Compte Rendu de Projet :

Table des matières

[I. Introduction : 2](#_Toc39360491)

[II. Réalisation du projet : 3](#_Toc39360492)

[a) Rappel de la tâche de l’étudiant 3](#_Toc39360493)

[b) Communication entre les différents éléments 4](#_Toc39360494)

[c) Communication 5](#_Toc39360495)

[d) Chronogrammes 6](#_Toc39360496)

[e) Système socket 7](#_Toc39360497)

[f) Code du client 7](#_Toc39360498)

[g) Code du serveur 8](#_Toc39360499)

[III. Réalisation de l’application : 8](#_Toc39360500)

[h) Récupération de l’UID 8](#_Toc39360501)

[i) Allumage des LEDs et déclanchement du module relais 9](#_Toc39360502)

# Introduction :

#### Rappel du cahier des charges :

Afin que la partie se déroule correctement un système pour les « médaillons » est à fournir.

Ce système sera au cœur même de la salle de l’escape game et devra prendre en charges les demandes suivantes :

* Le lecteur RFID doit pouvoir lire un UID (*User Identifier*) à travers une plaque de contreplaqué de 3mm.
* Utiliser le bus SPI (*Serial Peripheral Interface*) de la Raspberry pour le contrôle des lecteurs RFID.
* Le superviseur doit pouvoir récupérer UID (*User Identifier*) afin de le changer facilement pour le programme principal dans le cas d’une perte d’un des médaillons.
* Le joueur doit pouvoir constater l’allumage ou non d’une LED dans la seconde en fonction de l’état d’un lecteur RFID (si un badge ou « médaillon » valide est posé dessus ou non).
* Dans le cas où le joueur place correctement quatre badges la gâche électrique doit s’activé permettant l’ouverture de la porte. Un ordre de fin de partie doit également être envoyer via un système de client socket.
* L’application de supervisions doit pouvoir détecter l’ordre de fin de partie via un système serveur socket.

# Réalisation du projet :

## Rappel de la tâche de l’étudiant

Au sein de ce projet d’escape game, ma partie consiste à créer un système permettant :

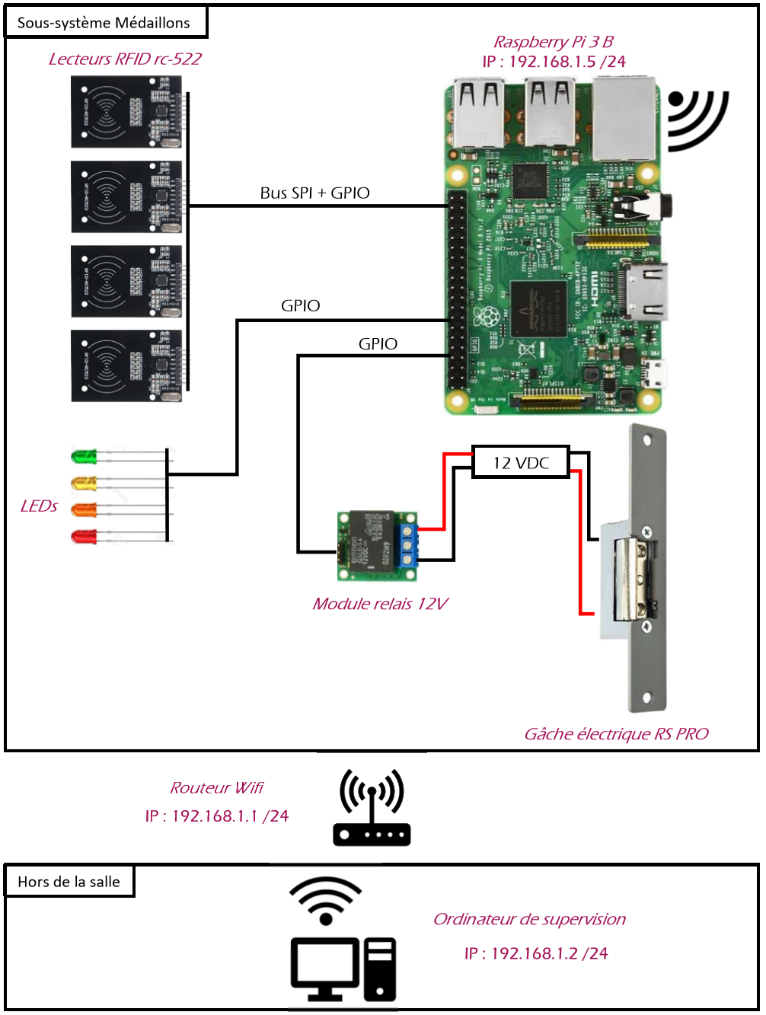
* + La détections des « médaillons ».
  + L’allumage des différentes LED.
  + L’ouverture de la porte.
  + La communication entre système « médaillons » et l’application de supervision

Le langage python est utilisé avec différentes librairies, comme « pi-RC522 » qui permet la gestion des lecteurs RFID, « socket » qui permet l’utilisation des services socket. Comme IDE j’ai choisi « Thonny » car très simple d’utilisation, ce choix m’était libre.

Le système doit être développé sur une carte Raspberry sous Raspbian. Le contrôle du superviseur de cette dernière sera possible via l’application « Anydesk » qui est compatible avec les deux environnement, Raspbian et Windows. Elle permet le contrôle à distance via un protocole TCP.

## Communication entre les différents éléments

*Synoptique*



L’ordinateur de supervision, le routeur wifi ainsi que Raspberry sont adressés en IP statiques.

## Communication

Le bus SPI (Serial Peripheral Interface) est un bus de données série synchrone en full-duplex. Il fonctionne sous un schéma maître-esclave. Dans ce cas le maître est la Raspberry et les esclaves sont les lecteurs RFID rc-522.

Ce modèle possède 8 interfaces dont 4 qui sont dédiés au bus SPI :

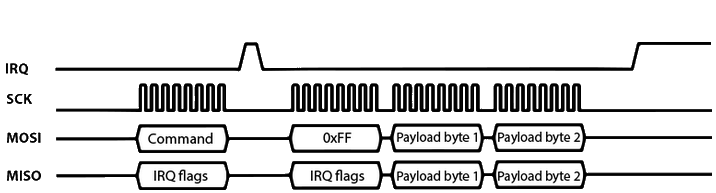
* SCK – Consacrer à l’horloge et généré par le maître.
* MOSI – Master Output, Slave Input (généré par le maître).
* MISO – Master Input, Slave Output (généré par l’esclave).
* NSS – La sélection de l’esclave (généré par l’esclave).

Les 4 interfaces restante sont :

* VCC – Alimente le module avec un voltage de 3,3 Volt.
* GND – La terre.
* RST – Permet la réinitialisation et la mise hors tension du module.
* IRQ – Permet une interruption, cela alerte le module quand un tag RFID se trouve à proximité

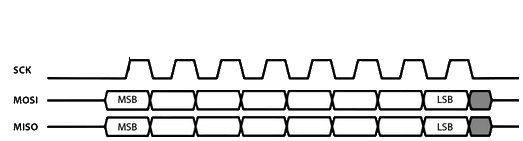
## Chronogrammes

Dans le premier chronogramme on voit que quand un badge passe à proximité donc quand l’IRQ signal le passage d’un tag RFID. La première transaction qui se passe avant le signal de l’IRQ, c’est l’octet (8 bites) de commande. Dans la second transaction 24 bits son transmis, les 16 derniers bits transmis sont l’UID (User Identifier) qui est propre à chaque taf RFID.



Chronogramme

Dans le second chronogramme on voit comment se passe la transaction d’un octet, le bite de poids fort (MSB) est transmis en premier.



Chronogramme

## Système socket

Pour envoyer l’ordre de fin partie un système de socket a été choisi avec comme « client » l’application du sous-système médaillons et comme serveur l’application de supervision. Ainsi le serveur sera en « écoute » en attendant que le client envoie un message qui confirmera la réussite de l’équipe qui participera à la session de jeu de l’escape game.

Pour ce faire les informations suivantes sont requises :

Pour le client :

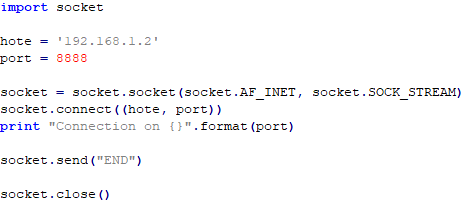
* Adresse IP : **192.168.1.2** (adresse IP l’ordinateur de supervision)
* Port : **8888**

Pour le serveur :

* Adresse IP : **192.168.1.5** (adresse IP de la Raspberry)
* Port : **8888**

## Code du client

Afin que l’application puisse envoyer l’ordre de fin de partie qui un simple message « END », il faut utiliser le code suivant en python :



La méthode *import* d’accéder à différents modules, dans ce cas on importe le module socket.

Par la suite on définit l’hote et le port de communication. On crée ensuite le socket avec la méthode *socket(socket.AF\_INET, socket.SOCK\_STREAM)* puis on envoie un requête de connexion au serveur avec la méthode *socket.connect()*. Si la connexion n’est pas rejeté on envoie le message de fin avec la méthode *socket.send()* puis en ferme le socket avec *socket.close()*.

## Code du serveur

Encore à étudier

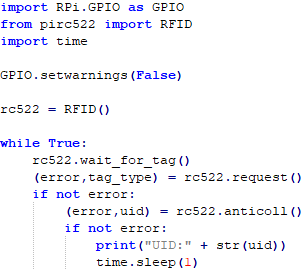
# Réalisation de l’application :

Afin que la partie la partie se déroule, il est nécessaire d’avoir une application qui permettra de détecter les médaillons, allumer les LEDs, déclencher le module relais et enfin envoyer l’ordre de fin partie comme si les médaillons sont bien placer.

## Récupération de l’UID

En premier lieu il faut pour le superviseur pouvoir récupérer l’UID d’un badge RFID dans le cas d’une perte et/ou vol.

Pour ce faire le code suivant est utilisé.

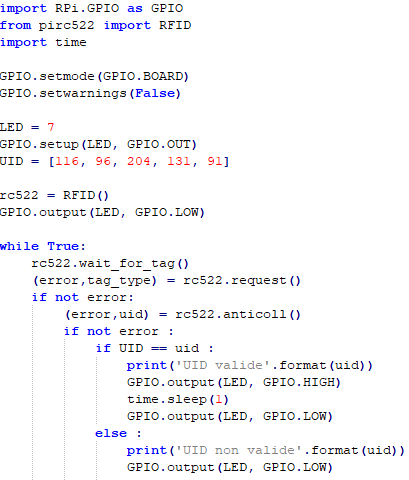


Dans un premier temps importe le module RFID depuis la librairie « pi-rc522 » et on instancie la librairie avec *rc522 = RFID()*.

Ensuite on crée une boucle infinie pour lire en boucle. Dans cette boucle on utilise la fonction *wait\_for\_tag()* pour attendre qu’un badge RFID passe à portée de lecture. Quand un badge passe à proximité on récupère son UID avec *request()* et on nettoie les possibles collisions avec *anticoll()*, ça arrive si plusieurs badge passent à porter en même temps. Puis un on affiche UID unique du badge avec str(). Pour la fin on attend 1 pour ne pas lire le badge des centaines de fois en quelques milli-secondes avec *sleep(1)*, 1 étant 1 seconde.

## Allumage des LEDs et déclanchement du module relais

L’allumage de la LED doit pouvoir se faire dans la seconde en fonction de l’état du capteur donc si un badge RFID est proximité ou non. Pour cela le code suivant est utilisé.



Une fois qu’un UID a été récupérer on peut l’utiliser pour faire un comparatif avec l’UID récupérer par le lecteur RFID et ainsi allumer la LED. Pour ce faire une défini le numéro du port GPIO qui va contrôler la LED, ici port n°7 et on active le contrôle du GPIO avec *setup(LED,GPIO.OUT)* puis on initialise ce port à l’état bas (LED éteinte) avec *output(LED,GPIO.LOW)*. Ensuite on reprend la boucle d’écoute décrit précédemment et on fait la comparaison des deux UID. Si l’UID est valide on allume la LED avec *output(LED,GPIO.HIGH)* puis on attend 1 seconde et on éteint la LED. Si le badge reste à porter du lecteur RFID la LED restera allumer car la boucle est infinie.

Pour le déclanchement du module relais on utilise sensiblement la même démarche.