Rapport projet Systèmes embarqués 2023 - 2024

Par Emilien MENDES et Téo ZWIEBEL
Enseignant encadrant :
Pierre-François BONNEFOI

1 Introduction

Les microbits sont des systèmes embarqués avec des capacités techniques bien inférieures à un ordinateur classique et ne peuvent donc pas toujours supporter un OS entier car cela ralentirait de manière significative la vitesse de calcul. C'est pour cela que seul une partie de l'OS est utilisée, ou bien sans OS dans l'architecture baremetal. Cela permet d'utiliser uniquement certaines fonctionnalités et évite l'usage d'un grand nombre de librairie mais dépend de l'architecture de chaque microbit.

2 Sélection d'émoji

Dans un premier temps, on crée un programme permettant de passer un émoji d'un microbit A à un microbit B. On se basera sur l'exemple **x20-radio** de Mike Spivey afin de pouvoir faire passer les données de l'un vers l'autre. Il y a deux phases à distinguer dans cette partie : l'envoi de données et sa réception.

2.1 Envoi de données

On utilisera le type de données déjà fourni : **IMAGE**. Ce type est un tableau d'entier mais il est inutile d'étudier ce type en détail pour les besoins de ce projet. On se contentera de définir la taille de l'image :

$$Image\_size = 10 \times sizeof(int) = 10 \times 4 = 40$$

L'image est sous le format d'une matrice de 5 x 5 avec une valeur à 1 si le pixel doit être allumé et 0 sinon. Pour le besoin du projet, on utilisera une liste d'émoji, que l'on passera en variable globale dans une liste. Afin de pouvoir réaliser l'envoi en continue, on ajoute une tâche associé à la fonction d'envoi, permettant au programme principal de ne pas rester sur celle-ci indéfiniment. La tâche d'envoi va donc lire les différentes valeurs d'entrée du microbit sur les boutons A et B .
Afin d'éviter le rebond, on ne se contentera pas de lire une seul fois les valeurs mais d'attendre un délai assez long pour que l'entrée soit correct, mais suffisamment courte pour que l'utilisateur ne puisse pas s'en rendre compte. On choisira ici un temps de **100** ms. Dans le programme principal, une des tâches est utilisé pour la gestion du timer, qu'on utilisera pour le rebond.
À chaque tour de boucle, on vérifiera si l'utilisateur a appuyé sur au moins un des boutons et on sauvegardera l'état des boutons A et B. Après le délai qui a été fixé, on vérifie de nouveau l'état des boutons et on effectue les actions en conséquences.L'image envoyé fait partie de la liste définie précédemment, on utilisera donc un index pour se déplacer dans cette liste. Ensuite,on effectue une vérification des trois cas dans l'ordre suivant :

1. Le bouton A et B sont pressés en même temps. Dans ce cas on envoie l'image correspondant dans notre liste.
2. Seul le bouton A est appuyé, on diminue l'index de notre liste d'image s'il est supérieur à 0.
3. Seul le bouton B est appuyé, on augmente l'index de notre liste d'image si ce n'est pas le dernier élément de la liste.

Une fois toute ces vérifications faites, on affiche l'image correspondante et on rajoute un nouveau delay pour éviter de changer trop rapidement les valeurs et donc les images.
L'envoi des données est fait par la fonction **radio_send** défini dans *microbian.h*.

2.2 Réception de données

La réception des données se fait grâce à une interruption. En effet, dans le code de la fonction *radio_receive*, une interruption est envoyé et elle est de type RECEIVE, type donné pour cette réception. Pour la fonction *radio_send*, le même principe est utilisé.
La tâche *receiver_task* va donc boucler tant qu'il ne reçoit pas d'interruption de la part de radio. L'entier **n** de la fonction de réception, permet de stocker la taille du paquet reçu par radio. On a définit précédemment la taille de notre image envoyé. Lorsque le microbit reçoit un signal radio avec un paquet de la taille de l'image on l'affiche avec la fonction *display_show*.

3 Dessin

Pour cette partie, plutôt que de passer par une liste d 'images, on va permettre à l'utilisateur de faire sa propre image et de l'envoyer à la manière de la partie précédente.

3.1 Principe

Pour représenter la création du dessin, on se servira d'une led qui clignote. Elle indique l'emplacement sur lequel l'utilisateur peut agir. S'il presse le bouton pendant une certaine durée, la led s'allume définitivement (ou bien s'éteint si elle était allumé précédemment).
En faisant un appui court on se déplace verticalement ou horizontalement en fonction du bouton appuyé. Si on appuie sur les deux boutons, on envoie l'image à l'autre microbit. Par rapport à la version précédente , il y aura trois tâches

1. Sender_task : Gère les appuis de bouton et la création de l'image
2. Receiver_task : Gère la réception de l'image reçu par radio.
3. Led_task : Gère la partie du clignotement et allumage/extinction de la led pour l'image.

3.2 Tache de la led

Pour structurer la led, on la symbolisera par une structure de données curseur. Cela comprend sa position, l'état (allumé ou éteinte) et si la led a déjà été utilisé. On définit aussi une image globale pouvant être modifié par toutes les tâches. Le curseur sera également globale.
Pour faire clignoter la led, on ajoute une fonction **image_unset** . Cette fonction copie celui de **image_set** déjà défini à la seul différence de l'opération utilisé sur le pixel. Comme définit dans *hardware.h*, on utilise un ET pour l'allumage et un OU pour l'extinction. On alternera les deux phases toutes les 100 ms, mais on prendra soin de stocker l'état initial de la led avant clignotement.
Dans le cas où on change de position on remet l'état de la led à celui stocker.Pour étudier l'état de celle-ci on crée une nouvelle fonction **state_pixel**, qui retourne la même valeur que pour les fonctions précédentes avec l'opération ET avec le registre shifté de n bits.

3.3 Envoi du dessin

Les appuis de bouton sont plus complexes à gérer dans cette partie, car il faut différencier les appuis **courts** des appuis **longs**. Pour éviter les rebonds, on laisse un délai et on regarde de nouveau les boutons appuyés. Pour chacun des trois cas possibles, on vérifie si le ou les boutons sont toujours enfoncés. A la fin, on obtient le temps d'enfoncement des boutons. Ainsi on a :

1. Appuis cours : Moins d'1 seconde
2. Appuis longs : Plus d'1 seconde

Une fois le bouton relâche, on modifie la position du curseur en X ou en Y en fonction du bouton et on active/désactive dans le cas d'un appui long. Dans le cas où on appuie sur les deux boutons, on envoie l'image puis on se contente d'attendre que les deux soient relâchés.

3.4 Réception du dessin

La réception du dessin se fait de la même manière que pour la partie précédent à la différence que l'on modifie l'attribut pour dire que le bouton n'est pas pressé. La variable étant globale,cela permet d'éviter d'utiliser les interruptions sur le système.

3.5 Utilisation

Pour exécuter le programme, il faudra tout d'abord recompiler avec **make** le dossier microbian, car deux des fonctions ne sont pas présentes dans la version de Mike Spivey. Ensuite, on compile la version Draw et on ajoute le fichier .hex dans le microbit.

4 Améliorations

Un des objectifs de ce projet était de réussir à émettre un signal sonore lors de la réception du message. Les différents exemple de Mike Spivey n'incluant pas de documentation sur la gestion de son, nous avons tenté d'utiliser la documentation sur le microbit. On remarque que les pins 0 et 1 sont utilisés pour avoir du son.
Dans le fichier *hardware.h*, il y a une référence aux pin, également utilisé dans **x31-adc**, mais cet exemple ne produit pas de son. Malgré nos recherches, nous n'avons pas réussi à faire fonctionner la partie sonore de notre microbit.

Pin	Type	Function
0	Touch	Pad 0
1	Touch	Pad 1
2	Touch	Pad 2
3	Analog	Column 1
4	Analog	Column 2
5	Digital	Button A
6	Digital	Row 2
7	Digital	Row 1
8	Digital	
9	Digital	Row 3
10	Analog	Column 3
11	Digital	Button B
12	Digital	
13	Digital	SPI MOSI
14	Digital	SPI MISO
15	Digital	SPI SCK
16	Digital	
19	Digital	I2C SCL
20	Digital	I2C SDA

FIGURE 1 – Liste des pins du microbit V2

5 Conclusion

Pendant ce projet, nous avons pu appréhender un système bare-metal avec toutes les contraintes qui lui sont liés et les éléments fournis lui permettant d'exécuter des programmes. Cela permet de s'affranchir de l'OS mais nécessite un développement d'interactions spécialisés sans aide extérieure.