Faculté des Sciences & Techniques — Université de Limoges

Projet de Systèmes Embarqués

Anaïs Espicier & Corentin Clerc

Premier Firmware

Pour ce premier firmware, nous pouvons sélectionner un emoji et l'envoyer à l'autre micro-bit.

Pour cela, les deux micro-bits disposent des mêmes emojis :

Un compteur global détermine lequel est affiché.

Nous avons 3 tâches pour gérer les différentes fonctionnalités du firmware:

- une tâche Main, responsable de l'affichage de l'image sur le microbit
- une tâche Choice, permettant de modifier le compteur (et donc l'emoji affiché) à l'aide des boutons du micro-bit
- une Receiver, permettant de modifier le compteur (et donc d'afficher un emoji) à partir d'une valeur reçue par radio

Affichage

L'affichage est décidé en fonction du compteur global. Il est réalisé dans une tâche spécifique, pour s'assurer que l'on a pas de collision pour la décision de quel emoji afficher.

Choix de l'emoji

Le choix de l'emoji se fait avec les boutons A et B. L'état de ces boutons est accessible à l'aide de `gpio_in(BUTTON_A)` et `gpio_in(BUTTON_B)` :

- 1 si le bouton est en haut (relâché)
- 0 si le bouton est en bas (appuyé).

Il nous est possible de vérifier qu'un bouton vient de changer d'état en conservant celui-ci en fin de boucle dans une variable à part, et de le comparer à l'état lu plus tôt.

Afin de gérer nos boutons, nous avons choisi de modifier notre compteur uniquement lors de la release, c'est-à-dire sur l'itération de boucle où l'état précédent vaut 0 et l'actuel vaut 1.

On augmente ou on réduit le compteur selon le bouton pressé, sans oublier de veiller à ne pas le laisser sortir de l'intervalle [0 ; nb d'émojis].

Réception du message

A la réception d'un message par radio sur le micro-bit, ce dernier est stocké dans un buffer et un 'beep' se fait entendre. Pour cela, nous avons déclaré notre speaker comme périphérique broché sur le port (0,0) en indiquant `#define SPEAKER_PIN DEVPIN(0, 0)` puis nous le déclenchons pendant 0.1 sec grâce à un `time_delay(100)` entre l'allumage `gpio_out(SPEAKER_PIN, 1)` et l'extinction `gpio_out(SPEAKER_PIN, 0)` . Ces opérations ont été réalisées grâce à l'aimable assistance de Pierre Metout, étudiant également en M1 cryptis info.
Ensuite, la valeur du compteur est actualisée avec le message reçu dans le buffer, que nous convertissons de bytes en int en faisant `- '0'`.

Priorités

L'ordre de priorité est définit de la façon suivante:

```
start("Receiver", emoji_received, 0, STACK);
start("Choice", emoji_choice, 1, STACK);
start("Main", main_task, 2, STACK);
```

Cet agencement des priorité permet d'éviter les collisions entre les tâches, en assurant que l'affichage soit toujours exécuté après que le `receiver` et `choice` aient décidé de la valeur de compteur, en donnant la priorité à l'utilisateur local.

Second Firmware

Pour ce second firmware, il s'agit de dessiner une image puis de l'envoyer par radio à l'autre micro-bit.

Gestion de l'image

Pour ce faire, il nous faut utiliser le type image. Ce type étant un tableau d'entiers de taille 10 sensé représenter une image de 25 pixels, manipuler directement cette image serait un casse-tête, aussi nous avons créé une méthode pour remplir une variable de ce type image à partir d'une matrice d'entiers. Il s'agit de la méthode suivante:

```
void apply_to_image(unsigned int drawing[5][5]){
      for(int i = 0; i < 5; i++){
      final_image[i*2] = BIT(getRowX(i)) | (!drawing[i][0]<<28) | (!
drawing[i][1]<<11) |
            (!drawing[i][2]<<31) | (!drawing[i][4]<<30);
      final_image[i*2 + 1] =  (!drawing[i][3]<<5);
      }
}
```

`getRow` est une méthode permettant d'obtenir la constante ROW associée à la pin de l'affichage du micro-bit à partir d'un entier.

Cette méthode sera appelée dans la tâche principale, avec comme paramètre d'entré `displayed_image`: une matrice d'entiers 5x5. Cette matrice sera notre représentation de ce que l'on affiche à l'écran.

Tâche principale

La tâche principale aura pour seule responsabilité d'afficher cette matrice.

```
void main_task(int dummy){
      while(1){
      // displayed_image[4][4] = displaying_recieved;
      apply_to_image(displayed_image);
      display_show(final_image);
      }
}
```

Le curseur

Pour gérer le curseur, nous créons une tâche à part. Le mettre dans la tâche main aurait introduit des delays au moment de l'affichage, rendant l'application peu réactive.

```
void blink_task(int dummy){
      while(1){
      if(displaying_recieved == 0){
                  // Le blink du curseur si on est pas en train de
      montrer un dessin reçu
                  displayed_image[cursor_x][cursor_y] = 0;
                  timer_delay(200);
                  displayed_image[cursor_x][cursor_y] = 1;
                  timer_delay(200);
          }
          timer_delay(100);
      }
}
```

On peut juste modifier le contenu de notre tableau `displayed_image` à la position du curseur.

Le dessin

Pour le dessin en lui-même, la partie la plus imposante est la gestion des inputs.
Nous avons pris le parti de ne prendre en compte que le up des boutons. Un flag nous permet de gérer cela : on le place à 1 lorsqu'il y a un changement d'état du bouton vers l'état haut, et on le "consomme" (remet à 0) une fois utilisé.

La gestion des pressions longues se fait simplement en gardant le moment où l'on presse le bouton, et en faisant la différence avec le moment actuel dans chaque boucle suivante. Si cette différence est plus

grande que le temps prévu pour une pression longue au moment de relâcher, on compte l'input comme pression longue. Sinon, on la compte comme simple input.

Pour gérer le dessin, nous avons une seconde matrice d'entiers 5x5. Ainsi, cela permet de garder le dessin séparément de ce qui est concrètement affiché, ce qui permet non seulement de modifier ce qui est affiché pour le curseur sans empiéter sur le dessin, mais aussi de garder en mémoire ce que dessinait l'utilisateur lorsqu'il reçoit un autre dessin.

Envoi d'un dessin

Lorsque l'utilisateur relâche les deux boutons après les avoir maintenus un certain temps, il envoie le dessin en cours au second micro-bit. L'envoi radio étant sous forme de byte, nous faisons la conversion de notre matrice d'entier en un tableau de char pour ce faire, avant d'envoyer.

```
void send_drawing(){
	char sending[25];
	for(int x = 0; x < 5; x++){
		for(int y = 0; y < 5; y++){
			sending[x*5 + y] = drawing[x][y] + '0';
		}
	}
	radio_send(sending, 25);
}
```

Réception d'un dessin

Le dessin reçu étant lui aussi en format byte, il faut également le convertir. On gardera le résultat de cette conversion dans une variable locale, qui sera celle que l'on copiera sur le display (avec la méthode `copy_image`).

```
void drawing_received(int dummy)
{
	byte buf[RADIO_PACKET];
	int n;
	while (1) {
	n = radio_receive(buf);
	if(n == 25){
			// BEEP !
			gpio_out(SPEAKER_PIN, 1); // Allumer
			timer_delay(100); // 0.1 sec de beep
			gpio_out(SPEAKER_PIN, 0); // Eteindre

			unsigned int recv_drawing[5][5] ={
				{0,0,0,0,0},
				{0,0,0,0,0},
				{0,0,0,0,0},
				{0,0,0,0,0},
				{0,0,0,0,0}
```

```
                    };
                    for(int x = 0; x < 5; x++){
                              for(int y = 0; y < 5; y++){
                              recv_drawing[x][y] = buf[x * 5 + y] -'0';
                              }
                    }
                    copy_image(recv_drawing);
                    displaying_recieved = 1;
          }
          timer_delay(100);
     }
}
```

La variable `displaying_recieved` nous permet de ne pas afficher le curseur lorsque l'on affiche une image reçue. Cette variable est remise à zéro lorsqu'un bouton est pressé.
Le beep est géré de la même façon que dans le premier firmware.