

12/01/2025

Hexarium CyberTrail

Projet Stellaris Evalbot



Présenté par : HAKIM Justine et GHAZANFAR Aman
Ingénieurs à ESIEE Paris
E3FI_1I_Groupe 5

Sommaire

1

Description du Projet

Objectif général, Scénario, Fonctionnalités développées

2

Codes Assembleurs Développés

Structure du Code, Explications des sections clés

3

Explications Techniques

Choix des GPIO, Justification des choix

4

Déroulement des Programmes

Organigramme

5

Conclusion

Résultats globaux, Problèmes rencontrés

1. Description du Projet

GitHub

Le code source de ce projet est disponible sur GitHub à l'adresse web suivante : <https://github.com/ZedRoff/Evalbot>

1.1 Objectif général

Le projet consiste à utiliser un Evalbot pour développer un robot qui doit infiltrer un autre Evalbot fictif afin de purger un virus de son système. Ce robot devra interagir avec divers composants de la carte Evalbot, notamment les LED, les switchs, les moteurs et les bumpers. L'objectif est de programmer un robot capable de suivre un chemin, de résoudre des énigmes et de faire face à des obstacles afin de purifier le système du virus.

1.2 Scénario

Nous sommes dans un laboratoire de recherche sur les langages de bas niveau, et c'est la panique totale aujourd'hui : un Evalbot a été infecté par un virus. Si ce virus parvient à infecter complètement cet Evalbot, cela pourrait compromettre l'ensemble du système. Face à cette menace, Hexarium intervient. Le robot démarre sur une case de départ, suivant un chemin dessiné sur une feuille. Il avance en ligne droite jusqu'à ce qu'un obstacle soit détecté par le bumper. Lors de la collision, une énigme sera proposée à l'utilisateur pour lui permettre de continuer son chemin. À chaque nœud du parcours, un obstacle oblige l'activation d'un des bumpers avant du robot. Il y a au total quatre épreuves à résoudre pour franchir le parcours.

Les épreuves sont les suivantes :

- Une question de type vrai/faux
- Un jeu de Simon
- Une énigme binaire
- Un test de vitesse

Épreuve n°1 : Vrai ou Faux - Le virus et les opérations logiques sur les registres

Dans le cadre de l'infection, le virus pourrait manipuler les registres internes du robot de manière inattendue. Une des opérations logiques qui pourrait être affectée est l'instruction XOR, qui est utilisée pour effectuer des comparaisons. Nous devons comprendre si une telle manipulation peut entraîner un résultat incorrect dans certaines situations. *L'instruction XOR entre deux registres identiques met-elle leur contenu à zéro ?*

À vous de jouer ! Utilisez les switchs pour répondre. La bonne réponse est "Vrai". Si deux registres ont les mêmes valeurs, l'opération XOR entre eux met effectivement leur contenu à zéro, car $1 \text{ XOR } 1 = 0$ et $0 \text{ XOR } 0 = 0$. Le virus perturbe donc le comportement du robot en effaçant des informations critiques pour le bon déroulement des opérations. Ce problème devra être résolu afin de s'assurer que les données ne soient pas altérées accidentellement, ce qui pourrait compromettre la mission de l'Evalbot.

A noter que si vous faites une erreur lors de cette question, les deux LED vont s'allumer et s'éteindre, vous indiquant qu'il faut réessayer.

Épreuve n°2 : Simon - L'infection altère les processus de mémoire et de décision

Le virus a perturbé les processus de mémoire de l'Evalbot, en rendant difficile la mémorisation de séquences complexes. Dans cette épreuve, l'Evalbot doit tester votre capacité à mémoriser une séquence et à la reproduire.

À vous de jouer ! Vous devez appuyer sur les switchs pour reproduire la séquence lumineuse. Le switch du haut correspond à la LED gauche, et le switch du bas à la LED droite. *Séquence à mémoriser : Droite, Gauche, Gauche, Droite.* Les LED de l'Evalbot s'allument pour former une séquence lumineuse. Si vous vous trompez, la séquence recommence. Faites attention à la mémoire du robot !

À noter que dans cette épreuve, vous pouvez appuyer sur le bumper gauche afin de visualiser à nouveau la trame du simon. Aussi, si vous faites une erreur, les deux LED vont s'allumer vous indiquant qu'il faut réessayer.

Épreuve n°3 : Calcul Binaire - Résolution de l'éénigme pour stopper l'infection

Le virus a laissé une trace numérique à travers l'adresse binaire de notre groupe. Cette adresse, manipulée par le virus, nécessite une série d'opérations binaires pour être corrigée.

À vous de jouer ! Résolvez cette énigme afin d'identifier le code clé nécessaire pour stopper l'infection. Le résultat du calcul doit être inscrit sur l'Evalbot. Les deux bumpers permettent d'allumer les LED du port femelle RJ45, tandis que les deux switchs contrôlent les LED à l'avant de l'Evalbot. *Prenez le numéro de notre groupe, convertissez-le en binaire. Effectuez un décalage à gauche, faites un XOR avec 1001, et ajoutez-y 8 en binaire.* Le numéro de notre groupe est 5, soit 101 en binaire. Effectuer un décalage à gauche : 101 devient 1010. Effectuer un XOR avec 1001 : 1010 XOR 1001 = 0011. Ajoutez-y 8 en binaire, et vous obtiendrez 1011 (soit 11 en binaire). Une fois toutes les LED allumées, l'Evalbot passe à l'épreuve suivante. Ce résultat est la clé qui permettra de stopper le virus et de rétablir le bon fonctionnement du robot.

À noter que dans cette épreuve, lorsque vous avez le bon résultat le robot se mettra à bouger tout seul.

Épreuve n°4 : Speed test - Vérification des réflexes du robot

Le virus a affecté la rapidité de réponse du robot, rendant difficile la gestion des événements à grande vitesse.

À vous de jouer ! Les LED de l'Evalbot vont s'allumer à des intervalles de plus en plus courts. Vous devez appuyer sur le switch correspondant à la LED allumée avant qu'elle ne s'éteigne. Si vous appuyez trop tard ou sur le mauvais switch, l'épreuve recommence. Faites preuve de rapidité et de précision pour prouver que l'Evalbot a récupéré ses capacités de réaction après l'infection.

A noter que dans cette épreuve, si vous n'êtes pas assez rapide, les deux LED vont s'allumer vous indiquant qu'il faut réussir et vous recommencerez le test de rapidité à partir de zéro.

À chaque épreuve réussie, l'Evalbot tourne de 90 degrés (vers la droite ou vers la gauche, selon le nœud), puis attend une nouvelle collision pour proposer une nouvelle énigme. Le but final est de résoudre les quatre épreuves et atteindre la case finale. Une fois cette étape franchie, l'Evalbot clignote et tourne sur lui-même. Il n'y a pas de condition de défaite : l'utilisateur peut tenter à nouveau une épreuve autant de fois que nécessaire jusqu'à la réussir.

1.3 Fonctionnalités développées

Pour réaliser le scénario décrit ci-dessus, nous avons développé plusieurs fonctionnalités essentielles permettant à l'Evalbot de mener à bien sa mission de purger le virus et de résoudre les différentes énigmes. Voici un aperçu des principales fonctionnalités mises en œuvre :

1 - Système de moteur

Un code de gestion des moteurs nous a été fourni, permettant de commander les mouvements de l'Evalbot. Nous avons dû analyser ce code, le comprendre et l'intégrer dans notre propre programme. Grâce à ce système, le robot est capable de se déplacer, de tourner, d'avancer et de reculer de manière fluide et contrôlée tout au long du parcours.

2 - Détection d'obstacle

Pour détecter les obstacles placés sur le chemin, nous avons utilisé les bumpers de l'Evalbot. Lorsqu'une collision se produit, le bumper envoie un signal qui permet d'interrompre l'avancement du robot. Cette détection est essentielle pour le bon déroulement des épreuves, car elle déclenche la proposition de l'énigme suivante.

3 - Système de LED

Les LED jouent un rôle-clé dans l'interaction avec l'utilisateur lors de certaines épreuves. Dans l'épreuve 3, les LED sont autonomes et peuvent clignoter en fonction des réponses du robot. À la fin du parcours, les LED du robot clignotent également pour signaler la fin de la mission. En outre, elles sont également utilisées de manière manuelle dans certaines épreuves, permettant à l'utilisateur de les activer directement pour résoudre les défis proposés.

4 - Activation des LED à travers les switchs

Les switchs de l'Evalbot sont utilisés pour contrôler les LED dans les épreuves 2 et 3. Dans l'épreuve 2, les switchs permettent à l'utilisateur de reproduire une séquence de LED, tandis que dans l'épreuve 3, les switchs servent à allumer les LED à l'avant de l'Evalbot pour compléter le calcul binaire. Le système d'activation des LED par les switchs offre une interface simple et intuitive pour l'utilisateur, rendant les épreuves interactives et ludiques.

Ces fonctionnalités ont été intégrées de manière cohérente pour permettre au robot de résoudre les épreuves et de suivre le chemin jusqu'à la case finale, purifiant ainsi le virus du système.

5 - Compte à rebours

Trois compte à rebours ont été développés, deux pour des temps d'attentes courts et longs, et un dans la dernière épreuve de "speed test" où si l'utilisateur n'est pas assez rapide, on repasse à l'état initial.

6 - Machine à état

Simulation d'une machine à état avec un registre qui s'incrémente, pour l'épreuve du simon, pour gérer les différentes étapes.

2. Codes Assembleurs Développés

2.1 Structure du Code

Notre code est organisé en plusieurs fichiers distincts, chacun correspondant à un composant spécifique du système. Cette organisation permet de séparer clairement les fonctionnalités, afin de faciliter leur gestion. Chaque fichier contient des méthodes et des instructions nécessaires pour la configuration et l'activation des composants, et tous ces fichiers sont appelés dans le fichier principal main.s.

Voici l'arborescence du projet, détaillant les fichiers et leurs rôles :

1. bumper.s

Ce fichier contient les méthodes pour activer et gérer les bumpers. Il permet de gérer les interactions avec les bumpers situés sur le port E, en activant les résistances pull-up et la fonction numérique.

2. LED.s

Le fichier est dédié à l'activation des LED. Il gère les registres nécessaires pour configurer la direction des pins (sortie), leurs intensités, et la fonction numérique des LED. Ce fichier contient également les méthodes pour allumer et éteindre les LED, que ce soit pour celles situées sur le connecteur RJ45 ou celles sur le devant de l'Evalbot.

3. moteur.s

Ce fichier contient les méthodes pour contrôler les moteurs, incluant l'activation des moteurs gauche et droit, ainsi que la gestion de leur direction et de leur vitesse. Il permet de configurer les ports nécessaires, d'activer les moteurs, et de gérer des actions comme avancer, reculer, ou inverser la rotation.

4. switch.s

Le fichier permet d'activer les switchs du système. Il configure les registres pour activer les résistances pull-up et la fonction numérique des switchs. Ce fichier est essentiel pour interagir avec les entrées utilisateurs via les switchs.

5. sysctl.s

Ce fichier est responsable de l'activation des clocks pour les différents composants utilisés dans le projet. Il configure la clock des ports D, E, et F nécessaires pour l'activation des composants comme les LED, les bumpers, et les switchs. Le fichier sysctl.s permet de garantir que toutes les parties du système fonctionnent correctement en synchronisant leur activation.

6. main.s

Le fichier principal main.s centralise toutes les actions en appelant les méthodes définies dans les autres fichiers. Il sert de point d'entrée pour l'exécution du programme et coordonne l'activation et l'interaction de tous les composants du système.

2.2 Explications des sections clés

SYSCTL.s

Le fichier est utilisé pour activer la clock des composants nécessaires. Dans notre cas, nous avons besoin des ports D, E et F, correspondant en hexadécimal à la valeur 0x38 (binaire : 0011 1000). La valeur est placée dans le registre R0. Chaque bit actif active un port GPIO :

- Bit 3 (valeur 0x08) : Active le port D.
- Bit 4 (valeur 0x10) : Active le port E.
- Bit 5 (valeur 0x20) : Active le port F.

D'après les indications du constructeur, la clock peut être modifiée à l'adresse 0x400FE108. Nous y stockons donc la valeur 0x38, correspondant aux ports GPIO requis. Ensuite, trois instructions NOP (No Operation) sont exécutées pour laisser le temps aux composants de s'activer, comme recommandé par le constructeur.

SWITCH.s

Le code initialise les deux boutons poussoirs connectés au port GPIO D, broches 6 et 7. Il active la résistance de pull-up (avec un état logique haut par défaut) et configure ces broches, prêtes à être utilisées comme entrées. Le fichier est conçu pour activer les registres GPIO_PUR et GPIO_DEN d'un switch. Le registre GPIO_PUR (Pull-Up Resistor) permet d'activer la résistance pull-up spécifiée dans le circuit électrique par le constructeur, visible dans la datasheet. Quant à GPIO_DEN (Digital Enable), il active la fonction numérique du switch. Pour le port D (dont l'adresse de base est 0x40007000), les adresses des registres PUR et DEN se situent respectivement à 0x510 et 0x51C après l'adresse de base. Dans notre cas, deux switchs situés sur les broches PD6 et PD7 (valeurs hexadécimales 0x40 et 0x80) doivent être activés. Nous définissons les constantes BROCHE6 et BROCHE7, et dans l'étiquette SWITCH_INIT, nous combinons leurs valeurs pour activer à la fois DEN et PUR pour ces deux switchs.

BUMPERS.s

Le fichier fonctionne de manière similaire à SWITCH.s, mais concerne les bumpers situés sur le port E. Les broches d'intérêt sont PE0 et PE1. Comme pour les switchs, les registres GPIO_PUR et GPIO_DEN sont activés pour permettre leur fonctionnement.

LED.s

Le fichier LED.s sert à activer les registres GPIO_DIR, GPIO_DR2R et GPIO_DEN pour les LED.

- GPIO_DIR configure la direction du signal (sortie dans ce cas).
- GPIO_DR2R règle l'intensité à 2 mA.
- GPIO_DEN active la fonction numérique.

Les LED sont situées sur le port F (adresse de base 0x40025000) :

- Les LED du connecteur RJ45 femelle sont sur les broches PF2 et PF3.
- Les LED frontales de l'Evalbot sont sur les broches PF4 et PF5.

L'étiquette LED_INIT initialise les registres DIR, DEN et DR2R pour ces broches. D'autres étiquettes, comme LED3_ON et LED3_OFF, contrôlent individuellement les LED. Par exemple :

- LED3_ON allume la LED en appliquant un masque (PIN2 << 2) à l'adresse de base.
- LED3_OFF éteint la LED en fixant la valeur à 0.

Les LED situées à l'arrière fonctionnent différemment : elles sont actives à l'état bas. Le code initialise efficacement les broches 2, 3, 4 et 5 du port GPIO F pour fonctionner comme sorties digitales avec les LED connectées. Grâce aux fonctions LEDx_ON et LEDx_OFF, il est possible de gérer facilement l'état des LED.

MOTEURS.s

Le fichier fourni avec le projet, contient des étiquettes pour gérer les moteurs droits et gauches, leur direction, leur vitesse, ainsi que des opérations comme avancer, reculer ou inverser la rotation.

Les moteurs utilisent les ports D et H de la clock, nécessitant l'activation des registres RCGC0 et RCGC2. Les GPIO requis sont DIR, DR2R, DEN, PCTL et AFSEL. Le mode PWM (Pulse Width Modulation) est également activé pour contrôler précisément la vitesse.

L'étiquette MOTEUR_INIT initialise les moteurs en activant les ports D et H pour les deux clocks mentionnées, et en configurant les registres nécessaires.

- Le moteur gauche est contrôlé via le port H.
- Le moteur droit est contrôlé via le port D.

Par exemple, pour avancer, une valeur 0 est écrite dans l'adresse appropriée, tandis que pour reculer, une valeur 2 est utilisée. L'inversion de direction est réalisée grâce à l'instruction EOR r0, r1, #GPIO_1.

Enfin, la vitesse est réglée par la variable VITESSE, initialisée à 0x152. Cela permet de gérer des actions spécifiques, comme un quart ou un demi-tour, en synchronisation avec des instructions WAIT.

MAIN.s

MAIN.s correspond au point d'entrée principal de notre code. C'est ici que tous les branchements inconditionnels sont réalisés, ainsi que l'appel des fonctions à travers des opérations BL.

Nous importons les broches (les pins) qui nous serons utiles pour la suite du code. Les broches PIN2, PIN3, PIN4 et PIN5 correspondent aux LED, BROCHE6 et BROCHE7 sont utilisées pour les switchs et enfin BROCHE0 et BROCHE1 sont associés aux bumpers. Ensuite, on définit les trois ports nécessaires pour lire la valeur des GPIO ainsi que pour écrire dans les ports de base (les ports D, E et F).

```
5  PIN2 EQU 0x04
6  PIN3 EQU 0x08
7  PIN4 EQU 0x10
8  PIN5 EQU 0x20
9
10 BROCHE6      EQU 0x40      ; bouton bumper 1
11 BROCHE7      EQU 0x80      ; bouton bumper 2
12
13 BROCHE0      EQU 0x01      ; bouton poussoir 1
14 BROCHE1      EQU 0x02      ; bouton poussoir 2
15
16 GPIO_PORTD_BASE EQU 0x40007000
17 GPIO_PORTE_BASE EQU 0x40024000
18 GPIO_PORTF_BASE EQU 0x40025000
19
```

```
24  IMPORT MOTEUR_INIT          ; initialise les moteurs (configure les pwms + GPIO)
25
26  IMPORT MOTEUR_DROIT_ON       ; activer le moteur droit
27  IMPORT MOTEUR_DROIT_OFF      ; déactiver le moteur droit
28  IMPORT MOTEUR_DROIT_AVANT    ; moteur droit tourne vers l'avant
29  IMPORT MOTEUR_DROIT_ARRIERE  ; moteur droit tourne vers l'arrière
30  IMPORT MOTEUR_DROIT_INVERSE   ; inverse le sens de rotation du moteur droit
31
32  IMPORT MOTEUR_GAUCHE_ON       ; activer le moteur gauche
33  IMPORT MOTEUR_GAUCHE_OFF      ; déactiver le moteur gauche
34  IMPORT MOTEUR_GAUCHE_AVANT    ; moteur gauche tourne vers l'avant
35  IMPORT MOTEUR_GAUCHE_ARRIERE  ; moteur gauche tourne vers l'arrière
36  IMPORT MOTEUR_GAUCHE_INVERSE   ; inverse le sens de rotation du moteur gauche
37
38  IMPORT LEDS_INIT
39  IMPORT LED2_ON
40  IMPORT LED2_OFF
41  IMPORT LED3_ON
42  IMPORT LED3_OFF
43  IMPORT LED4_ON
44  IMPORT LED4_OFF
45  IMPORT LED5_ON
46  IMPORT LED5_OFF
47  IMPORT SYSTCL_INIT
48  IMPORT SWITCH_INIT
49  IMPORT BUMPER_INIT
50  IMPORT MOTEUR_DOUBLE_AVANT
```

Ensuite, nous importons l'ensemble des étiquettes côté moteur qui seront nécessaires. Nous importons également les étiquettes que nous avons développées, par exemple, pour allumer et éteindre les LED individuellement, ainsi que pour initialiser les différents composants que nous allons utiliser.

Dans cette partie du code, nous faisons appel aux étiquettes d'initialisation des composants : bumpers, switchs et LED. Nous initialisons également la SYSTCL qui comme mentionné précédemment, configure l'horloge de l'Evalbot. Par ailleurs, nous éteignons les deux LED arrière (appelées "ON" mais qui, en réalité éteignent les LED puisqu'elles sont actives à l'état bas). Chaque épreuve de l'Evalbot est structurée en trois parties :

Step : Le robot tourne si nécessaire puis avance jusqu'à heurter un obstacle.

Tempo : Une phase de temporisation où le robot attend que l'utilisateur appuie sur le switch bas pour déclencher l'épreuve

Code de l'épreuve : Spécifique à chaque épreuve

Nous avons quatre épreuves que nous allons vous présenter dans cette partie.

main

```
BL SYSCTL_INIT
BL LEDS_INIT
BL SWITCH_INIT
BL BUMPER_INIT
BL MOTEUR_INIT
BL LED2_ON
BL LED3_ON
```

Nous allons présenter le code de la première épreuve : une question de type Vrai/Faux.

Dans un premier temps, nous activons les deux moteurs, puis nous faisons avancer les deux roues pour que le robot roule. Ensuite, nous initialisons les registres pour lire la valeur des bumpers. À l'aide d'un branchement inconditionnel, nous vérifions si l'un des deux bumpers est activé. Si c'est le cas, nous passons à la partie de temporisation associé à la question 1. Ce schéma sera répété avant chaque question.

```
; PREMIERE ETAPE: Le robot avance tout droit tant que l'un des deux bumpers n'est pas activé

step1
; Faire avancer le robot
BL MOTEUR_DROIT_ON
BL MOTEUR_GAUCHE_ON
BL MOTEUR_DOUBLE_AVANT

; Initialisation des bumpers
ldr r7, = GPIO_PORTE_BASE + (BROCHE1<<2)
ldr r9, [r7] ; bumper gauche
ldr r7, = GPIO_PORTE_BASE + (BROCHE0<<2)
ldr r8, [r7] ; bumper droit

; Si le bumper gauche est activé
cmp r9, #0
; on passe a la première question
BEQ question1_tempo

; Si le bumper droit est activé
cmp r8, #0
; on passe a la première question
BEQ question1_tempo

; sinon, on retourne a l'étape de base
B step1
```

Examinons maintenant la première temporisation :

```
; Etape qui temporise l'appui du bouton pour lancer la prochaine question
question1_tempo
; On éteint les moteurs
BL MOTEUR_DROIT_OFF
BL MOTEUR_GAUCHE_OFF
; on initialise le bouton switch bas
ldr r7, = GPIO_PORTD_BASE + (BROCHE7<<2)
ldr r6, [r7]
; si il est appuyé on passe a la question1
cmp r6, #0
beq question1
; sinon on reboucle sur l'étape actuelle
b question1_tempo
```

Tout d'abord, nous éteignons les deux moteurs afin que le robot soit à l'arrêt. Ensuite, nous initialisons le switch bas car c'est celui qui nous intéresse pour déclencher la première question. Enfin, nous vérifions si ce switch est appuyé. Si c'est le cas nous passons à la première question.

Enfin, nous arrivons à la question 1 :

Nous attendons un court instant avec WAIT_BLINK afin d'éviter que le switch appuyé pour déclencher la question ne soit capté par la question elle-même, ce qui pourrait être interprété comme une réponse. La question est de type Vrai/Faux et la réponse attendue est vraie. Nous initialisons donc les deux switchs et vérifions lequel est appuyé. En général, chaque question est accompagnée d'une méthode de gestion des erreurs. Dans ce cas, si une erreur est détectée, les deux LED s'allumeront et nous reviendrons à l'état initial en attendant la bonne réponse.

```

107 question1
108     BL WAIT_BLINK
109
110     ; Initialisation du switch haut
111     ldr r7, = GPIO_PORTD_BASE + (BROCHE6<<2)
112     ldr r6, [r7]
113     ldr r7, = GPIO_PORTD_BASE + (BROCHE7<<2)
114     ldr r5, [r7]
115     ; si le switch haut est appuyé
116     cmp r6, #0
117
118     ; on passe à l'étape suivante
119     BEQ step2
120     CMP r5, #0
121     BEQ error_question1
122     ; sinon, on repart au prompt de la question 1
123     B question1
124
125 error_question1
126     BL LED4_ON
127     BL LED5_ON
128     BL WAIT
129     BL LED4_OFF
130     BL LED5_OFF
131     B question1

```

Ensuite, il faut faire tourner le robot et le faire avancer jusqu'à l'épreuve2. Pour cette partie, c'est step2 qui s'en occupe.

```

step2
    ; On active les deux moteurs
    BL MOEUR_DROIT_ON
    BL MOEUR_GAUCHE_ON

    ; On effectue une rotation de 90deg
    BL MOEUR_DROIT_ARRIERE
    BL MOEUR_GAUCHE_AVANT
    BL WAIT

; Deuxième étape temporisée : pour pouvoir rebouclé, on crée cette sous étape qui ne fera qu'avancer
step2_bis
    ; On fait avancer le robot
    BL MOEUR_DROIT_AVANT
    BL MOEUR_GAUCHE_AVANT

    ; On initialise les deux bumpers gauche et droit
    ldr r7, = GPIO_PORTE_BASE + (BROCHE1<<2)
    ldr r9, [r7] ; bumper gauche
    ldr r7, = GPIO_PORTE_BASE + (BROCHE0<<2)
    ldr r8, [r7] ; bumper droit

    ; Si le bumper gauche est activé
    cmp r9, #0
    ; On peut passer à la question 2
    BEQ question2_tempo

```

Pour les prochaines épreuves, où le robot doit toujours tourner, nous suivrons un schéma similaire. L'étiquette step fera tourner le robot en activant les roues dans des sens opposés, puis un WAIT sera effectué, correspondant à un quart de tour pour le robot. Ensuite, le robot avancera indéfiniment à l'aide de step2_bis, jusqu'à ce que l'Evalbot heurte un obstacle.

L'épreuve 2 correspond au jeu du Simon. Nous devons donc commencer par afficher la trame du jeu de Simon.

```

question2

    ; On réalise la trame du simon
    BL LED4_ON
    BL WAIT
    BL LED4_OFF
    BL LED5_ON
    BL WAIT
    BL LED5_OFF
    BL WAIT_BLINK
    BL LED5_ON
    BL WAIT
    BL LED5_OFF
    BL LED4_ON
    BL WAIT
    BL LED4_OFF

    LTORG
    ; R1 sera la variable d'état, qui nous indiquera où en est le joueur
    mov r1, #0

```

Nous exécutons une série d'instructions BL et WAIT afin de réaliser la trame du jeu. Il est important de noter l'utilisation de WAIT_BLINK dans le cas où la prochaine étape consisterait à allumer à nouveau la même LED.

Enfin, il y a la question, donc dans notre cas la question 1 :

Cette partie analyse l'entrée de l'utilisateur. Les deux switchs sont initialisés. Le registre r1 déclaré avant cette étiquette simule une « state machine » : il s'incrémente à chaque bonne réponse et repasse à zéro dès qu'un mauvais switch est appuyé. check_btn_bas et check_btn_haut vérifient si le bouton correspondant a été appuyé, en fonction de la valeur de r1. Enfin, nous initialisons le bumper gauche. Si ce dernier est appuyé, il sert de reset afin que l'utilisateur puisse revoir la trame du simon s'il l'a oubliée.

```
question2_bis
; On initialise les deux switchs
ldr r7, = GPIO_PORTD_BASE + (BROCHE7<<2)
ldr r6, [r7]
ldr r7, = GPIO_PORTD_BASE + (BROCHE6<<2)
ldr r5, [r7]

; Si le switch du bas est appuyé
cmp r6, #0
; On vérifie si il devait bien être appuyé selon r1
beq check_btn_bas
; Si le switch du haut est appuyé
cmp r5, #0
; On vérifie si il devait bien être appuyé selon r1
beq check_btn_haut
; pour recommencer le simon
ldr r7, = GPIO PORTE_BASE + (BROCHE1<<2)
ldr r9, [r7] ; bumper gauche
cmp r9, #0
beq question2
; Si aucun n'a été activé, alors on reboucle sur l'étape de temporisation
B question2_bis
```

Ensute, il faut faire tourner le robot et le faire avancer jusqu'à l'épreuve2. Pour cette étape, c'est step2 qui s'en occupe.

```
check_btn_haut

; Si on est à l'étape 1
cmp r1, #0
; Faux, c'est le switch du bas qui est censé s'activé à l'étape 1
BEQ not_ok

; Si on est à l'étape 2
cmp r1, #1
; Vrai, c'est bien ce switch qui doit être activé
BEQ second_ok

; Si on est à l'étape 3
cmp r1, #2
; Vrai, c'est bien ce switch qui doit être activé
BEQ third_ok

; Si on est à l'étape 4
cmp r1, #3
; Faux, c'est le switch du bas qui est censé s'activé à l'étape finale
BEQ not_ok
```

Ce code correspond à l'étiquette check_btn_haut, qui sera appelé chaque fois que le bouton haut est activé. Il est important de noter que ce n'est pas ce bouton qu'il faut activer à l'initialisation, mais plutôt à la deuxième et troisième étapes. La dernière étape, quant à elle, doit être un bouton bas

Il en va de même pour check_btn_bas, qui fonctionne de manière inverse à check_btn_haut en fonction des valeurs de r1.

```
check_btn_bas

; Si on est à l'étape 1
cmp r1, #0
; Vrai, c'est bien ce switch qui doit être activé à la première étape
BEQ first_ok

; Si on est à l'étape 2
cmp r1, #1
; Faux, c'est le switch du haut qui est censé s'activé à l'étape 2
BEQ not_ok

; Si on est à l'étape 3
cmp r1, #2
; Faux, c'est le switch du haut qui est censé s'activé à l'étape 3
BEQ not_ok

; Si on est à l'étape 4
cmp r1, #3
; Vrai, c'est bien ce switch qui doit être activé à l'étape finale
BEQ fourth_ok
```

Les deux étiquettes, en fonction des valeurs de r1, retournent soit `not_ok` :

Si l'état retourné est `not_ok`, cela indiquera une erreur à l'utilisateur en allumant et en éteignant les deux LED. De plus, la valeur de r1 sera réinitialisée.

```
not_ok
; On allume les deux leds
BL LED4_ON
BL LED5_ON
BL WAIT
; On les éteins
BL LED4_OFF
BL LED5_OFF
; On reset la valeur de r1
mov r1, #0
; On repart a l'étape de base
b question2_bis
```

Soit elles peuvent retourner `first_ok`, `second_ok`, `third_ok` et `fourth_ok`. Voici l'analyse de l'un des états ok, les autres suivant la même trame :

```
; Cas première étape réussie
first_ok
    ; On allume la led a droite
    BL LED4_ON
    BL WAIT
    ; On éteint la led a droite
    BL LED4_OFF
    ; On passe a l'état suivant
    mov r1, #1
    ; On repart a l'étape de vérification
    b question2_bis
```

On allume la LED et on attend pendant un certain temps avec un wait, pour éviter que la détection du switch ne soit considérée comme une double activation. Ensuite on éteint la LED, on incrémente le registre d'état r1, et enfin on revient à la lecture de l'entrée de l'utilisateur.

Pour l'épreuve 3, nous effectuons un step, ce qui permet au robot de tourner et d'avancer jusqu'à ce qu'il détecte un obstacle. Ensuite, nous attendons à nouveau que l'utilisateur appuie sur un bouton, en nous basant sur les codes mentionnés précédemment. Enfin, nous passons au code spécifique de l'épreuve 3.

```
; On setup les switchs ainsi que les bumpers
ldr r7, = GPIO_PORTD_BASE + (BROCHE6<<2)
ldr r11, [r7] ; switch haut
ldr r7, = GPIO_PORTD_BASE + (BROCHE7<<2)
ldr r10, [r7] ; switch bas
ldr r7, = GPIO_PORTE_BASE + (BROCHE1<<2)
ldr r9, [r7] ; bumper gauche
ldr r7, = GPIO_PORTE_BASE + (BROCHE0<<2)
ldr r8, [r7] ; bumper droit

; Si le switch haut est appuyé
cmp r11, #0
; On allume la led a gauche
BEQ led5_switch_2
; Si le switch bas est activé
cmp r10, #0
; On allume la led a droite
BEQ led4_switch_2
; Si le bumper gauche est activé
cmp r9, #0
; On allume la led rj45 gauche
BEQ led2_switch_2
; Si le bumper droit est activé
cmp r8, #0
; On allume la led rj45 droite
BEQ led3_switch_2
; Si aucun des cas n'est activé, alors on repart a l'étape de base
B question3
```

Nous éteignons d'abord les moteurs et attendons dans la phase de temporisation. Ensuite, nous initialisons les deux switchs ainsi que les deux bumpers. Cette troisième épreuve consiste en un calcul binaire que l'utilisateur doit effectuer et pour lequel il doit renseigné la réponse. La réponse correcte est 1011. L'utilisateur doit allumer les LED dans l'ordre suivant : la LED en haut à gauche (correspondant au bit de poids fort), puis la LED en bas à droite et enfin la LED en haut à droite, dans l'ordre qu'il souhaite. Ensuite, nous réalisons une suite de branchements inconditionnels afin d'allumer les LED en fonction du switch et/ou bumper appuyé.

Analysons l'un des cas, les autres suivent la même trame :

```

led4_switch_2
    ldr r7, = GPIO_PORTF_BASE + (PIN4<<2)
    ldr r6, [r7]
    cmp r6, #0
    BEQ led4_on_2
    BL LED4_OFF
    B check_question3

led4_on_2
    BL LED4_ON
    B check_question3

```

Nous allons lire l'état de la LED. Si elle est éteinte, nous l'allumons ; si elle est déjà allumée, nous l'éteignons. Il est important de noter que toutes les issues renvoient à check_question3.

Ou bien, elles peuvent retourner first_ok, second_ok, third_ok et fourth_ok. Voici l'analyse de l'un des états « ok », les autres suivant la même trame :

```

check_question3
    ; Setup des leds (pour lire leurs états)
    ldr r7, = GPIO_PORTF_BASE + (PIN5<<2)
    ldr r6, [r7] ; led5
    ldr r7, = GPIO_PORTF_BASE + (PIN4<<2)
    ldr r5, [r7] ; led4
    ldr r7, = GPIO_PORTF_BASE + (PIN3<<2)
    ldr r4, [r7] ; led3
    ldr r7, = GPIO_PORTF_BASE + (PIN2<<2)
    ldr r3, [r7] ; led2

    ; Si la led droite est allumé
    cmp r6, #PIN5
    ; On vérifie si l'autre led est activée
    beq led5_allume_2
    ; Sinon, on repart a l'étape de base
    b question3

```

Nous effectuons une comparaison cmp avec #PIN4 pour vérifier si la LED est allumée. Si elle est éteinte, la valeur serait #0. Si la LED est allumée, nous procédons à la vérification des autres LED.

Dans ce cas, si la LED droite est allumée, nous pouvons passer à la vérification des LED côté port femelle RJ45.

```

led4_allume_2
    ; Si la led rj45 droite est allumée
    cmp r4, #0
    ; On vérifie si l'autre led est éteinte
    beq led3_eteint_2
    ; Sinon, on repart a l'étape de base
    b question3
; Cas où la led droite, gauche et droite rj45 sont toutes allumées,
led3_eteint_2
    ; Si la led rj45 gauche est éteinte
    cmp r3, #PIN2
    ; On peut passer a l'étape 4
    beq step4
    ; Sinon, on repart a l'étape de base
    b question3

```

Nous initialisons les 4 LED, puis nous vérifions successivement si elles sont allumées et éteintes dans l'ordre 1011. Ainsi, si la LED droite est allumée, nous pouvons vérifier si les autres suivent correctement la trame.

```

; Cas où la led droite est allumée, on veut vérifier si la led gauche est allumée
led5_allume_2
    ; Si la led a gauche est allumée
    cmp r5, #PIN4
    ; On vérifie si l'autre led est activée
    beq led4_allume_2
    ; Sinon, on repart a l'étape de base
    b question3

```

Le même procédé est appliqué, à l'exception que cette fois, si le dernier cas (la LED à gauche, côté arrière du robot) est éteint, cela signifie que la trame entrée est correcte. Dans ce cas, nous passons à step4, qui va poursuivre l'acheminement du robot.

La quatrième et dernière épreuve consiste à un test de vitesse, des LED vont s'allumer et l'objectif est d'appuyer au bon moment sur le switch correspondant à la LED.

```
;; Quatrième et dernière épreuve : Un test de rapidité
;; Principe : Appuyé au bon moment sur le switch de la led allumée
question4

; On setup les switchs ainsi que les bumpers
ldr r7, = GPIO_PORTD_BASE + (BROCHE6<<2)
ldr r11, [r7] ; switch haut
ldr r7, = GPIO_PORTD_BASE + (BROCHE7<<2)
ldr r10, [r7] ; switch bas
ldr r7, = GPIO_PORTE_BASE + (BROCHE1<<2)
ldr r9, [r7] ; bumper gauche
ldr r7, = GPIO_PORTE_BASE + (BROCHE0<<2)
ldr r8, [r7] ; bumper droit
```

Nous initialisons les registres de lecture des deux switchs et des deux bumpers.

Nous initialisons r1 à 0x55555 ce qui correspond à un temps assez court que l'utilisateur aura pour entrer sa réponse. Ensuite, nous éteignons toutes les LED. S'ensuit l'étiquette etape1, où nous allumons la LED à droite (LED4). Nous initialisons également le switch haut car LED4 correspond à PD6 dans notre scénario. Si l'utilisateur appuie sur ce switch, nous passons à l'étape de temporisation suivante. Sinon, nous décrémentons le timer. Si ce dernier atteint 0, nous revenons à l'étape1. Ces étapes sont répétées de manière similaire pour chaque LED, en utilisant des bumpers et des switchs associés, un total de sept fois.

```
etape1_tempo
    ldr r1, =0x55555
    BL LED4_OFF
    BL LED5_OFF
    BL LED2_ON
    BL LED3_ON

etape1
    BL LED5_OFF

    BL LED4_ON
    ldr r7, = GPIO_PORTD_BASE + (BROCHE6<<2)
    ldr r11, [r7] ; switch haut
    cmp r11, #0
    bne etape1_tempo

    subs r1, #1
    bne etape1
    B error_question4
```

On effectue la dernière rotation du robot pour le diriger ensuite vers le virus.

```
fin_bis
    BL LED4_OFF
    BL MOTEUR_DROIT_ON
    BL MOTEUR_GAUCHE_ON
    BL MOTEUR_DROIT_AVANT
    BL MOTEUR_GAUCHE_ARRIERE
    BL WAIT
fin
```

```
BL MOTEUR_GAUCHE_AVANT
BL MOTEUR_DROIT_AVANT
ldr r7, = GPIO_PORTE_BASE + (BROCHE1<<2)
ldr r9, [r7] ; bumper gauche
ldr r7, = GPIO_PORTE_BASE + (BROCHE0<<2)
ldr r8, [r7] ; bumper droit

; Si le bumper gauche est activé
cmp r9, #0
; On peut passer au blink
BEQ blink
; Si le bumper droit est activé
cmp r8, #0
; On peut passer au blink
BEQ blink
B fin
```

Le robot avance jusqu'à ce qu'il atteigne un obstacle. Si l'un des deux bumpers est activé, nous passons à l'état blink, qui correspond à l'état où le robot a terminé sa marche.

La quatrième et dernière épreuve consiste en un test de vitesse. Des LED vont s'allumer, et l'objectif est d'appuyer au bon moment sur le switch correspondant à la LED allumée.

```
blink
    BL MOTEUR_GAUCHE_ON
    BL MOTEUR_DROIT_ON
    BL MOTEUR_GAUCHE_ARRIERE
    BL MOTEUR_DROIT_AVANT
    BL LED4_ON
    BL WAIT_BLINK
    BL LED4_OFF
    BL LED5_ON
    BL WAIT_BLINK
    BL LED5_OFF
    BL LED2_OFF
    BL WAIT_BLINK
    BL LED2_ON
    BL LED3_OFF
    BL WAIT_BLINK
    BL LED3_ON
    LTORG
; Rebouclage a la fin
b blink
```

Le robot tourne sur lui-même et fait clignoter ses LED dans un sens anti-horaire.

Voici les deux fonctions utilitaires d'attentes qui ont aussi été développées :

```
; Partie qui permet de faire attendre le programme pendant un court instant, est utilisée pour l'attente entre deux clignotements
WAIT_BLINK ldr r2, =0xFFFF
wait_blink subs r2, #1
            bne wait_blink
            BX LR

; Partie qui permet de faire attendre le programme pendant un bref instant, est utilisée pour l'attente entre une rotation ou une avancée
WAIT    ldr r2, =0xAFFFFF
waitl   subs r2, #1
            bne waitl
            ;; retour à la suite du lien de branchement
            BX LR
```

Nous initialisons r2 à une valeur assez élevée pour que le processeur prenne du temps pour soustraire sa valeur. Si r2 atteint 0, nous pouvons sortir de l'étiquette avec bx lr.

3. Explications Techniques

3.1 Choix des GPIO

GPIO_PORT_F, GPIO_PORT_D, GPIO_PORT_E, GPIO_PORT_H, GPIO_I_PUR, GPIO_PWM, GPIO_O_DIR,
GPIO_O_DEN, GPIO_O_DR2R, GPIO_I_PUR, GPIO_SYSCTL_RCGC0

3.2 Justification des choix

Nous avons besoin du GPIO_PORT_F car il définit la base pour les LED. En effet, en réalisant un offset avec GPIO_DIR, DEN et DR2R, nous devons modifier les valeurs de ces adresses afin d'activer les LED. Nous connectons cela aux broches PIN2, PIN3, PIN4 et PIN5 afin de définir un 1 sur ces adresses. Ensuite, en réalisant un masquage de ce GPIO avec l'un des pins, nous pouvons accéder à la DATA de cette LED, pour savoir si elle est allumée ou éteinte, ou pour l'allumer ou l'éteindre.

Nous avons besoin du GPIO_PORT_D car il est utilisé dans un premier temps par le fichier moteurs.s. En effet, ce fichier, qui nous a été fourni, contrôle le moteur droit via ce port. De plus, dans nos scripts que nous avons rédigés, nous avons besoin de ce GPIO pour accéder ou modifier les switchs. Les broches 6 et 7 nous intéressent car elles gèrent respectivement le switch haut et le switch bas. Contrairement aux LED, il suffit de réaliser un offset avec GPIO_PUR et DEN.

Nous avons besoin du GPIO_PORT_E car il commande les bumpers. Les bumpers fonctionnent de la même manière que les switchs, et ainsi ce GPIO, dans notre programme, a la même utilité que le GPIO_PORT_D.

Le fichier moteur.s a besoin du GPIO_PORT_H car c'est ce port qui contrôle le moteur gauche. Ce dernier a également besoin du GPIO_PWM, qui génère un signal de pulse width modulation, qui simule une sortie analogique. Enfin, l'usage de GPIOPCTL permet de contrôler le port et GPIOAFSET est utilisé pour sélectionner la fonction alternative (l'alternate function select).

En ce qui concerne GPIO_O_DIR, il permet de spécifier si le composant que l'on paramètre est une entrée (0) ou une sortie (1). Dans notre cas, seules les LED sont des sorties. Ce dernier se trouve à un offset de 0x400 par rapport au gpio_port_base.

Quant à GPIO_O_DEN, il permet d'activer la fonction digitale et se trouve à un offset de 0x51C du gpio_port_base.

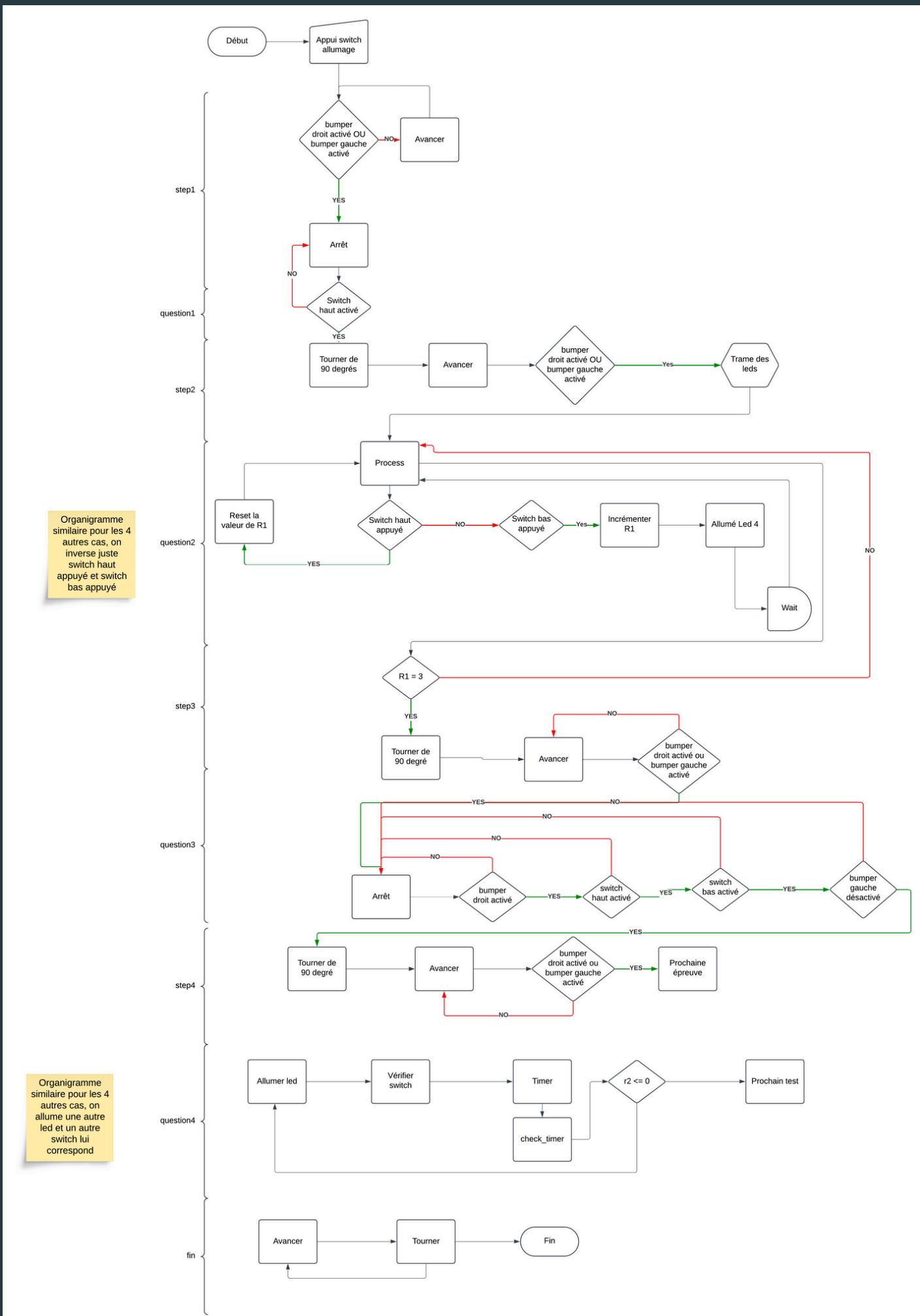
GPIO_O_DR2R permet de fixer l'intensité des LED à 2mA. Ce dernier se trouve à un offset de 0x500 par rapport au gpio_port_base.

Le registre GPIO_I_PUR est essentiel pour activer les résistances de pull-up sur les boutons poussoirs. Ce choix garantit que les broches utilisées comme entrées digitales maintiennent un état logique haut par défaut.

Enfin, SYSCTL_RCGC0 permet de modifier la valeur de l'horloge (clock) de l'Evalbot. En effet, il est nécessaire d'activer les composants avant de les utiliser. Dans notre cas, nous définissons la valeur à 0x38 car nous avons besoin du port D,F et E, les autres ports étant déjà gérés par le fichier moteur.s

Les ports choisis répondent aux spécifications électriques et logiques des composants connectés, notamment les LED, les boutons poussoirs, et les bumpers.

4. Déroulement des programmes



5. Conclusion

5.1 Résultats globaux

Nous sommes parvenus à développer un robot Evalbot qui comporte quatre épreuves, en utilisant les deux switchs, les deux bumpers, les deux moteurs ainsi que les quatre LED, le tout en suivant un scénario précis.

5.2 Problèmes rencontrés

- Nous n'avions pas la documentation pour les deux LED côté port femelle RJ45, et nous ne savions pas que ces LED étaient actives à l'état bas. Nous avons donc dû essayer plusieurs valeurs de pins afin d'identifier quelle LED correspondait à quel pin.
- Le fichier Moteurs.s était assez conséquent à comprendre. Nous avons dû réaliser des tests avec le code afin d'utiliser correctement les étiquettes et comprendre le rôle du gpio pwm.
- Pour réaliser un clignotement rapide entre deux allumages de la même LED, nous avons dû adapter le code wait fourni de base pour ajuster le temps d'attente.
- Lors des rotations, notre robot n'effectuait pas un quart de tour correctement et présentait toujours un décalage. Nous avons donc ajusté les valeurs de vitesse dans le fichier moteurs.s afin de trouver la vitesse idéale pour les rotations.
- Lorsque l'un des switchs était appuyé, il s'activait plusieurs fois simultanément en raison du signal, et le programme ayant une clock trop rapide, nous avons dû implémenter un délai d'attente pour permettre à l'utilisateur de retirer son doigt et éviter des appuis multiples.
- Lors de l'élaboration de la dernière épreuve, nous avons rencontré une erreur : « Directive LTORG may be in an executable position ». Cette erreur indique que la référence était trop éloignée de l'instruction où elle était utilisée. Nous avons donc ajouté l'instruction LTORG à la fin du programme et supprimé certaines lignes pour résoudre ce problème, bien que nous ne comprenions pas totalement son origine. Cela a limité notre capacité à créer d'autres épreuves.
- Dans l'épreuve 4, nous n'arrivions pas à faire en sorte que l'épreuve se réinitialise après un court délai si l'utilisateur était trop lent. Nous avons résolu ce problème en implémentant directement le code de l'étiquette wait.
- Enfin, nous n'avons pas réussi à séparer le code en plusieurs étiquettes comme initialement prévu. En effet, lorsque nous réalisions un jump vers une étiquette, nous ne parvenions pas à reprendre l'exécution normale du code après cette étiquette, comme cela se ferait avec un BX LR après un BL. Cela est dû au fait que, lorsqu'on effectue un BL sur une étiquette distante, on ne peut pas utiliser un BL dans cette étiquette, car il y a trop de profondeur. Le registre LR change deux fois et le registre PSR ne parvient plus à identifier son état initial.