Berachem MARKRIA & Joshua LEMOINE - E3FI - 2I - Groupe 2

Projet GIG-Bot

Cours d'Architecture - avec EvalBot

Sommaire

Introduction

Le présent projet vise à étendre les capacités de l'EVALBOT en exploitant efficacement ses périphériques clés tels que LED(s), bumper(s), switch(s), et moteur(s). La validation du scénario repose sur l'utilisation de ces éléments. A noter que l'ajout d'autres périphériques est envisageable.

Dans ce cadre, notre groupe propose le "GIG-Bot", une solution innovante visant à remplacer les chiens d'intervention du GIGN lors de missions délicates. Le but est de créer un robot éclaireur capable d’ouvrir la voie à travers un fumigène tout en indiquant les divers obstacles rencontrés grâce à ses Leds.

Notre projet actuel représente une étape solide dans l'amélioration de l'EVALBOT. Nous reconnaissons également qu'il offre une base propice à des développements ultérieurs. Nous envisageons d'explorer des possibilités supplémentaires pour enrichir davantage les fonctionnalités et les performances du robot, ce qui contribuera à renforcer encore plus son impact et son utilité

Description du projet

Ce projet a été réalisé en binôme dans le cadre du cours d'Architecture en E3FI. Le code a été écrit en assembleur Cortex M3 (ARM M3) et a été déployé sur des robots EVALBOT (Stellaris, Texas instrument). Pour faciliter la collaboration et le partage du code, nous avons utilisé l'IDE Keil et un repository GitHub.

Le but de ce projet était d'étendre les capacités de l'EVALBOT en exploitant efficacement ses périphériques clés tels que les LED(s), les bumper(s), les switch(s), et les moteur(s). Nous avons validé le scénario en utilisant ces éléments. Il est à noter que l'ajout d'autres périphériques est envisageable.

Dans ce cadre, notre groupe a proposé le "GIG-Bot", une solution innovante visant à remplacer les chiens d'intervention du GIGN lors de missions délicates. Le but était de créer un robot éclaireur capable d'ouvrir la voie à travers un fumigène tout en indiquant les divers obstacles rencontrés grâce à ses Leds.

Notre projet actuel représente une étape solide dans l'amélioration de l'EVALBOT. Nous reconnaissons également qu'il offre une base propice à des développements ultérieurs. Nous envisageons d'explorer des possibilités supplémentaires pour enrichir davantage les fonctionnalités et les performances du robot, ce qui contribuera à renforcer encore plus son impact et son utilité.

Scénario

Le GIG-Bot est conçu pour être déployé lors d'interventions du GIGN, offrant une alternative technologique aux chiens d'interventions traditionnels. Le robot est équipé de fonctionnalités spécifiques qui lui permettent de naviguer et de réagir de manière autonome dans des environnements complexes.

Le déploiement du GIG-Bot lors d'interventions du GIGN présente plusieurs avantages, notamment en ce qui concerne le respect envers les chiens d'interventions traditionnels. En optant pour cette alternative technologique, on minimise les risques et les contraintes auxquels les chiens pourraient être exposés lors de missions délicates. Les fonctionnalités autonomes du robot permettent une navigation précise et réactive dans des environnements complexes, offrant ainsi une solution fiable et sécurisée pour les opérations du GIGN.

De plus, le GIG-Bot présente un avantage significatif en termes de sécurité dans des situations spécifiques, telles que l'exposition à des gaz potentiellement dangereux. Contrairement aux chiens d'interventions, le robot est capable d'intervenir efficacement en cas de présence de gaz, assurant

ainsi une intervention continue et minimisant les risques pour les agents humains. Cette capacité renforce la polyvalence du GIG-Bot, en en faisant un atout essentiel pour les opérations du GIGN, tout en garantissant le bien-être et la sécurité des animaux traditionnellement impliqués dans ces missions.

Composition du code :

Notre code est séparé en 5 fichiers distincts, 4 concernant les différents périphériques : leur configuration et diverses routines utiles. Et un Main comprenant le cœur du programme. Voici leur composition :

> Leds.s : comprends 6 routines :

- *LEDS_INIT* : initialise les deux leds
- *LEDS_ON* : allume les deux leds
- *LEDS_OFF* : éteint les deux leds
- *LEDS_SWITCH* : change l'état des deux leds
- *LED4_ON* : allume la led 4
- *LED5_ON* : allume la led 5

> Bumper.s : comprend 4 routines

- *BUMPERS_INIT* : initialise les deux bumpers
- *READ_BUMPER1* : lis l'état du bumper 1 et le compare avec 0 afin de savoir si il est enfoncé
- *READ_BUMPER2* : lis l'état du bumper 2 et le compare avec 0 afin de savoir si il est enfoncé
- *READ_BUMPERS* : lis l'état des deux bumpers et le compare avec 0 afin de savoir si les deux sont enfoncé simultanément

> Switch.s : comprend 3 routines

- *SWITCH_INIT* : initialise les deux switchs
- *READ_SWITCH1* : lis l'état du switch 1 et le compare avec 0 pour savoir si il est activé
- *READ_SWITCH2* : lis l'état du switch 1 et le compare avec 0 pour savoir si il est activé

> Moteur.s : comprend 14 routines

- *MOTEUR_INIT* : initialise les configurations des moteurs
- *MOTEUR_DROIT_ON* : active le moteur droit
- *MOTEUR_DROIT_OFF* : désactive le moteur droit
- *MOTEUR_GAUCHE_ON* : active le moteur gauche
- *MOTEUR_GAUCHE_OFF* : désactive le moteur gauche
- *MOTEUR_DROIT_AVANT* : configure le moteur droit pour une rotation vers l'avant
- *MOTEUR_DROIT_ARRIERE* : configure le moteur droit pour une rotation vers l'arrière
- *MOTEUR_GAUCHE_AVANT* : configure le moteur gauche pour une rotation vers l'avant
- *MOTEUR_GAUCHE_ARRIERE* : configure le moteur gauche pour une rotation vers l'arrière
- *MOTEUR_DROIT_INVERSE* : inverse la direction du moteur droit
- *MOTEUR_GAUCHE_INVERSE* : inverse la direction du moteur gauche
- *SET_VITESSE_DEFAULT* : configure les moteurs pour qu'il roule à la vitesse par défaut
- *SET_VITESSE_FORCING* : configure les moteurs pour qu'il roule à la vitesse par de forcing (plus élevée pour pousser des obstacles)

> Main.s : composé de 11 parties et 3 routines

- ***Les parties :***
    - *INITIALISATION* : appelle les routines d'initialisation de tous les composants et met le compteur de rotation (r4) à 0
    - *PAUSE :* désactive les moteurs et allume les deux leds
    - *PAUSE_LOOP :* lis l'état de switch 1 en boucle, si appuyé va à la partie *PLAY*
    - *PLAY :* active les moteurs et les lances en avant
    - *PLAY_LOOP :* fait clignoter les leds et lis l'état des bumpers et du switch 2, en fonction de leur activation décide d'une action à réaliser
    - *COLISION_FRONTALE :* décrémente le compteur de rotation (r4), si il est à 0 - qu'un demi tour complet à été effectué - force le passage sinon tourne à droite
    - *FORCING :* met la vitesse en mode forcing, reset le compteur de rotation (r4), si au bout d'un certain temps les bumpers sont toujours enfoncés le robot se met en pause.

- *ACTION_BUMPER_1 :* reset le compteur de rotation (r4) et effectue une rotation à gauche
- *ACTION_BUMPER_2 :* reset le compteur de rotation (r4) et effectue une rotation à droite
- *ROTATION_DROITE :* recule, allume la led4 (à droite) pivote vers la droite et retourne à l'étape *PLAY*
- *ROTATION_GAUCHE :* recule, allume la led5 (à gauche) pivote vers la gauche et retourne à l'étape *PLAY*

- ***Les routines :***
  - *WAIT_LED :* attente du programme pendant un temps court (utilisé pour le clignotement des leds). Se compose aussi de la boucle *WAIT_LED_LOOP*
  - *WAIT_ROTATION :* attente du programme pendant un temps moyen (utilisé pour les rotations). Se compose aussi de la boucle *WAIT_ROTATION_LOOP*
  - *LEGER_RECUL :* fait reculer le robot pendant un court instant. A noter que faisant appel à la routine *WAIT_ROTATION* l'adresse de l'instruction de retour est sauvegardé sur la pile grâce à PUSH et lue grâce à POP

A noter qu'en plus de cette composition nous avons fait le choix de définir des rôles aux différents registres afin de nous simplifier là vie. Cela n'est possible que grâce aux nombre réduit de périphériques et fonctionnalités.

> Rôle des registres

- R12 : Bouton pressoir 2 [COMPOSANT]
- R11 : Bouton pressoir 1 [COMPOSANT]
- R10 : LEDs [COMPOSANT]
- R9 : Bumper 1 [COMPOSANT]
- R8 : Bumper 2 [COMPOSANT]
- **R7 : LIBRE** [COMPOSANT] (pour un futur enrichissement)
- R6 : Moteur [COMPOSANT]
- R5 : Comparaison (là où on STR toutes les résultats des méthodes de read)
- R4 : Compteur de collision restantes avant de Forcer

- **R3 : LIBRE**
- **R2 : LIBRE**
- **R1 : LIBRE**
- **R0 : LIBRE**

Configuration des périphériques

Voici les différentes configurations des différents périphériques que nous avons réalisé :

> Leds

- branchement du port F à la clock
- configuration des broches 4 et 5 en tant que sortie avec *GPIO_O_DIR*
- configuration de la fonction numérique sur les broches 4 et 5 avec *GPIO_O_DEN*
- configuration de l'ampérage à 2mA sur les broches 4 et 5 avec *GPIO_O_DR2R*

> Bumpers

- branchement du port E à la clock
- configuration de la résistance de pull-up sur les broches 0 et 1 avec *GPIO_I_PUR*
- configuration de la fonction numérique sur les broches 0 et 1 avec *GPIO_O_DEN*

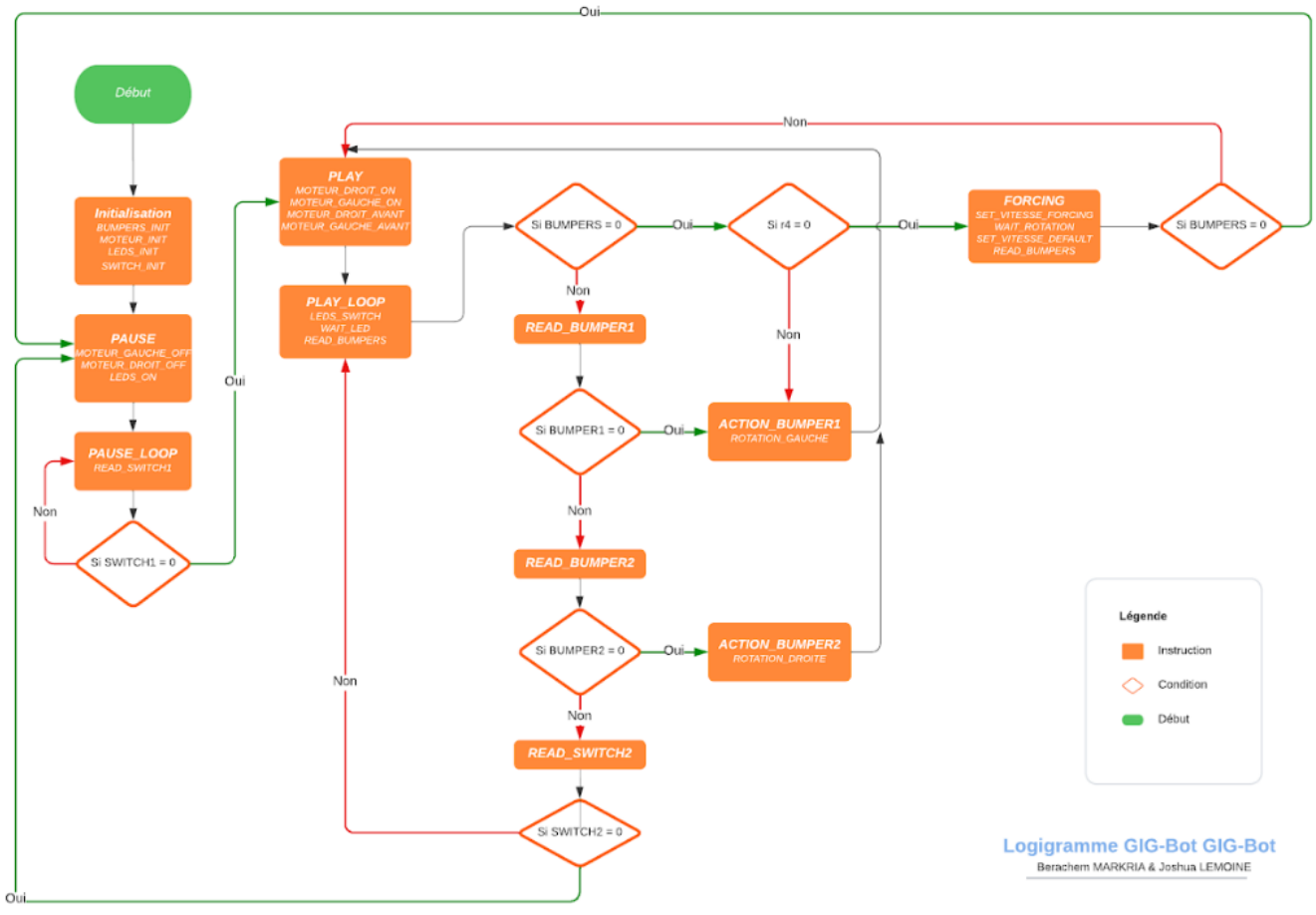
> Switch

- branchement du port D à la clock
- configuration de la résistance de pull-up sur les broches 6 et 7 avec GPIO_I_PUR
- activation de la fonction numérique sur les broches 6 et 7 avec GPIO_O_DEN

Représentations du code

Dans cette section, nous aborderons le pseudo-code et le logigramme du GIG-Bot, détaillant les étapes clés de son fonctionnement autonome. Ces représentations visuelles permettront une compréhension approfondie de l'algorithme et de la logique sous-jacente qui guident le robot lors de ses interventions.

> Logigramme



> Pseudo-code

```
Début

Initialiser les bumpers, moteurs, LEDs, et switchs

Définir le nombre de collisions frontales avant forcing à 5

Boucle PAUSE
    Désactiver les moteurs
    Allumer les LEDs
    Tant que SWITCH1 n'est pas activé
        Attendre
    Fin de la boucle

Boucle PLAY
    Activer les moteurs
    Faire avancer les moteurs
    Tant que SWITCH2 n'est pas activé
        Changer l'état des LEDs
        Attendre un court moment
        Si une collision frontale est détectée
            Si le nombre de collisions frontales atteint le seuil
                Passage en force
            Sinon
                Rotation à droite
            Fin de la condition
        Sinon si BUMPER1 est activé
            Rotation à gauche
        Sinon si BUMPER2 est activé
            Rotation à droite
        Fin de la condition
    Fin de la boucle

Fin de la boucle PLAY

Fin
```

Réalisations

Si on allume le GIG-Bot:

1. Celui-ci avance en faisant clignoter ses deux lumières le rendant plus visible à travers des fumigènes notamment.

- moteurs 1 & 2 AVANT
- clignotement des LED 1 & 2

2. On peut faire PAUSE de l'exécution en appuyant sur le switch 1, ce qui gardera allumé de façon constante les deux lumières en attendant que l'agent appuie sur le switch 2

- lecture des Switch 1 & 2
- allumage des LED 1 & 2

3. Lorsqu'il bute un obstacle (bumpers), il va arrêter de clignoter ses LEDs, effectuer une rotation (si l'obstacle est à gauche, vers la droite. si l'obstacle est à droite, vers la gauche. si l'obstacle est en face pile, vers l'arrière) allumant la LEDs du côté de où il va tourner et changer de direction

- lecture des Bumpers 1 & 2
- clignotement de LED 1 OU 2
- moteur 1 OU 2 ARRIÈRE

4. S'il s'agît de la 4ème collision frontale (deux bumpers), le GIG-Bot active un mode "Forcing" en accélérant et forçant sur l'obstacle. S'il refait une collision frontale après son forcing (l'obstacle n'a pas bougé) alors il se met en PAUSE sinon il reprend son processus de base (étape 1).

- augmentation/diminution vitesse moteur

5. On revient à l'étape 1.

Difficultés rencontrés et solutions

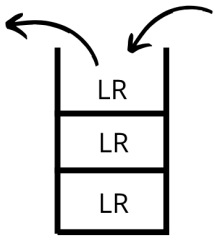
> Séparation du code en plusieurs fichiers

Notre principal défi résidait dans l'ajustement nécessaire à l'extrême linéarité de l'assembleur, une approche bien éloignée de la pensée plus fluide et structurée que certains d'entre nous avaient déjà développée en travaillant avec des langages de haut niveau. L'assembleur, en tant que langage bas niveau, impose une séquence rigide d'instructions, ce qui contraste fortement avec la flexibilité conceptuelle offerte par les langages plus abstraits.

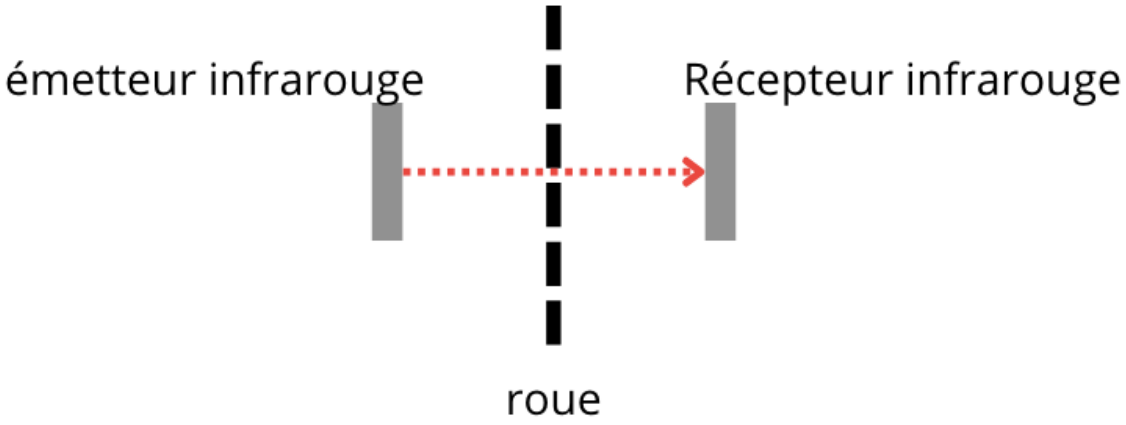
La solution s'est à été trouvée au moment où nous avons appris comment créer une architecture modulaire en assembleur. En utilisant des concepts comme les routines par exemple. Nous avons donc divisé notre code en plusieurs sections logiques, chacune étant responsable d'une fonctionnalité spécifique du robot. Chaque section a été écrite dans un fichier séparé, garantissant que chaque composant pouvait être compilé et testé indépendamment des autres, facilitant ainsi la gestion et la maintenance du code.

> Pile de LR

Enfin, un autre défi était de gérer les retours successifs au code précédent avec les instructions BX LR. Parfois, l'adresse précédente était écrasée, nous avons donc choisi d'utiliser les instructions PUSH & POP qui utilisent la pile d'instructions comme stockage .



Evolutions envisagées : Proof of concepts



Sous les conseils de M. LORENS, nous avons envisagé de gérer la distance et le temps écoulés grâce aux **émetteurs et récepteurs infrarouges situés aux extrémités des roues** du GIG-Bot en analysant la succession des “petits trous sur la roue”. Cependant, après avoir réalisé des tests, nous avons constaté que la difficulté était trop élevée.

Selon la datasheet de Texas Instruments (p1206), le module QEI (Quadrature Encoder Interface) est capable d’interpréter le code gray à deux bits produit par une roue codeuse en quadrature pour intégrer la position dans le temps et déterminer la direction de rotation. De plus, il peut capturer une estimation en cours de la vitesse de la roue.

Les signaux de phase, PhA et PhB, peuvent être interprétés comme un signal d’horloge et de direction. Le module “QEI” dispose également d’un filtre numérique de bruit sur ses signaux d’entrée qui peut être activé pour prévenir une opération erronée. Ce filtre nécessite que les entrées soient stables pendant un certain nombre de cycles d’horloge consécutifs avant de mettre à jour le détecteur de bord.

Cependant, malgré ces fonctionnalités avancées, l'implémentation de cette technologie dans notre projet s'est avérée être un défi majeur quoique très instructif.

Conclusion

Ainsi le projet GIG-Bot nous à permis d'exploiter efficacement les périphériques clés de l'EVALBOT, tels que les LEDs, bumpers, switches, et moteurs, afin de concevoir un robot éclaireur innovant. Notre approche modulaire a été cruciale pour maintenir la clarté et la gestion du code, permettant à chaque membre du groupe de se concentrer sur des parties spécifiques du projet. Les différents fichiers, chacun dédié à un composant particulier, ont facilité la collaboration et la maintenance du code.

De plus, nous avons pensé à la suite. En termes d'évolutions, nous avons envisagé des proof of concepts, notamment l'utilisation de capteurs infrarouges pour mesurer la distance et le temps écoulé. Une première tentative nous a d'ailleurs fourni des insights précieux pour de futures améliorations.

En conclusion, le GIG-Bot représente une étape solide dans l'amélioration de l'EVALBOT, offrant une base propice à des développements ultérieurs. Nous sommes impatients d'explorer des possibilités supplémentaires pour enrichir davantage les fonctionnalités et les performances du robot, consolidant ainsi son impact et son utilité dans des scénarios d'intervention du GIGN.

Nous sommes heureux du rendu final du projet. Il nous a réellement aidé à mettre un pied concret dans le monde de l'assembleur et ce de façon ludique ! 🙂

Annexe

> Fichier Main.s

```
          AREA    |.text|, CODE, READONLY
          ENTRY
          EXPORT    __main

          IMPORT    MOTEUR_INIT                         ;
initialise les moteurs (configure les pwms + GPIO)

          IMPORT    MOTEUR_DROIT_ON               ; activer le
moteur droit
          IMPORT  MOTEUR_DROIT_OFF           ; d'activer le
moteur droit
          IMPORT  MOTEUR_DROIT_AVANT              ; moteur
droit tourne vers l'avant
          IMPORT  MOTEUR_DROIT_ARRIERE       ; moteur droit
tourne vers l'arri're
          IMPORT  MOTEUR_DROIT_INVERSE       ; inverse le sens
de rotation du moteur droit

          IMPORT    MOTEUR_GAUCHE_ON              ; activer le
moteur gauche
          IMPORT  MOTEUR_GAUCHE_OFF          ; d'activer le
moteur gauche
          IMPORT  MOTEUR_GAUCHE_AVANT             ; moteur
gauche tourne vers l'avant
          IMPORT  MOTEUR_GAUCHE_ARRIERE      ; moteur gauche
tourne vers l'arri're
          IMPORT  MOTEUR_GAUCHE_INVERSE      ; inverse le sens
de rotation du moteur gauche

          IMPORT SET_VITESSE_DEFAULT          ; mettre la
vitesse par defaut
          IMPORT SET_VITESSE_FORCING              ; mettre la
```

```
vitesse en mode forcing

          IMPORT BUMPERS_INIT                    ; initialiser
les bumber
          IMPORT READ_BUMPER1                    ; lire l'etat
du bumper 1
          IMPORT READ_BUMPER2                    ; lire l'etat
du bumper 2
          IMPORT READ_BUMPERS                    ; lire l'etat
des deux bumpers

          IMPORT LEDS_INIT                       ;
initialisation des leds
          IMPORT LEDS_SWITCH                     ; changements
de l'etat des deux leds
          IMPORT LEDS_ON                         ; allumer les
deux leds
          IMPORT LEDS_OFF                        ; eteindre
les deux leds
          IMPORT LED5_ON                         ; allumer la
led5
          IMPORT LED4_ON                         ; allumer la
led4


          IMPORT SWITCH_INIT                     ; initialiser
les boutons switchs
          IMPORT READ_SWITCH1                    ; lire l'etat
du switch 1
          IMPORT READ_SWITCH2                    ; lire l'etat
du switch 2

__main
;initialisations
```

```
     BL BUMPERS_INIT
     BL MOTEUR_INIT
     BL LEDS_INIT
     BL SWITCH_INIT
     LDR r4, =5 ;nombre de colision frontale avant forcing

;Boucle mettant le robot en pause
PAUSE
     BL MOTEUR_GAUCHE_OFF
     BL MOTEUR_DROIT_OFF
     BL LEDS_ON

PAUSE_LOOP
     BL READ_SWITCH1
     CMP r5,#0x00
     BEQ PLAY

     B PAUSE_LOOP

;Boucle principale, utilisée pour faire avancer le robot tout
droit
PLAY
     BL MOTEUR_DROIT_ON
     BL MOTEUR_GAUCHE_ON

     BL MOTEUR_DROIT_AVANT
     BL MOTEUR_GAUCHE_AVANT
PLAY_LOOP

     BL LEDS_SWITCH
     BL WAIT_LED

     BL READ_BUMPERS
     BEQ COLISION_FRONTALE ;decision arbitraire
```

```
     BL READ_BUMPER1
     BEQ ACTION_BUMPER1

     BL READ_BUMPER2
     BEQ ACTION_BUMPER2

     BL READ_SWITCH2
     BEQ PAUSE

     B PLAY_LOOP

;instructions pour gérer la colision frontale
COLISION_FRONTALE
     SUBS r4,#1
     BEQ FORCING
     B ROTATION_DROITE

;instrutions pour realiser un passage en force
FORCING
     BL SET_VITESSE_FORCING
     LDR r4, =5 ;nombre de colision frontale avant forcing
     BL WAIT_ROTATION
     BL SET_VITESSE_DEFAULT
     BL READ_BUMPERS
     BEQ PAUSE
     B PLAY

;instructions quand le bumber1 est active
ACTION_BUMPER1
     LDR r4, =5 ;nombre de colision frontale avant forcing
     BL ROTATION_GAUCHE

;instructions quand le bumber2 est active
```

```
ACTION_BUMPER2
     LDR r4, =5 ;nombre de colision frontale avant forcing
     BL ROTATION_DROITE


;instructions pour effectuer une rotation à droite
ROTATION_DROITE
     BL LEGER_RECUL
     BL LEDS_OFF
     BL LED4_ON
     BL MOTEUR_GAUCHE_AVANT
     BL WAIT_ROTATION
     BL LEDS_OFF
     B PLAY

;instructions pour effectuer une rotation à gauche
ROTATION_GAUCHE
     BL LEGER_RECUL
     BL LEDS_OFF
     BL LED5_ON
     BL MOTEUR_DROIT_AVANT
     BL WAIT_ROTATION
     BL LEDS_OFF
     B PLAY

;routine d'attente avec un timer court, utilisé pour le
clignotement des leds
WAIT_LED

     ldr r1, =0xAFFFF
WAIT_LED_LOOP
     subs r1, #1
     bne WAIT_LED_LOOP
     ;; retour ' la suite du lien de branchement
```

```
     BX   LR

;routine d'attente avec un timer long, utilisé pour la rotation
WAIT_ROTATION
     ldr r1, =0xAFFFFF
WAIT_ROTATION_LOOP
     subs r1, #1
     bne WAIT_ROTATION_LOOP
     ;; retour ' la suite du lien de branchement
     BX   LR

;routine pour effectuer un leger recul
LEGER_RECUL
     PUSH {LR} ;sauvegarde du retour sur la pile (LR va être
réécrit pas les appels au autres routines)

     BL LEDS_ON
     BL MOTEUR_DROIT_ARRIERE
     BL MOTEUR_GAUCHE_ARRIERE
     BL WAIT_ROTATION

     POP {LR} ;recupération du retour sur la pile
     BX LR

;fin du programme
     NOP
     NOP
     END
```

> Fichier Leds.s

```
          ;; RK - Evalbot (Cortex M3 de Texas Instrument)
          ;; Fichier contenant l'initialisation des LEDs et
leur fonctions d'interaction avec le Main

          AREA    |.text|, CODE, READONLY

; This register controls the clock gating logic in normal Run
mode
SYSCTL_PERIPH_GPIO EQU        0x400FE108; SYSCTL_RCGC2_R (p291
datasheet de lm3s9b92.pdf)

; The GPIODATA register is the data register
GPIO_PORTF_BASE     EQU       0x40025000; GPIO Port F (APB)
base: 0x4002.5000 (p416 datasheet de lm3s9B92.pdf)

; configure the corresponding pin to be an output
; all GPIO pins are inputs by default
GPIO_O_DIR          EQU  0x00000400  ; GPIO Direction (p417
datasheet de lm3s9B92.pdf)

; The GPIODR2R register is the 2-mA drive control register
; By default, all GPIO pins have 2-mA drive.
```

```
GPIO_O_DR2R         EQU  0x00000500  ; GPIO 2-mA Drive Select
(p428 datasheet de lm3s9B92.pdf)

; Digital enable register
; To use the pin as a digital input or output, the corresponding
GPIODEN bit must be set.
```

```
GPIO_O_DEN          EQU  0x0000051C  ; GPIO Digital Enable
(p437 datasheet de lm3s9B92.pdf)

; Pul_up
GPIO_I_PUR          EQU  0x00000510  ; GPIO Pull-Up (p432
datasheet de lm3s9B92.pdf)

; Broches select
BROCHE4                 EQU       0x10      ; led1
BROCHE5                 EQU       0x20      ; led2
BROCHE4_5           EQU       0x30      ; led1 & led2 sur
broche 4 et 5

          ;; The EXPORT command specifies that a symbol can be
accessed by other shared objects or executables.
          EXPORT    LEDS_INIT
          EXPORT  LEDS_ON
          EXPORT  LEDS_OFF
          EXPORT  LEDS_SWITCH
          EXPORT    LED4_ON
          EXPORT  LED5_ON


LEDS_INIT

          ; ;; Enable the Port F & D peripheral clock
(p291 datasheet de lm3s9B96.pdf)
          ldr r10, = SYSCTL_PERIPH_GPIO               ;; RCGC2
          ldr  r5, [r10]
        ORR r5, r5, #0x00000020                   ;; Enable
clock sur GPIO  F ou est branchés les leds
        str r5, [r10]

          ; ;; "There must be a delay of 3 system clocks before
```

```
any GPIO reg. access  (p413 datasheet de lm3s9B92.pdf)
          nop
          nop
          nop

        ldr r10, = GPIO_PORTF_BASE+GPIO_O_DIR    ;; 1 Pin du
portF en sortie (broche 4 : 00010000)
          ldr r5, [r10]
        orr r5, r5, #BROCHE4_5
        str r5, [r10]

          ldr r10, = GPIO_PORTF_BASE+GPIO_O_DEN   ;; Enable
Digital Function
          ldr r5, [r10]
        orr r5, r5, #BROCHE4_5
        str r5, [r10]

          ldr r10, = GPIO_PORTF_BASE+GPIO_O_DR2R  ;; Choix de
l'intensité de sortie (2mA)
          ldr r5, [r10]
        orr r5, r5, #BROCHE4_5
        str r5, [r10]



          BX LR

;routine pour allumer les deux leds
LEDS_ON
          ldr r10, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;; @data
Register = @base + (mask<<2) ==> LED1
        ldr r0, = BROCHE4_5
          STR r0, [r10]
          BX LR
```

```
;routine pour eteindre les deux leds
LEDS_OFF
          ldr r10, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;; @data
Register = @base + (mask<<2) ==> LED1
        mov r0, #0x00
          STR r0, [r10]
          BX LR

;routine pour changer l'etat des deux leds
LEDS_SWITCH
          ldr r10, = GPIO_PORTF_BASE + (BROCHE4_5<<2)
          ldr r0, [r10]
          EOR r0, r0, #0x30 ; BROCHE4_5
          STR r0, [r10]
          BX LR

;routine pour allumer la led 4
LED4_ON
          ldr r10, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;; @data
Register = @base + (mask<<2) ==> LED1
          ldr r0, = BROCHE4
          STR r0, [r10]
          BX LR

;routine pour allumer la led 5
LED5_ON
          ldr r10, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;; @data
Register = @base + (mask<<2) ==> LED1
          ldr r0, = BROCHE5
          STR r0, [r10]
          BX LR

          NOP
```

```
        NOP
        END
```

> Fichier Bumper.s

```
     AREA    |.text|, CODE, READONLY

; This register controls the clock gating logic in normal Run
mode
SYSCTL_PERIPH_GPIO  EQU       0x400FE108 ; SYSCTL_RCGC2_R (p291
datasheet de lm3s9b92.pdf)

; The GPIODATA register is the data register
GPIO_PORTE_BASE     EQU       0x40024000 ; GPIO Port E (ABP)
base : 0x4002.4000 (p291 datasheet de lm3s9b92.pdf)

; Digital enable register
; To use the pin as a digital input or output, the corresponding
GPIODEN bit must be set.
GPIO_O_DEN          EQU  0x0000051C  ; GPIO Digital Enable
(p437 datasheet de lm3s9B92.pdf)

; Pul_up
GPIO_I_PUR          EQU  0x00000510  ; GPIO Pull-Up (p432
datasheet de lm3s9B92.pdf)

BROCHE0             EQU     0x01        ; Broche 0
BROCHE1             EQU     0x02        ; Broche 1
BROCHE0_1           EQU     0x03        ; Broche 1 et 2

     ;; The EXPORT command specifies that a symbol can be
```

```
accessed by other shared objects or executables.
     EXPORT BUMPERS_INIT
     EXPORT READ_BUMPER1
     EXPORT READ_BUMPER2
     EXPORT READ_BUMPERS

;routine d'initialisation des bumpers
BUMPERS_INIT
     ;branchement du port E
     ldr r6, = SYSCTL_PERIPH_GPIO            ;; RCGC2
     mov r0, #0x10                                ;; Enable
clock sur GPIO E 0x10 = ob010000
     str r0, [r6]

     ;waiting for GPIO reg. access
     nop
     nop
     nop

     ;setup bumpers
     ldr r6, = GPIO_PORTE_BASE+GPIO_I_PUR    ;; Pul_up
     ldr r0, = BROCHE0_1
     str r0, [r6]

     ldr r6, = GPIO_PORTE_BASE+GPIO_O_DEN    ;; Enable Digital
Function
     ldr r0, = BROCHE0_1
     str r0, [r6]

     ldr r9, = GPIO_PORTE_BASE + (BROCHE0<<2)
     ldr r8, = GPIO_PORTE_BASE + (BROCHE1<<2)

     BX LR
```

```
;routine de lecture de bumper 1
READ_BUMPER1
     ;lecture de l'etat du bumper1

     ldr r9, = GPIO_PORTE_BASE + (BROCHE0<<2)
     ldr r5, [r9]
     cmp r5,#0x00
     BX LR

;routine de lecture de bumper 2
READ_BUMPER2
     ;lecture de l'etat du bumper2

     ldr r8, = GPIO_PORTE_BASE + (BROCHE1<<2)
     ldr r5, [r8]
     cmp r5,#0x00
     BX LR

;routine de lecture des bumpers
READ_BUMPERS
     ;lecture des deux ports en meme temps

     ldr r8, = GPIO_PORTE_BASE + (BROCHE0_1<<2)
     ldr r5, [r8]
     cmp r5,#0x00
     BX LR

;fin du programme
     NOP
     NOP
     END
```

> Fichier Switch.s

```
          AREA    |.text|, CODE, READONLY

; This register controls the clock gating logic in normal Run
mode
SYSCTL_PERIPH_GPIO EQU        0x400FE108; SYSCTL_RCGC2_R (p291
datasheet de lm3s9b92.pdf)

; The GPIODATA register is the data register
GPIO_PORTD_BASE     EQU       0x40007000     ; GPIO Port D
(APB) base: 0x4000.7000 (p416 datasheet de lm3s9B92.pdf)

; Digital enable register
; To use the pin as a digital input or output, the corresponding
GPIODEN bit must be set.
GPIO_O_DEN          EQU  0x0000051C  ; GPIO Digital Enable
(p437 datasheet de lm3s9B92.pdf)

; Pul_up
GPIO_I_PUR          EQU  0x00000510  ; GPIO Pull-Up (p432
datasheet de lm3s9B92.pdf)

; Broches select
BROCHE6                 EQU  0x40      ; bouton 1
BROCHE7                 EQU       0x80      ; bouton 2
BROCHE6_7           EQU       0xc0      ; bouton 1 & bouton 2


          ;; The EXPORT command specifies that a symbol can be
accessed by other shared objects or executables.
```

```
          EXPORT SWITCH_INIT
          EXPORT READ_SWITCH1
          EXPORT READ_SWITCH2

;routine d'initialisation des switchs
SWITCH_INIT

     ; ;; Enable the Port D peripheral clock
     ldr r12, = SYSCTL_PERIPH_GPIO                ;; RCGC2
     ldr r0, [r12]
     ORR r0, r0, #0x00000008
     str r0, [r12]

     ; ;; "There must be a delay of 3 system clocks before any
GPIO reg. access  (p413 datasheet de lm3s9B92.pdf)
     nop
     nop
     nop

     ldr r11, = GPIO_PORTD_BASE+GPIO_I_PUR    ;; Pull up
     ldr r0, [r11]
     orr r0, r0, #BROCHE6_7
     str r0, [r11]

     ldr r11, = GPIO_PORTD_BASE+GPIO_O_DEN    ;; Enable Digital
Function
     ldr r0, [r11]
     orr r0, r0, #BROCHE6_7
     str r0, [r11]

     ldr r11, = GPIO_PORTD_BASE + (BROCHE6<<2)  ;; @data
Register = @base + (mask<<2) ==> Switch
     ldr r12, = GPIO_PORTD_BASE + (BROCHE7<<2)  ;; @data
Register = @base + (mask<<2) ==> Switch
```

```
     BX LR

;routine de lecture de l'etat du switch 1
READ_SWITCH1
     ;Lecture du contenu de la valeur d'activation du Switch 1

     ldr r11, = GPIO_PORTD_BASE + (BROCHE6<<2)  ;; @data
Register = @base + (mask<<2) ==> Switch
     LDR r5, [r11]
     cmp r5,#0x00
     BX LR

;routine de lecture de l'etat du switch 2
READ_SWITCH2
     ;Lecture du contenu de la valeur d'activation du Switch 2

     ldr r12, = GPIO_PORTD_BASE + (BROCHE7<<2)  ;; @data
Register = @base + (mask<<2) ==> Switch
     LDR r5, [r12]
     cmp r5,#0x00
     BX LR


;fin du programme
     NOP
     NOP
     END
```

> Fichier Moteur.s

```
     ;; RK - Evalbot (Cortex M3 de Texas Instrument);
; programme - Pilotage 2 Moteurs Evalbot par PWM tout en ASM
(configure les pwms + GPIO)

;Les pages se referent au datasheet lm3s9b92.pdf

;Cablage :
;pin 10/PD0/PWM0 => input PWM du pont en H DRV8801RT
;pin 11/PD1/PWM1 => input Phase_R  du pont en H DRV8801RT
;pin 12/PD2          => input SlowDecay commune aux 2 ponts en
H
;pin 98/PD5          => input Enable 12v du conv DC/DC
;pin 86/PH0/PWM2 => input PWM du 2nd pont en H
;pin 85/PH1/PWM3 => input Phase du 2nd pont en H

;; Hexa corresponding values to pin numbers
GPIO_0          EQU       0x1
GPIO_1          EQU       0x2
GPIO_2          EQU       0x4
GPIO_5          EQU       0x20

;; pour enable clock    0x400FE000
SYSCTL_RCGC0    EQU       0x400FE100     ;SYSCTL_RCGC0: offset
0x100 (p271 datasheet de lm3s9b92.pdf)
SYSCTL_RCGC2    EQU       0x400FE108     ;SYSCTL_RCGC2: offset
0x108 (p291 datasheet de lm3s9b92.pdf)

;; General-Purpose Input/Outputs (GPIO) configuration
PORTD_BASE      EQU       0x40007000
GPIODATA_D      EQU       PORTD_BASE
GPIODIR_D       EQU       PORTD_BASE+0x00000400
GPIODR2R_D      EQU       PORTD_BASE+0x00000500
```

```
GPIODEN_D       EQU       PORTD_BASE+0x0000051C
GPIOPCTL_D      EQU       PORTD_BASE+0x0000052C ; GPIO Port
Control (GPIOPCTL), offset 0x52C; p444
GPIOAFSEL_D         EQU       PORTD_BASE+0x00000420 ; GPIO
Alternate Function Select (GPIOAFSEL), offset 0x420; p426

PORTH_BASE      EQU       0x40027000
GPIODATA_H      EQU       PORTH_BASE
GPIODIR_H       EQU       PORTH_BASE+0x00000400
GPIODR2R_H      EQU       PORTH_BASE+0x00000500
GPIODEN_H       EQU       PORTH_BASE+0x0000051C
GPIOPCTL_H      EQU       PORTH_BASE+0x0000052C ; GPIO Port
Control (GPIOPCTL), offset 0x52C; p444
GPIOAFSEL_H         EQU       PORTH_BASE+0x00000420 ; GPIO
Alternate Function Select (GPIOAFSEL), offset 0x420; p426

;; Pulse Width Modulator (PWM) configuration
PWM_BASE        EQU       0x040028000       ;BASE des Block PWM
p.1138
PWMENABLE       EQU       PWM_BASE+0x008    ; p1145

;Block PWM0 pour sorties PWM0 et PWM1 (moteur 1)
PWM0CTL             EQU       PWM_BASE+0x040 ;p1167
PWM0LOAD        EQU       PWM_BASE+0x050
PWM0CMPA        EQU       PWM_BASE+0x058
PWM0CMPB        EQU       PWM_BASE+0x05C
PWM0GENA        EQU       PWM_BASE+0x060
PWM0GENB        EQU       PWM_BASE+0x064

;Block PWM1 pour sorties PWM1 et PWM2 (moteur 2)
PWM1CTL             EQU       PWM_BASE+0x080
PWM1LOAD        EQU       PWM_BASE+0x090
PWM1CMPA        EQU       PWM_BASE+0x098
PWM1CMPB        EQU       PWM_BASE+0x09C
```

```
PWM1GENA        EQU        PWM_BASE+0x0A0
PWM1GENB        EQU        PWM_BASE+0x0A4


VITESSE_DEFAULT           EQU       0x192; Valeures plus petites
=> Vitesse plus rapide exemple 0x192
                                                    ; Valeures
plus grandes => Vitesse moins rapide exemple 0x1B2
VITESSE_FORCING           EQU     0x152

          AREA    |.text|, CODE, READONLY
          ENTRY

          ;; The EXPORT command specifies that a symbol can be
accessed by other shared objects or executables.
          EXPORT    MOTEUR_INIT
          EXPORT    MOTEUR_DROIT_ON
          EXPORT  MOTEUR_DROIT_OFF
          EXPORT  MOTEUR_DROIT_AVANT
          EXPORT  MOTEUR_DROIT_ARRIERE
          EXPORT  MOTEUR_DROIT_INVERSE
          EXPORT    MOTEUR_GAUCHE_ON
          EXPORT  MOTEUR_GAUCHE_OFF
          EXPORT  MOTEUR_GAUCHE_AVANT
          EXPORT  MOTEUR_GAUCHE_ARRIERE
          EXPORT  MOTEUR_GAUCHE_INVERSE
          EXPORT  SET_VITESSE_DEFAULT
          EXPORT  SET_VITESSE_FORCING


MOTEUR_INIT
          ldr r6, = SYSCTL_RCGC0
          ldr  r0, [R6]
        ORR     r0, r0, #0x00100000  ;;bit 20 = PWM recoit
```

```
clock: ON (p271)
        str r0, [r6]

     ;ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_1);PWM clock is
processor clock /1
     ;Je ne fais rien car par defaut = OK!!
     ;*(int *) (0x400FE060)= *(int *)(0x400FE060)...;

     ;RCGC2 :  Enable port D GPIO(p291 ) car Moteur Droit sur
port D
          ldr r6, = SYSCTL_RCGC2
          ldr  r0, [R6]
        ORR    r0, r0, #0x08  ;; Enable port D GPIO
        str r0, [r6]

     ;MOT2 : RCGC2 :  Enable port H GPIO  (2eme moteurs)
          ldr r6, = SYSCTL_RCGC2
          ldr  r0, [R6]
        ORR    r0, r0, #0x80  ;; Enable port H GPIO
        str r0, [r6]

          nop
          nop
          nop

     ;;Pin muxing pour PWM, port D, reg. GPIOPCTL(p444), 4bits
de PCM0=0001<=>PWM (voir p1261)
     ;;il faut mettre 1 pour avoir PD0=PWM0 et PD1=PWM1
          ldr r6, = GPIOPCTL_D
          ;ldr r0, [R6]   ;;  *(int
*)(0x40007000+0x0000052C)=1;
        ;ORR   r0, r0, #0x01 ;; Port D, pin 1 = PWM
          mov  r0, #0x01
        str r0, [r6]
```

```
     ;;MOT2 : Pin muxing pour PWM, port H, reg. GPIOPCTL(p444),
4bits de PCM0=0001<=>PWM (voir p1261)
     ;;il faut mettre mux = 2 pour avoir PH0=PWM2 et PH1=PWM3
          ldr r6, = GPIOPCTL_H
          mov  r0, #0x02
        str r0, [r6]

     ;;Alternate Function Select (p 426), PD0 utilise alernate
fonction (PWM au dessus)
     ;;donc PD0 = 1
          ldr r6, = GPIOAFSEL_D
          ldr  r0, [R6]    ;*(int *)(0x40007000+0x00000420)=
*(int *)(0x40007000+0x00000420) | 0x00000001;
        ORR    r0, r0, #0x01 ;
        str r0, [r6]

     ;;MOT2 : Alternate Function Select (p 426), PH0 utilise
PWM donc Alternate funct
     ;;donc PH0 = 1
          ldr r6, = GPIOAFSEL_H
          ldr  r0, [R6]    ;*(int *)(0x40007000+0x00000420)=
*(int *)(0x40007000+0x00000420) | 0x00000001;
        ORR    r0, r0, #0x01 ;
        str r0, [r6]

     ;;-----------PWM0 pour moteur 1 connecté à PD0
     ;;PWM0 produit PWM0 et PWM1 output
     ;;Config Modes PWM0 + mode GenA + mode GenB
          ldr r6, = PWM0CTL
          mov  r0, #2         ;Mode up-down-up-down, pas
synchro
        str r0, [r6]
```

```
          ldr r6, =PWM0GENA ;en decomptage, qd comparateurA =
compteur => sortie pwmA=0
                            ;en comptage croissant, qd
comparateurA = compteur => sortie pwmA=1
          mov  r0,  #0x0B0    ;0B0=10110000 => ACTCMPBD=00 (B
down:rien), ACTCMPBU=00(B up rien)
          str r0, [r6]   ;ACTCMPAD=10 (A down:pwmA low),
ACTCMPAU=11 (A up:pwmA high) , ACTLOAD=00,ACTZERO=00

          ldr r6, =PWM0GENB;en comptage croissant, qd
comparateurB = compteur => sortie pwmA=1
          mov  r0,  #0x0B00   ;en decomptage, qd comparateurB =
compteur => sortie pwmB=0
          str r0, [r6]
     ;Config Compteur, comparateur A et comparateur B
     ;;#define PWM_PERIOD (ROM_SysCtlClockGet() / 16000),
     ;;en mesure : SysCtlClockGet=0F42400h, /16=0x3E8,
     ;;on divise par 2 car moteur 6v sur alim 12v
          ldr  r6, =PWM0LOAD ;PWM0LOAD=periode/2 =0x1F4
          mov r0,   #0x1F4
          str  r0,[r6]

          ldr  r6, =PWM0CMPA ;Valeur rapport cyclique : pour
10% => 1C2h si clock = 0F42400
          mov  r0, #VITESSE_DEFAULT
          str  r0, [r6]

          ldr  r6, =PWM0CMPB ;PWM0CMPB recoit meme valeur.
(rapport cyclique depend de CMPA)
          mov  r0,  #0x1F4
          str  r0,  [r6]

     ;Control PWM : active PWM Generator 0 (p1167):
Enable+up/down + Enable counter debug mod
```

```
          ldr  r6, =PWM0CTL
          ldr  r0, [r6]
          ORR  r0,  r0,  #0x07
          str  r0,  [r6]

     ;;-----------PWM2 pour moteur 2 connecté à PH0
     ;;PWM1block produit PWM2 et PWM3 output
          ;;Config Modes PWM2 + mode GenA + mode GenB
          ldr r6, = PWM1CTL
          mov  r0, #2         ;Mode up-down-up-down, pas
synchro
        str r0, [r6];*(int *)(0x40028000+0x040)=2;

          ldr r6, =PWM1GENA ;en decomptage, qd comparateurA =
compteur => sortie pwmA=0
                            ;en comptage croissant, qd
comparateurA = compteur => sortie pwmA=1
          mov  r0,  #0x0B0    ;0B0=10110000 => ACTCMPBD=00 (B
down:rien), ACTCMPBU=00(B up rien)
          str r0, [r6]   ;ACTCMPAD=10 (A down:pwmA low),
ACTCMPAU=11 (A up:pwmA high) , ACTLOAD=00,ACTZERO=00

          ;*(int *)(0x40028000+0x060)=0x0B0; //
          ldr r6, =PWM1GENB   ;*(int
*)(0x40028000+0x064)=0x0B00;
          mov  r0,  #0x0B00   ;en decomptage, qd comparateurB =
compteur => sortie pwmB=0
          str r0, [r6]   ;en comptage croissant, qd
comparateurB = compteur => sortie pwmA=1
     ;Config Compteur, comparateur A et comparateur B
     ;;#define PWM_PERIOD (ROM_SysCtlClockGet() / 16000),
     ;;en mesure : SysCtlClockGet=0F42400h, /16=0x3E8,
     ;;on divise par 2 car moteur 6v sur alim 12v
          ;*(int *)(0x40028000+0x050)=0x1F4;
```

```
//PWM0LOAD=periode/2 =0x1F4
          ldr  r6, =PWM1LOAD
          mov r0,   #0x1F4
          str  r0,[r6]

          ldr  r6, =PWM1CMPA ;Valeur rapport cyclique : pour
10% => 1C2h si clock = 0F42400
          mov  r0,  #VITESSE_DEFAULT
          str  r0, [r6]  ;*(int *)(0x40028000+0x058)=0x01C2;

          ldr  r6, =PWM1CMPB ;PWM0CMPB recoit meme valeur.
(CMPA depend du rapport cyclique)
          mov  r0,  #0x1F4    ; *(int
*)(0x40028000+0x05C)=0x1F4;
          str  r0,  [r6]

     ;Control PWM : active PWM Generator 0 (p1167):
Enable+up/down + Enable counter debug mod
          ldr  r6, =PWM1CTL
          ldr  r0, [r6]  ;*(int *) (0x40028000+0x40)= *(int
*)(0x40028000+0x40) | 0x07;
          ORR  r0,  r0,  #0x07
          str  r0,  [r6]

     ;;-----Fin config des PWMs

     ;PORT D OUTPUT pin0 (pwm)=pin1(direction)=pin2(slow
decay)=pin5(12v enable)
          ldr  r6, =GPIODIR_D
          ldr  r0, [r6]
          ORR  r0,  #(GPIO_0+GPIO_1+GPIO_2+GPIO_5)
          str  r0,[r6]
     ;Port D, 2mA les meme
          ldr  r6, =GPIODR2R_D ;
```

```
          ldr  r0, [r6]
          ORR  r0,  #(GPIO_0+GPIO_1+GPIO_2+GPIO_5)
          str  r0,[r6]
     ;Port D, Digital Enable
          ldr  r6, =GPIODEN_D ;
          ldr  r0, [r6]
          ORR  r0,  #(GPIO_0+GPIO_1+GPIO_2+GPIO_5)
          str  r0,[r6]
     ;Port D : mise à 1 de slow Decay et 12V et mise à 0 pour
dir et pwm
          ldr  r6,
=(GPIODATA_D+((GPIO_0+GPIO_1+GPIO_2+GPIO_5)<<2))
          mov  r0, #(GPIO_2+GPIO_5) ; #0x24
          str  r0,[r6]

     ;MOT2, PH1 pour sens moteur ouput
          ldr  r6, =GPIODIR_H
          mov  r0,  #0x03;
          str  r0,[r6]
     ;Port H, 2mA les meme
          ldr  r6, =GPIODR2R_H
          mov r0, #0x03
          str  r0,[r6]
     ;Port H, Digital Enable
          ldr  r6, =GPIODEN_H
          mov r0, #0x03
          str  r0,[r6]
     ;Port H : mise à 1 pour dir
          ldr  r6, =(GPIODATA_H +(GPIO_1<<2))
          mov  r0, #0x02
          str  r0,[r6]

          BX   LR   ; FIN du sous programme d'init.
```

```
;Enable PWM0 (bit 0) et PWM2 (bit 2) p1145
;Attention ici c'est les sorties PWM0 et PWM2
;qu'on controle, pas les blocks PWM0 et PWM1!!!
MOTEUR_DROIT_ON
          ;Enable sortie PWM0 (bit 0), p1145
          ldr  r6,  =PWMENABLE
          ldr r0, [r6]
          orr r0,   #0x01 ;bit 0 à 1
          str  r0,  [r6]
          BX   LR

MOTEUR_DROIT_OFF
          ldr  r6,  =PWMENABLE
          ldr r0,   [r6]
          and  r0,  #0x0E;bit 0 à 0
          str  r0,  [r6]
          BX   LR

MOTEUR_GAUCHE_ON
          ldr  r6,  =PWMENABLE
          ldr  r0, [r6]
          orr  r0,  #0x04;bit 2 à 1
          str  r0,  [r6]
          BX   LR

MOTEUR_GAUCHE_OFF
          ldr  r6,  =PWMENABLE
          ldr  r0,  [r6]
          and  r0,  #0x0B;bit 2 à 0
          str  r0,  [r6]
          BX   LR

MOTEUR_DROIT_ARRIERE
          ;Inverse Direction (GPIO_D1)
```

```
          ldr   r6, =(GPIODATA_D+(GPIO_1<<2))
          mov   r0, #0
          str   r0,[r6]
          BX    LR

MOTEUR_DROIT_AVANT
          ;Inverse Direction (GPIO_D1)
          ldr   r6, =(GPIODATA_D+(GPIO_1<<2))
          mov   r0, #2
          str   r0,[r6]
          BX    LR

MOTEUR_GAUCHE_ARRIERE
          ;Inverse Direction (GPIO_D1)
          ldr   r6, =(GPIODATA_H+(GPIO_1<<2))
          mov   r0, #2 ; contraire du moteur Droit
          str   r0,[r6]
          BX    LR

MOTEUR_GAUCHE_AVANT
          ;Inverse Direction (GPIO_D1)
          ldr   r6, =(GPIODATA_H+(GPIO_1<<2))
          mov   r0, #0
          str   r0,[r6]
          BX    LR

MOTEUR_DROIT_INVERSE
          ;Inverse Direction (GPIO_D1)
          ldr   r6, =(GPIODATA_D+(GPIO_1<<2))
          ldr   r1, [r6]
          EOR   r0, r1, #GPIO_1
          str   r0,[r6]
          BX    LR
```

```
MOTEUR_GAUCHE_INVERSE
          ;Inverse Direction (GPIO_D1)
          ldr  r6, =(GPIODATA_H+(GPIO_1<<2))
          ldr  r1, [r6]
          EOR  r0, r1, #GPIO_1
          str  r0,[r6]
          BX   LR

SET_VITESSE_DEFAULT
     ldr    r6, =PWM0CMPA
     mov    r0, #VITESSE_DEFAULT
     str    r0, [r6]

     ldr    r6, =PWM1CMPA
     mov    r0, #VITESSE_DEFAULT
     str    r0, [r6]

     BX LR

SET_VITESSE_FORCING
          ldr    r6, =PWM0CMPA
     mov    r0, #VITESSE_FORCING
     str    r0, [r6]

     ldr    r6, =PWM1CMPA
     mov    r0, #VITESSE_FORCING
     str    r0, [r6]

     BX LR


     BX LR

END
```