Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Systèmes Embarqués

Corrections TP n°1

Programmation avec Arduino

Broches GPIO

1 – Récupérez le « *schematic* » de la carte de développement, « *devboard* », de Muse Lab sur
`https://github.com/wuxx/nanoESP32-C3/blob/master/schematic/nanoESP32C3-v1.0.pdf`

a. Trouvez la broche correspondant à :
- la LED RGB, de type WS2812 : *d'après le schematic, c'est la broche 8* ;
- le bouton "BOOT" : *broche 9* ;

b. D'après le « *schematic* », le bouton est-il :
- « *active low* », c-à-d quand on presse le bouton l'état logique est 0 ?
- « *active high* », c-à-d quand on presse le bouton l'état logique est 1 ?

D'après le schematic, le bouton connecte à la masse ou au « ground », il est donc « active low ».
Au niveau logique, on lira la valeur zéro lorsque le bouton est pressé et la valeur un quand il est relâché.

2 – a. Vous testerez le programme suivant :

```
const int buttonPin = 9;  // the number of the pushbutton pin
int buttonState = 0;  // variable for reading the pushbutton status

void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  if (buttonState == LOW) {
    Serial.println("L");
  }
}
```

Que fait-il ?
il n'arrête pas d'afficher 'L' même après avoir relâché le bouton.
Quel rapport avec la petite LED rouge présente sur la carte ?
elle indique une activité sur le port série
Si vous changez l'initialisation de la broche de `INPUT` à INPUT_PULLUP, que se passe-t-il ?
si on relâche le bouton l'affichage s'arrête.
Expliquez le comportement.
Le pullup ramène la broche au niveau haut alors que le bouton l'amène au niveau bas.

b. Modifiez le programme précédent pour qu'il n'affiche qu'**une seule fois** le message lorsque l'on appuie sur le bouton, c-à-d que si on maintient appuyé le bouton, un seul affichage se fait.

```
const int buttonPin = 9;  // the number of the pushbutton pin
int buttonState = 0;  // variable for reading the pushbutton status
int afficher_passage_a_low = false;

void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT_PULLUP);
  afficher_passage_a_low = false;
}

void loop() {
  buttonState = digitalRead(buttonPin);

  if ((buttonState == LOW) &&(afficher_passage_a_low == false )) {
    afficher_passage_a_low = true;
    Serial.println("L");
  }
  if (buttonState == HIGH)
  {
```

```
    afficher_passage_a_low = false;
  }
}
```

Est-ce que cela marche comme vous le voulez ?
Non, il peut en afficher plus d'un.
Pourquoi ?
À cause du rebond mécanique.
Est-ce que votre programme détecte un changement d'état sur la broche associée au bouton ?
Non, il lit au moment où le CPU le choisit.

c. Dans les exemples fournis par l'IDE, chargez « *File>Examples>02.Digital>Debounce* ».

Expliquez ce qu'il fait ?
Il introduit un certain délai avant de valider le changement d'état du bouton.

Qu'est-ce que le « *debounce* » d'un bouton ?
Un procédé qui permet d'annuler les effets de rebond mécanique du bouton : par exemple lorsque l'on appui sur le bouton, la pression mécanique peut faire un contact, puis le rebond libère temporarirement ce contact pour le rétablir ensuite définitivement.
Ce rebond est très court, inférieur à $10ms$*, ce qui plus rapide que la perception humaine mais suffisamment long pour simuler l'appui rapide et répété du bouton du point de vue du CPU.*

Modifiez le **programme précédent** pour intégrer le « *debounce* ».

```
const int buttonPin = 9;  // the number of the pushbutton pin
int buttonState = 0;  // variable for reading the pushbutton status
int lastButtonState = HIGH;
int lastDebounceTime = 0;
int lecture = HIGH;
int afficher_passage_a_low = false;

void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT_PULLUP);
  afficher_passage_a_low = false;
}

void loop() {
  lecture = digitalRead(buttonPin);

  if (lecture != lastButtonState)
  {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > 50)
  {
    if (lecture != buttonState)
    {
      buttonState = lecture;
      if(buttonState == LOW)
      {
        afficher_passage_a_low = true;
      }
    }
  }
  if (afficher_passage_a_low == true ) {
    afficher_passage_a_low = false;
    Serial.println("L");
  }
  lastButtonState = lecture;
}
```

IRQ

3 – Vous essaierez le programme suivant :

```
#include <Arduino.h>

struct Button {
    const uint8_t PIN;
    uint32_t numberKeyPresses;
    bool pressed;
};

Button button1 = {9, 0, false};

void ARDUINO_ISR_ATTR isr(void* arg) {
    Button* s = static_cast<Button*>(arg);
    s->numberKeyPresses += 1;
    s->pressed = true;
}

void setup() {
    Serial.begin(115200);
    pinMode(button1.PIN, INPUT_PULLUP);
    attachInterruptArg(digitalPinToInterrupt(button1.PIN), isr, &button1, RISING);
}

void loop() {
    if (button1.pressed) {
        Serial.printf("Button 1 has been pressed %u times\n", button1.numberKey
Presses);
        button1.pressed = false;
    }
}
```

a. Pourquoi met-on `RISING` et non pas `FALLING` ?
Parce que le bouton est «active low» : l'état passe de haut à bas à l'appui, puis de bas à haut lorsqu'on le relâche.
Ici, on détecte le moment où il est relâché.

b. Que se passe-t-il si vous déplacez l'affichage du compteur d'appui dans la fonction `isr` ?
Le programme peut planter si on appuie de manière rapide et répétée.
Pourquoi ?
Parce que l'interruption se déclenche et accède de nouveau au port série, avant que l'interruption précédente ait, elle-même fini d'y accéder, créant ainsi une corruption.

4 – a. Comment fonctionne une LED RGB WS2812 ?
`https://www.sdiplight.com/what-is-ws2812b-led-and-how-to-use-ws2812b-led/`
Elle dispose d'un micro-contôleur auquel on envoie l'information nécessaire à son état.
Vous installerez la bibliothèque permettant de piloter la LED WS2812 de Freenove.

```
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_COUNT  1
#define LEDS_PIN   8
#define CHANNEL      0

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHAN
NEL, TYPE_GRB);

void setup() {
  strip.begin();
  strip.setBrightness(20);
}

void loop() {
  for (int j = 0; j < 255; j += 2) {
    for (int i = 0; i < LEDS_COUNT; i++) {
      strip.setLedColorData(i, strip.Wheel((i * 256 / LEDS_COUNT + j) & 255));
    }
    strip.show();
    delay(10);
  }
}
```

b. Vous combinerez ce programme avec le précédent pour bloquer l'arc-en-ciel lorsque l'on appui sur un bouton avec un traitement par IRQ.

```
#include <Arduino.h>
#include "Freenove_WS2812_Lib_for_ESP32.h"

#define LEDS_PIN    8
#define CHANNEL     0

Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL,
TYPE_GRB);

struct Button {
    const uint8_t PIN;
    bool pressed;
};

Button button1 = {9, false};

void ARDUINO_ISR_ATTR isr(void* arg) {
    Button* s = static_cast<Button*>(arg);
    s->pressed = !digitalRead(s->PIN);
}

void setup() {
  Serial.begin(115200);
  pinMode(button1.PIN, INPUT_PULLUP);
  attachInterruptArg(digitalPinToInterrupt(button1.PIN), isr, &button1, CHANGE);
  strip.begin();
  strip.setBrightness(20);
}

void loop() {
  for (int i = 0; i < 255; i += 2) {
    while(button1.pressed) delay(10);
    strip.setLedColorData(0, strip.Wheel((i)));
    strip.show();
    delay(10);
      }
}
```

On utilise l'état `CHANGE`, pour déclencher l'interruption au moment de l'appui, mais aussi du relâchement du bouton.
On affecte la négation de la lecture du niveau sur la broche à l'état du bouton car il est « active low ».

Timers

5 – Vous essaierez le code suivant :

```
#include "esp_system.h"
hw_timer_t *timer = NULL;
bool tic = false;
int caracteres = 0;

void ARDUINO_ISR_ATTR toc() {
  tic = true;
}

void setup() {
  Serial.begin(115200);
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &toc, true);
  timerAlarmWrite(timer, 1000000, true);
  timerAlarmEnable(timer);
}

void loop() {
  if (tick == true)
  {
    Serial.printf("*");
    tic = false;
    if (++caracteres == 40)
    {
      caracteres = 0;
      Serial.println();
    }
  }
}
```

a. À quelle vitesse s'effectue l'affichage ?
Toutes les secondes.
En quelle unité est défini le « *timer* » ?
En μs soit $10^{-6}s$

b. À l'aide de la fonction `millis()` affichez le temps mesurez entre chaque « *tick* » du timer.

```
#include "esp_system.h"

hw_timer_t *timer = NULL;
bool tick = false;
int caracteres = 0;
volatile unsigned int lastTimer = 0;
volatile unsigned int currentTimer = 0;
void ARDUINO_ISR_ATTR resetModule() {
  currentTimer = millis();
  tick = true;
}

void setup() {
  Serial.begin(115200);
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &resetModule, true);  //attach callback
  timerAlarmWrite(timer, 1000000, true); //set time in us
  timerAlarmEnable(timer);
  currentTimer = millis();                           //enable interrupt
}

void loop() {
  if (tick == true)
  {
    Serial.printf("Duree : %d\n",currentTimer-lastTimer);
    lastTimer = currentTimer;
    tick = false;
  }
}
```

c. Que se passe-t-il :
- ⋆ si vous essayez de diminuer le temps de déclenchement du timer ?
- ⋆ si vous mettez le calcul et l'affichage de la durée dans la fonction `tick` ?

d. Soit le code suivant :

```
portMUX_TYPE m = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR toc()
{
portENTER_CRITICAL_ISR(&m);
...
portEXIT_CRITICAL_ISR(&m);
}
```

Que fait le code ?
Il fait un mutex.
À quoi sert ❶ ?
à mettre le code dans la ram et pas dans la flash.
Qu'est-ce que veut dire ❷ ?
Reprenez votre code pour intégrer cette opération.

```
#include "esp_system.h"
hw_timer_t *timer = NULL;
bool tick = false;
int caracteres = 0;
unsigned int duree = 0;
unsigned int nb_ticks = 0;
unsigned int lastTimer = 0;
unsigned int currentTimer = 0;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void ARDUINO_ISR_ATTR resetModule() {
  portENTER_CRITICAL_ISR(&timerMux);
  currentTimer = millis();
  nb_ticks++;
  tick = true;
  duree = currentTimer - lastTimer;
  //Serial.printf("Duree : %d\n",currentTimer-lastTimer);
  lastTimer = currentTimer;
  portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {
  Serial.begin(115200);
  timer = timerBegin(0, 80, true);
```

```
  timerAttachInterrupt(timer, &resetModule, true);  //attach callback
  timerAlarmWrite(timer, 500000, true); //set time in us
  timerAlarmEnable(timer);
  currentTimer = millis();                          //enable interrupt
}

void loop() {
  if (tick == true)
  {
    //Serial.printf("Duree : %d\n",currentTimer-lastTimer);
    Serial.printf("Tick numero: %d  ",nb_ticks);
    Serial.printf("Duree : %d\n",duree);
    //lastTimer = currentTimer;
    tick = false;
  }
}
```

Si vous n'arrivez pas à programmer votre ESP32

1. Maintenir appuyé le bouton "BOOT" ;
2. Appuyer le bouton "RST" ;
3. Relâcher le bouton "BOOT" ;

`https://docs.espressif.com/projects/esptool/en/latest/esp32c3/advanced-topics/boot-mode-selection.html`