Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Systèmes Embarqués

TP n°1

Programmation avec Arduino

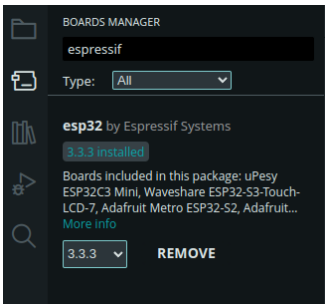
Utilisation de l'ESP32 s3

Vous téléchargerez l'IDE Arduino, lui donerez le droit d'exécution et le lancerez :

xterm

```
$ wget
https://downloads.arduino.cc/arduino-ide/arduino-ide_2.3.10_Linux_64bit.AppImage
$ chmod +x arduino-ide_2.3.10_Linux_64bit.AppImage
$ ./arduino-ide_2.3.10_Linux_64bit.AppImage
```

Et vous configurerez l'utilisation de l'ESP32 s3 dans l'IDE en sélectionnant le constructeur « espressif » :



Il est possible d'accéder directement à ce panneau en utilisant les icones réparties verticalement à gauche de la fenêtre de l'IDE.

Attention

La carte, ou « *devboard* », à sélectionner est la « *ESP32-S3-Matrix* ».

Broches GPIO

1 – Récupérez le « *schematic* » de la carte de développement, « *devboard* », de waveshare sur
`https://files.waveshare.com/wiki/ESP32-S3-Matrix/ESP32-S3-Matrix-Sch.pdf`

a. Trouvez la broche correspondant à :
- ◇ la « guirlande » de LEDs de type WS2812 : l'ensemble de la matrice est en série pilotée par une seule broche ;
  Sachant que l'ESP32 ne peut alimenter l'ensemble des LEDs, il pilote uniquement un transistor, un MOSFET DMG1012T-7, qui agit comme un interrupteur sur le circuit alimentant les LEDs.
- ◇ le bouton "BOOT" ;

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 ?

Expliquez ce que cela veut dire par rapport au états électriques.

2 – a. Vous testerez le programme suivant :

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

void setup() {
  pinMode(buttonPin, INPUT);
}

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

  if (buttonState == LOW) {
    printf("Low\n");
  }
}
```

Vous mettrez le numéro de la broche que vous avez trouvée

Que fait-il ?
Si vous changez l'initialisation de la broche de `INPUT` à INPUT_PULLUP, que se passe-t-il ?
Expliquez le comportement.

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.
Est-ce que cela marche comme vous le voulez ?
Pourquoi ?
Est-ce que votre programme détecte un changement d'état sur la broche associée au bouton ?

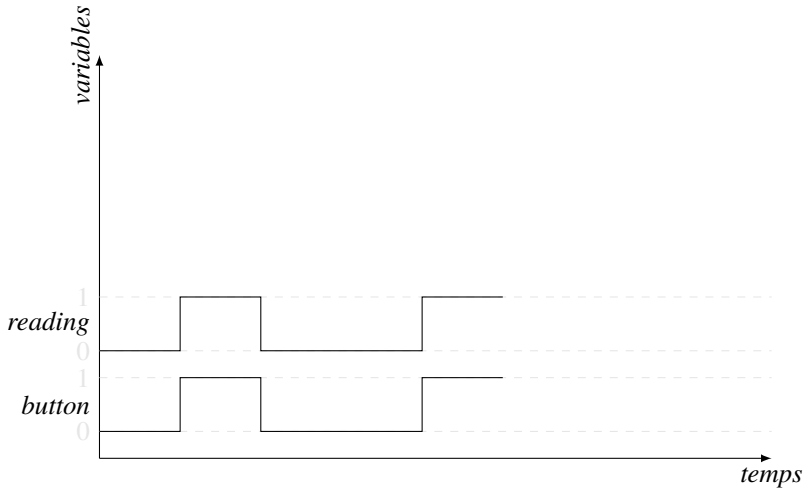
c. Dans les exemples fournis par l'IDE, chargez « *File>Examples>02.Digital>Debounce* ».

Expliquez ce qu'il fait ?

Qu'est-ce que le « *debounce* » d'un bouton ?

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

Modélisez dans un chronogramme qu'est-ce que deviennent les variables du programme lorsque l'on appuie sur le bouton.



IRQ

3 – Vous essaierez le programme suivant :

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

Button button1 = {0, 0, false};

void IRAM_ATTR mon_isr(void* arg) {
    Button* s = (struct Button *) arg;
    s->numberKeyPresses += 1;
    s->pressed = true;
}

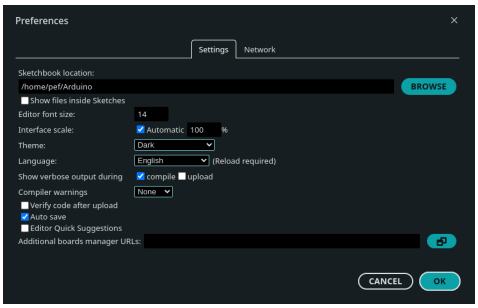
void setup() {
    pinMode(button1.PIN, INPUT_PULLUP);
    attachInterruptArg(digitalPinToInterrupt(button1.PIN), mon_isr, &button1, RI
SING);
}

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

a. Pourquoi met-on `RISING` et non pas `FALLING` ?

b. Que se passe-t-il si vous déplacez l'affichage du compteur d'appui dans la fonction `isr` ? Pourquoi ?

4 – On va regarder le placement de la fonction de gestion d'IRQ en mémoire.
Configurez dans les préférences « Show verbose output » :



Ainsi, après la compilation on devrait lire quelque chose de similaire à :

```
esptool v5.1.0
Wrote 0x400000 bytes to file '/home/pef/.cache/arduino/sketches/EE680B36CC6830B539E7D63362875653/sketch_nov13b.ino.merged.bin', ready to flash to offset 0x0.
/home/pef/.arduino15/packages/esp32/tools/esp-x32/2507/bin/xtensa-esp32s3-elf-size -A /home/pef/.cache/arduino/sketches/EE680B36CC6830B539E7D63362875653/sketch_nov13b.ino.elf
Sketch uses 275539 bytes (21%) of program storage space. Maximum is 1310720 bytes.
Global variables use 20840 bytes (6%) of dynamic memory, leaving 306840 bytes for local variables. Maximum is 327680 bytes.
```

On obtient le chemin d'accès au fichier « .elf », ici :

`/home/pef/.cache/arduino/sketches/EE680B36CC6830B539E7D63362875653/sketch_nov13b.ino.elf`

On va maintenant chercher l'adresse de la fonction « `mon_isr` » dans le « *firmware* » :

xterm

```
$ nm -C /home/pef/.cache/arduino/sketches/EE680B36.../sketch_nov13b.ino.elf | grep
mon_isr
4037799c T mon_isr(void*)
```

On obtient l'adresse de la fonction `mon_isr`

Si on supprime dans le code l'attribut de définition de la fonction :

```
void IRAM_ATTR mon_isr(void* arg) {
```

```
void mon_isr(void* arg) {
```

xterm

```
 nm -C /home/pef/.cache/arduino/sketches/EE680B36.../sketch_nov13b.ino.elf | grep
mon_isr
4201cba8 T mon_isr(void*)
```

Dans le fichier

`/.arduino15/packages/esp32/tools/esp32-arduino-libs/idf-release_v5.5-f1a1df9b-v3/esp32s3/ld/memory.ld`

on trouve la configuration du linker :

```
MEMORY
{
  /**
   *  All these values assume the flash cache is on, and have the blocks this uses subtracted from the length
   *  of the various regions. The 'data access port' dram/drom regions map to the same iram/irom regions but
   *  are connected to the data port of the CPU and eg allow byte-wise access.
   */
  /* IRAM for PRO CPU. */
  iram0_0_seg (RX) : org = (0x40370000 + 0x4000), len = (((0x403CB700 - (0x40378000 - 0x3FC88000)) - 0x3FC88000) + 0x8000 - 0x4000)
  /* Flash mapped instruction data */
  iram0_2_seg (RX) : org = 0x42000020, len = 0x800000-0x20
```

5 – Dans quelle zone mémoire se trouve la fonction `mon_isr` :

a. avec l'attribut « `IRAM_ATTR` » ?

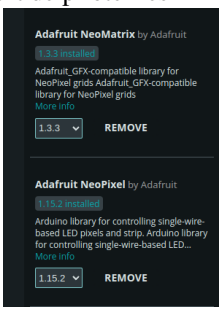
b. sans l'attribut « `IRAM_ATTR` » ?

Expliquez pourquoi ?

6 – a. Comment fonctionne une LED RGB WS2812 ?

`https://www.sdiplight.com/what-is-ws2812b-led-and-how-to-use-ws2812b-led/`

Vous installerez la bibliothèque permettant de piloter les LEDs WS2812 de Adafruit :




```
#define RGB_BRIGHTNESS 10 // ATTENTION ne pas dépasser sous peine de détruire l'ESP32
#define PIN_NEOPIXEL      // Vous mettrez la broche que vous avez trouvée
void setup() {
  // No need to initialize the RGB LED
}
// the loop function runs over and over again forever
void loop() {
  neopixelWrite(PIN_NEOPIXEL, RGB_BRIGHTNESS, 0, 0);  // Red
  delay(1000);
  neopixelWrite(PIN_NEOPIXEL, 0, RGB_BRIGHTNESS, 0);  // Green
  delay(1000);
  neopixelWrite(PIN_NEOPIXEL, 0, 0, RGB_BRIGHTNESS);  // Blue
  delay(1000);
  neopixelWrite(PIN_NEOPIXEL, 0, 0, 0);  // Off / black
  delay(1000);
}
```

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.

7 – Que fait le programme suivant :

```
#include <Adafruit_NeoPixel.h>

#define RGB_Control_PIN   14
#define Matrix_Row        8
#define Matrix_Col        8
#define RGB_COUNT         64

uint8_t Matrix_Data[8][8];
Adafruit_NeoPixel pixels(RGB_COUNT, RGB_Control_PIN, NEO_RGB + NEO_KHZ800);

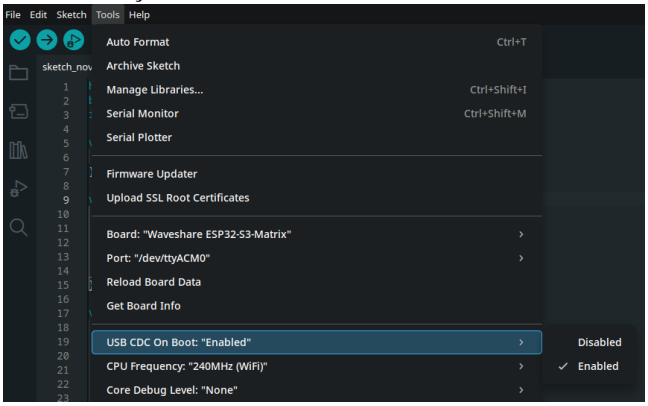
void Matrix_Init() {
  pixels.begin();
  pixels.setBrightness(10); // ATTENTION Ne pas dépasser cette valeur !!!
  memset(Matrix_Data, 1, sizeof(Matrix_Data));
}
void Matrix_zone(int zone, int color[3])
{
  int row_min = 0, row_max = 0;
  int col_min = 0, col_max = 0;
  switch(zone){
    case 0: row_min = 0; row_max = 3; col_min = 0; col_max = 3; break;
    case 1: row_min = 4; row_max = 7; col_min = 0; col_max = 3; break;
    case 2: row_min = 0; row_max = 3; col_min = 4; col_max = 7; break;
    case 3: row_min = 4; row_max = 7; col_min = 4; col_max = 7; break;
  }
  for (int row = row_min; row <= row_max; row++) {
    for (int col = col_min; col <= col_max; col++) {
      pixels.setPixelColor(row*8+col, pixels.Color(color[0],color[1],color[2]));
    }
  }
  pixels.show();
}
void setup()
{
  Matrix_Init();
}
//int x=0;
typedef enum {
    RED, GREEN, BLUE, BLACK, YELLOW, COLOR_COUNT
} Color;

int RGB[COLOR_COUNT][3] = {
    [RED]    = {255, 0,   0},
    [GREEN] = {0,   255, 0},
    [BLUE]   = {0,   0,   255},
    [BLACK] = {0,   0,   0},
    [YELLOW] = {255, 165, 0}
};

void loop()
{
  static int zone_choisie = 0;
  static int couleur_choisie = 0;


  Matrix_zone(zone_choisie, RGB[couleur_choisie]);
  delay(300);
  zone_choisie = (zone_choisie + 1) % 4;
  couleur_choisie = (couleur_choisie + 1 ) % COLOR_COUNT;
}
```

Vous configurerez Arduino de la façon suivante :



Timers

8 – Vous essaierez le code suivant :

```
hw_timer_t *timer = NULL;
bool tic = false;
int caracteres = 0;

void ARDUINO_ISR_ATTR toc() {
    tic = true;
}

void setup() {
  Serial.begin(115200);
  while (!Serial) { delay(10); }
  timer = timerBegin(1000000);
  timerAttachInterrupt(timer, &toc);
  timerAlarm(timer, 1000000, true, 0);
}

void loop() {
    if (tic == true)
  {
    Serial.printf("*");
    tic = false;
    if (++caracteres == 10)
    {
      caracteres = 0;
      Serial.println();
    }
  }
}
```

a. À quelle vitesse s'effectue l'affichage ?
En quelle unité est défini le « *timer* » ?

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

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 ❶IRAM_ATTR toc()
{
portENTER_CRITICAL❷ISR(&m);
...
portEXIT_CRITICAL_ISR(&m);
}
```

Que fait le code ?
À quoi sert ❶ ?
Qu'est-ce que veut dire ❷ ?
Reprenez votre code pour intégrer cette opération.

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`