Contents

1 Introduction **2**

2 TP arduino avec LED **2**
- 2.1 Vérifier le matériel . . . 2
- 2.2 1er vrai montage Arduino: Faire clignoter la LED . . . 3
  - 2.2.1 Montage . . . 3
  - 2.2.2 Code . . . 4
  - 2.2.3 Avant void setup() . . . 4
  - 2.2.4 void setup() . . . 4
  - 2.2.5 void loop() . . . 5
- 2.3 Envoyer le code sur la carte . . . 5

3 Contrôler une LED grâce à un bouton **6**
- 3.1 Montage . . . 6
- 3.2 Code . . . 7
  - 3.2.1 Avant void setup() . . . 7
  - 3.2.2 void setup() . . . 8
  - 3.2.3 void loop() . . . 8

4 Utilisation d'un détecteur DHT et d'un écran LCD I2C **9**
- 4.1 Utilisation d'un DHT . . . 9
  - 4.1.1 Montage . . . 9
  - 4.1.2 Code . . . 10
  - 4.1.3 Avant void setup() . . . 10
  - 4.1.4 void setup() . . . 10
  - 4.1.5 void loop() . . . 11
- 4.2 Utilisation d'un LCD . . . 11
  - 4.2.1 Montage . . . 12
  - 4.2.2 Code . . . 12
  - 4.2.3 Avant void setup() . . . 12
  - 4.2.4 void setup() . . . 13
  - 4.2.5 void loop() . . . 13
- 4.3 Utilisation d'un LCD pour afficher les mesures du DHT . . . 14

4.3.1 Montage 14
4.3.2 Code 15
4.3.3 Avant void setup() 15
4.3.4 void setup() 15
4.3.5 void loop() 16

5 Système d'irrigation simple **16**
5.1 Montage 16
5.2 Code 18
5.2.1 Avant void setup() 18
5.2.2 void setup() 18
5.2.3 void loop() 18

6 Annexes **20**
6.1 Conversion pins ESP 32 20

1 Introduction

Ce stage a pour but d'apprendre à utiliser Arduino, **n'hésitez pas à nous poser des questions** si vous en avez, ou bien de tester des choses par vous-même.

Vous trouverez ci-joint des captures d'écran du code à titre de comparaison, ne vous embêtez pas à les réécrire, les codes vous seront fournis directement sur les ordinateurs du Fab-Lab.

Si vous êtes intéressés, une version détaillée des protocoles et des explications détaillées des codes et montages sera disponible à la fin du stage.

2 TP arduino avec LED

2.1 Vérifier le matériel

Dans cette section nous aurons besoin de :

- 2 fils (1 rouge et 1 noir)
- 1 LED
- 1 résistance (de 230 ou 330 Ω)

Nous allons réaliser le montage suivant, dont le but est simplement de vérifier que la carte envoie du courant correctement une fois branchée sur un ordinateur.

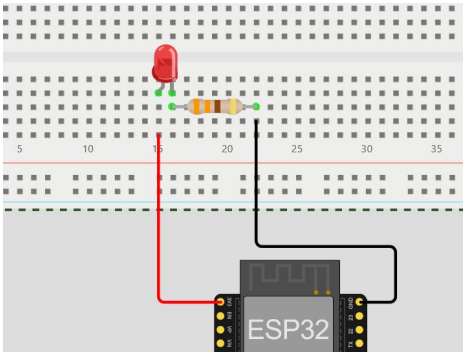

Figure 1: Schéma du montage pour allumer une LED

La LED devrait normalement s'allumer, si ce n'est pas le cas, vérifiez si le montage a bien été fait comme indiqué. Demandez de l'aide si besoin.

2.2 1er vrai montage Arduino: Faire clignoter la LED

L'intérêt d'une carte Arduino est de pouvoir programmer ce que l'on envoie et reçoit comme informations. Chaque partie à partir de maintenant contiendra les informations pour deux choses : le montage et le code associé à ce dernier.

2.2.1 Montage

Afin de faire clignoter une LED, il nous faut contrôler son alimentation en électricité. Pour cela, on change le fil connecté à la borne 3.3V sur un pin de notre choix (ici 18 pour l'exemple).

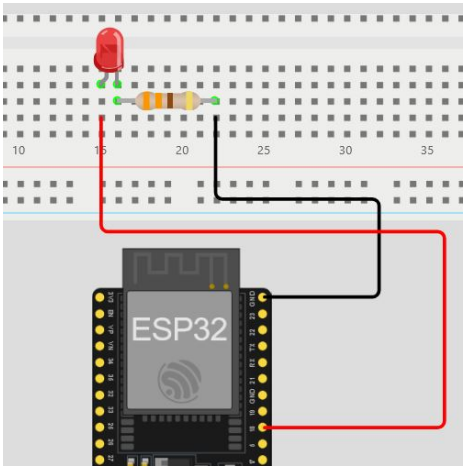

Figure 2: Schéma pour LED clignotante

2.2.2 Code

C'est dans cette partie que l'on traite les informations que l'on contrôle : les informations entrantes et sortantes de la carte.

Le code est composé principalement de 3 parties :

- Les définitions et ajouts de bibliothèques si nécessaire pour le matériel utilisé.
- `void setup()`, qui ne s'applique qu'une fois lorsque l'on envoie le code sur la carte. Souvent utilisé pour donner des informations comme le mode des pins, allumer le moniteur série et tout autre chose qu'on devrait spécifier ou réinitialiser afin d'éviter que des informations de codes précédents n'interfèrent.
- `void loop()`, qui se répète à l'infini une fois `void setup()` passé. On trouvera ici le cœur du traitement des informations.

2.2.3 Avant void setup()

Il n'y a pas de bibliothèques à utiliser pour les LED. On n'a donc seulement le pin de la LED qui est défini comme ceci.

Il faut faire attention à déclarer le bon pin, vous trouverez lequel donner en regardant les numéros après GPIO dans la table de conversion des pins qui est à la fin de ce manuel, dans l'annexe. Ici, on est sur le port numéro 18.

```
1  #define LED 18 // Attention à bien choisir le bon numéro
2
```

Figure 3: Déclaration du numéro de pin nécessaire pour allumer une LED

2.2.4 void setup()

Une fois le pin de la LED défini, il faut savoir ce que la carte doit faire, lire l'information de si la LED est allumée ou bien choisir d'allumer la LED. Ici, on cherche à allumer la LED, on va donc mettre son pin dans le mode OUTPUT.

```
3  void setup() {
4    pinMode(LED, OUTPUT);
5  }
```

Figure 4: Initialisation du code pour faire clignoter une LED

Attention à ne pas oublier le ";" après les lignes, sinon le code ne va pas fonctionner correctement.

2.2.5 void loop()

Dans le `void loop()`, on cherche à faire clignoter une LED, soit à l'allumer et l'éteindre successivement.

Dans le code, la fonction qui permet cela est la fonction `digitalWrite`, qui permet de dire à un port s'il doit envoyer un courant important (HIGH), ou non (LOW).

Afin de contrôler la durée du clignotement, on utilise la commande `delay`, qui fait que l'on attend un temps donné dans la commande avant d'effectuer la suite. Le temps y est exprimé en millisecondes, `delay(1000)` correspond donc à 1 seconde d'attente.

```
7   void loop() {
8     digitalWrite(LED, HIGH);
9     delay(500);
10    digitalWrite(LED, LOW);
11    delay(500);
12  }
```

Figure 5: Code principal qui se répètera

2.3 Envoyer le code sur la carte

Une fois le code entièrement écrit, il faut l'envoyer sur la carte. Afin de savoir s'il est utilisable, on va d'abord ajouter le nom de la carte dans la case suivante, il faut chercher "ESP32 Dev Module" et prendre la prise qui apparaît et qui n'est pas le COM 1 (généralement COM 3 ou COM 4).

Figure 6: Carte à rechercher

Si le bon nom de carte n'apparaît pas sur la liste, alors il faudra les ajouter manuellement avec le gestionnaire de cartes accessible avec ce bouton.

On y entrera `esp32` et normalement, 2 résultats devraient ressortir ; téléchargez les 2. S'il y a un message demandant des accès administrateurs, répondez simplement non.

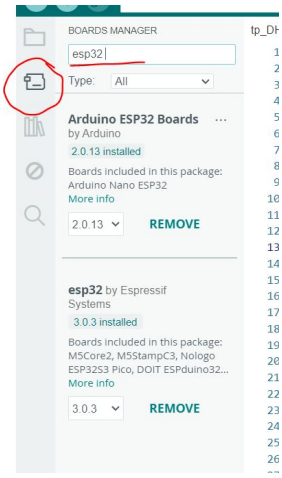

Figure 7: Capture d'écran du gestionnaire de cartes.

Figure 8: Bouton permettant de vérifier si le code contient des erreurs

Une fois qu'Arduino sait à quel type de carte on veut envoyer notre code et par quel port, il faut vérifier le code en lui-même avec le bouton suivant :

Si un message d'erreur apparaît, on va chercher quelles sont les raisons, et corriger les problèmes. Si aucun message d'erreur n'apparaît, alors on peut l'envoyer sur la carte. Pour cela, il faut cliquer sur le bouton suivant :

Figure 9: Bouton permettant de téléverser (envoyer sur la carte) le code

Important dans le cas des cartes que l'on utilise, une fois que le mot "Connecting" apparaît en haut de la fenêtre, il faut appuyer sur le bouton BOOT de la carte.

3 Contrôler une LED grâce à un bouton

3.1 Montage

On cherche ici à avoir 2 informations sur la carte :

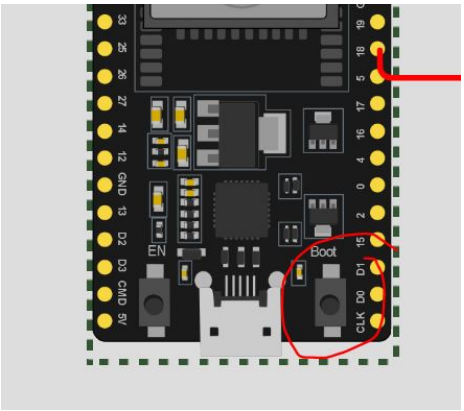

Figure 10: Bouton BOOT permettant de faire fonctionner le téléversement

- L'état du bouton
- Le courant qui passe dans la LED

Pour cela, on connecte d'abord les fils grounds (GND) et 3.3V sur les lignes - et + respectivement du breadboard. Cela permettra d'utiliser la même masse (GND) et le même courant pour les 2 éléments.

Pour la LED, on branche une résistance au ground (GND) et à un pin parmi les 5 à côté. On connecte la branche courte de la LED sur la même colonne que cette résistance. On connecte avec un fil l'autre borne de la LED avec le pin 21.

Pour le bouton, on va d'abord connecter le bouton à cheval entre 2 séries de 5 lignes (afin de ne pas connecter les entrées et les sorties entre elles). On connecte une entrée au ground, l'autre au 3.3V (indifféremment) et une des sorties (n'importe laquelle) au pin 22.

On obtient alors le schéma suivant :

3.2 Code

3.2.1 Avant void setup()

Aucune librairie spécifique n'est nécessaire pour les boutons et LED.

On peut donc se contenter de déclarer sur quels pins se trouvent le bouton et la LED :

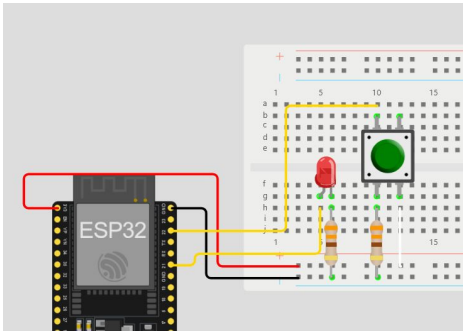

Figure 11: Schéma représentant le montage à réaliser avec le bouton, la LED et la carte

Library Manager

```
1   char redLedPin=21;
2   char redBtnPin=22;
```

Figure 12: Déclaration des numéros de pins correspondants au bouton et à la LED, ici respectivement 22 et 21

3.2.2 void setup()

On a cette fois 2 pins à configurer. Celui de la LED reste inchangé, car on cherche toujours à l'allumer ou à l'éteindre, donc on a un OUTPUT. Pour ce qui est du bouton, on cherche à savoir s'il est appuyé ou non, donc à savoir s'il laisse passer le courant ou non. On doit donc avoir un INPUT.

```
3   void setup() {
4     pinMode(redLedPin,OUTPUT);
5   pinMode(redBtnPin,INPUT);
6   }
```

Figure 13: Initialisation indiquant que la LED représente l'output (la sortie) et le bouton l'input (les informations rentrantes)

3.2.3 void loop()

On veut que la LED s'allume lorsqu'on appuie sur le bouton, on doit donc mettre en place une condition sur l'état du bouton. On va alors utiliser une fonction `if` (traduit par "si") pour vérifier si le bouton est allumé ou non.

Afin de tester l'état du bouton, on va utiliser la fonction `digitalRead`, qui au contraire de `digitalWrite` qui forçait un état sur un pin, va simplement lire

l'état du pin en question.

Dans la première parenthèse, qui sera exécutée si la condition est vérifiée, on va réutiliser le `digitalWrite` pour allumer la LED.

Si la condition n'est pas vérifiée, il pourrait ne rien se passer. Mais dans notre cas, on cherche à éteindre la LED, on va alors utiliser une fonction `else` (traduit par "sinon") qui s'exécutera uniquement si le IF (ou les ifs, s'il y en a plusieurs) n'est pas vérifié.

```
8  void loop() {
9    if (digitalRead(redBtnPin)==HIGH)
10   {
11   digitalWrite(redLedPin,HIGH);
12   }
13   else
14     {
15     digitalWrite(redLedPin,LOW);
16     }
17 }
```

Figure 14: Code principal, quand le bouton est pressé la LED s'allume, s'éteint dans le cas contraire

4 Utilisation d'un détecteur DHT et d'un écran LCD I2C

4.1 Utilisation d'un DHT

4.1.1 Montage

Pour ce montage, on va avoir besoin de :

- Les 2 fils, l'un rouge et l'autre noir des montages précédents.
- 1 fil jaune du montage précédent.
- 1 DHT 11 (boîtier bleu avec une LED et des prises vertes)
- 1 petit tournevis

Vous pouvez remarquer qu'il y a 3 prises sur le DHT, VCC, GND et data. Les 2 premiers sont respectivement le courant 3.3V et la masse. La 3ème prise (au centre) est celle avec laquelle le DHT renvoie ces données, que l'on va dans un premier temps afficher sur le moniteur série (trouvable dans le menu outils).

Ce dernier permet de nombreuses choses, détaillées dans le compte rendu plus précis, il servira ici à afficher les résultats du DHT en l'absence d'écran.

Le montage final est le suivant :

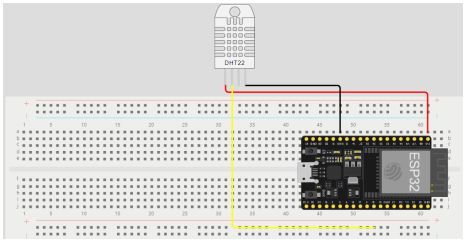

Figure 15: Schéma représentant le montage à réaliser avec le DHT11

4.1.2 Code

4.1.3 Avant void setup()

Le DHT est le 1er élément à avoir besoin de sa propre bibliothèque pour fonctionner. Une bibliothèque est une famille de fonctions qui permet de communiquer avec un élément extérieur à la carte, leurs noms finissent en .h et sont à déclarer en début de code.

On doit aussi déclarer le pin sur lequel on a connecté le DHT, ici le port 18.

Une fois la bibliothèque DHT et le port défini, il faut dire à la bibliothèque où envoyer ces informations et quel type de DHT on utilise, ici, on utilise un DHT 11. Le schéma ci-dessus montre une DHT22, n'en tenez pas compte.

tp_DHT.ino

```
1  #include "DHT.h"
2  #define DHT11_PIN 18
3
4  DHT dht11(DHT11_PIN, DHT11);
```

Figure 16: Informations à déclarer en début de code

4.1.4 void setup()

Dans le `setup` ici, on va juste demander deux choses :

- Que le DHT commence à fonctionner, via la fonction `begin` de la bibliothèque

- De démarrer le moniteur série avec `Serial.begin`. Le nombre à l'intérieur des parenthèses (appelé baudrate) permet de calculer la vitesse à laquelle le moniteur doit lire les informations envoyées afin que le message ne soit pas buggé.

```
6  void setup() {
7    Serial.begin(9600);
8
9    dht11.begin(); // initialize the sensor
10 }
```

Figure 17: Initialisation du code

4.1.5 void loop()

On commence par définir les variables avec nos valeurs. On les définit comme des `float` (nombre à virgules) et les fonctions de DHT11 en face permettant d'obtenir les valeurs souhaitées (ici le pourcentage d'humidité, la température en celsius et celle en farhenheit).

On souhaite ensuite vérifier que les valeurs soient bien mesurées et prises en compte. On utilise alors une boucle `if`, avec comme condition de savoir si nos 3 valeurs sont bien des nombres ou non. La fonction `isnan` signifiant "is not a number" (en français "n'est pas un nombre"), cette fonction vérifie donc si le paramètre (la valeur qui lui a été donnée) n'est pas un nombre.

Si ce n'est pas le cas, on envoie un message d'erreur sur le moniteur série.

Sinon, on part dans le `else`, où l'on écrit nos valeurs dans le moniteur série grâce à la suite de fonctions ci-dessous.

A noter que l'on utilise 2 fonctions différentes, `Serial.print` pour ce que l'on veut écrire, et `Serial.println` afin que l'on change de ligne pour afficher à la suite tout les `Serial.print` depuis le dernier changement de ligne.

Enfin, on impose un délai en fin de code (ici 2 secondes) afin d'avoir un minimum de temps entre chaque mesure et que le moniteur de série soit lisible.

4.2 Utilisation d'un LCD

Plus précisément, on utilisera ici un LCD I2C avec seulement 4 ports de sortie, vérifiez bien que vous avez un module au dos de votre LCD.

```
void loop() {
  // read humidity
  float humi  = dht11.readHumidity();
  // read temperature as Celsius
  float tempC = dht11.readTemperature();
  // read temperature as Fahrenheit
  float tempF = dht11.readTemperature(true);

  // check if any reads failed
  if (isnan(humi) || isnan(tempC) || isnan(tempF)) {
    Serial.println("Failed to read from DHT11 sensor!");
  } else {
    Serial.print("DHT11# Humidity: ");
    Serial.print(humi);
    Serial.print("%");

    Serial.print("  |  ");

    Serial.print("Temperature: ");
    Serial.print(tempC);
    Serial.print("°C ~ ");
    Serial.print(tempF);
    Serial.println("°F");
  }
  delay(2000);
}
```

Figure 18: Code principal pour utiliser le DHT

4.2.1 Montage

Pour ce montage, nous aurons besoin de :

- Les 2 fils rouge et noir précédents.
- 2 nouveaux fils un violet et un vert
- Un LCD

Il y a 4 bornes sur votre LCD :

- La borne GND, reliée par le fil noir à la borne GND de la carte
- La borne VCC, reliée par le fil rouge à la borne 3.3V de la carte
- La borne SDA, reliée par le fil vert à la borne 21 de la carte
- La borne SCL, reliée par le fil violet à la borne 22 de la carte.

Attention, les ports pour fils de data sont libres jusque-là mais pour le LCD il faut bien faire attention à se connecter spécifiquement au 21 et 22 pour que le montage fonctionne.

4.2.2 Code

4.2.3 Avant void setup()

On doit encore une fois utiliser la bibliothèque spécifique au LCD I2C que nous utilisons, sans donner de pin spécifique cette fois car les pins I2C sont définis par défaut dans la carte, d'où la nécessité de les connecter à ces ports spécifiques dans le montage.

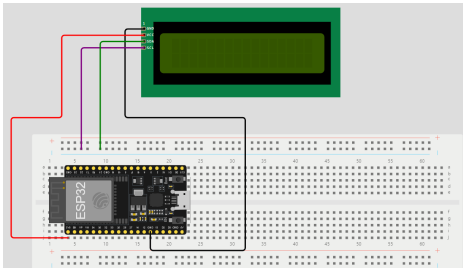

Figure 19: Montage à réaliser pour utiliser un LCD (écran)

On va ensuite donner à la bibliothèque les caractéristiques de notre écran, tout d'abord son adresse avec 0x27 puis sa forme avec 2, 16 (2 lignes, 16 colonnes)

tp_LCD.ino

```
1  #include <LiquidCrystal_I2C.h>
2
3  LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Figure 20: Informations à déclarer au début du code

4.2.4 void setup()

Dans le `void setup()`, on demande au LCD de s'initialiser et d'allumer ses lumières de fond.

```
5  void setup()
6  {
7    lcd.init();
8    lcd.backlight();
9  }
```

Figure 21: Initialisation code LCD

4.2.5 void loop()

Ici, dans une idée similaire au moniteur de série, on va utiliser la fonction `print` pour écrire le texte voulu, que vous pouvez modifier à volonté en évitant autant que possible les accents et les caractères spéciaux que le LCD ne comprend pas.

```
11  void loop()
12  {
13    lcd.clear();
14    lcd.setCursor(0, 0);
15    lcd.print("Arduino");
16    lcd.setCursor(2, 1);
17    lcd.print("Stage Ore");
18    delay(2000);
19
20    lcd.clear();
21    lcd.setCursor(3, 0);
22    lcd.print("Tuto utilisation LCD");
23    lcd.setCursor(0, 1);
24    lcd.print("Reussi");
25    delay(2000);
26  }
```

Figure 22: Code principal pour utiliser un LCD

Contrairement au moniteur de série cependant, on doit mettre à la ligne le texte de nous-même, d'où l'utilisation de la commande `setCursor` qui permet de placer le 1er caractère de notre texte. Les 2 chiffres correspondant d'abord à la colonne puis la ligne et la numérotation commence à 0.

On doit de plus utiliser la commande `clear` entre chaque message, sans cela les messages s'écriront au-dessus les uns des autres et créeront des erreurs, et les `delay` sont simplement là pour que les messages restent suffisamment longtemps pour être lisibles.

4.3 Utilisation d'un LCD pour afficher les mesures du DHT

Maintenant que nous avons vu l'utilisation du LCD et celle du DHT de manière séparée, voyons comment combiner les deux et utiliser le LCD pour visualiser les données collectées par le DHT au lieu d'utiliser le moniteur série.

4.3.1 Montage

Pour ce montage, nous aurons besoin des éléments suivants :

- Les 2 fils rouge et noir précédents plus deux nouveaux fils des mêmes couleurs.
- Les 2 fils violet et vert du montage précédent
- Un fil jaune

Le montage est sensiblement le même que pour les deux précédents, il faut simplement combiner les deux comme suit :

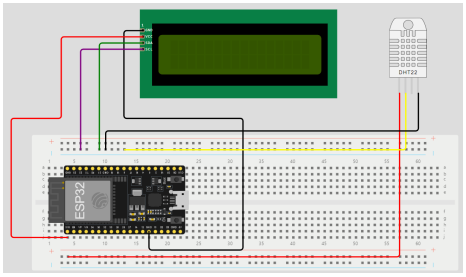

Figure 23: Montage à réaliser pour combiner les deux appareils et les utiliser ensemble

4.3.2 Code

4.3.3 Avant void setup()

Comme pour précédemment, il est nécessaire d'ajouter des déclarations, notamment les librairies utilisées, le pin du DHT et les précisions pour le LCD (si besoin, relire les explications des déclarations des deux montages réalisés avant).

```
1  #include <DHT.h>
2  #include <LiquidCrystal_I2C.h>
3  #include <WiFi.h>
4
5  #define DHT11_PIN 18
6
7  LiquidCrystal_I2C lcd(0x27, 16, 2);
8  DHT dht11(DHT11 PIN, DHT11);
```

Figure 24: Informations à indiquer dans le fichier avant le code principal

4.3.4 void setup()

Le `void setup()` est, lui aussi, une simple superposition des 2 codes précédents, l'exception notable du `Serial.begin`, qui est simplement recommandé à titre de vérifications, mais n'est plus nécessaire car le LCD devrait afficher les valeurs à la place du moniteur série.

```
10 void setup() {
11   Serial.begin(9600);
12   lcd.init(); // initialize the lcd
13   lcd.backlight();
14   dht11.begin();
15 }
```

Figure 25: Initialisation du code combinant LCD et DHT

4.3.5 void loop()

Le `void loop()` suit le même schéma que le `void setup()`, on peut aussi rajouter les `Serial.print` après le "if" si on souhaite régler d'éventuels soucis mais ce n'est pas du tout nécessaire (ces derniers seront présents en commentaire dans le code fourni).

```
17  void loop() {
18    delay(1000);
19    float humi  = dht11.readHumidity();
20    float tempC = dht11.readTemperature();
21    float tempF = dht11.readTemperature(true);
22    if (isnan(humi) || isnan(tempC) || isnan(tempF)) {
23        Serial.println("Failed to read from DHT11 sensor!");
24        lcd.setCursor(0, 0);         // move cursor to   (0, 0)
25      lcd.print("Failed to read data");
26      }
```

Figure 26: Code principal, partie if (si)

```
27    else {
28      Serial.print("DHT11# Humidity: ");
29      Serial.print(humi);
30      Serial.print("%");
31
32      Serial.print("  |  ");
33
34      Serial.print("Temperature: ");
35      Serial.print(tempC);
36      Serial.print("°C ~ ");
37      Serial.print(tempF);
38      Serial.println("°F");
39
40      lcd.clear();
41      lcd.setCursor(0,0);
42      lcd.print("Pseudo");
43      lcd.setCursor(0,1);
44      lcd.print("Station Meteo");
45      delay(1500);
46
47      lcd.clear();
48      lcd.setCursor(0,0);
49      lcd.print("Temperature:");
50      lcd.setCursor(0,1);
51      lcd.print(tempC);
52      lcd.print("C // ");
53      lcd.print(tempF);
54      lcd.print("F");
55      delay(1500);
56
57      lcd.clear();
58      lcd.setCursor(0,0);
59      lcd.print("Hueiddte: ");
60      lcd.setCursor(0,1);
61      lcd.print(humi);
62      lcd.print("%");
63      delay(1500);
64    }
65  }
```

Figure 27: Code principal, partie else (sinon)

5 Système d'irrigation simple

5.1 Montage

Pour le montage, on aura besoin de :

- 1 bouton
- 1 pompe simple
- 1 relais
- 2 fils rouges, 2 fils noirs et 2 fils jaunes précédemment utilisés
- 1 fil jaune, 1 fil noir et 2 fils rouges supplémentaires.

On va tout d'abord lier un ground (GND) et le 3.3V à des barres + ou - du breadboard afin d'avoir le plus d'espace possible pour connecter le reste du montage par la suite.

Pour la pompe, on va connecter sa borne rouge au relais sur la borne notée NC (normally closed, soit normalement fermé en français) et sa borne noire sera reliée au ground général du montage.

Pour le relais, il possède 6 connexions, 3 de chaque côté. L'un avec des ports : GND, VCC et IN, l'autre avec des ports: NC, NO et COM

Sur le premier, on connectera GND et VCC à la masse et à 3.3V respectivement. Le port "IN" sert quant à lui à donner l'information au relais de quel sortie il doit avoir, on le mettra donc avec un fil jaune relié à un port de la carte.

De l'autre côté, on aura comme dit plus haut la pompe sur le PORT NC, rien sur le port NO, qui pourrait servir si on devait avoir quelque chose d'alimenter par défaut en absence de pompe. Et le port COM doit être relié au courant nécessaire pour que la pompe fonctionne, ici on a besoin de 3.3V, mais on pourrait avoir n'importe quel autre courant.

On ajoute aussi un capteur d'humidité du sol au montage, ayant 3 bornes : VCC et GND pour le courant auquel on ajoute 1 borne pour les données qui seront envoyées sur un des ports analogiques de la carte (port noté ADC sur le pinout).

On rajoute en plus un bouton de la même manière que pour la LED.

Le montage finalement obtenu est le suivant, à la différence que **le relais n'est pas connecté au 5V mais bien au 3.3V**. Si on le connecte au 5V, des erreurs peuvent apparaître au niveau de la connexion USB avec le PC :

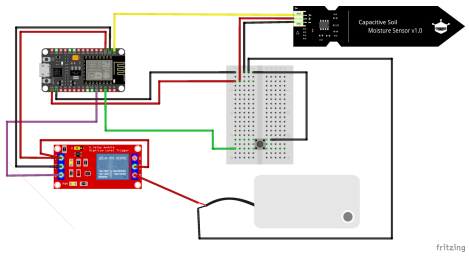

Figure 28: Schéma du montage de la pompe

5.2 Code

5.2.1 Avant void setup()

On doit définir ici les numéros des pins pour le bouton, le relais et le capteur d'humidité.

On va définir aussi 4 variables. La 1ère servant à définir le seuil d'humidité à partir duquel on doit mettre la pompe en marche. Les 3 autres serviront plus tard pour le fonctionnement du bouton, ainsi que à définir l'état du relais.

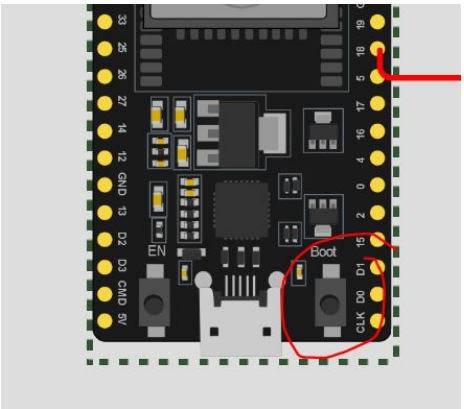

Figure 29: Informations à indiquer dans le fichier avant le code principal

5.2.2 void setup()

On définit ici le mode des pins pour les 4 pins qui nous intéressent. On peut aussi rajouter un `Serial.begin` pour print des valeurs par la suite afin de comprendre des bugs éventuels.

5.2.3 void loop()

On définit d'abord le pourcent d'humidité en fonction de la valeur en sortie du capteur. C'est tout l'intérêt de la fonction `map` qui permet de remettre sur une valeur entre 0 et 100 le range de valeurs que peut envoyer le capteur qui sont des entiers encodés sur 12 bits.

On check ensuite l'état du bouton que l'on stock dans une variable.

On a ensuite 2 fonctions `if`, la 1ère pour tester s'il y a moins d'humidité que le seuil que l'on a fixé plus tôt.

Une autre pour tester si l'état du bouton a changé.

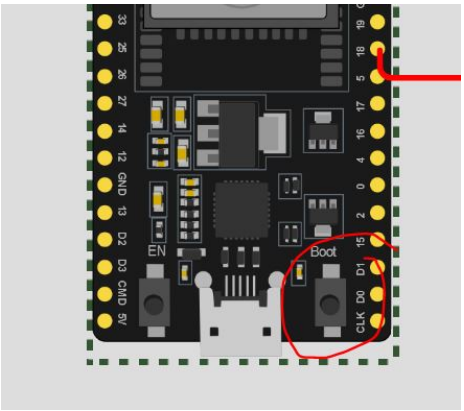

Figure 30: Initialisation du code combinant LCD et DHT

Dans les 2 cas, on a une commande qui demande au relais d'envoyer du courant dans la pompe.

Si aucune des 2 conditions n'est vérifiée, alors on exécute la fonction `else` (sinon) qui demande au relais de couper le courant dans la pompe. Suivi d'un `delay` en fin de code afin de ne pas griller les composants en leur demandant trop de changements en trop peu de temps.

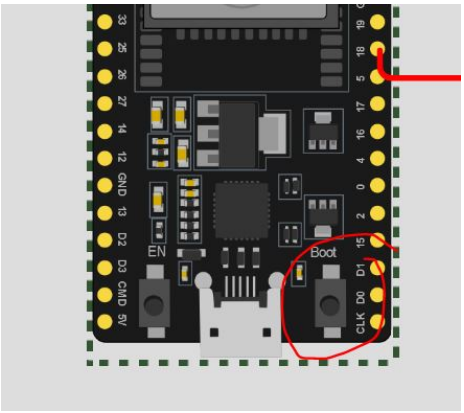

Figure 31: Code principal pour utiliser une pompe dans un système d'irrigation

6 Annexes

6.1 Conversion pins ESP 32

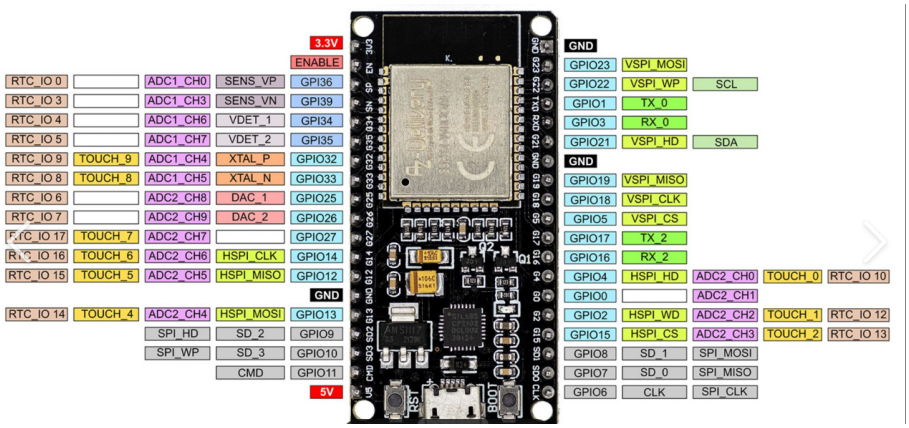

Figure 32: Carte de conversion des pins d'une ESP 32