Thème n°5: WiFi et web. Mise en oeuvre du SoC esp8266

CELECT7 : Applications des microcontrôleurs

M. Zwingelstein

rev. 2020

Travail préparatoire

Lectures de la partie péparatoire :

1. Lecture sur le modèle TCP/IP : [ici]
2. Lecture sur l'adressage IP : [ici]
3. Lectures sur le WiFi : à rechercher vous même pour pouvoir répondre aux questions

---

Questionnaire de la partie préparatoire :

Modèle TCP/IP

1. Associer chaque proposition ci-dessous à la couche TCP/IP qui lui correspond (couches n°1 à n°4) :

— protocole HTTP
— protocole DHCP
— équipement routeur
— adresse MAC
— adresse IP
— S'assurer que les paquets arrivent dans le bon ordre

Adressage IP

2. Comment s'appelle le protocole qui permet d'attribuer de façon dynamique une adresse IP à chaque machine du réseau ?
3. Quel est la valeur du masque de réseau du réseau IP identifié en notation CIDR par : 192.168.1.0/24 ?

   Une machine de ce réseau peut-elle avoir l'adresse 192.168.1.255 (justifier) ?
4. Un réseau IP est identifié par son adresse réseau (192.168.5.0) et par son masque de sous-réseau (255.255.255.248).
   — Combien de machines peut-il y avoir sur ce réseau ?
   — Quelles seront leurs adresses IP ?
   — Donner l'identification du réseau en notation CIDR.

WiFi

(citer vos sources)

5. Combien de canaux radio-fréquence la norme WiFi 802.11g autorise-t-elle en europe ?
   Quelle est la largeur fréquentielle de ces canaux ?
   De combien de MHz les fréquences porteuses de ces canaux sont elles séparées ?
   Pourquoi n'est-il pas possible d'utiliser des canaux adjacents pour déployer plusieurs réseaux WiFi dans un même lieu ?
6. Comment s'appelle le mécanisme qui permet de gérer l'accès multiple au réseau WiFi, c'est-à-dire le mécanisme qui gère les conflits d'accès au canal radio ? (donner l'acronyme).
   Comment ce mécanisme fonctionne-t-il ? (réponse synthétique svp, rédigée avec vos propres mots).

Cours 1 : ESP8266

Généralités

- Sorti en 2014
- Société Chinoise « Espressif Systems »
- ESP8266 est le nom d'un système sur puce (**SoC** : **S**ystem **o**n **C**hip) construit autour :
  - d'un processeur configurable 32 bits (DPU: Digital Processing Unit) à architecture RISC : « Tensilica Xtensa LX3 »
  - d'un module radio WiFi (hard) associé à une pile TCP/IP (soft) lui permettant :
    - de se connecter à un point d'accès,
    - d'être un point d'accès,
    - les deux en même temps.
- Caractéristiques :
  - cadencé jusqu'à 160 MHz
  - le programme doit être stocké dans une mémoire Flash externe
  - consommation : sous 3V, environ 150 mA en émission (Tx), et environ 50 mA en réception (Rx)
- L'esp8266 intègre :
  - des interfaces de communication série (UART, I2C, SPI, I2S, SDIO)
  - un convertisseur analogique/numérique
  - une interfaçe de commande infrarouge
  - des interfaces PWM
- Intégré dans de nombreuses realisations de type IoT utilisant le WiFi comme interface radio
- Evolution de l'esp8266 : esp32 qui intègre en plus une interface radio Bluetooth 4.0 (Low energy)

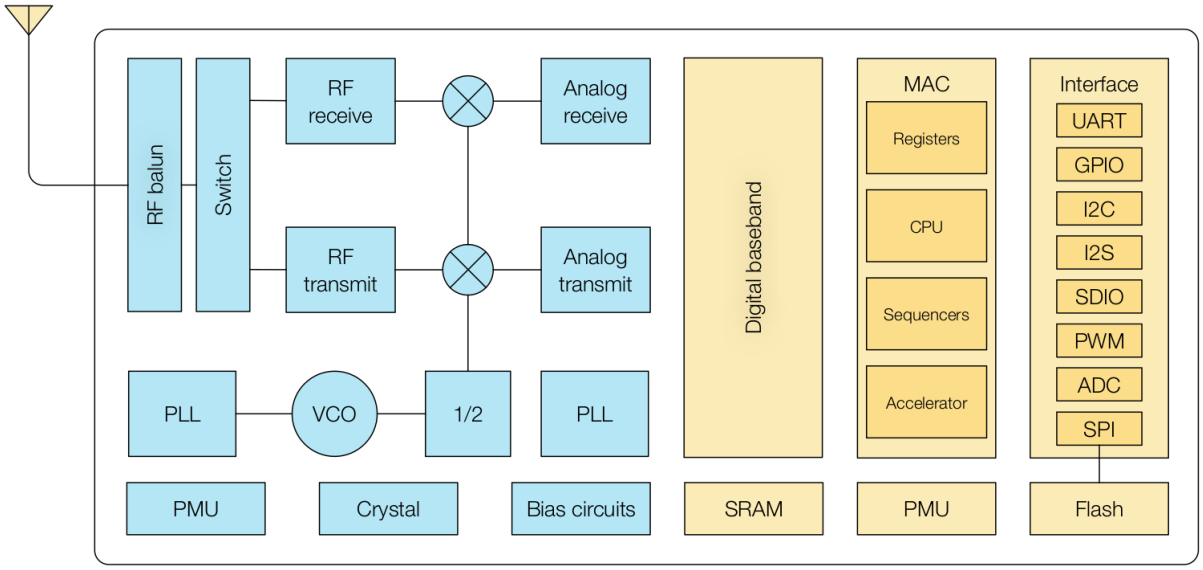


FIGURE 1 – Diagramme fonctionnel de l'esp8266.

Processeur configurable Xtensa

http://linux-xtensa.org

— « The Xtensa processor architecture is a *configurable*, *extensible*, and *synthesizable* 32-bit RISC processor core. Processor and SOC vendors can **select from various processor options** and even **create customized instructions** in addition to the base ISA to tailor the processor for a particular application »

— Customized instructions are recognized as « native » by the entire software development tool chain.

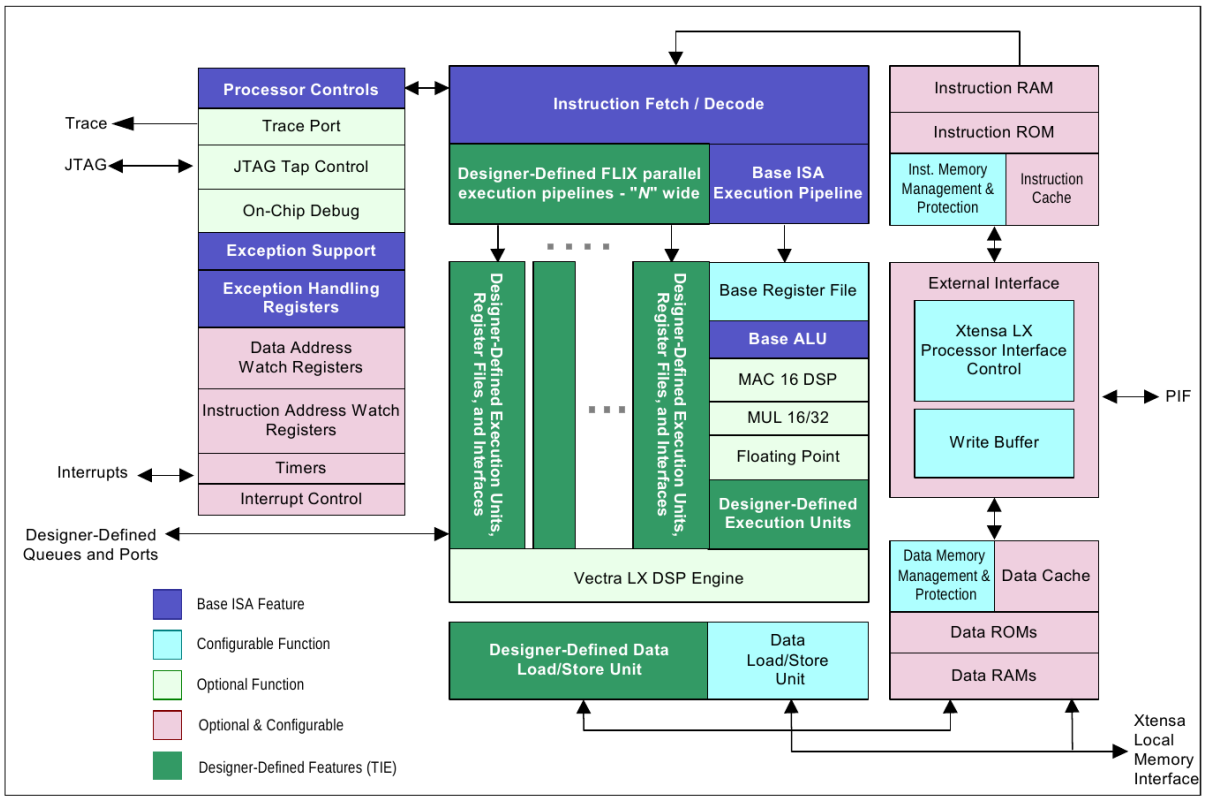


FIGURE 2 – Diagramme fonctionnel du Xtensa.

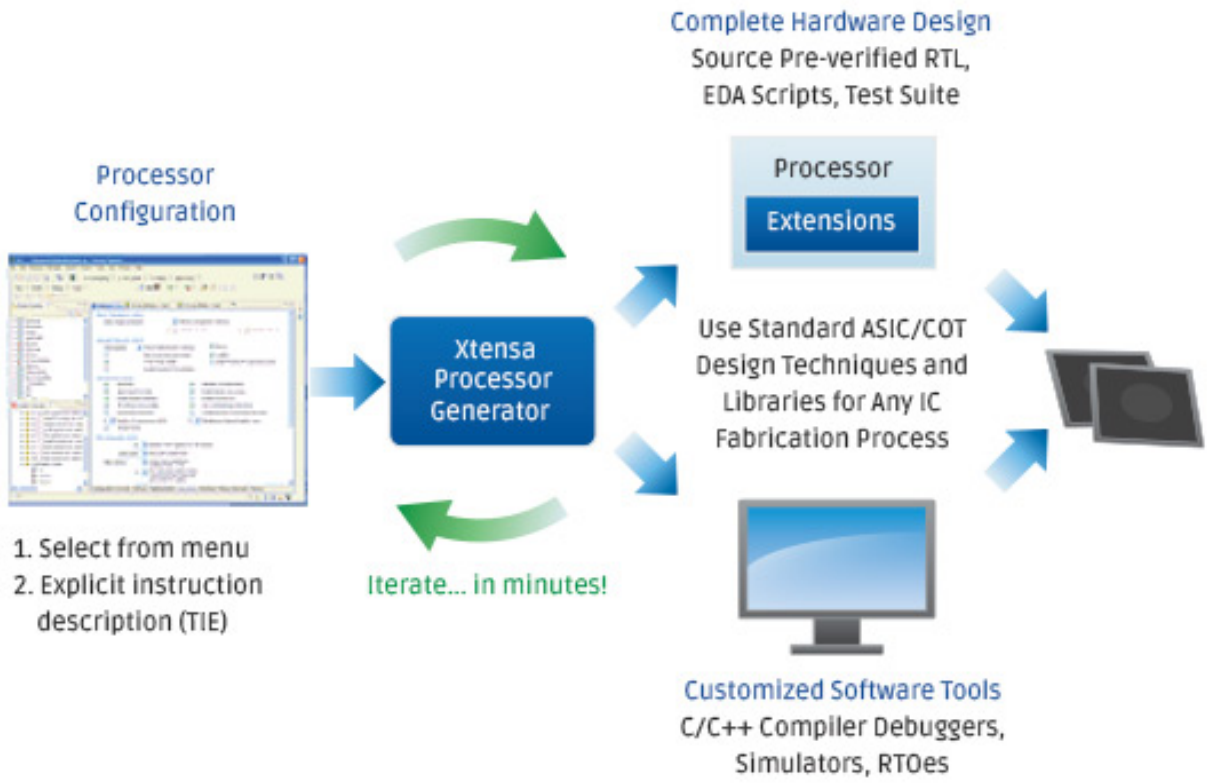


Figure 3 – Flux de conception du Xtensa.

Déclinaisons de l'esp8266

Figure 4 – Série ESP-01 à ESP-011.

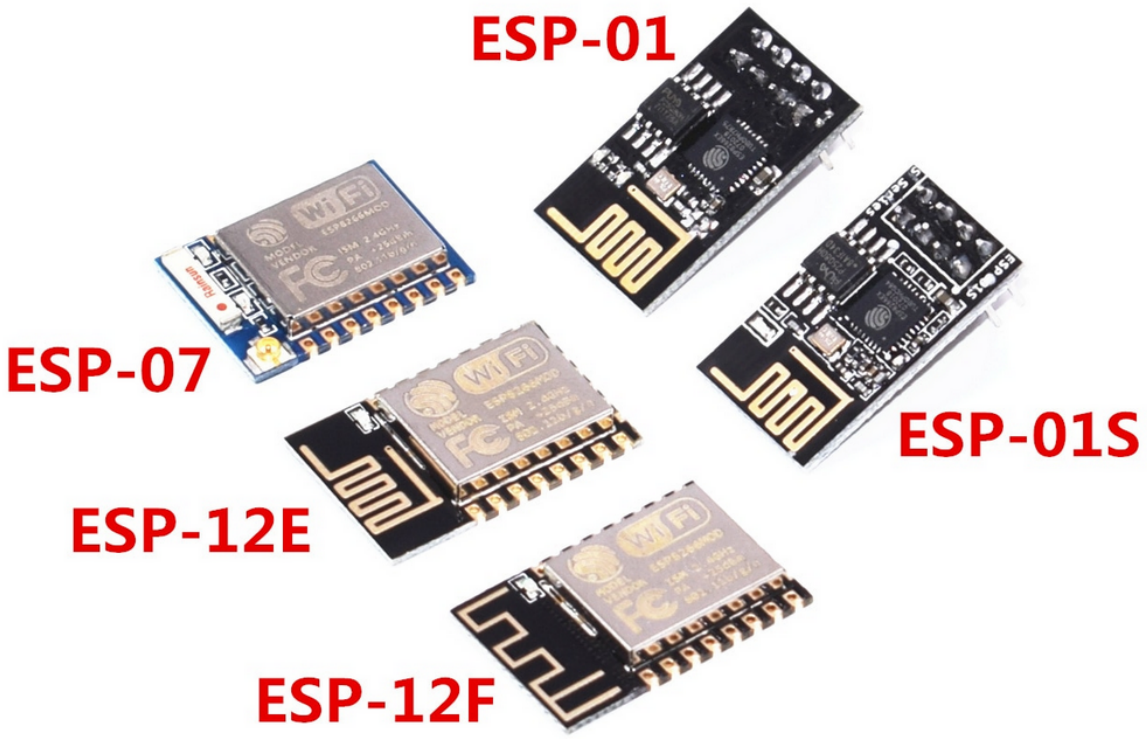
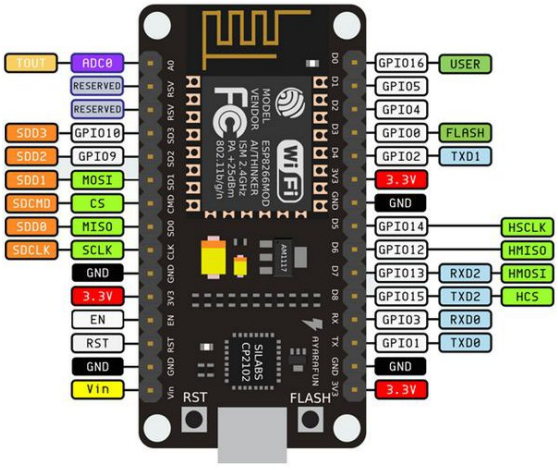
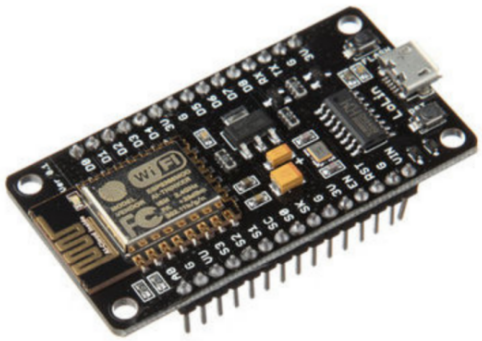


Figure 5 – Les ESP encore utilisés.





Carte de développement NodeMCU

Figure 6 – Carte de développement NodeMCU.

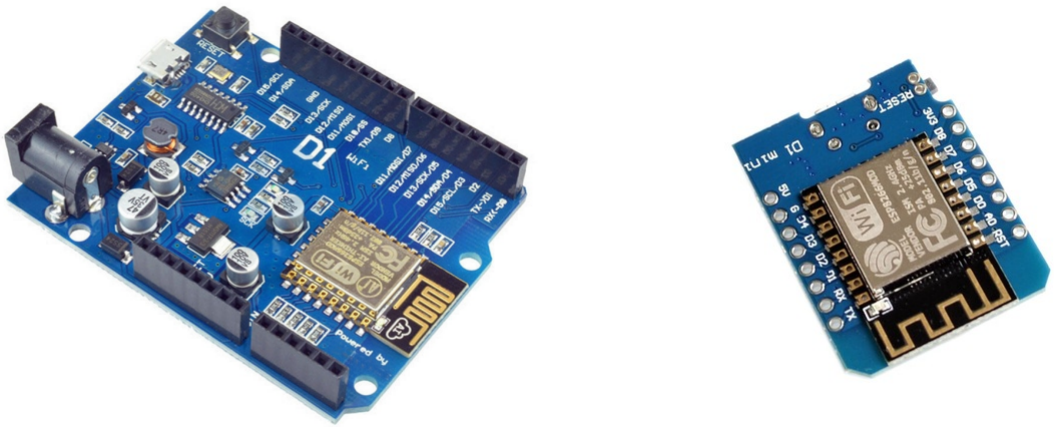


Figure 7 – Carte de développement Wemos D1.

ESP8266 et arduino

Configuration d'arduino

- La couche d'abstraction arduino a été rendue compatible avec l'esp8266
- Configuration de l'IDE arduino pour les cartes à base d'esp8266 :
  - Menu Fichier/Préférences
  - remplir le champ *URL de gestionnaire de cartes supplémentaires* avec :
    `http://arduino.esp8266.com/stable/package_esp8266com_index.json`

Bibliothèque arduino WiFi pour l'esp8266 : bibliothèque *ESP8266WiFi*

- Documentation : https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html
- Code source : https://github.com/esp8266/Arduino.

Avec cette bibliothèque, 3 modes de configuration de l'interface WiFi sont possibles :

- mode *station* (STA) $\rightarrow$ pour connecter l'esp8266 à un point d'accès WiFi existant,
- mode *point d'accès* (AP) $\rightarrow$ pour que l'esp8266 soit un point d'accès WiFi,
- mode combiné *station* et *point d'accès* (STA+AP)
  Remarque pour le mode combiné : l'interface WiFi (hard) devant être partagée entre les modes STA et AP, ces derniers devront utiliser le même canal radio WiFi.

Lab 1 - Prise en main

Présentation

Ce premier lab a pour objet d'apprendre à configurer l'interface WiFi du SoC esp8266 avec la bibliothèque *esp8266WiFi*.

Deux modes de configuration seront étudiées :

— mode station (STA)
— mode point d'accès (AP)

Documentation et matériel utilisé

Documentation

— Code source de la bibliothèque arduino *eps8266WiFi* : [ici]
— Documentation de la bibliothèque arduino *esp8266WiFi* : [là]

Matériel

— Carte arduino Wemos D1 ou carte nodeMCU (esp8266)
— Carte Raspberry Pi configurée en point d'accès WiFi (Partagée par tous les étudiants) :
  — SSID : TPWIFI
  — password : rpi_pass
  — adresse IP du réseau : 192.168.5.0
  — adresse IP de l'interface WiFi de la Raspberry Pi : 192.168.5.1

Informations sur la configuration de la carte RaspberryPI3 en AP WiFi

> Un point d'accès se compose d'un service se chargeant d'accepter et d'authentifier les connexions Wifi (HostAP), et d'un service distribuant les adresses IP (serveur DHCP) pour former un réseau. Pour assurer ces fonctions, il les programmes *hostapd* et *isc-dhcp-server* sont installés sur le Raspberry

1. L'adresse IP du Raspberry ainsi que le masque de sous-réseau pour le réseau WiFi sont configurées dans le fichier `/etc/network/interfaces` avec le code suivant :

```
iface wlan0 inet static
address 192.168.5.1
netmask 255.255.255.0
```

2. Le service *hostapd* est associé au fichier de configuration `/etc/hostapd/hostapd.conf` qui spécifie notamment le SSID, le mot de passe, le numéro de canal WiFi, le protocole de sécurité (wpa).
3. Le ficher de configuration du serveur DHCP *isc-dhcp-server* est : `/etc/dhcp/dhcpd.conf` (voir par exemple la documentation [ici]. Il permet de créer des listes d'association d'adresses matérielles (adresses MAC) et d'adresses IP.

   Par exemple:

   ```
   host_pc_prt {
   hardware ethernet 00:21:00:71:27:1a;
   fixed-address 192.168.5.101;
   }
   ```

Partie A : mode STATION

> Dans cette première partie, vous allez utiliser l'esp8266 en mode *station* afin de le connecter au réseau WiFi « TPWIFI » créé par la carte raspberry pi.

1. Ouvrir le fichier fourni `seance5_lab1_a`.
   - expliquer le fonctionnement du programme, en vous appuyant sur la documentation de la bibliothèque *esp8266WiFi*,
   - compiler et téléverser. Commenter.
2. Modifier le programme afin d'obtenir sur le terminal série l'affichage des éléments suivant (voir figure ci-dessous) :
   - adresse MAC de l'esp8266
   - un point (.) toute les 0,5 s tant que la connexion WiFi n'est pas établie,
   - « WiFi connecté » dès que la connexion est établie
   - l'adresse MAC du point d'accès du réseau WiFi
   - les paramètres réseau suivants :
     - adresse IP délivrée à l'esp8266 par le serveur DHCP du point d'accès
     - masque de sous-réseau
   - toutes les secondes : valeur du paramètre RSSI

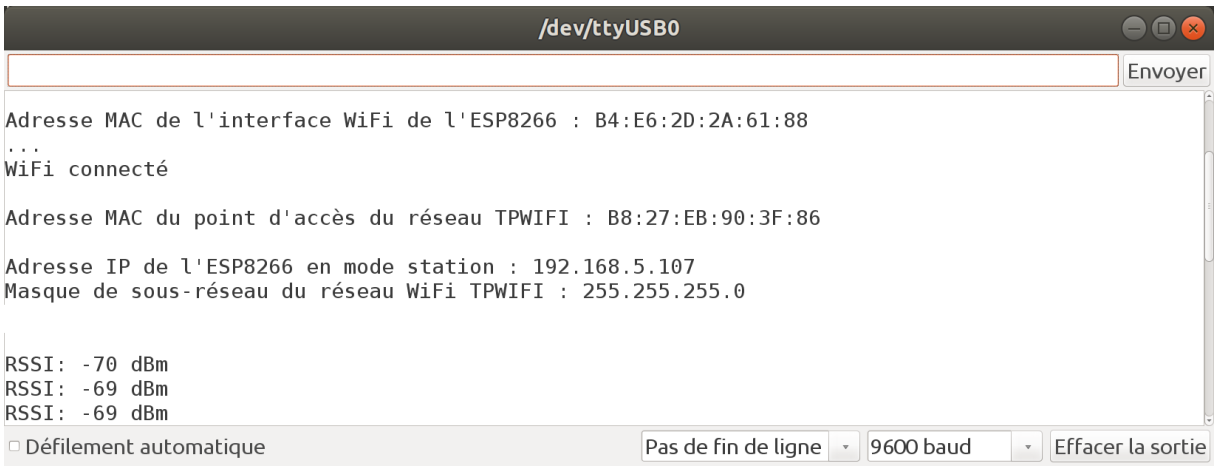


FIGURE 8 – Illustration des éléments à afficher sur le terminal série par le programme seance5_lab1_a.ino.

Partie B : mode Acces Point (AP)

> Dans cette seconde partie, vous allez créer un point d'accès WiFi avec votre carte Wemos D1 ou nodeMCU. Le ssid du réseau sera « esp8266_x »", où *x* est le numéro figurant sur l'étiquette de votre carte.

> Cette partie n'utilise plus la carte raspberry pi.

1. Analyser le programme fourni `seance5_lab1_b.ino`, et adapter le nom du SSID en fonction du numéro de votre carte Wemos D1 ou nodeMCU (étiquette sous la carte).
   Compiler et téléverser le programme puis vérifier sur le moniteur série que le point d'accès WiFi a bien été créé.
2. Organisez-vous avec un autre binôme pour lire les informations de votre réseau via une deuxième carte Wemos D1 ou node MCU programmée avec le programme `seance5_lab1_a.ino` que vous adapterez pour vous connecter au point d'accès créé.
   Noter les valeurs des paramètres suivants de votre AP :
   — adresse IP du point d'accès,
   — masque de sous-réseau.
3. Retrouver ces informations à partir de votre smartphone que vous connecterez à votre AP.
4. Modifier le programme afin de :

— forcer la configuration suivante (en utilisant la fonction `WiFi.softAPConfig()`) :
  — adresse IP du point d'accès : 172.16.4.1
  — adresse IP de la passerelle (gateway) : 172.16.4.0
  — masque de sous-réseau : tel que seules les adresses IP 172.16.4.1 à 172.16.4.14 puissent être attribuées sur le réseau
  — canal WiFi (au choix) : 1, 6 ou 10
— Afficher sur le terminal série ces informations de configuration ainsi que l'adresse MAC du point d'accès, comme indiqué sur la figure 9

Retrouver ces informations sur votre smartphone connecté au point d'accès ou bien, en concertation avec un autre binôme, avec une autre carte Wemos D1 ou nodeMCU programmée avec le code `seance5_lab1_a.ino` que vous adapterez pour se connecter au réseau de votre point d'accès.

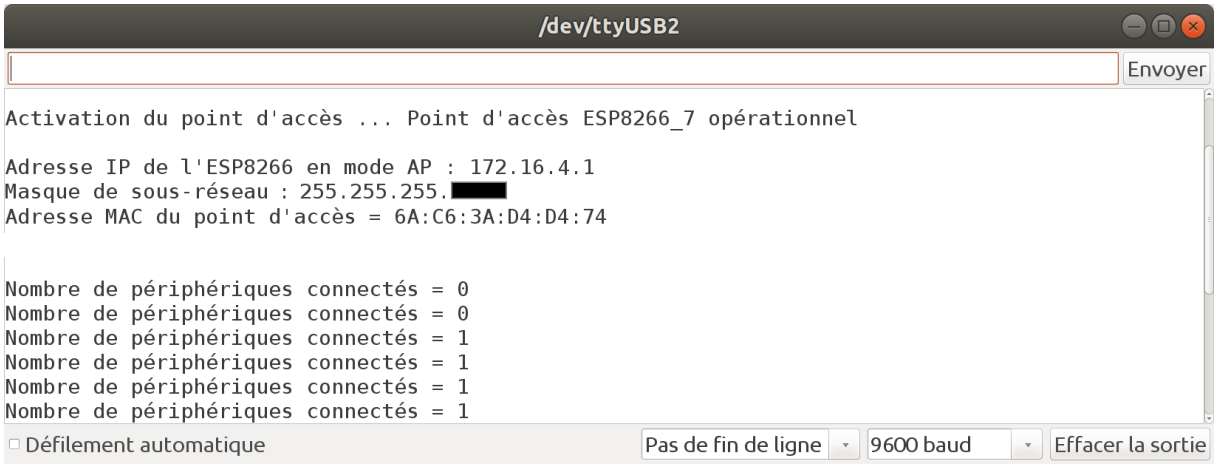


FIGURE 9 – Illustration des éléments à afficher sur le terminal série par le programme seance5_lab1_b.ino.

5. Apporter une dernière modification qui affichera toutes les secondes le nombre de périphériques WiFi connectés au réseau.

Cours 2 : HTTP

Généralités

- Le protocole HTTP est un protocole de la couche *application* (n°4) du modèle TCP/IP qui permet la communications entre un serveur et un client web.
  - le client effectue une requête au serveur, sur la base d'une URL dont le *nom de domaine* identifie le serveur sur le réseau,
  - le serveur répond à la requête du client.
- Exemple d'URL : http://www.uphf.fr/formation
  Le nom de domaine est www.uphf.fr, et /formation est la page web demandée sur le serveur web du domaine (seule une partie de l'URL, le *nom de domaine*, identifie le serveur, le reste indique la page web demandée).
- Le protocole **DNS** (**D**omain **N**ame **S**erver) mémorise les correspondances *adresse IP* $\leftrightarrow$ *nom de domaine* sur plusieurs *serveurs DNS*.

HTTP au dessus de TCP/IP

HTTP est mis en oeuvre au dessus de la couche TCP/IP.

TCP (**T**ransfert **C**ontrol **P**rotocol) est le protocole de la couche *transport* (n°3) du modèle TCP/IP le plus utilisé avec UDP (**U**ser **D**atagram **P**rotocol).

TCP permet :

- de gérer la connexion entre les deux entités qui échangent des paquets (mode connecté)
- de vérifier que les paquets arrivent bien à destination, et dans le bon ordre
- de contrôler les flux
- de diriger les paquets reçus du réseau vers la bonne application. Pour cela, on définit des *ports logiciels*, qui sont des numéros associés aux applications.
  Exemples : 53 (DNS service), 80 (Hypertext Transfer Protocol (HTTP)), 110 (Post Office Protocol (POP3))

Le protocole HTTP

Il existe plusieurs version du protocole HTTP : 0.9, 1.0, 1.1.

La connexion client/serveur peut être *non persistante* (fermeture TCP à chaque requête) ou *persitante* (plusieurs requêtes successives sans fermeture TCP, seulement pour la version HTTP 1.1)

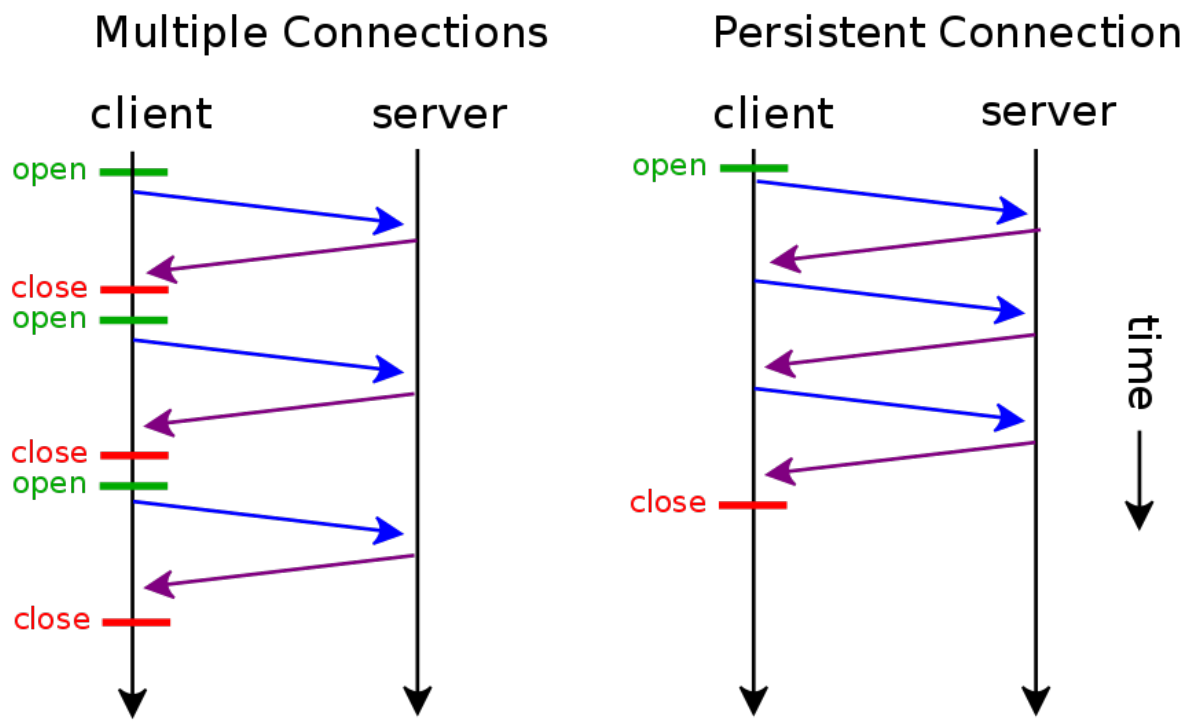


FIGURE 10 – Non persistance et persistance des connexions TCP avec HTTP.

En version 1.1, il existe également un mode *pipeline* qui permet de lancer plusieurs requêtes sans attendre les réponses.

Principe d'une requête client

Exemple minimaliste : → Récupération de la page d'accueil du site www.uphf.fr

```
1 GET / HTTP/1.1
2 Host: www.uphf.fr
```

Commandes possibles pour un client :

- GET (obtenir la page web demandée)
- HEAD (obtenir des informations sur la page, sans la consulter)
- POST (envoyer des données à une ressource sur le serveur), PUT (remplace ou ajoute une ressource sur le serveur), DELETE (supprime une ressource sur le serveur)
- OPTIONS (obtenir les options de communications utilisées par le serveur)
- CONNECT (gestion proxys)
- TRACE (diagnostic de connexion client/serveur).

Principe d'une réponse serveur

Exemple minimaliste : → affichage d'une page html

```
1 HTTP/1.1 200 OK
2 Content-Type: text/html
3 <!DOCTYPE HTML><html>Contenu page web en html<\html>
```

Codes de statut HTTP courants

- 200 : tout s'est bien passé
- 301 et 302 : redirection vers une autre page
- 403 : accès refusé
- 404 : page non trouvée
- 500 : erreur interne au serveur

Lab 2 : Communication client/serveur avec le protocole HTTP

Présentation

Le Lab 1 vous a appris à configurer un réseau WiFi à partir du SoC esp8266 et de l'IDE arduino avec la bibliothèque esp8266WiFi.

Dans ce Lab 2, vous allez apprendre à utiliser un réseau WiFi afin de faire communiquer de l'information entre deux entités du réseau, via le protocole HTTP :

— un client effectue une requête auprès d'un serveur web,
— le serveur web répond à la requête du client en lui transmettant des informations.

La bibliothèque *ESP8266WiFi* sera tout d'abord utilisée (partie A), puis la bibliothèque plus avancée *ESP8266WebServer* (partie B).

> Afin de pouvoir connecter votre smartphone au réseau « TPWIFI », recherchez dans ses paramètres la valeur de son adresse MAC et communiquez la à l'enseignant.

Partie A : le « Hello World » du serveur Web et du client web, avec la bibliothèsque *ESP8266WiFi*.

Dans cette partie, vous allez réaliser un serveur web puis un client web très simples, inspirés de l'exemple fourni [ici].

Serveur web

1. Ouvrir le fichier fourni `seance5_lab2_a_serveur.ino`. Compiler et téléverser.
   Verifier sur le terminal série que l'esp8266 se connecte bien au réseau TPWIFI, et que le serveur web a bien démarré.
2. Depuis un smartphone, connectez-vous au réseau TPWIFI et ouvrir un navigateur web pointant sur l'adresse IP de l'esp8266. Que voyez-vous ?
3. Etablir l'ordinogramme du programme, en explicitant par des mots les différents tests réalisés
4. Recopier le code correspondant à la requête du client.
   — quelle est le numéro de version du protocole HTTP utilisé ?
   — quelle ligne de la requête client indique au serveur l'adresse de la page web à envoyer ?

Client web

Arrangez-vous avec un autre binôme afin que l'une de vos cartes Wemos D1 ou nodeMCU conserve la programmation du code `seance5_lab2_a_serveur.ino`. Bien noter son adresse IP, que l'on notera *IP_add_server* par la suite.

5. Vérifier avec un smartphone que vous arrivez bien à vous connecter au réseau « TPWIFI » puis avec un navigateur web à la page web d'adresse ous avec un autre binôme afin que l'une de vos cartes Wemos D1 ou nodeMCU conserve la programmation du code `seance5_lab2_a_serveur.ino`. Bien noter son adresse IP, que l'on notera et que le message *Hello web server !* s'affiche bien.
6. Ouvrir le fichier fourni `seance5_lab2_a_client.ino`. Compiler et téléverser dans l'autre carte Wemos D1 ou nodeMCU (celle qui ne contient pas le code `seance5_lab2_a_serveur.ino`).
7. Etablir l'ordinogramme du programme, en explicitant les différents tests réalisés par des mots.
8. Recopier le code correspondant à la requête du client, puis celui correspondant à la réponse du serveur. Expliciter chaque ligne de ces codes.

Partie B : un « Hello World » de serveur web amélioré avec la bibliothèsque `ESP8266WebServer`.

Dans cette partie, vous allez créer un serveur web en utilisant cette fois la bibliothèque *ESP8266WebServer*, qui offre un niveau d'abstraction plus élevé que la bibliothèque *ESP8266WiFi*. Notamment, les messages formatés selon le protocole HTTP seront générés automatiquement par la méthode *send* de la classe *ESP8266WebServer*.

Vous allez également pouvoir gérer très facilement la gestion de requêtes clients à différentes adresses. Par exemple :

— si requête à l'adresse racine \ : appel de la fonction `handleroot()`
— si requête à l'adresse *LedOn* : appel de la fonction `handleLedOn()`
— si requête à une adresse inconnue : appel de la fonction `handleNotFound()`
— ...

1. Ouvrir le programme fourni `lab2_b_serveur.ino`. Le compiler et le téléverser dans l'esp8266 de la carte Wemos D1 ou nodeMCU.
   Noter son adresse IP sur le terminal série, puis connectez-vous à celle-ci depuis votre smartphone. Vous devriez visualiser la page de la figure ci-dessous.

Figure 11 – Page web affichée par le serveur web du programme lab2_b_serveur.ino.

2. Analyser maintenant le programme, en comparaison avec le premier serveur web du programme `seance5_lab2_a_serveur.ino`.
   — expliquer la ligne `server.handleClient();` dans la fonction `loop()`,
   — expliquer les valeurs numériques 200 et 404 du premier argument des appels à la fonction `server.send`.
3. Analyser et commenter les balises du code HTML de la page affichée (s'aider de ressources sur internet sur les bases de la programmation en HTML comme par exemple [ici]).

Lab 3 : Application

Présentation

Dans ce dernier Lab, vous allez appliquer ce qui vient d'être vu afin de créer un serveur web dont l'adresse racine \ est l'adresse IP de votre esp8266 connecté en mode *station* au réseau TPWIFI.

Le cahier des charges se décline de manière incrémentielle en trois phases à réaliser successivement.

Matériel

- Carte Wemos D1
- LED RVB
- Résistance de 150 Ω
- platine d'essai
- fils de câblage
- un smartphone ou un PC équipé d'une interface WiFi et d'un navigateur web

---

Phase 1

Réaliser une interface accessible à l'adresse *led* :

- avec trois boutons, permettant chacun d'allumer ou d'éteindre une composante de couleur (RED, GREEN ou BLUE) d'une LED RVB connectée sur trois broches de l'esp8266 (D5, D6, D7). L'action du bouton doit être indiquée sur le bouton (voir Fig. 12).
- l'allumage d'une composante de couleur entraîne l'extinction des autres
- l'interface doit afficher l'état des leds (« All leds are off », « Red led ON », « Green led on » ou « Blue led on ») (voir Fig. 12)

Figure 12 – Interface phase 1.

Indications :

— vous devez déterminer le nombre de pages web à générer, et associer à chacune d'elles une fonction que vous rattacherez avec la commande `server.on`, en suivant le même principe que dans le fichier `seance5_lab2_b_serveur.ino`.
— ces fonctions rattachées gèreront l'état de la led RVB ainsi que l'envoi de la page web adéquate au client, avec la fonction `serveur.send`.

Phase 2

— Ajouter une page web accessible à l'adresse *capteur* pour passer à la lecture d'un capteur analogique, qui affichera la tension mesurée sur la broche A0 (entre 0 et 3,3 V). Le capteur analogique sera simulé par un potentiomètre.

Phase 3 (Bonus)

Réaliser les fonctions supplémentaires suivantes :

— la page web accessible à l'adress racine \ doit faire apparaître deux boutons permettant d'aller vers les adresses *led* et *capteur* respectivement,
— ajout d'un bouton aux interfaces *led* et *capteur* pour revenir à l'adresse racine \.