DJIRÉ Aboubakar Malick et ADJE Axel

Rapport-Projet-G1d06-IGI-3001

ROBOT EXPLORATEUR PROGRAMMABLE

TABLE DES MATIERES:

Description du projet:

Notre robot nommé “Explora Bot-3000” est un drone terrestre concu pour faire de la reconnaissance dans des environnements risqués: champs de mines, zones contaminées, ou explorations extraterrestres. Le robot une fois mis sous tension entre dans un mode programmation ou l’utilisateur décidera du nombre n de pas qu’il souhaite que le robot fasse en cliquant sur le switch 1 n fois, il met ensuit fin à la phase de programmation en cliquant sur le switch 2.

Vient ensuite la phase d’execution où le robot équipé de capteurs avance tout en évitant les obstacles pour cartographier ou examiner une zone. La détection d'obstacles et la capacité à ajuster sa direction automatiquement sont des fonctionnalités programmées par les concepteurs (nous). Une fois que le robot a finit d’avancer le nombre de fois demandé, il s’arrête et entre de nouveau en mode programmation, prêt à recevoir de nouvelles consignes.

Scenarios:

Nous allons envisager 4 scénarios différents:

1. L'utilisateur n'appuis pas sur le switch 1 durant la phase de programmation

→ Dans ce cas le robot n'avance pas.

2. L'utilisateur appuis n fois sur le switch 1 durant la phase de programmation et il n'y a aucun obstacle sur la trajectoire du robot.

→ Dans ce cas le robot avance n fois.

3. L'utilisateur appuis n fois sur le switch 1 durant la phase de programmation et il y a un obstacle sur le coté gauche du robot.

→ Dans ce cas le robot avance jusqu'à l'obstacle, il le detecte grace au bumper gauche, allume uniquement la led droite et tourne à droite. Une fois qu'il a tourné, il ré-allume les deux LEDs, puis continu à avancer du nombre de pas qu'il lui reste à faire.

4. L'utilisateur appuis n fois sur le switch 1 durant la phase de programmation et il y a un obstacle sur le coté droit du robot.

→ Dans ce cas le robot avance jusqu'à l'obstacle, il le detecte grace au bumper droit, allume uniquement la led gauche et tourne à gauche. Une fois qu'il a tourné, il ré-allume les deux LEDs, puis continu à avancer du nombre de pas qu'il lui reste à faire.

Explication du choix des configurations des GPIO et initialisations:

code.s | config_led.s

```
52          AREA MyCode, CODE, READONLY
53          ENTRY
54
55          ;On exporte toute les fonctions qui seront utilisées dans le code principal.
56          EXPORT LED_INIT
57          EXPORT SWITCH_INIT
58          EXPORT COUNT_CLICK
59          EXPORT CLOCK_INIT
60          EXPORT BUMPER_INIT
61          EXPORT ReadState_BUMPER1
62          EXPORT ReadState_BUMPER2
63
64
65  CLOCK_INIT
66          ldr r6, = SYSCTL_PERIPH_GPIO
67          mov r0, #0x00000038                 ;; Active l'horloge sur les GPIO D et F E où  sont branchés tous les périphériques utiles (0x38 == 0b111000)
68          ; ;;                                                                                                                    (GPIO::FEDCBA)
69          str r0, [r6]
70
71          ; ;; "There must be a delay of 3 system clocks before any GPIO reg. access  (p413 datasheet de lm3s9B92.pdf)
72          nop                                 ;; tres tres important....
73          nop
74          nop
75          BX LR
76
77
78  LED_INIT
79          ldr r6, = GPIO_PORTF_BASE+GPIO_O_DIR    ;; mettre à 1 les pins 4 et 5 (00110000) à l'adresse du gpio_output_dir du port F définissant ainsi les pins comme des outputs.
80          ldr r0, = BROCHE4_5
81          str r0, [r6]
82
83          ldr r6, = GPIO_PORTF_BASE+GPIO_O_DEN    ;; mettre à 1 les pins 4 et 5 (00110000) à l'adresse du gpio_output_ditital_enable du port F définissant ainsi les pins comme des sorties numériques.
84          ldr r0, = BROCHE4_5
85          str r0, [r6]
86
87          ldr r6, = GPIO_PORTF_BASE+GPIO_O_DR2R   ;; mettre à 1 les pins 4 et 5 (00110000) à l'adresse du gpio_output_dr2r du port F définissant ainsi l'intensité du courant de sortie à 2mA.
88          ldr r0, = BROCHE4_5
89          str r0, [r6]
90
91          mov r2, #0x20
92          mov r9, #0x10
93
94          mov r3, #BROCHE4_5                  ;; va nous servir a allumer les deux LEDs
95          ldr r6, = GPIO_PORTF_BASE + (BROCHE4_5<<2)  ;;définition du data register des LEDs (@data Register = @adresse_base + (mask<<2) ==> LED1&2)
96
97          BX LR
```

```
100 SWITCH_INIT
101         ldr r8, = GPIO_PORTD_BASE+GPIO_I_PUR    ;; Pul_up
102         ldr r0, = BROCHE6_7
103         str r0, [r8]
104
105         ldr r8, = GPIO_PORTD_BASE+GPIO_O_DEN    ;; Enable Digital Function
106         ldr r0, = BROCHE6_7
107         str r0, [r8]
108
109         ldr r8, = GPIO_PORTD_BASE + (BROCHE6_7<<2)  ;; @data Register = @base + (mask<<2) ==> SWITCH1&2
110
111         BX LR
112
113
114 BUMPER_INIT
115         ldr r7, = GPIO_PORTE_BASE+GPIO_I_PUR    ;; Pul_up
116         ldr r0, = BROCHE0_1
117         str r0, [r7]
118
119         ldr r7, = GPIO_PORTE_BASE+GPIO_O_DEN    ;; Enable Digital Function
120         ldr r0, = BROCHE0_1
121         str r0, [r7]
122
123         ldr r7, = GPIO_PORTE_BASE + (BROCHE0_1<<2)  ;; @data Register = @base + (mask<<2) ==> BUMPER1&2
124
```

Code assembleur principal:

code.s | config_led.s

```
158  ReadState_BUMPER1
159          ;Vérifie si le bumper1 à été touché.
160          ldr r10,[r7] ; bumper 1 appuyé
161          CMP r10,#0x02
162          BNE RETURN   ; retour vers le programme principal si le bumper n'est pas appuyé.
163          STR r2, [r6] ; allumage de la LED2
164          B   TOURNER_GAUCHE
165
166
167  ReadState_BUMPER2
168          ;Vérifie si le bumper2 à été touché.
169          ldr r10,[r7] ; bumper 2 appuyé
170          CMP r10,#0x01
171          BNE RETURN   ; retour vers le programme principal si le bumper n'est pas appuyé.
172          STR r9, [r6] ; allumage de la LED1
173          B   TOURNER_DROITE
174
175  RETURN
176          BX LR
177
178
179  TOURNER_GAUCHE
180          ;Fait tourner le moteur gauche en arrière.
181          ldr r11, =(GPIODATA_H+(GPIO_1<<2))
182          mov r0, #2
183          str r0,[r11]
184
185          LDR r0,= DUREE_D
186  wait2   subs r0, #1; mini boucle d'attente.
187          bne wait2
188
189          ;Fait tourner le moteur gauche de nouveau en avant.
190          ldr r11, =(GPIODATA_H+(GPIO_1<<2))
191          mov r0, #0
192          str r0,[r11]
193          BX LR
194
195
196  TOURNER_DROITE
197          ;Fait tourner le moteur droite en arrière.
198          ldr r11, =(GPIO_PORTD_BASE+(GPIO_1<<2))
199          mov r0, #0
200          str r0,[r11]
201
202          LDR r0,= DUREE_D
203  wait3   subs r0, #1; mini boucle d'attente.
204          bne wait3
205
206          ;Fait tourner le moteur gauche de nouveau en avant.
207          ldr r11, =(GPIO_PORTD_BASE+(GPIO_1<<2))
208          mov r0, #2
209          str r0,[r11]
210          BX LR
211
```

```
        nop
        END
```

Difficultés rencontrées et solutions apportées:

→ Le moteur recule durant la phase de programmation.

Les moteurs reculent automatiquement lorsqu'ils sont allumés, nous avons donc allumés les moteurs uniquement lorsqu'on etait sur le point de les utiliser.

→ Dans le fichier code lors de la boucle wait, on verifie si les bumpers sont appuyés, mais une fois que c'est fait, il n'arrive plus a continuer l'execution normale du programme. Lors de la vérification du statut des bumpers, on a fait un branchement: "BL ReadState_BUMPER1", et si le bumper est éffectivement appuyé, on a fait un branchement utilisant "BL", ce qui a override le registre "LR". Le code ne peut donc pas retourner au point de départ.

2 Solutions:

Soit ne plus utiliser "BL" pour éviter les conflits et réécrire entièrement la procédure pour faire tourner le robot.

Soit utiliser PUSH {LR} et POP {LR} pour enregistrer le contenu du registre "LR" et pouvoir le ré-utiliser plus tard.

→ La première phase de programmation ne fonctionne pas, c'est à dire que juste après avoir mis le robot sous tension, lorsqu'on appuis sur le switch1 n fois et qu'on valide en appuyant sur le switch2, le robot n'avance pas.

Nous n'avons pas réussi à régler ce problème, il faut ainsi sauter la première phase de programmation, en appuyant directement sur le switch2 après la mise sous tension.

→ On a remarqué que lorsqu'on vérifiait l'état des périphériques a l'intérieur des boucles d'attente, comme dans wait (ligne 76), le temps d'exécution augmente, les boucles d'attente prennent beaucoup plus de temps à se terminer.

Pour remédier à cela nous avons juste diminué la valeur dans le registre à décrémenter: r1 dans le cas de wait.

→ Au début de la conception du code, nous avons souvent rencontré des bugs innatendus, cela était du a des conflits entre les registres. Nous utilisions un registre pour faire quelque chose, mais il était modifié à notre ainsu quelque part d'autre, ce qui a conduit à énormément de bugs.

Pour éviter ce genre de problème, nous avions deux choix, configurer les registres à chaque fois qu'on voulait les utiliser.

OU

Tenir compte de quel registre est utilisé pour quoi. C'est ce que nous avons fait à la ligne 28.

→ Nous avons aussi rencontré un bug majeur cette fois ci avec la boucle d'attente wait, elle ne se terminait jamais.

Il se trouvait que nous avions changé l'indicateur z lors de la verification des switchs et des bumpers. Pour remédier à ce problème, nous avons remarqué que l'instruction subs doit être mise après les les branchements pour éviter que l'indicateur z se fasse override durant l'exécution du code de vérification des périphériques.