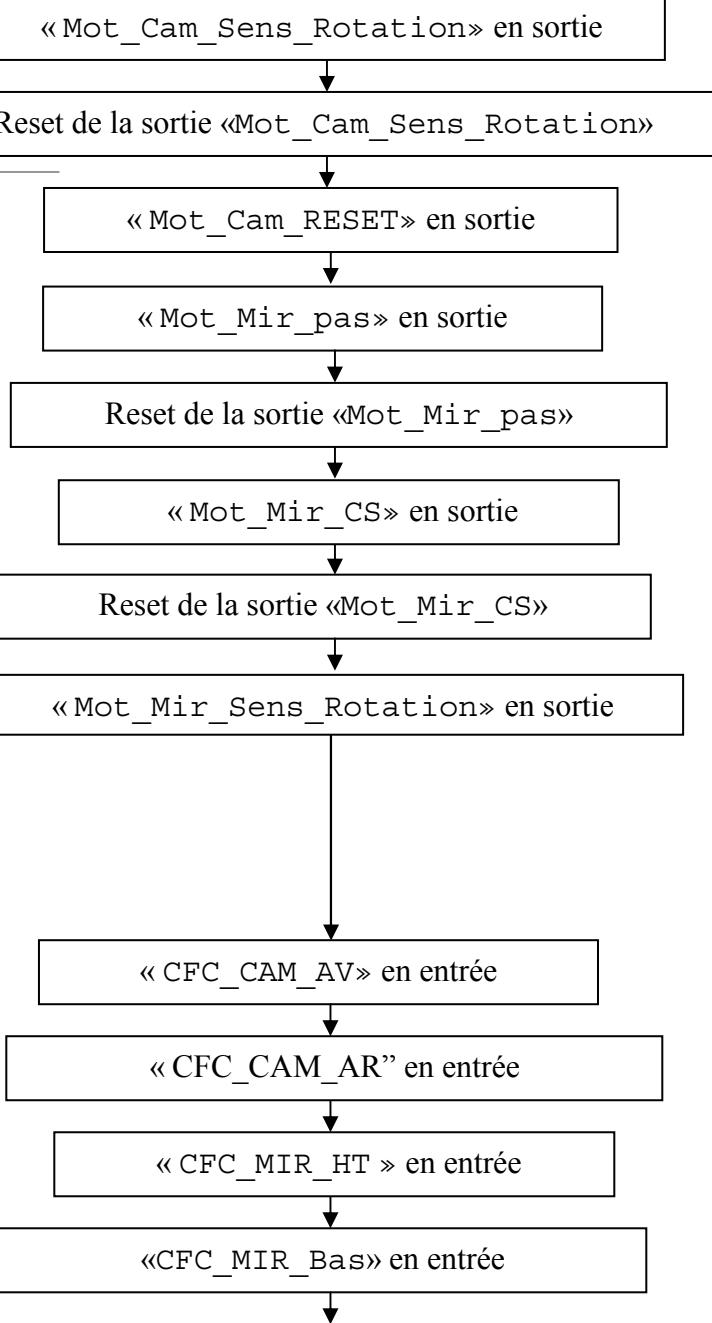
```
Mot_Cam_Sens_Rotation = 0;
Mot_Cam_Sens_Rotation_tris = 0;
Mot_Cam_RESET_tris = 0;
```

```
***** Init Moteur Mir *****
```

```
Mot_Mir_pas_mode = 0;
Mot_Mir_pas_mode_tris = 0;
Mot_Mir_CS = 0;
Mot_Mir_CS_tris = 0;
Mot_Mir_Sens_Rotation = 0;
Mot_Mir_Sens_Rotation_tris = 0;
Mot_Mir_RESET_tris = 0;
```

```
***** Init Capteur Fin Course*****
```

```
CFC_CAM_AV_tris = 1;
CFC_CAM_AR_tris = 1;
CFC_MIR_HT_tris = 1;
CFC_MIR_Bas_tris = 1;
```



Annexe DC - Programme PIC User.C

```
*****  
* Function:      void ProcessIO(void)  
*  
* PreCondition: None  
*  
* Input:        None  
*  
* Output:       None  
*  
* Side Effects: None  
*  
* Overview:     This function is a place holder for other user routines.  
*                 It is a mixture of both USB and non-USB tasks.  
*  
* Note:         None  
*****/
```

```
void ProcessIO(void)  
{  
  
    // User Application USB tasks  
    if((USBDeviceState < CONFIGURED_STATE) || (USBsuspendControl==1)) return;  
  
    //respond to any USB commands that might have come over the bus  
    ServiceRequests();  
  
}//end ProcessIO
```

```
*****  
* Function:      void ServiceRequests(void)  
*  
* PreCondition: None  
*  
* Input:        None  
*  
* Output:       None  
*  
* Side Effects: USB traffic can be generated  
*****
```

Annexe DC - Programme PIC User.C

```

/*
* Overview: This function takes in the commands from the PC from the
* application and executes the commands requested
*
* Note: None
***** */
void ServiceRequests(void)
{
    BYTE index;

    //Check to see if data has arrived
    if(!USBHandleBusy(USBGenericOutHandle))
    {
        //if the handle is no longer busy then the last
        //transmission is complete

        counter = 0;

        INPacket.CMD=OUTPacket.CMD;
        INPacket.len=OUTPacket.len;

        //process the command
        switch(OUTPacket.CMD)
        {
            case READ_VERSION:
                //dataPacket._byte[1] is len
                INPacket._byte[2] = MINOR_VERSION;
                INPacket._byte[3] = MAJOR_VERSION;
                counter=0x04;
                break;

            case CMD_MOTEUR_CAMIR:
                {
                    if (CFC_MIR_HT == 0)
                    {
                        val_tmr3H_IR = OUTPacket._byte[1];
                        val_tmr3L_IR = OUTPacket._byte[2];
                        nb_tour = OUTPacket._byte[3];
                        Mot_Cam_pas_mode = OUTPacket._byte[4];
                        Mot_Cam_Sens_Rotation = OUTPacket._byte[5];
                        Mot_Cam_CS = OUTPacket._byte[6];
                        mode_tour = OUTPacket._byte[7];
                        choix_cam = OUTPacket._byte[8];
                        temp = 0;
                    }
                }
                break;
        }
    }
}

```

Dans le protocole USB, cas de lecture de la trame de commande du moteur de la caméra UV-Visible.

Vérification de la position de la caméra UV et miroir en position haute ?

Non

Oui

Lecture des poids forts du timer3 (permet de définir la fréquence du signal d'horloge de commande de la caméra IR. byte [1])

Lecture des poids faibles du timer3 (permet de définir la fréquence du signal d'horloge de commande de la caméra IR. byte [2])

Lecture du nombre de tours (que doit exécuter le moteur). byte [3]

Lecture de mode de rotation du moteur (pas à pas ou demi pas). byte [4]

Lecture du sens de rotation de la caméra (Avance ou recul de la caméra IR). byte [5]

Lecture de la valeur du Chip Select (sélection du boîtier de commande des moteurs byte [6])

Lecture du mode de fonctionnement (mode pas à pas ou mode tour). byte [7]

Lecture du choix de la caméra. byte [8]

Fin de trame par tmp = 0

Annexe DC - Programme PIC User.C

```

case CMD_SERVO:
{
    val_tmr1H = OUTPacket._byte[1];
    val_tmr1L = OUTPacket._byte[2];
    num_servo = OUTPacket._byte[3];
    cycle = 0;
}
break;

case CMD_MOTEUR_CAMUV:
{
    if (CFC_CAM_AV == 0)
    {
        val_tmr3H_UV = OUTPacket._byte[1];
        val_tmr3L_UV = OUTPacket._byte[2];
        nb_tour = OUTPacket._byte[3];
        Mot_Mir_pas_mode = OUTPacket._byte[4];
        Mot_Mir_Sens_Rotation = OUTPacket._byte[5];
        Mot_Mir_CS = OUTPacket._byte[6];
        mode_tour = OUTPacket._byte[7];
        choix_cam = OUTPacket._byte[8];
        temp = 0;
    }
}
break;

case RESET:
    Reset();
    break;

default:
    Nop();
    break;
}//end switch()
if(counter != 0)
{
    if(!USBHandleBusy(USBGenericInHandle))
    {
        USBGenericInHandle = USBGenWrite(USBGEN_EP_NUM, (BYTE*)&INPacket, counter);
    }
}//end if

//Re-arm the OUT endpoint for the next packet
USBGenericOutHandle = USBGenRead(USBGEN_EP_NUM, (BYTE*)&OUTPacket, USBGEN_EP_SIZE);
}//end if

}//end ServiceRequests
***** */

```

