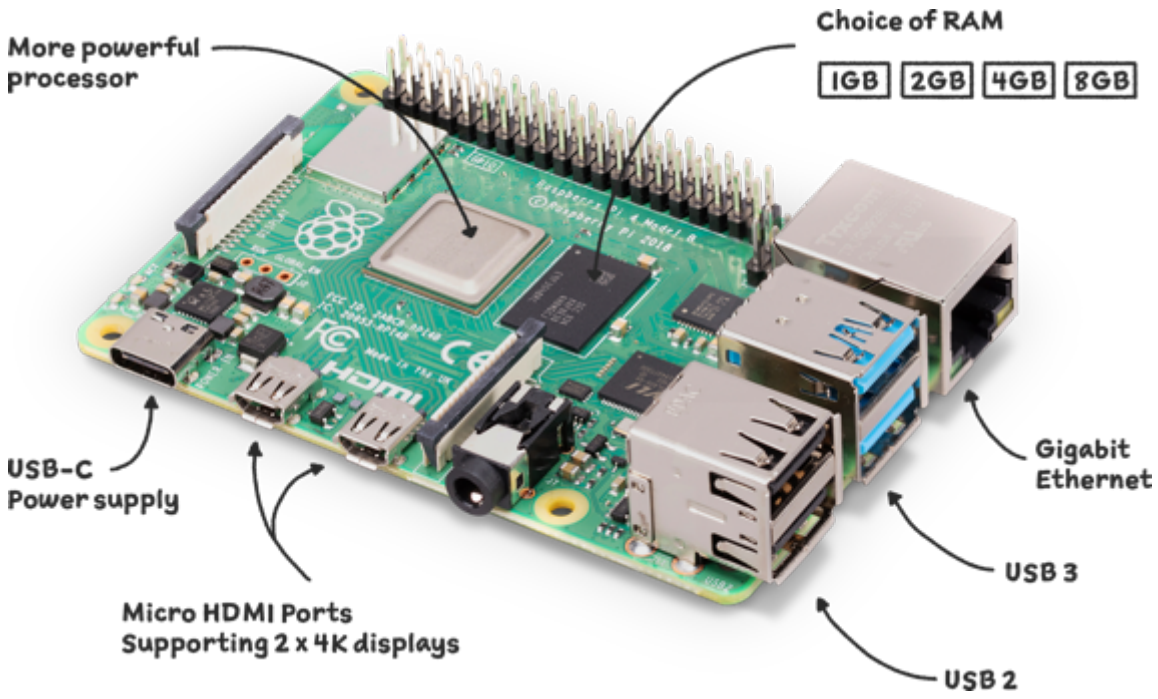
2022-2023

Rapport de projet - GEII

PROJET SHIELD RASPBERRY PI



Réalisé par

Corantin ALLORY
Audren VAUR

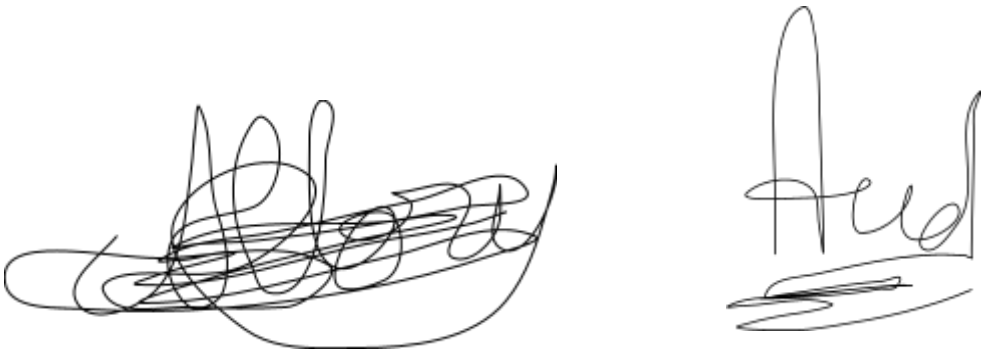
Encadré par

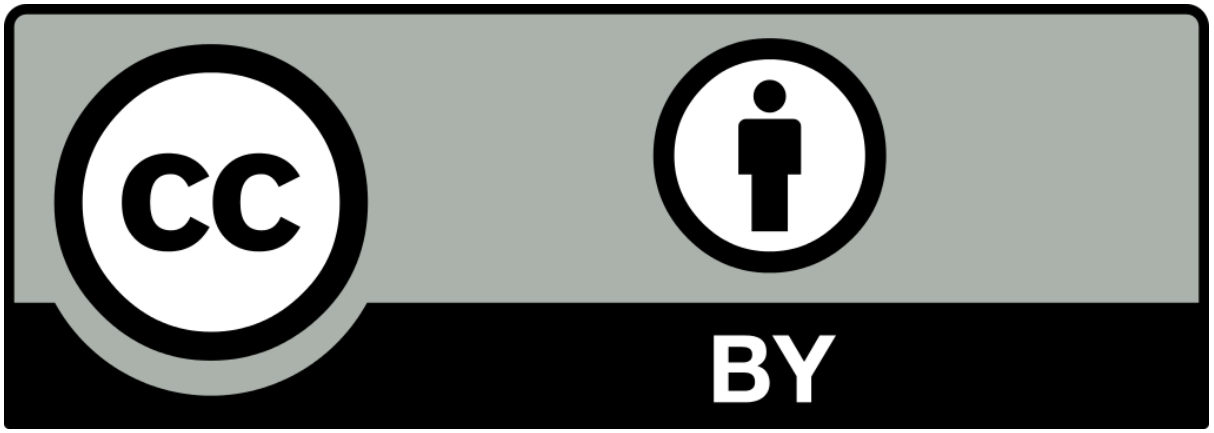
Philippe LUCIDARME
Lionel LEDUC

DÉCHARGE DE RESPONSABILITÉS

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

LICENCE





REMERCIEMENTS

Tout d’abord, nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance aux personnes suivantes, pour leur soutien dans la concrétisation de ce projet :

- M. Philippe LUCIDARME, responsable projet, pour ses conseils éclairés, sa patience, sa disponibilité et sa confiance qu’il nous a accordée lors de ce projet ; ainsi que pour sa mise à notre disposition de tous les moyens disponibles afin de mener à bien ce projet.
- M. Lionel LEDUC, responsable projet, pour nous avoir accordé toute la confiance nécessaire pour élaborer ce projet librement.
- M. Dorian BENECH, camarade de promotion en GEII, pour ses conseils avisés et sa précieuse aide lors de la conception de ce projet.
- M. Noah JUSTEAU, camarade de promotion en GEII, pour son aide précieuse, pour ses conseils et pour ses connaissances transmises durant notre projet.
- Ugo MAIURANO, camarade de promotion en GEII, pour son aide apportée durant ce projet.
- M. Gaëtan CORABOEUF, camarade de promotion en GEII, pour ses conseils apportés lors de ce projet.
- L’IUT d’Angers GEII pour sa coopération professionnelle tout au long de cette expérience et pour s’être mis à notre disposition lors de la réalisation de ce projet.

SOMMAIRE

Introduction.....1

1. Cahier des charges.....2

1.1. Le projet.....2
1.2. Diagramme de Gantt.....2
1.3. Les livrables.....3

2. Choix des composants..... 3

2.1. La Raspberry Pi..... 3
2.2. Le capteur de température.....4
2.3. L'encodeur..... 5
2.4. Les boutons poussoir et les LEDs..... 7

3. Réalisation.....8

3.1. Schémas général.....8
3.2. Le software..... 9
3.3. Le hardware..... 12

Conclusion et Perspectives.....17

Bibliographie.....18

Introduction

Dans le monde, il y a plus de 30 milliards d'objets connectés. Ce nombre ne cesse d'accroître et augmente de 20% chaque année. De nombreuses activités se voient impactées par cette augmentation tel que l'industrie avec le passage à l'industrie 4.0. L'industrie 4.0 est une nouvelle façon de produire en intégrant les nouvelles technologies comme les objets connectés. Un objet connecté est un objet ou un appareil faisant partie d'un réseau dans lequel il met à disposition ses données.

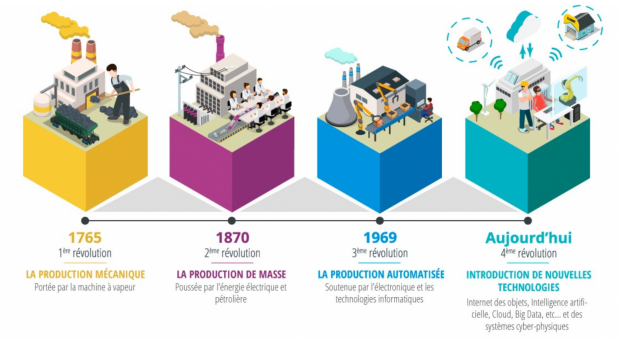
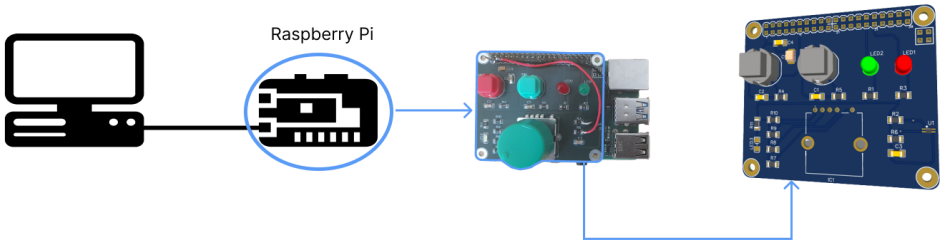


Schéma de l'évolution de l'industrie

Nous, GEII, sommes pleinement impactés par ces évolutions. Et afin de nous préparer au mieux au monde de demain, M. Fasquel et M. Lucidarme, tous deux professeurs à l'IUT d'Angers, ont décidé de mettre en place un nouveau module pour les troisièmes années. Ce module serait la réalisation d'un projet sur une raspberry pi. Ainsi notre projet de SAE est de réaliser une carte s'insérant sur ce raspberry pi. Nous sommes chargés de la partie IHM de l'objet connecté afin que les utilisateurs puissent utiliser les différents boutons et capteurs durant le projet.

Schéma général du projet :



1. Cahier des charges

1.1. Le projet

Notre projet est la réalisation d'une carte électronique s'insérant sur la Raspberry Pi. Notre projet part du dimensionnement de la carte et des composants jusqu'à la réalisation de la carte finale.

Il y a 3 points importants de notre projet. Tout d'abord la création d'une carte compact avec un routage optimisé, puis un IHM défini avec un nombre de capteurs et d'éléments définis. Pour finir, une librairie avec des fonctions précises à réaliser.

Les contraintes présentes dans cette SAE sont tout d'abord une contrainte de dimension de la carte. La carte doit pouvoir tenir sur la raspberry pi sans dépasser sur les côtés. Ensuite, il y a la contrainte due au GPIO et plus généralement due à la raspberry. Nous devons donc designer une carte qui respecte les contraintes dues aux GPIO. Aussi, des fonctions de la librairie sont déjà définies. La dernière contrainte est une contrainte de temps pour réaliser le projet, en effet le temps que nous pouvons accorder pour ce projet est relativement court.

1.2. Diagramme de Gantt

Ce projet a été réalisé sur une période de 4 semaines réparties entre le 11 avril 2023 et le 9 juin 2023, en suivant la méthodologie du diagramme de Gantt présentée ci-dessous.

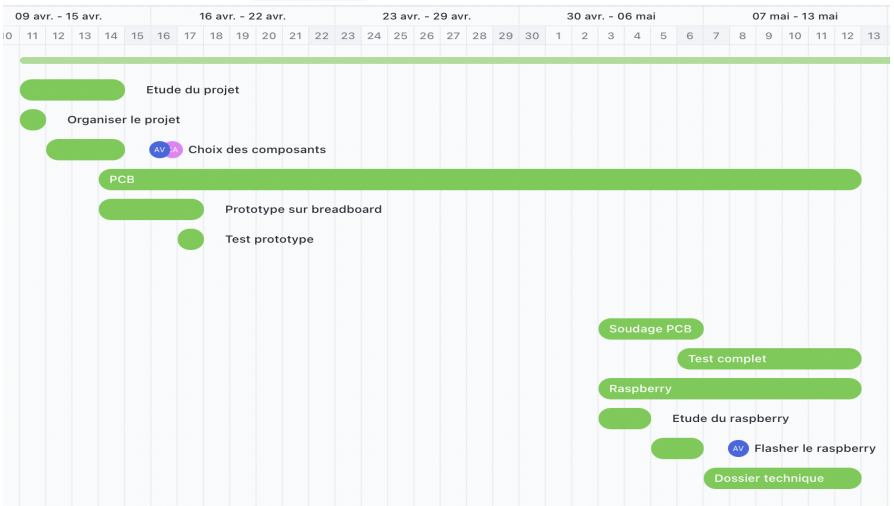


Diagramme de Gantt du projet SAE.

1.3. Les livrables

Lors de cette SAE, nous avons segmenté le projet et à la suite de chaque étape, nous avions un livrable. Ce livrable, nous permettait de justifier le bon fonctionnement d'une étape. Pour commencer, dans le but de comparer et de choisir le meilleur composant, nous avons réalisé des tableaux comparatifs entre différents modèles d'un même composant.

Ensuite nous sommes passés à l'étape de la réalisation de la carte. Pour cela nous avons découpé cette étape en deux : une première partie où nous avons réalisé un prototype sur breadboard afin de vérifier le bon fonctionnement de notre câblage, et ce breadboard fût un livrable à part entière. Puis une deuxième partie correspondant à la réalisation du PCB et au soudage de la carte. Cette partie fût accompagnée d'une fiche de test afin de justifier le bon fonctionnement.

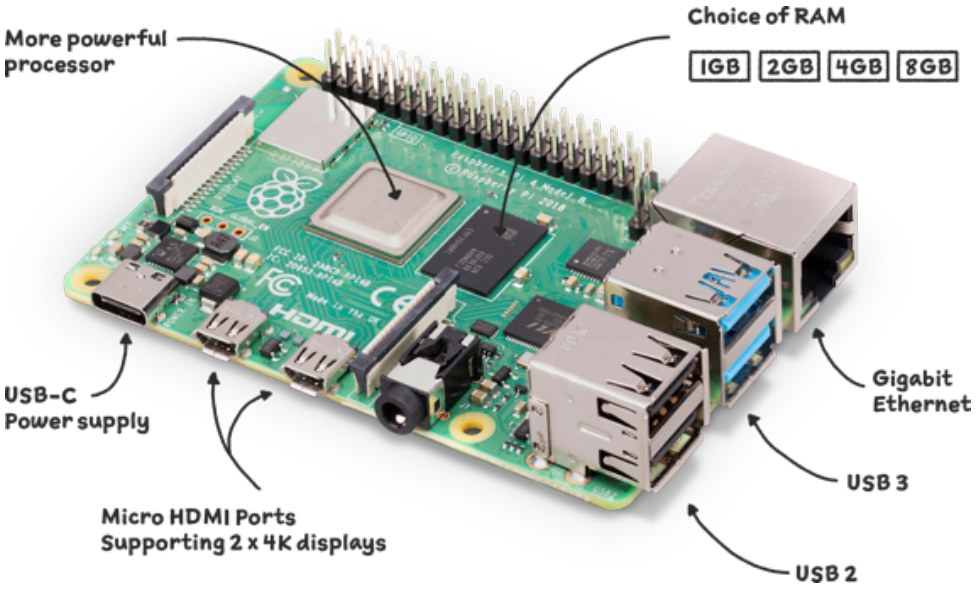
Pour finir, le dernier livrable était le code de test afin de tester le bon fonctionnement de la carte et de la librairie créée pour cette carte. Ce code de test a permis de montrer les différentes nuances entre les fonctions ainsi que leurs utilités.

2. Choix des composants

Pour réaliser ce projet nous avons donc commencé par étudier et choisir les composants qui allaient constituer notre shield. Notre shield devait être constitué d'un capteur de température et d'humidité, de deux boutons poussoirs, deux LEDs, et d'un encodeur. Lorsque nous avons débuté notre projet deux composants nous ont été imposés : la Raspberry Pi et le capteur de température.

2.1. La Raspberry Pi

Le cœur de notre projet est la Raspberry Pi qui est connecté à notre shield et gère le programme. Une Raspberry Pi est un micro ordinateur qui fonctionne sous Linux, qui est un peu plus grand qu'une carte bancaire et qui possède plusieurs connecteurs (USB, Ethernet, micro-hdmi…). Elle possède aussi plusieurs ports GPIO que nous utiliserons dans ce projet.
Le modèle de Raspberry Pi que nous avons utilisé pour ce projet est la Raspberry Pi 4 model B.



Raspberrry Pi 4 Model B

2.2. Le capteur de température

Le deuxième composant imposé était un capteur de température et d'humidité le SHT45. Pour comprendre pourquoi nous avons utilisé ce capteur nous avons comparé celui-ci avec le capteur DHT22, un autre capteur de température et d'humidité que nous avons déjà utilisé et qui est connu des bricoleurs arduino.

	DHT22	SHT45-AD1B-R2
Précision (°C)	+/-0.5 °C	+/-0.1°C
Précision d'humidité	+/-2%	+/- 1%
Alimentation	5 V	3.3 V
Consommation	1.5mA/ 50µA	0.4µA
Communication	One Wire	I2C
Taille	Small size : 14*18*5.5 mm	1.5*1.5*0.5mm
Plage de température	-40/+80°C	-40°C à 125°C

Nous pouvons voir qu'il y a plusieurs avantages à utiliser le capteur SHT45. Tout d'abord, ce capteur est plus précis que le DHT22 que ce soit pour la température ou pour l'humidité. Ensuite, l'alimentation du capteur correspond à ce que le Raspberry Pi peut délivrer. De plus, la consommation et la plage de température du SHT45 sont plus intéressantes que le DHT22. Enfin, le SHT45 communique en I2C ce qui est un avantage comparé au DHT22 qui communique en one wire c'est-à-dire que la réception et la transmission se font sur un seul et même fils.

Mais même s' il ne semble présenter que des avantages, il y a néanmoins un inconvénient à choisir le capteur SHT45. Nous pouvons voir que sa taille est très inférieure à celle du DHT22. De plus, il est en CMS ce qui en fait un composant difficile à souder.

Capteur SHT45

2.3. L’encodeur

Pour répondre à notre cahier des charges nous devions rajouter un composant pouvant être manipulé par une personne et ayant plusieurs positions. Notre premier choix se portait donc sur un potentiomètre, l'inconvénient est que le potentiomètre nous retourne une tension analogique ce qui ne peut pas être traité par les GPIO de la Raspberry Pi. Nous nous sommes alors dirigés vers un autre type de composant, l’encodeur, qui lui délivre une tension numérique traitable par les GPIO.

Nous avons donc hésité entre deux modèles :

	PAC18R1-43D19F	26GS22-01-1-16S-C
Positions	16	16
Crans	Oui	Oui
Type de sortie	Code de gray	Code de gray
Durée de vie	30000 Cycles	25000 Cycles

Comme nous pouvons le voir, il n’y a pas beaucoup de différences entre ces deux modèles. Mais nous avons choisi l’encodeur PAC18R1-43D19F car il à une durée de vie plus élevée et il a un coût inférieur à l’autre modèle. Ce modèle compte cinq pins de sortie dont quatre qui vont correspondre aux données et un relié à la masse.

Ces quatre pins vont transmettre un code de gray à leur sortie correspondant à ce tableau :

Pin Out Code 16 Positions

Position	Angle in °	P1	P2	P3	P4
1	0	0	0	0	0
2	22.5	1	0	0	0
3	45	1	1	0	0
4	67.5	0	1	0	0
5	90	0	1	1	0
6	112.5	1	1	1	0
7	135	1	0	1	0
8	157.5	0	0	1	0
9	180	0	0	1	1
10	202.5	1	0	1	1
11	225	1	1	1	1
12	247.5	0	1	1	1
13	270	0	1	0	1
14	292.5	1	1	0	1
15	315	1	0	0	1
16	337.5	0	0	0	1

Pour faciliter sa lecture nous convertirons dans le programme le code de gray en code binaire, nous y reviendrons plus tard.

PAC18R1-43D19F

Nous avons également hésité avec un autre modèle, le SRGAVQ1100, avec plus de positions. Nous n'avons pas pris celui-ci car son arbre est intérieur ce qui fait de lui un encodeur moins maniable ce qui ne répond pas à notre cahier des charges.

SRGAVQ1100

2.4. Les boutons poussoir et les LEDs

Enfin, notre carte est également dotée de deux LEDs et de deux boutons poussoirs. Pour ces composants nous avons fait au plus simple et nous avons pris ce que l'IUT avait à disposition.
Pour les boutons poussoirs nous avons choisi le modèle : D6C50F1LFS. Un rouge et un vert. Nous avons choisi ce modèle car il est voyant et plus maniable que les boutons poussoir des arduino les 1825910-6.
.

1825910-6

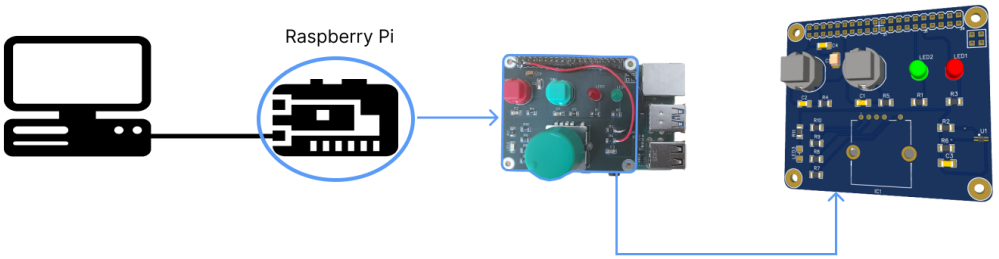
D6C50F1LFS

Pour les LEDs nous avons choisi des 5V de 5mm, une rouge et une verte pour correspondre aux boutons poussoir. De plus, nous avons choisi des LEDs traversantes pour qu'elles soient plus visibles sur notre carte.

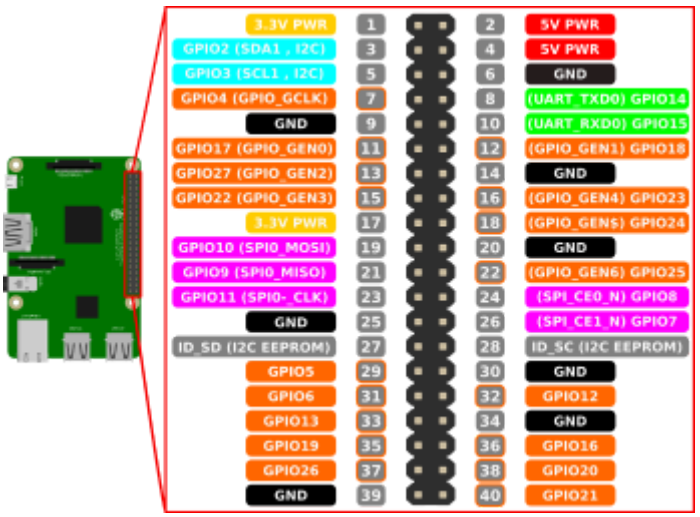
3. Réalisation

3.1. Schémas général

Pour comprendre un peu plus la structure de notre projet, expliquons notre schéma général.



Le Raspberry Pi de notre projet est connecté à un ordinateur qui a le même réseau que lui en filaire ou en ssh. Ensuite sur notre Raspberry présentée plus tôt, le shield réalisé par nos soins se connectera sur ses GPIO.



GPIO Raspberry PI 4 model B

Nous reviendrons sur les différents GPIO utilisés plus tard. Pour utiliser notre module un programme à été réalisé dans l'IDE VScode.
Le but de ce dispositif est de programmer les différents composants du shield présenté plus tôt afin de récupérer leurs états sur l'ordinateur.

3.2. Le software

Concernant la partie software, il y a une première partie connexion au raspberry pi et une autre partie programmation. Cependant avant tout cela, nous avons dû flasher la raspberry pi via raspberry pi imager.

Premièrement, nous avons dû se connecter au raspberry pi. Pour cela, nous avons branché le raspberry sur le même réseau que l'ordinateur que nous avons utilisé pour programmer. Une fois sur le même réseau, nous avons utilisé VS Code afin de se connecter en ssh à l'appareil. Pour cela nous avons utilisé la commande : *ping raspberrypi.local* sur un terminal afin d'obtenir l'adresse IP. Une fois l'adresse IP obtenue, nous nous y sommes connecté sur VS Code en utilisant Connect to Host. Il faut ensuite entrer le nom du raspberrypi@son adresse IP. On vous demandera son mot de passe ; le nôtre est azerty. Tous ces paramètres sont configurés lors du flashage de la carte SD du raspberry.

Deuxièmement, nous avons dû réaliser une librairie en python. Cette librairie comprend plusieurs fonctions. Ces fonctions utilisent 6 librairies :

```
import RPi.GPIO as GPIO
import time
import board
import adafruit_sht4x
import graycode
import smbus
```

Ensuite il y a une partie initialisation des GPIO et de l'I2C.

```
bus = smbus.SMBus(1)
# Initialisation de la numerotation et des E/S
GPIO.setmode(GPIO.BCM)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(18, GPIO.IN)
GPIO.setup(25, GPIO.IN)
GPIO.setup(23, GPIO.IN)
GPIO.setup(24, GPIO.IN)
GPIO.setup(5, GPIO.OUT, initial = GPIO.HIGH)
GPIO.setup(6, GPIO.OUT, initial = GPIO.HIGH)

i2c = board.I2C()   # uses board.SCL and board.SDA
sht = adafruit_sht4x.SHT4x(i2c)
sht.mode = adafruit_sht4x.Mode.NOHEAT_HIGHPRECISION
```

Une fois tout configuré, nous avons programmé les différentes fonctions. Pour commencer, la fonction getSensor est une fonction qui relève la température en degrés celsius et le taux d'humidité. Cette fonction utilise la librairie "sht" qui lit l'i2c et convertit les valeurs à la bonne unité.

```
def getSensor():
    """
   Read the temperature and the humidity level

   :param: none
   :return: the temperature and the humidity level
   """
    temperature, relative_humidity = sht.measurements
    return temperature, relative_humidity
```

Puis la fonction getButton est une fonction qui lit l'état du bouton. C'est une fonction assez simple qui retourne l'état du bouton.

```
def getButtonRed() :
    """
   Read red button status

   :param: none
   :return: the red button status
   """
    RButton = GPIO.input(17)
    return RButton


def getButtonGreen():
    """
   Read green button status

   :param: none
   :return: the green button status
   """
    GButton = GPIO.input(27)
    return GButton
```

Il y a ensuite les fonctions event qui sont des fonctions basées sur les poolings. Les poolings sont des interruptions gérées par le code et non par le matériel comme les interruptions. Cette fonction exécute une fonction que l'on lui entre en paramètre lorsqu'on appuie sur le bouton qui est associé à la fonction. Cette fonction va exécuter la fonction et mettre en pause le main principal.

```
def addEventButtonGreen(function) :
    """
   Execute the function when the green button is activited

   :param: a function
   :return: none
   """
    GPIO.add_event_detect(27, GPIO.RISING, callback=function, bouncetime = 1)


def addEventButtonRed(function):
    """
   Execute the function when the red button is activited

   :param: a function
   :return: none
   """

    GPIO.add_event_detect(17, GPIO.RISING, callback=function, bouncetime = 1)
```

Il y a également la fonction writeLed qui est une fonction qui écrit l'état à la led associée.

```
def writeLedRed(etat):
    """
   Write red led status

   :param: the state 0 or 1
   :return: the state 0 or 1
   """

    if etat == 1 :
        etatr =0
    else :
        etatr =1
    GPIO.output(5, etatr)

def writeLedGreen(etat):
    """
   Write green led status

   :param: the state 0 or 1
   :return: the state 0 or 1
   """
    if etat == 1 :
        etatr =0
    else :
        etatr =1
    GPIO.output(6, etatr)
```

Pour finir, la fonction getEncoder est une fonction relevant la valeur de la position du codeur. Le codeur étant en code de gray, nous devons traduire le code de gray. Cette traduction s'est faite grâce à la librairie "graycode" qui détient une fonction faisant la traduction.

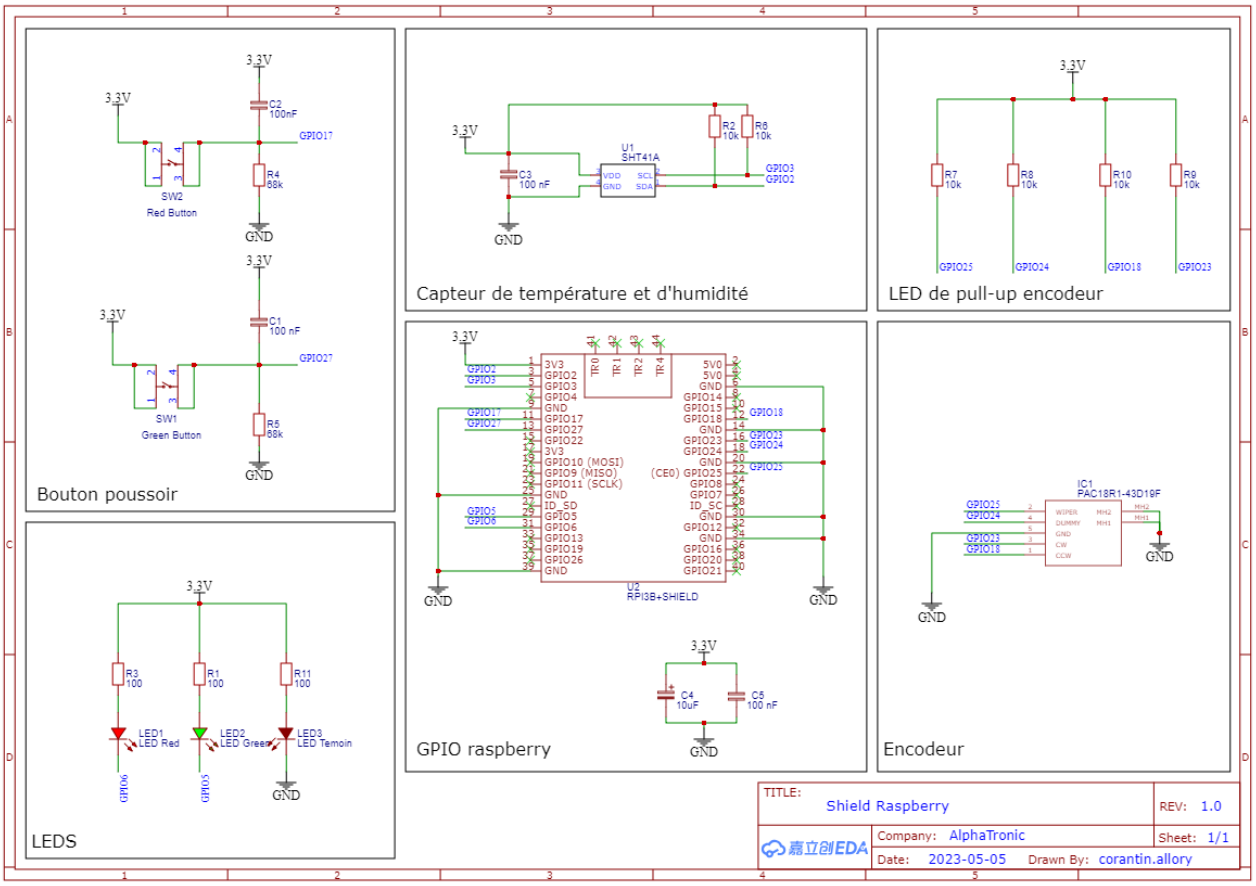
```
def getEncoder():
    """
   Read sensor value

   :param: none
   :return: the sensor value
   """
    valeurgray = (not(GPIO.input(18)))*1 + (not(GPIO.input(25)))*2 + (not(GPIO.input(23)))*4 + (not(GPIO.input(24)))*8
    valeurnor = graycode.gray_code_to_tc(valeurgray)+1
    return valeurnor
```

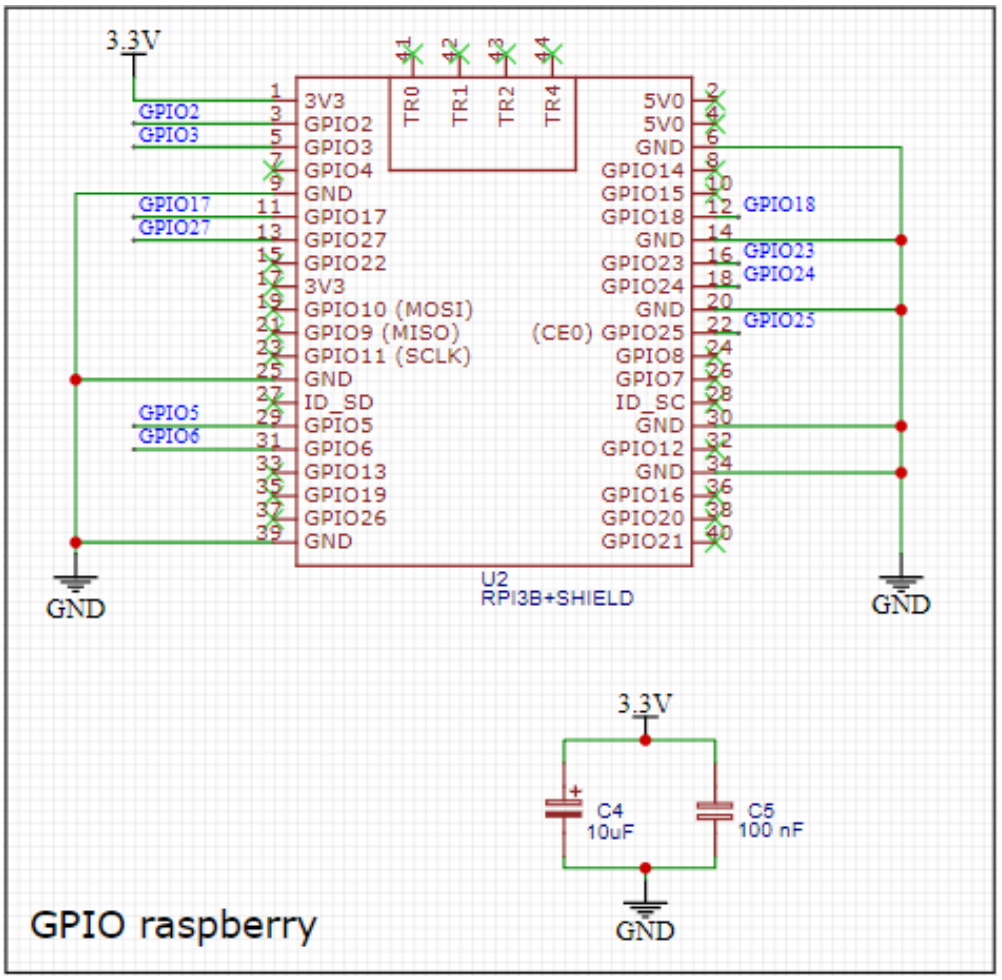
Les différentes fonctions ont été testées et présentées lors d'une démonstration. Ils sont donc fonctionnels.

3.3. Le hardware

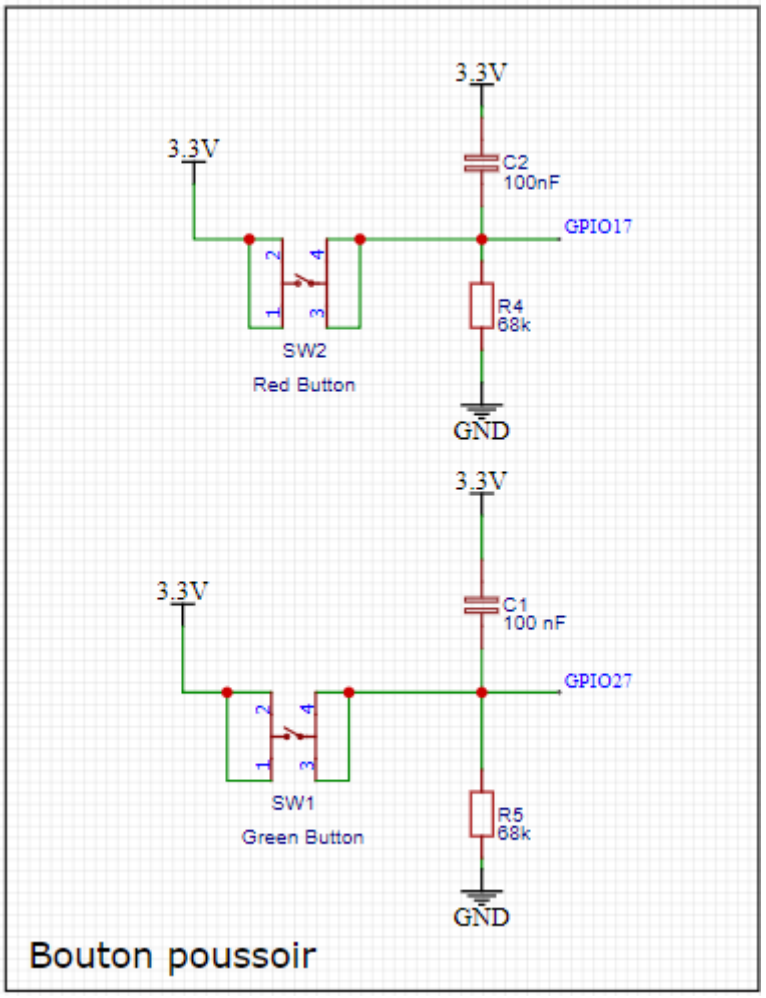
Concernant la partie hardware, la réalisation de la carte électronique à été effectuée sur le logiciel EasyEDA.



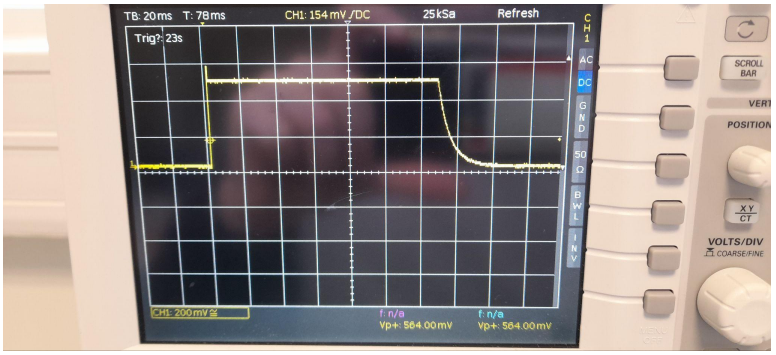
Le schéma se divise en cinq parties : il y a les LEDs, les boutons poussoir, le capteur de température et d'humidité, les différents GPIO du Raspberry et la partie de l'encodeur.



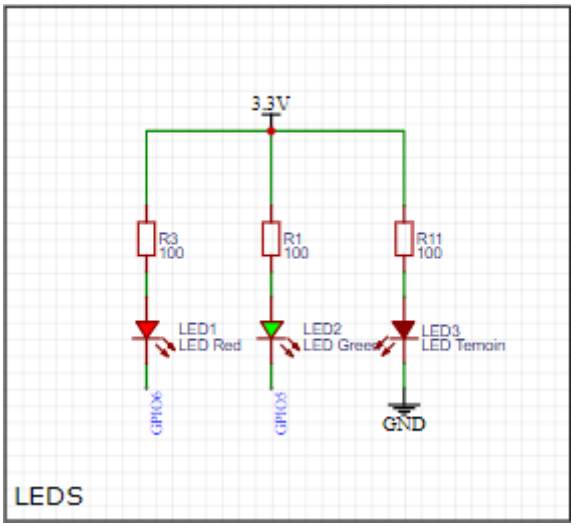
Voici les différents GPIO que nous utilisons pour le bon fonctionnement de notre carte. Nous utilisons le GPIO2 et le GPIO3 pour la communication I2C avec le capteur de température et d'humidité. Pour les autres composants nous utilisons les GPIO5, 6, 17, 27, 18, 23, 24 et 25. Au plus près de l'alimentation nous avons rajouté deux condensateur de découplage un de 100 nF et l'autre de 10µF.



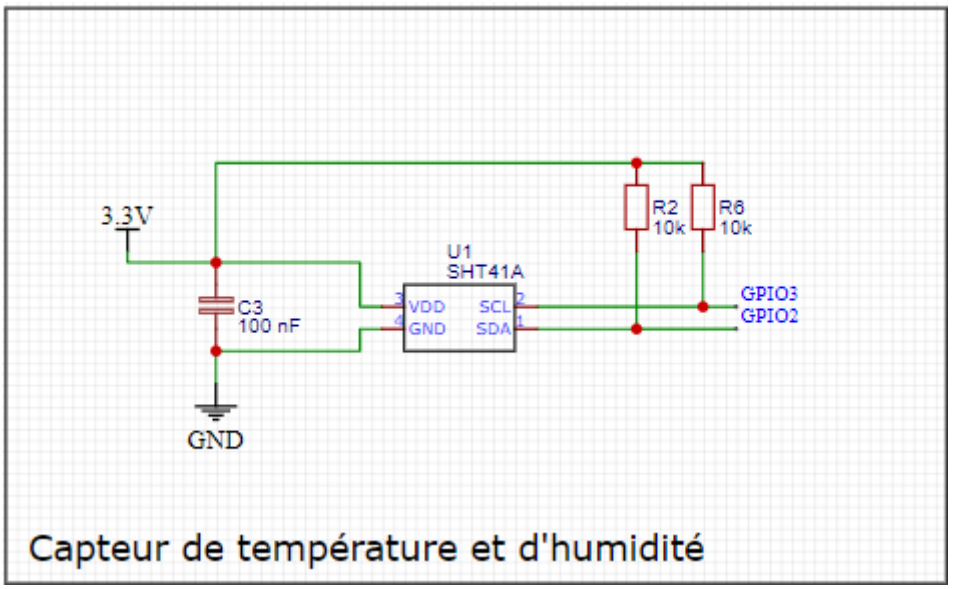
Les boutons poussoir sont reliés au GPIO17 et au GPIO27. Ils sont câblés à une résistance de pull-down de 68 kOhms pour fixer le bouton à l'état bas quand il n'est pas activé. Également, un condensateur de 100 nF à été mis en parallèle afin de réaliser un filtre anti rebonds pour éviter les états transitoires du bouton.
Pour tester le filtre anti rebonds nous avons alimenté la carte avec une alimentation de laboratoire et nous avons observé le signal à l'aide d'un oscilloscope.



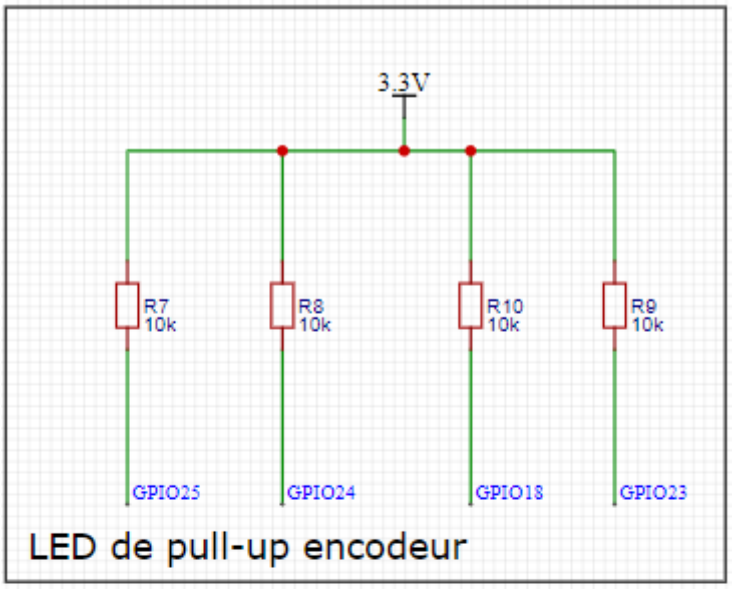
Nous pouvons voir sur l'oscilloscope que le filtre anti rebonds fonctionne bien en observant la décharge du condensateur à la fin du signal.

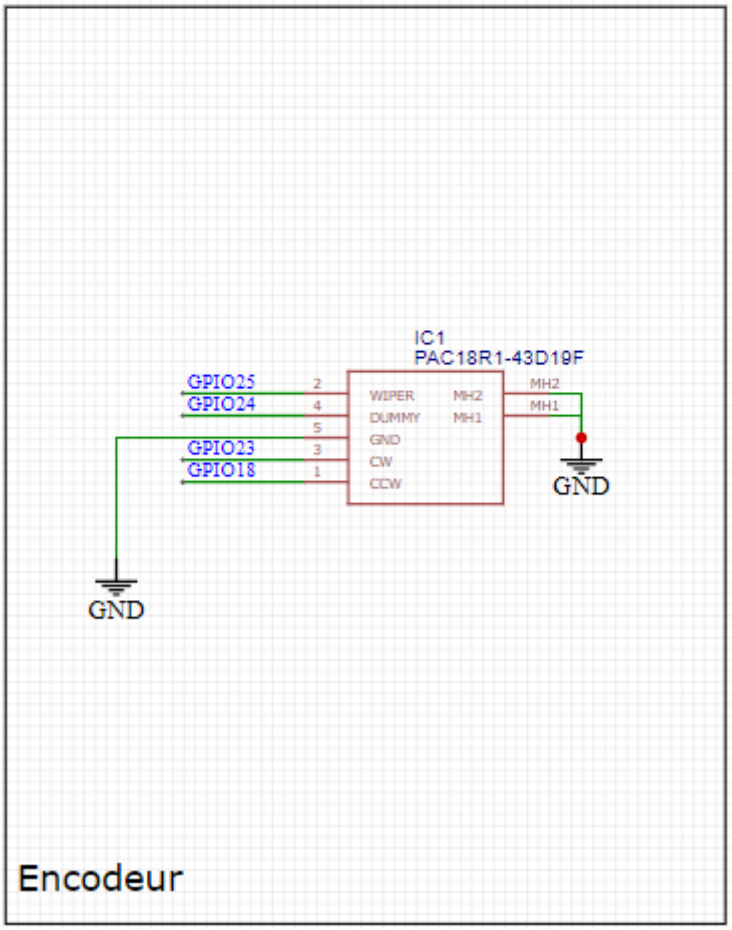


Les LEDs qui sont contrôlables par la Raspberry Pi sont les LEDs rouges et vertes qui sont reliées au GPIO5 et au GPIO6. La troisième LEDs nous sert à savoir si notre carte est bien alimentée par le Raspberry Pi, elle n'est pas obligatoire sur la version finale de la carte. Les trois LEDs sont reliées à une résistance de 100 Ohms qui sert à faire chuter la tension aux bornes des LEDs.
Pour tester le bon fonctionnement des deux LEDs programmable nous avons directement testé avec un programme de test.



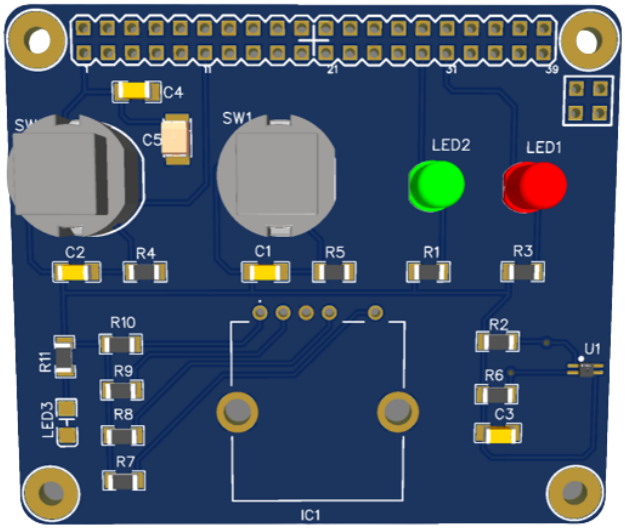
Le capteur de température et d'humidité est relié aux GPIO 2 et 3 qui sont ceux responsables de la communication en I2C sur le Raspberry Pi. Pour que le capteur fonctionne dans de bonnes conditions nous lui avons ajouté un condensateur de découplage de 100 nF et deux résistances de de pull-up de 10kOhms qui sont nécessaires pour que l'I2C fonctionne correctement.
Comme pour les LEDs, les tests ont été effectués directement grâce à un programme.

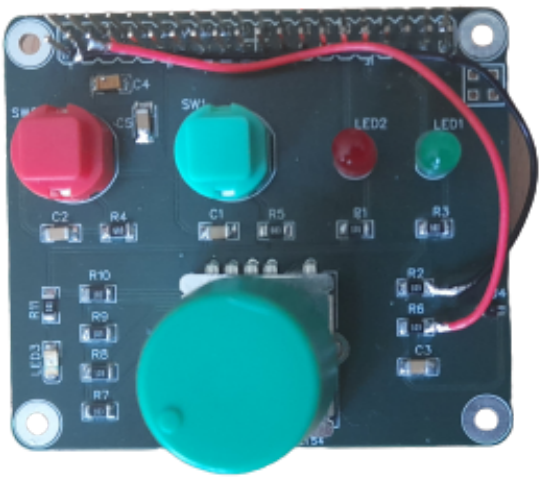




Pour l'encodeur, les différents GPIO utilisés sont les 18, 23, 24 et 25. Nous avons également relié les différents pins de l'encodeur à des résistances de pull-up pour fixer leurs états lorsqu'ils ne sont pas sélectionnés par l'utilisateur.
Pour les tests de l'encodeur nous avons fait une première partie où nous regardions à l'oscilloscope si les pins s'activaient quand nous tournions l'arbre et une deuxième partie avec un programme test.

Une fois tous ses composants routés nous avons obtenus la carte suivante :





Nous avons pu la tester avec la Raspberry Pi et les différentes fonctions réalisées pour les différents composants et elle fonctionne.
Pour la réalisation de cette carte nous avons rencontré quelques difficultés. La première à était l'oublie de connecter le plan de masse à la masse, la deuxième à été de connecter l'alimentation au 5v plutôt qu'au 3.3 V ce qui à cramé une Raspberry Pi. La dernière à été durant la réalisation du PCB chez le sous traitant car tous les fichiers de perçage ne contenaient pas toutes les mêmes informations.

Conclusion et Perspectives

Ainsi, à la fin de notre projet, nous pouvons conclure que celui-ci fonctionne. Après quelques recherches et des problèmes rencontrés au niveau hardware notamment, nous avons finalement réussi à concevoir un shield qui se connecte sur les GPIO de la Raspberry Pi contenant un capteur de température et d'humidité, un encodeur, deux boutons poussoirs et 2 LEDs. De plus, grâce au software qui à été développé nous arrivons à récupérer différentes informations, tels que la position de l'encodeur, la température, l'humidité, l'état des boutons et des LEDs. Cependant, même si notre projet fonctionne, il y a toujours la possibilité de l'améliorer pour qu'il initie aux objets connectés de manières optimales.

Afin d'améliorer notre projet, plusieurs solutions sont envisageables. Pour que notre Raspberry Pi se rapproche le plus possible d'un objet connecté, il pourrait être envisageable de connecter un écran à la Raspberry afin d'afficher différentes informations (température, humidité, position encodeur…). De plus, un buzzer ou un haut parleur pourrait être connecté à la Raspberry pour alerter lorsqu'il y a un problème ou écouter de la musique.

Bibliographie

Lors de ce projet, nous avons utilisé plusieurs sources d'information. Ces informations nous ont été utiles pour la réalisation du projet (présentations et rapports).

Le Monde :
-https://www.lemonde.fr/blog/binaire/tag/internet-des-objets/

Le blog de lulu :
-https://lucidar.me/fr/

Gammatroniques :
-https://gammatroniques.fr/

GitHub :
-https://github.com

Raspberry pi :
-https://www.raspberrypi.org/courses/learn-python

Stack Overflow :
-https://stackoverflow.com/questions/70853452/getting-an-event-on-input

Résumé

Envie de vous initier au monde des objets connectés ? Ces objets dont tout le monde parle, qui sont de plus en plus présents dans notre société dans différents domaines, que ce soit dans l'industrie ou la santé en passant par notre vie quotidienne. Dans ce rapport vous trouverez l'essentiel pour réaliser un shield qui se connecte sur une Raspberry Pi afin de simuler un objet connecté. Le shield comprend plusieurs composants que l'on peut retrouver sur les objets connectés comme des boutons, des leds ou un capteur de température. Vous trouverez également la partie software de ce projet afin de comprendre comment les objets connectés fonctionnent. La partie hardware a été développée sur le logiciel Easy EDA et la partie software a été développée dans l'IDE VScode. Ce projet a demandé du temps de documentation sur la compréhension des différents composants afin de recréer le plus fidèlement possible un objet connecté. A ce projet, des améliorations peuvent être apportées par chacun si cela lui semble nécessaire.

Do you want to learn about the world of connected objects? These objects that everyone is talking about, which are more and more present in our society in different fields, whether in industry or health through our everyday life. In this report you will find the essentials for making a shield that connects to a Raspberry Pi in order to simulate a connected object. The shield includes several components that can be found on connected objects such as buttons, LEDs or temperature sensor. You will also find the software part of this project to understand how connected objects work. The hardware part was developed on the Easy EDA software and the software part was developed in the VScode IDE. This project required documentation time for the comprehension of the various components in order to recreate a connected object as faithfully as possible. In this project, improvements can be made by everyone if it seems necessary.