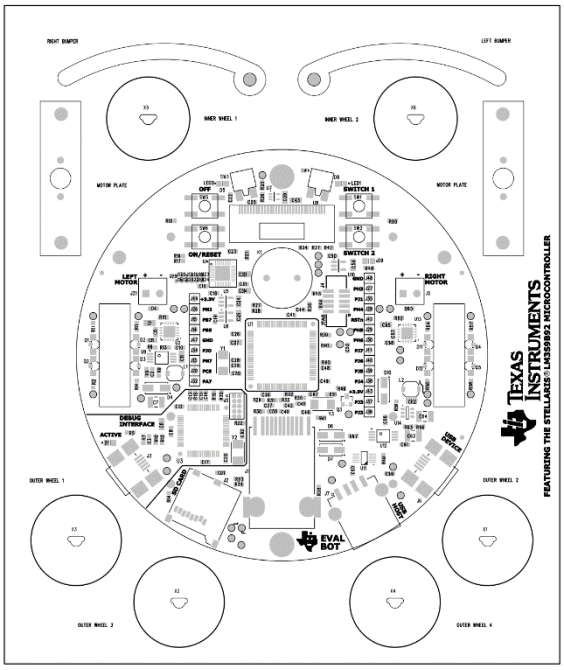
ESIEE Paris

Projet kit Evalbot

E3FI – Groupe 2



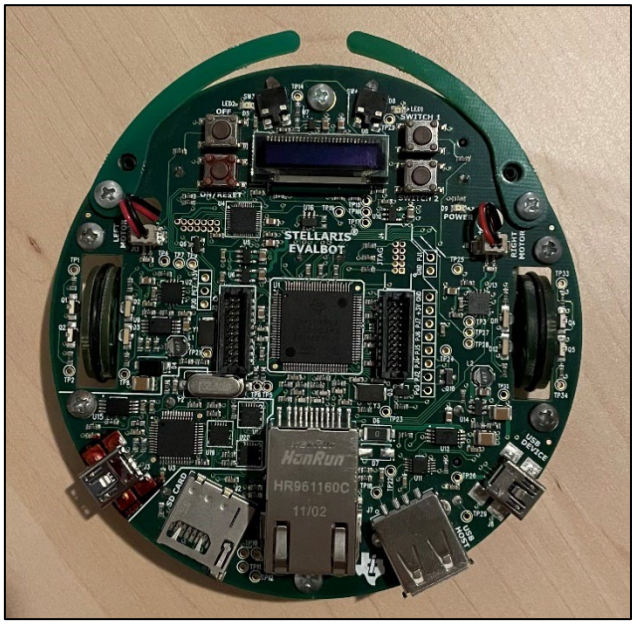
Alex Foulon & Erwan Gautier
22/11/2022

Table des matières

Description du projet

Dans le cadre du module d'Architecture, nous avons à l'ESIEE Paris un projet consistant à développer sur le kit Evalbot développé par Texas Instrument. Ce dernier nous est fourni par l'école et doit être programmé en assembleur, il est équipé d'un Cortex-M3, c'est un processeur RISC 32-bits d'architecture ARM servant à la fois de microprocesseur et de microcontrôleur.

Nous avons travaillé en binôme sur ce projet, nous avons commencé le projet par la programmation simple de chaque périphérique séparément à l'aide des code fournis. Une fois le code permettant de contrôler les LEDs, les bumpers, les switchs et les moteurs, nous avons imaginé différents scénarios que le robot pourrait exécuter. La programmation se fait sur l'IDE Keil µVision5.



Vu d'ensemble du robot

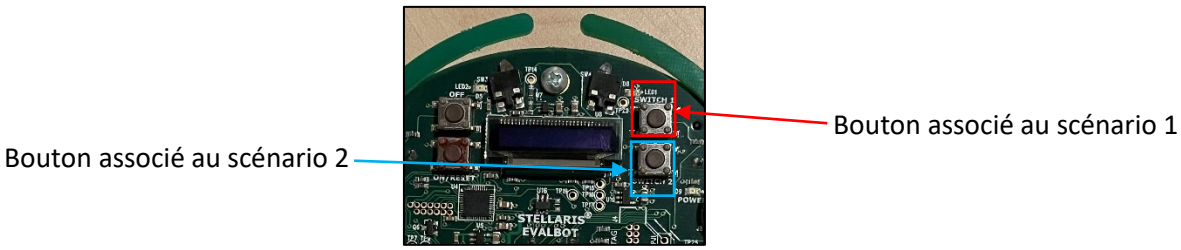
Le robot inclue les fonctionnalités suivantes :

- 2 moteurs commandés en PWM par des ponts en H
- Un afficheur OLED graphique
- Une interface de téléchargement de code, débogage par USB (type JTAG)
- 2 LEDs
- 2 switchs
- 2 bumpers
- USB, ethernet, etc.

Les scenarios

L'objectif du premier scénario était de proposer un scénario permettant l'utilisation de tous les périphériques étudiés, à savoir : les LEDs, les switchs, les bumpers et les moteurs.

Pour commencer, il faut allumer le robot en appuyant sur le bouton ON, ensuite on choisit le scénario que le robot veut exécuter (le 1 ou le 2) en appuyant sur le bouton qui est associé au scénario. Une fois le scénario terminé, il faut éteindre le robot et le rallumer pour exécuter à nouveau le scénario voulu.

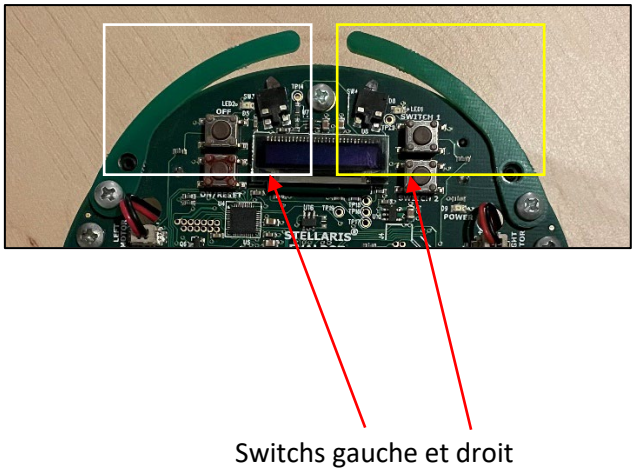


Scénario 1

Le premier scénario que nous avons développé permet au robot de se déplacer tout droit à l'aide des moteurs. Dès que ce dernier rencontre un objet, le bumper en collision va passer de l'état haut à l'état bas, ainsi le robot va reculer puis tourner pour éviter l'obstacle et avancer de nouveau. Lorsque l'un des bumpers change d'état, la LED qui lui est associée va s'allumer le temps que le robot change de direction. Le robot peut également faire demi-tour si les deux switchs sont activés.

Scénario 2

Pour le deuxième scénario, nous souhaitons que l'utilisateur programme à l'aide des switchs le parcours que le robot va faire. Pour commencer, il faut que l'utilisateur appuie sur le bouton n°2 afin de sélectionner le deuxième scénario, ensuite il peut enregistrer jusqu'à 32 instructions. Pour ce faire l'utilisateur doit appuyer sur le switch de gauche pour dire au robot d'aller à gauche durant sa course et inversement pour le switch de droite. Par exemple si l'utilisateur souhaite faire : gauche – gauche – droite – gauche – droite – droite, il doit faire la même série avec les switchs associés à la direction. Ensuite il faut appuyer sur un des deux boutons poussoirs et le robot commence le scénario imaginé par l'utilisateur.



GPIO utilisés

Switchs & bumpers

Les switchs et les bumpers fonctionnent de la même manière d'un point vu électronique, la seule différence c'est la manière dont nous appliquons la force mécanique nécessaire pour les actionner.

Lorsque l'on appuie sur le contact, le circuit entre la terre et le switch/bumper se ferme, ainsi le courant est nul. C'est-à-dire que nous lisons une valeur égal à 0 quand ils sont appuyés et une valeur égale à 1 quand ils sont relâchés.

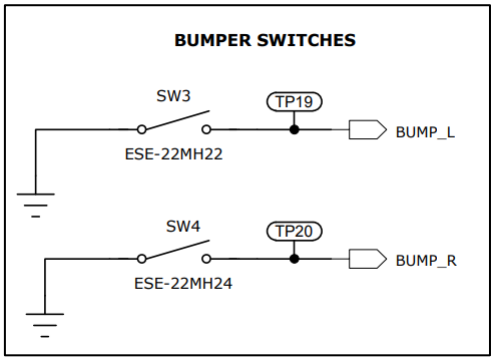


Schéma électrique des bumpers

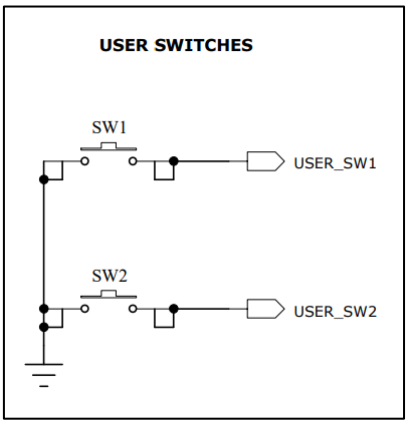
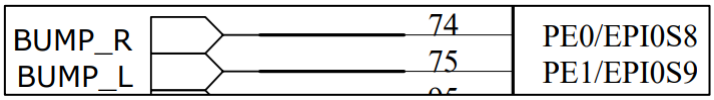


Schéma électrique des switchs

Bumpers

Dans la datasheet, nous trouvons également le port et les broches auxquels sont connectés les bumpers. Ici les bumpers sont sur le port E, le droit est sur la broche 0 et le gauche sur la broche 1.



Label des bumpers vers leurs entrées

Sur le Cortex-M3, les registres font 8 bits, ainsi les cas possibles que nous pouvons lire pour les bumpers sont :

Valeur lue dans le registre associé	Interprétation physique
0000 0000	Les deux bumpers sont activés
0000 0001	Le bumper de gauche est activé
0000 0010	Le bumper de droite est activé
0000 0011	Aucun bumper n'est activé

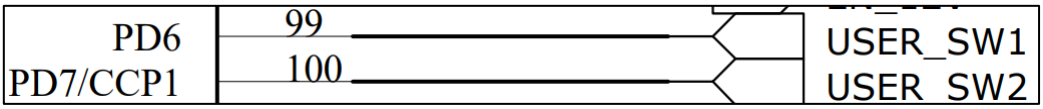
En assembleur la configuration des bumpers donne ça :

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

ldr r8, = GPIO_PORTE_BASE+GPIO_I_PUR    ;; Pul_up
ldr r0, = BROCHE0_1
str r0, [r8]

ldr r8, = GPIO_PORTE_BASE+GPIO_O_DEN    ;; Enable Digital Function
ldr r0, = BROCHE0_1
str r0, [r8]

ldr r8, = GPIO_PORTE_BASE + (BROCHE0_1<<2)  ;; @data Register = @base + (mask<<2) ==> Bumper

;vvvvvvvvvvvvvvvvvvvvvvvvvvFin configuration Bumper
```

Nous utilisons le registre R8 pour stocker les valeurs des bumpers.

Switchs

Dans la datasheet, nous trouvons également le port et les broches auxquels sont connectés les switchs. Ici les bumpers sont sur le port D, le droit est sur la broche 6 et le gauche sur la broche 7.



Label des switchs vers leurs entrées

Les cas possibles que nous pouvons lire pour les switchs:

Valeur lue dans le registre associé	Interprétation physique
0000 0000	Les deux switchs sont activés
0100 0000	Le switch de gauche est activé
1000 0000	Le switch de droite est activé
1100 0000	Aucun switch n'est activé

En assembleur la configuration des switchs donne ça :

```
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^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

;vvvvvvvvvvvvvvvvvvvvvvvvvvFin configuration Switcher
```

Nous utilisons le registre R7 pour stocker les valeurs des switchs.

LEDs

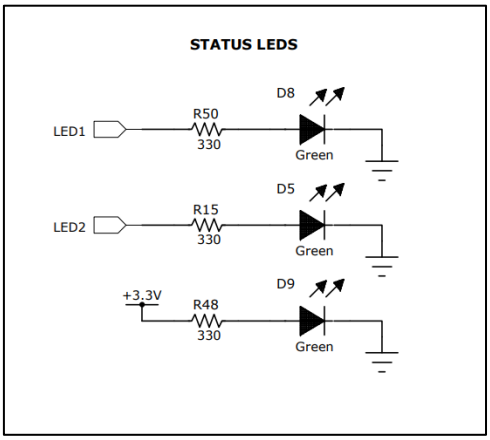
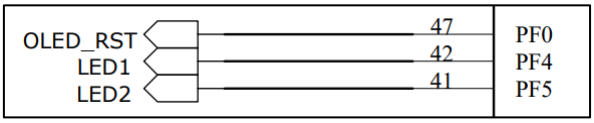


Schéma électrique des LEDs

Dans la datasheet, nous trouvons également le port et les broches auxquels sont connectés les bumpers. Ici les LEDs sont sur le port F, celle de droite est sur la broche 4 et celle de gauche sur la broche 5.



Label des LEDs vers leurs sorties

Les cas possibles que nous pouvons lire pour les leds sont :

Valeur écrite dans l'emplacement mémoire	Interprétation physique
0000 0000	Les deux LEDs sont éteintes
0001 0000	La LED de gauche est allumée
0010 0000	La LED de droite est allumée
0011 0000	Les deux LED sont allumées

En assembleur la configuration des switchs donne ça :

```
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^CONFIGURATION 2 LEDs

ldr r6, = GPIO_PORTF_BASE+GPIO_O_DIR    ;; 1 Pin du portF en sortie (broche 4 : 00010000)
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'intensit? de sortie (2mA)
ldr r0, = BROCHE4_5
str r0, [r6]

 mov r2, #0x000                         ;; pour eteindre LED

; allumer la led broche 4 (BROCHE4_5)
mov r3, #BROCHE4_5      ;; Allume LED1&2 portF broche 4&5 : 00110000

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

;vvvvvvvvvvvvvvvvvvvvvvvvFin configuration LED
```

Nous utilisons le registre R7 pour stocker les valeurs des switchs.

Codes

La première boucle, permet à l'utilisateur de choisir quel scénario il souhaite exécuter. Pour exécuter le scénario 1, il faut appuyer sur le bouton poussoir 1 ou le bouton poussoir 2 pour le scénario 2.

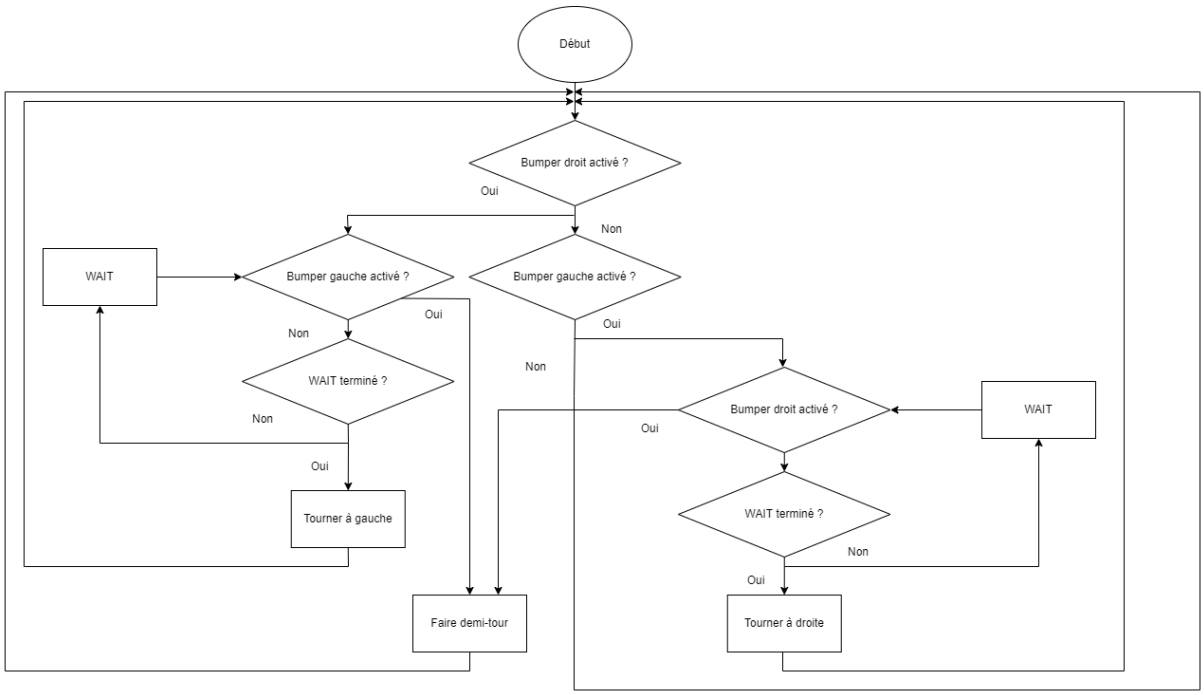
```
125         ;Boucle permettant de choisir le programme 1 ou 2 en pressant le bouton associé
126 ChooseProgram
127         ldr r10, [r7]
128         CMP r10, #0x80 ; Check if switch 1 is pushed
129         BEQ Program1
130         CMP r10, #0x40 ; Check if switch 2 is pushed
131         BEQ Program2
132         B ChooseProgram
```

Scénario 1

Au début du programme 1, nous initialisons les moteurs. Ensuite, le robot avance tout en vérifiant l'état des bumpers en continu. Si un des bumpers est actionné, le robot attend un court délai pour détecter si le deuxième l'est aussi. Si les deux sont actionnés, le robot fait demi-tour, sinon, il tourne dans la direction opposée à l'obstacle puis continue sa course. Chaque changement de direction est signalé par la correspondante.

```
153  CheckBumpers
154          ldr r10, [r8]
155          CMP r10, #0x02 ; Check if right one is pushed
156          BEQ waitBumperRight
157          CMP r10, #0x01 ; check if left one is pushed
158          BEQ waitBumperleft
159
160          B TurnOffLeds ; if none, turn off leds
```

Voici le diagramme expliquant l'algorithme du scénario 1 :



Scénario 2

Le scénario 2 est plus intéressant d'un point de vue programmation, même si celui-ci utilise les entrées / sorties du robot, le challenge était dans l'algorithme. En effet nous devions stocker les directions choisies, puis les lire à nouveau. Pour cela, au lieu d'utiliser un tableau classique, nous avons eu l'idée de stocker les décisions dans un seul registre de sorte à ce que chaque bit représente une direction (de droite à gauche).

Une première boucle permet à l'utilisateur de programmer le parcours du robot en appuyant sur le bumper associé à sa direction.

Prenons l'exemple suivant :

L'utilisateur souhaite faire le parcours suivant : gauche – gauche – droite – gauche – droite – droite – gauche

Nous traduisons son parcours de la manière suivante : 110100b → 0x34

Soit le **bit 1 pour aller à gauche** et le **bit 0 pour aller à droite**.

Initialement nous ajoutions 1 ou 0 au registre puis nous décalions à gauche, en répétant cette étape les directions étaient bel et bien enregistrées mais dans l'ordre inverse. Nous avons donc pensé l'algorithme suivant qui permet de ranger les bits du plus ancien à droite au plus récent à gauche, voici notre algorithme appliqué à l'exemple 1101001 :

Nous voulons enregistrer dans le registre l'inverse pour que ça soit lu dans le bon ordre, soit : 1001011.

Nous avons R5 notre compteur et R4 où nous écrivons le parcours.

Etape	Lu	Etat de R5	Etat de R4
1	Gauche (1)	0	$0 + 2^0 = 1$
2	Gauche (1)	1	$1 + 2^1 = 3$
3	Droite (0)	2	$3 + 0 = 3$
4	Gauche (1)	3	$3 + 2^3 = 11$
5	Droite (0)	4	$11 + 0 = 11$
6	Droite (0)	5	$11 + 0 = 11$
7	Gauche (1)	6	$11 + 2^6 = 75$

Nous retrouvons avec notre algorithme le parcours écrit dans le bon sens dans le registre, maintenant nous devons lire un à un les n premiers bits du registre 4 (n correspondant à la valeur du compteur).

Pour cela, nous utilisons l'opérateur AND pour savoir si le bit est à 0 ou 1 :

$$\text{Bit } 1 : 100\ 1011\ \&\ 000\ 0001 = 1$$

$$\text{Bit } 2 : 100\ 1011\ \&\ 000\ 0010 = 2$$

$$\text{Bit } 3 : 100\ 1011\ \&\ 000\ 0100 = 0$$

$$\text{Bit } 4 : 100\ 1011\ \&\ 000\ 1000 = 8$$

$$\text{Bit } 5 : 100\ 1011\ \&\ 001\ 0000 = 0$$

$$\text{Bit } 6 : 100\ 1011\ \&\ 010\ 0000 = 0$$

$$\text{Bit } 7 : 100\ 1011\ \&\ 100\ 0000 = 64$$

Nous observons que si l'on va à gauche, le résultat est supérieur à 0, sinon, il est égal à 0.

Nous automatisons cette étape de calcul en utilisant le décalage à gauche (LSL en assembleur).

Conclusion

Ce projet nous a permis de mieux comprendre comment fonctionne la programmation en assembleur, avec la gestion des périphériques, de la mémoire, etc. Pouvoir programmer sur un robot permet de comprendre plus facilement ce que l'on code, car ça un impact physique sur l'objet. L'approche d'un point vu électronique n'est pas forcément évident, mais cela permet de mieux comprendre le fonctionnement des objets qui nous entourent, même si nous ne sommes pas allés en profondeur dans le sujet.

Nous n'avons pas pu explorer en profondeur les fonctionnalités du kit Evalbot. Nous aurions aimé programmer l'écran OLED présent sur le robot, mais s'agissant d'une tâche complexe, nous nous sommes concentrés sur nos scénarios, qui nous ont demandé un travail de recherche (surtout le scénario 2).

Pour le scénario 2, nous avons travaillé bit par bit avec un seul registre, nous aurions bien entendu pu faire différemment en stockant chaque valeur dans la mémoire. Pour des raisons d'optimisation et de simplicité nous avons choisis de travailler sur un seul registre. Nous voyons bien que cela continent des limites (32 décisions maximum), mais dans notre cas nous pouvons nous le permettre. Il serait toutefois intéressant de développer une solution plus puissante, avec des choix non-binaires (avancer tout droit par exemple) et un nombre illimité de décisions.

Index

Code

GitHub: https://github.com/X13-A/Evalbot.git

```
                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)
GPIO_PORTD_BASE         EQU         0x40007000          ; GPIO Port D (APB) base:
0x4000.7000 (p416 datasheet de lm3s9B92.pdf)
GPIO_PORTE_BASE         EQU         0x40024000          ; GPIO Port E (APB) base:
0x4002.4000 (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_5               EQU         0x30            ; led1 & led2 sur broche 4 et 5

BROCHE4                 EQU     0x10            ; led1 sur broche 4

BROCHE5                 EQU     0x20            ; led2 sur broche 5

BROCHE6_7               EQU     0xC0            ; bouton poussoir 1 et 2 sur broche 6 et 7

BROCHE6                     EQU     0x40            ; bouton poussoir 1 sur broche 6

BROCHE7                     EQU     0x80            ; bouton poussoir 2 sur broche 7

BROCHE0_1               EQU     0x03            ; bumpers 1 et 2 sur broche 0 et 1

                ENTRY
```

```
                EXPORT__main

                ;; The IMPORT command specifies that a symbol is defined in a shared object at
runtime.
                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

__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 o?
sont branch?s les leds (0x28 == 0b111000)
                ; ;;

            (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 2 LEDs

    ldr r6, = GPIO_PORTF_BASE+GPIO_O_DIR    ;; 1 Pin du portF en sortie (broche 4 : 00010000)
    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'intensit? de sortie
(2mA)
    ldr r0, = BROCHE4_5
    str r0, [r6]

                 mov r2, #0x000                                          ;; pour eteindre LED

                ; allumer la led broche 4 (BROCHE4_5)
                mov r3, #BROCHE4_5              ;; Allume LED1&2 portF broche 4&5 : 00110000

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

                ;vvvvvvvvvvvvvvvvvvvvvvvFin configuration LED

                ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^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

                ;vvvvvvvvvvvvvvvvvvvvvvvFin configuration Switcher

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

                ldr r8, = GPIO_PORTE_BASE+GPIO_I_PUR        ;; Pul_up
    ldr r0, = BROCHE0_1
    str r0, [r8]

                ldr r8, = GPIO_PORTE_BASE+GPIO_O_DEN        ;; Enable Digital Function
    ldr r0, = BROCHE0_1
    str r0, [r8]

                ldr r8, = GPIO_PORTE_BASE + (BROCHE0_1<<2)  ;; @data Register = @base +
(mask<<2) ==> Bumper

                ;vvvvvvvvvvvvvvvvvvvvvvvFin configuration Bumper

                ;Boucle permettant de choisir le programme 1 ou 2 en pressant le bouton associé
ChooseProgram
```

```
                ldr r10, [r7]
                CMP r10, #0x80 ; Check if switch 1 is pushed
                BEQ Program1
                CMP r10, #0x40 ; Check if switch 2 is pushed
                BEQ Program2
                B ChooseProgram


                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Début programme 1;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Program1
                BL WAIT
                BL WAIT
                ; Configure les PWM + GPIO
                BL      MOTEUR_INIT
                ; Activer les deux moteurs droit et gauche
                BL      MOTEUR_DROIT_ON
                BL      MOTEUR_GAUCHE_ON
                ; Evalbot avance droit devant
                BL      MOTEUR_DROIT_AVANT
                BL      MOTEUR_GAUCHE_AVANT
                B CheckBumpers

CheckBumpers
                ldr r10, [r8]
                CMP r10, #0x02 ; Check if right one is pushed
                BEQ waitBumperRight
                CMP r10, #0x01 ; check if left one is pushed
                BEQ waitBumperleft
                B TurnOffLeds ; if none, turn off leds

; Allume les LEDs
TurnOnLeds
                ldr r6, = GPIO_PORTF_BASE + (BROCHE4_5<<2)
                ldr r3, = BROCHE4_5
                str r3, [r6]
                B HalfTurn

;Allume la LED 1
TurnOnLed1Program1
                ldr r6, = GPIO_PORTF_BASE + (BROCHE5<<2)
                ldr r3, = BROCHE5
                str r3, [r6]
                B LeftDirection

;Allume la LED 2
TurnOnLed2Program1
                ldr r6, = GPIO_PORTF_BASE + (BROCHE4<<2)
                ldr r3, = BROCHE4 ; Turns on Led 1
                str r3, [r6]
                B RightDirection
```

```
;Eteind les deux LEDs
TurnOffLeds
                ldr r6, = GPIO_PORTF_BASE + (BROCHE4_5<<2)
                str r2, [r6]
                B CheckBumpers

;Tourne à droite et avance
RightDirection
                BL      MOTEUR_DROIT_ARRIERE
                BL      MOTEUR_GAUCHE_ARRIERE
                BL WAIT
                BL      MOTEUR_DROIT_ARRIERE
                BL      MOTEUR_GAUCHE_AVANT
                BL      WAIT
                BL      MOTEUR_DROIT_AVANT
                B CheckBumpers

;Tourne à gauche et avance
LeftDirection
                BL      MOTEUR_DROIT_ARRIERE
                BL      MOTEUR_GAUCHE_ARRIERE
                BL WAIT
                BL      MOTEUR_GAUCHE_ARRIERE
                BL      MOTEUR_DROIT_AVANT
                BL      WAIT
                BL      MOTEUR_GAUCHE_AVANT
                B CheckBumpers

;Fait un demi-tour et avance
HalfTurn
                BL      MOTEUR_DROIT_ARRIERE
                BL      MOTEUR_GAUCHE_ARRIERE
                BL WAIT
                BL      MOTEUR_GAUCHE_ARRIERE
                BL      MOTEUR_DROIT_AVANT
                BL WAIT
                BL WAIT
                BL      MOTEUR_GAUCHE_AVANT
                B CheckBumpers

; Boucle d'attente pour bumper droit
;Permet d'attendre afin de détecter l'activation du deuxième bumper avant d'éxecuter l'action
suivante
waitBumperRight
                ldr r1, =0x2BFFF
wait2
                ldr r10, [r8]
                CMP r10, #0x0 ; Check if left and right are pushed
                BEQ TurnOnLeds
                subs r1, #1
    bne wait2
                B TurnOnLed1Program1
```

```
;Boucle d'attente pour bumper gauche
;Permet d'attendre afin de détecter l'activation du deuxième bumper avant d'éxecuter l'action suivante
waitBumperleft
                ldr r1, =0x2BFFF
wait3
                ldr r10, [r8]
                CMP r10, #0x0 ; Check if left and right are pushed
                BEQ TurnOnLeds
                subs r1, #1
    bne wait3
                B TurnOnLed2Program1

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Fin programme 1;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Début programme 2;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Program2
                ldr r4, =0x0 ; Directions
                ldr r5, =0x0 ; Compteur
                ldr r9, =0x0 ; Pour copier R4 dans R9
                ldr r2, =0x1 ; Pour le décalage à gauche et le calcul and
                ldr r0, =0x2 ; Pour la constante de 2^n
                ; Configure les PWM + GPIO
                BL      MOTEUR_INIT
                BL WAIT

;Enregistrement des directions lues par les bumpers
Input
                ldr r10, [r8]
                ldr r11, [r7]
                CMP r10, #0x02 ; Check if right one is pushed
                BEQ AddRight
                CMP r10, #0x01 ; check if left one is pushed
                BEQ AddLeft
                CMP r11, #0xC0 ; Check if switch 1 is pushed
                BNE StartCycle
                B Input

;Ajouter la valeur gauche au registre R5
AddLeft
                LDR r1, =0x1 ; Pour la puissance à 0 qui vaut 1 par convention
                CMP r5, #0
                BEQ SequenceAddLeft
                MOV r12, r5
                B PowerLoop

SequenceAddLeft
                ADD r5, #1
                ADD r4, r4, r1
```

```
                BL TurnOnLed1Program2
                BL WAIT
                LDR r12, =0x0
                BL TurnOffLedsProgram2
                B Input

;Allume la LED 2 quand le bumper 1 est pressé
TurnOnLed1Program2
                ldr r6, = GPIO_PORTF_BASE + (BROCHE5<<2)
                ldr r3, = BROCHE5
                str r3, [r6]
                BX LR

;Calcul de la puissance 2^n
PowerLoop
                MUL r1, r1, r0
                SUB r12, #1
                CMP r12, #0
                BEQ SequenceAddLeft
                B PowerLoop

;Ajouter la valeur droite au registre R5
AddRight
                ADD r5, #1
                BL TurnOnLed2Program2
                BL WAIT
                BL TurnOffLedsProgram2
                B Input

;Allume la LED 1 quand le bumper 2 est pressé
TurnOnLed2Program2
                ldr r6, = GPIO_PORTF_BASE + (BROCHE4<<2)
                ldr r3, = BROCHE4 ; Turns on Led 1
                str r3, [r6]
                BX LR

;Eteind les deux LEDs
TurnOffLedsProgram2
                ldr r6, = GPIO_PORTF_BASE + (BROCHE4_5<<2)
                str r2, [r6]
                BX LR

;Lecture des instructions enregistrées
StartCycle
                BL WAIT
                ; Activer les deux moteurs droit et gauche
                BL      MOTEUR_DROIT_ON
                BL      MOTEUR_GAUCHE_ON
                ; Evalbot avance droit devant
                BL      MOTEUR_DROIT_AVANT
                BL      MOTEUR_GAUCHE_AVANT
                BL WAIT
```

```
                BL WAIT

;Permet d'éxécuter les actions enregistrées par l'utilisateur
ExecuteDirections
                AND r9, r4, r2
                CMP r5, #0x0
                BEQ.W EndProgram
                SUB r5, #1
                BL Motors
                LSL r2, r2, #0x1
                B ExecuteDirections

Motors
                PUSH {LR}
                CMP r9, #0x0
                BNE TurnLeft
                B TurnRight

TurnRight
                BL      MOTEUR_DROIT_ARRIERE
                BL WAIT
                BL      MOTEUR_DROIT_AVANT
                BL WAIT
                ldr r9, =0x0
                POP {LR}
                BX      LR

TurnLeft
                BL      MOTEUR_GAUCHE_ARRIERE
                BL WAIT
                BL      MOTEUR_GAUCHE_AVANT
                BL WAIT
                ldr r9, =0x0
                POP {LR}
                BX      LR

                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Fin programme 2;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;Boucle d'attente
WAIT
                ldr r1, =0x2BFFFF
wait1
                subs r1, #1
    bne wait1
                ;; retour à la suite du lien de branchement
                BX      LR

EndProgram
                BL      MOTEUR_DROIT_OFF
                BL      MOTEUR_GAUCHE_OFF
```

```
        nop
        END
```