08/12/2021

Compte rendu Activitée 4

SAE – Concevoir la partie GEII système

RICCI Matyas H
IUT GEII TOULON

Dans ce compte rendu, le but sera d'expliquer dans l'intégralité le fonctionnement du programme de test, ainsi que toutes les fonctions et opérations qui y sont liés.

Dans la page 2 se trouvent les explication sur les instructions #include et #define.
Dans la page 3 se trouvent les explication sur les définitions de fonctions en C, sur l'attribution des variables globales utilisées dans notre programmes et des explications relatives concernant leurs types et un résumé de notre fonction setup et son utilisation générale en Arduino.
Dans la page 4 se trouvent les explications sur le moniteur de série et sur les modes des broches à l'aide de pinMode.
Dans la page 5 se trouvent les explications sur la fonction loop() et sur l'initialisation du VEML7700 et de notre récepteur infrarouge.
Dans la page 6 commencent les explications des différentes fonctions de notre programme, en commençant par la fonction testLeds().
Dans la page 7 se trouvent les explications de la fonction testBoutons et testAnalogique().
Dans la page 8 se trouvent les explications de la fonction testPwm()
Dans la page 9 se trouvent les explications de la fonction testVeml() et de la fonction testIr().

Notre programme de test expliqué ci-dessous permet de vérifier le bon fonctionnement de tous les composants de la carte. Pour rappel, notre carte a pour objectif final de pouvoir contrôler une lampe LED Ikea de manière autonome grâce à des capteurs de luminosité, et pourra donc ajuster la luminosité d'un environnement de travail, ou via une IHM qui sera dans notre cas deux boutons poussoirs et une télécommande infrarouge. Cette carte utilisera des modulations de largeur d'impulsion (PWM) afin de régler la luminosité émise par notre lampe IKEA.

1. #include

L'instruction #include est utilisée pour insérer des Bibliothèques externes dans notre programme. Ces instructions sont placées en début de programmes.

```
10
11 #include <IRremote.h>
12 #include <Wire.h>
13 #include "Adafruit_VEML7700.h"
14
```

Dans notre cas, on utilise la bibliothèque IrRemote.h, qui permet de recevoir et décoder du code infrarouge.
On utilise aussi la bibliothèque Wire.h qui permet de communiquer en I2C avec notre capteur de luminosité ambiante VEML7700.
Et enfin, on utilise aussi la bibliothèque Adafruit_VEML7700.h qui permet d'utiliser notre capteur de luminosité VEML7700.

2. #define

```
15 //------------- Définition des différentes broches utilisées de l'arduino Nano, ainsi que leur composant associé.
16
17 #define TX 1                    // Pin TX
18 #define RX 0                    // Pin RX
19 #define BP2 2                   // Bouton poussoir 2
20 #define PWM2 3                  //
21 #define BP1 4                   // Bouton poussoir 1
22 #define LED_R 5                 // Led Rouge de la led RGB
23 #define LED_G 6                 // Led verte de la led RGB
24 #define DOUT 7                  // Permet de contrôler un bandeau de led qui serais connecté à la carte
25 #define IR 8                    // Pin du récepteur infrarouge de la carte
26 #define PWM1 9                  // Pin contrôlant la grille du mosfet associé à la lampe.
27 #define SPI_SS 9                // Pin permettant une liaison SPI
28 #define LED_B 10                // LED Bleue de la led RGB
29 #define TEMP A0                 // Entrée de température via le bus I2C
30 #define A1 A1                   // Pin d'entrée associé à notre capteur de lumière ambiante VEML7700
31 #define LED2 A2                 // Pin permettant de contrôler la Led2
32 #define LED1 A3                 // Pin permettant de contrôler la Led1
33 #define SDA A4                  //
34 #define SCL A5                  //
35 #define POT A6                  // Pin d'entrée du potentiomètre de la acrte
36 #define LIGHT1 A7               // Pin d'entrée associé à notre photorésistance
37
```

L'instruction #define est un élément du langage C qui permet au programmeur de donner un nom à une constante avant que le programme soit compilé. Les constantes ainsi définies dans le langage Arduino ne prennent aucune place supplémentaire en mémoire dans le microcontrôleur. Le compilateur remplacera les références à ces constantes par la valeur définie au moment de la compilation. Cela nous permet de garder un programme lisible et compréhensible tout en optimisant la place prise par ce dernier une fois compilé.

Dans le premier paramètre, on définit le nom de notre constante et en deuxième paramètre, la valeur associée à ce dernier.
De manière générale, toutes les instructions précédées d'un # sont des instructions destinées et interprétées par le compilateur et ne prennent pas de place dans le programme compilé.

3.Définitions des fonctions

Après avoir inséré les bibliothèques et définit les constantes qui seront utilisés dans le programme, vient la déclaration des différentes fonctions qui seront utilisés dans le programme.

```
41 //------------Déclaration des sous-programmes (Prototypes)
42   void testLeds();
43   void testBoutons();
44   void testAnalogique();
45   void testPwm();
46   void testVEML();
47   void testIr();
```

On définit les fonctions avec un type « void » car les fonctions ne vont pas retourner de valeurs.
Ici, on définit les prototypes des fonctions permettant de tester les leds, les boutons, les entrées analogiques, les sorties PWM, le capteur de luminosité ambiante VEML ainsi que la fonction de test du récepteur infrarouge.

4.Variables globales

```
50 //-------------Déclaration des variables globales (Utilisées dans tout le programme)
51   unsigned long currentMillis1,currentMillis2,previousMillis1 = 0,previousMillis2 = 0;
52   int k=0;
53   Adafruit_VEML7700 veml = Adafruit_VEML7700();
```

Après avoir défini nos prototypes de fonction, on définit nos variables globales de notre programme.
Les deux variables currentMilis1 et currentMillis2, ainsi que previousMillis1 et previousMilis2 permettrons l'exécution des fonctions avec un délai définit entre deux exécutions sans utiliser la fonction delay() qui bloquerais l'exécution du programme entier durant le temps donné en argument.

Ces deux variables sont déclarées avec le type « unsignned long ».
Les variables de type long non signé sont des variables de taille élargie pour le stockage de nombre entier qui stocke les valeurs sur 4 octets (32 bits). A la différence des variables de type long standard, les variables de type long non signé ne peuvent pas stocker des nombres négatifs, la fourchette des valeurs qu'elles peuvent stocker s'étendant de 0 à 4 294 967 295.

La variable k est un entier (int) qui sera plus tard utilisée dans la fonction testPwm() et sera incrémentée.
Les variables de type int sont le type de base pour le stockage de nombres, et ces variables stockent une valeur sur 2 octets. Elles peuvent donc stocker des valeurs allant de - 32 768 à 32 767.

La variable Adafruit_VEML7700 est quant à elle nécessaire à l'exploitation de notre capteur de luminosité ambiante.

5. Fonction setup() :

Après avoir défini nos variables globales vient la fonction void setup(). Cette fonction est exécutée une seule fois uniquement au démarrage du programme. Elle est très utile car elle permet de définir les modes des broches (Entrée ou sortie) et permet aussi d'effectuer toutes les actions qui serais nécessaire une seule fois dans notre programme.

6. Moniteur de série

La première chose que notre fonction setup fait est de démarrer le moniteur de série avec un baud rate de 9600 à l'aide de la fonction Serial.begin(« baudrate »). Le baud rate définit le nombre de bits transmis par secondes. Sachant que plus cette valeur est élevée, plus la quantité d'information qu'il est possible de transmettre par seconde est grande, mais plus la communication est sujette à des interférences. Dans notre cas, 9600 est largement suffisant car on a très peu de données qui transitent par notre moniteur de série.
La sélection du débit permet d'adapter la vitesse de transmissions sur les broches 0 et 1 avec par exemple un composant qui nécessite un débit particulier et qui serais connectés à ces broches.
Après ça, on affiche dans le moniteur de série « Test carte SmartLight » à l'aide de la fonction Serial.println(« text »). Cette fonction permet d'afficher des informations (Du texte, des valeurs, …) en caractères ASCII dans le moniteur de série, et ajoute un caractère entré et un caractère retour à la ligne à la fin. Cette fonction a le même principe de fonctionnement que la fonction Serial.print(), mais avec le retour à la ligne et le caractère entrée en moins.

7. PinMode(broche, mode)

Cette fonction configure la broche spécifiée pour que cette dernière soit une broche d'entrée ou de sortie.
Elle est située dans notre fonction setup() étant donné qu'il est nécessaire de définir le mode des pins qu'une seule fois.

```
57 void setup() {
58   Serial.begin(9600);
59   Serial.println("Test carte SmartLight ");
60   pinMode(LED1, OUTPUT);
61   pinMode(LED2, OUTPUT);
62   pinMode(BP1, INPUT_PULLUP);
63   pinMode(BP2, INPUT_PULLUP);
64   pinMode(LIGHT1, INPUT);
65   pinMode(POT, INPUT);
66   pinMode(A1, INPUT);
67   pinMode(LED_R, OUTPUT);
68   pinMode(LED_G, OUTPUT);
69   pinMode(LED_B, OUTPUT);
70   pinMode(PWM1, OUTPUT);
```

Dans le premier paramètre, on donne le numéro de la broche de la carte Arduino dont le mode de fonctionnement (entrée ou sortie) doit être défini. Dans le deuxième paramètre, on définit le mode de fonctionnement de cette broche.

Il y a 3 possibilités de mode de fonctionnement pour les broches Arduino.
→ INPUT, indiquant à la broche de se comporter comme une entrée
→ OUTPUT, indiquant à la broche à se comporter comme une sortie
→ INPUT_PULLUP, indiquant à la broche à se comporter comme une entrée et à associer une résistance de tirage. Cela a pour effet que la broche sera à l'état haut par défaut en tant qu'entrée.

8. Initialisation du VEML7700 et du récepteur Infrarouge

```
77   veml.begin();
78   veml.setGain(VEML7700_GAIN_1);
79   veml.setIntegrationTime(VEML7700_IT_800MS);
80
81
82     // configurations/initialisations pour le
83   IrReceiver.begin(IR);
84
```

Les dernières choses que l'on effectue dans notre fonction Setup est d'initier nos capteurs. On initialise le capteur de luminosité ambiante avec la fonction veml.begin(), puis on indique les paramètres de fonctionnement de notre capteur. Dans notre cas, en premier on définit le gain de notre capteur à l'aide de la fonction veml.setGain(), puis on indique la durée d'une mesure de luminosité (temps d'exposition) à l'aide de la fonction veml.setIntegrationTime(). Le gain est paramétré à 1 et le temps d'une durée de luminosité à 800ms, 8x plus que la valeur par défaut. Le but étant d'obtenir une valeur plus précise, étant donné que de toute façon, une rapidité plus élevée ne nous servirait à rien.
Enfin vient l'initialisation du récepteur infrarouge à l'aide de la fonction IrReceiver.begin(). Cette fonction démarre l'écoute infrarouge du programme.

9. Fonction void loop()

```
89 void loop() {
90
91
92   currentMillis1 = millis();
93   if (currentMillis1 - previousMillis1 >= 10) {
94     previousMillis1 = currentMillis1;
95
96     testBoutons();
97     testPwm();
98   }
99
100
101
102  currentMillis2 = millis();
103  if (currentMillis2 - previousMillis2 >= 1000) {
104    previousMillis2 = currentMillis2;
105
106    testAnalogique();
107    testVEML();
108    testIr();
109  }
110 }
```

Cette fonction loop() est la fonction principale d'un programme Arduino. Elle est exécutée en boucle en permanence dès que la fonction setup() a fini son exécution.
Dans notre fonction loop(), on y a intégré un moyen d'exécuter toutes nos autres fonctions de notre programme avec un temps déterminé entre deux exécutions. Ceci est très pratique et nous permet par exemple de pouvoir incrémenter une valeur à chaque temps donné, afin de voir l'effet d'une variation de la modulation de largeur d'impulsion sur l'allumage d'une lampe. Si on ne contrôlait pas le temps entre deux exécutions de fonction, les changements s'effectueraient si vite que l'on ne verrait aucune différence.
Pour cela, on utilise la fonction Millis() qui renvoie le nombre de millisecondes depuis que la carte Arduino a commencé à exécuter le programme courant. Attention cependant à prendre en compte que ce nombre débordera et sera remis à 0 après une cinquantaine de jours.
La méthode ici est donc de garder dans une variable le temps actuel d'exécution, et de soustraire cette valeur avec la variable previousMillis, qui a en valeur le temps qui s'est écoulé entre l'exécution du programme et la dernière fois que cette variable a été touchée. Sachant que cette variable est modifiée uniquement lorsque la valeur de cette dernière et de currentMillis sont espacés d'une certaine valeur, on peut donc savoir si le temps

donné s'est écoulé ou non, et donc si on peut exécuter les fonctions associées ou non. Par exemple, pour currentMillis1 et previousMillis1, il faut que la différence des deux soit supérieur à 10, et donc que 10ms se soient écoulées pour que les fonctions testPwm() et testBoutons() soient exécutées. C'est le même fonctionnement pour currentMillis2 et previousMillis2, avec un délai de 1s pour l'exécution des fonctions testAnalogique(), testVEML() et testIr().
Il est nécessaire de faire comme ceci pour, on le rappel, ne pas bloquer l'exécution du programme entier avec la fonction delay() qui est plus simple à utiliser mais n'est pas appropriées ici.

- Définitions des fonctions

10. Fonction testLeds

Ensuite, on en vient à la définition de nos fonctions. La première est la fonction testLeds() et ressemble à ceci :

```
114  void testLeds(){
115    Serial.println("test des Leds");
116    for (int i = 0; i < 5; i++) {
117      digitalWrite(LED1, HIGH);
118      delay(500);
119      digitalWrite(LED1, LOW);
120      delay(500);
121
122      }
123    for (int j = 0; j < 5; j++) {
124      digitalWrite(LED2, HIGH);
125      delay(500);
126      digitalWrite(LED2, LOW);
127      delay(500);
128      }
129
130  }
```

Premièrement, on annonce rentrer dans la fonction testLeds() dans le moniteur de série grâce à la fonction Serial.println(). Ensuite, on utilise une boucle for associé à une variable i allant de 0 à 4. Dans cette boucle, on bascule notre broche associé à notre Led1 à l'état haut grâce à la fonction digitalWrite, ce qui a pour effet d'allumer la led, on attend 500ms et on bascule notre broche à nouveau mais à l'état bas cette fois-ci, ce qui éteint notre led. On répètre ce processus 5 fois pour la led1 et la led2.
La fonction digitalWrite() Met un niveau logique HIGH (HAUT en anglais) ou LOW (BAS en anglais) sur une broche numérique. Si la broche a été configurée en SORTIE avec l'instruction pinMode(), sa tension est mise à la valeur correspondante : 5V (ou 3.3V sur les cartes Arduino 3.3V) pour le niveau HAUT, 0V (masse) pour le niveau BAS. En paramètre on peut donc écrite soit HIGH soit LOW, soit 1 soit 0. Cette fonction est utilisable sur toutes les broches numérique et analogique sauf sur les broches A6 et A7 pour notre arduino nano.

11. Fonction testBoutons()

Ensuite viens notre fonction testBoutons(), qui comme son nom l'indique, permet de tester les boutonspoussoirs présent sur notre carte. Voici à quoi elle ressemble :

```
132  void testBoutons(){
133    if (!digitalRead(BP1)) {
134      digitalWrite(LED1, HIGH);
135    } else {
136      digitalWrite(LED1, LOW);
137      }
138    if (!digitalRead(BP2)) {
139      digitalWrite(LED2, HIGH);
140    } else {
141      digitalWrite(LED2, LOW);
142      }
143
144  }
```

On utilise la fonction digitalRead() afin de lire l'état de la broche BP1. Cette fonction renvoie soit HIGH (1) ou LOW (0) en fonction de la tension de la broche. On inverse cette entrée car notre broche BP1 est configuré en entrée INPUT_PULLUP, elle est donc relié par le microprocesseur de l'arduino a une résistance de tirage tirant la broche à l'état haut. Le bouton poussoir est de l'autre côté relié à la masse, cela signifie que lors d'un appui, le bouton poussoir fais la liaison avec la masse est la broche est alors imposée à l'état bas. Si on lit « 1 » sur la broche BP1, cela signifie que le bouton poussoir n'est pas appuié. On inverse donc cette entrée. Une fois que l'on a déterminé que le bouton poussoir étais appuyé, alors on allume la led associé tant que le bouton est enfoncé. Ensuite, au relâchement de ce dernier, on éteint la led associé. Ce fonctionnement est valable pour la led1 et la led2.

12. Fonction TestAnalogique() :

Dans cette fonction, le but sera de tester les entrées analogiques associées au potentiomètre, à la photodiode ainsi qu'à la photorésistance. Voici à quoi ressemble cette fonction :

```
146  void testAnalogique(){
147    Serial.println((String)"Photorésistance = " + analogRead(A1));
148
149
150
151    Serial.println((String)"Potentiomètre = " + analogRead(POT));
152    Serial.println((String)"Photodiode = " + analogRead(LIGHT1));
153    Serial.println("\n");
154
155  }
```

Ici, on utilisera la fonction analogRead().
La fonction analogRead prend comme argument la broche associé, et lit la valeur de la tension présente sur cette broche. L'arduini se charge ensuite de convertir une tension d'entrée entre 0 et 5v en une valeur numérique entière comprise entre 0 et 1023 grâce à un convertisseur analogique-numérique 10 bits. La précision est de 5 volt / 1024 = 4.9mV.
Dans cette fonction, on affiche dans le moniteur de série les valeurs converties pour les broches associées à la photorésistance, au potentiomètre et à la photodiode. Ici, j'ai utilisé (String) en début d'argument de la fonction println(), indiquant à la fonction de traiter toutes les données qui vont suivre comme étant des chaines de charactères. Cela permet ensuite de faire des additions entre des charactères, ce que sait faire l'arduino contrairement à une addition entre une chaîne de charactère et une valeur numérique. A la fin de la fonction, on affiche un retour à la ligne plus gagner en lisibilité dans le moniteur de série.

13. Fonction testPwm() :

Cette fonction a pour but de tester les sorties PWM de notre carte arduino et donc de faire varier la luminosité perçue des lumières. Voici à quoi ressemble notre fonction :

```
void testPwm(){
  if (k >= 1280) {


    analogWrite(LED_G, 0);
    analogWrite(LED_B, 0);
    analogWrite(LED_R, 0);
    analogWrite(PWM1, 0);
    k = 9999;

    } else k++;

    if (k < 256) {


      analogWrite(LED_R, k);
      analogWrite(PWM1, k);

    }
    else {

      if (k < 512) {
        analogWrite(LED_G, k - 256);

        analogWrite(PWM1, 0);
        analogWrite(LED_R, 0);
        }
```

```

      else
      {
       if (k < 768) {
         analogWrite(LED_G, 0);
         analogWrite(LED_B, k - 512);
         }
        else {

         if (k < 1024) {
           analogWrite(LED_R, k - 768);
           }
          else {
           if (k < 1280) {
             analogWrite(LED_G, k - 1024);

             }

           }
         }

       }

     }
}
```

Dans cette fonction, on va incrémenter la valeur de k entre 0 et 1279. Pour rappel, cette fonction est exécutée toutes les 10ms, et donc une incrémentation de k est effectuée toutes les 10ms.
On utilisera la fonction analogWrite(). Cette fonction permet de Générer une impulsion de largeur sur une broche de la carte Arduino (onde PWM - Pulse Width Modulation en anglais ou MLI - Modulation de Largeur d'Impulsion en français). Ceci peut être utilisé pour faire briller une LED avec une luminosité variable ou contrôler dans notre cas un Mosfet en tant que relais de puissance contrôlant notre lampe Ikea.
Après avoir appelé la fonction analogWrite, la broche associée générera une onde carrée avec un rapport cyclique spécifiée en pourcentage. La fréquence de cette onde PWM est approximativement de 490Hz.
Cette fonction prend en argument une valeur comprise entre 0 et 255, qui est ensuite donc convertie en pourcentage. A 0, le pourcentage à l'état haut du rapport cyclique est donc de 0, le signal est donc plat à l'état bas. A 255, le pourcentage à l'état haut du rapport cyclique est donc de 100%, le signal est donc plat à l'état haut.
Passons à l'explication de cette fonction…
Tout d'abord, au début de la fonction on regarde si on a déjà effectué un tour de cette fonction (si k est supérieur ou égal à 1280), pour éteindre toutes les lumières afin de préserver nos yeux lors du test et du développement de la carte et que les tests ont été effectués. 1280 car après 12,8 secondes, on a fini toutes les vérifications souhaitées. Si on n'a pas encore dépassé cette valeur, alors on incrémente k de 1. Ensuite, si k < 256, donc si 2,56s ne se sont pas écoulées, on allume progressivement la led Rouge de la led RGB grâce à notre PWM, et en parallèle on fait la même chose pour notre lampe Ikea. De cette façon, notre lampe et notre led rouge s'allument progressivement pendant 2,56s comme demandé dans l'activité 4. Ensuite, Si k est compris entre 256 inclut et 512 exclu, alors j'ai décidé d'éteindre ma lampe IKEA. Dans cet intervalle, on éteint aussi led rouge, et on effectue la même chose que pour la led rouge mais pour la led verte de la led RGB. On soustrait 256 à notre valeur de k car la fonction analogWrite() a besoin d'une valeur comprise entre 0 et 255 en argument. De cette manière, quand la condition est vérifiée (k < 512 et k >=256), notre valeur en argument de analogWrite() est bien comprise entre 0 et 255. C'est exactement le même principe pour k compris entre 512 inclus et 768 exclus, avec une soustraction cette fois-ci de k – 512 pour bien avoir une valeur comprise entre 0 et 255. On

éteint notre led verte et on allume progressivement la led bleue. Pour k compris entre 768 inclus et 1024 exclus, le but est d'afficher des nuances de couleurs et d'avoir les 3 leds de notre led RGB totalement allumée en même temps à la fin, afin d'afficher une couleur blanche. Pour cela, on n'éteint pas la led bleue, qui est donc à 100% de rapport cyclique, et on allume la led rouge progressivement avec un rapport cyclique de plus en plus élevé. En argument de la fonction analogWrite(), ici, on soustrait 768 de k pour toujours garder un argument compris entre 0 et 255. Et enfin, quand k est compris entre 1024 inclus et 1280 exclus, on allume progressivement la led verte avec en argument de la fonction analogWrite(), k – 1024. A la fin, on a donc la led RGB allumée en blanc.

14. Fonction TestVEML() :

Cette fonction est très courte, et utilise la bibliothèque précédemment importée « Adafruit_VEML7700 ». Voici la fonction :

```
211  void testVEML(){
212    Serial.print("Luminosité de rérérence : "); Serial.print(veml.readLux()); Serial.println("lux");
213  }
```

Cette fonction sert à afficher dans le moniteur de série la valeur de la luminosité ambiante, convertie en lux grâce à la bibliothèque associée. Pour cela, on utilise la fonction définie dans cette bibliothèque « veml.readLux() » qui renvoie donc la valeur en lux captée par le capteur de luminosité ambiante.

15. Fonction testIr() :

Cette fonction est la dernière fonction de notre programme de test de notre carte. Voici à quoi elle ressemble :

```
215  void testIr(){
216    if (IrReceiver.decode()) {
217      uint16_t command = IrReceiver.decodedIRData.command;
218      Serial.println((String)"Code touche télécommande :" + command);
219
220      if (command == 147) {
221        digitalWrite(LED1, HIGH);
222        } else if (command == 146) {
223          digitalWrite(LED2, HIGH);
224          } else if (command == 144) {
225            digitalWrite(LED1, LOW);
226            digitalWrite(LED2, LOW);
227            }
228       delay(500);
229      IrReceiver.resume();
230        }
231
232
233  }
```

Tout d'abord, on attend que IrReceiver.decode() ne soit pas à 0, auxquels cas cela signifie que le récepteur infrarouge a intercepté et décodé un signal infrarouge. Ensuite, on définit une variable commande, dans laquelle on va attribuer la valeur de la commande décodée à l'aide de la fonction IrReceiver.decodedIRData.command provenant de la Bibliothèque IRremote. Ensuite, dans cette fonction, on affiche la commande décodée. Puis, on compare la commande avec celles qu'on attend. Si la commande décodée a pour valeur 147, cela signifie que l'on a appuyé sur le bouton 1 de la télécommande. Dans ce cas, on allume la led1. Si la commande a pour valeur 146, cela signifie que l'on a appuyé sur la touche 2 de la télécommande infrarouge, dans ce cas, on allume la led2. Si la commande a pour valeur 144, cela signifie que l'on a appuyé sur le bouton ON/OFF de la télécommande. Dans ce cas, on éteint les deux leds. A la fin de la fonction, on utilise un délai de 500ms à l'aide de la fonction delay() pour éviter la répétion de l'affichage sur un appui long. Et enfin, à la fin de la fonction, on ré initialise le récepteur infrarouge qui est mis en pause à chaque commande reçue, et qui reste dans cet état tant que la fonction IrReceiver.resume() n'a pas été appelé.