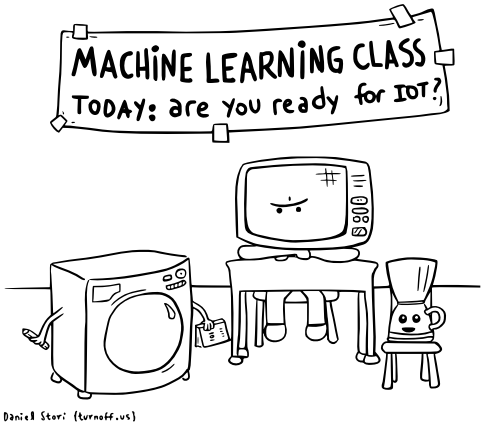
INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

4A STI



Mobilité - OS Embarqué — P-F. Bonnefoi

Version du 21 mars 2021

Qu'est-ce qu'un système embarqué ?
Quels périphériques sont présents ?

CPU pour l'embarqué

CPU
- ☐ exécution du code ;
- ☐ tout le reste est externe : mémoire RAM d'exécution, mémoire contenant le programme ;

Micro Contrôleur
- ☐ CPU ;
- ☐ périphériques intégrés : un peu de mémoire RAM, un contrôleur d'interruptions, un timer, de l'EPROM pour contenir le programme ;

SoC, «*System-on-a-Chip*» : un «*Core*» (CPU nouvelle génération) et de **nombreux** périphériques.

CPU Core	Unité programmable
MMU	Gestion de la mémoire virtuelle nécessaire pour un OS «*High End*»
DSP	Analyse de signal
Power Consumption	Batterie, génération de chaleur
Peripherals	A/D, UART, MAC, USB, Bluetooth, WiFi
Built-in RAM	vitesse et simplicité
Built-in cache	vitesse
Built-in EEPROM or FLASH	mise à jour en exploitaiton, «Field upgradeable»
JTAG Debug Suppot	Deboguage matériel
Tool-Chain	Compilateur, débogueur, ...

Différents usages

▷ **Application** : processeurs 32 ou 64 bits, permet de faire des calculs poussés (présence de DSPs), du multi-média, peuvent faire tourner des OS comme Linux.

▷ **Temps réel** : contrôle de moteurs, robotique : latence basse et sûreté de fonctionnement élevée. Adaptés à des routeurs réseau, des lecteurs multimédias où les données doivent être disponible à un instant donné ;

▷ **Micro-contrôleur** : gestion de matériel (fournis comme «*softcore*» dans des FPGAs), dépourvu de MMU (pas de Linux) mais intègrent de la mémoire et des périphériques.

CPU pour l'embarqué

- ☐ «**von Neumann**» : données et programme sont accédés par le même bus de données et le même bus d'adresse :
  - ◇ le CPU nécessite moins de broches d'E/S et est plus facile à programmer ;
  - ◇ le programme peut être mis à jour après déploiement⟹problème de sécurité ;
- ☐ «**Harvard**» : les données et le programme utilisent des bus différents :
  - ◇ très populaire avec les DSPs, «Digital Signal Processor», utilisant des instructions «Multiply-accumulate» où les opérandes de la multiplication sont au choix :
    - ⋆ une constante en provenance du programme ;
    - ⋆ une valeur fournie par le calcul précédent ou bien en provenance d'un convertisseur A/D ;

    L'architecture Harvard permet de récupérer cette constante et cette valeur **simultanément**.
- ☐ **RISC**, «*Reduced Instruction Set Computing*», *vs* **CISC**, «*Complex Instruction Set Computing*» :
  - ◇ CISC : une instruction peut réaliser une opération complexe mais elle nécessite plus de cycles d'horloge ;
  - ◇ RISC : une instruction peut être plus simple et s'exécuter plus rapidement mais il en faut plusieurs pour réaliser la même opération complexe ;
  - ◇ la programmation en assembleur est plus complexe en RISC, mais l'utilisation de compilateur rend la programmation directe en assembleur inutile dans la plupart des cas.
- ☐ exploitation de l'effet «**pipeline**» :
  - ◇ une instruction se décompose en plusieurs étapes : chercher l'instruction, la décoder, chercher les opérandes, exécuter l'instruction, stocker le résultat.
  - ◇ pour utiliser les bus de manière plus efficace, le CPU peut réaliser les différentes étapes sur différentes instructions :

	instruction 1	instruction 2	instruction 3	...
fetch			✓	
decode		✓		
fetch	✓			
execute				
store				

temps ↓

- ☐ «***endianness***» : «little-endian» *vs* «big-endian» : exemple sur 32bits, soient 4 octets :

@	+1	+2	+3
78	56	34	12

vs

@	+1	+2	+3
12	34	56	78

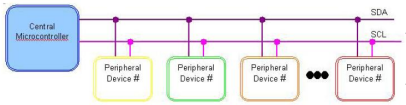
Les périphériques de l'embarqué

- **Interrupt Controller** : gérer les différentes interruptions et leur priorités ;
- **DMA**, «*Direct Memory Access*» : bouger des zones mémoires indépendamment du processeur :
  - «*burst-mode*» : le circuit DMA prend le contrôle complet du bus aux dépends du CPU ;
  - «*cycle-stealing*» : négociation entre le DMA et le CPU ;
  - «*transparent*» : le DMA n'utilise le bus que lorsque le CPU ne l'utilise pas ;
- **MAC**, «*Medium Access Control*» : contrôle la couche 2 d'une interface réseau ;
- convertisseur **A/D** : numérise une valeur analogique en une valeur numérique suivant une résolution de 10 à 12 bits (avec un taux bas d'échantillonnage et un fort *jitter*.
- **UART**, «*Universal Asynchronous Receive/Transmit*» : liaison série de faible vitesse (par exemple RS232), en général de 9600 baud à 115200 baud/s avec des données sur 8bits, pas de contrôle hardware et 1 bit stop : «57600 N 8 1».
  3 fils pour relier deux appareils : le GND partagé, la broche TX de l'un reliée à la broche RX de l'autre et vice-versa.
- **USB** : liaison série haut débit, offrant différents «*Device Classes*» : périphérique HID, «*Human Interface Device*» : clavier/souris, tunnel TCP/IP, mémoire de masse, son *etc.* USB OTG, «*On The Go*», permet dávoir le rôle de maître ou de périphérique.
- **CAN**, «Controller Area Network» : bus inventé par Bosch pour les communications entre les différents circuits dans une voiture et utilisé dans les usines, entre des capteurs, *etc.*
- **WiFi** : échange continue d'information : débit élevé et données de taille quelconque mais consommateur d'énergie. l'antenne peut être externe ou incorporée dans le PCB, «printed circuit board» du circuit ;
- **Bluetooth**, BLE, «*Bluetooth Low Energy*» : échange intermittent d'information : faible débit de données réduites mais avec une très faible consommation.

Les périphériques de l'embarqué

- **bus** :
  - I^2C, SPI communication intelligente de données entre composants électroniques : associés à de la mémoire locale sur le périphérique : décharge le CPU de la gestion d'interruptions de composants disposant de leur propre rythme de fonctionnement (mesure de température, écran, autre CPU *etc.*) ;
  - GPIOs, «*General Purpose I/O*» : PWM, «*Pulse Width Modulation*» : contrôle de périphérique/moteur/radio (télécommande en 433Mhz, ou IR), , «*bit-banging*» : émulation de bus exotique ou connexion directe de composant (détecteur PIR de mouvement, interrupteurs *etc.*)
- **RTC**, «*Real Time Clock*» : maintenir l'heure et la date (utilisation d'une batterie séparée).
  Si le composant est «connecté» il peut utiliser un serveur NTP, «*Network Time Protocol*».
- **Timers** : compteur incrémentés ou décrémentés en fonction du temps gérés de manière indépendante du CPU
  - «*watchdog timer*» : un compteur qui doit être réinitialisé, «*kicked*», de manière logicielle avant qu'il n'atteigne zéro ⟹ s'il atteint zéro, le CPU subit un reset : l'idée est qu'il est dans une boucle infinie ou bien dans un interblocage ;
  - «*fast timers*» : mesurer la longueur d'impulsion ou pour les générer (PWM) ;
- **Memory controller** : obligatoire pour la DRAM, «dynamic RAM» : rafraîchissement de la mémoire de manière régulière (souvent intégré au CPU). Gérer la mémoire FLASH persistente.
- **co-processeur cryptographique** : réaliser des opérations de chiffrement/déchiffrement et signature avec des algorithmes symétriques et surtout asymétriques (coutêux pour le CPU).
  Embarque des clés de chiffrement qui peuvent être figées dans sa mémoire
  (exemple ATECC608 : propose du chiffrement sur courbe elliptique).
- système de **localisation satellitaire** : GPS américain, Glonass russe, Beidou chinois et Galileo européen.
  Permet de disposer de la position et de l'heure à une précision de 50 ns.

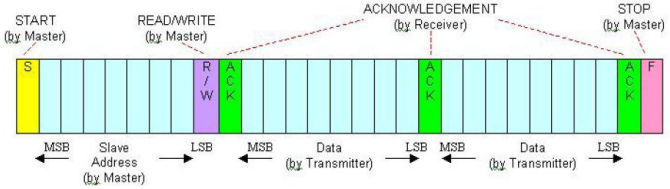
Le bus de communication I2C



Le bus I2C, «*Inter-Integrated Circuit*» :

- un bus générique proposé par Philips dans les années 80, beaucoup utilisé dans les télévisions ;
- synchrone ;
- **débit** : jusqu'à 400 Kbps ;

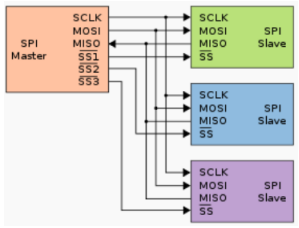
- seulement 2 signaux :
  - `SCL`, «*Signal Clock*» : le contrôleur «Master» génère l'horloge ;
  - `SDA`, «*Signal Data*» : le «Master» transmets les informations et le «Slave» transmets l'acquittement : si aucun acquittement n'est reçu la communication peut être stoppée ou réinitialisée.



▷ plusieurs «*Slaves*» peuvent être connectés au même bus ;

▷ chaque *Slave* doit disposer d'une **adresse** sur 8bits, composée de :
  - une partie fixe qui dépend du constructeur ;
  - une partie configurable ;
  - le dernier bit qui définit le sens de la communication : 0 pour écrire et 1 pour lire ;
  - les communications commencent par un bit de début, «*start bit*», suivi de l'adresse sur 8 bits, le bit d'acquittement, un octet de donnée, un autre bit d'acquittement and à la fin un bit d'arrêt

Le bus de communication SPI, «*Serial Peripheral Interface*»



- un bus générique proposé par Motorola dans les années 80 ;
- communications :
  - full duplex ;
  - synchrone ;
  - lien «Master/Slave» : c'est le master qui initie le transfer des trames de données ;
  - plusieurs liens simultanés possibles : un fil par slave permet de sélectionner celui avec lequel on veut communiquer ;
- **Débit** : quelques dizaines de Mbps ;

- 4 signaux :
  - `SCLK`, «*Clock*» : l'horloge obligatoire pour la transmission synchrone ;
  - `MOSI`, «*Master Out Slave Input*» : communication Master ⟹ Slave ;
  - `MISO`, «*Master Input Slave Output*» : communication Slave ⟹ Master ;
  - `SS`, «*Slave Select*» : un fil par Slave pour pouvoir le sélectionner ;

«SPI *vs* I2C» : Quel bus choisir ?

- le bus SPI permet des débits plus rapides ;
- le bus I2C ne nécessite que 2 fils **mais** nécessite un protocole de communication plus complexe : adressage, définition de trames, gestion de l'acquittement.

	SPI	I2C
Application	Better suited for data streams between processors	Occasional data transfers. Generally used for slave configuration
Data rates	>10 Mb/s	< 400 kb/s
Complexity	3 bus lines More wires more complex wiring More pins on a chip	Simple, only 2 wires Complexity does not scale up with number of devices
Addressing	Hardware (chip selection)	Built-in addressing scheme
Communication	No acknowledgment mechanism, Only for short distances	Better data integrity with collision detection, acknowledgment mechanism, spike rejection
Specification	No official specification	Existing official specifications
Licensing	free	free

Gestion de la mémoire

Les différents types de mémoire

- la mémoire **non volatile** où des données peuvent être stockées et où elles resteront mémorisées malgré les resets et extinction du module :
  - Flash : l'espace programme où le firmware est stocké ;
  - EEPROM : utilisé pour des données utilisateurs
- SRAM ou «*Static Random Access Memory*» : où les les variables sont créées et manipulées lors de l'exécution du programme, «*at runtime*» ;

La quantité mémoire disponible suivant la «board» utilisée

ATMega328 (UNO)
- Flash 32Ko (0,5Ko utilisé par le «*bootloader*») ;
- SRAM 2Ko ;
- EEPROM 1Ko.

ATMega2560 (MEGA)
- Flash 256Ko (8Ko utilisé par le «*bootloader*») ;
- SRAM 8Ko ;
- EEPROM 4Ko.

Consommation de mémoire et programmation

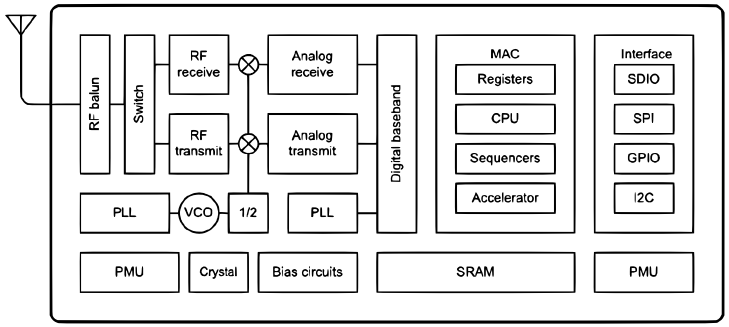
```
1 char *mon_texte = "Super la programmation sur Arduino !";
```

Cette déclaration de $36 + 1$ caractères consomme de la SRAM qui est limitée à 2048 octets sur un Arduino Uno.

Il est possible de loger cette chaîne de caractères dans la **mémoire Flash** grâce à l'utilisation du mot-clé `PROGMEM`.

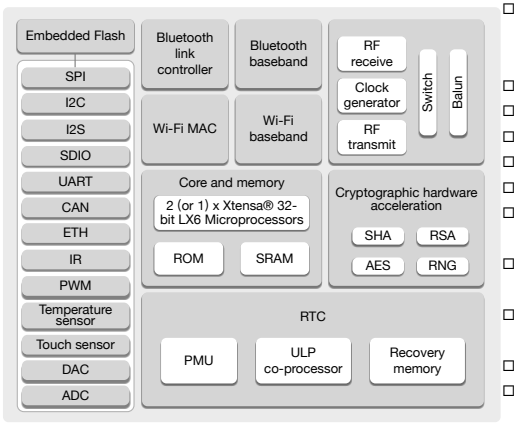
La documentation est disponible à l'adresse : `http://arduino.cc/en/Reference/PROGMEM`

ESP8226



- ESP8266 SoC by Shanghai-based Chinese manufacturer, Espressif Systems ;
- CPU : Tensilica Xtensa L106 : 32bits à 80/160MHz, architecture Harvard modifiée ;
- 64Ko instruction, 96Ko données, FLASH externe de 512ko à 4Mo ;
- consommation : 3,3v 215mA ;
- WiFi b/g/n mode STA ou AP ;
- timers, deep sleep mode, JTAG debugging ;
- GPIO (16), PWM (3), A/DC 10 bits (1), UART, I^2C, SPI, PMU «*Power management unit*».

ESP32 : la nouvelle plateforme avec 2 cœurs + BLE

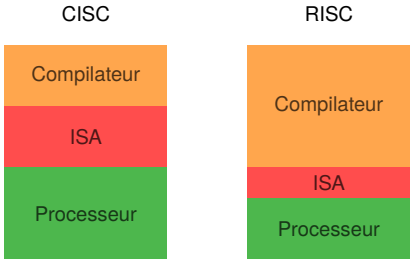


- Xtensa® single/**dual-core** 32-bit LX6 microprocessor(s), up to 600 DMIPS
  (200 DMIPS for single-core mi- croprocessor)
- **448 kB** ROM, **520 kB** SRAM, **16 kB SRAM** in RTC
- QSPI flash/SRAM, up to 4 x 16 MB
- Power supply: 2.3V to 3.6V
- Internal 8 MHz oscillator with calibration
- Internal RC oscillator with calibration
- External 2 MHz to 60 MHz crystal oscillator
  (40 MHz only for Wi-Fi/BT functionality)
- External 32 kHz crystal oscillator for RTC with calibration
- Two timer groups, including 2 x 64-bit timers and 1 x main watchdog in each group
- RTC timer with sub-second accuracy
- RTC **watchdog**

- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit DAC
- 10 × touch sensors
- Temperature sensor
- 4 × **SPI**, 2 × I2S, 2 × **I2C**, 3×**UART**
- 1 host (SD/eMMC/SDIO), 1 slave (SDIO/SPI)
- **Ethernet MAC** with DMA and IEEE 1588 support
- **CAN 2.0**
- IR (TX/RX)
- Motor PWM
- LED PWM up to 16 channels
- **Hall sensor**
- Ultra-low-noise analog pre-amplifier

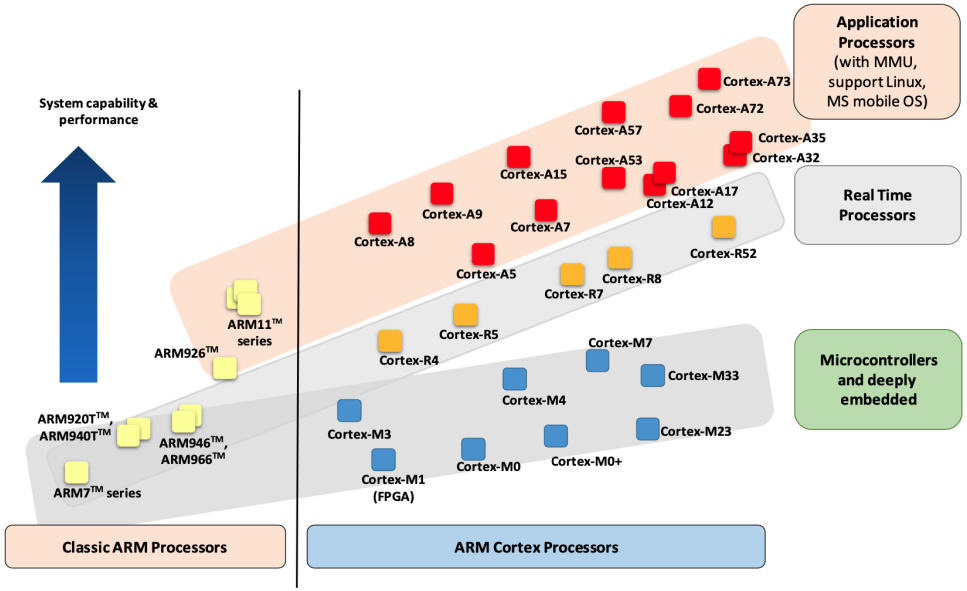
Qu'est-ce que «*RISC-V*» ?

- «*Open Standard Instruction Set*», ISA ;
- basé sur les principes RISC, «*Reduced Instruction Set Computer*» ;

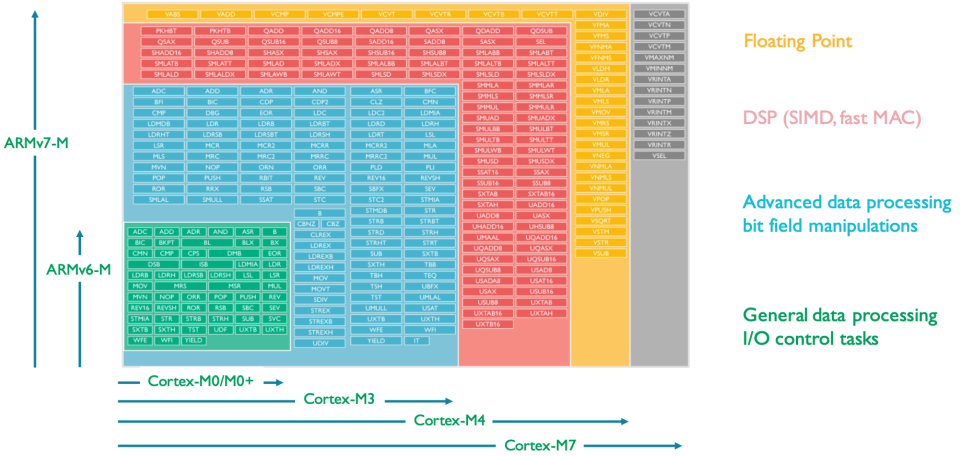


- licence «*Open Source*» sans royalties, mais des extensions payantes... ;
- supporté par :
  - différentes entreprises au niveau de l'offre hardware : sifive ;
  - différents OS : Linux, FreeRTOS ;
  - différents «*toolchains*» : compilateur, linkeur, constructeur de firmware ;
  - sous formes de différentes «*IPs*» pour FPGA, «*Field Programmable Gate Array*».

ARM Cortex et ISA



ARM Cortex et ISA



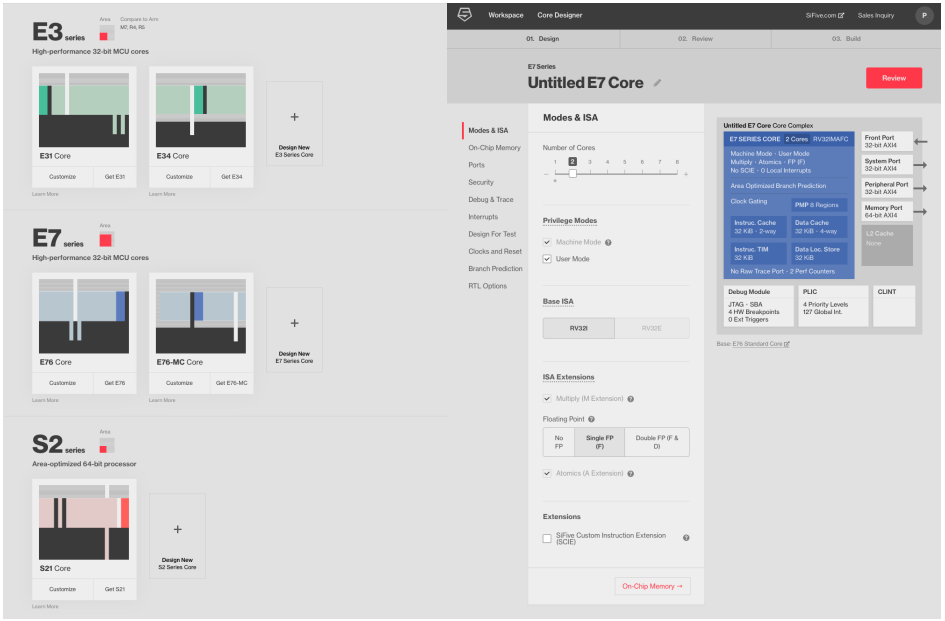
Le Jeu d'instruction RISC-V

Name	Description	Version	Status	Instruction count
	Base			
RVWMO	Weak Memory Ordering	2.0	Ratified	
RV32I	"Base Integer Instruction Set 32-bit"	2.1	Ratified	49
RV32E	"Base Integer Instruction Set (embedded) 32-bit 16 registers"	1.9	Open	49
RV64I	"Base Integer Instruction Set 64-bit"	2.1	Ratified	14
RV128I	"Base Integer Instruction Set 128-bit"	1.7	Open	14
	Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified	8
A	Standard Extension for Atomic Instructions	2.1	Ratified	11
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified	25
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified	25
Zicsr	Control and Status Register (CSR)	2.0	Ratified	
Zifencei	Instruction-Fetch Fence	2.0	Ratified	
G	*"Shorthand for the IMAFDZicsr Zifencei base and extensions intended to represent a standard general-purpose ISA"*	N/A	N/A	
Q	Standard Extension for Quad-Precision Floating-Point	2.2	Ratified	27
L	Standard Extension for Decimal Floating-Point	0.0	Open	
C	Standard Extension for Compressed Instructions	2.0	Ratified	36
B	Standard Extension for Bit Manipulation	0.93	Open	42
J	Standard Extension for Dynamically Translated Languages	0.0	Open	
T	Standard Extension for Transactional Memory	0.0	Open	
P	Standard Extension for Packed-SIMD Instructions	0.2	Open	
V	Standard Extension for Vector Operations	0.10	Open	186
N	Standard Extension for User-Level Interrupts	1.1	Open	3
H	Standard Extension for Hypervisor	0.4	Open	2
Zam	Misaligned Atomics	0.1	Open	
Ztso	Total Store Ordering	0.1	Frozen	

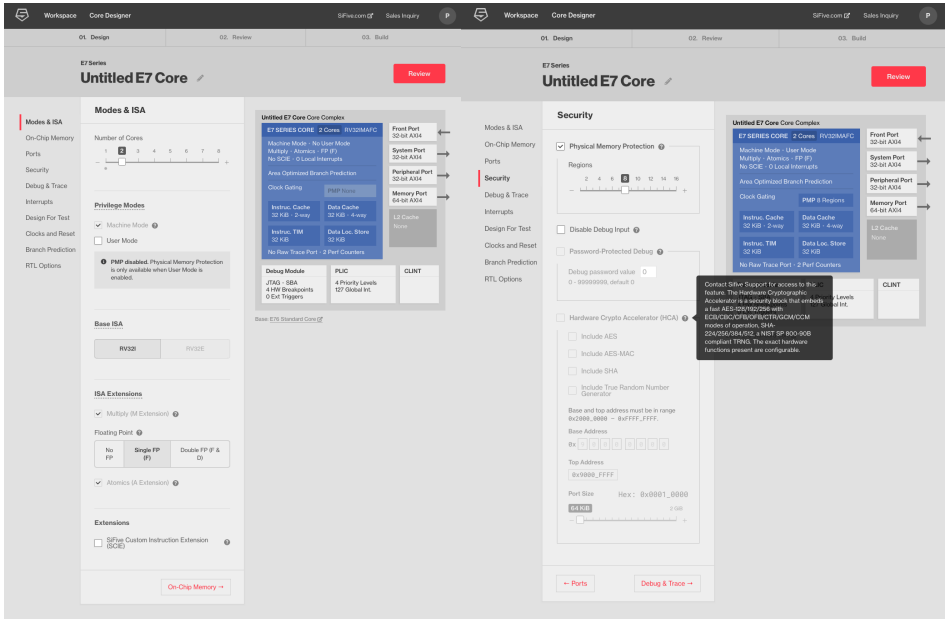
Floating point operation

Le choix des extensions peut amener à des coûts supplémentaires...

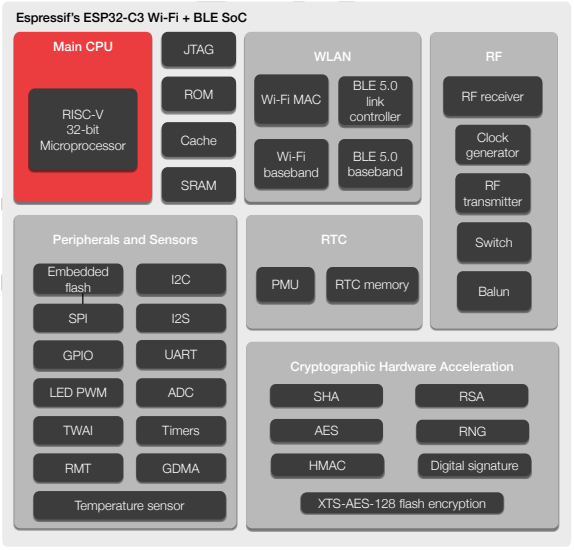
La configuration pour la production d'un processeur sur *scs.sifive.com*



La configuration pour la production d'un processeur sur *scs.sifive.com*



ESP32-C3 : le RISC-V d'Espressif

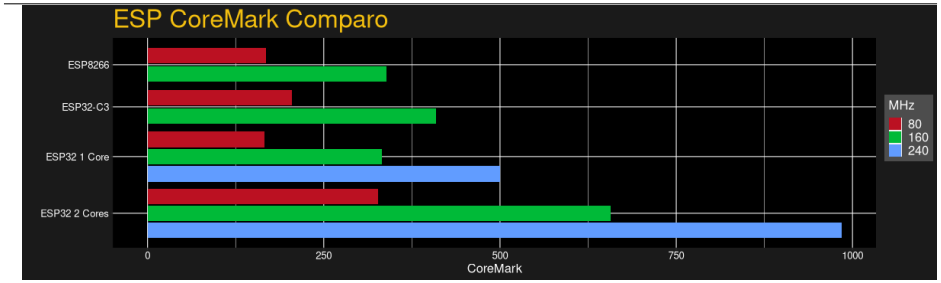


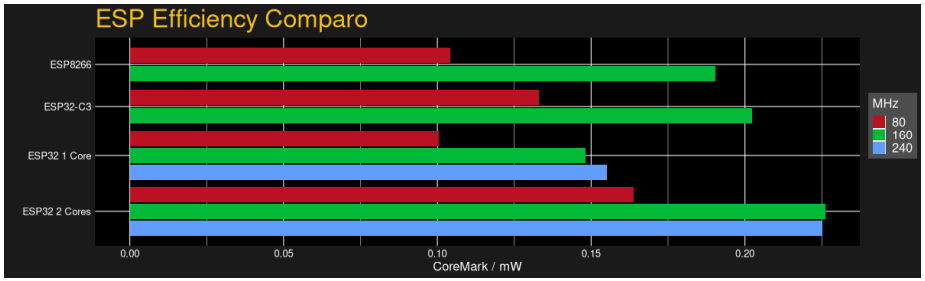
- A complete **Wi-Fi subsystem** that complies with IEEE 802.11b/g/n protocol and supports Station mode, SoftAP mode, SoftAP + Station mode, and promiscuous mode
- A **Bluetooth LE subsystem** that supports features of Bluetooth 5 and Bluetooth mesh
- State-of-the-art power and RF performance
- **32-bit RISC-V single-core processor** with a four-stage pipeline that operates at up to 160 MHz
- **400 KB of SRAM** (16 KB for cache) and **384 KB of ROM** on the chip, and SPI, Dual SPI, Quad SPI, and QPI interfaces that allow connection to **external flash**
- Reliable **security features** ensured by :
  - **Cryptographic hardware accelerators** that support AES-128/256, Hash, RSA, HMAC, digital signature and secure boot
  - **Random number generator**
  - **Permission control** on accessing internal memory, external memory, and peripherals
  - **External memory encryption** and decryption
- Rich set of peripheral **interfaces** and **GPIOs**, ideal for various scenarios and complex applications.

ESP32-C3 family has a low-power **32-bit RISC-V single-core microprocessor** with the following features:

- **RV32IMC** ISA (*voir table sur transparent précédent*) ;
- 32-bit **multiplier** and 32-bit **divider**
- up to **32 vectored interrupts** at seven priority levels
- up to **16 PMP regions** «*Physical Memory Protection*».

ESP32-C3 : comparaisons avec les autres ESPs





La programmation ?

IoT : la programmation

Embarqué *vs* IoT : c'est la même chose, mais l'IoT utilise toujours une pile TCP/IP.

Aucun système d'exploitation : «*polling*» uniquement

Avant de démarrer le programme :

- ▷ construction du programme : compilation, édition de liens et localisation en mémoire ;
- ▷ utilisation d'un «chargeur» : copie du programme en mémoire, bloquer les interruptions, initialiser les zones mémoire pour les données, préparer la pile et les pointeurs de pile.

Conception basée sur une **boucle infinie** :

```
1 int main()
2 {
3    for (;;)
4    {
5       TravailA();
6       TravailB();
7       TravailA();
8       TravailC();
9    }
10}
```

- ▷ chaque fonction correspond à un «*processus*» ;
- ▷ ordonnancement«*round robin*», ou *tourniquet* ;
- ▷ ligne 3 : boucle infinie ;
- ▷ ligne 5&7 : la fonction «`TravailA`» obtient le CPU à des intervalles plus courts que les autres fonctions;
- ▷ chaque «*processus*» réalise la lecture des entrées quand elle a du temps : «*polling*».

Le «polling» : source potentielle de problèmes lors de l'intégration

Boucle de «*polling*» utilisée pour surveiller une entrée qui ne s'arrête que lorsque l'entrée passe d'un état à un autre :

- ▷ *Si cette boucle est intégrée dans `TravailB`, alors le résultat peut être catastrophique pour les fonctions `TravailA` et `TravailC` : elles ne s'exécuteront que lorsque le changement d'état interviendra !*
- ▷ *Et si le changement d'état dépend d'une opération réalisée par `TravailA` ou `TravailC` alors on peut aboutir à un **deadlock** !*
- ▷ *Dans tous les cas, on fait de l'**attente active** ou* «busy waiting» *qui **gaspille de l'énergie**, car le CPU ne peut se mettre en veille et économiser son énergie.*

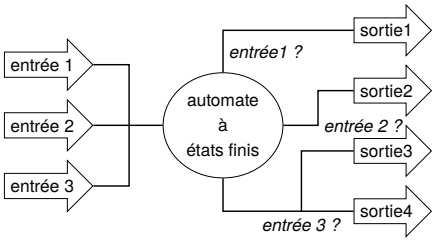
IoT : la programmation

Machine à nombre d'états finis

```
1 int main()
2 {
3    for(;;)
4    {
5       lectureCapteurs();
6       extraireEvenements();
7       transitionEtatEvenement();
8    }
9 }
```

- ▷ ligne 5 : lecture des différentes entrées ;
- ▷ ligne 6 : analyse des entrées pour déterminer un événement : mesure supérieure à un seuil, bouton pressé...
- ▷ ligne 7 : traitement des événements : déclencher les actions à entreprendre ou changer d'état (évolution de l'analyse des entrées et déclencher de nouveaux événements).

Exemple d'utilisation d'un automate à états finis



- ☐ événement sur «*entrée 1*»⟹ «sortie 1» ;
- ☐ événement sur «*entrée 2*»⟹ «sortie 2» ;
- ☐ événement sur «*entrée 3*»⟹ «sortie 3» et «sortie 4» ;

Il est nécessaire de définir :

- ▷ *des **états** : ils permettent de mémoriser les événements déjas reçus dans le cas d'une combinaison de plusieurs événements à prendre en compte pour déclencher une opération ;*
- ▷ *des **transitions** : réception d'un événement ou gestion d'un compteur de temps (on compare une valeur mémorisée à une valeur courante et la différence indique l'intervalle de temps écoulé) ;*
- ▷ *des **actions** : le fait d'effectuer une transition peut déclencher une opération à réaliser sur une des sorties.*

IoT : la programmation

Les «co-routines»

- ☐ une «*co-routine*» peut être exécutée **plusieurs fois**, comme par exemple une fois par ressource identifiée ;
- ☐ une «*co-routine*» peut être **suspendue** pour laisser le CPU exécuter un autre traitement : elle conserve son état et peut reprendre son exécution là où elle s'était stoppée ;
- ☐ cette **suspension peut être invoquée** depuis la «*co-routine*» elle-même au profit d'une autre «*co-routine*» à l'aide de l'instruction «*yield*» et son exécution peut être reprise à l'aide de l'instruction «*resume*» ;
- ☐ il est possible de retourner des valeurs lors du «*yield*» et d'en recevoir lors du «*resume*».

Co-routine et Lua : la bonne façon d'en tirer partie

Version sans co-routines qui peut produire des crashes

```
1 while 1 do
2 gpio.write(3, gpio.HIGH)
3 tmr.delay(1000000) -- waits a second
4 gpio.write(3, gpio.LOW)
5 tmr.delay(1000000) -- and again
6 end
```

L'OS ne reçoit pas de temps pour lui avec `tmr.delay`

Version avec co-routines

```
1 flashDelay = 200 -- ms
2 function flasher()
3   while 1 do
4     gpio.write(3, gpio.HIGH)
5     coroutine.yield(flashDelay)
6     gpio.write(3, gpio.LOW)
7     coroutine.yield(flashDelay)
8   end
9 end
```

Le programme est appelé par `driveCoroutine(flasher)` en version «*good*» ou «*bad*» :

```
1 -- buggy one that will likely
2 -- crash the ESP8266
3 function driveCoroutineBad(proc)
4   co = coroutine.create(proc)
5   while 1 do
6     -- TODO: check bool here and end if appro
priate
7     bool, time = coroutine.resume(co)
8     tmr.delay(time * 1000)
9   end
10end
```

Ne donne pas de temps aux autres co-routines et à l'OS.

```
1 function driveCoroutineGood(proc)
2   co = coroutine.create(proc)
3   delay = 1
4   function resumeAfterDelay()
5     -- TODO: handle bool
6     bool, delay = coroutine.resume(co)
7     tmr.alarm(0, delay, 0, resumeAfterDelay)
8   end
9
10  resumeAfterDelay()
11end
```

L'utilisation d'une alarme, `tmr.alarm` donne du temps à l'OS.

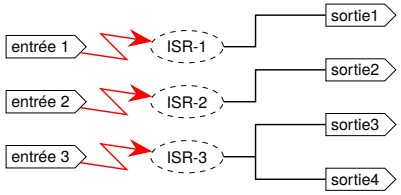
IoT : la programmation

Les interruptions

Elles permettent d'éviter le «*polling*» : une **interruption** peut être générée lorsqu'une entrée change.

Une interruption correspond à un **changement d'exécution** du CPU :

- ▷ une broche d'E/S est associée à un numéro : «*interrupt number*» ;
- ▷ un tableau, le «*interrupt vector*», est stocké à un emplacement précis de la mémoire :
  - ◇ chaque **numéro** est associé à **l'adresse d'une fonction** appelée lors du déclenchement de l'interruption associée ;
  - ◇ chacune de ces **fonctions** est appelée une **ISR**, «*Interrupt Service Routine*» ;
  - ◇ lors d'une interruption, on appelle l'ISR :
    - ⋆ on sauvegarde les registres d'exécution de la fonction courante et on les empile sur la pile ;
    - ⋆ on peut activer ou non la prise en compte de nouvelles interruptions, «*nested*», éventuellement suivant des priorités.



Ici, on a un traitement purement piloté par les interruptions :

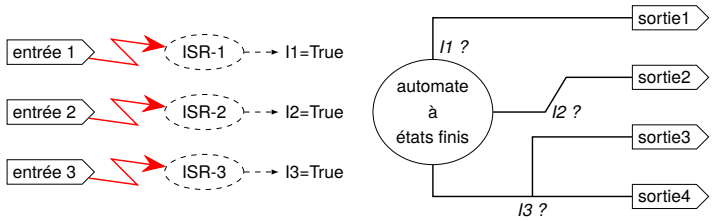
- *– **tout le travail** lié à une entrée est effectué dans une ISR ;*
- *– dans le cas où l'on **autorise** le* «nested» *:*
  - ◇ *une interruption de **priorité supérieure** doit connaitre l'état précis du système ;*
  - ◇ *il peut ne pas y avoir suffisamment de priorités pour gérer toutes les entrées.*
- *– dans le cas où l'on **n'autorise pas** le* «nested» *:*
  - ◇ *toutes les autres interruptions doivent attendre que la première soit terminée⟹ il peut y avoir des retards, «latency», sur les priorités les plus hautes.*

En général, plusieurs entrées déclenchent la même interruption et le premier travail de l'interruption est de trouver quelle est l'entrée qui a changé d'état : l'ordre de recherche défini une **sorte de priorité**.

Par exemple, lorsque l'on a besoin de traiter plusieur entrées sur une même interruption car il n'y a pas suffisamment d'interruptions disponibles pour traiter chaque entrée séparément.

IoT : la programmation

Interruptions et automate à états finis



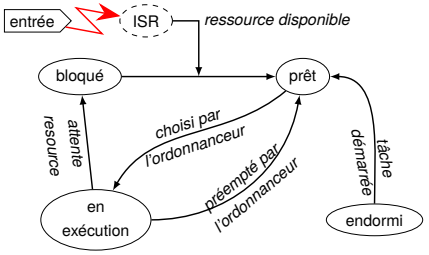
- ▷ *le traitement de l'interruption est rapide :*
  - ◇ *juste positionner un drapeau, «flag», à vrai ;*
  - ◇ *faible latence pour le traitement des autres interruptions.*
- ▷ *c'est l'automate qui gère les priorités s'il y en a besoin.*

Attention

Il est possible de **choisir** comment est déclencher l'interruption :

- ☐ «*level triggered*» : l'interruption se déclenche **tant que** le niveau (haut ou bas) sur la broche, ou «*pin*», est maintenu dans l'état associé à l'interruption ⟹ soit le matériel change le niveau au déclenchement de l'interruption, soit l'ISR doit changer le niveau, sinon l'interruption se **déclenchera de nouveau**.
- ☐ «*edge triggered*» : l'interruption ne se déclenche que lors de la **transition d'un niveau à l'autre**, c-à-d sur à la bordure montante ou descendante de l'impulsion. Si les interruptions n'étaient pas actives lors de ce **court instant**, alors **on n'obtiendra pas d'interruption** (à moins que le système la mémorise pour nous).

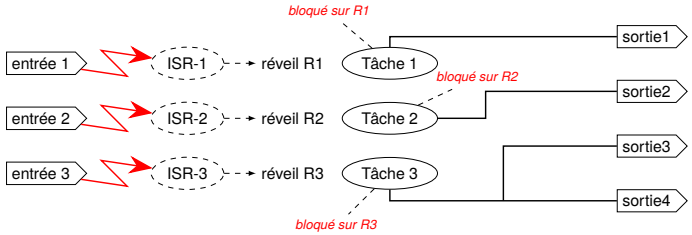
Noyau temps réel



On dispose d'un SE, «Système d'Exploitation», qui gère les différentes tâches et alloue le CPU entre elles :

- ☐ «endormi» : la tâche est créée mais elle n'est pas encore démarrée : elle sera démarrée par le programme.
- ☐ «prêt» : la tâche peut être exécutée, mais elle *attend* le CPU ;
- ☐ «en exécution» : la tâche s'exécute, elle *possède* le CPU ;
- ☐ «bloqué» : la tâche est en attente d'un événement extérieur : une interruption liée à une entrée ou bien un événement réseau.

- ☐ ordonnanceur et préemption : si le noyau supporte la préemption, une tâche est **suspendue** après un «*timeslice*» : tous les registres sont sauvegardés et un **changement de contexte** est réalisé (dans le cas d'une interruption, on peut ne sauvegarder que les registres utilisés par l'ISR).

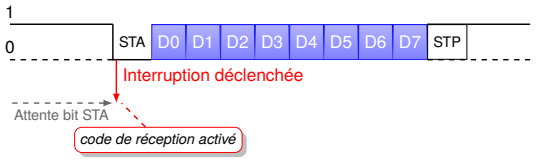


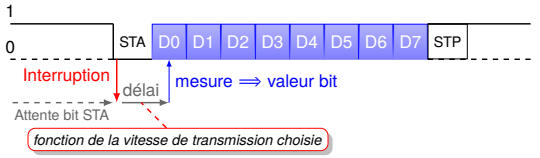
Chaque tâche est suspendue jusqu'à ce que le SE la réveille lorsque un événement se produit au travers d'une ISR.
Les tâches peuvent êtres synchronisées par des **sémaphores**, et communiquées entre elles par des «***message queues***».

Pourquoi un OS temps réel ?

Réception asynchrone : le port série

Le récepteur active une **interruption** qui s'active lors du passage du niveau haut au niveau bas : il exécute le **code de réception** lors du déclenchement de l'interruption :

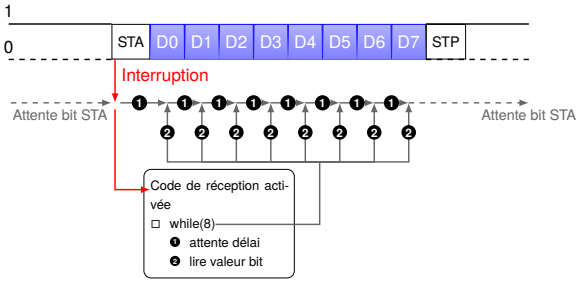




Le **code de réception** :

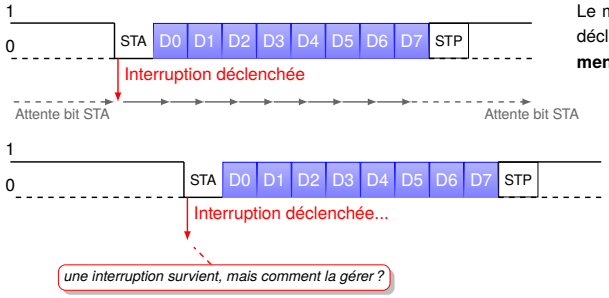
- ▷ attend pendant un certain délai ;
- ▷ mesure le niveau afin de déterminer la valeur du bit ;
- ▷ recommence jusqu'à la réception des 8 bits de données.

Le **code de réception** :



Pourquoi un OS temps réel ?

Gestion de deux canaux de réception asynchrone

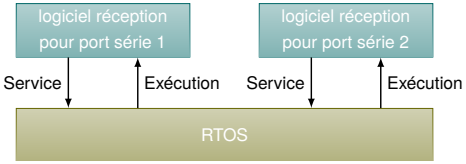


Le modèle de gestion basé sur les routines déclenchées par «*interruption*» est **difficilement extensible** !

La solution : utiliser un OS temps réel, «*RTOS*»

Avec RTOS, la configuration se simplifie :

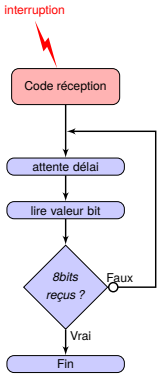
- ▷ on utilise **deux occurences** d'une tâche de gestion de port série ;
- ▷ l'OS fournit un service et s'occupe de son exécution.



Pourquoi un OS temps réel ?

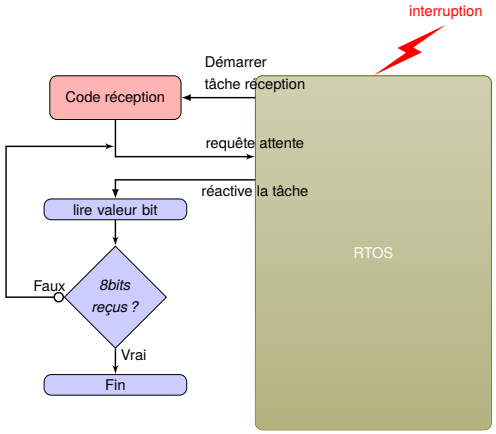
Le code de réception pour une communication asynchrone

Sans RTOS :



Le délai est interne au code de réception.

Avec RTOS :



Le délai d'attente de la tâche de réception est du temps rendu à RTOS.

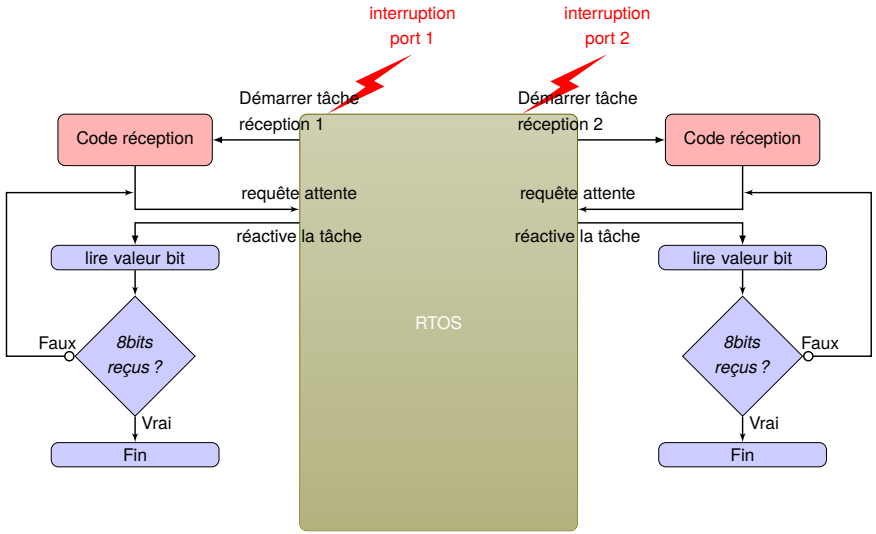
RTOS :

- ☐ gère l'interruption ;
- ☐ déclenche la tâche de réception ;
- ☐ récupère le délai d'attente de la tâche de récepytion ⟹ Il peut l'utiliser pour faire autre chose !

Pourquoi un OS temps réel ?

Gestion de deux canaux de réception asynchrone

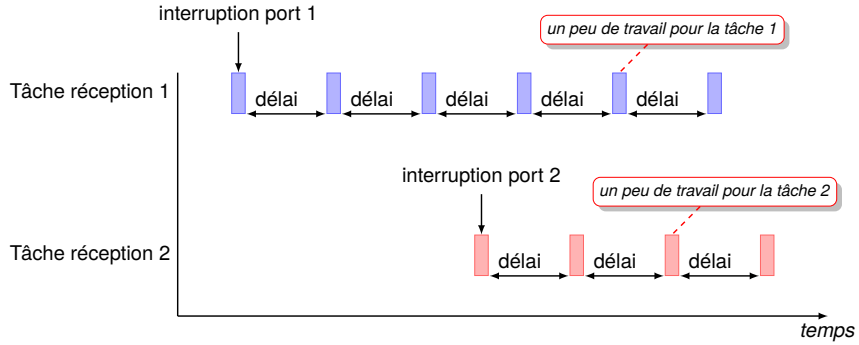
- ☐ On demande l'allocation de deux tâches de réception à RTOS ;
- ☐ RTOS partage le temps entre les deux tâches.



Pourquoi un OS temps réel ?

La responsabilité de RTOS

- ☐ exploite les ressources «*hardware*» de manière efficace :
  ⟹ fournit un mécanisme de bascule entre les différentes tâches ;
- ☐ fournit différents services aux tâches.



Pourquoi un OS temps réel ?

Les avantages liés à l'utilisation de RTOS

- ☐ le logiciel est **modulaire** : chaque partie de logiciel s'occupe d'une tâche bien identifiée et isolée ;
- ☐ ces parties deviennent des **composants réutilisables** ;
- ☐ le logiciel est **plus sûr**, «*reliable*» : plus facile d'isoler et de corriger les erreurs ;
- ☐ **le développement est plus efficace**.

Un RTOS particulier : FreeRTOS

`https://www.freertos.org`
Acheté et développé par Amazon.
«*Has a minimal ROM, RAM and processing overhead. Typically an RTOS kernel binary image will be in the region of 6K to 12K bytes*».

FreeRTOS

Developed in partnership with the world's leading chip companies over a 15-year period, and now downloaded every 175 seconds, FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of libraries suitable for use across all industry sectors. FreeRTOS is built with an emphasis on reliability and ease of use.

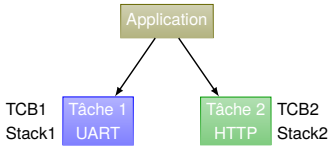
FreeRTOS : la notion de tâche

Soit une **application** constituée de **deux tâches** :

1. gestion d'un port série sous interruption ;
2. gestion d'un serveur HTTP ;

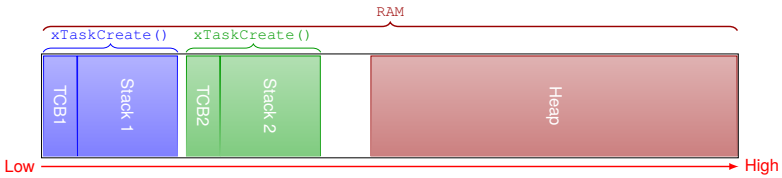
Dans **FreeRTOS**, chaque tâche :

- ☐ est gérée par un TCB, «*Task Control Block*» ;
- ☐ dispose d'une pile, «*stack*», pour ses variables et ses appels de fonction.




```
1 void app_main()
2 {
3     static httpd_handle_t server = NULL;
4    init_uart();
5     // Creation d'une tache pour la gestion de l'UART par interruption
6     xTaskCreate(uart_event_task, "uart_event_task", 8192, NULL, 12, NULL);
7    // Creation d'une tache pour le serveur HTTP
8     xTaskCreate(https_get_task_alt, "https_get_task", 8192, NULL, 5, &tache_https);
9 }
```

Lors de la création de la tâche avec `xTaskCreate()`, la tâche reçoit une **zone fixe** de mémoire pour sa pile.
⟹ un **arrêt de l'application** survient si la tâche **dépasse** cette taille de pile allouée.



La **mémoire RAM du composant embarqué** est divisée en deux parties :

- ▷ une allouée aux différentes **tâches/piles** ;
- ▷ la seconde affectée au **tas**, «*heap*» de l'application complète.

IoT : la programmation dans un OS temps réel

- ☐ On utilise les concepts de la programmation concurrente :

atomic	assure l'atomicité d'une variable susceptible d'être modifiée par différentes tâches
section critique	une section de code qui ne doit être exécutée que par une tâche à la fois
sémaphore	permet de gérer l'accès à n instances d'une même ressource : une tâche est bloquée si elle demande la sémaphore et que celle-ci tombe à zéro après décrémentation. Quand une tâche libère une sémaphore, celle-ci est incrémentée et réveille une tâche en attente de cette-ci.
loquet	implémenté par un mutex
mutex	similaire à une sémaphore mais avec un compteur limité à un
queue	mécanisme d'échange entre tâches : «*message passing*»
ré-entrant	une fonction qui peut être appelée de manière récursive : elle n'utilise pas de données statiques mais uniquement la pile
thread-safe	une fonction qui peut être utilisée par différentes threads en concurrence et dont le code est protégé par des loquets et des sections critiques

- ☐ on décompose le travail du système embarqué en différentes **tâches** ou «*threads*» indépendantes ;
- ☐ on associe les **interruptions** à des **sémaphores**, avec au choix :
  - ◇ l'arrivée d'une interruption active la tâche associée à son traitement ;
  - ◇ on crée un «*message*» reprenant les informations de l'interruption et on le mets dans une «*queue*»⟹ une tâche sera activée pour récupérer et traiter le message ;
- ☐ on utilise les «*queues*» pour :
  - ◇ **synchroniser** les tâches entre elles ;
  - ◇ **échanger** des données entre les tâches ;
  - ◇ traiter des **interruptions**.

Les communications IoT ou M2M

M2M communication : «Machine To Machine»

IIoT, Industrial Internet of Things

M2M : communications entre machines, c-à-d communications temps réel de données sans intervention humaine :

- ▷ télémétrie ;
- ▷ information temps réel en cas d'échec ;
- ▷ contrôle à distance de l'état d'une machine ;
- ▷ acquisition temps réel de données.

Différents protocoles

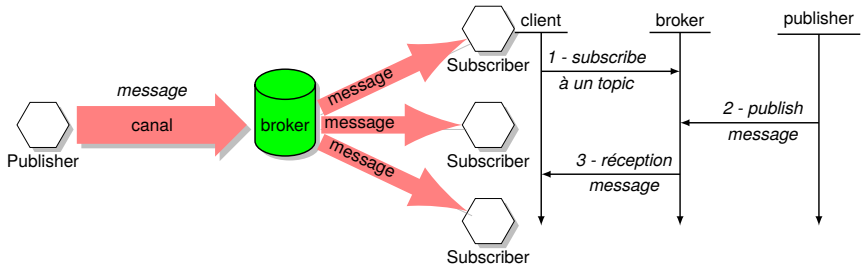
Protocol Name	Transport Protocol	Messaging Model	Security	Best-Use Cases	Architecture
AMPQ	TCP	Publish/Subscribe	High-Optional	Enterprise integration	P2P
CoAP	UDP	Request/Response	Medium-Optional	Utility field	Tree
DDS	UDP	Publish/Subscribe Request/Response	High-Optional	Military	Bus
MQTT	TCP	Publish/Subscribe Request/Response	Medium-Optional	IoT messaging	Tree
UPnP	-	Publish/Subscribe Request/Response	None	Consumer	P2P
XMPP	TCP	Publish/Subscribe Request/Response	High-Compulsory	Remote management	Client/Server
ZeroMQ	UDP	Publish/Subscribe Request/Response	High-Optional	CERN	P2p

MQTT, «*Message Queue Telemetry Transport*»

- Ports réseau :
  - **1883**: This is the default MQTT port. 1883 is defined at IANA as **MQTT over TCP**.
  - **8883**: This is the default MQTT port for **MQTT over TLS**. It's registered at IANA for **Secure MQTT**.

```
xterm
sudo nmap -sS -sV -v -p 1883,8883 --script mqtt-subscribe p-fb.net
```

- Publish/Subscribe modèle :



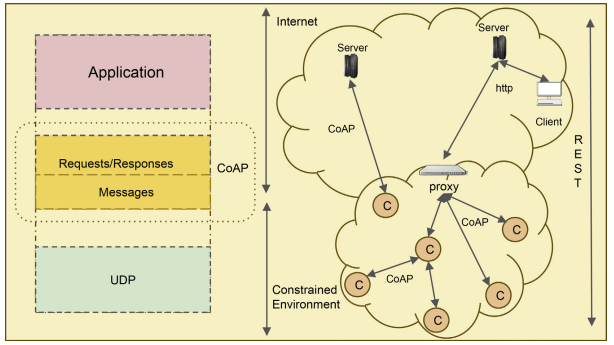
There are a number of **threats** that solution providers should consider.

For example:
- Devices could be **compromised**
- Data at rest in Clients and Servers might be **accessible**
- Protocol behaviors could have **side effects** (e.g. "timing attacks")
- **Denial of Service** (DoS) attacks
- Communications could be **intercepted**, altered, re-routed or disclosed
- Injection of **spoofed Control Packets**

MQTT, «*Message Queue Telemetry Transport*»

- le message peut être au format **JSON** ;
- un message est identifié par des **topics** qui sont organisés en arborescence où chaque niveau est séparé par un « / » :
  - l'opérateur # permet de sélectionner l'ensemble des sous-niveaux : «`capteurs/temperature/maison/#`» permet d'obtenir :
    - `capteurs/temperature/maison/couloir`
    - `capteurs/temperature/maison/chambre`
    - `capteurs/temperature/maison/chambre/fenêtre`
    - utiliser juste «#» renvoie la totalité des topics ;
    - `capteurs/temperature/maison\#` est invalide ;
    - `capteurs/\#/maison/` est invalide ;
  - l'opérateur + permet de «*matcher*» un seul niveau : «`capteurs/temperature/maison/+`» permet d'obtenir :
    - `capteurs/temperature/maison/couloir`
    - `capteurs/temperature/maison/chambre`
    - «+» est valide ;
    - «`+/maison/#`» est valide ;
    - «`capteurs+`» est invalide ;
    - «`capteurs/+/maison`» est valide ;
  - «$SYS/» permet d'obtenir des informations sur le serveur MQTT.
- **sécurité** : le message est en clair, mais la communication peut avoir lieu en TLS/SSL ;
- **QoS**, «*Quality of Service*» :
  - QoS 0, «At Most Once» : un message est délivré au plus une fois ou pas délivré ;
  - QoS 1, «At least Once» : un message est délivré au moins une fois et si le récepteur n'acquitte pas la réception le message est transmis de nouveau ;
  - QoS 2, «Exactly only Once» : un message est délivré une seule fois ;
- MQTT solutions are often deployed in **hostile communication environments**.
  In such cases, implementations will often need to provide mechanisms for:
  - **Authentication** of users and devices
  - **Authorization** of access to Server resources
  - **Integrity** of MQTT Control Packets and application data contained therein
  - **Privacy** of MQTT Control Packets and application data contained therein

CoAP : Constrained Application Protocol



- asynchrone et basé sur UDP, port 5683, RFC 7252, `http://coap.technology` ;

```
xterm
nmap -p U:5683 -sU --script coap-resources p-fb.net
```

- peut fonctionner pour des environnements < 10ko ;
- Quatre types de message :
  - Acknowledgement
  - Reset
  - Confirmable
  - Non-Confirmable : envoyer des requêtes qui n'ont pas besoin de «*reliability*»
- les requêtes sont proches du modèle REST : GET, POST, PUT et DELETE
- le contenu du message peut être au format JSON ;
- le chiffrement peut être basé sur DTLS.

Le protocole HTTP : Modèle CRUD et API REST

CRUD

HTTP	Usage	SQL
POST	«*C*»reates information	INSERT
GET	«*R*»etrieves information	SELECT
PUT	«*U*»pdates information	UPDATE
DELETE	«*D*»eletes information	DELETE

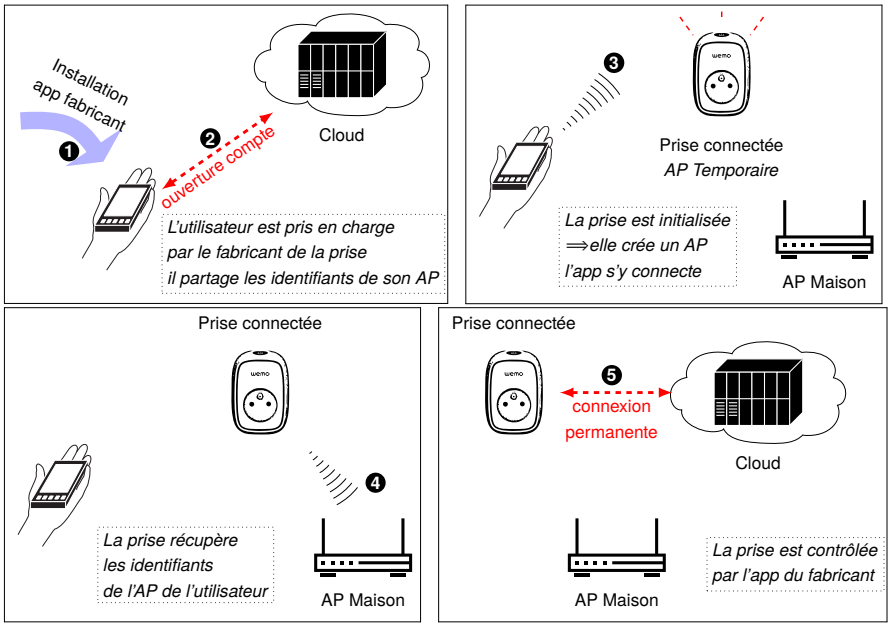
Différence entre PUT et POST ? GET récupère une ressource donnée par son URL et PUT permet de la mettre à jour. Comment créer un objet quand une URL pour y accéder n'existe pas encore ? On utilise POST : POST permet de créer un objet en demandant au containeur parent de créer un élément enfant et l'on reçoit en retour l'URL exacte de cet élément.

Architecture REST, «*Representational State Transfer*»

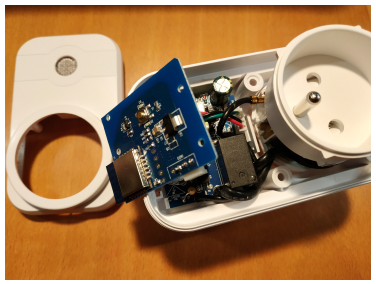
Règle	**Explication**
Accessible	tout est vu comme une ressource qui peut être accessible au travers d'une URI/URL
Sans état	le client et le serveur ne peuvent être désynchronisés. *Contre exemple : lors d'une demande de téléchargement dans un échange FTP, le client et le serveur doivent s'entendre sur le répertoire courant où l'opération aura lieu ⟹ un premier échange ne peut être fait par un serveur FTP puis le second par un autre serveur FTP ⟹ non* «scalable»
Sûr	l'information peut être retrouvée sans causer d'effet de bord *Récupérer une page plusieurs fois ne doit pas avoir d'effet sur le serveur*
Idempotent	la même action peut être réalisée plusieurs fois sans effet de bord *Si une requête correspond à «incrémenter» une valeur de 5 à 6, alors elle doit demander 6 ⟹ elle ne doit pas incrémenter une valeur courante !*
Uniforme	utiliser une interface simple et connue de tous ⟹ HTTP et le CGI, «*Common Gateway Interface*».

Exemple d'IoT : Alexa et une prise connectée
Le matériel, firmware et protocoles

Scénario de l'intégration d'une prise connectée à la maison

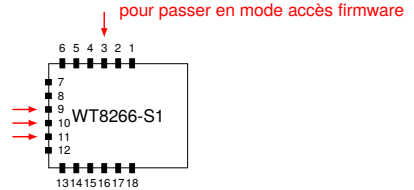


Récupération du firmware



Identifier les broches du WT8266-S1 à l'aide d'un multimètre et de la doc constructeur

Pin	Info
3	IO0
9	URXD
10	GND
11	UTXD



Pour récupérer le firmware présent dans la mémoire flash de 16Mb ou 2MB avec un adaptateur USB/série :

xterm
```
$ esptool.py --port /dev/ttyUSB0 read_flash 0x00000 0x200000 tuya.bin
```

Récupération du firmware

Une fois le firmware récupéré, on peut regarder ce qu'il contient :

xterm
```
$ strings tuya.bin | less
http://a.gw.tuyaeu.com/gw.json
http://a.gw.tuyaus.com/gw.json
...
mq.gw.airtakeapp.com
mq.gw1.airtakeapp.com
...
mqtt_client.c
...
ESP.ty_ws_mod.dev_ap_ssid_key={"ap_ssid":"SmartLife","ap_pwd":null}
ESP.ty_ws_mod.gw_active_key={"token":null,"key":null,"lo
cal_key":null,"http_url":null,"mq_url":null,"mq_url_bak":null,"timeZone":null,"re
gion":null,"reg_key":null,"wxappid":null,"uid_acl":null}
ESP.ty_ws_mod.dev_if_rec_key={"id":"04200063b4e62d00468a","sw_ver":"1.0.0","sche
ma_id":null,"etag":null,"product_key":"n8iVBAPLFKAAAszH","ability":0,"bind":false,"sync":false}
ESP.ty_ws_mod.wf_nw_rec_key={"ssid":null,"passwd":null,"wk_mode":0,"mode":0,"type":0,"source":0,
"path":0,"time":0,"random":0}
ESP.ty_ws_mod.gw_sw_ver_key={"sw_ver":"1.0.0","bs_ver":"5.06","pt_ver":"2.1"}
:ESP.device_mod.dp_data_key={"relay_switch":false,"switch":true,"work_mode":1,"bright":180,"tem
per":255,"colour_data":"1900000000ff19","scene_data":"00ff0000000000","rouguang_scene_da
ta":"ffff500100ff00","binfeng_scene_data":"ffff8003ff000000ff000000ff00000000000000000","xuan
cai_scene_data":"ffff5001ff0000","banlan_scene_da
ta":"ffff0505ff000000ff00ffff00ff00ff0000ff000000"}
ESP.device_mod.fsw_cnt_key={"fsw_cnt_key":0}
ESP.device_mod.appt_posix_key={"appt_posix":0}
ESP.device_mod.power_stat_key={"power":0}
{"mac":"68a","prod_idx":"04200063","auz_key":"TJJ7AGs644QGICVFovUcpeVeGz0JTbR1","prod_test":false}
```

xterm
```
$ dig +short mq.gw.airtakeapp.com
120.55.106.107
$ curl http://a.gw.tuyaus.com/gw.json
{"t":1537555117,"e":false,"success":false,"errorCode":"API_EMPTY","errorMsg":"API"}
```

Détection d'appareils connectés par Alexa d'Amazon

Utilisation du protocole uPnP

- adresse multicast : `239.255.255.250` ;
- port : `1900`

Alexa détecte mes appareils

```
xterm
(b'M-SEARCH * HTTP/1.1\r\nHOST: 239.255.255.250:1900\r\nMAN: "ssdp:discover"\r\nMX: 1\r\nST:
urn:dial-multiscreen-org:service:dial:1\r\n\r\n', ('192.168.0.108', 50892))
(b'M-SEARCH * HTTP/1.1\r\nHOST: 239.255.255.250:1900\r\nMAN: "ssdp:discover"\r\nMX: 15\r\nST:
urn:Belkin:device:**\r\n\r\n', ('192.168.0.118', 50000))
(b'M-SEARCH * HTTP/1.1\r\nHOST: 239.255.255.250:1900\r\nMAN: "ssdp:discover"\r\nMX: 15\r\nST:
urn:schemas-upnp-org:device:basic:1\r\n\r\n', ('192.168.0.118', 50000))
```

La recherche qui nous intéresse est `urn:schemas-upnp-org:device:basic:1`.

La réponse de l'objet connecté

En envoi par uPnP :

```
upnp_response = (b'HTTP/1.1 200 OK\r\n'
                 b'CACHE-CONTROL: max-age=3600\r\n'
                 b'LOCATION: http://%s:%s/setup.xml\r\n'
                 b'ST: urn:Belkin:device:**\r\n'
                 b'USN: uuid:%s::urn:Belkin:device:**\r\n\r\n'
                )
```

On remarque que uPnP utilise du HTTP encapsulé dans un datagramme UDP.

- ▷ l'uuid est un identifiant unique sur 14 octets⟹il identifie l'objet ;
- ▷ le `LOCATION:` indique une URL permettant de se connecter à l'objet.

Accès à un appareil connecté par Alexa d'Amazon

L'objet connecté doit hébergé un serveur HTTP et servir deux URIs :

- ▷ `/setup.xml` :

```
b"""<?xml version="1.0"?>
<root>
  <device>
    <deviceType>urn:MakerMusings:device:controllee:1</deviceType>
    <friendlyName>%s</friendlyName>
    <manufacturer>Belkin International Inc.</manufacturer>
    <modelName>Emulated Socket</modelName>
    <modelNumber>3.1415</modelNumber>
    <UDN>uuid:%s</UDN>
  </device>
</root>
"""
```

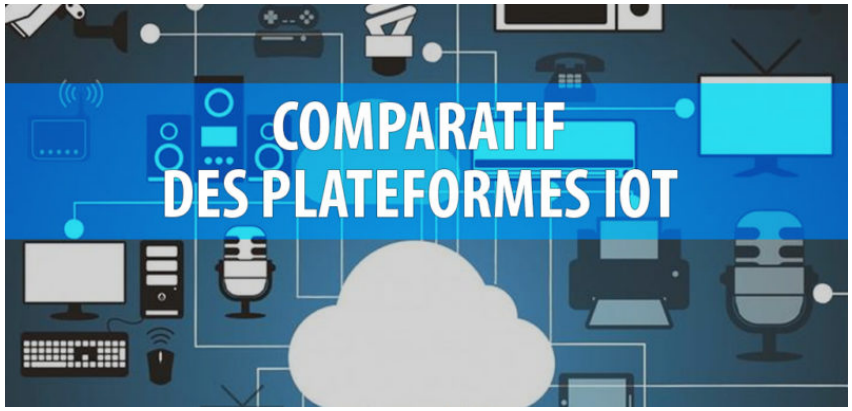
*Où «*friendlyName*» est le nom utilisé pour contrôler l'objet à la voix*.

- ▷ `/upnp/control/basicevent1` :

```
b'<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encoding
Style="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body><u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState></u:SetBinaryState>
</s:Body></s:Envelope>'
```

- requête POST dont le contenu est au format SOAP ;
- le `<BinaryState>1</BinaryState>` indique l'allumage de l'objet connecté.

Panorama des plateformes IoT : Cloud & Infrastructure



Le choix d'une plateforme IoT nécessite de comprendre le principe de base de cette infrastructure particulière. Elle fait le **lien entre**, le **composant**, l'**objet**, la **gateway**, les **données sur le cloud**, les **applications logiciels**, etc. Elle permet de gérer avec granularité ces différents aspects. Elle prend le rôle d'agrégateur de données, d'outils Big Data donc et d'analyse. **Les fournisseurs proposent ainsi un ensemble d'outils décisifs dans différents secteurs.** Certains se spécialisent dans l'installation d'infrastructures au sein des usines, **d'autres s'adressent aussi aux "makers"**, concepteurs qui voudraient monter une preuve de concept rapidement.

`https://www.objetconnecte.com/comparatif-plateforme-iot/`

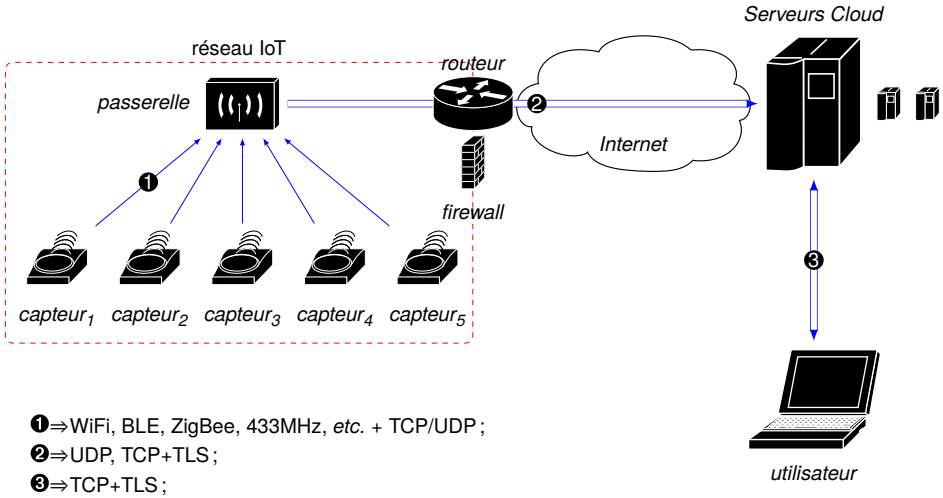
ps://www.telegrafik.fr/wp-content/uploads/2017/07/Panorama-de-Internet-des-objets-V2.pdf

Panorama des architectures IoT et de la Sécurité associée

Plan

- ☐ Architecture IoT : le contexte choisi ;
- ☐ La plateforme matérielle choisie pour l'expérimentation : l'ESP8266 ;
- ☐ Utilisation d'un réseau WiFi sécurisé et d'une application basée REST ;
- ☐ Communication sécurisée par TLS et suivant le modèle «publier/souscrire» ;
- ☐ Composant dédié à la sécurité de l'embarqué : l'ATECC508 ;
- ☐ Communication longue distance avec LoRa et problèmatique de sécurité des communications par paquets ;
- ☐ Nouvelles pistes matérielles et limitation

Composition d'une infrastructure IoT



❶⇒WiFi, BLE, ZigBee, 433MHz, *etc.* + TCP/UDP ;
❷⇒UDP, TCP+TLS ;
❸⇒TCP+TLS ;
L'«*application IoT*» est composée :
- de capteurs ;
- d'une composante logicielle hébergée dans le Cloud ;
- d'une passerelle assurant l'aggrégation et la liaison.

Utilisation de la sécurité fournie par la passerelle et d'une liaison basée WiFi

Création d'un «soft-ap» pour du WiFi avec du WPA2-PSK

xterm

```
pef@cube:~$ cat hotspot
#!/bin/bash
INTERFACEWAN=wlp1s0
# dongle
INTERFACE=wlx48ee0c232796
SSID=IoT

CONFIG=$(cat <<END
interface=$INTERFACE
channel=11
ssid=$SSID
hw_mode=g
wpa=2
wpa_pairwise=TKIP
rsn_pairwise=CCMP
wpa_passphrase=12344321
END
)
hostapd <(echo "$CONFIG") &
ip a add 10.90.90.254/24 dev $INTERFACE
sysctl net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -s 10.90.90.0/24 -o $INTERFACEWAN  -j MASQUERADE
dnsmasq -d -z -i $INTERFACE -F 10.90.90.100,10.90.90.150,255.255.255.0 -O 6,10.90.90.254,8.8.8.8 -A
/serveur.iot.com/10.90.90.254 -l /tmp/leases
```

Association du FQDN `serveur.iot.com` avec l'adresse de la passerelle

La passerelle :

- ▷ fournit le réseau WiFi utilisé par le capteur pour se connecter et communiquer ;
- ▷ crée un réseau local privé, ici en `10.90.90.0/24` ;
- ▷ réalise au besoin :
  - ◇ SNAT pour l'accès à Internet ;
  - ◇ DNS pour la résolution de nom et l'adaptation éventuelle des adresses des services extérieurs.

Utilisation de la sécurité fournie par la passerelle et d'une liaison basée WiFi

Le capteur client se connecte en WPA2 : il obtient son adresse IP, passerelle et serveur DNS

xterm

```
pef@cube:~$ sudo ./hotspot
Configuration file: /dev/fd/63
net.ipv4.ip_forward = 1
Using interface wlx48ee0c232796 with hwaddr 48:ee:0c:23:27:96 and ssid "IoT"
dnsmasq-dhcp: DHCP, IP range 10.90.90.100 -- 10.90.90.150, lease time 1h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface wlx48ee0c232796
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq: read /etc/hosts - 10 addresses
wlx48ee0c232796: interface state UNINITIALIZED->ENABLED
wlx48ee0c232796: AP-ENABLED
wlx48ee0c232796: STA a0:20:a6:2c:84:25 IEEE 802.11: authenticated
wlx48ee0c232796: STA a0:20:a6:2c:84:25 IEEE 802.11: associated (aid 1)
wlx48ee0c232796: AP-STA-CONNECTED a0:20:a6:2c:84:25
wlx48ee0c232796: STA a0:20:a6:2c:84:25 RADIUS: starting accounting session CD0BF4D64A1A71B2
wlx48ee0c232796: STA a0:20:a6:2c:84:25 WPA: pairwise key handshake completed (RSN)
dnsmasq-dhcp: DHCPDISCOVER(wlx48ee0c232796) a0:20:a6:2c:84:25
dnsmasq-dhcp: DHCPOFFER(wlx48ee0c232796) 10.90.90.113 a0:20:a6:2c:84:25
dnsmasq-dhcp: DHCPDISCOVER(wlx48ee0c232796) a0:20:a6:2c:84:25
dnsmasq-dhcp: DHCPOFFER(wlx48ee0c232796) 10.90.90.113 a0:20:a6:2c:84:25
dnsmasq-dhcp: DHCPREQUEST(wlx48ee0c232796) 10.90.90.113 a0:20:a6:2c:84:25
dnsmasq-dhcp: DHCPACK(wlx48ee0c232796) 10.90.90.113 a0:20:a6:2c:84:25 ESP_2C8425
```

Un programme μpython de transmission de la luminosité

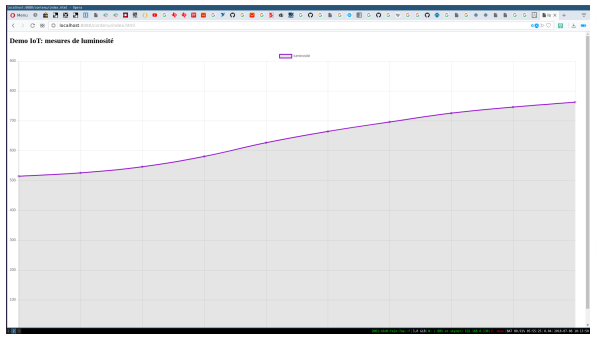
```
from machine import *
import urequests

url="http://serveur.iot.com:8080/luminosity/%s"
adc = ADC(0)

def envoyer_valeur(t):
    v = adc.read()
    urequests.get(url%str(v))

t = Timer(-1)
t.init(period=1000,mode=Timer.PERIODIC, callback=envoyer_valeur)
```

requête DNS réalisée par le capteur le redirigeant vers la passerelle.

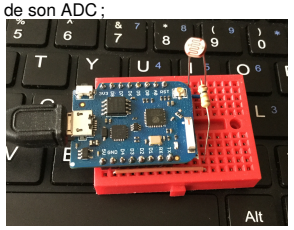
lit la valeur courante de l'ADC et réalise la requête avec la valeur lue.

Utilisation de la sécurité fournie par la passerelle et d'une liaison basée WiFi

Un serveur exploitant les WebSockets

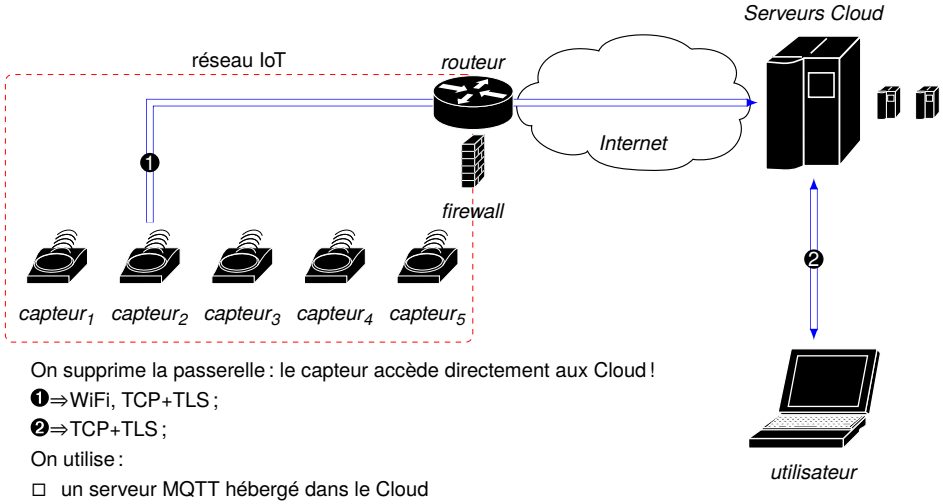


- ☐ un navigateur se connecte sur le serveur logiciel tournant sur la passerelle :
  - ◇ une liaison permanente est établie basée sur un stream SSE, «*Server Sent Events*» ;
  - ◇ il obtient en temps réel les nouvelles valeurs transmises par le capteur ;
  - ◇ affiche un graphe des dix dernières valeurs reçus ;
- ☐ le capteur réalise périodiquement des requêtes REST vers le serveur logiciel de la passerelle : il envoie la valeur courante de son ADC ;



Code source du serveur disponible sur `https://git.p-fb.net/pef/iot_bottle_sse`

Composition d'une infrastructure IoT avec accès direct capteur/Cloud



On supprime la passerelle : le capteur accède directement aux Cloud !

❶⇒WiFi, TCP+TLS ;

❷⇒TCP+TLS ;

On utilise :

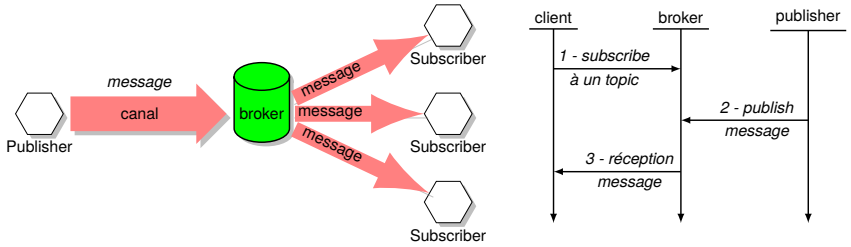
- ☐ un serveur MQTT hébergé dans le Cloud
- ☐ une connexion sécurisée capteur/serveur MQTT avec du TLS ;
- ☐ des **certificats** client et serveur.

MQTT, «*Message Queue Telemetry Transport*»

- Ports réseau :
  - **1883**: This is the default MQTT port. 1883 is defined at IANA as **MQTT over TCP**.
  - **8883**: This is the default MQTT port for **MQTT over TLS**. It's registered at IANA for **Secure MQTT**.

```
xterm
sudo nmap -sS -sV -v -p 1883,8883 --script mqtt-subscribe p-fb.net
```

- Publish/Subscribe modèle :



There are a number of **threats** that solution providers should consider.
For example:

- Devices could be **compromised**
- Data at rest in Clients and Servers might be **accessible**
- Protocol behaviors could have **side effects** (e.g. "timing attacks")
- **Denial of Service** (DoS) attacks
- Communications could be **intercepted**, altered, re-routed or disclosed
- Injection of **spoofed Control Packets**

MQTT, «*Message Queue Telemetry Transport*»

- le message peut être au format JSON ;
- un **message** est identifié par des topics qui sont organisés en arborescence où chaque niveau est séparé par un « / » :
  - l'opérateur # permet de sélectionner l'ensemble des sous-niveaux : «`capteurs/temperature/maison/#`» permet d'obtenir :
    - `capteurs/temperature/maison/couloir`
    - `capteurs/temperature/maison/chambre`
    - `capteurs/temperature/maison/chambre/fenêtre`
    - utiliser juste «#» renvoie la totalité des topics ;
    - `capteurs/temperature/maison\#` est invalide ;
    - `capteurs/\#/maison/` est invalide ;
  - l'opérateur + permet de «*matcher*» un seul niveau : «`capteurs/temperature/maison/+`» permet d'obtenir :
    - `capteurs/temperature/maison/couloir`
    - `capteurs/temperature/maison/chambre`
    - «+» est valide ;
    - «`+/maison/#`» est valide ;
    - «`capteurs+`» est invalide ;
    - «`capteurs/+/maison`» est valide ;
  - «$SYS/» permet d'obtenir des informations sur le serveur MQTT.
- **sécurité** : le message est en clair, mais la communication peut avoir lieu en SSL ;
- **QoS**, «Quality of Service» :
  - QoS 0, «At Most Once» : un message est délivré au plus une fois ou pas délivré ;
  - QoS 1, «At least Once» : un message est délivré au moins une fois et si le récepteur n'acquitte pas la réception le message est transmis de nouveau ;
  - QoS 2, «Exactly only Once» : un message est délivré une seule fois ;
- MQTT solutions are often deployed in **hostile communication environments**.
  In such cases, implementations will often need to provide mechanisms for:
  - **Authentication** of users and devices
  - **Authorization** of access to Server resources
  - **Integrity** of MQTT Control Packets and application data contained therein
  - **Privacy** of MQTT Control Packets and application data contained therein

→ Utilisation de cryptographie asymétrique et de certificats

Utilisation d'un composant de sécurité dédié pour l'embarqué

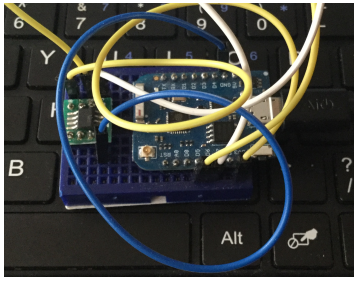
ATECC508A ☆

Status: In Production

View Datasheet

Features:

- Easy way to run ECDSA and ECDH Key Agreement
- ECDH key agreement makes encryption/decryption easy
- Ideal for IoT node security
- Authentication without the need for secure storage in the host
- No requirement for high-speed computing in client devices
- Cryptographic accelerator with Secure Hardware-based Key Storage



*DH : Diffie-Hellman permet d'assurer la PFS, «*Perfect Forward Secrecy*» : la récupération de la clé privée du capteur ne permet pas de déchiffrer les messages échangés précedemments, elle ne sert qu'à l'authentification, pas pour génerer une clé de session*

Connexion directe capteur cloud TLS + Authentification client et serveur

Utilisation du composant ATECC508A

```
[Jul  8 21:26:43.467] esp_mgos_init2        Mongoose OS 1.18 (20180201-160338/1.18@f1e3dd62)
[Jul  8 21:26:43.479] esp_mgos_init2        SDK 2.1.0(ce90efd); flash: 16M; RAM: 52928 total, 49876 free
[Jul  8 21:26:43.479] esp_print_reset_info  Reset cause: 6 (sys reset)
[Jul  8 21:26:43.484] mgos_vfs_dev_open     sysflash () -> 0x3ffefd4c
[Jul  8 21:26:43.494] mgos_vfs_mount        Mount SPIFFS @ / (dev 0x3ffefd4c, opts {"addr": 32768, "size": 262144})
[Jul  8 21:26:43.549] mgos_vfs_mount        /: size 233681, used: 15060, free: 218621
[Jul  8 21:26:43.624] mgos_sys_config_init MAC: A220A62D7268
[Jul  8 21:26:43.627] mgos_sys_config_init WDT: 30 seconds
[Jul  8 21:26:43.638] mgos_deps_init        init ca_bundle...
[Jul  8 21:26:43.638] mgos_deps_init        init mqtt...
[Jul  8 21:26:43.639] mgos_deps_init        init rpc...
[Jul  8 21:26:43.641] mgos_deps_init        init rpc...
[Jul  8 21:26:43.647] mg_rpc_channel_mqtt   0x3fff06dc esp8266_2D7268/rpc/#
[Jul  8 21:26:43.654] mg_rpc_add_channel_i  0x3fff06dc '*' MQTT, trusted
[Jul  8 21:26:43.655] mgos_deps_init        init i2c...
[Jul  8 21:26:43.659] mgos_i2c_create       I2C GPIO init ok (SDA: 12, SCL: 14)
[Jul  8 21:26:43.663] mgos_deps_init        init atca...
[Jul  8 21:26:43.703] mgos_atca_init        ATECC508 @ 0x60: rev 0x5000 S/N 0x123e94281967668ee, zone lock status: yes,
yes; ECDH slots: 0x0c
[Jul  8 21:26:47.583] mgos_mqtt_global_con MQTT connecting to serveur.iot.com:8883
...
[Jul  8 21:26:47.859] find_mount_by_path    ca.pem -> /ca.pem pl 1 -> 1 0x3ffefd5c
[Jul  8 21:26:47.866] mgos_vfs_open         ca.pem 0x0 0x1b6 => 0x3ffefd5c ca.pem 1 => 257 (refs 1)
[Jul  8 21:26:47.871] mgos_vfs_fstat        257 => 0x3ffefd5c:1 => 0 (size 611)
[Jul  8 21:26:47.876] mgos_vfs_read         257 1024 => 0x3ffefd5c:1 => 611
[Jul  8 21:26:47.886] mgos_vfs_close        257 => 0x3ffefd5c:1 => 0 (refs 0)
[Jul  8 21:26:48.018] ATCA ECDSA verify ok, verified
[Jul  8 21:26:48.024] ssl_socket_recv       0x3ffef2dc <- 5
[Jul  8 21:26:48.028] ssl_socket_recv       0x3ffef2dc <- 149
[Jul  8 21:26:48.161] ATCA ECDSA verify ok, verified
[Jul  8 21:26:48.165] ssl_socket_recv       0x3ffef2dc <- 5
...
[Jul  8 21:26:48.181] ssl_socket_send       0x3ffef2dc 422 -> 422
[Jul  8 21:26:48.312] ATCA:3 ECDH get pubkey ok
[Jul  8 21:26:48.386] ATCA:3 ECDH ok
[Jul  8 21:26:48.391] ssl_socket_send       0x3ffef2dc 75 -> 75
[Jul  8 21:26:48.540] ATCA:0 ECDSA sign ok
[Jul  8 21:26:48.545] ssl_socket_send       0x3ffef2dc 84 -> 84
...
[Jul  8 21:26:48.592] mgos_mqtt_ev          MQTT Connect (1)
```

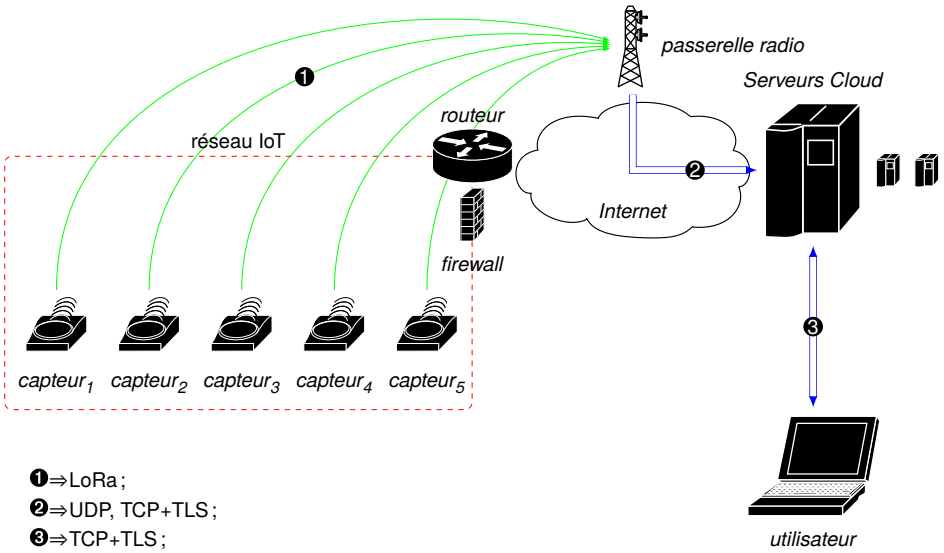
composant ATECC508 reconnu et initialisé.

Temps de connexion ? 1 seconde !

Comparaison des différentes technologies de communication radio

Technology	802.11ah	WLAN	ZigBee	LTE-M	Sigfox	LoRa
Sensitivity	-106 dBm	-92 dBm	-100 dBm	-117 dBm	-126 dBm	-134 dBm
Link Budget	126 dB	112 dB	108 dB	147 dB	146 dB	154 dB
Range (O=Outdoor, I=Indoor)	O: 700m I: 100m	O: 200m I: 30m	O: 150m I: 30m	1.7km urban 20km rural	2km urban 20km rural	3km urban 30km rural
Data rate	100kbps	6 Mbps	250 kbps	1 Mbps	600 bps	12.5 – 0.970 kbps
Tx current consumption	300 mA 20 dBm	350 mA 20 dBm	35 mA 8 dBm	800 mA 30 dBm	120 mA 20 dBm	120 mA 20 dBm
Standby current	NC	NC	0.003mA	3.5mA	0.001mA	0.001mA
RX current	50 mA	70 mA	26 mA	50 mA	10 mA	10 mA
Battery life 2000 mAh				18 months	90 months	105 months
Localization	no	1- 5m	no	200m	no	10-20m
Interference Immunity	moderate	moderate	bad	moderate	bad	good
Network Type	Star	Star	Mesh	Star	Star	Star
End Node Capacity	Large	Medium	Small		(*) 1 Msg per day	>1.3Mu*

Composition d'une infrastructure IoT avec LoRa



❶⇒LoRa ;
❷⇒UDP, TCP+TLS ;
❸⇒TCP+TLS ;

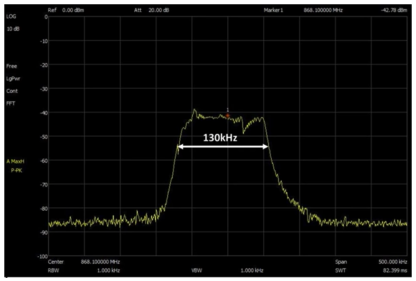
Contournement du firewall de l'utilisateur...plus de passerelle...un lien (presque) direct vers le Cloud !

LoRa vs WiFi

- LoRa ne transmets des données qu'avec un **faible débit** ;
- le **temps de transmission** d'un message dépend essentiellement de la taille de ce message ;
- LoRa utilise différents «*spreading factor*» qui influencent la **portée** de la transmission du signal ;
- SF7 est le plus **rapide** mais aussi le moins «sûr» quand à la fiabilité de la transmission ;
- SF12 est le plus **lent** mais offre une **meilleure portée** ;
- SF12 **consomme plus d'énergie** que SF7 à cause de la durée de transmission du message (la radio reste plus longtemps allumée) ;

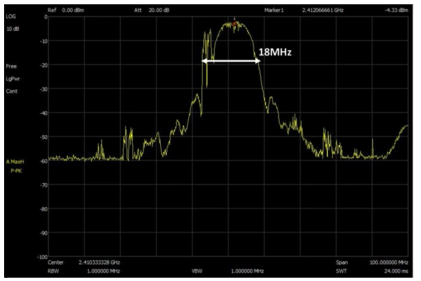
Comparaison entre WiFi et LoRa

LoRa bandwidth :



consommation de 125mA pendant la durée de transmission

WiFi bandwidth :



consommation de 380mA pendant la durée de transmission

LoRa : le composant SX1272

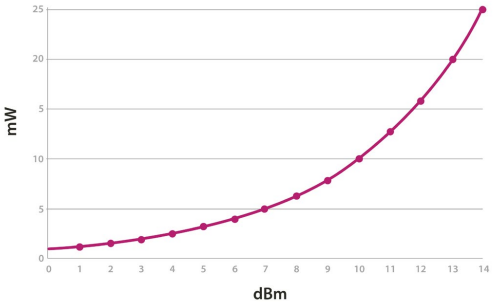
Choix du canal et de la puissance de transmission

Channel Number	Central frequency
CH_10_868	865.20 MHz
CH_11_868	865.50 MHz
CH_12_868	865.80 MHz
CH_13_868	866.10 MHz
CH_14_868	866.40 MHz
CH_15_868	866.70 MHz
CH_16_868	867 MHz
CH_17_868	868 MHz

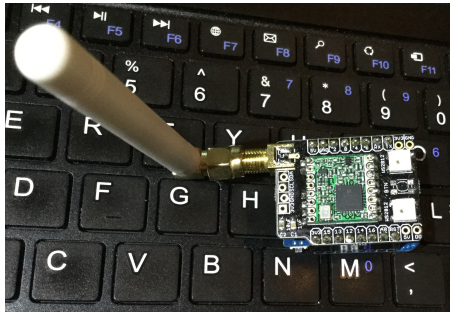
Le WiFi utilise une puissance d'émission de 20dB soient 100mW.

Parameter	SX1272 power level
'L'	0 dBm
'H'	7 dBm
'M'	14 dBm

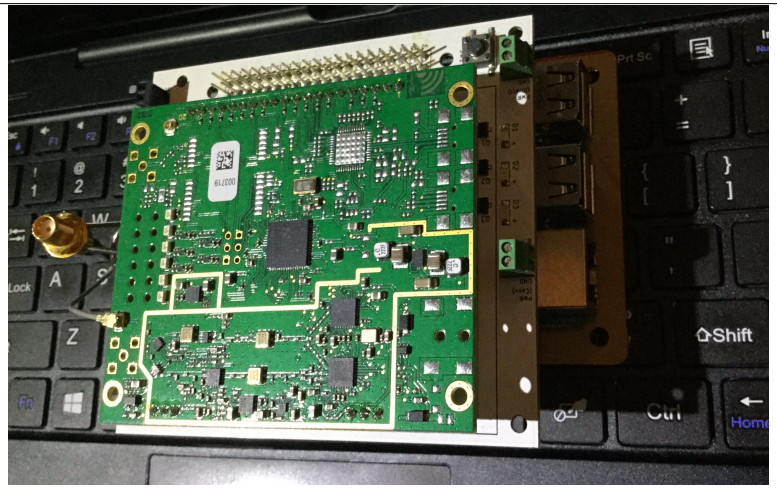
SX1272 output power level



ESP8266+LoRa : un seul canal de communication



Raspberry Pi + LoRa : une passerelle multi-canaux

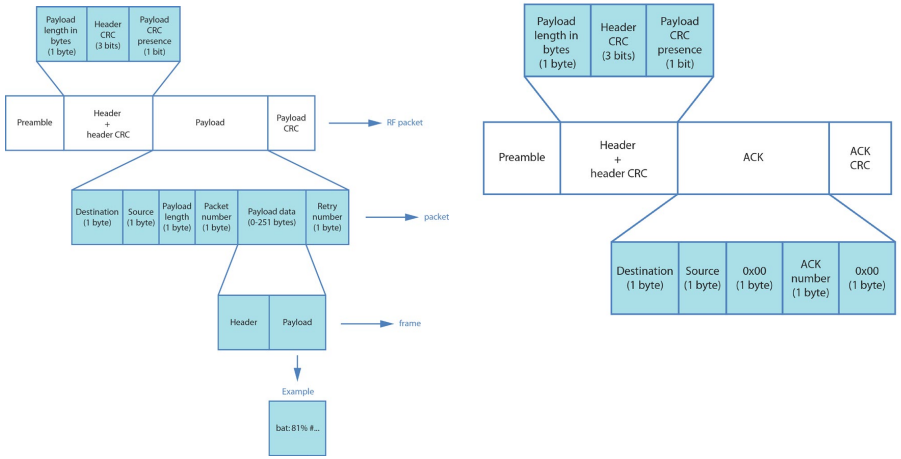


LoRaWAN : MAC

Format des trames échangées dans LoRaWAN

Taille maximale d'une trame : 250 octets

dst	src	packnum	length	data	retry
(1 Byte)	(1 Byte)	(1 Byte)	(1 Byte)	(Variable Bytes)	(1 Byte)

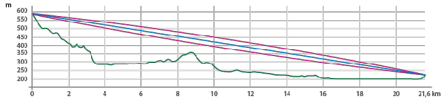


LoRa : test

Coupe du terrain :

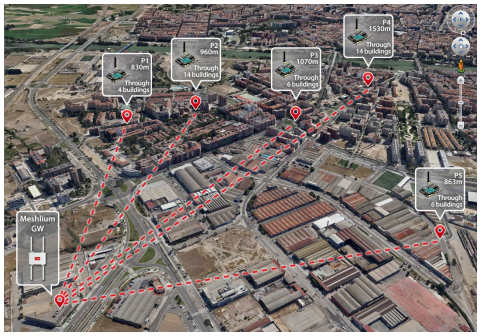
- la ligne bleue représente la ligne de vue ;
- l'ellipse mauve représente la zone de Fresnel ;

On notera qu'il n'y a pas d'obstacles dans la zone, ce qui minimise la FSPL, «Free-Space Path Loss».



LoRa Mode	Range	Power	Channel	Success (%)	Mean SNR (dB)	Mean RSSI (dBm)	Mean RSSI packet(dBm)	Sensitivity (dB)	Margin (dB)
Mode 1	21.6 km (13.4 miles)	High	CH_12_868	100	-9.79	-113.72	-126.79	-134	7.21
		Max		100	-4.33	-113.76	-121.76	-134	12.24
		High	CH_16_868	100	-10.06	-114.28	-127.06	-134	6.94
		Max		100	-3.20	-113.97	-120.21	-134	13.79
Mode 3	21.6 km (13.4 miles)	High	CH_12_868	95	-10.29	-114.16	-127.29	-129	1.71
		Max		95	-3.73	-114.08	-120.73	-129	8.27
Mode 6	21.6 km (13.4 miles)	High	CH_12_868	99	-14.77	-107.22	-125.77	-125.5	-0.27
		Max		100	-8.42	-106.60	-119.43	-125.5	6.07
Mode 9	21.6 km (13.4 miles)	High	CH_12_868	0	-	-	-	-117	-
		Max		49	-9.95	-107.68	-120.95	-117	-3.95

LoRa : test



Les différents points :

1. le signal passe par 4 bâtiments : 3 élevés et un bas, avec un espace ouvert mais pas de LOS ;
2. 14 bâtiments dont un groupe résidentiel ;
3. 6 bâtiments dont des bâtiments industriels ;
4. 14 bâtiments pour le plus long chemin avec des bâtiments résidentiels et industriels et un espace ouvert ;
5. 6 bâtiments industriels et pas d'espace ouvert.

Point	Range (m)	Number of Buildings (signal going through)	Success (%)	Mean SNR (dB)	Mean RSSI (dBm)	Mean RSSI packet (dBm)	Margin (dB)
Point 1	830	4	96	-7.89	-112.95	-124.89	9,11
Point 2	960	14	92	-14.26	-111.26	-131.26	2,74
Point 3	1070	6	98	-3.22	-114.14	-120.24	13,76
Point 4	1530	14	98	-13.16	-112.24	-130.16	3,84
Point 5	863	6	100	-3.42	-113.48	-120.42	13,58

BLE, ou «*Bluetooth Low Energy*»

Bluetooth Dumb *vs* Smart

Rivalité entre WiFi et Bluetooth :

- **WiFi** : maximiser vitesse et débit avec un peu d'économie d'énergie pour l'intégration dans des machines portables :
  ⟹ être capable de streamer de la vidéo, du son *etc.* ;
- **Bluetooth** : «*petits appareils connectés à un téléphone*» avec une très faible consommation d'énergie ;
  ⟹ Oreillettes, enceintes musicales, *etc.*

Caractéristique	Bluetooth classique/Dumb	Bluetooth Low Energy/Smart
Band	2,4GHz	2,4GHz
Distance	30m	50m
Datarate	2100 kBps	260(650)kBps
Tx Power Max	100 dBm	10 dBm
Peak Current Max	30 mA	15 mA
Sleep Current Max	-	1 micro A
Broadcast/Beacon concept	No	Yes
Connect Up+Down	300ms	3ms

BLE, «*Bluetooth Low Energy*» :

- première version : utilisation de trames de 39 octets ;
- version 4.2 : trames étendues à 257 octets.

La version «iBeacon» d'Apple ne permet que de diffuser, «broadcast», et sert uniquement dans le cadre de boutiques ou de musées.

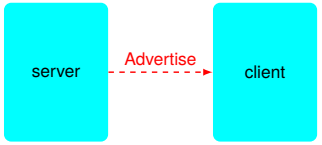
Nº	Nom	Fonction	Usage
1	GATT	API REST	Serveur
2	HTTP	Service Proxy	Client
3	6LoWPAN	tunnel IPv6	Client et Serveur

Le «device» ne se connecte pas directement au «*Cloud*» : il passe par une passerelle : téléphone ou «box».
1 : Le **BLE Gateway**, disposant d'un serveur HTTP, peut être consulté au travers d'Internet⟹il consulte l'IoT grâce au GATT.
2 : l'IoT se **connecte directement** vers le Cloud au travers d'un proxy (qui peut être une passerelle similaire au 1).
3 : **Accès direct en IPv6** à l'IoT : lien vers le Cloud avec tout type de protocole basé IP (éventuellement, une phase GATT permet de configurer l'IP de l'IoT). *L'adresse Ipv6 peut aussi résider seulement sur une passerelle à laquelle l'IoT est relié directement (en bluetooth dumb par exemple).*

BLE, «*Bluetooth Low Energy*»

Deux types de modules :

- serveur : fournisseur de données, il diffuse son service, «*advertisement*» ;
- client : scanne le canal de communication ;



Chaque service est identifié par un UUID, «*Universally Unique ID*».

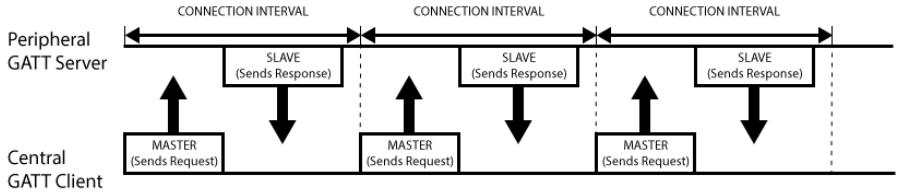
UUID :

- identifie les «*services*», «*characteristics*» et les «*descriptors*» ;
- diffusé par radio :
  - en version courte sur 16bits pour économiser du temps et de l'énergie ;
    Il existe une base d'enregistrement à `https://www.bluetooth.com/specifications/gatt/services`

Heart Rate	org.bluetooth.service.heart_rate	0x180D	GCD

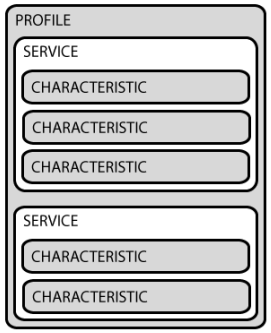
  - en version complète sur 128bits qui identifie de manière unique l'appareil (pas de registre commun d'enregistrement) ;

Modèle «client/esclave»



BLE, «*Bluetooth Low Energy*»

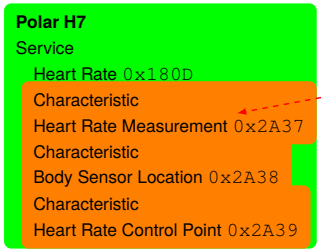
GATT, *Generic Attribute Profile*



- ☐ Chaque «*device*» dispose d'un profile qui définie et organise ses **services**.
- ☐ Chaque service dispose de «*characteristics*» dont la présence, «*requirement*», peut être `optional`, `mandatory`, obligatoire, ou `excluded`.
- ☐ Les «*characteristics*» ont des **propriétés** dont la présence peut être `optional`, `mandatory`, ou `excluded`;

Exemple pour le «*Heart Rate*» service, il dispose de :

– la «*characteristic*» «*Heart Rate Measurement*» :
  - ◇ `mandatory` : obligatoire⟹ cette characteristic est obligatoire dans le cas d'un capteur cardiaque ;
  - ◇ propriétés :
    - ⋆ `Read`, `Write`, `WriteWithoutResponse`, `SignedWrite`, `Broadcast`, *etc.* : `Excluded` ⟹ interdite ;
    - ⋆ `Notify` : `Mandatory` ⟹ le serveur informe à intervalles réguliers ;



Vous trouverez le code pour l'ESP32 d'émulation du capteur cardiaque Polar H7 à l'adresse suivante : `https://github.com/SensorsIot/Bluetooth-BLE-on-Arduino-IDE`

BLE : Découvrir le service

```
xterm
pef@cube:~$ sudo bleah

        .n                  .                  .                  n.
  .   .dP                  dP                  9b                 9b.    .
 4    qXb         .       dX    BLEAH v1.0.0     Xb        .       dXp     t
dX.    9Xb      .dXb      __                                      dXb.    dXP     .Xb
9XXb._       _.dXXXXb dXXXXbo.                 .odXXXXb dXXXXb._       _.dXXP
 9XXXXXXXXXXXXXXXXXXXVXXXXXXXXOo.           .oOXXXXXXXXVXXXXXXXXXXXXXXXXXXXP
  `9XXXXXXXXXXXXXXXXXXXX'~   ~`OOO8b   d8OOO'~   ~`XXXXXXXXXXXXXXXXXXXXXX'
    `9XXXXXXXXXXP' `9XX'     *      `98v8P'      *     `XXP' `9XXXXXXXXXP'
        ~~~~~~~        9X.           .db|db.           .XP        ~~~~~~~
                         )b.  .dbo.dP'`v'`9b.odb.  .dX(
                       ,dXXXXXXXXXXb      dXXXXXXXXXXb.
                     dXXXXXXXXXXP'   .    `9XXXXXXXXXXb
                    dXXXXXXXXXXXb   d|b   dXXXXXXXXXXXb
                    9XXb'    `XXXXXb.dX|Xb.dXXXXX'    `dXXP
                      `'       9XXXXXX(   )XXXXXXP       `'
                               XXXX X.`v'.X XXXX
                               XP^X'`b   d'`X^XX
                               X. 9  `   '  P )X
                               `b  `       '  d'
                                `              '

                  Made with  by Simone 'evilsocket' Margaritelli

@ Scanning for 5s [-128 dBm of sensitivity] ...

 24:0a:c4:81:69:e2 (-63 dBm)
 Vendor                 Espressif
 Allows Connections
 Flags                  LE General Discoverable, BR/EDR
 Tx Power               u'eb'
 Complete 16b Services  u'0d18'
 0x12                   u'20004000'
 Complete Local Name    FT7
```

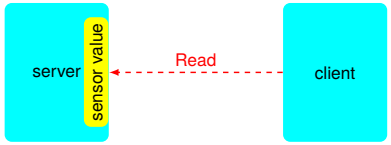
BLE, «*Bluetooth Low Energy*»

```
xterm
pef@cube:~$ sudo bleah -b "24:0a:c4:81:69:e2" -e

        .n                  .                  .                  n.
  .   .dP                  dP                  9b                 9b.    .
 4    qXb         .       dX    BLEAH v1.0.0     Xb        .       dXp     t
dX.    9Xb      .dXb      __                                      dXb.    dXP     .Xb
9XXb._       _.dXXXXb dXXXXbo.                 .odXXXXb dXXXXb._       _.dXXP
 9XXXXXXXXXXXXXXXXXXXVXXXXXXXXOo.           .oOXXXXXXXXVXXXXXXXXXXXXXXXXXXXP
  `9XXXXXXXXXXXXXXXXXXXX'~   ~`OOO8b   d8OOO'~   ~`XXXXXXXXXXXXXXXXXXXXXX'
    `9XXXXXXXXXXP' `9XX'     *      `98v8P'      *     `XXP' `9XXXXXXXXXP'
        ~~~~~~~        9X.           .db|db.           .XP        ~~~~~~~
                         )b.  .dbo.dP'`v'`9b.odb.  .dX(
                       ,dXXXXXXXXXXb      dXXXXXXXXXXb.
                     dXXXXXXXXXXP'   .    `9XXXXXXXXXXb
                    dXXXXXXXXXXXb   d|b   dXXXXXXXXXXXb
                    9XXb'    `XXXXXb.dX|Xb.dXXXXX'    `dXXP
                      `'       9XXXXXX(   )XXXXXXP       `'
                               XXXX X.`v'.X XXXX
                               XP^X'`b   d'`X^XX
                               X. 9  `   '  P )X
                               `b  `       '  d'
                                `              '

                  Made with  by Simone 'evilsocket' Margaritelli

@ Connecting to 24:0a:c4:81:69:e2 ... connected.
@ Enumerating all the things ...

 Handles       Service > Characteristics                                          Properties  Data

 0001 -> 0005  Generic Attribute ( 00001801-0000-1000-8000-00805f9b34fb )
 0003            Service Changed ( 00002a05-0000-1000-8000-00805f9b34fb )          INDICATE

 0014 -> 001c  Generic Access ( 00001800-0000-1000-8000-00805f9b34fb )
 0016            Device Name ( 00002a00-0000-1000-8000-00805f9b34fb )              READ        u'FT7'
 0018            Appearance ( 00002a01-0000-1000-8000-00805f9b34fb )               READ        Unknown
 001a            Central Address Resolution ( 00002aa6-0000-1000-8000-00805f9b34fb ) READ      '\x00'

 0028 -> ffff  Heart Rate ( 0000180d-0000-1000-8000-00805f9b34fb )
 002a            Body Sensor Location ( 00002a38-0000-1000-8000-00805f9b34fb )     READ        '\x02'
 002d            Heart Rate Measurement ( 00002a37-0000-1000-8000-00805f9b34fb )   NOTIFY
```

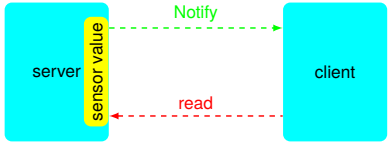
BLE, «*Bluetooth Low Energy*»

❶⇒ le capteur cardiaque «Polar 7» offre son service, :
Le client le découvre.

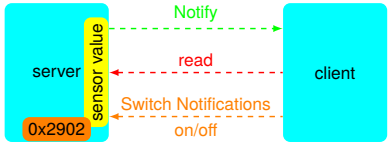
❷⇒ le téléphone et son application mobile lit les données :
Pour la lecture des valeurs du capteur, le client risque de ***gâcher de l'énergie****: il ne sait pas quand le capteur dispose d'une nouvelle valeur.*



❸⇒ Il faut que le client soit «**notifié**» de la modification de la valeur :



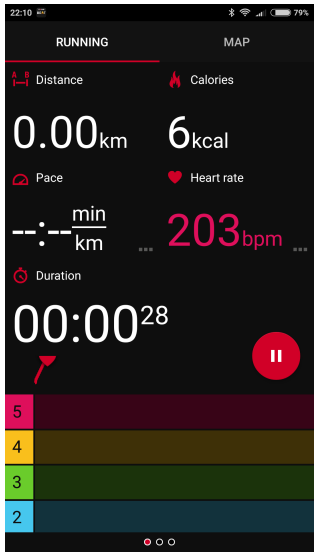
❹⇒ Pour **éviter** que le capteur diffuse ses notifications sans qu'il n'y est de client, il est nécessaire au travers du service d'UUID `2902`, d'activer ou de désactiver ces notifications :



BLE, «*Bluetooth Low Energy*»

201...202...203

Ca fait combien un octet ?

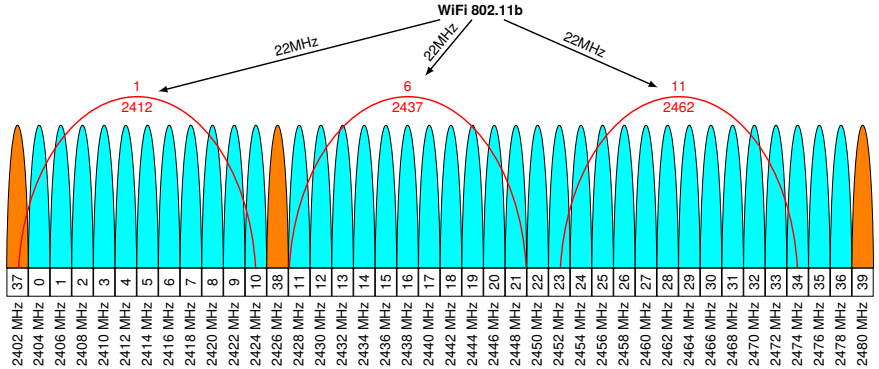


Ici, un ESP32, modèle supérieur à l'ESP8266 disposant d'un module BLE en plus du WiFi, simule une bande de capture du ryhteme cardiaque.
Les valeurs transmises à l'application Android proviennent d'un simple compteur...transmettant un rythme cardiaque de 0 à 255 !

BLE : Radio

BLE utilise 40 canaux de 2400MHz à 2480MHz regroupé en deux types :

- données :
  - de 0 à 36 ;
  - communication bidirectionnelle entre éléments connectés ;
  - saut de fréquence adaptatif : $f_{n+1} = (f_n + \text{hop}) mod 37$ avec hop allant de 5 à 16 ;
- «*advertising*» : 37, 38 et 39 ;

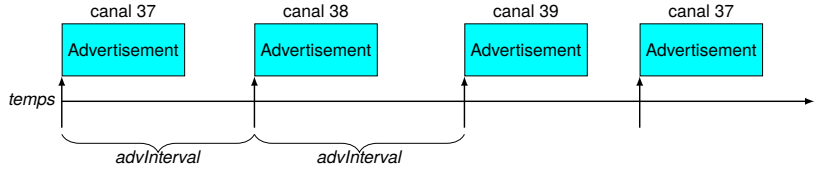


Wifi 802.11b en mode DSSS, des blocs de 22MHz, soient 3 blocs indépendants (sans chevauchement) :

- ▷ Channel 1 : centré sur 2412, de 2401 (2412-11) à 2423 (2412+11) ;
- ▷ Channel 6 : centré sur 2437, de 2426 (2437-11) à 2448 (2437+11) ;
- ▷ Channel 11 : centré sur 2462, de 2451 (2462-11) à 2473 (2462+11) ;

BLE, 3 canaux d'«*advertisement*» : 37, 38 et 39 ;

BLE : Radio



$advInterval = advDelay + advRandom$ avec :

- $advDelay$ de 20ms à 10.24s ;
- $advRandom$ de 0ms à 10ms.

Les sauts de fréquences en BLE