Projet : C8051F930 avec oscillateur à 20MHz

L'objectif est de piloter un moteur pas à pas possédant quatre phases, commandées par quatre broches du microcontrôleur. Entre le moteur et le microcontrôleur un étage de puissance est intercalé afin de délivrer le courant nécessaire dans les inducteurs. Les signaux de commande des phases en sorties du MCU doivent varier périodiquement (période T = 2.4ms).

Dans ce projet, je travaille théoriquement sur la partie de programmation avec le microcontrôleur C8051F930 car je n'ai pas de ce Chip pour tester l'algorithme. Donc, je ne peux pas vérifier sur le microcontrôleur si le résultat est correct ou incorrect.

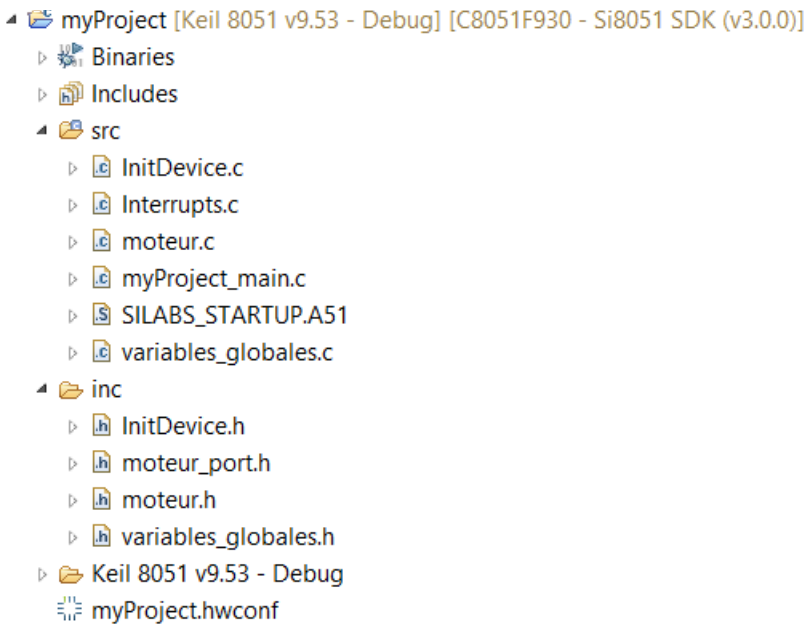
Pour résoudre ce problème, je réalise en testant d'abord ce projet sur le Chip ATMEL AT89C52 (base de 8051) avec l'oscillateur à 20MHz à l'aide de 2 logiciels : Keil C (code), Proteus (simulation). Puis, j'applique le même algorithme sur C8051F930. Dans ce rapport, je vais extraire les images sur le simulateur Proteus pour montrer le résultat.

I. Organisation des librairies du projet

Pourquoi on doit organiser bien le code ?

- ⇨ Pour diviser le travail (partie indépendant et dépendant le hardware)
- ⇨ Pour modifier facilement quand il y a quelque chose change.
- ⇨ Pour utiliser les libraires quand on en a besoin.

Dans ce projet, je travaille comme structure suivante :



Les libraires utilisées :

- « InitDevice » il y a 2 fichiers : InitDevice.h et InitDevice.c

  Je n'utilise pas le CONFIGURATOR mais travaille avec le code. Les paramètres initials du Microcontrôleur sont tous définis dans cette libraire.

- « Interrupts » il y a 1 fichiers : Interrupts.c
  Cette libraire gère tous les interruptions dans ce projet.
- « variables_globales » il y a 2 fichiers : variables_globales .h et variables_globales .c
  Cette libraire est responsable des variables globales
- « moteur » il y a 3 fichiers : moteur_port.h , moteur.h et moteur .c
  Cette libraire gère tous les fonctions qui concernent le moteur.

II. Programmation

1. Etap1 : Gestionnaire de tâche et timer0

a. Librairie « InitDevice ».

Au de démarrage du Microcontrôleur, En dépendant de chaque microcontrôleur on doit configurer normalement les paramètres initiales: Systèmes clock – diviseur, Watchdog, Ports, Timer. Ces paramètres sont déclarés et initialisés dans une libraire « InitDevice » et dans ce projet, on intéresse :

- Au système clock (on utilise quel diviseur pour la fréquence préfère),
- Aux ports (on utilise quelle(es) porte(es) ? quel broche ?) (on va configurer dans la libraire « Moteur »)
- Aux Timers (on utilise quel timer ? pour un compteur ou temporisateur ?)
- À la permission les interruptions

La libraire « InitDevice» est constitué par 2 fichiers:

initDevice.h : prototype des fonctions utilisées (code indépendant du matériel) :

```
extern void initDevice(void);                // pour appeler tous les fonctions initiales.
extern void configuration_timer0(void);      // configuration du timer 0
extern void configuration_timer1(void);      // configuration du timer 1
extern  void permetion_interrupt(void);      // permissions des interruptions
```

initDevice.c : programmations des fonctions (code indépendant du matériel).

- La fonction InitDevice() pour appeler tous les fonctions initiales dans la libraire « initDivice »

```
extern void initDevice(void){
  permetion_interrupt();
  configuration_timer0();
  configuration_timer1();
}
```

- La fonction *configuration_timer0()* pour configurer les paramètres du Timer 0.
  (Voir explication dans la partie III. Questions)

```
extern void configuration_timer0(void){
        CKCON &= ~0x0F;          //  fsys/12 avec SCA1:SCA0 = 00
```

```
        TMOD &= ~0x0F;          // mettre les bits poids faible du regitre TMOD timer0 a 0
        TMOD |= 0x02;           // mode 2(8 bits auto reload comme un temporaseur)
                                // => [10] pour les 1 et 0

        TL0= 0x06;              // valeur init: 06(H) voir dans l'explication
        TH0 = 0x06;             // il commence a 6 jusqu'a 255 pour remplire 8 bits

        TCON_TR0 = 1;           // lancer timer 0
}
```

- Il faut écrire une fonction *permetion_interrupt()* qui permettre les interruptions lorsque un overflow.

```
extern void permetion_interrupt(void){      // interruption
        TCON_IT0 =1;                // permission de l'interruption timer 0
        IE_ET0 =1;                  // du Timer0 est enable

        IE_ES0 =1;                  // du UART0 est enable

        IE_EA = 1;                  // EA =1 permission tous les interruptions
}
```

Pour les utiliser, il faut inclure la libraire « InitDevice » dans le programme principal :

```
#include "InitDevice.h"
```

b. Librairie « Interrupts ».

Dans le programme principal, on attend un overflow et s'il y a un overflow, il appelle une interruption. Pour facilement d'organiser, il faut écrire une librairie « Interrupts » qui gère tous les fonctions d'interruption. Cette libraire a un seul fichier « interrupts.c » et dedans, on programme 3 fonctions d'interruption qui sont automatiquement appelées par l'ISR lorsque un overflow :

- Interruption du timer 0

```
INTERRUPT (TIMER0_ISR, TIMER0_IRQn){ // une base de temps de 250 cycles = 150us

        TCON_TF0 = 0;           // etteint le flage d'interruption du timer0
        new_task = 1;           // activer le scheduler et lancer une tache
}
```

- Interruption du timer 2

```
INTERRUPT (TIMER2_ISR, TIMER2_IRQn){

        TMR2CN_TF2H = 0;        // flag overflow du timer 2
        changePhase();          // pour changer d'état de la phase.
}
```

Il y a encore une autre interruption, c'est l'interruption du Timer 1 qui sert créer la vitesse de Baud pour l'UART0.

On remarque que la priorité d'interruption du timer 0 est l'ordre de 1. Pour les étapes suivant, on utilisera encore Timer1 et Timer2, alors, Donc, Timer 0 > Timer 1 (UART) > Timer 2.

On a toujours les règles pour la priorité :

Si 2 événements arrivent en même temps :
L'interruption de niveau le plus élevé s'exécute puis celle de niveau inférieur.
Si un événement arrive pendant l'exécution d'une routine d'IT :
- 1 interruption de niveau inférieur ne peut pas interrompre une interruption de niveau supérieur.

- 1 interruption de niveau supérieur peut éventuellement interrompre une interruption de niveau inférieur.

c. Ordonnanceur-Scheduler

Pourquoi utilise ordonnanceur-scheduler ?

- Le programme du microcontrôleur est scinde en taches.
- L'ordonnanceur pilote l'exécution de chacune des taches.

Comment ?

On utilise un Timer0 qui sert de base de temps de 9ms et chaque interruption active l'exécution d'une étape dans le programme principal.

Dans ce cas, on a besoin 4 taches ce qui signifie que la tâche numéro x sera exécutée toutes les 36ms, on écrit une fonction *void scheduler(void)* avec son prototype au début de programme principal :

```
//-----------------------------------------------------------------
// prototype
//-----------------------------------------------------------------
void scheduler(void);

//-----------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------
void main(){

        initDevice();                   // initialisations du microcontrôleur

        while (1){
                while(new_task==0);     // polling sur le flag de la base de temps
                                        // Attente activation du gestionnaire de tache
                new_task=0;             // Lorsque finir l'interruption, remettre le flag = 0
                scheduler();            // lancer la function scheduler()
        }
}

// Chaque tache de scheduler prend une duree de temps de 60*150us = 9ms
```

```
void scheduler(void){                    // gestionnaire des taches

        switch(scheduler_ct) {           // ordonnanceur comporte 4 taches
                case 0: {                //tache 1 => faire quelque chose VOIR EPAPE 3
                        break;
                }
                case 60: {                //tache 2=> faire quelque chose
                        break;
                }
                case 120: {               //tache 3=> faire quelque chose
                        break;
                }
                case 180: {               //tache 4=> faire quelque chose
                        break;
                }
                default: break;
        }

        scheduler_ct++;                   // basculement d'une tache vert une autre
        if(scheduler_ct==240) scheduler_ct=0;       // après la tache 4, la tâche sera traité tache 0
}
```

On utilise la librairie « Variables Globales » pour la déclaration et l'initialisation des variables, cette librairie contient 2 fichiers :

variables_globales.h : déclaration de *scheduler_ct* et *new_task et les autres variables.*

```
#ifndef VARIABLES_GLOBALES_H_
#define VARIABLES_GLOBALES_H_

// les variables pour l'ordonnanceur
extern xdata unsigned int  scheduler_ct;
extern bit new_task;

// la variable pour garder l'etat des phases de sortie
extern xdata unsigned int  etat_phase ;

// la variables contient valeur recu de UART0
extern xdata unsigned char val;

#endif /* VARIABLES_GLOBALES_H_ */
```

variables_globales.c : initialisation de *scheduler_ct, new_task et les autres variables*

```
// les variables pour l'ordonnanceur
extern xdata unsigned int  scheduler_ct;
extern xdata bit new_task=0;

// la variable pour garder l'etat des phases de sortie
extern xdata unsigned int  etat_phase ;

// la variable contient valeur recu de UART0
extern xdata unsigned char val=6;
```

Ils sont inclus dans le programme principal :

```
#include "variables_globales.h"
```

2. Étape 2 : libraire « moteur »

La libraire « Moteur » est constitué par 3 fichiers :

Moteur_port.h : déclaration et macro de configuration des I/Os (code dépendant du matériel)

Il faut configurer que les bits les 4 bits poids faible de P1 sont en PUSH PULL et que le bit 3 de P0 est en OPEN DRAIN car le microcontrôleur va sortir les signaux sur le port P1 et entrer l'état du Bouton poussoir.

```
#ifndef MOTEUR_PORT_H_
#define MOTEUR_PORT_H_

// Signaux sorties
sbit  S1 = P1^0;        // signal sorti sur le broche P1.0
sbit  S2 = P1^1;        // signal sorti sur le broche P1.1
sbit  S3 = P1^2;       // signal sorti sur le broche P1.2
sbit  S4 = P1^3;       // signal sorti sur le broche P1.3

//les 4 bits poids faible de P1 sont en PUSH PULL
#define P1MDOUT |= 0x0F;

// Bouton poussoir
sbit BUTTON = P0^2;

//le bit 3 de P0 est en OPEN DRAIN
#define P0MDOUT &= ~0x04;

// Définition des phases
#define Phase1;                 S1 =0;S2 =0;S3 =1;S4 =1;    // la phase 1 est 0011
#define Phase2;                 S1 =1;S2 =0;S3 =0;S4 =1;    // la phase 2 est 1001
#define Phase3;                 S1 =1;S2 =1;S3 =0;S4 =0;    // la phase 3 est 1100
#define Phase4;                 S1 =0;S2 =1;S3 =1;S4 =0;    // la phase 4 est 0110

#endif
```

Moteur.h : prototype des fonctions utilisées (code indépendant du matériel).

```
#ifndef MOTEUR_H_
#define MOTEUR_H_

// initialisation du moteur
void InitMoteur(void);

// change le phase chaque une base de temps du Timer 2
void changePhase(void);

// arrêt ou démarrage du moteur
void demarage_arret_Moteur(void);
```

```
#endif
```

Moteur.c : programations des fonctions (code indépendant du matériel).

- **Fonction d'initialisation du moteur qui configure le timer 2** (voir explication dans la partie III)

```
void InitMoteur(void){              // initialisation du moteur

  CKCON &= ~0x30;        // T2ML = 0 et T2MH=0, la fréquence d'incrémentation
                         //dépend de T2XCLK
  TMR2RLL = 0x18;        // valeur reload: FC18 (H)
  TMR2RLH = 0xFC;
  TMR2L = 0x18;          // valeur init: FC18 (H)
  TMR2H = 0xFC;

  TMR2CN = 0x04;         // 16bits auto-reload et f =fsys/12 avec T2XCLK = 00
                         // Lancer timer2 = moteur tourne au démarrage du microcontrôleur
  IE_ET2 = 1;            // Timer 2 Interrupt Enable
}
```

- **ISR du timer 2 qui modifie les broches P1.0, P1.1, P1.2 et P1.3.**

La fonction suivante permet de changer en circulant l'état des signaux sortis pour chaque base de temps de 600μs. Apres la phase 4, l'état phase retourne la phase 1. Lorsqu'il y a un overflow (interruption), cette fonction est appelée.

```
void changePhase(void){          // change le phase chaque une base de temps du Timer 2

        switch(etat_phase) {
            case 0: {
                        Phase1;   // sortie phase 1
                        break; }
            case 1: {
                       Phase2;     // sortie phase 2
                    break;}
            case 2: {
                     Phase3;       // sortie phase 3
                     break;}
            case 3: {
                    Phase4;        // sortie phase 4
                    break;}
        }
   etat_phase++;                    // basculement vers une autre phase
   if(etat_phase==4)  etat_phase=0;       // après la phase 4, il retourne la phase 1
}
```

- **Fonction d'arrêt et de démarrage du moteur.**

Pour changer d'état de fonctionnement du moteur, on effectue un changement sur le Timer 2. Le timer 2 est lancé => les signaux est sortis => Moteur fonctionne

Le timer 2 est éteint => les signaux ne sont plus sortis => Moteur s'arrête.
Alors, on bascule le bit TR2 du Timer 2 lorsqu'il y a un appel la fonction suivante :

```
void demarage_arret_Moteur(void){          // Pour arrêt et démarrage du moteur

        TMR2CN_TR2 = !TMR2CN_TR2;    // basculement d'état du TMR2CN_TR2
                                     // => stopper ou lancer le timer2
}
```

Pour utiliser cette libraire dans le programme principal et interrupts.c, on doit inclure la libraire :

```
#include "moteur.h"
#include "moteur_port.h"
```

3. Étape 3 : utilisation de la libraire dans l'ordonnanceur

On travaille sur la tache 1 (case 0):

- initialiser le moteur (c'est-à-dire le timer 2) au démarrage du microcontrôleur.
- Examiner l'état d'un bouton poussoir connecté sur la broche P0.2.
- L'activation du bouton poussoir devra démarrer le moteur s'il est à l'arrêt ou le stopper si le moteur est en rotation.

a. initialisation du moteur.

Dans la fonction *Scheduler()* qui se place dans le programme principale, on déclare et initialise un flag *flag_init_moteur* qui présente si le moteur est initialisé. Pour l'objectif de vérifier que si le moteur a initialisé (flag =1), on n'a pas besoin de le faire, sinon on le fait.

```
static bit flag_init_moteur =0 ;
```

Dans la tache 1, on vérifie l'état du flag:

```
if(flag_init_moteur == 0){        // vérifier l'état du flag si le moteur n'initialise pas =>initialiser

    flag_init_moteur = 1;   // Mettre le flag = 1 pour la prochaine fois, il ne le réinitialise pas
    InitMoteur();           // Appel fonction initMoteur() pour initialiser le moteur
}
```

b. État d'un bouton poussoir et activation du bouton poussoir

On déclare d'abord dans le fichier moteur_port.h

```
// Bouton poussoir
sbit BUTTON = P0^2;
```

La proche P0.2 du microcontrôleur connecte au bouton poussoir est en entrée et active .

Dans la fonction *Scheduler()*, on déclare et initialise un flag *etat_button* qui présente si le bouton est appuyé puis lâché.

```
static bit etat_button=0;                    // les flags en statique
```

Dans la tâche 1 :

```
if(BUTTON == 0 && etat_button != 0){      //  lorsque le bouton est lâché,
        demarage_arret_Moteur();          //  appel cette fonction pour stopper ou démarrer le moteur
        etat_button = 0;                  // remettre le flag = 0 après de lâcher pour la prochaine fois
 }
else if(BUTTON != 0){                                  // si le bouton est appuyé
        etat_button = 1;                               // le flag = 1 pour la prochaine fois
 }
```

4. Étape 4 : UART0

Comme vous demandez d'effectuer la configuration de UART0 sur le Configurator mais dans ce cas, on n'a pas de Configurator, Pour résoudre ce problème, je configure avec le code à l'aider de datasheep de C8051F930. (Voir explication dans la partie III)

a. Initialisation UART0

```
extern void configuration_timer1 (void){  // configuration du timer 1

        CKCON &= ~0x0F;     //  fsys/12 avec SCA1:SCA0 = 00 meme frequence avec Timer 0

        TMOD &= ~0xF0;      // mettre les bits poids fort du registre TMOD a 0
        TMOD |= 0x02;       // mode 2(8 auto-reload bits comme un temporisateur)
                            // => [10] pour les bits 5 et 4
        TH1 = 0xA9;     // voir la détaille du Timer 1 pour la vitesse de Baud (l'erreur de 0.22%)

        SCON0_REN=1;        // permission de la réception UART0
        SCON0_TI =1;        // prêt à transmission UART0

        TCON_TR1 = 1;       // lancer timer 0
}
```

L'interruption UART0 est activée dans la fonction *permetion_interrupt()* de librairie « InitDevice »

```
        IE_ES0 =1;              // UART0 est active
```

b. Modifier la configuration du Timer 2

- **Configuration du timer 2**

Le timer 2 sert actuellement une base de temps de 100 μs. On change seulement la valeur initiale qui est contenu dans les registres TM2RLL, TM2RLH (auto rechargement), TM2L et TM2H avec le mode de 16 bits auto reload.

Alors f = 20.10^6 / 12 (Hz)

Donc, la durée pour un cycle d'instruction : T = 1/f = $\frac{12}{20.10^6}$ (s)

Pour 100μs, le nombre de cycle : Nb = $\frac{100.10^{-6}.20.10^6}{12} = 167$

On prend le mode de 16 bits du Timer 0 s'il y a un overf1ow de 0hFFFF => 0h0000, on a besoin de 65535 cycles de machine.

Le timer2 commence à compter avec la valeur :

65536 D – 167 D = 65369 D= FF59 H

La fonction InitMoteur_Base100us() permet de créer une base de temps de 100us avec son protocole est déclaré dans le fichier moteur.h.

```
void InitMoteur_Base100us(void){  // configurer le timer 2

        CKCON &= ~0x30;     // T2ML = 0 et T2MH=0, la fréquence d'incrémentation
                            //dépend de T2XCLK

        TMR2RLL = 0x59;      // valeur init: FFEF (H)
        TMR2RLH = 0xFF;
        TMR2L = 0x59;         // valeur init: FFEF (H)
        TMR2H = 0xFF;

        TMR2CN = 0x04;   // 16bits auto-reload et f =fsys/12 avec T2XCLK = 00
                         // Lancer timer2 = moteur tourne au démarrage du microcontrôleur
      IE_ET2 = 1;        // Timer 2 Interrupt Enable
}
```

Cette fonction est appelé dans l'ordonnanceur à la place de la fonction *InitMoteur()*.

- **Modifier l'interruption du Timer 2**

On écrit une fonction *changePhase2()* et la utilise a la place de la fonction *changePhase()*

suivante pour créer une base de temps de val*100 μs , chaque fois d'interruption du Timer 2, la valeur de etat_phase va monter jusqu'à 4*val (corresponde a une période), elle va retourner à 0.

```
void ChangePhase2(void){
        if(etat_phase==0){          // Au debut
                Phase1;              // sortir la phase 1
        }
        else if(etat_phase==val){    // Apres val fois d'interruption,
                                     // le compteur etat_phase = la valeur reçu par UART0
                Phase2;              // sortir la phase 2, la duree correspond a 100*val
        }
        else if(etat_phase==2*val){  // Apres une durée de 100*val
                Phase3;               // sortir la phase 3
        }
        else if(etat_phase==3*val){  // Apres une durée de 100*val
                Phase4;              // sortie la phase 4
        }
   etat_phase++;             // le compteur va monter chaque l'interruption
        if(etat_phase==4*val) etat_phase =0;  // Apres 4*val*100us = une periode,
                                              //  le compteur retoune a 0
}
```

Dans ce cas, on ne peut pas utiliser le `switch()` comme dans la fonction changePhase() car on doit comparer le nombre d'interruption avec la valeur reçu par UART0 pour trier et traiter pour chaque cas de val, 2*val,3*val, 4*val. Ici, la valeur de val n'est pas constante. Donc, on ne peut pas mettre dans le `switch()`.

c. Interruptions

Dans la libraire « variables_globales », on doit déclarer une variable de type *char* qui va contient la valeur reçu en décimal par l'UART0 :

variables_globales.h

```
// la variables contient valeur recu de UART0
  extern xdata unsigned char val;
```

variables_globales.c

```
// la variable contient valeur recu de UART0
xdata unsigned char val=6;
```

on initialise val = 6 pour que s'il n'y a pas d'information reçu, la signal sortie est période de 2.4 ms

```
INTERRUPT (UART0_ISR, UART0_IRQn) {

    static bit pass_R=0;
    if(SCON0_RI ==1){        // si il y un message qui vient de Terminal
                             // il attend la fin de reception, le flag monter automatiquement a 1
          SCON0_RI = 0;      // remttre valeur de RI0 = 1=>RI0 = 0
          if(pass_R==0){     // si le flag == 0, il est pret a recois encore des information
             val = SBUF0;    // la variable Val (un octet) prend valeur de SBUF0
             pass_R =1;      // Apres de commencer a recevoir l'information,
                             // il prend de temps pour finis la reception
          }
          else {             // Le PC tourne au programme pricipal pour
                             // faire un autre chose, en attendant cette reception finit
            pass_R=0;        // quand, il y a une autre interruption,
                             // il va verifier si la lecture est finis
           }
    }
}
```

III. Questions

1. Détailler la configuration des timers

Configuration du Timer 0(8bits auto-reload)

Dans ce projet le Timer0 est configuré comme un temporisateur qui sert de base de temps de 9ms, il faut d'abord calculer la valeur initiale à laquelle le timer0 commence à compter jusqu'au overflow :

On a : fsys = 20.10^6 Hz

On admet cette fréquence est divisée par n, on va choisir la valeur de n (4, 8,12 ou 48) et déduire la configuration du registre CKCON.

La fréquence après de la division par n : $f = 20.10^6 / n$ (Hz)

Donc, la durée pour un cycle d'instruction : $T = 1/f = \frac{n}{20.10^6}$ (s)

Pour 9 ms, le nombre de cycle : $nb = \frac{9.10^{-3}.20.10^6}{n} = \frac{180000}{n}$

Si on utilise le mode de 8 bits auto reload, ce n'est pas assez grand pour compter ce nb de cycle car :

$$\min(nb) = 180000/48 = 3750 > 2^8$$

Alors, on créer une base de temps de 250 cycles de machine (150 micro secondes) et la multiplie avec un nombre d'interruption pour avoir une base de temps de 9ms.

- Pour n =12, le nombre de cycle de machine : nb = 180000/12 = 15000=250*60

Donc, le nombre de l'interruption est égal à 60.

- Pour n = 48, le nombre de cycle de machine : nb = 180000/48 = 3750=250*15

Donc, le nombre de l'interruption est égal à 15

Avec le mode de 8 bits du Timer 0 s'il y a un overf1ow de 0hFF => 0h00, on a besoin de 256 cycles de machine.
Donc, le timer 0 commence avec la valeur : 0x06 pour avoir 250 cycles de machine.

On peut déduire la configuration des registres du timer0:

- CKCON
  T0M = 0 et SCA [1 :0] = 00 car on divise fsys par 12. Dans le cas,on veut la divise par 48, SCA[1 0] =10
- TMOD
  C/T0 = 0 car timer0 est un temporisateur
  T0M [1 :0] = 10 car mode 2 de 8 bits rechargement
- TCON
  TR0 = 1 pour lancer timer 0
- TL0
  TL0 = 0x06 car ce registre contient la valeur initiale.
- TH0
  TH0 = 0x06 car ce registre contient la valeur rechargement

Configuration du Timer 1 (8bits auto reload)

Dans ce projet le Timer1 est configuré en mode 2, 8bits auto-reload pour générer la vitesse de Baud, il faut d'abord calculer la valeur initiale à laquelle le timer 1 commence à compter jusqu'au overflow :

```
CKCON &= ~0x0F;          // fsys/12 avec SCA1:SCA0 = 00
```

```
TMOD &= ~0xF0;   // mettre les bits poids faible du registre TMOD timer1 a 0
TMOD |= 0x20;    // mode 2(8 auto-reload bits comme un temporisateur)
                 // => 10 pour les bits 5 et 4

TH1 = 0xA9;              // voir la détaille du Timer 1 pour la vitesse de Baud(l'erreur de 0.22%)

SCON0_REN=1;             // permission de la réception
SCON0_TI =1;             // prêt à transmission
SCON0_RI =1;             // prêt à réception

TCON_TR1 = 1;            // lancer timer 0
```

Apres le Datasheep de C8051F930, on a les formules pour calculer la valeur initiale du Timer 1 suivantes:

A) $\text{UartBaudRate} = \frac{1}{2} \times \text{T1\_Overflow\_Rate}$

B) $\text{T1\_Overflow\_Rate} = \frac{\text{T1}_{CLK}}{256 - \text{TH1}}$

On a : fsys = 20.10^6 Hz

La fréquence après de la division par 12 : f = 20.10^6 / 12 (Hz)

À partir de A, on a T1_Overflow_Rate = 9600.2 = 19200 bis/s

À partir de B, on peut déduire : TH1 =256 - $\frac{20.10^6}{12.19200}$ = 169 D = A9 H

Pour TH1 = 169 => UartBaudRate = 9578 bis/s

Il apparait l'erreur = (9600-9578)/9600 = 0.22%

65536 D – 180000/12 D = 50536 D= C568 H

En conclure, la configuration des registres du timer0:

- CKCON
  T1M = 0 et SCA [1 :0] = 00 car on divise fsys par 12 et l'utilise
- TMOD
  C/T1 = 0 car timer0 est un temporisateur
  T1M [1 :0] = 01 car mode 2 de 8 bits auto-reload
- TCON
  TR1 = 1 pour lancer timer 0
- TH1
  TH1 = 0xA9 car ce registre contient la valeur initiale

Configuration du Timer 2

Dans ce projet le Timer2 est configuré comme un temporisateur qui sert de base de temps de 600μs, il faut d'abord calculer la valeur initiale à laquelle le timer2 commence à compter jusqu'au overflow :

On a : fsys = 20.10^6 Hz

La fréquence entre Timer 2: f = 20.10^6 / n (Hz)

Avec n = 1 si on utilise le clock du système ou n = 12 si si T2ML de registre CKCON sont à 0, alors la fréquence de l'incrémentation dépend des bits T2XCLK[1 :0] = 00 de registre TM2RCN.

Alors f = f = 20.10^6 / 12 (Hz)

Donc, la durée pour un cycle d'instruction : T = 1/f = $\frac{12}{20.10^6}$ (s)

Pour 600μs, le nombre de cycle : Nb = $\frac{600.10^{-6}.20.10^6}{12} = 1000 > 2^8$

Donc, on prend le mode de 16 bits du Timer 0 s'il y a un overf1ow de 0hFFFF => 0h0000, on a besoin de 65535 cycles de machine.

Le timer2 commence à compter avec la valeur :

65536 D – 1000 D = 64536 D= FC18 H

On peut déduire la configuration des registres du timer2:

- CKCON
  T2ML = 0
- TMR2CN = 0x04
  T2SPLIT = 0 pour 16 bits auto reaload
  T2XCLK[1 : 0] = 00 pour f = fsys/12
  TF2LEN=0 et TF2CEN =0 pour ni capture ni IRQ sur over flow
  TR2 = 1 pour lancer timer 2
- TMR2RLL = 0x18 car ce registre contient 8bits poids faible de la valeur initiale
- TMR2RLH = 0xFC car ce registre contient 8bits poids fort de la valeur initiale
- TMR2L = 0x18 car ce registre contient 8bits poids faible de la valeur initiale
- TMR2H = 0xFC car ce registre contient 8bits poids fort de la valeur initiale

2. Quelle est la valeur maximale de T ?

La valeur de T dépend la valeur de val (ou x) qui est reçu par l'UART0,

Si on décale pour cette variable de type de char non signée, alors, sa valeur maximale est 255 et Tmax = 4*255*100 μs = 0.102 s.

Si on décale pour cette variable de type de char signée et fixe la base de temps de 100 μs, alors, sa valeur maximale est 127 et Tmax = 4*127*100 μs = 0.0508 s.

De plus, La valeur de T dépend la valeur d'une base de temps, c'est à dire que le temps pour une interruption est maximale => overflow apres 65536 cycles de machine => valeur initiale pour timer 2 =0.

En générale, T = 4*val*65536*n/fsys

Avec : val ∈ [0,255] non signe

n =1 ou 12

fsys = 20 MHz la fréquence du oscillateur

Alors, Tmax = 255*65536*12/20 (μs) = 10.027008 s

Qu'est-ce qui détermine la valeur minimale de T ?

Les facteurs influencent à la valeur minimale de T :

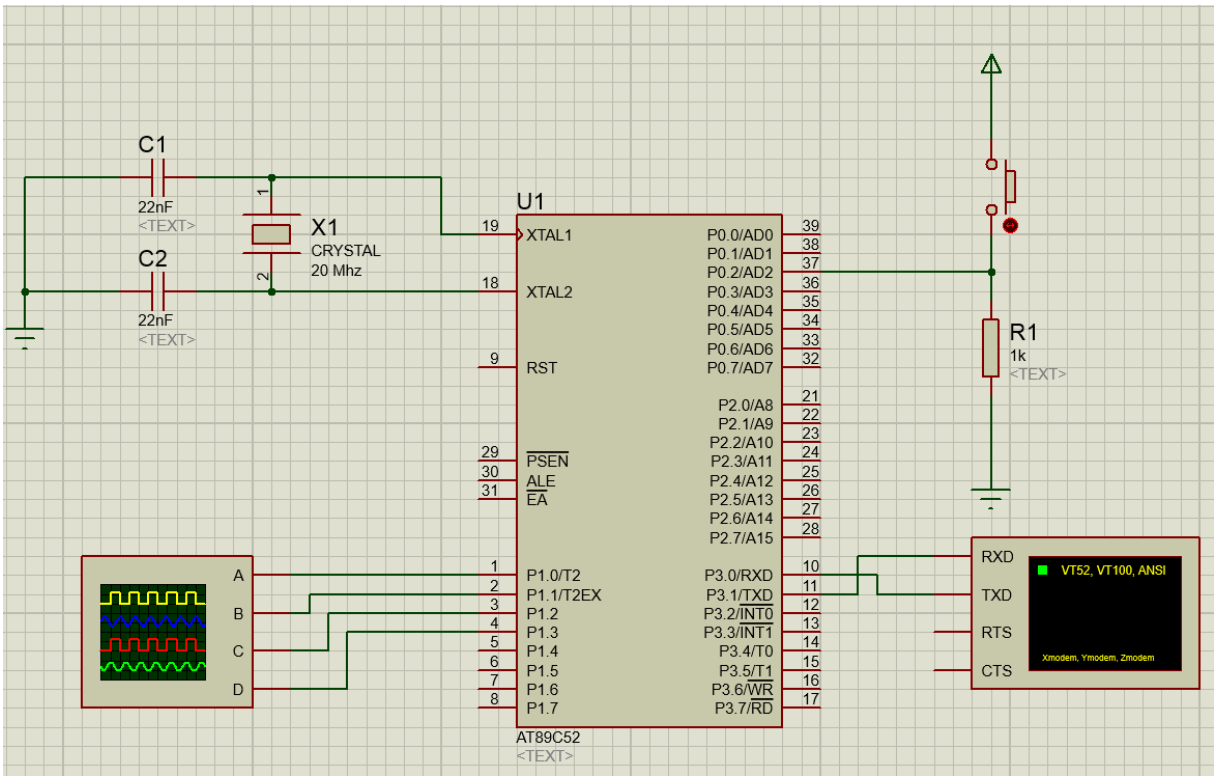
- Valeur reçu : val ∈ [0,255] non signe
- diviser le clock : n =1 ou 12
- la fréquence du oscillateur fsys = 20 MHz

IV. Résultats

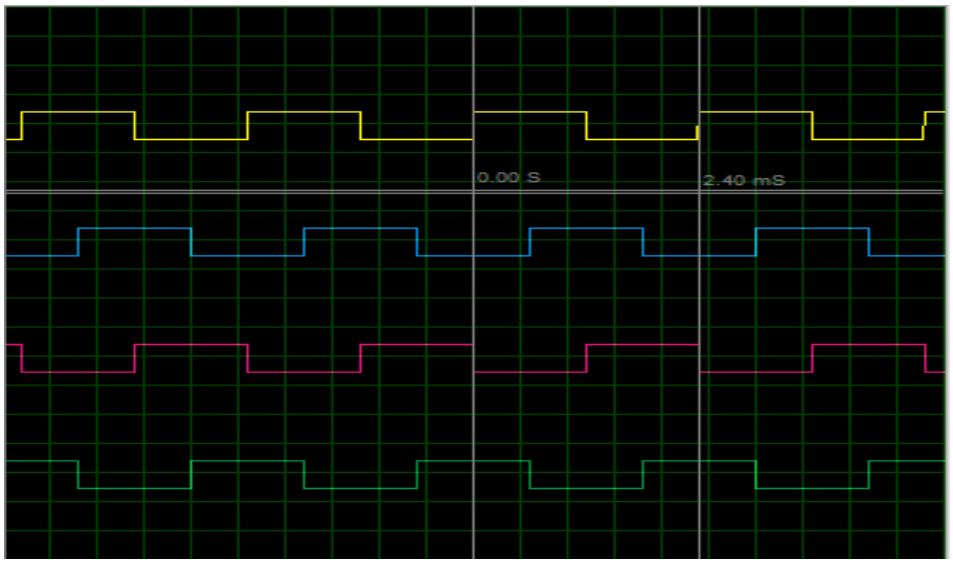
Dans cette partie, je présente les résultats que j'ai testés avec le simulink Proteus.

L'objectif de cette partie est la vérification l'algorithme appliqué sur C8051F930 est correct en appliquant sur AT89C52 (Base de 8051) :

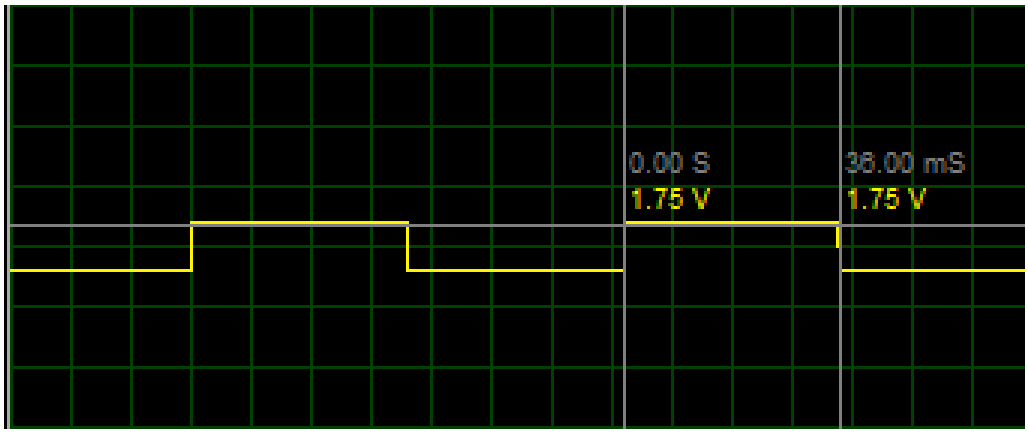
- Le montage :



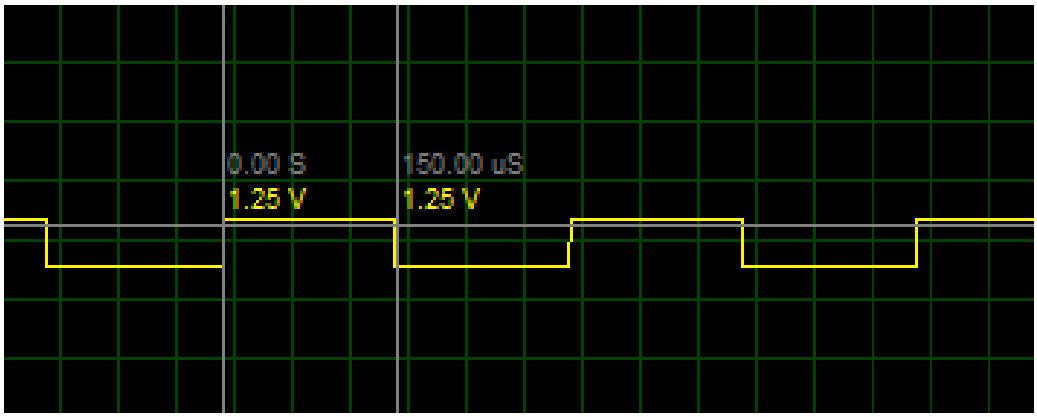
- Les signaux sortis avec la période de T = 2.4 ms



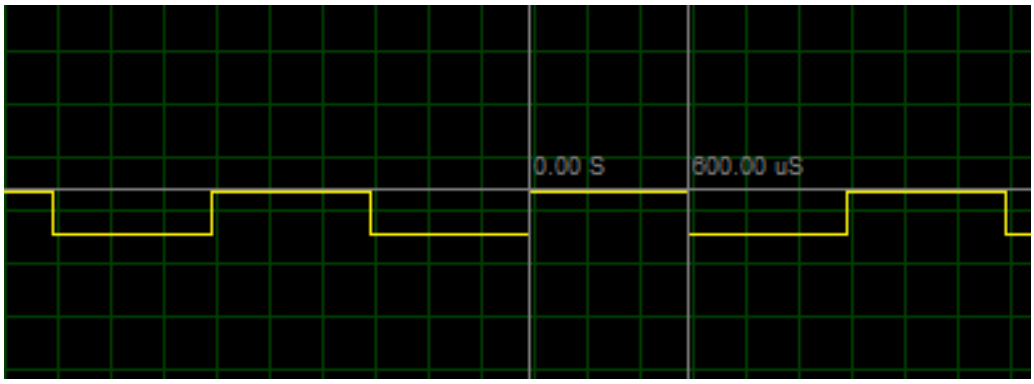
- La base de temps de 36ms du Timer 0 pour une tache



- La base de temps de 150us du Timer 0



- La base de temps de 600us du Timer 2



V. Annexe

- **Main.c**

```
//-----------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------
#include <SI_C8051F930_Register_Enums.h>
#include "InitDevice.h"
#include "variables_globales.h"
#include "moteur.h"
#include "moteur_port.h"

//-----------------------------------------------------------------------
// procotype
//-----------------------------------------------------------------------
void scheduler(void);

//-----------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------
void main(){

        initDevice();               // initialisations du microcontrôleur

        while (1){
                while(new_task==0);     // polling sur le flag de la base de temps
                                        // Attente activation du gestionnaire de tache
                new_task=0;             // Lorsque finir l'interruption, remettre le flag = 0
                scheduler();            // lancer la function scheduler()
        }

}


// chaque tache de scheduler prend une duree de temps de 60*150us = 9ms
void scheduler(void){                   // gestionnaire des taches
```

```
        static bit flag_init_moteur =0, etat_button=0;   // les flags en static

        switch(scheduler_ct) {              // ordonnanceur comporte 4 taches
                case 0: {                   //tache 1 => faire quelque chose
                        if(flag_init_moteur == 0){   // vérifier l'état du flag
                                                     // si le moteur n'initialise pas =>initialise le
                            flag_init_moteur = 1;    // Mettre le flag = 1 pour la prochaine fois,
                                                     // il ne le réinitialise pas
                                InitMoteur();        // initialiser le moteur
                        }

                    if(BUTTON == 0 && etat_button != 0){     //  lorsque le bouton est lâché,
                            demarage_arret_Moteur();         //  appel cette fonction pour stopper
                                                             //ou démarrer le moteur
                         etat_button = 0;                    // remettre le flag = 0 après de lâcher
                    }
                    else if(BUTTON != 0){                    // si le bouton est appuyé
                            etat_button = 1;                 // le flag = 1
                    }
                        break;
                }
                case 60: {                  //tache 2=> faire quelque chose
                        break;
                }
                case 120: {                 //tache 3=> faire quelque chose
                        break;
                }
                case 180: {                 //tache 4=> faire quelque chose
                        break;
                }
                default: break;
        }

        scheduler_ct++;                     // basculement d'une tache vert une autre
        if(scheduler_ct==240) scheduler_ct=0;       // après la tache 4, la tâche sera traité tache 0
}
```

- **Libraire « InitDevice »** il y a 2 fichiers : InitDevice.h et InitDevice.c

InitDevice.h

```
//=========================================================
// Prototype des fonctions
//=========================================================
#ifndef __INIT_DEVICE_H__             // s'il a deja declare,
#define __INIT_DEVICE_H__             // on a pas besion de faire encore

extern void initDevice(void);         // pour appeler tous les fonctions initiales.
extern void configuration_timer0(void);  // configuration du timer 0
extern void configuration_timer1(void);  // configuration du timer 1
extern  void permetion_interrupt(void);  // permissions des interruptions
```

```
#endif
```

InitDevice.c

```
//-----------------------------------------------------------------
// Déclaration et Initialisation les paramètres au démarrage
// du Microcontrôleur:
// 1. Fonction appelle les autres fonctions
// 2. Fonction pour la configuration du timer 0
// 3. Fonction pour la configuration du timer 1
//-----------------------------------------------------------------

// includes
#include <SI_C8051F930_Register_Enums.h>
#include "InitDevice.h"

// foncitons
extern void initDevice(void){          // pour appeler tous les fonctions initiales.
            permetion_interrupt();
            configuration_timer0();
            configuration_timer1();

}
extern void configuration_timer0(void){  // configuration du timer 0
            CKCON &= ~0x0F;                  //  fsys/12 avec SCA1:SCA0 = 00

            TMOD &= ~0x0F;  // mettre les bits poids faible du regitre TMOD timer0 a 0
            TMOD |= 0x02;      // mode 2(8 bits auto reload comme un temporaseur)
                               // => [10] pour les 1 et 0

            TL0= 0x06;      // valeur init: 06(H) voir dans l'explication
            TH0 = 0x06;     // il commence a 6 jusqu'a 255 pour remplir 8 bits

            TCON_TR0 = 1;   // lancer timer 0
}
extern void configuration_timer1(void){  // configuration du timer 1
            CKCON &= ~0x0F;            //  fsys/12 avec SCA1:SCA0 = 00 meme frequence avec Timer 0

            TMOD &= ~0xF0;    // mettre les bits poids fort du registre TMOD a 0
            TMOD |= 0x02;     // mode 2(8 auto-reload bits comme un temporisateur)
                              // => [10] pour les bits 5 et 4
            TH1 = 0xA9;       // voir la détaille du Timer 1 pour la vitesse de Baud(l'erreur de 0.22%)

            SCON0_REN=1;              // permission de la réception UART0
            SCON0_TI =1;              // prêt à transmission UART0

            TCON_TR1 = 1;             // lancer timer 0
}
extern  void permetion_interrupt(void){  // permissions des interruptions
                  TCON_IT0 =1;                // permission de l'interruption
                  IE_ET0 =1;                  // du Timer0 est enable
                  IE_ES0 =1;                  // du UART0 est enable
                  IE_EA = 1;                  // EA =1 permission tous les interruptions
}
```

- **Librairie « Interrupts »** il y a 1 fichier : Interrupts.c

Interrupts.c

```
//=========================================================
// Lorsqu'il y a un overflow, le systeme appele l'interruption
// 1. Interruption du Timer 0
// 2. Interruption du timer 2
//=========================================================

// includes

#include <SI_C8051F930_Register_Enums.h>
#include "InitDevice.h"
#include "variables_globales.h"
#include "moteur.h"
#include "moteur_port.h"

INTERRUPT (TIMER2_ISR, TIMER2_IRQn){

        TMR2CN_TF2H = 0;            // flag overflow du timer 2
   changePhase();             // pour changer d'état de la phase.
}

INTERRUPT (TIMER0_ISR, TIMER0_IRQn){ // une base de temps de 250 cycles = 150us

        TCON_TF0 = 0;               // etteint le flage d'interruption du timer0
        new_task = 1;              // activer le scheduler et lancer une tache
}

INTERRUPT (UART0_ISR, UART0_IRQn) {

        static bit pass_R=0;
        if(SCON0_TI ==1){   // fin de transmission
          SCON0_TI =0;
        }
        else if(SCON0_RI ==1){  // si il y un message qui vient de Terminal
                                // il attend la fin de reception, le flag monter automatiquement a 1
          SCON0_RI = 0;          // remttre valeur de RI0 = 1=>RI0 = 0
          if(pass_R==0){            // si le flag == 0, il est pret a recois encore des information
             val = SBUF0;   // la variable Val (un octet) prend valeur de SBUF0
             pass_R =1;      // Apres de commencer a recevoir l'information,
                                // il prend de temps pour finis la reception
          }
          else {             // Le PC tourne au programme pricipal pour faire un autre chose,
                             //  en attendant cette reception finit
            pass_R=0;     // quand, il y a une autre interruption, il va verifier si la lecture est finis
          }
        }
}
```

- **Libraire « variables_globales »** il y a 2 fichiers : variables_globales .h et variables_globales .c

variables_globales .h

```
//=========================================================
// Declaration les variables globales
//
//=========================================================

#ifndef VARIABLES_GLOBALES_H_
#define VARIABLES_GLOBALES_H_

// les variables pour l'ordonnanceur
extern xdata unsigned int  scheduler_ct;
extern bit new_task;

// la variable pour garder l'etat des phases de sortie
extern xdata unsigned int  etat_phase ;

// la variables contient valeur recu de UART0
extern xdata unsigned char val;

#endif /* VARIABLES_GLOBALES_H_ */
```

variables_globales .c

```
//=========================================================
// Declaration les variables globales
//
//=========================================================

// les variables pour l'ordonnanceur
xdata unsigned int  scheduler_ct;
bit new_task=0;

// la variable pour garder l'etat des phases de sortie
xdata unsigned int  etat_phase ;

// la variable contient valeur recu de UART0
xdata unsigned char val=6;
```

- **Libraire « moteur »** il y a 3 fichiers : moteur_port.h , moteur.h et moteur .c

moteur_port.h

```
//=========================================================
// Déclaration et macro de configuration des I/Os
// (code dépendant du matériel)
//=========================================================
#ifndef MOTEUR_PORT_H_
#define MOTEUR_PORT_H_
```

```
// Signaux sorties
sbit  S1 = P1^0;        // signal sorti sur le broche P1.0
sbit  S2 = P1^1;        // signal sorti sur le broche P1.1
sbit  S3 = P1^2;        // signal sorti sur le broche P1.2
sbit  S4 = P1^3;        // signal sorti sur le broche P1.3

//les 4 bits poids faible de P1 sont en PUSH PULL
#define P1MDOUT |= 0x0F;

// Bouton poussoir
sbit BUTTON = P0^2;

//le bit 3 de P0 est en OPEN DRAIN
#define P0MDOUT &= ~0x04;

// Définition des phases

#define Phase1;                 S1 =0;S2 =0;S3 =1;S4 =1;    // la phase 1 est 0011
#define Phase2;                 S1 =1;S2 =0;S3 =0;S4 =1;    // la phase 2 est 1001
#define Phase3;                 S1 =1;S2 =1;S3 =0;S4 =0;    // la phase 3 est 1100
#define Phase4;                 S1 =0;S2 =1;S3 =1;S4 =0;    // la phase 4 est 0110

#endif /* MOTEUR_PORT_H_ */
```

moteur.h

```
//=========================================================
// Prototype des fonctions
// (code dépendant du matériel)
//=========================================================

#ifndef MOTEUR_H_
#define MOTEUR_H_

void InitMoteur(void);              // initialisation du moteur
void changePhase(void);             // change le phase chaque une base de temps du Timer 2
void demarage_arret_Moteur(void);   // arrêt et démarrage du moteur
void InitMoteur_Base100us(void);
void ChangePhase2(void);

#endif /* MOTEUR_H_ */
```

moteur .c

```
#include <SI_C8051F930_Register_Enums.h>
#include "moteur_port.h"
#include "variables_globales.h"

void InitMoteur(void){              // initialisation du moteur

  CKCON &= ~0x30;       // T2ML = 0 et T2MH=0, la fréquence d'incrémentation
```

```
                    //dépend de T2XCLK
  TMR2RLL = 0x18;        // valeur init: FC18 (H)
  TMR2RLH = 0xFC;
  TMR2L = 0x18;           // valeur init: FC18 (H)
  TMR2H = 0xFC;

  TMR2CN = 0x04;         // 16bits auto-reload et f =fsys/12 avec T2XCLK = 00
                    // Lancer timer2 = moteur tourne au démarrage du microcontrôleur
  IE_ET2 = 1;           // Timer 2 Interrupt Enable
}

void changePhase(void){           // change le phase chaque une base de temps du Timer 2

          switch(etat_phase) {
              case 0: {
                        Phase1;    // sortie phase 1
                        break;}
              case 1: {
                        Phase2;          // sortie phase 2
                        break;}
              case 2: {
          Phase3;    // sortie phase 3
                        break;}

              case 3: {
                       Phase4;     // sortie phase 4
                       break;}
          }
    etat_phase++;                       // basculement vers une autre phase
    if(etat_phase==4)  etat_phase=0;       // après la phase 4, il retourne la phase 1
}

void demarage_arret_Moteur(void){// Pour arrêt et démarrage du moteur
          TMR2CN_TR2 = !TMR2CN_TR2;    // basculement d'état du TMR2CN_TR2
                        // => stopper ou lancer le timer2
}

void InitMoteur_Base100us(void){  // configurer le timer 2

        CKCON &= ~0x30;     // T2ML = 0 et T2MH=0, la fréquence d'incrémentation
                            //dépend de T2XCLK

        TMR2RLL = 0x59;        // valeur init: FFEF (H)
        TMR2RLH = 0xFF;
        TMR2L = 0x59;           // valeur init: FFEF (H)
        TMR2H = 0xFF;

        TMR2CN = 0x04;   // 16bits auto-reload et f =fsys/12 avec T2XCLK = 00
                         // Lancer timer2 = moteur tourne au démarrage du microcontrôleur
        IE_ET2 = 1;          // Timer 2 Interrupt Enable
}

void ChangePhase2(void){
          if(etat_phase==0){          // Au debut
                    Phase1;             // sortir la phase 1
```

```
        }
        else if(etat_phase==val){     // Apres val fois d'interruption,
                                      // le compteur etat_phase = la valeur recu par UART0
                Phase2;               // sortir la phase 2, la duree correspond a 100*val
        }
        else if(etat_phase==2*val){   // Apres une duree de 100*val
                Phase3;               // sortir la phase 3
        }
        else if(etat_phase==3*val){
                Phase4;
        }
  etat_phase++;                       // le compteur va monter chaque l'interruption
        if(etat_phase==4*val) etat_phase =0;  // Apres 4*val*100us = une période,
                                              //  le compteur retoune a 0
}
```