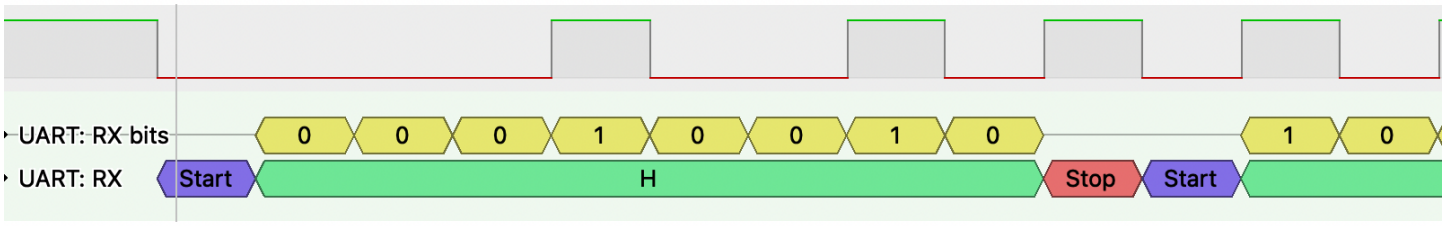
Les protocoles série

1 UART

On le voit dans Wokwi, lorsqu'on lance un circuit où le microcontrolleur se programme à l'aide de micropython, une console s'ouvre qui permet de dialoguer avec le microcontrolleur. Le fonctionnement usuel (sans émulation donc) d'un microcontrolleur qui tourne sous micropython en est très proche, à ceci près que la console nécessite quelquefois une configuration, car les échanges entre le microcontrolleur et l'ordinateur utilisent le protocole série (UART : Universal Asynchronous Receiver Transmitter). En pratique : un fil transmet l'information du microcontrolleur à l'ordinateur, et un fil transmet l'information de l'ordinateur au microcontrolleur.

Au repos, le transmetteur applique un signal haut sur la broche émettrice. Dès qu'un caractère doit être émis (supposons qu'il s'agisse ici d'un caractère formé de 8 bits, pour 256 valeurs différentes), le signal devient bas pendant une unité de temps, puis pendant 8 unités de temps un signal haut ou bas est appliqué selon le bit à transmettre, enfin la transmission s'achève par un signal haut pendant encore une unité de temps. Si un deuxième caractère doit être transmis, on recommence avec un signal bas (bit de départ), puis 8 signaux haut ou bas et un bit de terminaison, et ainsi de suite jusqu'à revenir à un signal constant haut.



Dans l'exemple ci-dessus, un premier caractère est émis, 'H' qui est codé (codage ascii) par l'entier 72 dont l'écriture binaire est 1001000 (on voit donc ici que le chiffre le moins significatif est émis le premier)

On comprend avec ce mode de communication que le microcontrolleur et l'ordinateur ne peuvent se comprendre que s'ils se sont mis d'accord sur :

- la vitesse de transmission, laquelle se mesure en bits par seconde, ou bauds par seconde (quelquefois abrégé en baud).
- Le nombre de bits de chaque donnée à transférer, sinon on confondrait les bits de terminaison et de départ avec les bits de l'information à transmettre

Exercice 1. Se connecter à Wokwi, et créer un nouveau projet en partant d'un microcontrolleur ESP32 tournant sous micropython. Un petit script avec un `print("Hello, ESP32!")` est déjà programmé. Inutile de changer quoi que ce soit.

Rajouter un analyseur logique (cliquer sur le + violet et choisir celui-ci)

Relier la masse (GND) de l'ESP32 à la masse de l'analyseur logique

Relier la broche TX0 à l'une des entrées de l'analyseur logique (D0 par exemple)

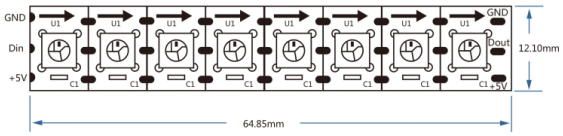
Lancer la simulation, puis arrêter aussitôt ou presque celle-ci, un fichier d'extension vcd devrait alors être téléchargé.

Pour analyser le fichier obtenu, la documentation wokwi (https://docs.wokwi.com/guides/logic-analyzer) nous invite à installer PulseView (ou GTKWave, mais je n'ai pas testé). Suivre les indications de la page ci-dessus, et ouvrir le fichier vcd obtenu. (Il est recommandé de choisir un facteur de sous-échantillonnage de 50 (downsampling factor) et c'est en effet préférable, sinon PulseView peut avoir besoin de beaucoup (vraiment beaucoup) de mémoire et de temps de calcul.

Cliquer sur la petite icone en haut à droite (decoder selector) et choisir UART, puis associer RX (ou TX, peu importe à vrai dire ici) à l'entrée choisie de l'analyseur logique. Vous pouvez laisser les choix par défaut (115200bds, 8bits sans bit de parité etc…), à l'exception peut-être du format de décodage qui sera plus clair en choisissant l'option Ascii que le Hex (hexadécimal) par défaut.

2 Neopixels

Les néopixels (nom technique WS2812b) sont des leds RGB le plus souvent reliées les unes aux autres et qu'un protocole assez précis permet de contrôler.



PIN	Function
+5V	Power supply LED (4-7V works)
Dout	Control data signal output
GND	Ground
Din	Control data signal input

Trois cables suffisent à commander une bande de néopixels : deux pour l'alimentation électrique, et un pour les données de colorimétrie à faire passer. A priori, chaque néopixel reçoit 24 bits d'information (8 par composante primaire : rouge, vert, bleu), puis si d'autres bits d'information suivent, ils sont transférés au suivant. A la fin de la transmission (comme au début d'ailleurs), le niveau du canal de donnée est bas.

Exercice 2. Créer un projet Wokwi avec : un ESP32, un anneau de 16 néopixels (ou pas d'ailleurs, puisque ce qu'on va analyser, c'est le signal émis par le microcontrolleur, donc si vous êtes paresseux, nul besoin d'ajouter quoi que ce soit d'autre que l'analyseur logique) et connecter celui-ci à une broche de sortie du ESP32, la masse et la broche à une source Vcc (rien n'interdit avec Wokwi de le relier directement au microcontrolleur, mais autant prendre de bonnes habitudes)

1. Rajouter un peu de code dans le script à gauche pour importer la bibliothèque neopixel, définir une bande de néopixels et la broche par laquelle le signal doit être transmis. Vous pouvez prendre exemple sur :

```
import neopixel
from machine import Pin
neo = neopixel.NeoPixel(Pin(5),16)

for i in range(16):
    neo[i] = (16*i, 16*i+1,16*i+2)
neo.write()
```

2. Rajouter un analyseur logique, connecter la masse et une des broches d'entrée à la broche de l'ESP32 choisie pour le signal. Lancer la simulation et l'arrêter presque aussitôt. Ouvrir le fichier .vcd obtenu à l'aide encore de PulseView selon le même schéma que l'exercice précédent. Vous devriez trouver le signal transmis par le microcontrolleur sur une petite durée de temps, zoomer sur celui-ci et comparer au signal UART de l'exercice précédent (Une première remarque évidente, et comme indiqué ci-dessus, le signal est bas au contraire de l'UART, mais ce n'est pas la seule différence). Conjecturer le protocole pour le transfert d'un bit.

3. Vérifier votre conjecture à l'aide du décodeur adapté de PulseView (cherchez du côté de RGB LED WS2812…)

3 Quelques autres protocoles importants

Nous ne rentrerons pas dans les détails de tous les protocoles que l'on peut rencontrer avec les microcontrolleurs, mais je signale pour les plus curieux :

- le protocole I2C très utilisé par nombre de capteurs et d'écrans (la différence essentielle avec les précédents est d'une part que plusieurs dispositifs peuvent être reliés, jusqu'à un maximum théorique de 128 équipements car chacun doit disposer d'une adresse propre, codée sur 7 bits, et que outre la masse, deux lignes sont nécessaires pour échanger des informations : l'une est un signal d'horloge et sur l'autre transitent les données)

- les liaisons SPI : ressemble un peu au protocole I2C mais avec deux canaux pour les données, pour une communication simultanée dans les deux sens (si nécessaire) là où dans le cas de I2C, il n'y a jamais qu'un seul dispositif qui peut à un moment donné transmettre des données

- I2S : protocole utilisé pour le transfert de données audio