09/01/2024

Projet de Course avec Evalbot : Programmation Assembleur

23_E3FI_3I_IN1 - Architecture

Réalisé Par :

CHAABA Hamza & MOUHAMAD Afouane

Introduction

Au sein de notre unité d'architecture, nous sommes actuellement engagés dans la programmation de l'Evalbot en utilisant les composants intégrés tels que les bumpers, les switches, les LEDs et les moteurs. Ce rapport vise à documenter notre approche, les défis rencontrés et les résultats obtenus dans le cadre de ce projet, mettant en lumière le fait que la programmation s'effectue en langage assembleur.

Description du projet

Notre projet consiste à organiser une course avec un autre groupe pour évaluer les stratégies mises en place et déterminer lequel d'entre nous a adopté la meilleure approche pour remporter la victoire. Dans cette compétition, l'adaptabilité du robot à divers types de circuits, tels que des virages et des obstacles, sera cruciale. Ainsi, nous devrons définir une stratégie qui permette au robot de s'ajuster efficacement à toutes les conditions du parcours, maximisant ainsi nos chances de succès.

Présentation des différents scénarios

Notre robot Evalbot a été programmé pour une course dynamique. Dès son allumage, les LED s'illuminent, et en appuyant sur le switch 1, le robot tourne sur lui-même et les LED clignotent pour se préparer à la course, revenant ensuite à son état initial. Lors de l'appui sur le switch 2, il entre en mode course.

Stratégie adoptée : Dans le cadre de la course, nous avons mis en place une stratégie astucieuse. Lorsqu'un des bumpers détecte un contact avec un obstacle, la LED du côté touché s'éteint, et le robot effectue une légère rotation du côté opposé. Pour maximiser notre efficacité, nous avons choisi la stratégie de "suivre le mur", permettant au robot de rester en contact avec le mur pour obtenir la trajectoire la plus optimale possible.

1. Démarrage du robot avec illumination des LED.
2. Appui sur le switch 1 : les LED clignotent, et le robot tourne sur lui-même pour saluer les fans, puis revient à l'état initial.
3. Appui sur le switch 2 : le robot entre en mode course.
4. Contact d'un bumper : la LED du côté touché s'éteint.
5. Le robot tourne légèrement du coté opposé du bumper actif.
6. Adoption de la stratégie "suivre le mur" pour une course optimale.

Codes assembleurs développés

Main.s

```
        ;; RK - Evalbot (Cortex M3 de Texas Instrument)

        ;; Les deux LEDs sont initialement allumées

        ;; Ce programme lis l'état du bouton poussoir 1 connectée au port GPIOD broche 6

        ;; Si bouton poussoir fermé ==> fait clignoter les deux LED1&2 connectée au port GPIOF broches 4&5.


                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)


; The GPIODATA register is the data register

GPIO_PORTD_BASE         EQU         0x40007000          ; GPIO Port D (APB) base: 0x4000.7000 (p416 datasheet de lm3s9B92.pdf)


; The GPIODATA register is the data register

GPIO_PORTE_BASE     EQU         0x40024000


; 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)

BROCHE4         EQU  0x10        ; ledD
BROCHE5         EQU  0x20        ; ledG
BROCHE4_5       EQU  0x30        ; ledD_G
BROCHE1         EQU  0x01        ; bumperD
BROCHE2         EQU  0x02        ; bumperG
BROCHE1_2       EQU  0x03        ; bumperD_G
BROCHE6         EQU  0x40        ; switch1
BROCHE7         EQU  0x80        ; switch2
```

```
BROCHE6_7                   EQU         0xC0        ;; switch1_2



; fréquence clignotement
DUREE                       EQU     0x1E2FFF



                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            ; déactiver le moteur droit
                IMPORT  MOTEUR_DROIT_AVANT              ; moteur droit tourne vers
l'avant
                IMPORT  MOTEUR_DROIT_ARRIERE        ; moteur droit tourne vers l'arrière
                IMPORT  MOTEUR_DROIT_INVERSE        ; inverse le sens de rotation du
moteur droit

                IMPORT      MOTEUR_GAUCHE_ON            ; activer le moteur gauche
                IMPORT  MOTEUR_GAUCHE_OFF               ; déactiver le moteur
gauche
                IMPORT  MOTEUR_GAUCHE_AVANT             ; moteur gauche tourne
vers l'avant
                IMPORT  MOTEUR_GAUCHE_ARRIERE       ; moteur gauche tourne vers
l'arrière
```

```
                IMPORT  MOTEUR_GAUCHE_INVERSE           ; inverse le sens de rotation du
moteur gauche


__main


                ; ;; Enable the Port F & D peripheral clock                (p291 datasheet de
lm3s9B96.pdf)
                ; ;;
                ldr r6, = SYSCTL_PERIPH_GPIO                ;; RCGC2
    mov r0, #0x00000038                                ;; Enable clock sur GPIO D et F où sont
branchés les leds (0x28 == 0b101000)
                ; ;;

            (GPIO::FEDCBA)
    str r0, [r6]


                ; ;; "There must be a delay of 3 system clocks before any GPIO reg. access  (p413
datasheet de lm3s9B92.pdf)
                nop                                                                    ;;
tres tres important....
                nop
                nop                                                                    ;;
pas necessaire en simu ou en debbug step by step...



                ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CONFIGURATION LED


    ldr r6, = GPIO_PORTF_BASE+GPIO_O_DIR
    ldr r0, = BROCHE4_5
    str r0, [r6]
```

```
                ldr r6, = GPIO_PORTF_BASE+GPIO_O_DEN        ;; Enable Digital Function
    ldr r0, = BROCHE4_5
    str r0, [r6]

                ldr r6, = GPIO_PORTF_BASE+GPIO_O_DR2R       ;; Choix de l'intensité de sortie
(2mA)
    ldr r0, = BROCHE4_5
    str r0, [r6]

                mov r2, #0x000                                  ;; pour eteindre LED

                ; allumer la led droite et gauche (BROCHE4_5)
                mov r3, #BROCHE4_5

                ldr r6, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;; @data Register = @base +
(mask<<2) ==> LED1

                ;;vvvvvvvvvvvvvvvvvvvvvvvvFin configuration Bumper


                ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CONFIGURATION Bumper

                ldr r7, = GPIO_PORTE_BASE+GPIO_I_PUR        ;; Pul_up
    ldr r0, = BROCHE1_2
    str r0, [r7]

                ldr r7, = GPIO_PORTE_BASE+GPIO_O_DEN        ;; Enable Digital Function
    ldr r0, = BROCHE1_2
    str r0, [r7]
```

```
        ldr r7, = GPIO_PORTE_BASE + (BROCHE1_2<<2)  ;; @data Register = @base +
(mask<<2) ==> Bumper

        ;vvvvvvvvvvvvvvvvvvvvvvvvFin configuration Bumper


        ;;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CONFIGURATION Switcher

        ldr r7, = GPIO_PORTD_BASE+GPIO_I_PUR        ;; Pul_up
    ldr r0, = BROCHE6_7
    str r0, [r7]

        ldr r7, = GPIO_PORTD_BASE+GPIO_O_DEN        ;; Enable Digital Function
    ldr r0, = BROCHE6_7
    str r0, [r7]

        ldr r7, = GPIO_PORTD_BASE + (BROCHE6_7<<2)  ;; @data Register = @base +
(mask<<2) ==> Switcher

        ;;vvvvvvvvvvvvvvvvvvvvvvvvFin configuration Switcher


        ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CLIGNOTTEMENT

        str r3, [r6]        ;; Allume LED1&2 portF broche 4&5 : 00110000 (contenu de r3)

        ; Configure les PWM + GPIO
        BL      MOTEUR_INIT
```

```
                ; Activer les deux moteurs droit et gauche
                BL      MOTEUR_DROIT_ON
                BL      MOTEUR_GAUCHE_ON

; Démarrage de l'evalbot
ReadState_ini

                BL      MOTEUR_GAUCHE_OFF
                BL      MOTEUR_DROIT_OFF



                ldr r8, = GPIO_PORTD_BASE + (BROCHE6<<2)
                ldr r8, [r8]
                CMP r8, #0x00
                BEQ salutation_fan

                ldr r8, = GPIO_PORTD_BASE + (BROCHE7<<2)
                ldr r8, [r8]
                CMP r8, #0x00
                BEQ ReadState ; Passer en mode course


                mov r4, #0              ;Initialisation du compteur

                B ReadState_ini
```

```
salutation_fan

                BL      MOTEUR_DROIT_ON
                BL      MOTEUR_GAUCHE_ON

                BL      MOTEUR_GAUCHE_AVANT
                BL      MOTEUR_DROIT_ARRIERE



                ldr r9, = GPIO_PORTF_BASE + (BROCHE4_5<<2)
                str r2, [r9]                    ;; eteint la led1_2
                ldr r1, = DUREE                 ;; pour la duree de la boucle d'attente1 (wait_ini)

wait_led_off_ini

                subs r1, #1
    bne wait_led_off_ini

    str r3, [r9]                                ;; allume la led1_2
                ldr r1, = DUREE                 ;; pour la duree de la boucle d'attente2 (wait2)

wait_led_on_ini

                subs r1, #1
                bne wait_led_on_ini

                adds r4, r4, #1                         ;; Incrémentation du compteur
                CMP r4, #9              ;; Vérification si le compteur a atteint la valeur maximale
```

```
                BEQ ReadState_ini                ;; Si oui, retourner au programme initial

                b salutation_fan


; Mode course
ReadState

                BL      MOTEUR_DROIT_ON
                BL      MOTEUR_GAUCHE_ON

                BL      MOTEUR_DROIT_AVANT
                BL      MOTEUR_GAUCHE_AVANT

                ldr r8, = GPIO_PORTE_BASE + (BROCHE1<<2)
                ldr r8, [r8]
                CMP r8, #0x00
                BEQ bumper_droit

                ldr r9, = GPIO_PORTE_BASE + (BROCHE2<<2)
                ldr r9, [r9]
                CMP r9, #0x00
                BEQ bumper_gauche

                mov r4, #0              ;Initialisation du compteur

                B ReadState
```

```
; Bumper Droit activer
bumper_droit

                BL      MOTEUR_GAUCHE_ARRIERE
                ldr r9, = GPIO_PORTF_BASE + (BROCHE4<<2)
                str r2, [r9]                    ;; eteint la led1
                ldr r1, = DUREE                 ;; pour la duree de la boucle d'attente1 (wait1)


wait_led_off_droit

                subs r1, #1
    bne wait_led_off_droit

    str r3, [r9]                                ;; allume la led1

                adds r4, r4, #1                 ;; Incrémentation du compteur

                CMP r4, #1                                          ;;
Vérification si le compteur a atteint la valeur maximale

                BEQ ReadState                   ;; Si oui, terminer le programme

                b bumper_droit


bumper_gauche
```

```
                BL      MOTEUR_DROIT_ARRIERE
                ldr r8, = GPIO_PORTF_BASE + (BROCHE5<<2)
                str r2, [r8]
                ldr r1, = DUREE                 ;; pour la duree de la boucle d'attente1 (wait1)

wait_led_off_gauche

                subs r1, #1
    bne wait_led_off_gauche

    str r3, [r8]                ;; Allume LED1&2 portF broche 4&5 : 00110000 (contenu de r3)
    ldr r1, = DUREE                         ;; pour la duree de la boucle d'attente2 (wait2)

                adds r4, r4, #1         ; Incrémentation du compteur

                CMP r4, #1              ;Vérification si le compteur a atteint la valeur maximale
                BEQ ReadState                           ; Si oui, terminer le programme
                b bumper_gauche

                nop
                END
```

Config_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 réfèrent 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                 EQU             0x0001 ; Valeures plus petites => Vitesse plus rapide exemple 0x192
                                                ; Valeures plus grandes => Vitesse moins rapide exemple 0x1B2


                AREA    |.text|, CODE, READONLY
                ENTRY

                ;; The EXPORT command specifies that a symbol can be accessed by other shared objects or executables.
                EXPORTMOTEUR_INIT
                EXPORTMOTEUR_DROIT_ON
                EXPORT  MOTEUR_DROIT_OFF
                EXPORT  MOTEUR_DROIT_AVANT
                EXPORT  MOTEUR_DROIT_ARRIERE
                EXPORT  MOTEUR_DROIT_INVERSE
                EXPORTMOTEUR_GAUCHE_ON
                EXPORT  MOTEUR_GAUCHE_OFF
                EXPORT  MOTEUR_GAUCHE_AVANT
                EXPORT  MOTEUR_GAUCHE_ARRIERE
                EXPORT  MOTEUR_GAUCHE_INVERSE


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
                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
                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


END
```

Explication des choix de configuration des GPIO utilisés

Pour le choix de configuration des GPIO utilisés, nous nous sommes grandement inspirés de ce qui a été fait lors des TP précédents notamment avec l'utilisation du fichier Rk_Config_Moteur.s

Voici un exemple de configuration pour les 2 bumpers (très similaire à la configuration des switchs)

```
BROCHE1             EQU     0x01        ; bumperD
BROCHE2             EQU     0x02        ; bumperG
BROCHE1_2           EQU     0x03        ; bumperD_G

;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CONFIGURATION Bumper

ldr r7, = GPIO_PORTE_BASE+GPIO_I_PUR    ;; Pul_up
ldr r0, = BROCHE1_2
str r0, [r7]

ldr r7, = GPIO_PORTE_BASE+GPIO_O_DEN    ;; Enable Digital Function
ldr r0, = BROCHE1_2
str r0, [r7]

ldr r7, = GPIO_PORTE_BASE + (BROCHE1_2<<2)  ;; @data Register = @base + (mask<<2) ==> Bumper

;vvvvvvvvvvvvvvvvvvvvvvvvvvFin configuration Bumper
```

Déroulement du programme

Lors du démarrage du robot, les 2 leds s'allument et l'evalbot est dans l'attente de l'appui d'un des 2 switchs.

Si le switch 1 est appuyé, le mode « Salutations des fans » est activé. Le robot tourne autour de lui-même tout en clignotant les leds. Une fois 10 clignotements atteint, le robot revient à son état initial.

Si le switch 2 est appuyé, le mode « Course » est activé. Le robot va en ligne droite,

si l’un des bumpers est activé, la LED de ce côté s’éteint et une rotation légère du côté opposé s’effectue. Le but est de suivre le mur de la piste un maximum afin de gagner le plus de temps.

Apres avoir fait nos différents test et essais nous pensons avoir trouver la stratégie la plus optimale qui répond à l’objectif de notre projet, nous avons beaucoup travailler sur la rotation du robot lors d’une collision ce qui nous permettra potentiellement de gagner un maximum de temps