Systèmes embarqués. Conception d'objets connectés

email : kadionik@enseirb-matmeca.fr
web : kadionik.enseirb-matmeca.fr

Patrice KADIONIK
ENSEIRB-MATMECA

HISTORIQUE

- V 1.0 09/02 : Création du document.
- …
- V11.0 09/19 : Suppression partie Linux TR pour intégration dans cours IT332.
- V11.1 11/19 : MAJ et remise en forme.
- V11.2 06/20 : MAJ et remise en forme.
- V11.3 11/20 : MAJ et remise en forme.
- V11.4 05/22 : MAJ.
- V11.4 11/22 : MAJ mineure
- V11.5 11/23 : MAJ.

CHAPITRE 0 :
INTRODUCTION

INTRODUCTION

- Ce module a pour but de présenter les éléments techniques pour appréhender au mieux le monde des systèmes embarqués :
  - Caractéristiques des systèmes embarqués.
  - *Codesign* : développement conjoint matériel/logiciel.
  - Internet embarqué. Importance de l'Internet des objets.
  - Linux embarqué. Les concepts. Le panorama. Mise en œuvre pratique de Linux embarqué sur cible Blackfin.
  - Temps Réel sous Linux. Les concepts. Le panorama.

INTRODUCTION

- Ce module a pour but de présenter les éléments techniques pour appréhender au mieux le monde des objets connectés :
  - Contrôle et communication des objets connectés.
  - Réseau LPWAN LoRaWAN basé sur la technologie LoRa.
  - Conception d'objets connectés pour l'IoT par prototypage rapide.

IoT=*Internet of Things* LPWAN=*Low Power Wide Area Network* LoRa= *Long Range*

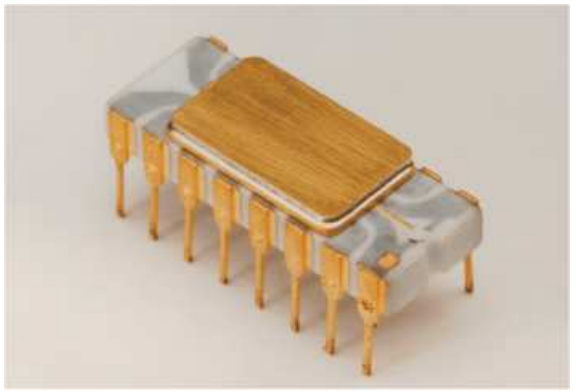
CHAPITRE 1 :
LES SYSTEMES EMBARQUES AUJOURD'HUI

IMPORTANCE DU MARCHE DE L'EMBARQUE

- Les systèmes embarqués ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs.
  - 1971 : premier microprocesseur 4 bits 4004 d'Intel à 750 kHz (2300 transistors) vendu 200 $. Le succès a été là tout de suite.
  - Juin 1978 : premier processeur x86 8086 à 4,77 MHz (technologie 3 µm, 29000 transistors), bus d'adresse 20 bits à 9,1 Mo/s, bus de données 16 bits.
  - Juin 1979 : 8088 intégré dans le premier IBM-PC en 1981.
  - Motorola/Freescale, Zilog, TI ont emboîté le pas…

IMPORTANCE DU MARCHE DE L'EMBARQUE

Calculatrice Busicom 141-PF de Nippon Calculating Machine Corp. (1969)



Processeur 4004 (1971)

IMPORTANCE DU MARCHE DE L'EMBARQUE

- Le marché des microprocesseurs est un marché qui croît de façon exponentielle.

- Trois lois empiriques sont vérifiées depuis 30 ans :
  - Loi de MOORE : pour une surface de silicium donné, on double le nombre de transistors intégrés tous les 18 mois.
  - Loi de JOY : la puissance CPU en MIPS double tous les 2 ans.
  - Loi de RUGE : on a besoin d'une Bande Passante de 0,3 à 1 Mb/s par MIPS.

MIPS=*Million d'Instructions Par Seconde*

- Le marché du microprocesseur a aussi tiré le marché des systèmes embarqués et des télécommunications.

IMPORTANCE DU MARCHE DE L'EMBARQUE

- Grâce aux progrès de l'intégration sur silicium, on est passé rapidement du processeur 4 bits (taille des registres et du bus) au :
  - Processeur 8 bits.
  - au…
  - Processeur 64 bits.

- Il ne faut pas croire que le marché du microprocesseur se résume à celui du PC via les processeurs x86.

IMPORTANCE DU MARCHE DE L'EMBARQUE

- 2010 : 14 milliards de processeurs vendus.
- 2014 : 21 milliards de processeurs vendus.
- 2017 : 28 milliards de processeurs vendus.

- Prix moyen par processeur :
  - 2010 : 1,06 $.
  - 2014 : 0,77 $.
  - 2017 : 0,69 $.

Source : Electronique. Septembre 2014

IMPORTANCE DU MARCHE DE L'EMBARQUE

- Si l'on regarde le prix moyen aujourd'hui d'un processeur tout type confondu, on arrive à 0,69 $ par unité à comparer au prix moyen de 150 $ par unité pour un processeur pour PC.

- Le marché du processeur pour PC est très faible en volume mais extrêmement lucratif !

IMPORTANCE DU MARCHE DE L'EMBARQUE

- En 1999, moins de 10 % des processeurs vendus sont des processeurs 32 bits.
- En 2017, 55 % des processeurs vendus sont des processeurs 32 bits (22 % pour les 16 bits et 23 % pour les 8 bits).
- Cela montre la migration rapide vers les processeurs 32 bits dans l'embarqué, condition nécessaire pour pouvoir mettre en oeuvre un système d'exploitation comme Linux.
- Actuellement, les processeurs 32 bits doivent être la norme. Dans ce cas, il faut avoir le réflexe aussi d'embarquer un véritable système d'exploitation…

LE CHOIX D'UN PROCESSEUR POUR L'EMBARQUE

Embedded Processor	System Requirement	Feature
Microcontroller	I/O Control	I/O Ports with bit-level control
	Peripheral Communication	Serial Ports : SPI, I²C, Microwire, UART, CAN
	Precision control of motors and actuators	Sophisticated timers and PWM peripherals
	Quickly resolve complex software program control flow	Conditional jumps Bit test instructions Interrupt priority control
	Fast response to external events	External interrupts with multiple priority levels
	Conversion of sensor data	Analog-to-Digital (A/D) Converters

LE CHOIX D'UN PROCESSEUR POUR L'EMBARQUE

Embedded Processor	System Requirement	Feature
DSP	Software Filters	Multiply/Accumulate Unit Zero-overhead loops
	Interface to codecs	High-speed serial ports
	High data Throughput from serial ports	Peripheral DMA
	Fast data access	Harvard architectures and variants

DSP=*Digital Signal Processor*

SYSTEME EMBARQUE : DEFINITION

- Un système embarqué peut être défini comme un système électronique et informatique autonome ne possédant pas des entrées/sorties standards.

- Le système matériel et l'application sont intimement liés et noyés dans le matériel et ne sont pas aussi facilement discernables comme dans un PC classique.

Un système embarqué est donc un système électronique et informatique autonome, qui est dédié à une tâche bien précise.

SYSTEME EMBARQUE : DEFINITION

- Un système embarqué :
  - Est un système numérique.
  - Utilise un processeur.
  - N'a pas réellement de clavier standard (BP, clavier matriciel...) ou n'existe pas du tout.
  - L'affichage n'est pas non plus standard (leds, écran LCD…) ou n'existe pas du tout.

SYSTEME EMBARQUE : DEFINITION

- Différences avec un ordinateur de bureau PC :
  - Le logiciel a une fonctionnalité fixe à exécuter et est spécifique à l'application embarquée.
  - Des circuits numériques FPGA ou ASIC sont utilisés en plus pour augmenter les performances du système. C'est le cas pour les systèmes embarqués complexes.
  - L'interface Interface Homme Machine peut être très simple comme très complexe.

FPGA=*Field Programmable Gate Array* ASIC=*Application Specific Integrated Circuit*

TYPES DE SYSTEMES EMBARQUES

Contrôle de systèmes :
Contrôle de systèmes en Temps Réel : automobile, *process* chimique, process nucléaire, aéronautique.

Calcul général :

Application similaire à une application de bureau mais embarquée.

Traitement du signal :

Radar, sonar, vidéo, audio

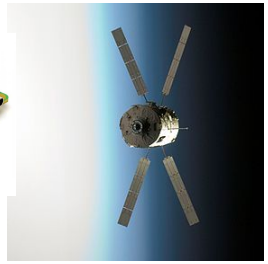
Communications et réseaux :

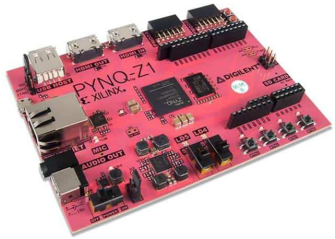
Transmission d'information et commutation.

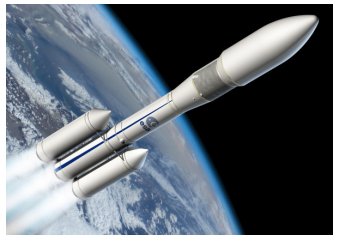
Téléphone, Internet.

EXEMPLES


12°C
Route nationale 20
Preset 3
BNR - Dancefloor
Summary
68 / 100
Favourites

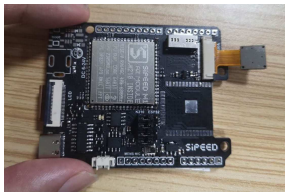



PYNQ-Z1




FriendlyARM


SiPEED


SiPEED

CARACTERISTIQUES D'UN SYSTEME EMBARQUE

- Faible consommation :
  - Autonomie de 8 heures et plus (PC portable : 2 heures) voire 2 à 5 ans pour un objet connecté.
  - Une consommation excessive augmente le prix de revient du système embarqué car il faut alors des batteries de plus forte capacité.

- Faible encombrement, faible poids :
  - Electronique compacte : applications portables où l'on doit minimiser la consommation électrique.
  - Cohabitation dans un volume contraint d'électronique analogique, électronique numérique ou RF sans interférences.

CARACTERISTIQUES D'UN SYSTEME EMBARQUE

- Environnement :
  - Le système n'évolue pas dans un environnement contrôlé.
  - Température, vibrations, chocs, variations d'alimentation, interférences RF, corrosion, eau, feu, radiations.
  - Prise en compte des évolutions des caractéristiques des composants en fonction de la température, des radiations...

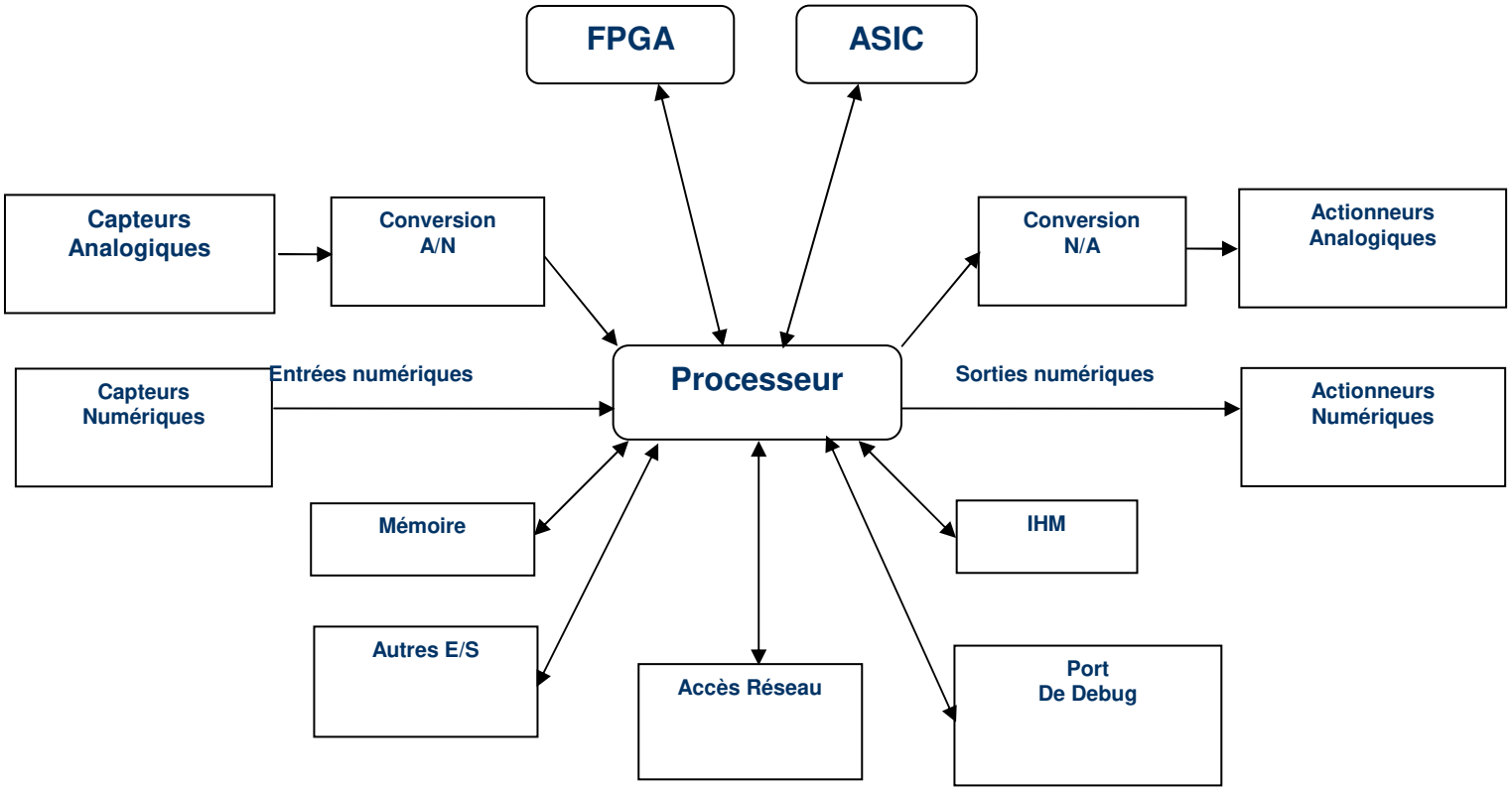
- Fonctionnement critique pour la sécurité des personnes :
  - Le système doit toujours fonctionner correctement et rester opérationnel.

CARACTERISTIQUES D'UN SYSTEME EMBARQUE

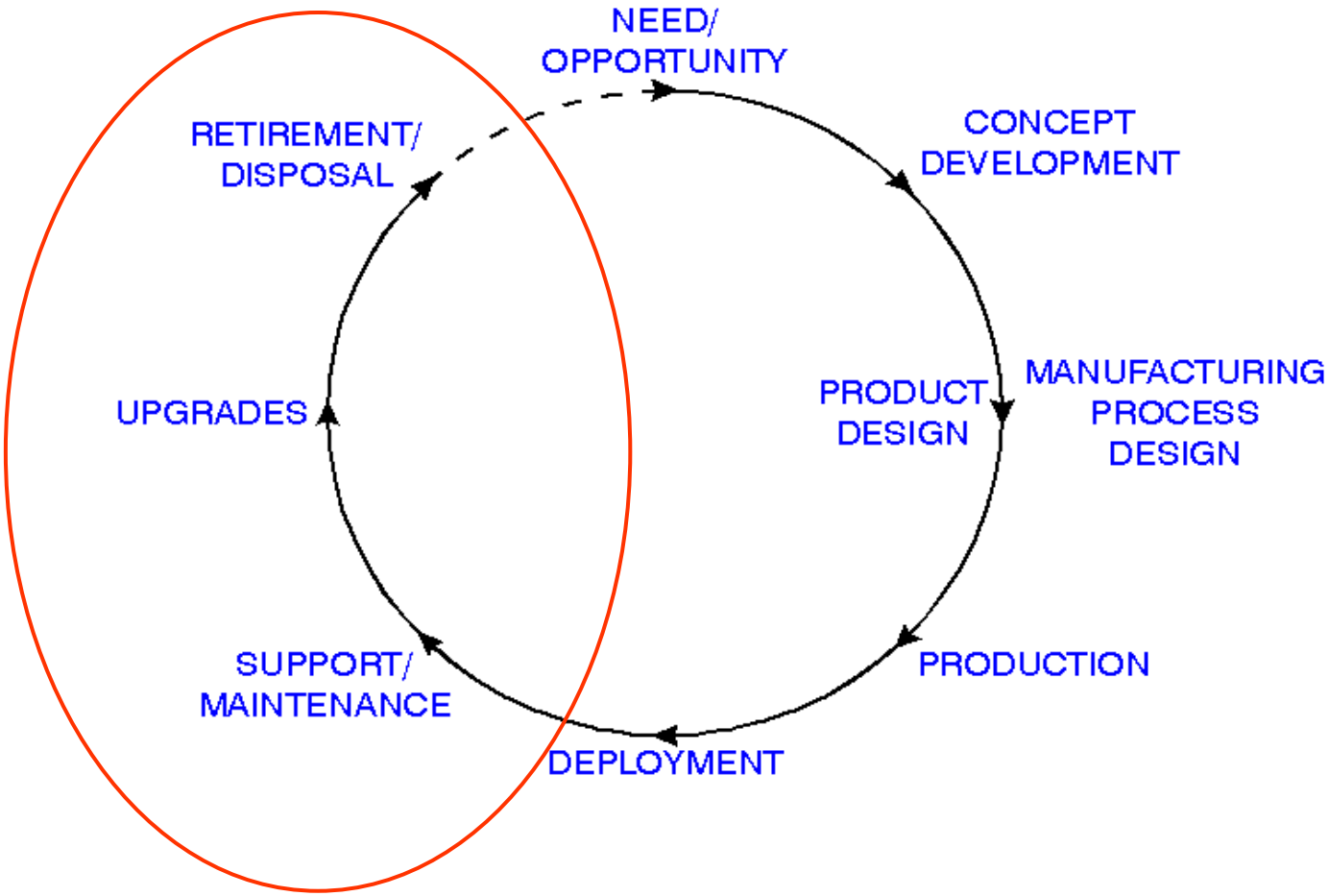
- Beaucoup de systèmes embarqués sont fabriqués en grande série et doivent avoir des prix de revient extrêmement faibles, ce qui induit :
  - Une capacité mémoire au plus juste.
  - Un processeur au plus juste.

- Faible coût du matériel :
  - Optimisation du prix de revient.

SYSTEME EMBARQUE TYPIQUE



CYCLE DE VIE D'UN SYSTEME EMBARQUE



LES SYSTEMES EMBARQUES ET LE TEMPS REEL

- Généralement, un système embarqué doit respecter :
  - Des contraintes temporelles *(Real Time*).
  - On y trouve enfoui un système d'exploitation ou un noyau Temps Réel (dur) (RTOS).

- Le Temps Réel est un concept un peu vague. On pourrait le définir comme :

Un système est dit Temps Réel (dur) lorsque l'information après acquisition et traitement reste encore pertinente.

RTOS=*Real Time Operating System*

LES SYSTEMES EMBARQUES ET LE TEMPS REEL

- Cela veut dire par exemple que dans le cas d'une information arrivant par exemple de façon périodique (sous forme d'une interruption périodique du système), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information pour qu'elle garde son sens.

- Pour cela, il faut que le noyau ou le système Temps Réel soit *déterministe* et *préemptif* (condition nécessaire mais pas suffisante).

ISR=*Interrupt Sub Routine*

LES SYSTEMES EMBARQUES ET LE TEMPS REEL

- Une confusion classique est de mélanger Temps Réel et rapidité de calcul du système donc puissance du processeur.

- On entend souvent :

Être temps Réel, c'est avoir beaucoup de puissance : des MIPS, des MFLOPS….

NON. Il faut répondre au plus juste dans un temps maximum garanti a priori !

MIPS=*Million d'Instructions Par Seconde*

LES SYSTEMES EMBARQUES ET LE TEMPS REEL

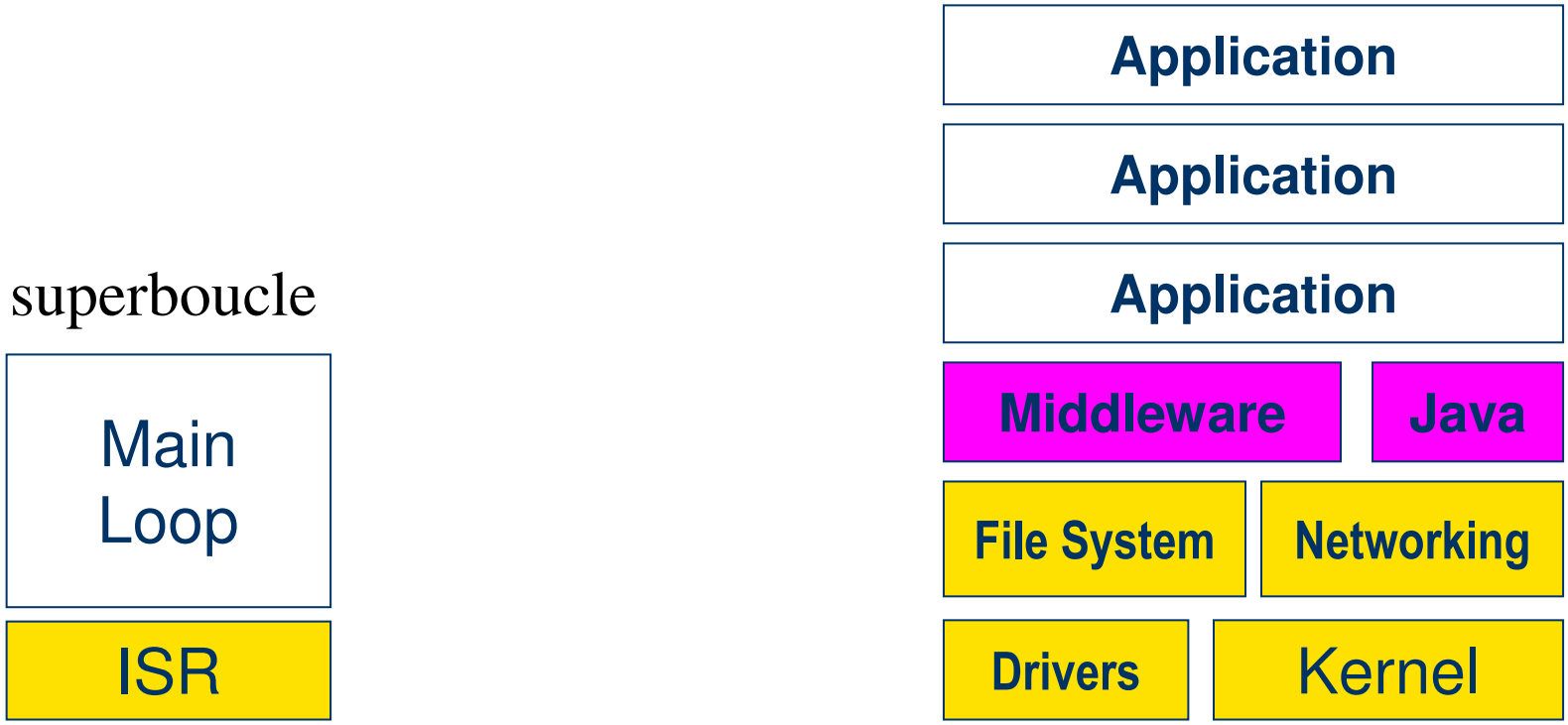
- Fonctionnement en Temps Réel :
  - Réactivité : des opérations de calcul doivent être faites en réponse à un événement extérieur (interruption matérielle).
  - La validité d'un résultat (et sa pertinence) dépend du moment où il est délivré.

- Rater une échéance va causer une erreur de fonctionnement : :
  - Temps Réel dur (*hard Real Time*) : plantage **irréversible**, destruction.
  - Temps Réel mou (*soft Real Time*) : dégradation temporaire **réversible** non dramatique des performances ou du fonctionnement du système.

LES (RT)OS AUJOURD'HUI

- La question d'utiliser un système d'exploitation Temps Réel ou non ne se pose plus aujourd'hui pour des raisons évidentes :
  - Simplification de l'écriture de l'application embarquée.
  - Maintenance.
  - Maîtrise des coûts.
  - …

LES (RT)OS AUJOURD'HUI



ISR=*Interrupt Sub Routine*

LES SYSTEMES EMBARQUES ET LINUX

- Pourquoi retrouve-t-on Linux dans l'embarqué ? Tout d'abord pour ses qualités qu'on lui reconnaît maintenant dans l'environnement plus standard du PC grand public :
  - Stable et efficace.
  - Disponible gratuitement au niveau source.
  - Nombre de plus en plus important de logiciels disponibles.
  - Connectivité IP en standard.

LES SYSTEMES EMBARQUES ET LINUX

- Linux a aussi d'autres atouts très importants pour les systèmes embarqués :
  - Portage sur processeurs autres que x86.
  - Taille du noyau modeste compatible avec les tailles de mémoires utilisées dans un système embarqué.
  - Différentes distributions adaptées à un environnement : routeur (OpenWrt), PDA, téléphone, matériels libres (Raspberry Pi OS)…

LES SYSTEMES EMBARQUES ET LINUX

- On a en fait entendu parler pour la première fois officiellement de Linux embarqué à une exposition *Linux World* en 1999.

- En 2000 a été créé le consortium Linux embarqué (*Embedded Linux Consortium*) dont le but est de centraliser et de promouvoir les développements de solutions Linux embarqué.

CHAPITRE 2 :
INGENIEUR : ETRE CITOYEN ET RESPONSABLE

CHARTE DE L'INGENIEUR

- L'ingénieur est un **citoyen responsable** assurant le lien entre les sciences, les technologies et la communauté humaine ; il s'implique dans les actions civiques visant au bien commun.

- L'ingénieur diffuse son savoir et transmet son expérience au service de la Société.

- L'ingénieur a conscience et fait prendre conscience de **l'impact des réalisations techniques sur l'environnement**.

D'après la charte d'Ethique de l'Ingénieur de l'Institut des Ingénieurs et Scientifiques de France :

home.iesf.fr/offres/doc_inline_src/752/150731_Charte_ethique.pdf

LA FACE CACHEE DU NUMERIQUE

- Le numérique pose des questions environnementales.
- Il faut mesurer les conséquences d'un usage sans fin des services et appareils numériques.
- Voici quelques chiffres pour s'en convaincre :

LA FACE CACHEE DU NUMERIQUE

- 8,9 équipements électroniques par personne en 2021 en Europe occidentale contre 5,3 en 2016.
- 48 milliards d'objets numériques en 2025 : + 5000 % en 15 ans !
- 1 milliard de *smartphones* ont été vendus dans le monde en 2019.
- Une donnée numérique parcourt en moyenne 15000 km.

D'après le document « En route vers la sobriété numérique » de l'ADEME 09/22

LA FACE CACHEE DU NUMERIQUE

- 10 % de la consommation électrique française sont pour les services numériques.

- Internet, c'est 67 millions de serveurs, 1,1 milliards d'équipements réseaux.

D'après le document « En route vers la sobriété numérique » de l'ADEME 09/22

LA FACE CACHEE DU NUMERIQUE

- Internet, c'est 10 à 12 milliards d'emails par heure !
  - 1 email sans pièce jointe équivaut à 4 g de CO_2.
  - 1 email avec pièce jointe équivaut à 35 g de CO_2.
  - 1 requête Web équivaut à 6,65 g de CO_2.
  - 1 tweet équivaut à 0,02 g de CO_2.

- Sur sa durée de vie :
  - 1 ordinateur fixe équivaut à 169 kg de CO_2.
  - 1 ordinateur portable équivaut à 156 kg de CO_2.
  - 1 *smartphone* équivaut à 16,5 kg de CO_2.

D'après le document « En route vers la sobriété numérique » de l'ADEME 09/22

LA FACE CACHEE DU NUMERIQUE

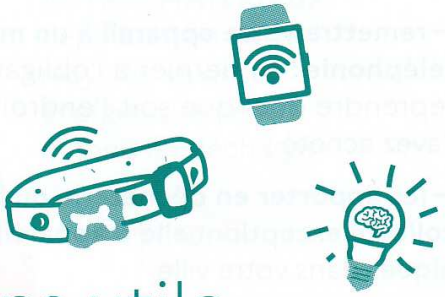
- La consommation annuelle pour gérer le bitcoin correspond à la consommation annuelle d'un pays comme la Suisse.
  - Source : le Monde : www.lemonde.fr/sciences/article/2022/09/24/la-consommation-energetique-annuelle-du-bitcoin-equivalente-a-celle-de-la-suisse-pourrait-etre-divisee-par-mille_6143045_1650684.html

- Les *datacenters* ont utilisé 21 % du total de la consommation électrique mesurée en 2023 en Irlande contre 5 % en 2015 et 18 % en 2022.
  - Source : le Monde : www.lemonde.fr/pixels/article/2024/07/24/en-irlande-la-consommation-electrique-des-centres-de-donnees-depasse-celle-des-maisons-en-ville_6256716_4408996.html

LA FACE CACHEE DU NUMERIQUE

Objets connectés : tous indispensables ?

Le trafic des données représente 55 % de la consommation mondiale annuelle d'énergie du secteur. Réduire son impact environnemental, c'est aussi s'orienter vers ce qui est vraiment utile.



Utile

Le pilotage du chauffage à distance
Couper, baisser, augmenter le chauffage via son smartphone, c'est plus d'économies et de confort !

Certaines applis domotiques
Pratiques pour signaler un dysfonctionnement, un accident domestique ou une panne.

Moins utile

Les LED « intelligentes »
Elles annulent les économies d'énergie propres aux LED car elles consomment en permanence.

Les gadgets en tout genre
Montres, parasols ou brassards connectés, colliers de chiens communicants…: ils consomment de l'énergie et récupèrent vos données personnelles en continu…

En route vers la sobriété numérique | 7 |

D'après le document « En route vers la sobriété numérique » de l'ADEME 09/22

LA FACE CACHEE DU NUMERIQUE

- Pour agir pour la planète, il faut aller vers la sobriété numérique !

« *Science sans conscience n'est que ruine de l'âme* »
François Rabelais (1483-1553)

CHAPITRE 3 :
IMPORTANCE DU CODESIGN DANS L'EMBARQUE

QUAND LE MATERIEL REJOINT LE LOGICIEL

- La capacité de conception des systèmes numériques permet aujourd'hui de tout intégrer dans un même composant (concept du *single chip*).

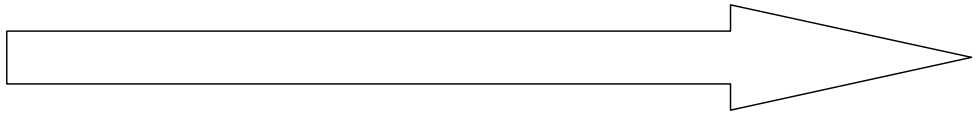
- On travaille donc au niveau système et non plus au niveau porte élémentaire ou schématique. On parle de système sur silicium SoC ou SoPC sur circuit FPGA.

- Ceci est lié à la loi empirique de Moore.

SoC= *System on Chip* SoPC= *System on Programmble Chip*

QUAND LE MATERIEL REJOINT LE LOGICIEL

	1999	2001	2009	2012	2016	2024
Technologie	0,18 µm	0,15 µm	40 nm	20 nm	10 nm	2 nm
Complexité	2M portes	5-10 M	1 G	10 G	10 G+	100 G+
Loi de Moore						



porte=*NAND à 2 entrées*

QUAND LE MATERIEL REJOINT LE LOGICIEL

- On utilise maintenant des langages de description du matériel (VHDL, Verilog) pour synthétiser et aussi tester les circuits numériques.

- On a ainsi une approche logicielle pour concevoir du matériel.

- Avec l'augmentation de l'intégration, les systèmes numériques se sont complexifiés alors que la mise sur le marché doit être la plus rapide possible.

QUAND LE MATERIEL REJOINT LE LOGICIEL

- On a ainsi vu apparaître la notion de blocs IP (*Intellectual Property*) qui est possible par l'utilisation des langages de description du matériel.

- Il existe différents types de blocs IP :
  - Interface (bus CAN…), traitement du signal (FFT, DCT…), processeur, chiffrement…

- Il existe des blocs IP propriétaires que l'on achète.

- Il existe aussi des blocs IP libres :
  - Site communautaire : opencores.org

HARDWARE OU SOFTWARE ?

- La difficulté est maintenant de savoir comment implémenter une fonctionnalité. Exemple d'un algorithme de compression vidéo :
  - Plus rapide par *hardware* mais plus cher.
  - Plus flexible par *software* mais plus lent.

- On doit être capable de jongler en plus avec les paramètres suivants :
  - Coût.
  - Rapidité.
  - Robustesse.

APPROCHE TRADITIONNELLE

- 1. Choix du matériel (composants électroniques, processeur…) pour le système embarqué.
- 2. Donner le système ainsi conçu aux programmeurs.
- 3. Les programmeurs doivent réaliser un logiciel qui « colle » au matériel en n'exploitant que les ressources offertes.

COMPLEXITE GRANDISSANTE

- MAIS les systèmes embarqués sont de plus en plus complexes.
- Il est de plus en plus difficile de penser à une solution globale optimisée du premier jet.
- Il est de plus en plus difficile de corriger les *bugs*.
- Il est de plus en plus difficile de maintenir le système au cours du temps (obsolescence des composants…).
- En conséquence, l'approche traditionnelle de développement d'un système embarqué doit évoluer...

CODESIGN HARDWARE/SOFTWARE

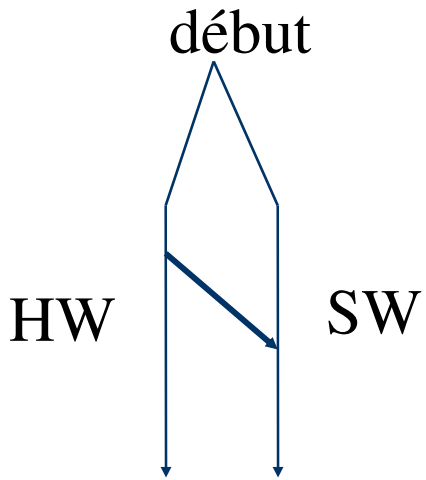
- Le *codesign* dans la méthodologie de conception d'un système embarqué est de plus en plus utilisé notamment pour les systèmes les plus complexes.

- Le *codesign* permet de concevoir en même temps à la fois le matériel et le logiciel pour une fonctionnalité à implémenter.

- Le *codesign* permet de repousser le plus loin possible dans la conception du système les choix matériels à faire contrairement à l'approche classique où les choix matériels sont faits en premier lieu !

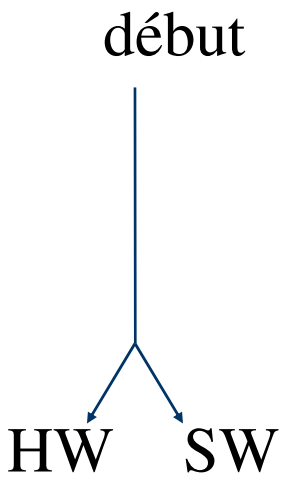
CONCEPTION ET CODESIGN

Conception traditionnelle



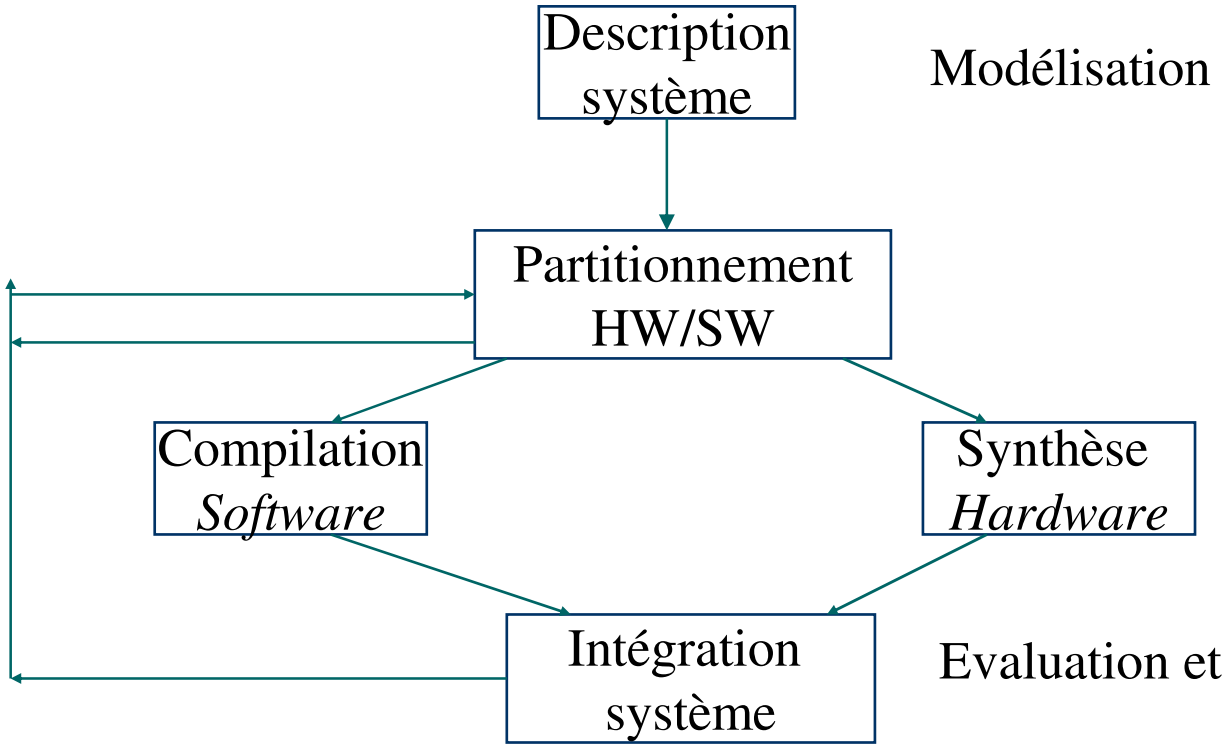
Réalisée par des groupes d'ingénieurs indépendants

Codesign (flot concurrent)

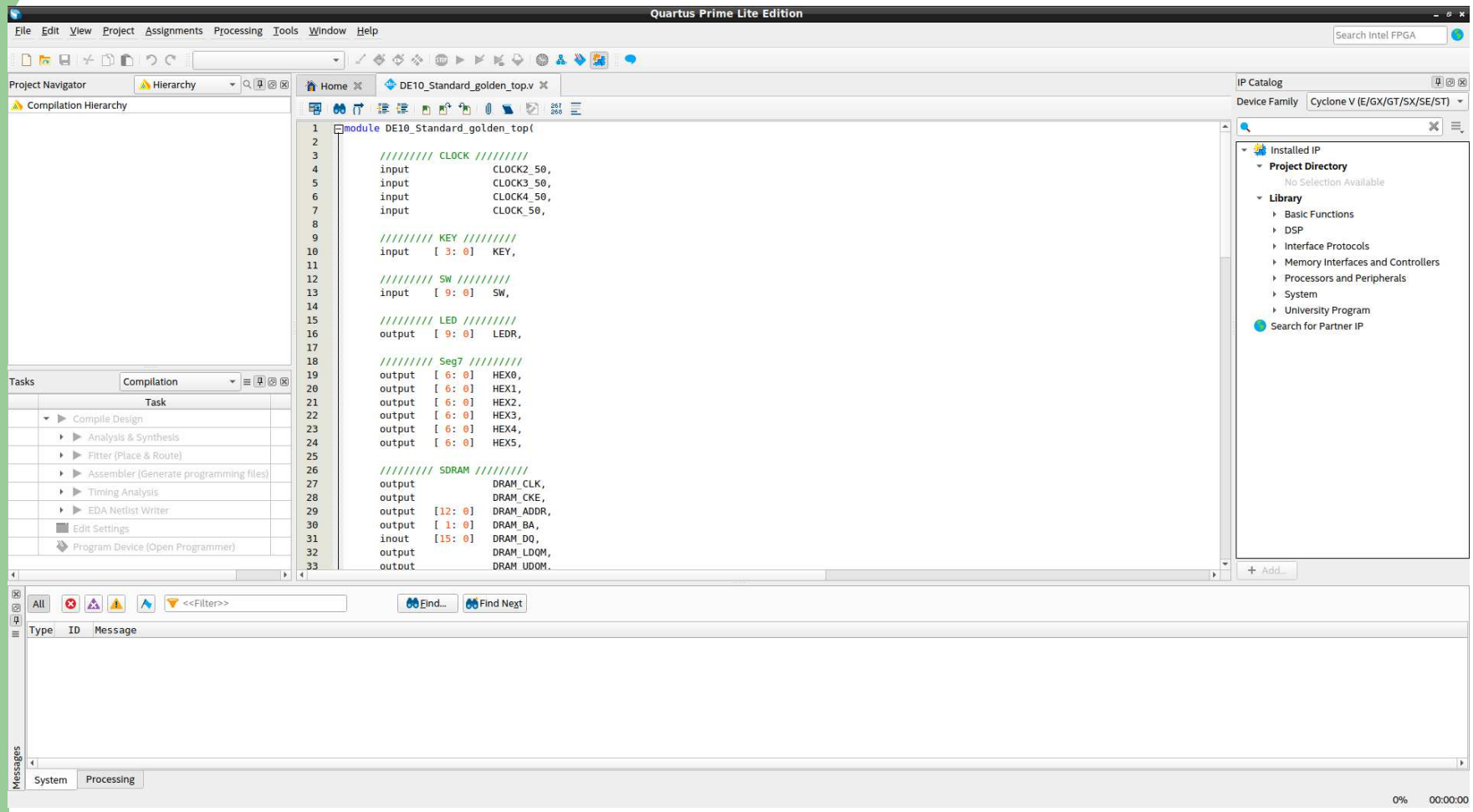


Réalisée par le même groupe d'ingénieurs en coopération

LES ETAPES DANS LE CODESIGN

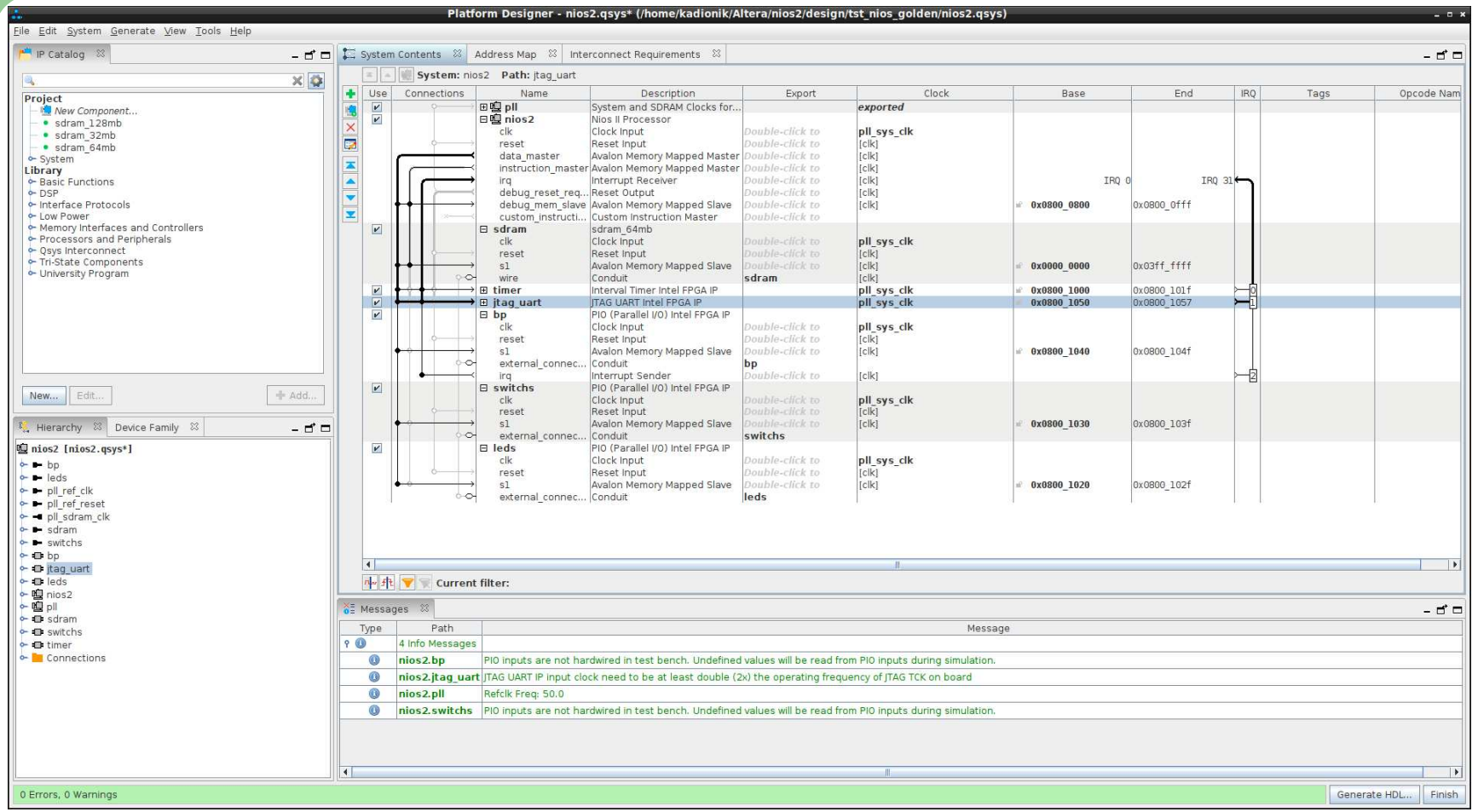


CODESIGN HARDWARE/SOFTWARE



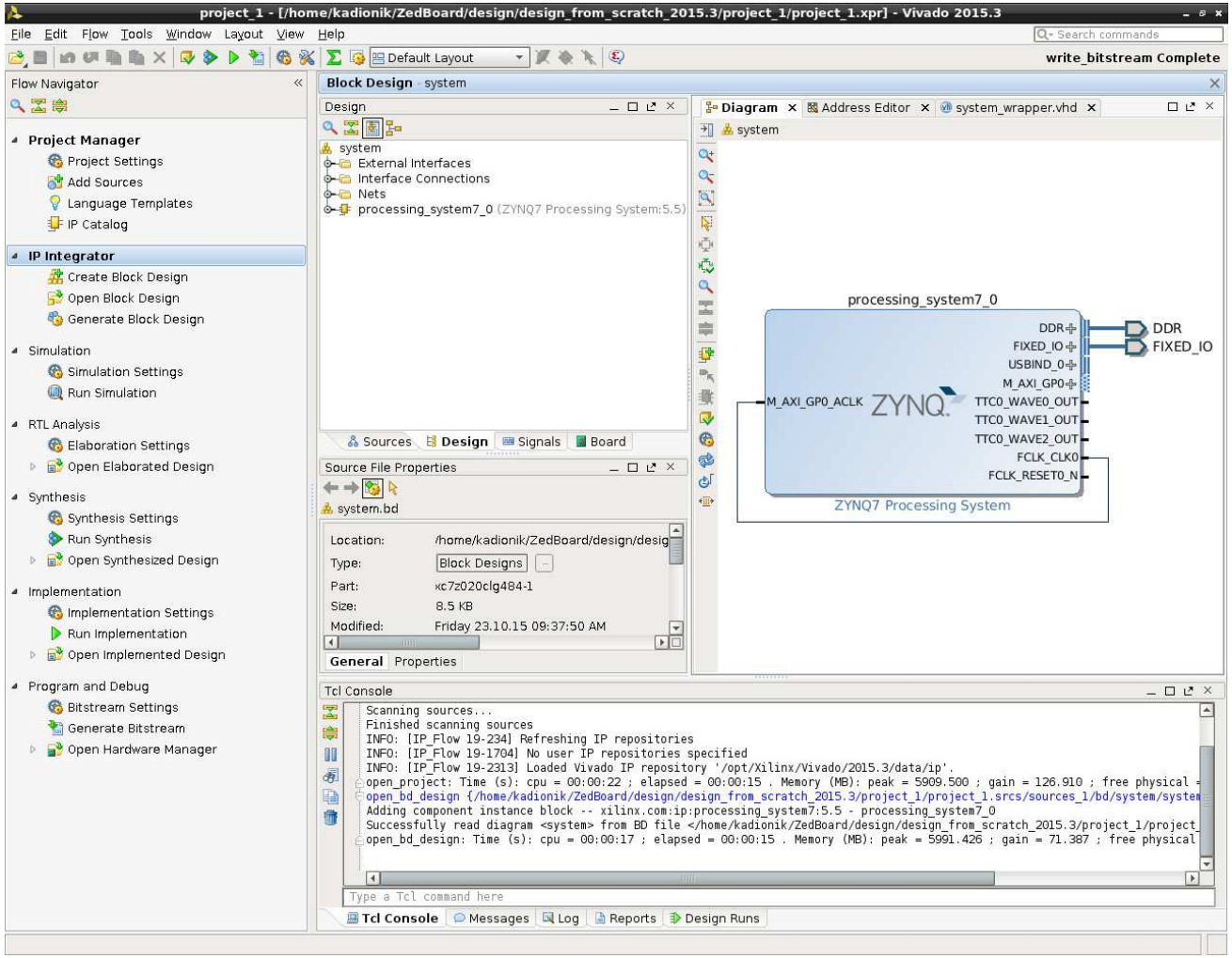
Outil Altera/Intel Quartus Prime

CODESIGN HARDWARE/SOFTWARE



Outil Altera/Intel Quartus Prime

CODESIGN HARDWARE/SOFTWARE



Outil Xilinx/AMD Vivado

AVANTAGES DU CODESIGN

- Le *codesign* est donc intéressant pour la conception des systèmes embarqués complexes :
  - Amélioration des performances : parallélisme, architecture spécialisée…
  - Indépendance vis à vis des évolutions technologiques des circuits logiques programmables.
  - Mise à profit des améliorations des outils de conception fournis par les fabricants de circuits logiques programmables : synthèse plus efficace, performances accrues.

CHAPITRE 4 :
QUELLE APPROCHE POUR CONCEVOIR UNE FONCTIONNALITE ?

QUELLE APPROCHE ?

- On vient de voir que l'on avait différentes approches pour concevoir son système embarqué :
  - Logique câblée : conception purement matérielle d'une fonctionnalité spécifique.
  - Logique programmée : conception purement logicielle d'une fonctionnalité (qui s'appuie sur une base matérielle standard).
- Le *codesign* est une combinaison maîtrisée d'un mélange savant de logique câblée et de logique programmée.
- Mais il y a une troisième voie ? Laquelle ?

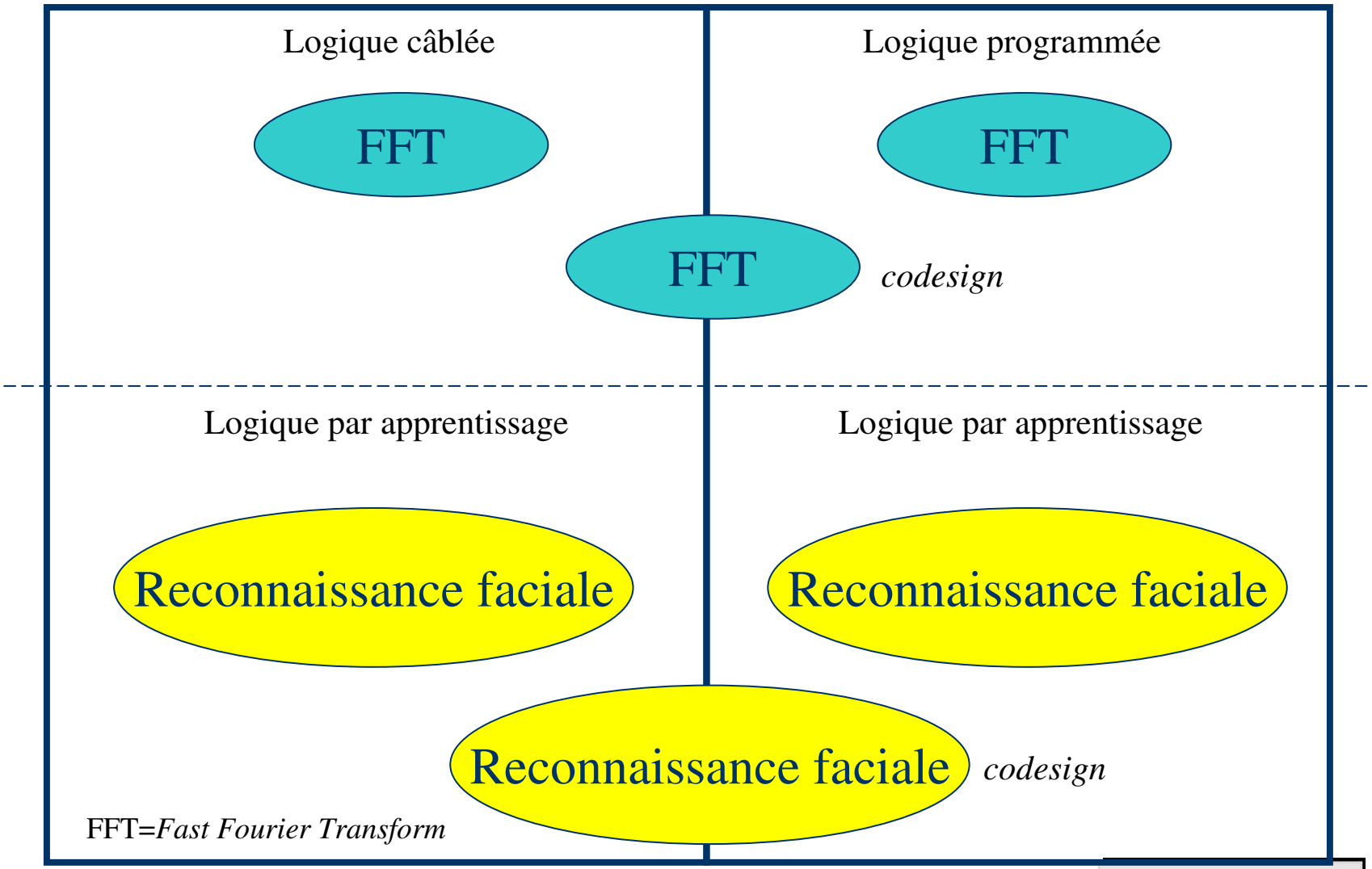
QUELLE APPROCHE ?

- La logique par apprentissage !
- La logique par apprentissage inclut le *Machine Learning* (ML) et le *Deep Learning* (DL).
- La logique par apprentissage peut reposer sur une logique programmée et/ou sur une logique câblée.

QUELLE APPROCHE ?

- Le choix de la « bonne » approche parmi les 3 possibles dépendra des contraintes que doit respecter la fonctionnalité du système embarqué comme :
  - Performances.
  - Consommation.
  - Temps Réel.
  - Coûts.
  - …

QUELLE APPROCHE ?



CHAPITRE 5 :
INTERNET EMBARQUE
INTERNET DES OBJETS

CONNECTIVITE INTERNET

- La connectivité Internet permet de raccorder tout système embarqué au réseau Internet. On parle aussi de connectivité IP.

- La connectivité IP demande d'embarquer une suite de protocoles Internet sur le système électronique pour pouvoir être mise en œuvre.

- On parle alors de protocoles Internet embarqués (sur le système) ou plus simplement d'Internet embarqué.

CONNECTIVITE INTERNET

- La suite des protocoles IP à embarquer est plus ou moins importante en fonction du service à implanter :
  - Contrôle par une application réseau spécifique (API *sockets*) : action synchrone.
  - Contrôle par le web : action synchrone.
  - Envoi d'*emails* : action asynchrone.
  - Autres protocoles détournés :
    - SNMP (*Simple Network Management Network*) : actions synchrone et asynchrone.
    - SIP (*Session Initiation Protocol*) : actions synchrone et asynchrone.

API=*Application Programming Interface*

CONNECTIVITE INTERNET

- L'Internet embarqué est apparu fin années 1990. Il est grandement développé conjointement avec l'usage de Linux pour l'embarqué qui intègre la connectivité IP en standard…

- L'Internet embarqué a explosé depuis quelques années par le développement de l'Internet des Objets IoT (*Internet of Things*).

- Il est aussi important que la connectivité IP se fasse par une liaison sans fil avec l'objet à contrôler…

INTERNET DES OBJETS

- Selon l'Union Internationale des Télécommunications, l'IoT peut aussi se définir comme :

  « a global infrastructure for the Information Society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies »

INTERNET DES OBJETS

- L'IoT correspond à une opportunité répondant à un nouveau besoin du moment :
  - La mise en œuvre d'un ensemble de technologies déjà éprouvées : connectivité IP, Linux embarqué, Java embarqué, *middlewares*, *frameworks*…
  - Un saut technologique : réseaux sans fil LPWAN (LoRa, Sigfox).
  - Un nouveau besoin qui met en œuvre ces technologies.

- Il s'était déjà produit la même chose avec le Web début années 1990. On peut situer le démarrage de l'IoT dans les années 2005-2010…

LPWAN=Low Power Wide Area Network

INTERNET DES OBJETS

- L'Internet des objets, c'est :
  - 80 milliards d'objets connectés en 2020 :
    - 85 % d'objets grand public.
    - 11 % de terminaux mobiles (*smartphones*).
    - 4 % de M2M (*Machine to Machine*).
    - 11,5 milliards d'euros de CA sur 10 ans.

- Domaines où l'on retrouve des objets connectés :
    - Santé.
    - Sécurité.
    - Maison connectée.
    - Mobilité.

INTERNET DES OBJETS

- Au début de l'IoT, le développement des objets connectés était plutôt complexe car :
  - Il faisait appel à des environnements de développement propriétaires.
  - Le développement était bas niveau : langage C, assembleur…
  - Les objets connectés utilisés des bus ou réseaux de communication qui ne permettaient pas une connexion directe au réseau Internet.
  - Il fallait mettre en œuvre une passerelle bus_objets/Internet.
  - Il était très difficile voire impossible de faire communiquer des objets connectés de différents constructeurs.

INTERNET DES OBJETS

- Depuis quelques années (2010-2015), le développement des objets connectés s'est grandement simplifié car :
  - Il fait appel à des réseaux et bus standardisés (LoRa, ZigBee (IEEE 802.15.4), BLE…).
  - La connectivité à Internet est directement intégrée dans les objets connectés.
  - Il fait appel à des logiciels libres.
  - Il met en œuvre des *middlewares* pour cacher les spécificités matérielles et logicielles des objets connectés.
  - Il met en œuvre des canevas de programmation ou *frameworks* pour faciliter le développement logiciel et le déploiement des objets connectés.

BLE=Bluetooth Low Energy

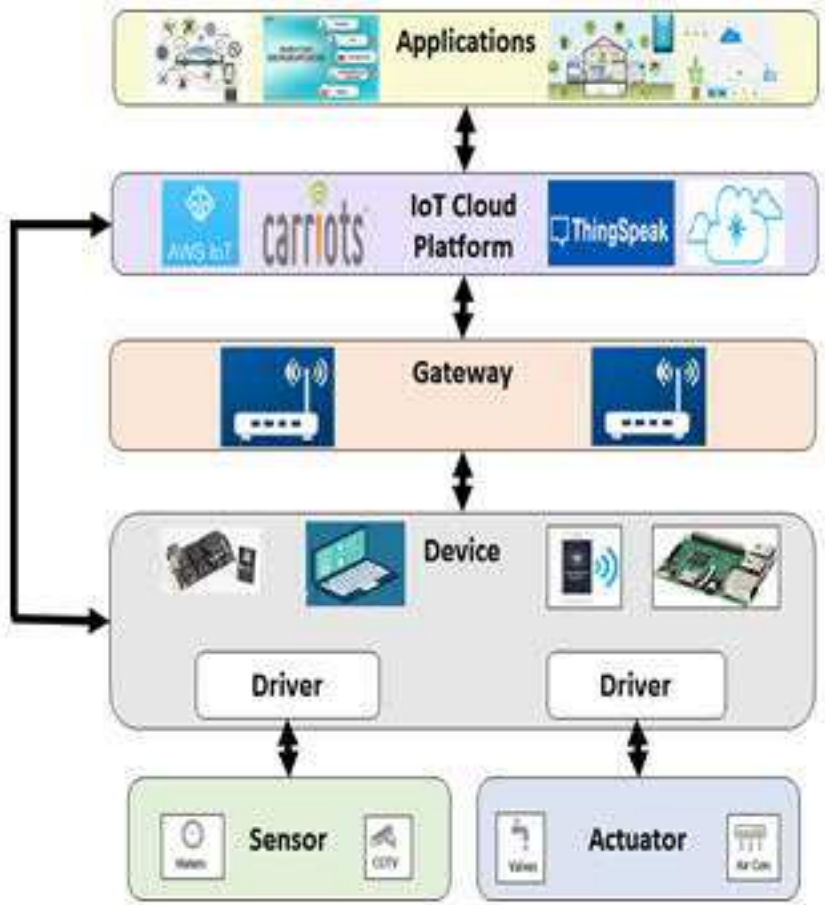
INTERNET DES OBJETS

- L'architecture de l'IoT consiste en un continuum d'équipements (*device*) ou objets interconnectés utilisant principalement des technologies de transmission sans fil.

- Les objets sont généralement très hétérogènes.

- Cela impose alors d'unifier l'ensemble par l'usage de couches logicielles d'abstraction pour assurer l'interopérabilité mais aussi cacher les spécificités techniques bas niveau.

INTERNET DES OBJETS



Architecture de l'IoT

INTERNET DES OBJETS

- Les capteurs (*sensor*) et actionneurs (*actuator*) permettent d'interagir avec le monde réel : acquisition de grandeurs physiques en entrée (volume, vitesse, température…) et pilotage de grandeurs en sortie.

- L'objet (*device*) est l'équipement HW/SW qui intègre les capteurs et actionneurs. C'est l'intelligence locale (*edge computing*) de contrôle. L'objet s'interface à une passerelle vers Internet par une connectivité Internet ou non.

- La passerelle (*gateway*) permet de se connecter à Internet.

INTERNET DES OBJETS

- Le *cloud* IoT va collecter les données des différents objets et les contrôler. Il fournit les données aux applications IoT. Cela s'intègre dans une plateforme de développement (*framework*) fournissant un *middleware* (*middleware layer*).

- Les applications IoT implémentent les services IoT (*smart city*, *smart industry*…).

INTERNET DES OBJETS

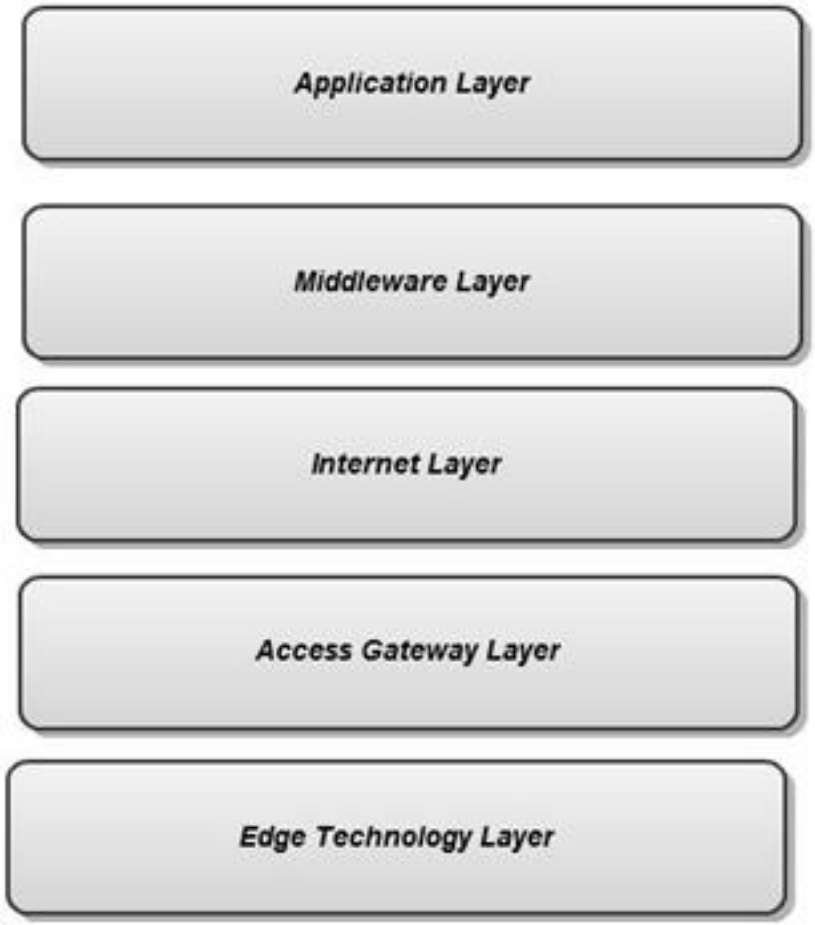
- Il y a deux éléments importants dans l'Internet des objets pour le grand public notamment pour la domotique :
  - La *box* ADSL : connexion permanente à Internet.
  - Le *smartphone* : application *user centric* de contrôle à distance.

INTERNET DES OBJETS

Internet

INTERNET DES OBJETS

- On peut aussi raisonner en couches dans l'architecture IoT :



INTERNET DES OBJETS

- Couche *edge technology* : c'est la couche matérielle qui traite les entrées/sorties et gère les communications.
- Couche *access gateway* : c'est la couche qui gère le transport des données et leur routage vers Internet.
- Couche *middleware* : c'est la couche qui gère le traitement de données et fournit aux applications l'accès aux objets.
- Couche *application* : C'est la couche qui fournit différents services aux applications IoT.

OPTION SE

Voir le cours « Internet des objets et Middleware »

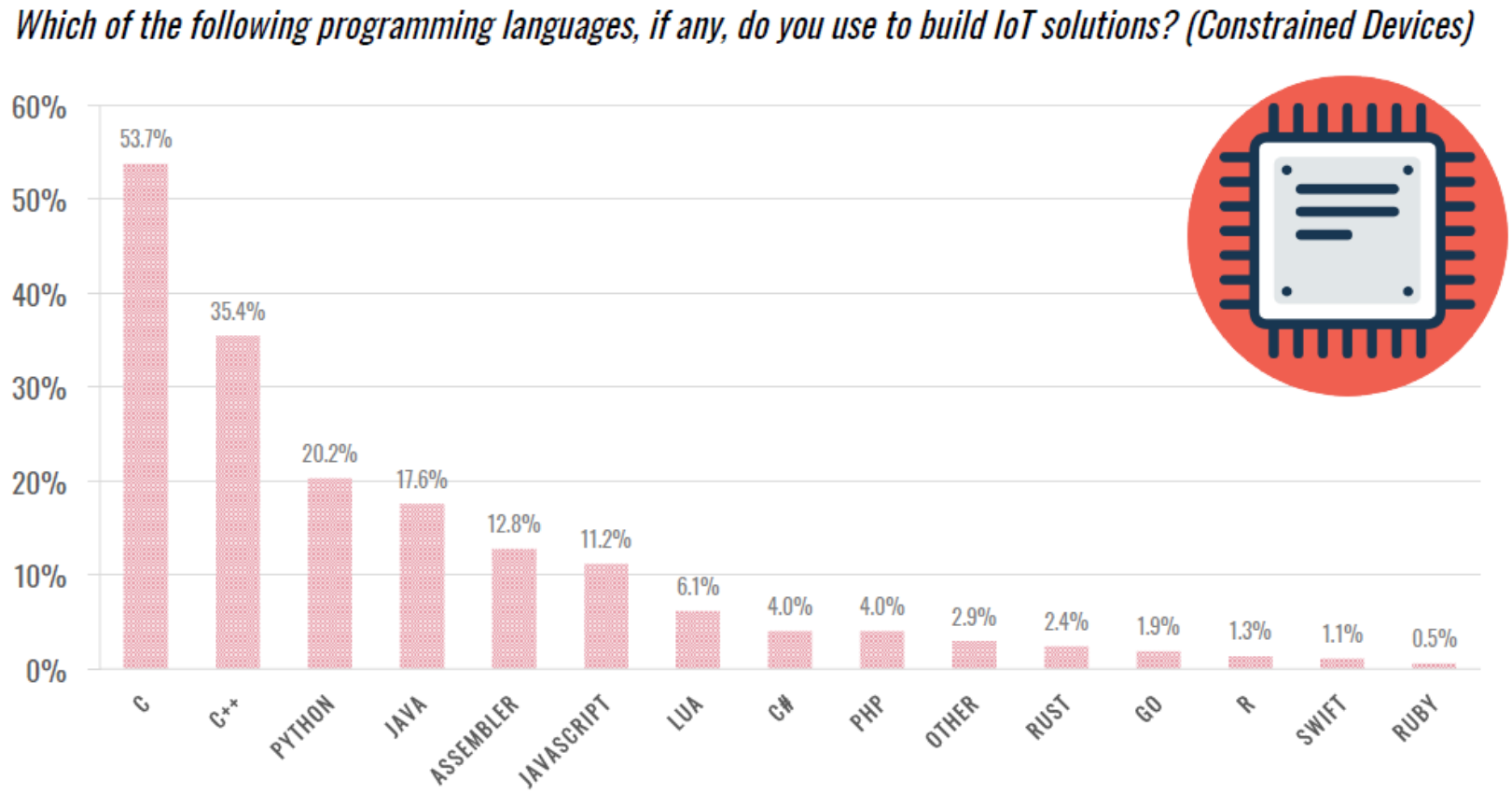
TECHNOLOGIES POUR L'IOT

- Les technologies mise en œuvre dans l'Internet des objets sont déjà existantes et pour certaines existantes depuis des décennies ont retrouvé un regain d'intérêt.

- On s'appuiera sur les résultats de l'enquête « *IoT Developer Survey* » de 2018.

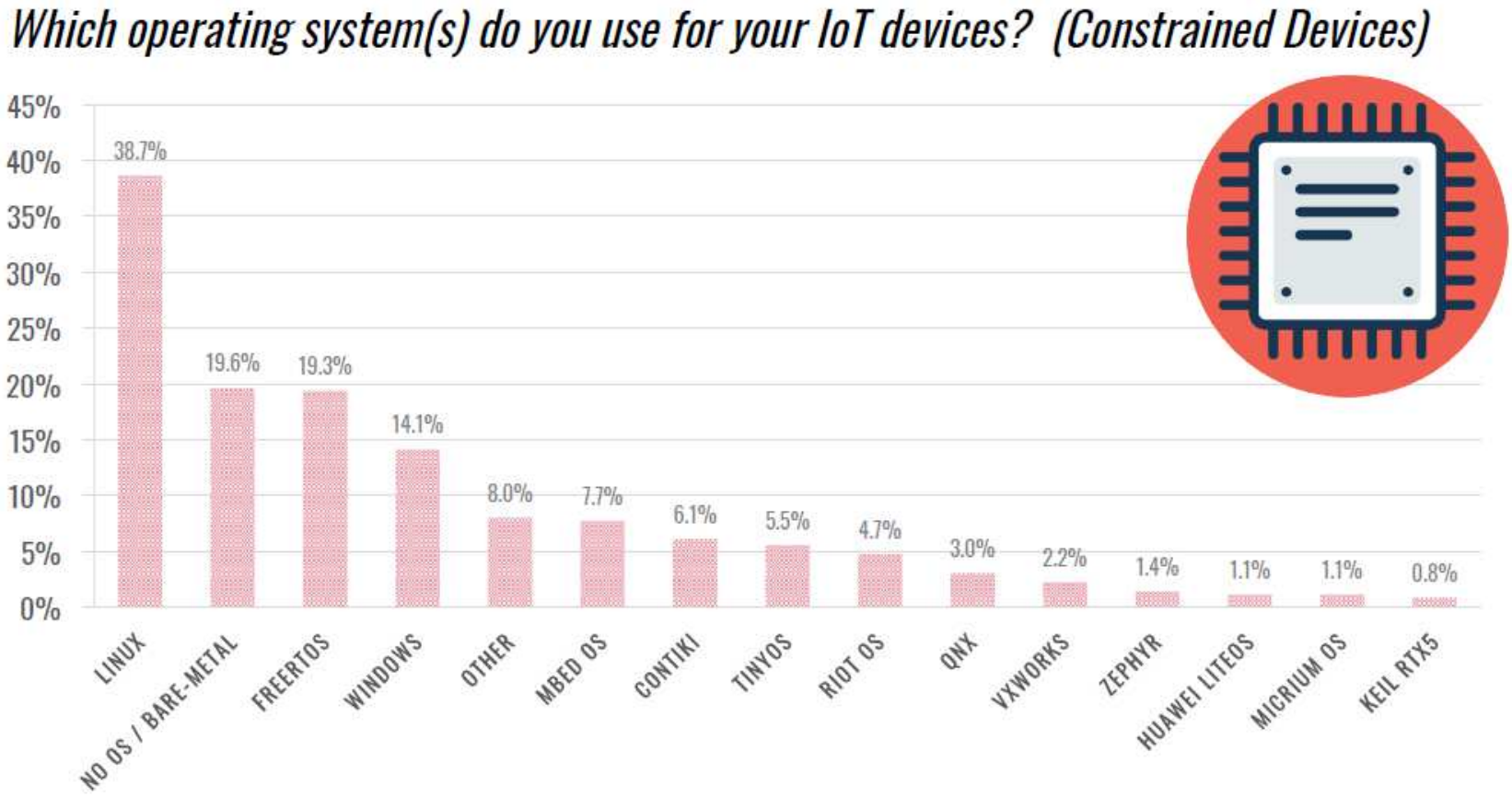
TECHNOLOGIES POUR L'IOT

- Langages de programmation :
  - Java, C, Python, Java Script.



TECHNOLOGIES POUR L'IOT

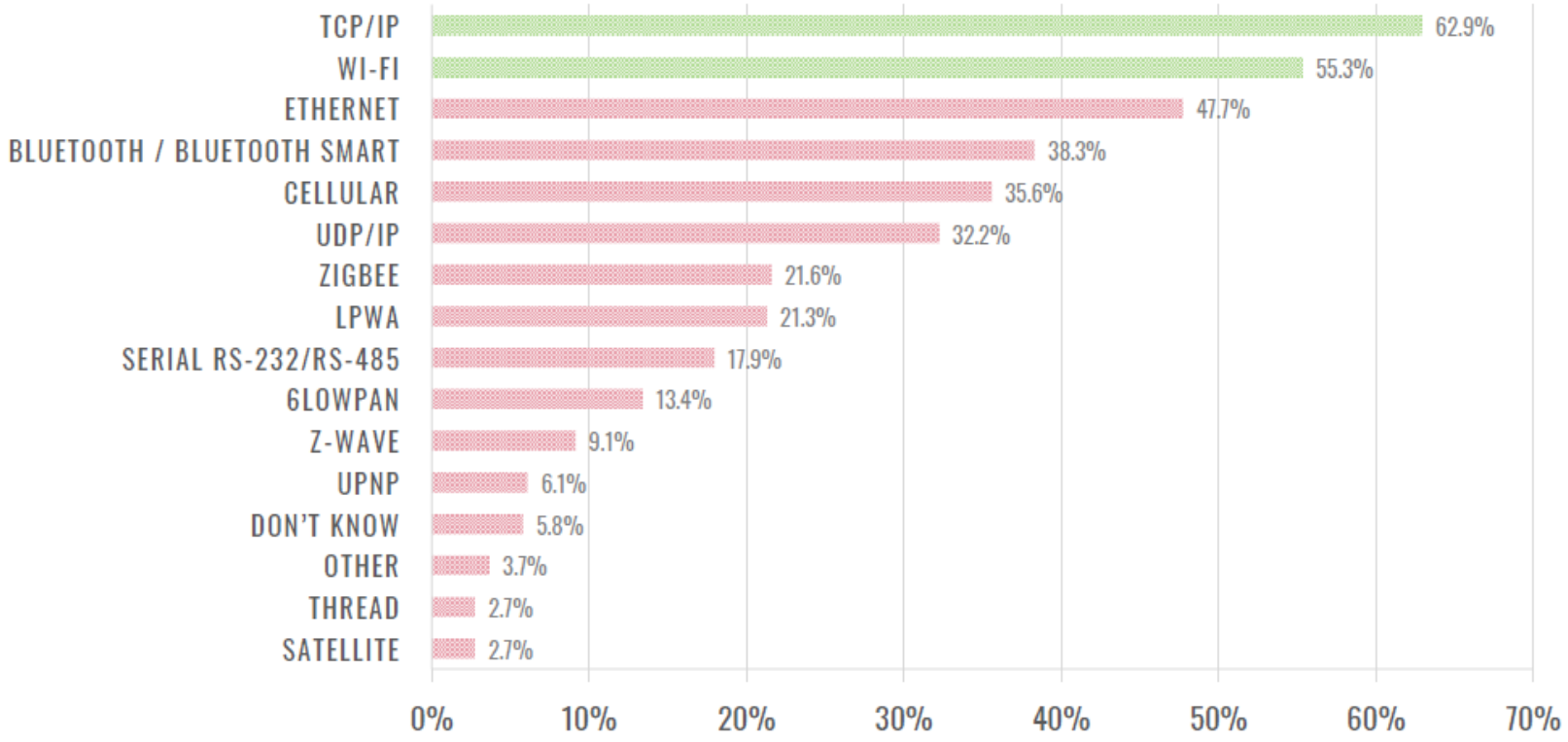
- Systèmes d'exploitation :
  - Linux, *Bare-metal*, FreeRTOS.



TECHNOLOGIES POUR L'IOT

- Protocoles pour connectivité :
  - TCP/IP, WiFi, Ethernet.

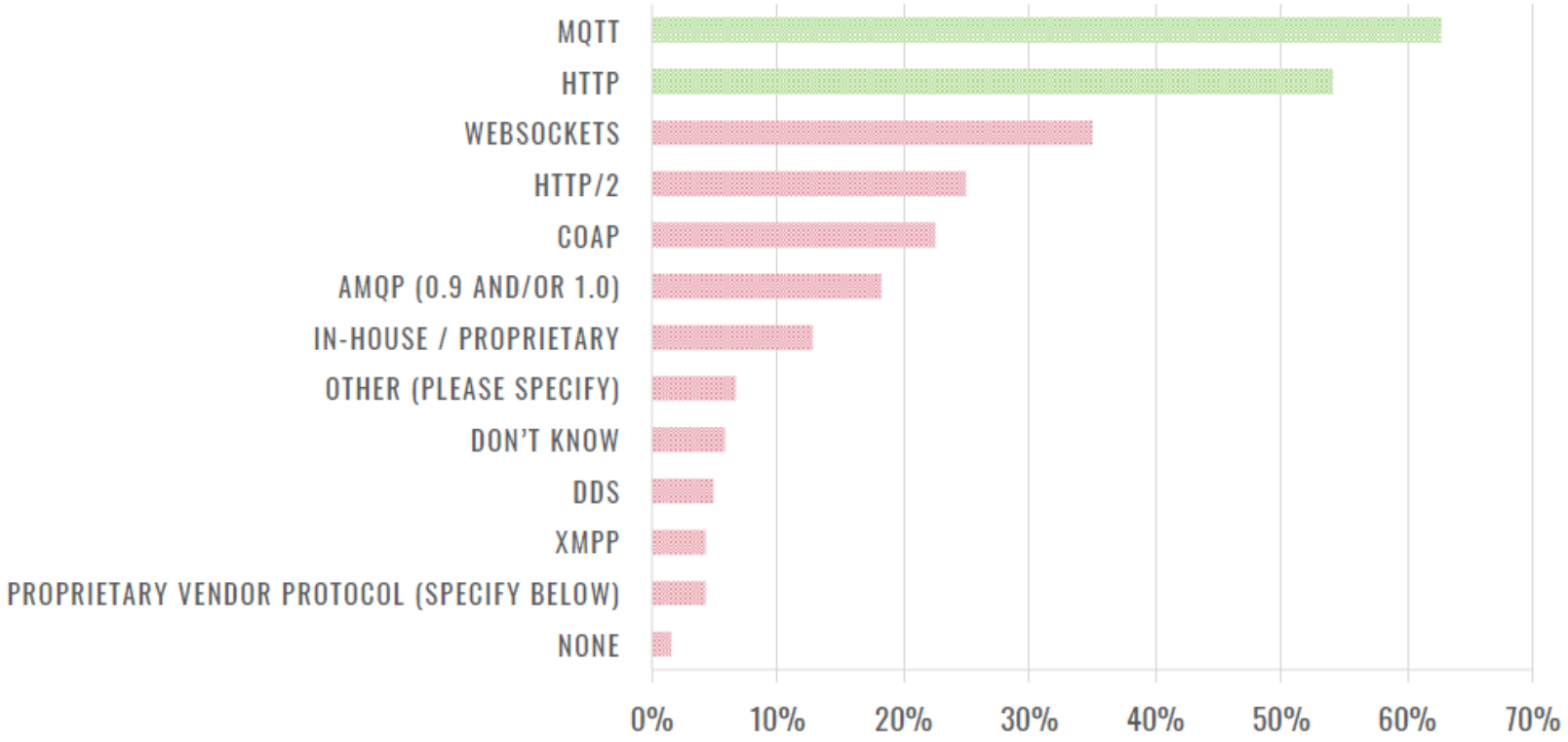
What connectivity protocol(s) do you use for your IoT solution?



TECHNOLOGIES POUR L'IOT

- Protocoles pour échanges de messages :
  - MQTT, HTTP.

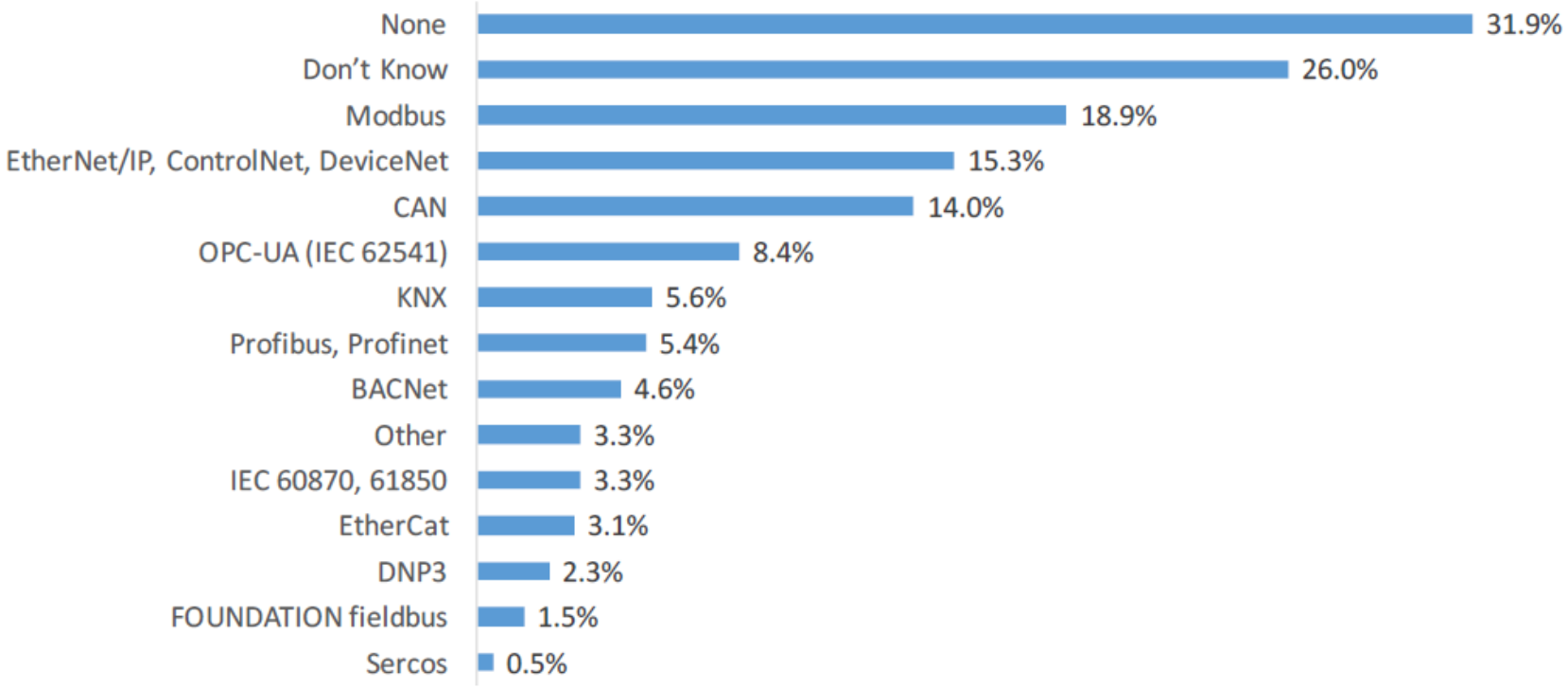
What messaging protocol(s) do you use for your IoT solution?



TECHNOLOGIES POUR L'IOT

- Réseaux industriels :
  - Aucun, ne sait pas.

What industrial protocol(s) do you use in your IoT solution?



TECHNOLOGIES POUR L'IOT

- Il faut enfin noter l'importance de l'usage du Libre dans l'IoT :
  - 58 % des sondés participent à des projets open source pour l'IoT.
  - 52 % utilisent du matériel libre (Raspberry Pi, Arduino, BeagleBone…) dans leur projet IoT pour le prototypage ou le déploiement.

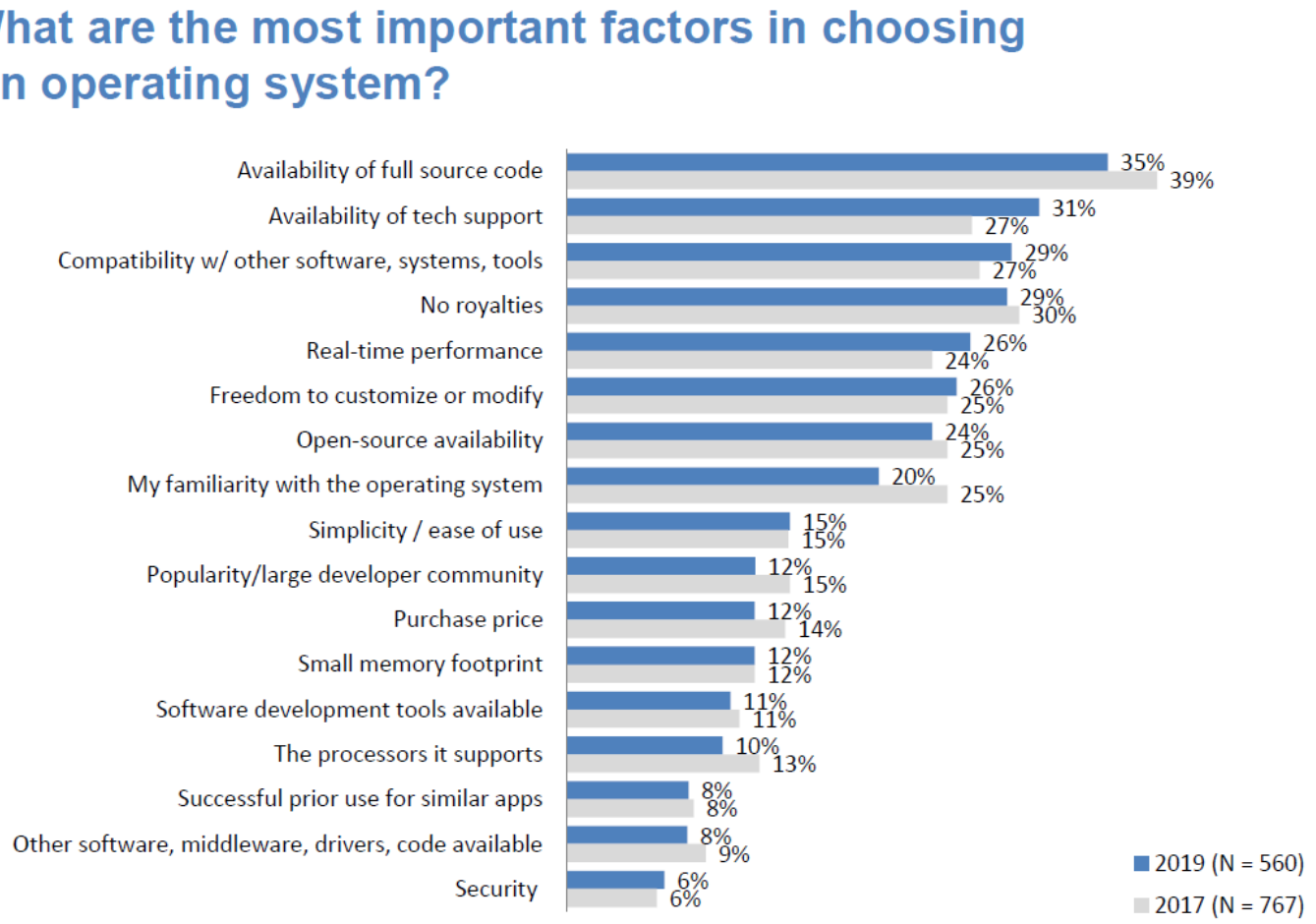
- Il en ressort aussi que connaître Linux pour l'embarqué est incontournable dans la conception d'objets connectés.

- Les langages C, Java et Python sont tout aussi importants dans la conception d'objets connectés.

CHAPITRE 6 :
LINUX EMBARQUE

PARTIE 1 :
LE BESOIN D'EMBARQUER LINUX

POINTS IMPORTANTS D'UN OS



Code source disponible, pas de *royalties*, robuste et fiable

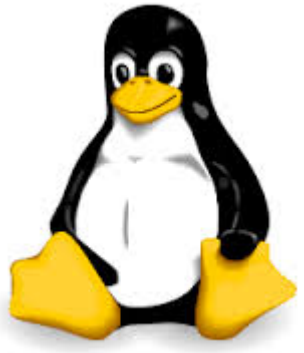
Source : EETimes Embedded Markets Study 2019 (958 réponses)

QU'EST-CE QUE LINUX ?

- Linux est un système d'exploitation libre de type UNIX créé par le finlandais Linus Torvalds en 1991 avec l'assistance de milliers de développeurs dans le monde.

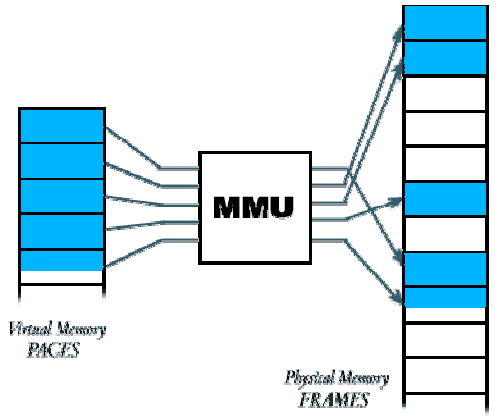
- Son succès tient au fait qu'il est développé sous licence GPL (*General Public License*), ce qui signifie que le code source est disponible à tout le monde et est gratuit.

- Son emblème est un manchot : le *tux*.



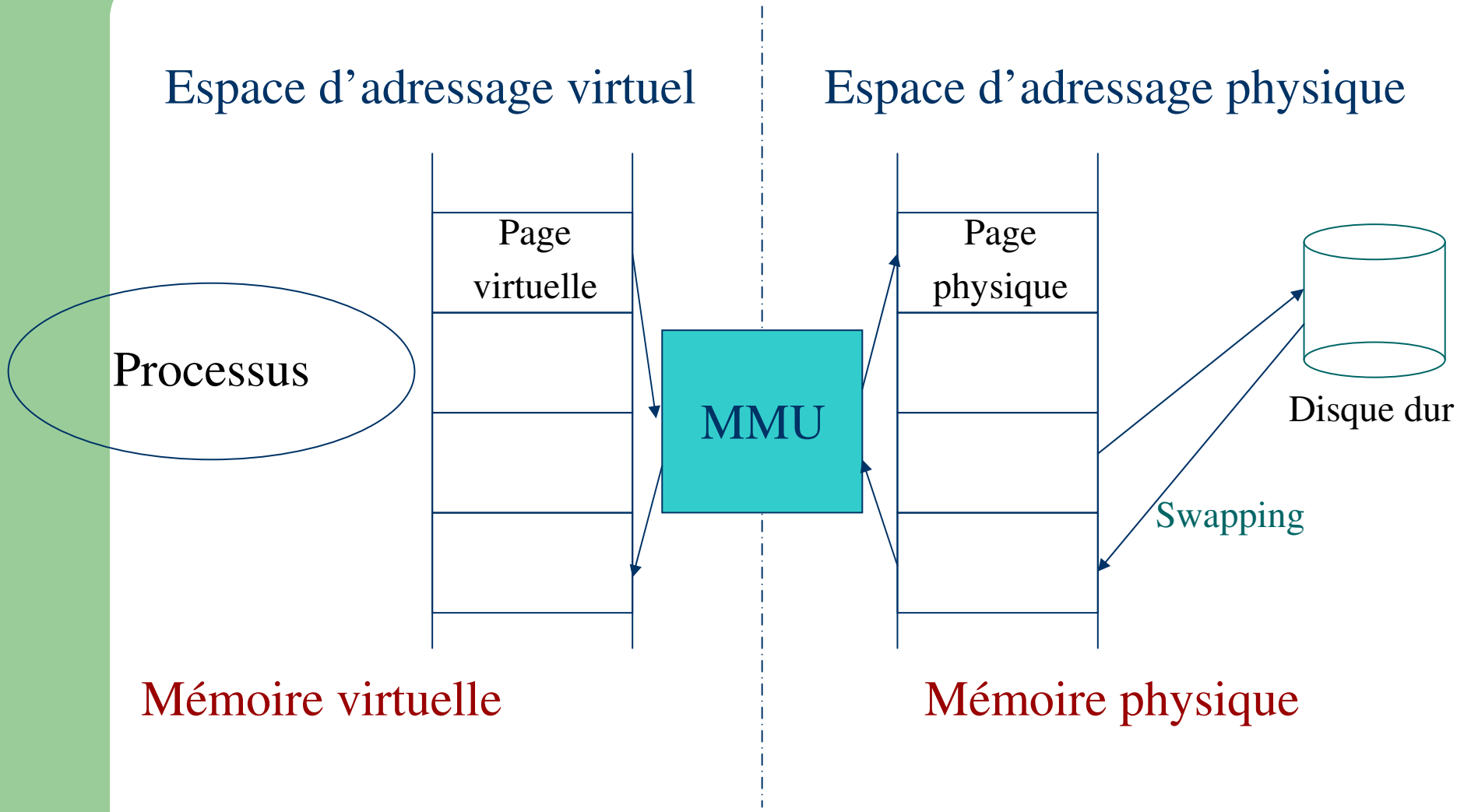
QU'EST-CE QUE LINUX ?

- Linux correspond au système d'exploitation : le noyau.
- Linux tourne originellement sur plateforme x86 32 bits (i386) et supérieure avec 8 Mo de RAM.
- **IL FAUT DONC UN PROCESSEUR 32 BITS AVEC MMU (OU A DEFAUT 32 BITS SANS MMU AVEC µClinux).**



MMU=*Memory Management Unit*

MÉMOIRE VIRTUELLE



MÉMOIRE VIRTUELLE

- La logique de décodage de la MMU prend un peu de temps.
- Dans le cas de l'embarqué, il est fréquent que la MMU soit dévalidée dans le portage du SE sur le système électronique.
- Dans ce cas, il y a confusion entre adresse logique et adresse physique.

MÉMOIRE VIRTUELLE

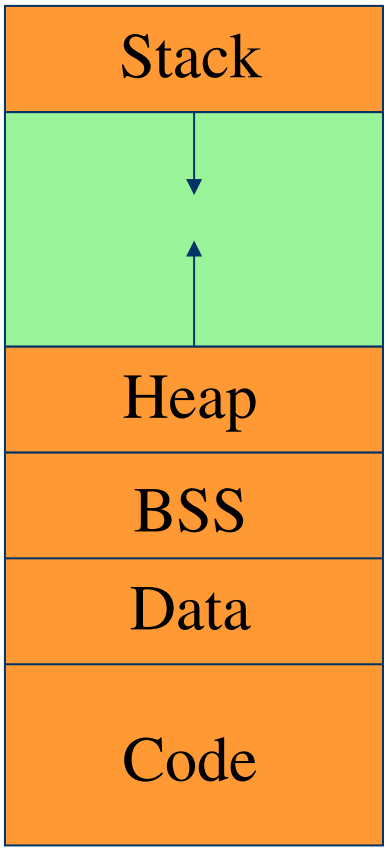
- Avoir une MMU :
  - Assure une meilleure fiabilité du système.
  - Pour accéder à un périphérique, on doit écrire le pilote que l'on incorpore dans le SE.
  - Il est néanmoins possible d'accéder à un périphérique en mode user. Dans le cas de Linux, on écrit un pilote en mode user (*user mode driver*) en utilisant */dev/mem* par exemple... Un exemple : la bibliothèque svgalib.

MÉMOIRE VIRTUELLE

- Ne pas avoir de MMU ou la dévalider :
  - Adresse logique = adresse physique.
  - Permet d'être plus rapide.
  - C'est souvent le cas dans l'embarqué.
  - Le système peut être moins fiable car on accède directement à l'espace d'adressage.
  - Pour accéder à un périphérique, il n'y a plus de pilote à écrire. Néanmoins, il vaut mieux en écrire un pour garder l'homogénéité du SE.

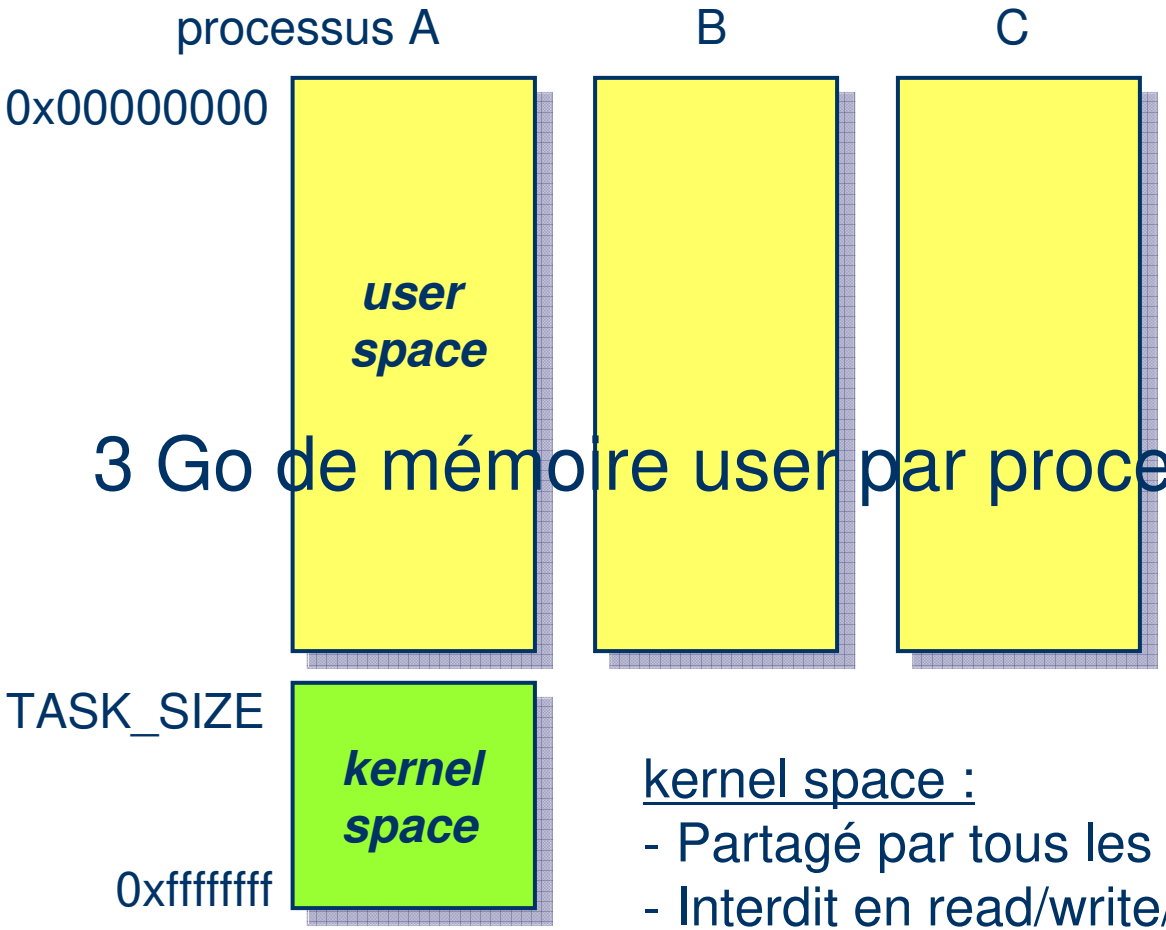
MÉMOIRE VIRTUELLE

- Pile (*stack*) :
  - Retours d'appel de fonctions et variables locales aux fonctions.
  - Variables locales et paramètres.
- Tas (*heap*) :
  - Allocation dynamique de variables.
- Code (*Text*) :
  - Code objet en lecture seulement.
- Données :
  - Data : données initialisées.
  - BSS (*Bloc Starting by Symbol*) : données non initialisées.



Processus *NIX en mémoire

MÉMOIRE VIRTUELLE



3 Go de mémoire user par processus

kernel space :
- Partagé par tous les processus
- Interdit en read/write/execute en user mode

TASK_SIZE=0xc0000000 pour x86
0xbf000000 pour ARM

MÉMOIRE VIRTUELLE

$ cat /proc/<PROCESS_ID>/maps

```
00240000-003ae000 r-xp 00000000 08:01 1018138    /lib/libc-2.9.so
003ae000-003b0000 r--p 0016e000 08:01 1018138    /lib/libc-2.9.so
003b0000-003b1000 rw-p 00170000 08:01 1018138    /lib/libc-2.9.so
08048000-080fb000 r-xp 00000000 08:01 1171471    /bin/bash
080fb000-08100000 rw-p 000b2000 08:01 1171471    /bin/bash
08100000-08105000 rw-p 08100000 00:00 0
09b25000-09b67000 rw-p 09b25000 00:00 0          [heap]
b7de8000-b7fe8000 r--p 00000000 08:01 1826818    /usr/lib/locale/locale-archive
b7fe8000-b7fea000 rw-p b7fe8000 00:00 0
b7ffe000-b8000000 rw-p b7ffe000 00:00 0
b8000000-b8007000 r--s 00000000 08:01 2261245    /usr/lib/gconv/gconv-modules.cache
bf8f2000-bf907000 rw-p bffeb000 00:00 0          [stack]
. . .
```

Address Range — file offset — inode — file name

device major:minor

r: read
w: write
x: execute
s: shared
p: private (copy on write)

QU'EST-CE QUE LINUX ?

- Linux est complété des outils GNU (gcc, binutils, gdb…). Pour être précis, on parle de GNU/Linux.
- Linux grand public est disponible sous forme de distributions : Debian, RedHat, Fedora, Ubuntu, SuSE, Slackware...
- Linux est utilisé avec une interface graphique : Gnome, KDE, Xfce, LXDE…

GNU=*Gnu is Not Unix*

QU'EST-CE QUE LINUX ?

- Linux version 0.01 en septembre 1991, c'était :
  - 88 fichiers.
  - 10 239 lignes de code.
  - Une seule architecture : i386.
- Linux version 5.8 en 2020, c'est :
  - 69325 fichiers.
  - 28 442 673 lignes de code.
  - 30 architectures.
- Linux se porte bien !

Source : 2020 Linux Kernel History Report. The Linux Fundation

POURQUOI UTILISER LINUX ?

- Linux est *open source* :
  - Le code source est disponible au public.
  - Le code source inclut :
    - Le noyau Linux.
    - Les pilotes de périphérique (*driver*).

- On peut ainsi voir directement à travers les fichiers sources ce que fait le noyau Linux voire modifier son comportement au besoin. On n'a donc pas une boîte noire (avec comme seul interlocuteur une *hot line* !).

POURQUOI UTILISER LINUX ?

- Linux est fiable :
  - Grâce à une gestion mémoire optimisée, Linux peut tourner sur une machine des années sans plantage et sans « écran bleu de la mort ».

- Linux est portable :
  - Une application Linux écrite pour une plateforme PC sous Linux peut être facilement portée sur une plateforme Linux embarquée.

POURQUOI UTILISER LINUX ?

- Linux est sécurisé :
  - Linux est conçu pour que les processus ne puissent pas lire en mémoire code et données sans provoquer une violation des règles de sécurité du système (*segmentation violation*).
  - Sécurisation du système de fichiers avec des droits d'accès.
  - Sécurisation des accès physiques à la plateforme.
  - Sécurisation des accès à distance par le réseau.

POURQUOI UTILISER LINUX ?

- Linux possède un support efficace à travers la communauté de développeurs.
- On trouve toujours une application Linux correspondant à son besoin (ou très proche).
- On capitalise son expérience UNIX en travaillant sous Linux car Linux est *UNIX like* d'où des temps de formation réduits.

POURQUOI UTILISER LINUX ?

- Les coûts de mise en œuvre de Linux sont réduits :
  - Les sources du noyau Linux sont disponibles gratuitement au téléchargement par Internet.
  - Les outils de développement (compilateurs, IDE…) sont disponibles gratuitement (outils GNU…).

IDE=*Integrated Development Environment*

POURQUOI UTILISER LINUX ?

- Linux est sans *royalties* à payer pour chaque produit vendu à base de Linux.
- Les *royalities* consistent à reverser à son éditeur une « redevance » sur chaque produit vendu qui utilise son logiciel.
- Ce point est une (r)évolution dans le domaine de l'embarqué où les outils de développement sont chers et où l'on paye en plus des *royalties* non négligeables sur chaque produit conçu avec.

LINUX ET LE LOGICIEL LIBRE

- Linux est un logiciel libre : cela donne le pouvoir aux utilisateurs d'utiliser ce logiciel comme ils l'entendent :

  "Free software is a matter of liberty, not price … 'free' as in free speech, not as in free beer…". FSF

- La licence communément utilisée est la licence GPL.

FSF=*Free Software Fundation*

L'OPEN SOURCE

- L'*open source* donne accès aux sources du logiciel.

- L'*open source* permet :
  - Une interopérabilité entre applications et les différentes plateformes.
  - L'accès aux sources permet d'optimiser des parties de code pour des performances accrues.
  - Les idées et algorithmes deviennent des standards et sont disponibles à tous sans brevet.
  - La formation par analyse des sources.

LIBRE OU OPEN SOURCE ?

« Le mouvement *open source* est une méthodologie de développement ;

le mouvement du logiciel libre, un mouvement social ».

Richard Stallman. FSF

LICENCE GPL : DROITS

- Définition technique du logiciel libre (*free software*) sous licence GPL d'après la FSF (*Free Software Foundation)* :

- *Users have the freedom to :*
  - *(1) run the software, for any purpose;*
  - *(2) study how the program works and adapt it to their needs;*
  - *(3) redistribute copies;*
  - *(4) improve the program and release improvements to the public*

- *Access to source code is necessary for (2) and (4) so "Free" can include "Open Source"*

LICENCE GPL : DROITS

- Un logiciel libre GPL peut être modifié et utilisé.
- Les améliorations d'un logiciel libre GPL peuvent être proposées par tous sous forme d'une nouvelle *release* (à éviter à cause du *fork*).
- Il est possible de gagner de l'argent avec le logiciel libre :
  - Aide, support, formation.
  - Développement logiciel.
  - Pilote de périphérique (*driver*) d'un matériel sous forme d'un module Linux (fourniture du fichier objet .ko).

LICENCE GPL : DEVOIRS

- Mise à disposition du code source.
- Les modifications apportées au programme doivent être clairement indiquées et datées (*Changelog*).
- Un logiciel libre GPL reste un logiciel libre GPL.

LICENCE GPL : DEVOIRS

- Pour distribuer un logiciel libre sous licence GPL, vous transmettez tous les droits que vous possédez.
- Il faut s'assurer que les destinataires reçoivent le code source ou peuvent se le procurer (par Internet).
- Il faut leur remettre copie de la licence GPL afin qu'eux aussi connaissent leurs droits (et leurs devoirs).

LICENCE LGPL

- La licence GPL a un pouvoir de **contamination** : toute inclusion d'une seule ligne de code d'un projet GPL dans un projet propriétaire rend le projet GPL.

- Il n'est donc pas possible de développer un projet propriétaire utilisant du code GPL sans mettre à disposition le code du projet en entier !

- On a alors créé la licence LGPL (*Less* GPL) qui est une licence faite pour le monde industriel qui utilise du logiciel *open source*.

- La licence LGPL n'a pas de pouvoir de contamination.

LICENCE LGPL

- Un logiciel propriétaire qui inclut du logiciel LGPL n'a pas à mettre à disposition son code source. Cela permet donc de développer un projet propriétaire avec du logiciel LGPL.
- Le noyau est GPL. La bibliothèque *libc* doit être et donc est LGPL.
- Un *driver* inclus dans le code du noyau est toujours GPL (*driver* statique).
- Un *driver* propriétaire doit donc être développé hors du noyau sous forme d'un module Linux (*driver* dynamique).

AUTRES LICENCES

Licence/Action	Mix avec du code non libre	Modifications secrètes	Tout le monde peut changer la licence	Compatible GPL (V2)	Copyleft
GPL V2	Non	Non (Distribution)	Non	Oui	Oui
LGPL	Oui	Non (Distribution)	Non	Oui	Oui
BSD	Oui	Oui	Non	Non	Non
MPL	Oui	Oui	Non	Non	Oui, mais relative
CeCILL V2	Non	Non	Non	Oui	Oui
Apache V2	-	Oui	-	Officieuse, prévoir la double licence	Non
Domaine Public	Oui	Oui	Oui	Oui	Non

D'après korben.info

LINUX ET LA PORTABILITE

- Linux est fortement portable. Une même application peut être utilisée (portée) sur :
  - Un nombre important de processeurs *hardcores* : RISC-V, MIPS, Blackfin, ARM, PowerPC, Zynq…
  - Un nombre important de processeurs *softcores* : RISC-V, NIOS II (Altera/Intel), MicroBlaze (Xilinx/AMD), Leon...
  - Un nombre important de plateformes cibles ou BSP (*Board Support Package*).

- Linux est un système d'exploitation de choix pour les systèmes embarqués. On parle alors de **Linux embarqué**.

LINUX EMBARQUE

- Linux embarqué est donc une version du noyau Linux standard (*vanilla*) adaptée à un système embarqué.

- Suivant les capacités du système embarqué (processeur), on retrouve toutes les fonctionnalités du noyau Linux standard ou bien des fonctionnalités modifiées (par exemple quand pas de MMU).

OUTILS POUR LINUX EMBARQUE

- On utilise pour le développement sous Linux embarqué les mêmes outils traditionnels GNU sur PC :
  - Compilateurs croisés C/C++.
  - IDE.
  - GDB (GNU *DeBugger*).

IDE=*Integrated Development Environment*

OUTILS POUR LINUX EMBARQUE

- Le langage C est préférable pour limiter la taille des exécutables mais aussi la consommation énergétique :

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

OUTILS POUR LINUX EMBARQUE

- On utilise pour le développement sous Linux embarqué un PC de développement sous Linux (hôte) avec une chaîne de compilation croisée en fonction du processeur embarqué sur le système (cible).

- L'exécutable ainsi produit est téléchargé dans la cible pour pouvoir y être testé. On utilisera alors GDB pour *debugger* l'application par le réseau que l'on pourra coupler avec une interface graphique.

- C'est le principe du développement croisé.

OUTILS POUR LINUX EMBARQUE

- On a un environnement de développement croisé :

ICE=*In Circuit Emulator* JTAG=*Joint Test Action Group*

OUTILS POUR LINUX EMBARQUE

- Il existe des simulateurs tournant sur le PC hôte pour simuler la cible :
  - *qemu*.

- Cela permet de créer une machine virtuelle tournant un Linux embarqué correspondant à la carte cible (BSP) et à son type de processeur.

PROCESSEUR POUR LINUX EMBARQUE

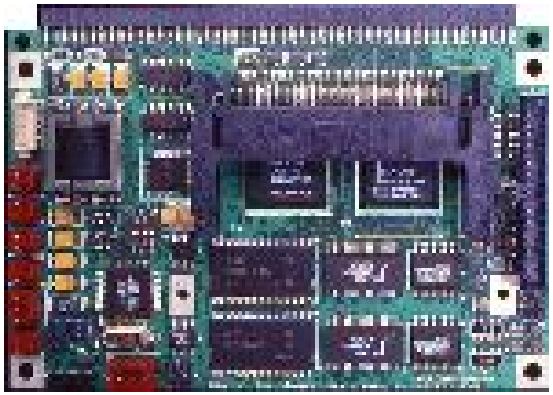
Besoin	**Petit**	**Moyen**	**Gros**	**Embarqué hautes performances**
Taille RAM	Qq diz. Mo	Qq cent. Mo	> Go	> Go
Taille ROM/FLASH	Qq Mo	Qq diz. Mo et SD	> Go et SD	Go-To
Processeurs	ARM, RISC-V, ColdFire, Blackfin, Zynq, NIOS II, MicroBlaze, Leon		ARM, RISC-V, x86, PowerPC, Zynq	x86

Choix suivant puissance de calcul, mémoire...

PARTIE 2 :
CARTES POUR LINUX EMBARQUE

CARTES POUR LINUX EMBARQUE

- Cartes COTS (*Commercial Off-The-Shelf*) :



- Carte au format *Little Board* (5.75 x 8.0 inch)
- Carte au format PC/104 avec bus ISA (3.6 x 3.8 inch)

- Carte PC/104-Plus avec bus PCI

pc104.org

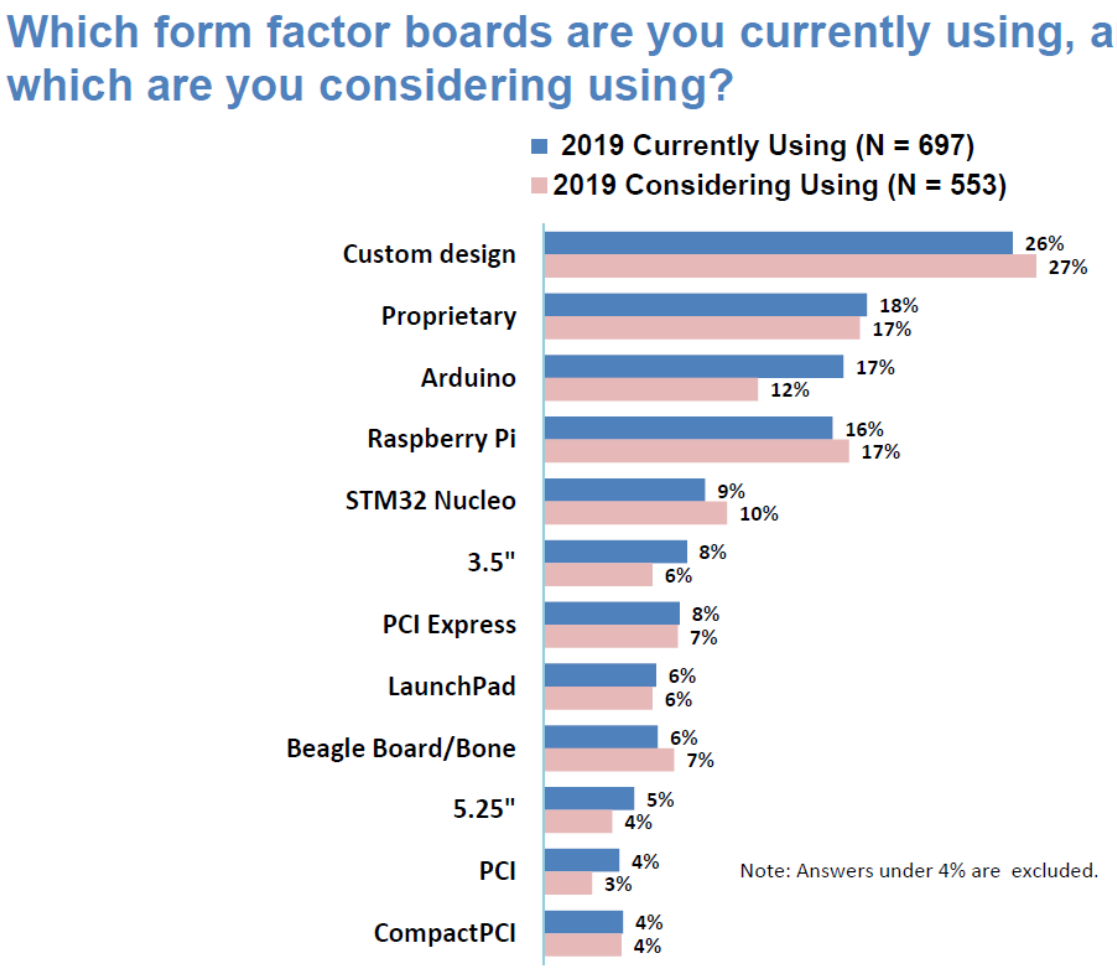
CARTES POUR LINUX EMBARQUE

- Cartes au format industriel VME, VXI, PXI… :

Carte au format VME (*Versa Module Eurocard*)

Carte au format PXI (*PCI eXtended to Instrumentation*)

FORMAT DES CARTES POUR LINUX EMBARQUE



Maison, COTS, matériels libres

Source : EETimes Embedded Markets Study 2019 (958 réponses)

CARTES POUR GEEK DE L'EMBARQUE

- On peut toujours réutiliser une vieille carte mère de PC avec un adaptateur IDE/Compact Flash sur un port IDE de la carte mère ou bien une clé USB. Cela permet d'avoir une mémoire Flash sur laquelle on peut démarrer.

- Mais la présence du BIOS est gênante car il est préférable d'avoir un véritable *bootloader* Linux !

BIOS=*Basic I/O System*

CARTES POUR GEEK DE L'EMBARQUE

- Pour s'adonner à Linux embarqué, il est possible de trouver des cartes proches du monde industriel pour 150 € voire nettement moins avec l'arrivée de la carte Raspberry Pi à 40-80 €.

- Ces cartes doivent avoir de la mémoire Flash pour y intégrer le *bootloader*, le noyau Linux et son système de fichiers *root*.

- Un PC de développement sous Linux permettra le développement croisé.

CARTES POUR GEEK DE L'EMBARQUE

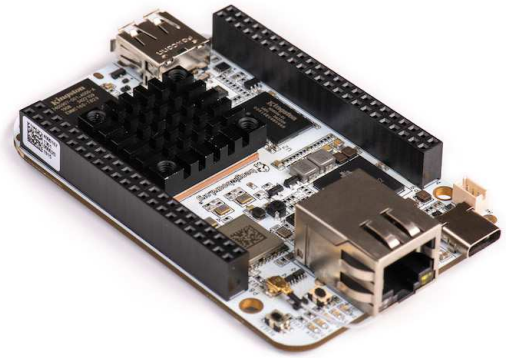
- Carte Raspberry Pi :
  - www.raspberrypi.org
  - Carte comprenant :
    - SoC Broadcom avec processeur Cortex-A53 (3B/3B+) ou Cortex-A72 (4B) ou Cortex-A76 (5B) et processeur graphique propriétaire.
    - Jusqu'à 8 Go de RAM, carte SD ou microSD.
    - Modèles Zero, A, A+, B, B+, 2B, 3B, 3B+,4B, 5B depuis 2012.
    - 40 €-100 €.
    - 2012-2024 : 60 millions de cartes vendues !

CARTES POUR GEEK DE L'EMBARQUE

- Carte BeagleBone Black :
  - beagleboard.org
  - Carte comprenant :
    - Processeur ARM Cortex-A8 à 1 GHz.
    - 512 Mo de RAM, 4 Go de Flash, carte microSD.
    - 43 €.

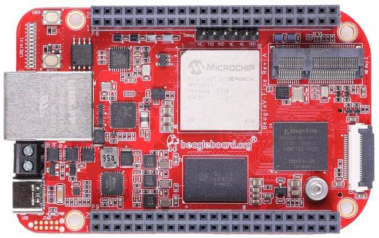
CARTES POUR GEEK DE L'EMBARQUE

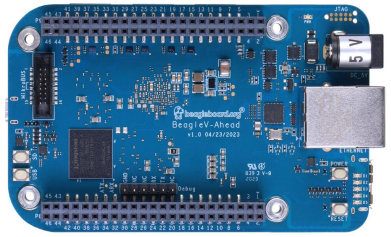
- Carte BeagleBone AI-64 :
  - beagleboard.org
  - Carte comprenant :
    - Processeur ARM Cortex-A72.
    - 4 Go de RAM, 16 Go de Flash, carte microSD.
    - 125 €.



CARTES POUR GEEK DE L'EMBARQUE

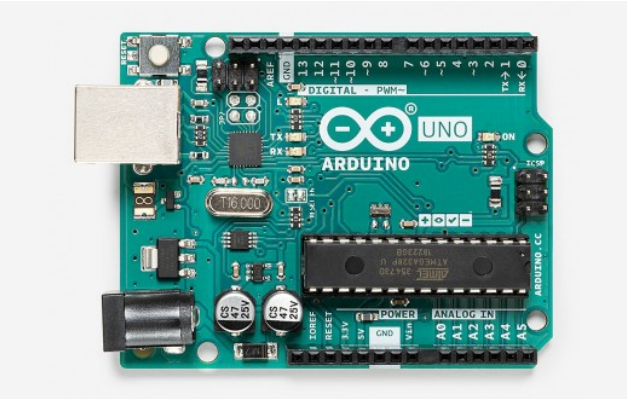
- Carte BeagleV-Fire BeagleV-Ahead :
  - beagleboard.org
  - Cartes à base de processeurs RISC-V :
    - Fire : processeur Microchip PolarFire® MPFS025T@667 MHz avec 4 cœurs 64 bits RV64GC et 1 coeur 64bits RV64IMAC, 2 Go de RAM et 4 Go de Flash. 165 €.
    - Ahead : processeur T-Head TH1520@2 GHz avec 4 cœurs 64 bits RV64GC, 4 Go de RAM et 16 Go de Flash. 156 €.

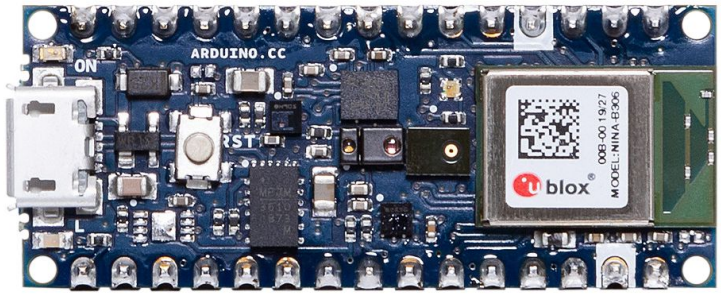




CARTES POUR GEEK DE L'EMBARQUE

- Cartes Arduino :
  - www.arduino.cc
  - Différentes cartes pour du prototypage rapide en langage C *bare-metal*. Modèle Arduino nano 33 BLE Sense adaptée à l'IA embarquée (TinyML).
    - IDE spécifique : Arduino IDE.
    - 20-60 €.





CARTES POUR GEEK DE L'EMBARQUE

- Cartes Armadeus :
  - www.armadeus.com
  - Carte APF51 avec carte APF51_DEVcomprenant :
    - Processeur ARM Cortex-A8 à 800 MHz.
    - FPGA Spartan 6A. Idéal pour le *codesign* !
    - 256 Mo de RAM, 64 Mo à 1 Go de Flash, carte SD.
    - 144 €.

CARTES POUR GEEK DE L'EMBARQUE

- Cartes Armadeus :
  - www.armadeus.com
  - Carte APF6_SP avec carte APF6_SP_DEVcomprenant :
    - Processeur ARM Cortex-A9 à 1 GHz.
    - FPGA Cyclone V. Idéal pour le *codesign* !
    - 1 Go de RAM, 4 Go de Flash, carte SD.

CARTES POUR GEEK DE L'EMBARQUE

- On peut aussi recycler du matériel grand public :
  - Routeur Fonera avec OpenWrt (openwrt.org).
  - Routeur LinkSys avec OpenWrt (openwrt.org).
  - Nintendo DS avec DSLinux (www.dslinux.org).
  - …

POINTS FAIBLES DE LINUX EMBARQUE

- Les *drivers* Linux pour un périphérique donné ne sont pas toujours disponibles.
- Il n'existe pas un Linux embarqué mais des Linux embarqués.
- Le modèle de la licence GPL mal compris (droits et surtout devoirs).
- Linux n'est pas certifié :
  - Norme FDA 510k pour le médical.
  - Norme DO-178B pour l'aéronautique.

PARTIE 3 :
LES OFFRES LINUX EMBARQUE

LES OFFRES LINUX EMBARQUE

- Les offres Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
  - Les distributions **Linux classiques** :
    - RedHat, Ubuntu, Fedora, Debian…
    - Pour processeurs x86 et ARM.

- Suivant la quantité de mémoire disque du système embarqué, il est possible d'édulcorer une distribution classique (quelques centaines de Mo). Cela tient alors dans une mémoire Flash.

LES OFFRES LINUX EMBARQUE

- Les offres Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
  - Les distributions **Linux embarqué commerciales** :
    - non TR : Montavista, Intel, LynuxWorks (BlueCat)…
    - TR : Montavista, Intel (Wind River/RTLinux), LynuxWorks (BlueCat RT)…

LES OFFRES LINUX EMBARQUE

- Les offres Linux embarqué (et Temps Réel) peuvent être rangées dans l'une des 3 catégories suivantes :
  - Les distributions **Linux embarqué libres** :
    - non TR : à la main, µClinux. *Build systems* Buildroot, Yocto.
    - TR dur : Xenomai, RTAI (et aussi Buildroot, Yocto).
    - TR mou : PREEMPT-RT (et aussi Buildroot, Yocto).
    - Pour information : autres non Linux : FreeRTOS, eCOS...

LINUX EMBARQUE COMMERCIAL

- MontaVista Professional Edition :
  - Outil de génération d'un Linux embarqué (OS+RFS).
  - www.mvista.com

OS=*Operating System* RFS=*Root File System*

LINUX EMBARQUE COMMERCIAL

- Caractéristiques de MontaVista Professional Edition :
  - Board Hardware Support :
    - *Support for over seventy popular COTS, Evaluation, and Reference boards.*
    - *Support for seven target CPU families with more than 25 CPU variants.*
  - Development Environment :
    - *KDevelop IDE.*
    - *MontaVista Target Configuration Tool.*
    - *MontaVista Library Optimizer Tool.*
    - *Graphical binary and source-level debug.*
    - *Graphical kernel configuration tool.*
    - *Kernel debug (KGDB and hardware debuggers).*
    - *File system populator.*

LINUX EMBARQUE OPEN SOURCE

- µClinux est fait pour :
  - Les processeurs 32 bits sans MMU.
  - Outil de génération d'un Linux embarqué (OS+RFS).
  - www.uclinux.org

- Caractéristiques de µClinux :
  - *µClinux is the ideal OS for non-MMU microprocessors and high-volume embedded systems featuring posix-4, real-time functions, and TCP/IP. µClinux includes a complete TCP/IP stack supporting Ethernet, PPP and SLIP as well as many wireless protocols. µClinux is perfect for remote sensing, monitoring and control applications. And, because µClinux is an open source product, you will never be stuck on a dead end development path.*

BUILD SYSTEMS

- Buildroot est un *build system* ou un système de construction de distribution :
  - Outil de génération d'un Linux embarqué (OS+RFS+*bootloader*+compilateur croisé).
  - buildroot.uclibc.org

- Buildroot génère une distribution Linux statique proche de ce que donne µClinux que l'on peut aussi considérer aussi comme un *build system*. Il ne produit pas de paquetages que l'on peut installer à chaud dans le système Linux embarqué.

- On se rapproche d'un *firmware*.

BUILD SYSTEMS

- Caractéristiques de Buildroot :
  - *Buildroot is a tool that simplifies and automates the process of building a complete Linux system for an embedded system, using cross-compilation. In order to achieve this, Buildroot is able to generate a cross-compilation toolchain, a root filesystem, a Linux kernel image and a bootloader for your target. Buildroot can be used for any combination of these options, independently. Buildroot is useful mainly for people working with embedded systems. Embedded systems often use processors that are not the regular x86 processors everyone is used to having in his PC. They can be PowerPC processors, MIPS processors, ARM processors, etc. Buildroot supports numerous processors and their variants; it also comes with default configurations for several boards available off-the-shelf.*

- Buildroot est une solution industrielle.

BUILD SYSTEMS

- Yocto est un *build system* sponsorisé par Intel qui a intégré le projet OpenEmbedded :
  - Outil de génération d'un Linux embarqué (OS+RFS+*bootloader*+compilateur croisé).
  - www.yoctoproject.org

- Yocto génère une distribution Linux dynamique et produit des paquetages que l'on peut installer à chaud dans le système Linux embarqué comme dans le cas d'une distribution Linux classique (avec *apt*, *rpm*).

BUILD SYSTEMS

- Caractéristiques de Yocto :
  - *The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. The Yocto Project provides resources and information catering to both new and experienced users, and includes core system component recipes provided by the OpenEmbedded project. The Yocto Project also provides pointers to example code built demonstrating its capabilities. These community-tested images include the Yocto Project kernel and cover several build profiles across multiple architectures including ARM, PPC, MIPS, x86, and x86-64. Specific platform support takes the form of Board Support Package (BSP) layers for which a standard format has been developed. The project also provides an Eclipse IDE plug-in and a graphical user interface to the build system.*
- Yocto est une solution industrielle.
- Yocto est plus compliqué à mettre en œuvre que Buildroot.

BUILD SYSTEMS

- Petalinux est un *build system* développé par Xilinx/AMD basé sur Yocto :
  - Outil de génération d'un Linux embarqué (OS+RFS+*bootloader*+compilateur croisé).
  - www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html
- Petalinux est utilisable avec les circuits FPGA Xilinx Versal, Zynq UltraScale+, Zynq et avec le processeur *softcore* MicroBlaze.
- Le fichier BOOT.BIN produit contient le FSBL, *u-boot* et le fichier de programmation du circuit FPGA system.bit.

FSBL=*First Stage Boot Loader*

LE CHOIX D'UN LINUX EMBARQUE

- Le choix est à faire en fonction de ses compétences en interne et des TTM à respecter.
- Choisir un Linux embarqué commercial est rassurant. Cela a aussi un coût.
- Le choix pour concevoir sa distribution Linux embarqué dépendra de la complexité de son projet :
  - Application métier avec quelques paquetages : construction manuelle.
  - Application**s** métier avec des dizaines de paquetages : construction automatique avec un *build system*.

TTM=*Time To Market*

PARTIE 4 :
BOITE A OUTILS LIBRES
POUR LINUX EMBARQUE

BOITE A OUTILS LINUX EMBARQUE

- Il existe un ensemble de logiciels libres utilitaires (projets) qui facilitent la conception logicielle d'un système sous Linux embarqué.

- On utilise dans ce cadre des utilitaires à faible empreinte mémoire adaptés à l'embarqué. Ils possèdent leur équivalent sous Linux standard.

- Il convient de connaître la boîte à outils libres pour concevoir son système Linux embarqué.

COMPILATEURS CROISES

Outil	Adresse Internet	Description
gcc	**www.gnu.org/software/gcc/**	**Compilateur C, C++, Objective-C, Fortran...**
buildroot	**buildroot.uclibc.org/**	***Build system***
Yocto	**www.yoctoproject.org/**	***Build system***
crosstool **crosstoolng**	**www.kegel.com/crosstool/** **crosstool-ng.org/**	**Générateur automatique d'un compilateur C, C++ et la bibliothèque *glibc* pour processeurs alpha, ARM, x86, IA64, MIPS, PowerPC, SH4...**

COMPILATEURS CROISES

Outil	Adresse Internet	Description
ELDK	**ftp://mirror.switch.ch/mirror/eldk/eldk/**	***Embedded Linux Development Kit* de la société DENX. Contient les compilateurs croisés compilés pour processeurs PowerPC, ARM, MIPS**
Codesourcery	**www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/**	**Compilateur croisé compilé pour processeur ARM**
bootlin	**toolchains.bootlin.com/**	**Compilateurs croisés compilés pour les principaux processeurs, glibc et uClibc**
	www.uclinux.org/pub/uClinux/m68k-elf-tools/	**Compilateur croisé compilé pour processeurs m68k, ColdFire et ARM**
	www.linux-mips.org/wiki/MIPS_SDE_Installation	**Compilateur croisé compilé pour processeur MIPS**

COMPILATION NATIVE D'UNE APPLICATION

- Il suffit de créer le fichier `Makefile` suivant pour le fichier source `hello.c` :

```
#CC=gcc
CFLAGS = -c
OBJS =    hello.o

main : $(OBJS)
    $(CC) $(LDFLAGS) $(OBJS) -o hello
%.o: %.c
    $(CC) $(CFLAGS) $<
clean:
    rm -f *.o *~ hello
```

- Il s'agit d'une compilation native. Par défaut, CC vaut gcc. Si on précise CC, on l'initialisera à :

```
CC=gcc
```

COMPILATION CROISEE D'UNE APPLICATION

- Il suffit de créer le fichier `Makefile` suivant pour le fichier source `hello.c` :

```
CC=arm-buildroot-linux-gnueabihf-gcc
CFLAGS = -c
OBJS =    hello.o

main : $(OBJS)
    $(CC) $(LDFLAGS) $(OBJS) -o hello
%.o: %.c
    $(CC) $(CFLAGS) $<
clean:
    rm -f *.o *~ hello
```

- Il faut utiliser un compilateur croisé, par exemple ici pour un processeur ARM. La variable CC est alors initialisée à :

```
CC=arm-buildroot-linux-gnueabihf-gcc
```

COMPILATION NATIVE DU NOYAU LINUX

- On considère Linux pour un PC avec un processeur x86 64 bits. On retrouve la suite d'actions :

- Configuration par défaut du noyau Linux pour un processeur x86 64 bits :

```
% make x86_64_defconfig
```

- Configuration du noyau :

```
% make menuconfig
```

- Compilation native du noyau :

```
% make
```

COMPILATION CROISEE DU NOYAU LINUX

- On considère Linux pour une carte ZedBoard avec circuit FPGA Xilinx contenant un processeur SoC ARM. On retrouve la suite d'actions :

- Configuration par défaut du noyau Linux pour un processeur ARM 32 bits :

```
% make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf-
xilinx_zynq_defconfig
```

- Configuration du noyau :

```
% make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf- menuconfig
```

- Compilation croisée du noyau :

```
% make ARCH=arm CROSS_COMPILE=arm-buildroot-linux-gnueabihf-
UIMAGE_LOADADDR=0x8000 uImage -j8
```

COMPILATION CROISEE DU NOYAU LINUX

- On considère Linux pour une carte ZuBoard avec circuit FPGA Xilinx contenant un processeur MPSoC ARM. On retrouve la suite d'actions :

- Configuration par défaut du noyau Linux pour un processeur ARM 64 bits :

```
% make ARCH=arm64 CROSS_COMPILE=aarch64-buildroot-linux-gnu-
xilinx_zynqqmp_defconfig
```

- Configuration du noyau :

```
% make ARCH=arm64 CROSS_COMPILE=aarch64-buildroot-linux-gnu-  menuconfig
```

- Compilation croisée du noyau :

```
% make ARCH=arm64 CROSS_COMPILE=aarch64-buildroot-linux-gnu- -j8
```

COMPILATION CROISEE DU NOYAU LINUX

- Pour une compilation croisée du noyau Linux, il faudra donc préciser les variables suivantes :
  - ARCH : type du processeur.
  - CROSS_COMPILE : préfixe du compilateur croisé.

- L'installation du noyau Linux se fera manuellement suivant l'environnement de démarrage de la carte cible.

BIBLIOTHEQUES LIBC

Outil	Adresse Internet	Description
glibc	**www.gnu.org/software/libc/**	***libc* standard GNU. Peu adaptée à l'embarqué**
µClibc	**www.uclibc.org/**	***libc* pour l'embarqué**
newlib	**sources.redhat.com/newlib/**	***libc* pour l'embarqué**
dietlibc	**www.fefe.de/dietlibc/**	***libc* pour l'embarqué**

DEBUG

Outil	Adresse Internet	Description
gdb	**www.gnu.org/software/gdb/gdb.html**	***Debugger* GNU. Une version *cross compilée* est généralement fournie avec les chaînes de compilation croisée précompilées**
DDD	**www.gnu.org/software/ddd/**	***Data Display Debugger*. *Front end* à *gdb***

- NB : kgdb est la version gdb pour le *debug* du noyau Linux.

NOYAU LINUX

Outil	Adresse Internet	Description
Noyau Linux vanilla	**www.kernel.org/**	**Noyau Linux standard**
Noyau Linux	**www.denx.de/**	**Noyau Linux pour processeurs ARM, PowerPC et MIPS (ELDK)**
Noyau Linux	**www.linux-mips.org/**	**Noyau Linux pour processeur MIPS**
Noyau Linux	**rocketboards.org/foswiki/Documentation/NiosIILinuxUserManual**	**Noyau Linux pour processeur *softcore* NIOS II/Intel avec MMU**
Noyau Linux	**github.com/Xilinx/linux-xlnx**	**Noyau Linux pour processeur *softcore* Microblaze/AMD avec MMU**

NOYAU µCLINUX

Outil	Adresse Internet	Description
Noyau µClinux	**www.uclinux.org/**	**Noyau µClinux pour processeur sans MMU**
Noyau µClinux	**www.uclinux.org/ports/coldfire/**	**Noyau µClinux pour processeur ColdFire**
Noyau µClinux	**www.uclinux.org/pub/uClinux/ports/arm7tdmi/**	**Noyau µClinux pour processeur ARM7**
Noyau µClinux	**blackfin.uclinux.org/**	**Noyau µClinux pour processeur Blackfin**

BOOTLOADERS LINUX

Outil	Adresse Internet	Description
u-boot	**sourceforge.net/projects/u-boot**	***Universal Boot. Bootloader* Linux pour processeurs PowerPC, ARM, x86, MIPS, Xscale, ColdFire 52x2,PXA... et processeurs *softcore* MicroBlaze et NIOS II**
colilo	**www.reasonability.net/uclinux/colilo/**	***Bootloader* Linux pour processeur ColdFire**
RedBoot	**ecos.sourceware.org/redboot/**	***Bootloader* Linux pour processeurs ARM, CalmRISC, FR-V, H8, IA32, 68k, Matsushita AM3x, MIPS, NEC V8xx, PowerPC, SPARC, SuperH. S'utilise avec eCos**
FreeBIOS	**freebios.sourceforge.net/**	**BIOS pour carte mère à base de processeur x86**
OpenBIOS	**www.openbios.info/**	**BIOS pour carte mère à base de processeurs x86, Alpha et AMD64**

SHELLS ET COMMANDES LINUX

Outil	Adresse Internet	Description
busybox	**busybox.net/**	**Véritable couteau suisse ! *busybox* est incontournable aujourd'hui**

- NB : busybox fournit en un seul exécutable toutes les commandes UNIX usuelles et l'arborescence standard avec ses fichiers de configuration d'un système de fichiers *root*.
- Un système de fichiers *root* doit respecter la norme UNIX FHS.

FHS=*Filesystem Hierarchy Standard*

EDITEURS DE TEXTES

Outil	Adresse Internet	Description
vi	**busybox.net/**	**vi est dans *busybox***
ae	**www.nyangau.fsnet.co.uk/**	***Andy's Editor*. Editeur**
e3	**www.sax.de/~adlibit/**	**Editeur compatible *vi*, *emacs* et *Wordstar***
nano	**www.nano-editor.org/**	**Editeur**

OUTILS RESEAU ET CONNECTIVITE IP

Outil	Adresse Internet	Description
boa	**www.boa.org/**	**Serveur web. Le serveur web pour l'embarqué. Support des scripts CGI boa+php pour µClinux disponible ici : www.menie.org/georges/uClinux/boa-php.html**
thttpd	**www.acme.com/software/thttpd/**	***tiny/turbo/throttling HTTP server*. Serveur web**
busybox	**busybox.net/**	***busybox* contient un serveur web**
lightTPD	**lighttpd.net/**	**Serveur web. Support des scripts CGI**

OUTILS RESEAU ET CONNECTIVITE IP

Outil	Adresse Internet	Description
Iproute	**packages.debian.org/unstable/net/iproute.html**	**Outil pour la configuration réseau**
ntpclient	**doolittle.faludi.com/ntpclient/**	**Client NTP**
dropbear	**www.ucc.gu.uwa.edu.au/~Ematt/dropbear/dropbear.html**	**Client et serveur SSH. Peut être compilé avec µClibc**
tinylogin	**tinylogin.busybox.net/**	**Outil pour gérer le contrôle d'accès au système embarqué : login, passwd, getty...**
udhcp	**udhcp.busybox.net/**	**Client et serveur DHCP (intégré dans *busybox*)**

INTERFACES GRAPHIQUES

Outil	Adresse Internet	Description
Frame Buffer	**www.linux.org**	**Intégré au noyau Linux**
Nano-X	**www.microwindows.org/**	**Environnement graphique tournant au-dessus du *Frame Buffer*, X Window, SVGAlib. Il existe 2 API (*Application Programming Interface*) :** **- Nano-X : API compatible Xlib** **- Win32 : API compatible Windows Win32**
Qt embedded	**qt-project.org/**	**Environnement graphique Qt tournant au-dessus du *Frame Buffer***

- NB : Qt embedded possède un système à double licence : licence GPL ou licence propriétaire payante pour un produit industriel.

AUTRES

Outil	Adresse Internet	Description
MTD	**www.linux-mtd.infradead.org/**	***Memory Technology Device*. Bibliothèque pour gérer la mémoire Flash. Intégré au noyau Linux**
JFFS2	**sources.redhat.com/jffs2/**	***Journalling Flash File System, version 2.* Système de fichiers pour mémoire Flash**
qemu	**qemu.org/**	**Emulateur pour processeurs x86, ARM, SPARC et PowerPC. Peut émuler un système Linux pour processeurs x86, x86_64 et PowerPC**
JamVM	**jamvm.sourceforge.net/**	**Machine virtuelle Java**

PARTIE 5 :
BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Pour bien concevoir son système embarqué libre, il convient de respecter quelques points importants.

- Il s'agit de règles simples de bon sens !

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Le système d'exploitation est Linux. Il faut donc choisir un processeur 32 bits supporté par Linux ou µClinux (avec ou sans MMU).

- Il faut choisir des composants électroniques supportés par Linux et non l'inverse. On n'aura à développer que les *drivers* spécifiques de son application.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Très tôt dans le processus de conception, on utilisera une carte d'évaluation du commerce utilisant le même processeur que celui de son *design*.

- Il est même possible d'utiliser cette carte d'évaluation comme carte finale de son *design* en rajoutant éventuellement des cartes filles pour sa partie spécifique.

- Cela est d'autant plus facile que l'on utilise des cartes au format industriel.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Il serait étonnant de ne pas avoir le portage de Linux embarqué pour la carte d'évaluation choisie !

- Il est alors possible de développer la partie logicielle sur la carte d'évaluation en ajoutant les cartes filles spécifiques. Cela est généralement simplifié par la présence de connecteurs d'E/S permettant de s'interfacer sur le bus du processeur.

- Il est fourni généralement les schémas électroniques de la carte d'évaluation, ce qui permet une reprise de CAO pour la carte finale.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

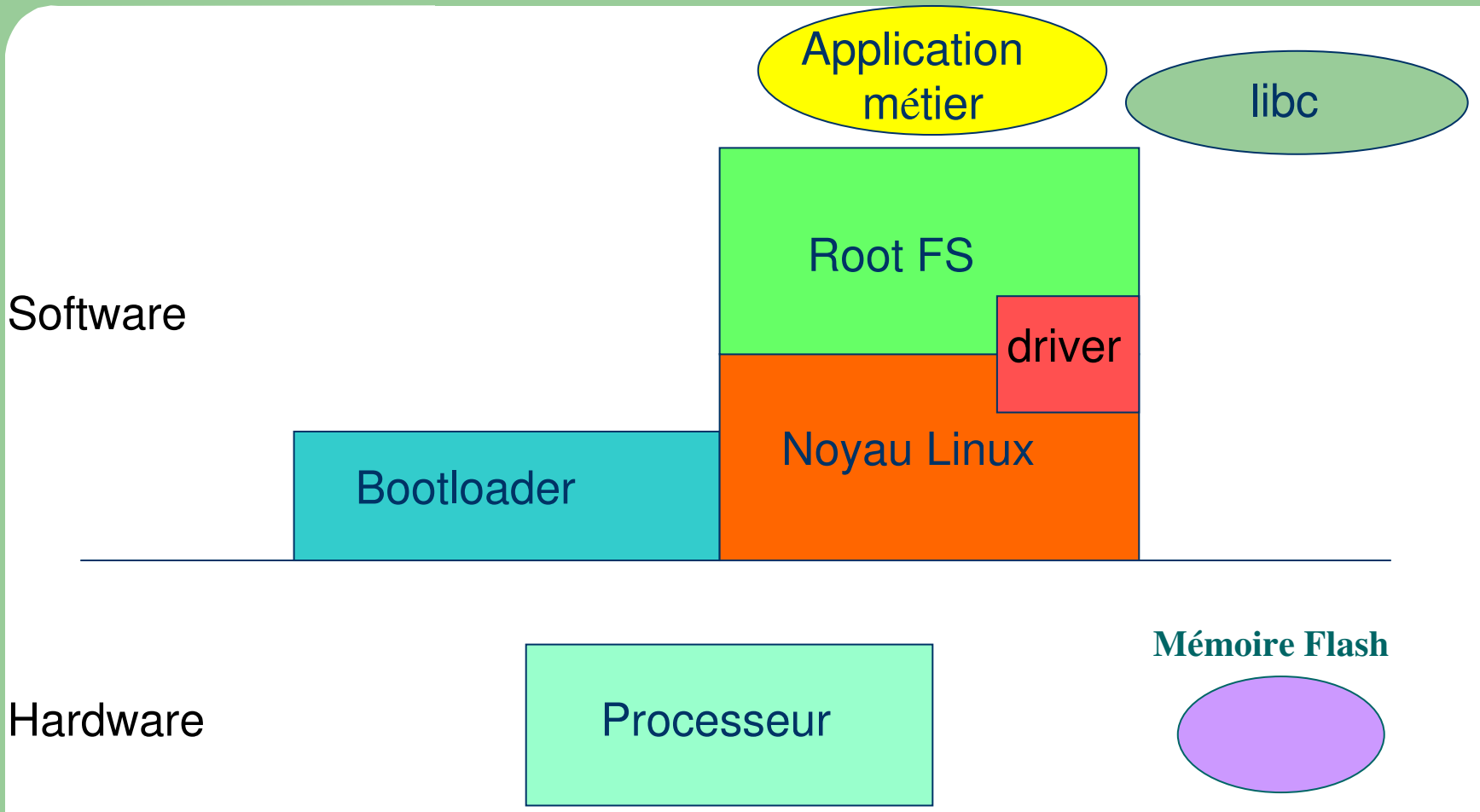
- Il ne faut pas hésiter à s'abonner et poser des questions dans les forums adéquats. La réponse est rapide.
- Il ne faut pas hésiter à suivre des formations pour gagner du temps. C'est un investissement pour l'avenir !
- L'expérience acquise dans le libre pour l'embarqué peut déboucher vers d'autres marchés…

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- Il faut privilégier la solution Linux embarqué la plus simple. On **construira la distribution à la main**. On peut aussi s'orienter vers les *build systems* buildroot ou Yocto.

- Au minimum, pour concevoir son système Linux embarqué **à la main**, on a besoin **de 5 briques** :
  - Compilateur croisé gcc : choisir une chaîne déjà compilée.
  - *Bootloader* Linux : u-boot.
  - Noyau Linux ou µClinux.
  - Bibliothèque *libc* : glibc, µClibc : prendre les fichiers de la chaîne de compilation croisée.
  - Système de fichiers *root* (RFS) : busybox.

BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE



BIEN CONCEVOIR SON SYSTÈME EMBARQUE LIBRE

- u-boot : on adaptera le *bootloader* à sa carte cible.
- Noyau Linux : le noyau Linux sera configuré et éventuellement adapté à sa carte cible puis cross-compilé.
- Bibliothèque libc : on utilisera une bibliothèque adaptée comme µClibc ou l'on prendra les fichiers de la libc du compilateur croisé.
- Busybox : le « couteau suisse » sera configuré puis cross-compilé.

PARTIE 6 :
MISE EN ŒUVRE DE LINUX EMBARQUE
µClinux SUR BLACKFIN

PRESENTATION DE µClinux

- µClinux est originellement un dérivé du noyau Linux version 2.0 pour les microcontrôleurs sans MMU.
- µClinux a été porté en premier lieu sur le microcontrôleur Motorola M68328 DragonBall.
- www.uclinux.org

MMU=*Memory Management Unit*

PRESENTATION DE µClinux

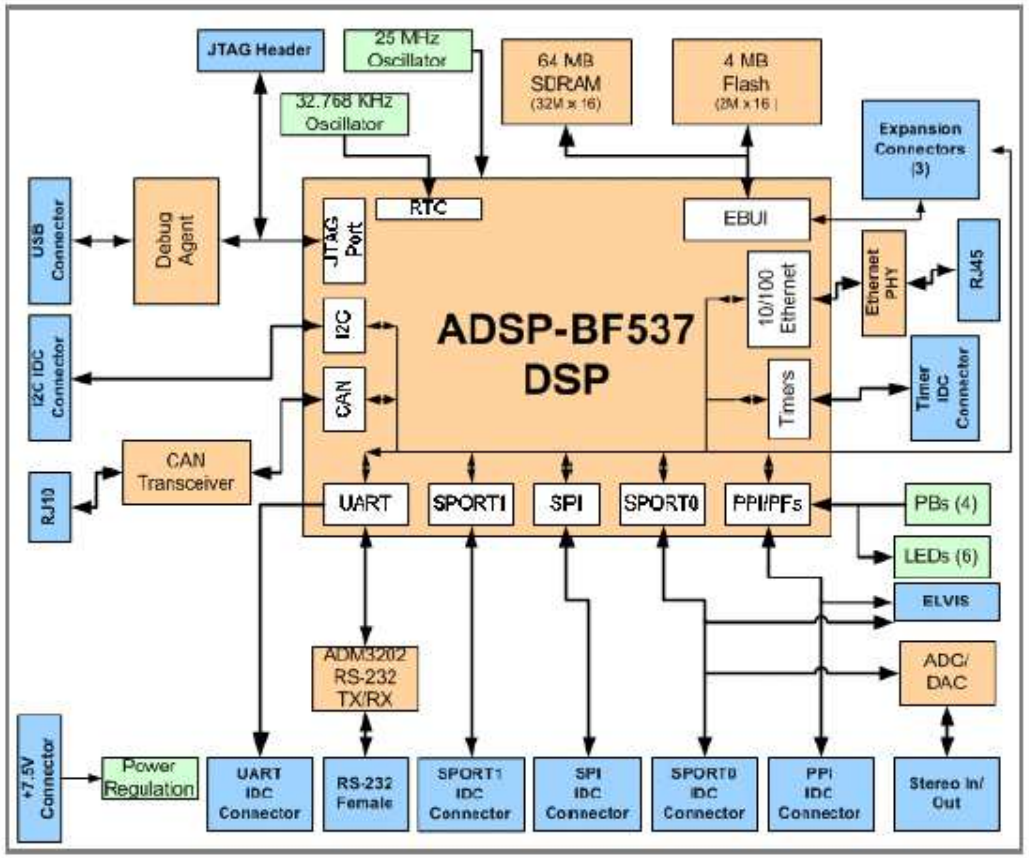
- µClinux a été ensuite porté sur différents processeurs :
  - Motorola DragonBall M68328.
  - ColdFire.
  - Blackfin.
  - ARM7.
  - MicroBlaze sans MMU (abandonné).
  - NIOS II sans MMU (abandonné).
  - …

µClinux vs Linux

- L'absence de MMU impose quelques limitations d'usage par rapport à l'environnement Linux :
  - La mémoire virtuelle n'existe pas.
  - La taille de la pile est fixe pour chaque processus.
  - L'appel système *fork()* n'est pas supporté. Il faudra utiliser alors une implémentation de l'appel système *vfork()* (le processus parent est suspendu jusqu'à ce que le processus fils appelle *exec()* ou *exit()*) .
  - L'appel système *exec()* ne peut pas charger actuellement une image binaire supérieure à 256 Ko.

MISE EN ŒUVRE DE µClinux SUR Blackfin

- Le processeur Blackfin est un processeur d'Analog Devices dédié au traitement du signal.

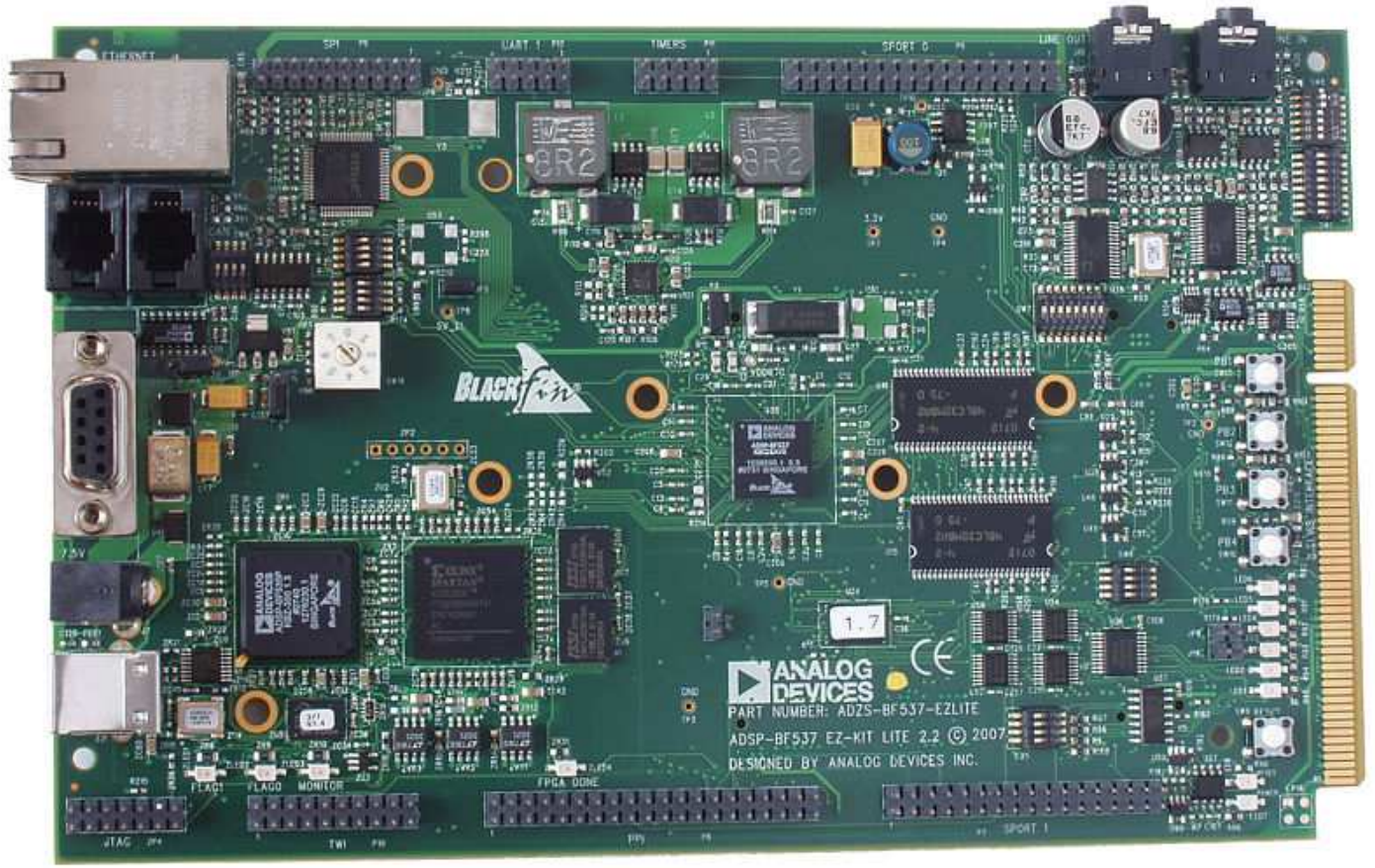


MISE EN ŒUVRE DE µClinux SUR Blackfin

- La mise en œuvre de µClinux sera faite sur carte Analog Devices Blackfin BF537 EZ-KIT Lite.

- La carte d'évaluation Blackfin BF537 EZ-KIT Lite possède :
  - Un processeur Blackfin BF537 à 600 MHz.
  - 64 Mo de mémoire SDRAM.
  - 4 Mo de mémoire Flash.
  - 1 port série.
  - Une liaison Ethernet IEEE 802.3 10/100BaseT.
  - Un port de *debug* JTAG.
  - …

MISE EN ŒUVRE DE µClinux SUR Blackfin



MISE EN ŒUVRE DE µClinux SUR Blackfin

- En étant simple utilisateur, dans son *Home Directory*, installer le *package* µClinux :

```
% cd  ~

% tar –xvzf uClinux.tgz
```

- Un répertoire *~/blackfin-linux-dist* est alors créé. Se placer dans ce répertoire :

```
% cd blackfin-linux-dist
```

MISE EN ŒUVRE DE µClinux SUR Blackfin

- Configurer le noyau µClinux comme sous Linux (choix de la carte, des applications à embarquer…) :

```
% make menuconfig
```

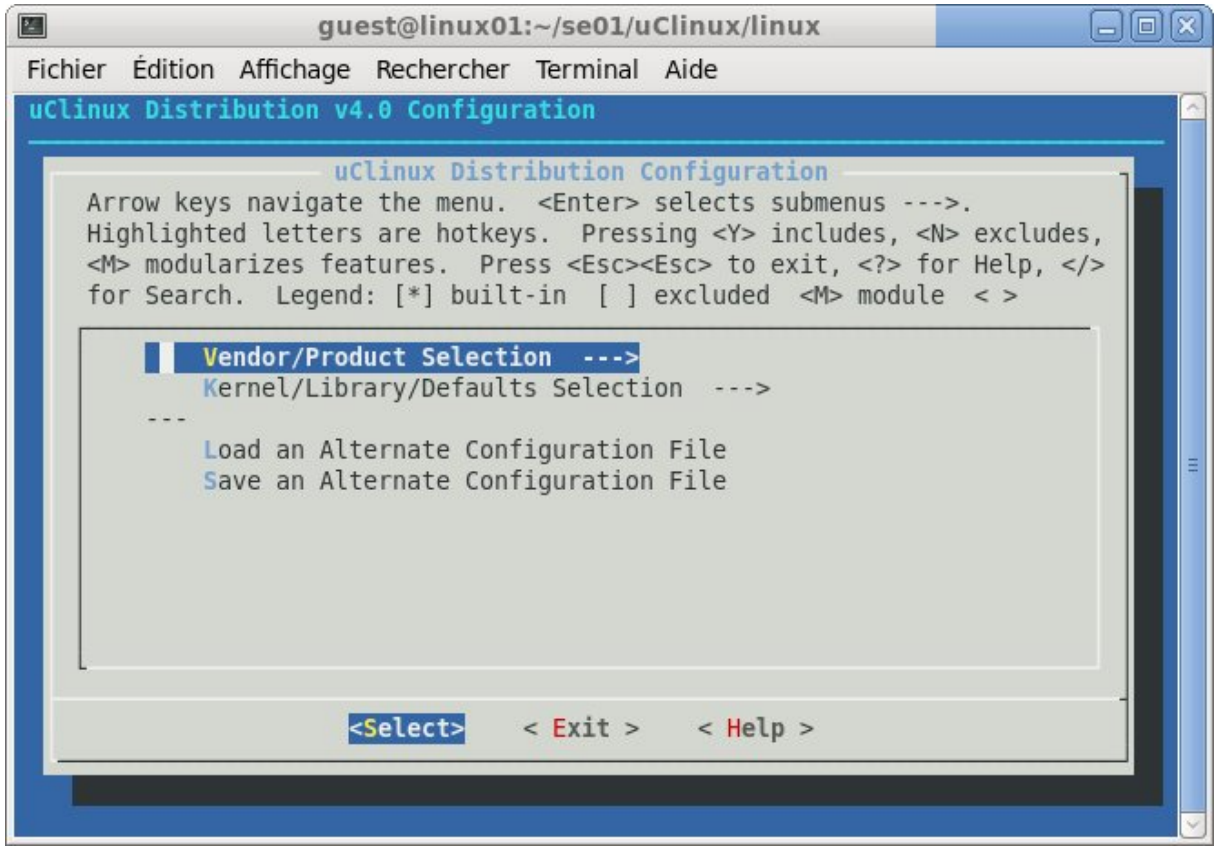
- Construire l'image à télécharger dans la carte :

```
% make
```

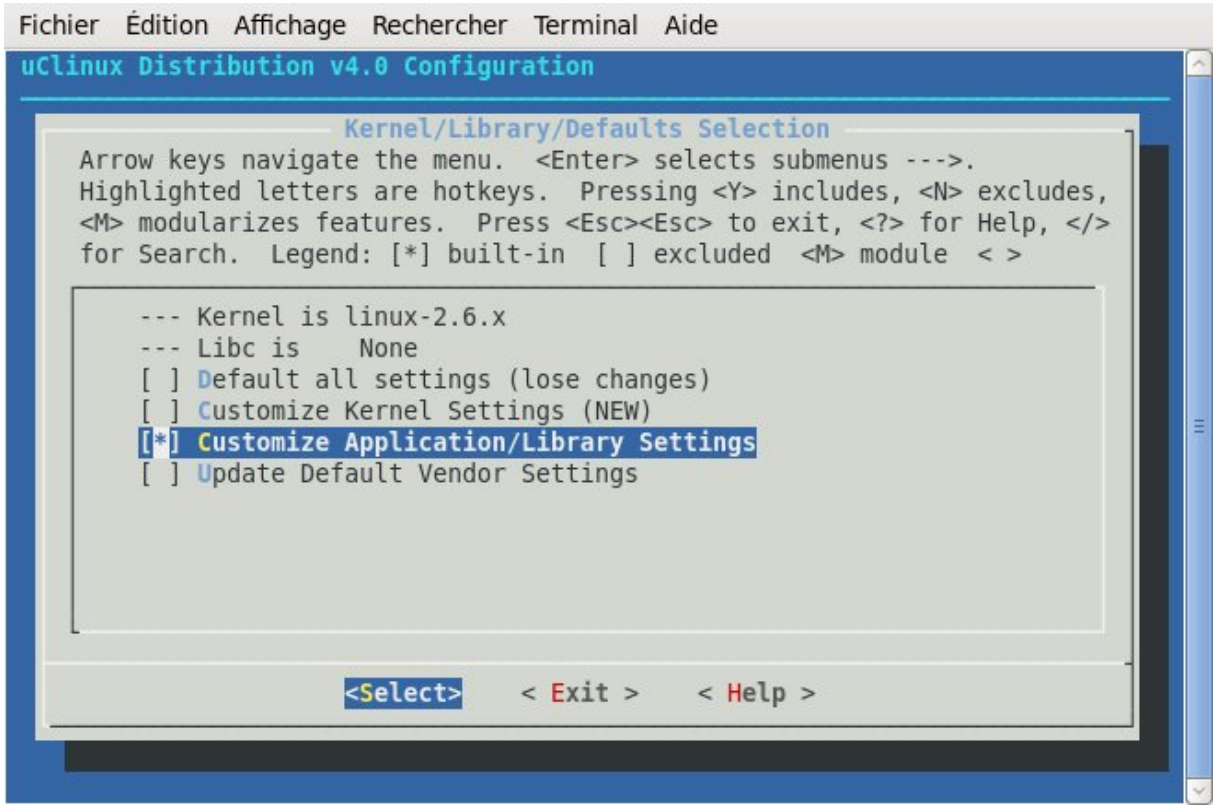
- Télécharger le fichier image (fichier *uImage* OS+RFS) dans la carte depuis le *bootloader* u-boot et lancer le noyau µClinux.

OS=*Operating System* RFS=*Root File System*

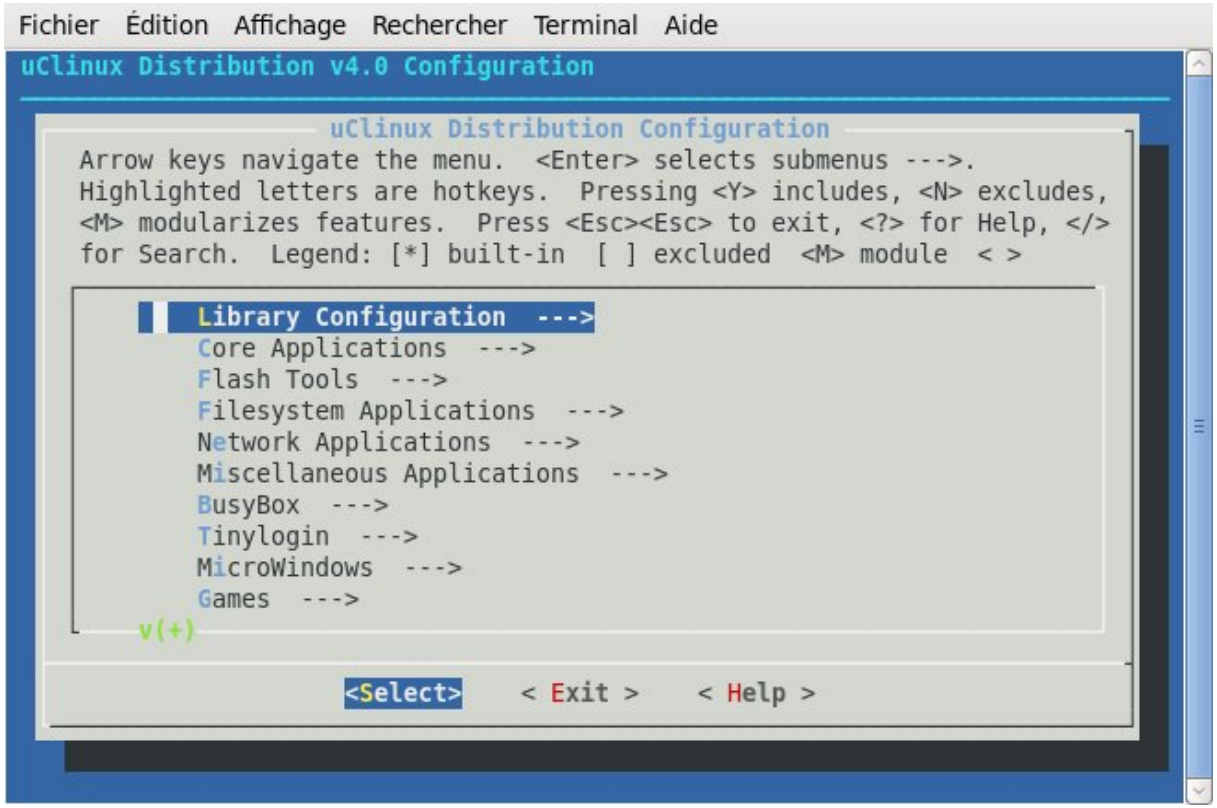
MISE EN ŒUVRE DE µClinux SUR Blackfin



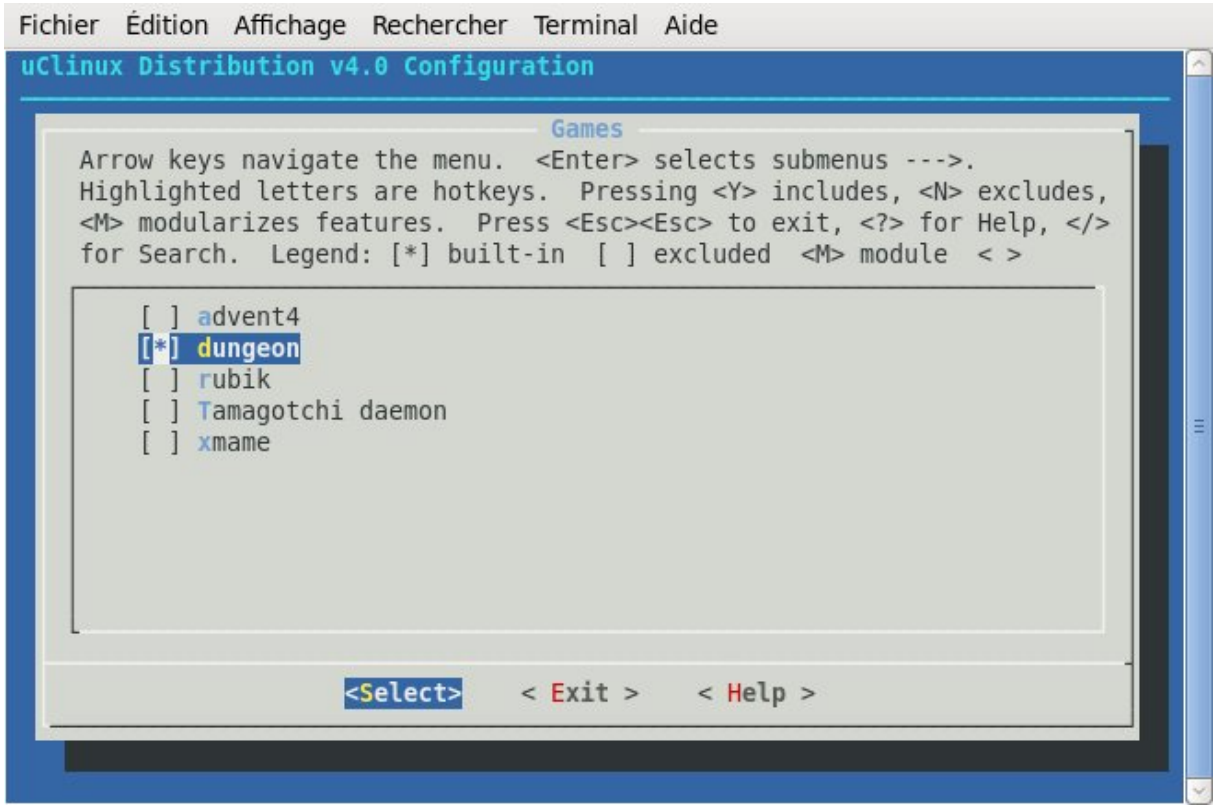
MISE EN ŒUVRE DE µClinux SUR Blackfin



MISE EN ŒUVRE DE µClinux SUR Blackfin



MISE EN ŒUVRE DE µClinux SUR Blackfin



- Le fichier image (fichier *uImage* OS+RFS) est recopié sous */tftpboot*.

- Ce fichier sera ensuite téléchargé par le réseau depuis le *bootloader* u-boot de la carte Blackfin par le protocole TFTP :

```
bfin> tftp uImage
bfin> bootm
```

- Il convient donc d'avoir un serveur TFTP actif sur le PC hôte.

PARTIE 7 :
BILAN

LE CHOIX D'UN LINUX EMBARQUE

Complexité de mise en œuvre maximale

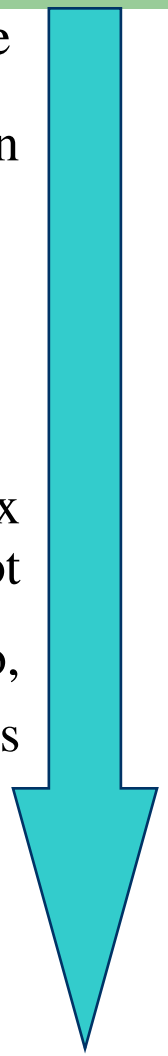
A la main

µClinux
Buildroot

Yocto,

Montavista, Intel, LinuxWorks

Complexité de mise en œuvre minimale



CHAPITRE 7 :
LE TEMPS REEL SOUS LINUX
INTRODUCTION AU TEMPS REEL

OPTION SE

Voir le cours « Systèmes d'exploitation Temps Réel »

PARTIE 1 :
INTRODUCTION

TEMPS REEL DUR

- On parle de **Temps Réel dur** (*hard Real Time*) quand les événements traités trop tardivement ou perdus provoquent des conséquences catastrophiques pour la bonne marche du système (perte d'informations cruciales, plantage…).

- Les systèmes à contraintes dures ne tolèrent qu'une gestion stricte du temps afin de conserver l'intégrité du service rendu.

- On citera comme exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique.

TEMPS REEL DUR

- **Ces systèmes garantissent un temps maximum d'exécution dans 100 % des cas.**

- On a ici une répartition stricte et totalitaire du temps CPU entre tâches.

- On peut dire qu'un système Temps Réel dur doit être prévisible (*predictible),* les contraintes temporelles maximales garanties pouvant descendre à quelques µs.

- Le Temps Réel dur est celui de l'électronicien apparu dans les années 1960.

TEMPS REEL MOU

- On parle de **Temps Réel mou** (*soft Real Time*) quand les événements traités trop tardivement ou perdus sont sans conséquence catastrophique pour la bonne marche du système.

- **On ne garantit pas un temps de traitement maximum dans 100 % des cas tout en essayant de s'y rapprocher.**

TEMPS REEL MOU

- On peut citer l'exemple des systèmes multimédia : si quelques images ne sont pas affichées correctement, cela ne met pas en péril le fonctionnement correct de l'ensemble du système. Au bout de quelques images perdues, le système retrouve son fonctionnement normal. Cela ne nuit pas à l'usager…

- Le Temps Réel mou ne doit pas être utilisé s'il met en jeu la sécurité des personnes.

- Le Temps Réel mou est celui de l'informaticien apparu dans les années 2000.

TEMPS REEL ET QoS

- Le Temps Réel dur et le Temps Réel mou gèrent le temps de façon strict ou plus lâche.
- Cela définit une certaine Qualité de Service (QoS) sur la gestion du temps.
- Le Temps Réel peut être donc vu comme une QoS offerte à une application sur la gestion du temps !
- **La QoS est imposée par le processus et son environnement extérieur à contrôler non pas par le système de contrôle lui-même !**

QOS=*Quality of Service*

TEMPS REEL ET QoS

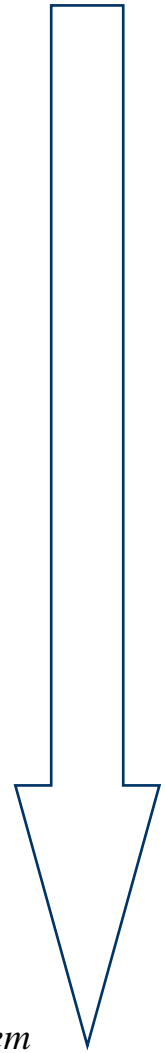
QoS lâche

Best effort : temps partagé — **GPOS**

Temps réel mou — **RTOS**

Temps réel dur — **RTOS**

QoS stricte



GPOS=*General Purpose Operating System*

LINUX VANILLA ET LE TEMPS REEL

- Le noyau Linux de base ou standard sans application de modifications (*patchs*) sur les sources est appelé noyau *vanilla.*
- Les sources du noyau *vanilla* sont disponibles sur le site : www.kernel.org
- On considère alors le noyau *vanilla* avec sa configuration standard sans amélioration de la préemption :
  - Configuration *No Forced Preemption* (PREEMPT_NONE).
- A noter l'existence des configurations améliorant la préemption du noyau *vanilla* : *Voluntary Kernel Preemption* (PREEMPT_VOLONTARY) et *Preemptible Kernel* (PREEMPT).

LINUX VANILLA ET LE TEMPS REEL

- Le noyau *vanilla* n'est pas un système d'exploitation Temps Réel dur car :
  - Le noyau Linux possède de longues sections de code où tous les événements extérieurs sont masqués (non interruptible).
  - Le noyau Linux n'est pas préemptible durant toute l'exécution d'un appel système par un processus et ne le redevient qu'en retour d'appel système.

LINUX VANILLA ET LE TEMPS REEL

- Le noyau *vanilla* n'est pas un système d'exploitation Temps Réel dur car :
  - Le noyau Linux n'est pas préemptible durant le service d'une interruption ISR (*Interrupt Sub Routine*). La routine ISR acquitte **immédiatement** l'interruption (*Top Half*) puis programme un traitement **différé** des données dans une file (*workqueue*) par un *thread* du noyau (*Bottom Half*).
  - En cas d'occurrence d'une interruption durant l'exécution d'un appel système en mode noyau, le BH programmé par l'ISR (et éventuellement les autres BH des autres ISR) ne sera exécuté qu'à la fin de l'exécution complète de l'appel système d'où un temps de latence important et non borné !

LINUX VANILLA ET LE TEMPS REEL

- Le noyau *vanilla* n'est pas un système d'exploitation Temps Réel dur car :
  - L'ordonnanceur de Linux essaye d'attribuer de façon équitable le CPU à l'ensemble des processus. C'est une approche égalitaire et *best effort*.
  - La gestion des priorités de processus Linux est secondaire.

LINUX VANILLA ET LE TEMPS REEL

- **Le noyau *vanilla* ne peut être pas être considéré comme Temps Réel dur. C'est un système d'exploitation généraliste GPOS.**

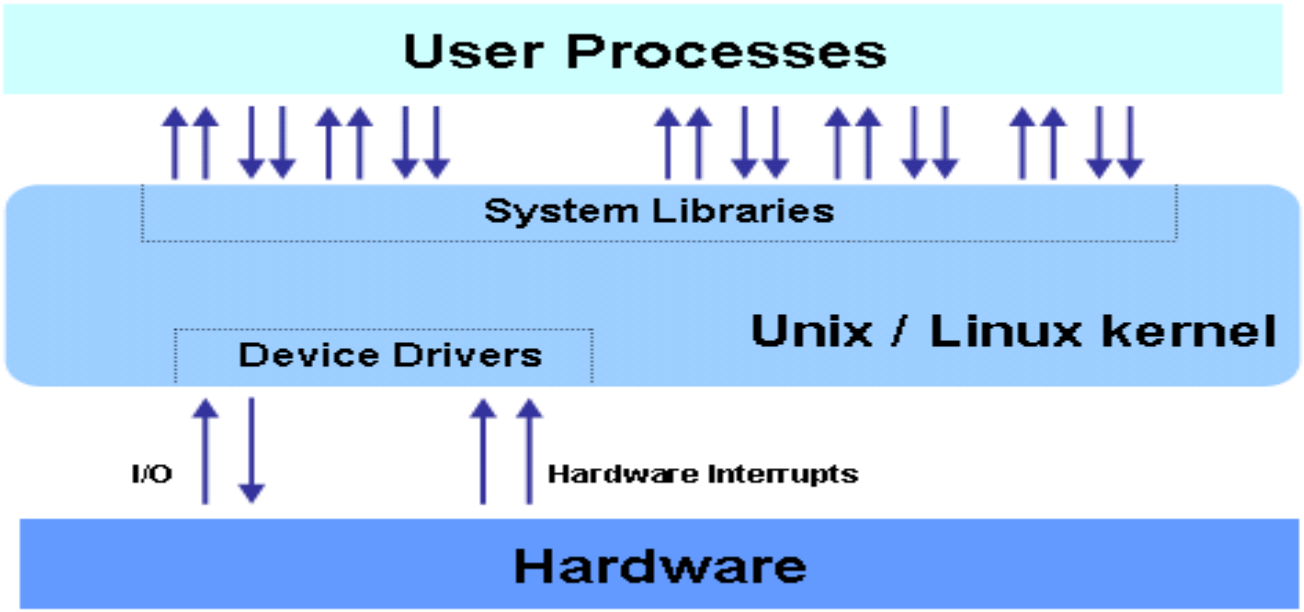
- Il existe néanmoins des solutions **Linux Temps Réel dur** pour une réactivité garantie dans 100 % des cas de quelques µs à quelques dizaines de µs.

- Il existe des solutions **Linux Temps Réel mou** pour une réactivité non garantie dans 100 % des cas de quelques dizaines de µs à quelques centaines de µs.

GPOS=*General Purpose Operating System*

EXTENSIONS TEMPS REEL POUR LINUX

- Implémentation du noyau Linux *vanilla* :
  - Pas de support du Temps Réel.
  - Séparation entre le matériel et les processus Linux.
  - ...



EXTENSIONS TEMPS REEL POUR LINUX

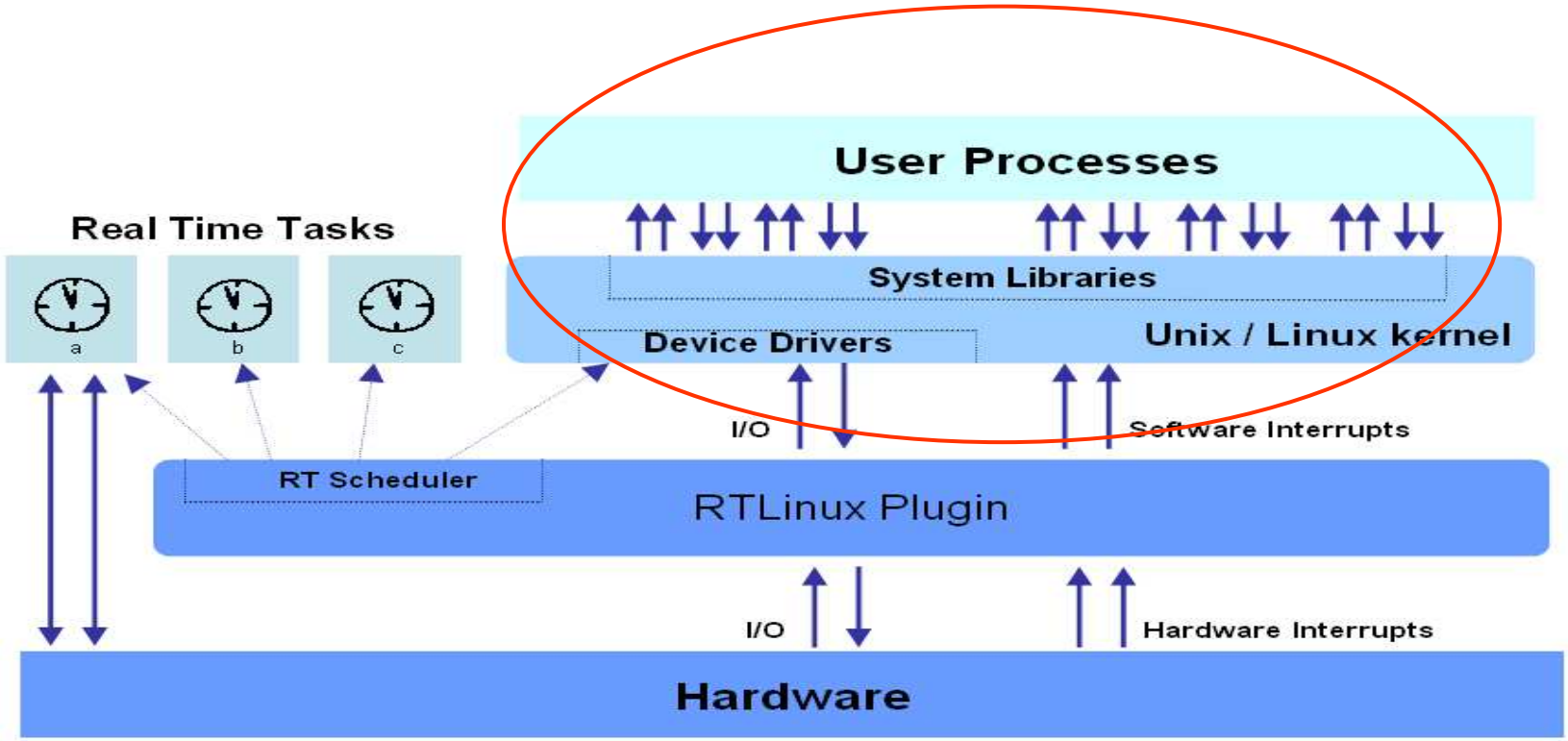
- Solution 1 pour une **extension Temps Réel mou** de Linux :
  - Modification du noyau Linux par application de *patchs* pour améliorer les contraintes Temps Réel : dévalider les interruptions le moins longtemps possible, appeler l'ordonnanceur le plus souvent possible (fonction *schedule()* du noyau), en retour d'interruption par exemple, réduction des temps de latence.

EXTENSIONS TEMPS REEL POUR LINUX

- Solution 2 pour une **extension Temps Réel dur** de Linux :
  - Ajout d'une couche d'abstraction entre le matériel et le noyau Linux.
  - Ajout d'un deuxième ordonnanceur (ou co-noyau) de tâches Temps Réel dur et considérer le noyau Linux et ses processus comme tâche de fond.
  - Pas de séparation entre le matériel et les tâches Temps Réel.
  - Plus difficile que la première solution.

EXTENSIONS TEMPS REEL POUR LINUX

- Solution 2 pour une **extension Temps Réel dur** de Linux :



LINUX COMME RTOS

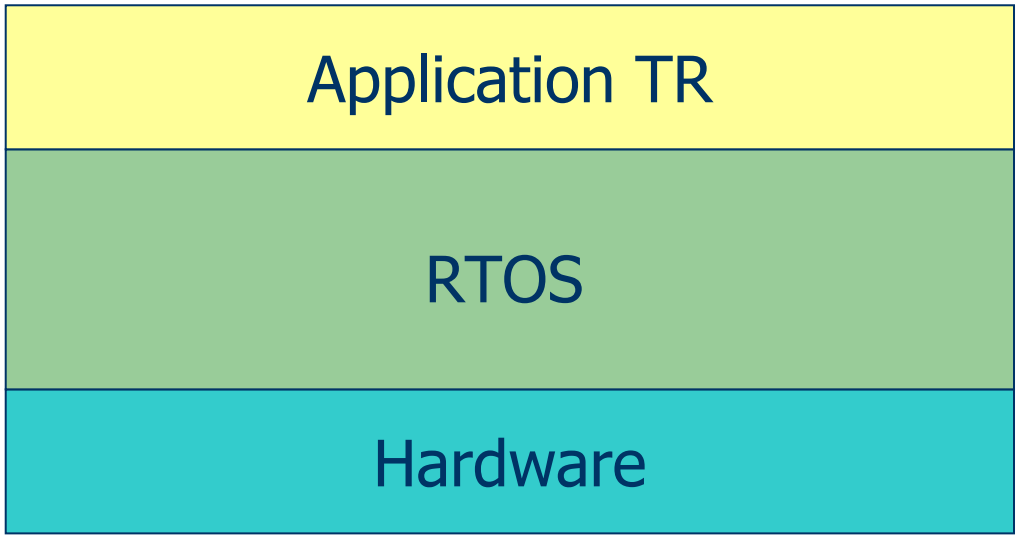
- Avantages : de nombreuses fonctionnalités, coopération entre tâches Temps Réel et processus Linux non Temps Réel.

- Inconvénients : n'est pas un véritable RTOS monolithique.

RTOS MONOLITHIQUE

- Avantages : simplicité, monolithique, fait pour le Temps Réel dur.

- Inconvénients : fonctionnalités plus ou moins limitées.



PARTIE 2 :
LES OFFRES LINUX TEMPS REEL

LES OFFRES LINUX TEMPS REEL

- Les offres de version de Linux embarqué et Temps Réel peuvent être rangées dans l'une des 2 catégories suivantes :
  - Les distributions Linux Temps Réel commerciales.
  - Les distributions Linux Temps Réel libres.

- Nous dresserons la liste des principales offres.

LINUX TEMPS REEL COMMERCIAL

- Montavista (Professional Edition) :
  www.mvista.com

- Intel (Wind River/RTLinux) :
  www.windriver.com

- LynuxWorks (BlueCat RT) :
  www.lynuxworks.com

LINUX TEMPS REEL OPEN SOURCE

- Xenomai :
  xenomai.org

- PREEMPT-RT :
  rt.wiki.kernel.org

- RTAI (*Real Time Application Interface*) :
  www.aero.polimi.it/~rtai

RTOS COMMERCIAUX

- Il y a toujours les RTOS commerciaux non Linux :
  - VxWorks (Intel).
  - µC/OS II (Micrium).
  - pSOS.
  - QNX.
  - …

RTOS OPEN SOURCE

- FreeRTOS :

  www.freertos.org

- eCOS :

  sources.redhat.com/ecos

XENOMAI

- Mise en place d'un noyau Temps Réel dur (générique) par extension de Linux. Pour Xenomai 3, il s'agit de Xenomai *Cobalt*.
- Emulation d'API propriétaires : VxWorks, pSOS.
- API standard POSIX.
- Programmation en mode noyau et en mode utilisateur.
- Il est possible d'utiliser Xenomai juste avec le noyau Linux hôte (à combiner avec le patch PREEMPT-RT). Pour Xenomai 3, il s'agit de Xenomai *Mercury*.

POSIX=*Portable Operating System Interface*

XENOMAI

- Xenomai est sûrement la solution Temps Réel dur libre sous Linux la plus prometteuse.

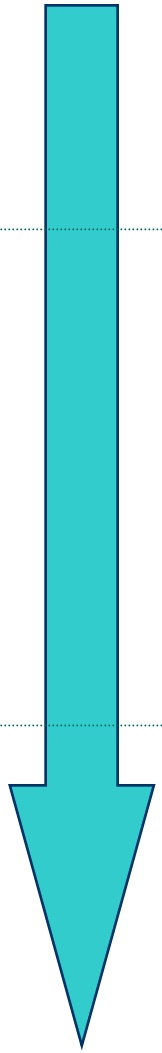
- Elle est à privilégier.

PARTIE 3 :
BILAN

LE CHOIX D'UN LINUX TEMPS REEL

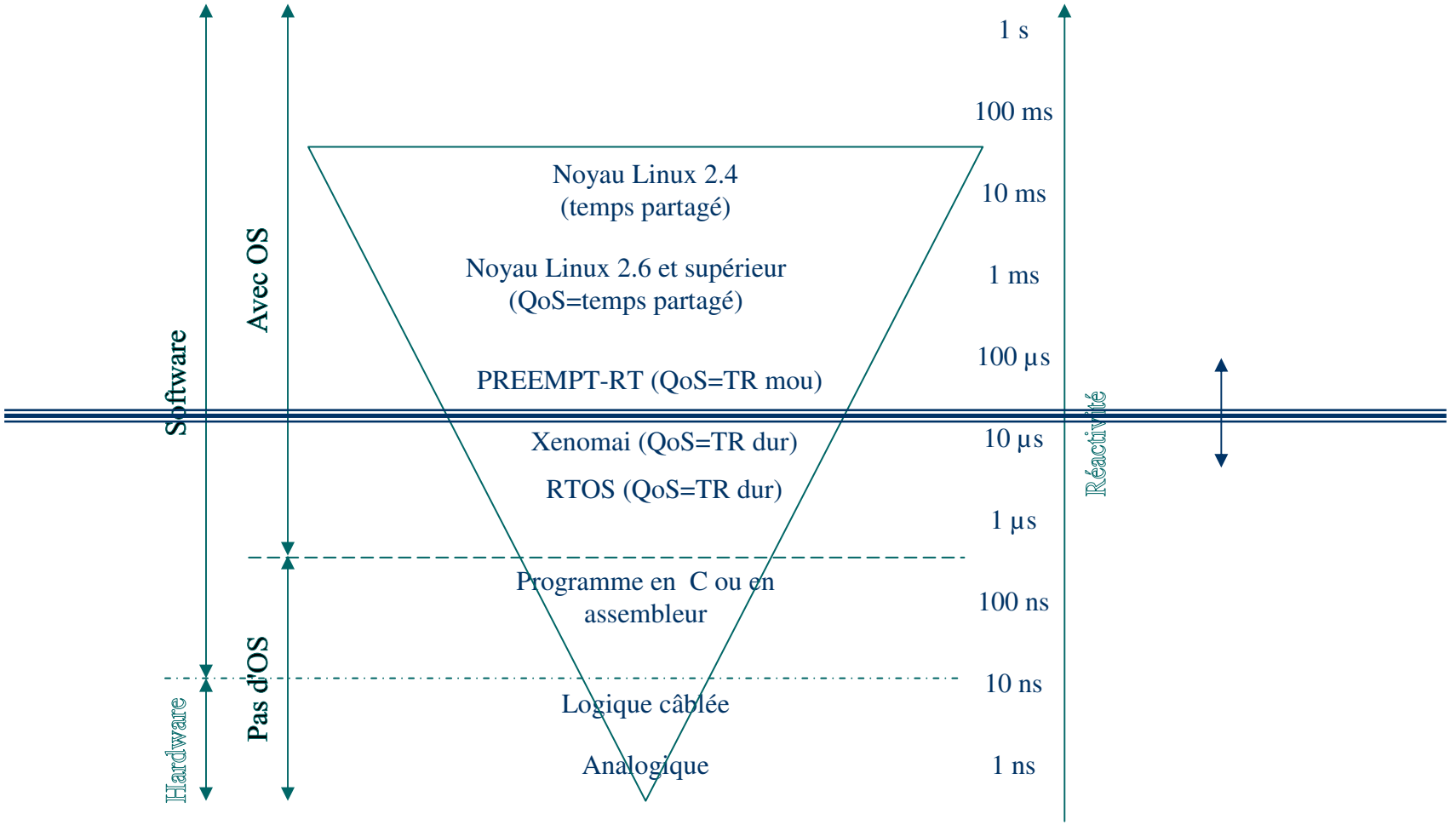
- Le choix est à faire en fonction des contraintes Temps Réel que doit respecter le système :
  - Pas de contraintes. Temps partagé. Réactivité **en moyenne** de qq 10ms à qq 100 ms :
    - Linux vanilla 2.6 et supérieur.
  - Temps Réel mou. Réactivité au plus de qq 10 µs à qq 100 µs:
    - PREEMPT-RT.
  - Temps Réel dur. Réactivité **au plus** de qq µs à qq 10 µs :
    - Xenomai.

LE CHOIX D'UN LINUX TEMPS REEL



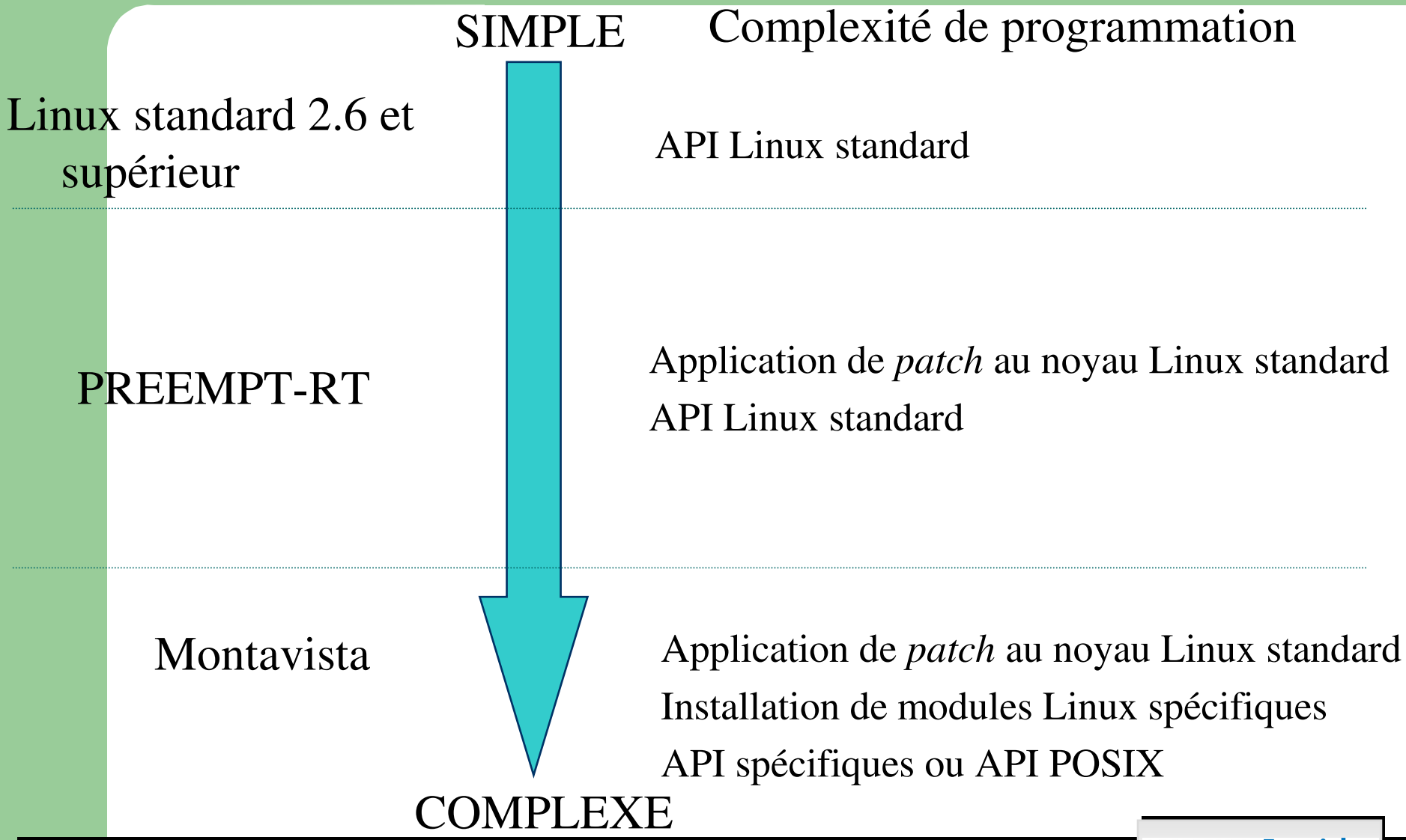
	QoS	Réactivité
Linux standard 2.6 et supérieur	Temps partagé	qq 10-100 ms
PREEMPT-RT	Temps Réel mou	qq 10-100 µs
Xenomai	Temps Réel dur	qq 1-10 µs

LE CHOIX D'UN LINUX TEMPS REEL



NB : la QoS est un paramètre supplémentaire imposé par le processus à contrôler

LE CHOIX D'UN LINUX TEMPS REEL



LE CHOIX D'UN LINUX TEMPS REEL

- Le choix final se fera en fonction des contraintes temporelles imposées par le processus à contrôler depuis Linux en prenant aussi en compte la complexité de mise en oeuvre dans chaque cas :
  - Configuration du noyau.
  - Ecriture de l'application Temps Réel.

- Choisir là aussi un Linux Temps Réel commercial est rassurant. Cela a aussi un coût.

CHAPITRE 8 :
CONTRÔLE ET COMMUNICATION DES OBJETS CONNECTES

PARTIE 1 :
INTRODUCTION

INTRODUCTION

- Les objets connectés restent *in fine* des systèmes embarqués que l'on contrôle à distance directement par Internet ou via Internet.

- Ils diffèrent néanmoins d'un système embarqué classique car :
  - Ils utilisent généralement un lien de communication sans fil (LoRa, Sigfox)… : bande ISM (*Industriel, Scientifique et Médical*), longue portée et faible consommation.
  - Le débit et la quantité de données échangées sont faibles : quelques dizaines d'octets par jour (pour télécontrôle, télérelevé…).
  - L'électronique basse consommation augmente l'autonomie sur batterie.

INTRODUCTION

- Il convient de voir quelles méthodes de contrôle à distance on peut utiliser.

- Il existe beaucoup de méthodes de contrôle à distance qui mettent en œuvre différents *middlewares* et protocoles :
  - *Web Services*, REST, XMPP, AMPQ, PubSubHubBub, CoAP, DDS – RTPS, XML, EMLL, Atom RSS, JSON/BSON/UBSON, KML, oBIX, COBR, SIP, SNMP.
  - **HTTP, MQTT**.
  - …

D'après D. Donsez. Internet des Choses : Aspects Intergiciels

INTRODUCTION

- Le contrôle à distance d'un objet connecté peut se faire par différents modes d'interaction, principalement :
  - Interaction synchrone : par commande/réponse. On interroge l'objet qui exécute une action et renvoie une réponse.
  - Interaction asynchrone : par événement. L'objet envoie un événement ou une alarme.

- Nous allons voir 2 protocoles utilisés dans l'IoT illustrant ces 2 modes d'interaction :
  - Protocole HTTP : *Hyper Text Transport Protocol*.
  - Protocole MQTT : *Message Queuing Telemetry Transport*.

INTRODUCTION

- Pour les liens de communication sans fil, les objets connectés mettent en œuvre des réseaux LPWAN (*Low Power Wide Area Network*) comme LoRaWAN utilisant des techniques de modulations avancées comme LoRa.

- Nous allons donc voir sommairement la modulation LoRa qui concerne la couche physique du modèle OSI (*Open System Interconnexion*) pour nous attarder sur l'infrastructure LoRaWAN illustré par un exemple à travers le réseau communautaire TTN (*The Things Network*).

PARTIE 2 :
PROTOCOLE HTTP

PROTOCOLE HTTP

- Le protocole HTTP a été créé en 1990 par Tim Berners-Lee, ingénieur au CERN et a ensuite été normalisé par l'IETF :
  - 1996 : RFC 1945. HTTP/1.0.
  - 1997 : RFC 2068. HTTP/1.1.
  - 1999 : RFC 2616. HTTP/1.1.
  - 2014 : RFC 7230 à 7237. HTTP/1.1 republié.

- HTTP est un protocole synchrone du type commande/réponse.

IETF=*Internet Engineering Task Force* *RFC=Request For Comments*

PROTOCOLE HTTP

- HTTP est le protocole utilisé entre un serveur Web et un client Web ou navigateur.

- HTTP est utilisé au-dessus de TCP/IP et utilise le numéro de port 80 pour la communication non chiffrée et 443 pour la communication chiffrée utilisant SSL/TLS.

PROTOCOLE HTTP

- Une commande HTTP est de la forme (notation BNF) :

```
Request =
Request-Line
*(( general-header
| request-header
| entity-header ) CRLF)
CRLF
[ message-body ]
```

```
Request-Line =
Method SP Request-URI SP HTTP-Version CRLF
```

BNF=*Backus Notation Form*

PROTOCOLE HTTP

- Une commande HTTP est de la forme :

```
Method =
"GET"
| "HEAD"
| "POST"
| "PUT"
| "DELETE"
| "TRACE"
| "CONNECT"
```

PROTOCOLE HTTP

- Un exemple de commande HTTP est :

```
GET /exam.html HTTP/1.1
Host: brahmane.enseirb.fr:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:62.0) Gecko/20100101 Firefox/62.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

PROTOCOLE HTTP

- Une réponse HTTP est de la forme :

```
Response =
Status-Line
*(( general-header
| response-header
| entity-header ) CRLF)
CRLF
[ message-body ]
```

```
Status-Line =
HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

PROTOCOLE HTTP

- Une réponse HTTP est de la forme :

Status-Code =

```
"100" ; Continue
| "101" ; Switching Protocols
| "200" ; OK
. . .
| "301" ; Moved Permanently
. . .
| "305" ; Use Proxy
. . .
| "400" ; Bad Request
. . .
| "403" ; Forbidden
| "404" ; Not Found
. . .
| "500" ; Internal Server Error
| "501" ; Not Implemented
. . .
```

PROTOCOLE HTTP

- Une réponse HTTP est de la forme :

Le `Status-Code` est ainsi classé :

- 1xx : Informational - Request received, continuing process.
- 2xx : Success - The action was successfully received, understood, and accepted.
- 3xx : Redirection - Further action must be taken in order to complete the request.
- 4xx : Client Error - The request contains bad syntax or cannot be fulfilled.
- 5xx : Server Error - The server failed to fulfill an apparently valid request.

PROTOCOLE HTTP

- Un exemple de réponse HTTP est :

```
HTTP/1.1 200 OK
Date: Wed, 03 Oct 2018 10:59:51 GMT
Server: Apache/2.4.10 (Unix) mod_jk/1.2.40
Last-Modified: Mon, 04 May 2015 12:15:04 GMT
ETag: "434-515408308420O"
Accept-Ranges: bytes
Content-Length: 1076
Connection: close
Content-Type: text/html

<html>
<head>
<meta http-equiv="refresh" content="2; URL=exam.html">
<meta http-equiv="Content-Type" content="text/html; charset=iso-
   8859-1">
. . .
```

PROTOCOLE HTTP

- Dans le cas d'un objet connecté ou d'un système embarqué, il faut pouvoir interagir avec le matériel au travers du serveur HTTP.

- L'interaction se fera par extension du serveur Web via une interface pour pouvoir exécuter une action donc pour pouvoir interagir avec le matériel.

- On peut citer comme interface :
  - Interface CGI (*Common Gateway Interface*).
  - Connecteur Apache Tomcat (module mod_jk).

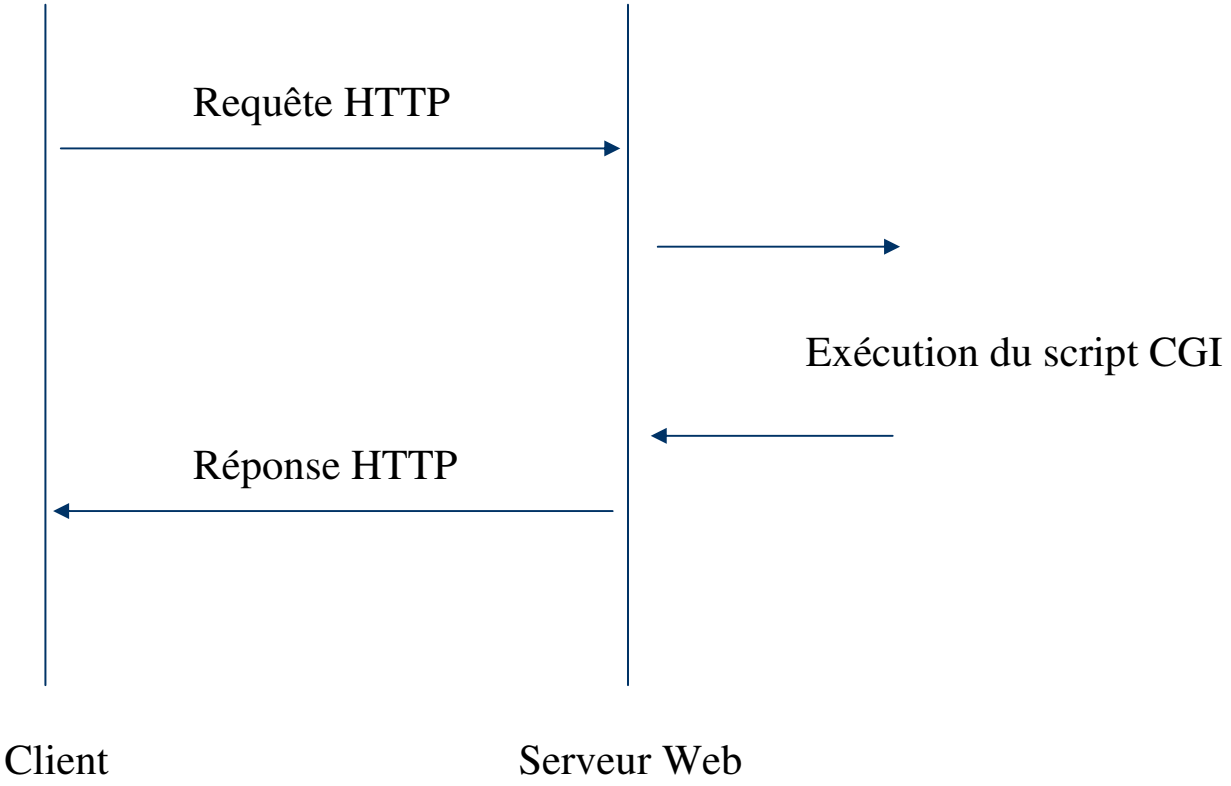
PROTOCOLE HTTP

- Interface CGI :
  - Serveurs Apache, Nginx et boa.
  - Cela permet d'exécuter un exécutable (exécutable obtenu après compilation ou bien *shell script*) côté serveur appelé script CGI.

- Connecteur Apache Tomcat :
  - Serveur Apache.
  - Cela permet d'exécuter une application Java côté serveur Web appelée *servlet* (pendant de l'*applet* côté *client*) pour interagir avec le matériel.

PROTOCOLE HTTP

- Fonctionnement de l'interface CGI :



PROTOCOLE HTTP

- Avec l'interface CGI, les E/S de la requête/réponse HTTP sont redirigées.

- Le serveur Web interagit avec le script CGI en entrée à l'aide de variables d'environnement :
  - Méthode GET : les données d'entrée sont récupérées par la script CGI à l'aide de la variable d'environnement QUERY_STRING (appel système `getenv()` pour un programme C).
  - Méthode POST (formulaire) : les données d'entrée arrivent sur l'entrée standard du script CGI et la variable d'environnement CONTENT_LENGTH fournit la taille en octets.

PROTOCOLE HTTP

- Une requête CGI est du style :

  `http://@IP:port/cgi-bin/mon_script`

  Exemple : `http://bf01/cgi-bin/enseirb?10`

  La valeur 10 représente le paramètre initialisant QUERY_STRING.

- Les scripts CGI sont placés traditionnellement dans le répertoire `cgi-bin/` du répertoire des fichiers du serveur Web.

- Le script CGI en sortie forge lui-même la réponse HTTP qui est au minimum (`printf()` pour un programme C) :

```
Status-Line
Content-Type: text/html CRLF
CRLF
[ message-body ]
```

PROTOCOLE HTTP

- L'interface CGI pose des problèmes de sécurité :
  - Le client qui n'est pas forcément sûr émet une requête qui provoque l'exécution d'un programme.
  - Le programme utilise alors les données pas forcément sûres du client.

- Pour la sécurité globale du système Linux embarqué, on ne doit donc pas accorder sa confiance au client.

PROTOCOLE HTTP

- Il existe différents serveurs HTTP (serveur Web) écrits dans différents langages de programmation :
  - C : Apache, Zeus Web Server, nginx, lighttpd, Cherokee, Hiawatha Webserver.
  - ASP/ASP.Net (C#, VB.net) : IIS.
  - Java : Tomcat, Jetty, GlassFish, JBoss, JOnAS, Vert.x.
  - Python : Zope.
  - Ruby : WEBrick, Mongrel, Thin.
- Il existe des versions avec une faible empreinte mémoire adaptées aux systèmes embarqués :
  - C : boa, busybox.
  - Apache ou Nginx deviennent acceptables si l'on a suffisamment de mémoire et de place.

PARTIE 3 :
PROTOCOLE MQTT

PROTOCOLE MQTT

- Le protocole MQTT a été créé en 1999 par Andy Stanford-Clark d'IBM et Arlen Nipper d'Arcom. Bien que finalement pas récent, il a trouvé un regain d'intérêt avec les objets connectés.

- MQTT possède sa norme et son site :
  - mqtt.org
  - docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

PROTOCOLE MQTT

- MQTT est un protocole asynchrone du type événement *publish/subscribe*.

- MQTT est le protocole utilisé entre un client MQTT et un serveur MQTT appelé aussi *broker* (courtier).

- MQTT est utilisé au-dessus de TCP/IP et utilise le numéro de port 1883 pour la communication non chiffrée et 8883 pour la communication chiffrée utilisant SSL/TLS.

PROTOCOLE MQTT

- MQTT est un protocole ouvert, simple et léger pour répondre aux besoins suivants :
  - Utilisation avec des réseaux sans fils
  - Limitation de la consommation en énergie par un protocole compact (entête de 2 octets seulement).
  - Fiabilité (QoS).

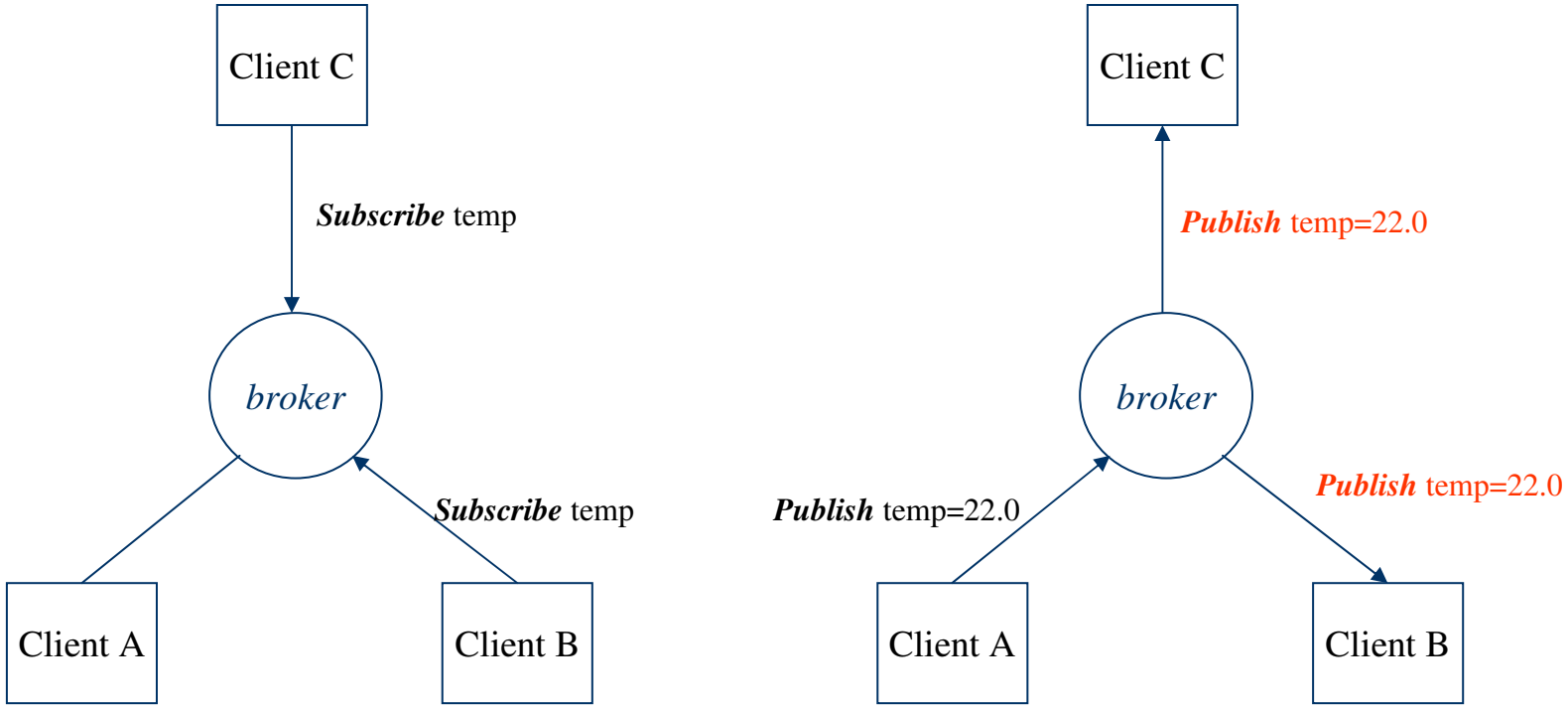
PROTOCOLE MQTT

- Le *broker* MQTT relaye les messages des sources vers les clients qui s'y abonnent.

- Un client s'abonne par une souscription au *broker* à un sujet ou *topic*. Tous les messages correspondant à ce *topic* seront relayés par le *broker* aux abonnés.

- Les abonnés et les sources de messages n'ont pas à se connaître directement et ne communiquent qu'à travers les *topics*.

PROTOCOLE MQTT



PROTOCOLE MQTT

- Un *topic* se présente sous forme d'une chaîne de caractères avec une hiérarchie de *sub-topics* séparés par le caractère / (analogie avec les OID du protocole SNMP) :
  - `maison/salon/temperature`.
  - `maison/cuisine/temperature`.

- Il existe des caractères *wildcards* :
  - `maison/+/température` : capteurs de température de toutes les pièces de la maison.
  - `maison/salon/#` : tous les capteurs du salon.
  - `maison/#` : tous les capteurs de la maison.

PROTOCOLE MQTT

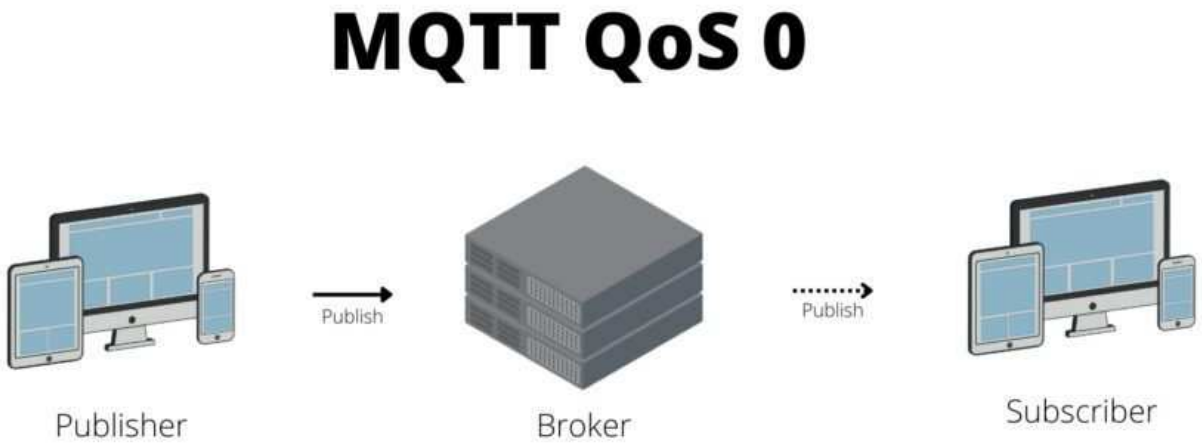
- Il est aussi possible de sécuriser les échanges MQTT si besoin par :
  - Un transport des données chiffrées SSL/TLS.
  - Une authentification par certificat SSL/TLS.
  - Une authentification par nom/mot de passe.

PROTOCOLE MQTT

- MQTT intègre la notion de qualité de service QoS (*Quality of Service*) dans les échanges : :
  - QoS niveau 0 (*at most once*) (une fois max) : la source envoie le message tout au plus une fois. Le message est donc envoyé sans garantie de réception.
  - QoS niveau 1 (*at least once*) (au moins une fois) : la source envoie le message au moins une fois avec confirmation. Il pourra transmettre plusieurs fois le message s'il ne reçoit pas de confirmation. Le message peut être reçu plusieurs fois.
  - QoS niveau 2 (*exactly once*) (exactement une fois) : la source envoie le message une seule fois en utilisant une procédure sécurisée spécifique en 4 étapes..

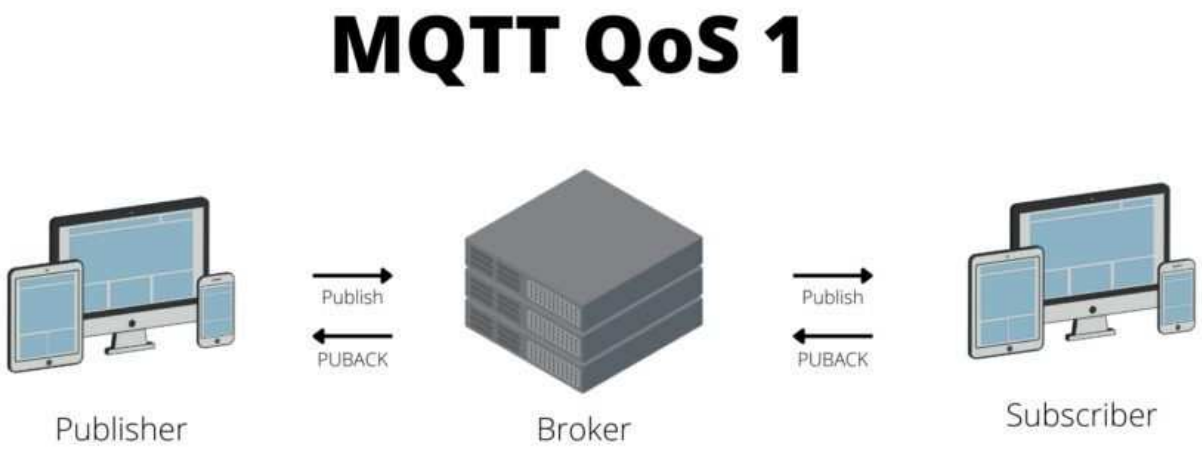
- NB : Un message pourrait être perdu au niveau application bien qu'au niveau transport, l'échange est en mode connecté et fiable (TCP).

PROTOCOLE MQTT



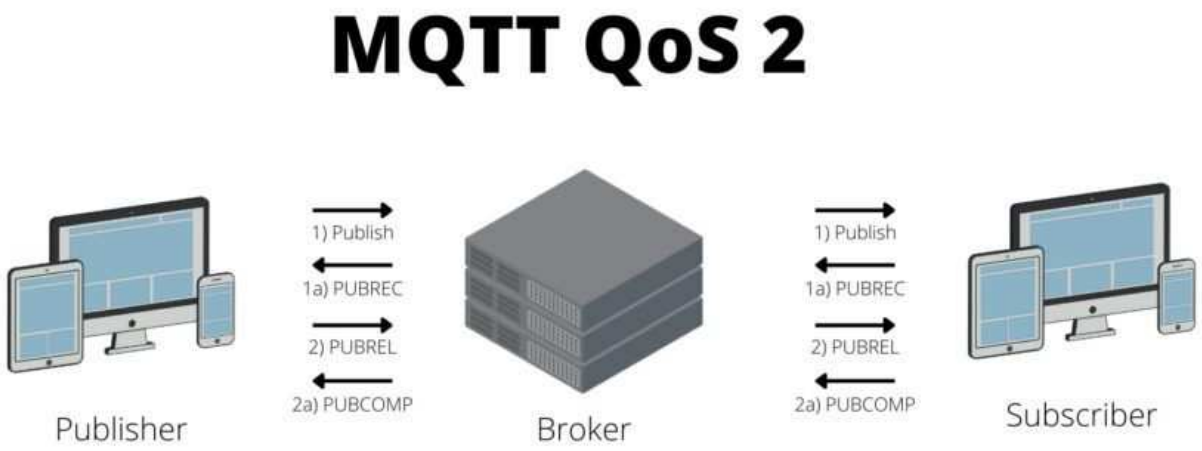
Source : cedalo.com/blog/understanding-mqtt-qos/

PROTOCOLE MQTT



Source : cedalo.com/blog/understanding-mqtt-qos/

PROTOCOLE MQTT



Source : cedalo.com/blog/understanding-mqtt-qos/

QoS publishing	QoS subscribing	QoS Broker -> Subscriber
0	1 ou 2	0
1 ou 2	0	0
2	1	1
1	1	1

Si la source utilise une QoS=2 et que le souscripteur a souscrit à une QoS=1, le *broker* délivre alors le message au souscripteur avec une QoS=1.

PROTOCOLE MQTT

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

Remaining Length : nombre d'octets restant dans le message

Entête MQTT de 2 octets

PROTOCOLE MQTT

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Champ *Control Packet Type*

PROTOCOLE MQTT

Control Packet	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	0	0	0	0
CONNACK	0	0	0	0
PUBLISH	DUP	QoS	QoS	RETAIN
PUBACK	0	0	0	0
PUBREC	0	0	0	0
PUBREL	0	0	1	0
PUBCOMP	0	0	0	0
SUBSCRIBE	0	0	1	0
SUBACK	0	0	0	0
UNSUBSCRIBE	0	0	1	0
UNSUBACK	0	0	0	0
PINGREQ	0	0	0	0
PINGRESP	0	0	0	0
DISCONNECT	0	0	0	0

Champ *Flags*

PROTOCOLE MQTT

- Il existe différentes implémentations de clients et de serveurs MQTT écrits dans différents langages de programmation :
  - C, C++, Java, Python, JS (Node.JS), Lua…

- Projet Mosquitto : mosquitto.org

HTTP VS MQTT

- HTTP est un protocole synchrone du type *request/response* (commande/réponse). Il est simple mais la taille de l'entête est variable et importante. Il n'est pas toujours adapté pour les objets connectés sans fil. La sécurité est son point faible.

- MQTT est un protocole asynchrone du type *publish/subscribe*. Il est relativement simple et la taille de l'entête est constante et faible (2 octets). Il est parfaitement adapté pour les objets connectés sans fil. La sécurité est son point fort.

- On n'oublie pas que ces 2 protocoles sont au-dessus de TCP/IP qui possède une entête totale de (20+20) octets !

PARTIE 4 :
LORA ET LORAWAN

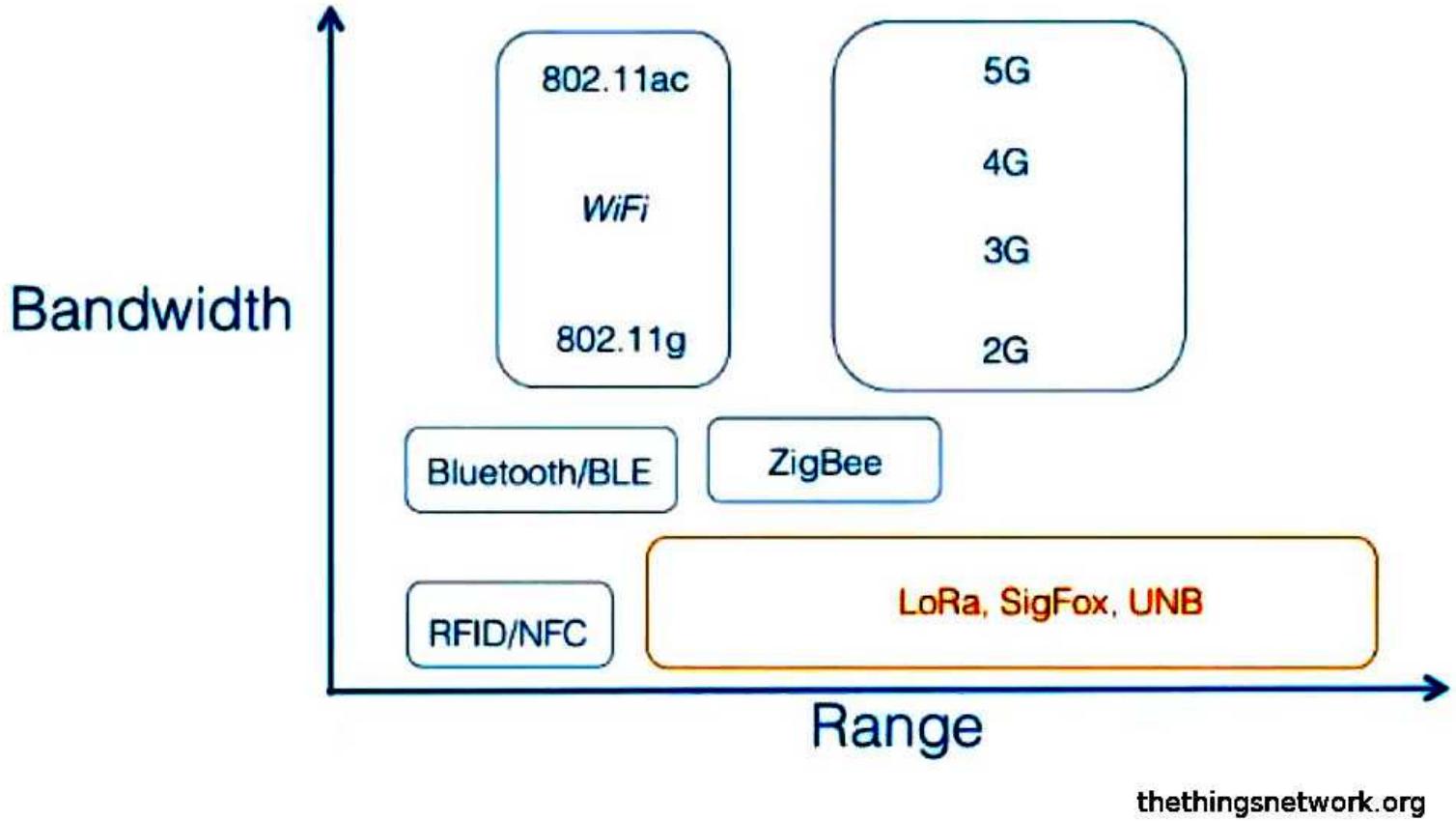
LORA

- LoRa (*Long Range*) est une technique de modulation numérique inventée par Semtech utilisée dans le réseau LPWAN (*Low Power Wide Area Network*) LoRaWAN.

- LoRa correspond à la couche physique du modèle OSI (*Open System Interconnexion*).

- LoRa utilise la bande de fréquences à usage libre ISM partagées avec d'autres technologies sans fil : 868 MHz en Europe, 915 MHz aux USA.

LORA


Bandwidth
802.11ac
WiFi
802.11g
5G
4G
3G
2G
Bluetooth/BLE
ZigBee
RFID/NFC
LoRa, SigFox, UNB
Range
thethingsnetwork.org

LORA

- LoRa est utilise une modulation à étalement de spectre pour une portée importante (plus de 10 km en moyenne si l'antenne est correcte) avec une sensibilité très importante côté récepteur.

- Le débit binaire est faible de 250 b/s à 11 kb/s selon le facteur d'étalement du spectre.

LORA

- La portée d'une communication LoRa dépend ainsi de sa bande passante, de sa puissance d'émission mais aussi du facteur d'étalement SF (*Spreading Factor*).

- L'étalement du signal augmente la portée mais réduit le débit binaire car il est transmis sur une plus longue période. Cela augmente aussi la consommation électrique.

LORA

- LoRa supporte 6 facteurs d'étalements (de SF7 à SF12) dans la bande de fréquences de 868 MHz.

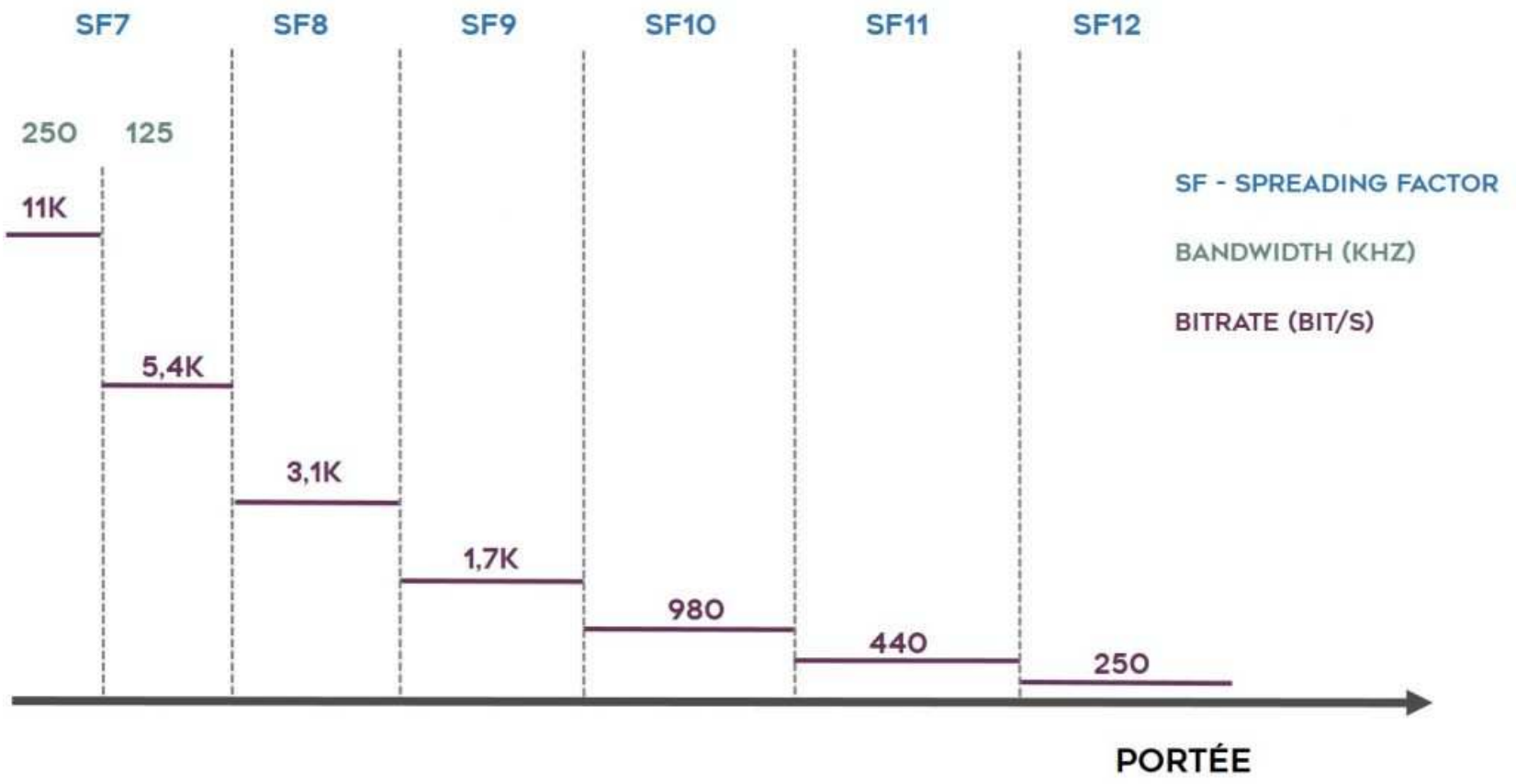
- L'algorithme ADR (*Adaptive Data Rate*) de gestion du débit adaptatif permet de changer au fil du temps la valeur SF d'un équipement terminal. Il s'agit d'avoir le débit le plus grand pour réduire le temps d'émission et donc la consommation d'énergie.

LORA

- Une passerelle LoRaWAN doit supporter au minimum les 3 canaux suivants : 868,10 MHz, 868,30 MHz et 868.50 MHz avec une bande passante de 125 kHz.

- L'infrastructure LoRaWAN pour la fréquence 868MHz doit respecter :
  - Puissance d'émission d'au plus +14 dBm.
  - Débit compris entre 250 b/s et 11kb/s.

LORA


SF7
SF8
SF9
SF10
SF11
SF12
250
125
11K
5,4K
3,1K
1,7K
980
440
250
SF - SPREADING FACTOR
BANDWIDTH (KHZ)
BITRATE (BIT/S)
PORTÉE

LORA ET ANTENNE

- Il faut noter l'importance est dans le choix de l'antenne. Tout est dans l'antenne car les communications numériques restent de la radioélectricité.

- Une antenne sous forme d'un point est théorique et n'existe pas. Elle est isotrope : son diagramme de rayonnement est une sphère.

- Son gain vaut 1 soit 0 dB_i si elle sert comme référence.

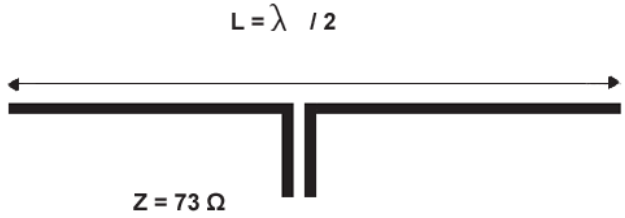
- Plus le gain d'une antenne est élevé, plus elle devient directive dans certaines directions. Elle a un diagramme d'antenne.

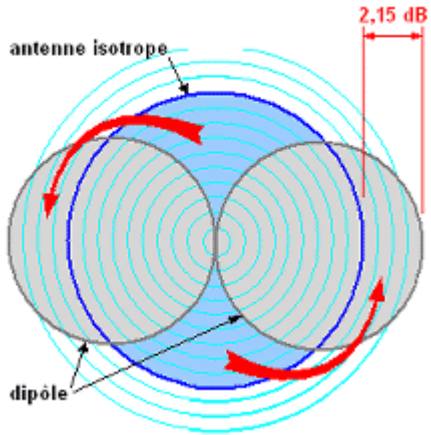
LORA ET ANTENNE

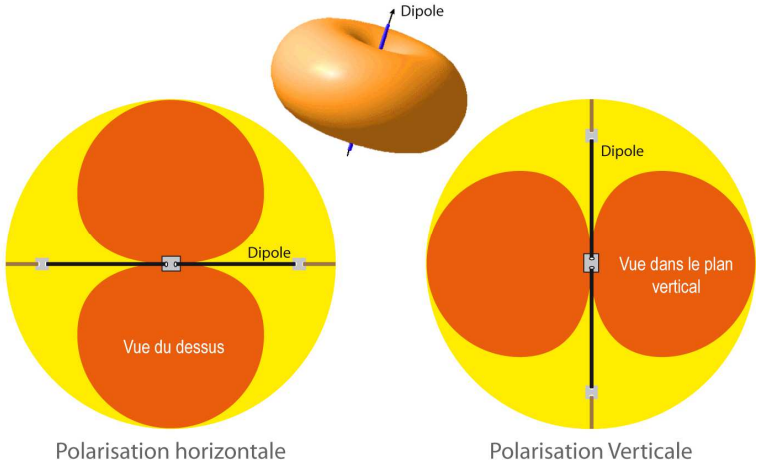
- Une antenne réaliste est le dipôle. Elle est directive et possède un gain de 2,15 dB_i.
- On rapporte généralement le gain des autres antennes au gain du dipôle : c'est le dB_d.
- Attention à l'arnaque commerciale d'utiliser les dB_i au lieu des dB_d :
  - gain en dB_d = gain en dB_i - 2,15 dB
    gain en dB_i = gain en dB_d + 2,15 dB

LORA ET ANTENNE

- Antenne dipôle :







Source : F5ZV, Radioscope.fr F4HTZ

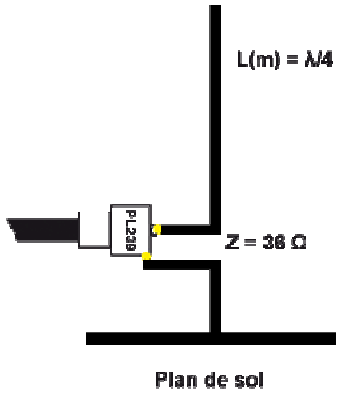
LORA ET ANTENNE

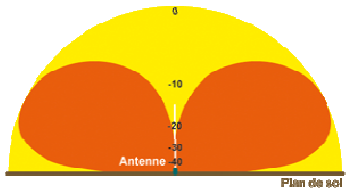
- Une antenne communément utilisée est l'antenne quart d'onde ($\lambda/4$) ou *ground plane* qui est verticale. L'autre partie du $\lambda/4$ est remplacée par un plan de sol conducteur naturel (terre, mer…) ou artificiel (toit de voiture, toiture…).

- Son gain est équivalent à celui du dipôle soit 2,15 dB_i.

LORA ET ANTENNE

- Antenne quart d'onde ou *ground plane* :





Vue dans le plan vertical

Vue du dessus

Source : Radioscope.fr F4HTZ

LORA ET ANTENNE

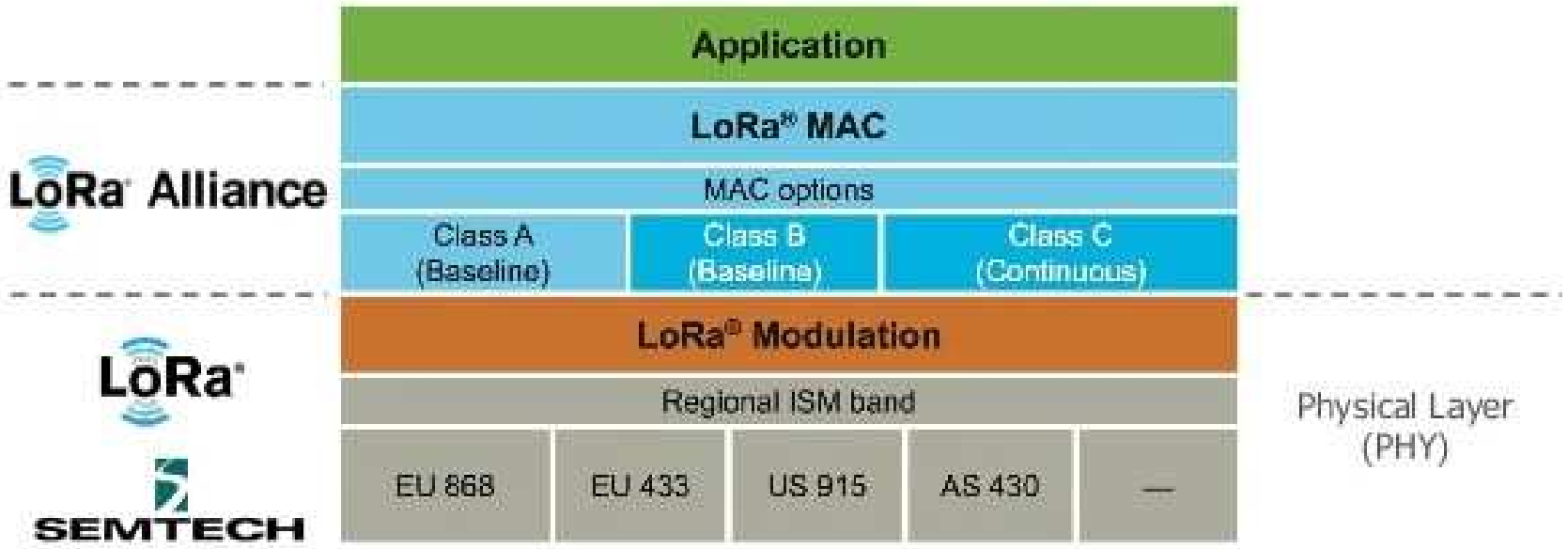
- On rappelle :
  - $\lambda = c/f$ où f est la fréquence de la porteuse.
  - Gain dB en puissance = 10 x Log (P1 / P2).
  - Gain en dBm = 10 x Log P avec P en mW.
  - 1 mW correspond à 0 dBm sur 50 Ohms soit 223,6 mV.

LORAWAN

- Créé par le consortium « LoRa Alliance », LoRaWAN (*Long Range Radio Wide Area Network*) est un réseau dédié à l'Internet des objets (IoT).

- LoRaWAN est un réseau de type LPWAN (*Low Power Wide Area Network*) basé sur LoRa.

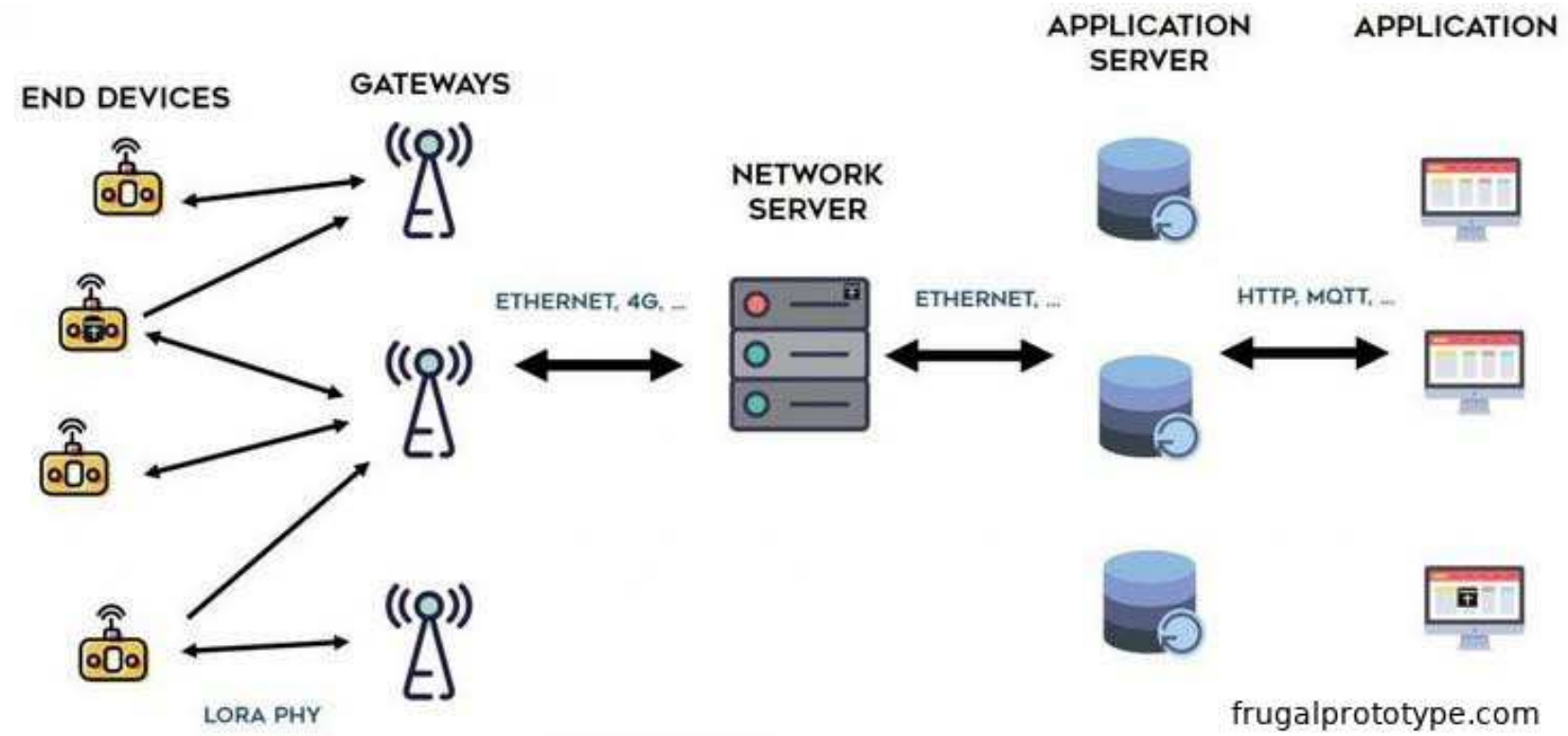
LORAWAN


Application
LoRa® MAC
MAC options
Class A (Baseline)
Class B (Baseline)
Class C (Continuous)
LoRa® Modulation
Regional ISM band
EU 868
EU 433
US 915
AS 430
—
LoRa Alliance
LoRa
SEMTECH
Physical Layer (PHY)

LORAWAN

- La topologie est en étoile d'étoiles.
- Les équipements terminaux généralement des objets connectés (*end device*) communiquent avec des passerelles (*gateway*).
- Les passerelles centralisent les données reçues avant de les retransmettre sur Internet vers un serveur réseau (*network server*) et qui seront exploitées par des applications.

LORAWAN


END DEVICES
GATEWAYS
NETWORK SERVER
APPLICATION SERVER
APPLICATION
ETHERNET, 4G, ...
ETHERNET, ...
HTTP, MQTT, ...
LORA PHY
frugalprototype.com

LORAWAN

- LoRaWAN a une méthode de contrôle d'accès de type Aloha pour l'envoi des messages.

- LoRaWAN définit 3 classes d'équipements terminaux :
  - Classe A (*All*).
  - Classe B (*Beacon*).
  - Classe C (*Continuous*).

LORAWAN

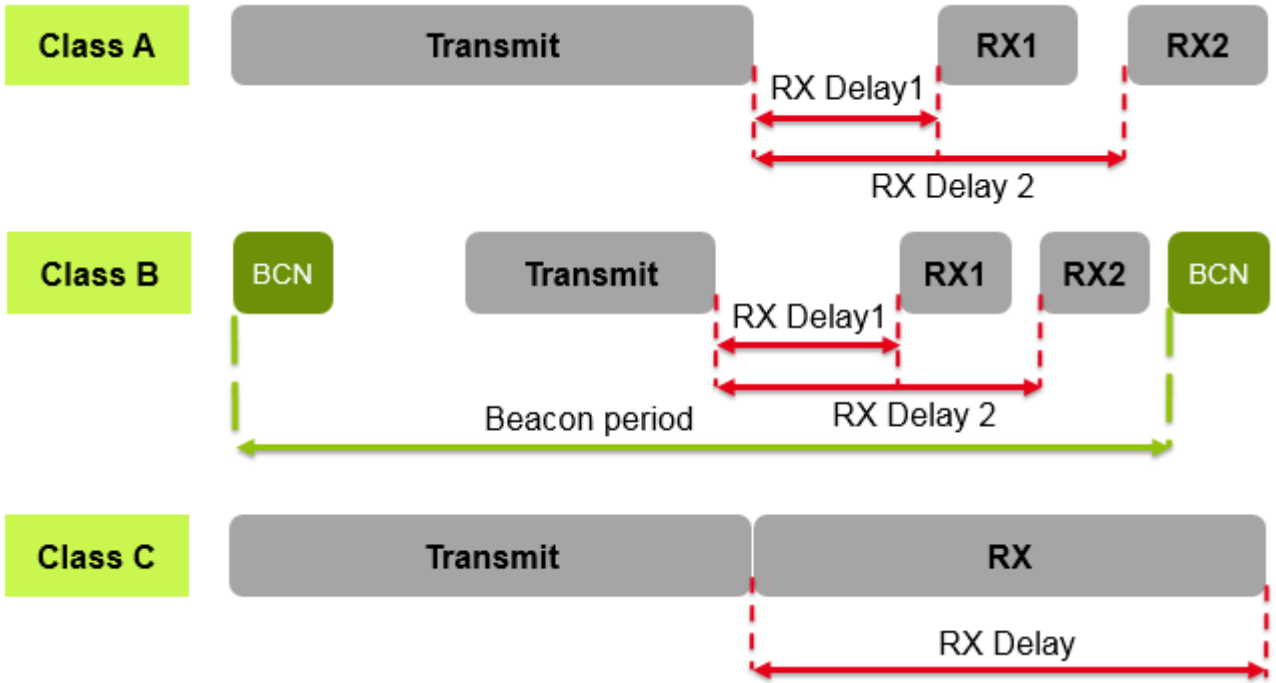
- Classe A :
  - Lorsque l'équipement terminal a des données à envoyer, il le fait puis il ouvre 2 fenêtres d'écoute successives pour des éventuels messages provenant de la passerelle. Ces 2 fenêtres sont les seules durant lesquelles la passerelle peut envoyer des données à l'équipement terminal.
  - Cette classe a la consommation énergétique la plus faible.

LORAWAN

- Classe B :
  - L'équipement terminal ouvre en plus une fenêtre de réception à des intervalles réguliers pour des messages envoyés par la passerelle (*beacon*).

- Classe C :
  - L'équipement terminal a une fenêtre d'écoute permanente.
  - Cette classe a la consommation énergétique la plus forte mais permet des communications bidirectionnelles en continu.

LORAWAN


Class A
Transmit
RX1
RX2
RX Delay1
RX Delay 2
Class B
BCN
Transmit
RX1
RX2
BCN
RX Delay1
Beacon period
RX Delay 2
Class C
Transmit
RX
RX Delay

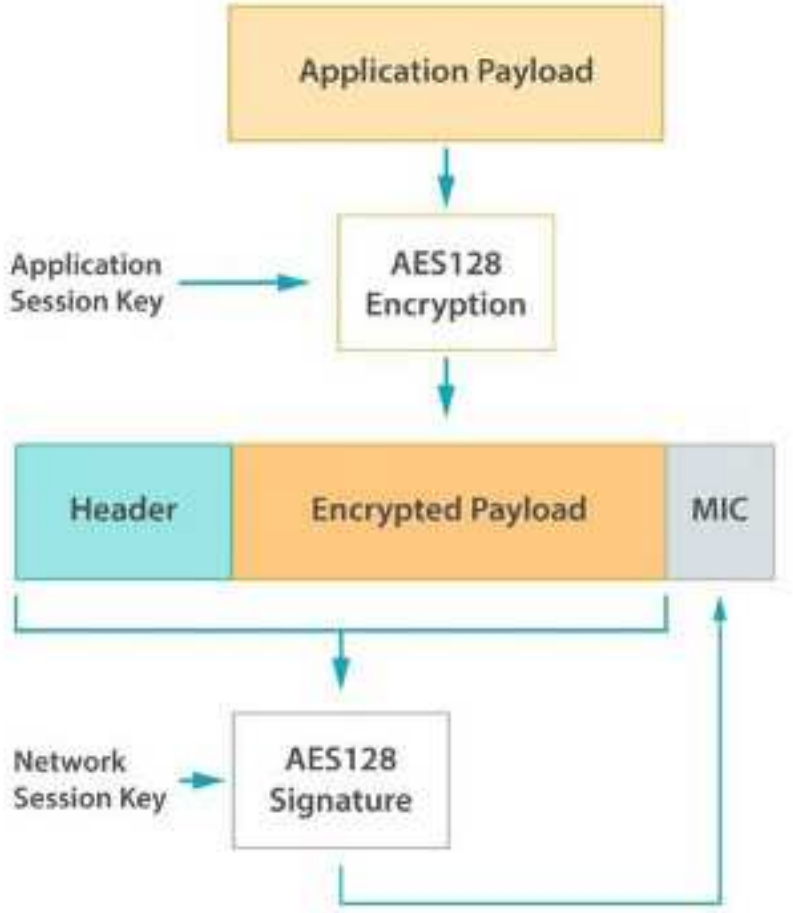
LORAWAN

- Le protocole LoRaWAN utilise plusieurs identificateurs pour les équipements du réseau :
  - DevEUI : identificateur unique de l'équipement terminal (*end device*). Format EUI-64.
  - AppEUI : identificateur unique de l'application. Format EUI-64.
  - GatewayEUI : identificateur unique de la passerelle. Format EUI-64.
  - DevAddr : adresse de l'équipement terminal. 32 bits et pas forcément unique.

LORAWAN

- Concernant la confidentialité des communications, LoRaWAN définit 3 clefs de chiffrement AES-128 (128 bits) :
  - NwkSKey : clef de session réseau qui est utilisée lors des échanges entre l'équipement terminal et le réseau. Elle assure l'authenticité des équipements terminaux en calculant et en vérifiant un MIC (*Message Integrity Code*) calculé à partir de l'entête et du *payload* chiffré du message.
  - AppSKey : clef de session applicative propre à un équipement terminal qui est utilisée pour chiffrer et déchiffrer le *payload*.
  - AppKey : clef applicative connue seulement par l'application et par l'équipement terminal et qui permet de déduire les deux clefs précédentes.

LORAWAN



source: Libelium waspmote LoRaWAN networking guide

LORAWAN

- Pour faire partie d'un réseau LoRaWAN, chaque équipement terminal doit avoir ses deux clefs de session NwkSKey et AppSKey. C'est l'étape d'activation.

- Il existe 2 types d'activation :
  - Activation dynamique OTAA (*Over-The-Air Activation*).
  - Activation statique ABP (*Activation By Personalization*).

Méthode OTAA :

- L'équipement terminal transmet au réseau une demande d'accès (*Join Request*).
- Il envoie la requête qui contient les valeurs de DevEUI, AppEUI ainsi qu'un MIC calculé avec la clé de chiffrement AppKey.
- Cette requête est transmise au serveur d'enregistrement qui vérifie le MIC via la clé AppKey qu'il connaît aussi.

Méthode OTAA :

- Si tout est conforme alors la requête d'acceptation (*Join Accept*) est envoyée à l'équipement terminal.

- A partir de la réponse, l'équipement terminal va calculer les clés de session NwkSKey et AppSKey mais aussi connaître son adresse DevAddr.

Méthode ABP :

- Les clés de session NwkSKey et AppSKey ainsi que l'adresse DevAddr sont enregistrés au préalable de façon statique dans l'équipement terminal qui peut alors directement communiquer et n'a plus besoin d'une demande d'accès préalable.

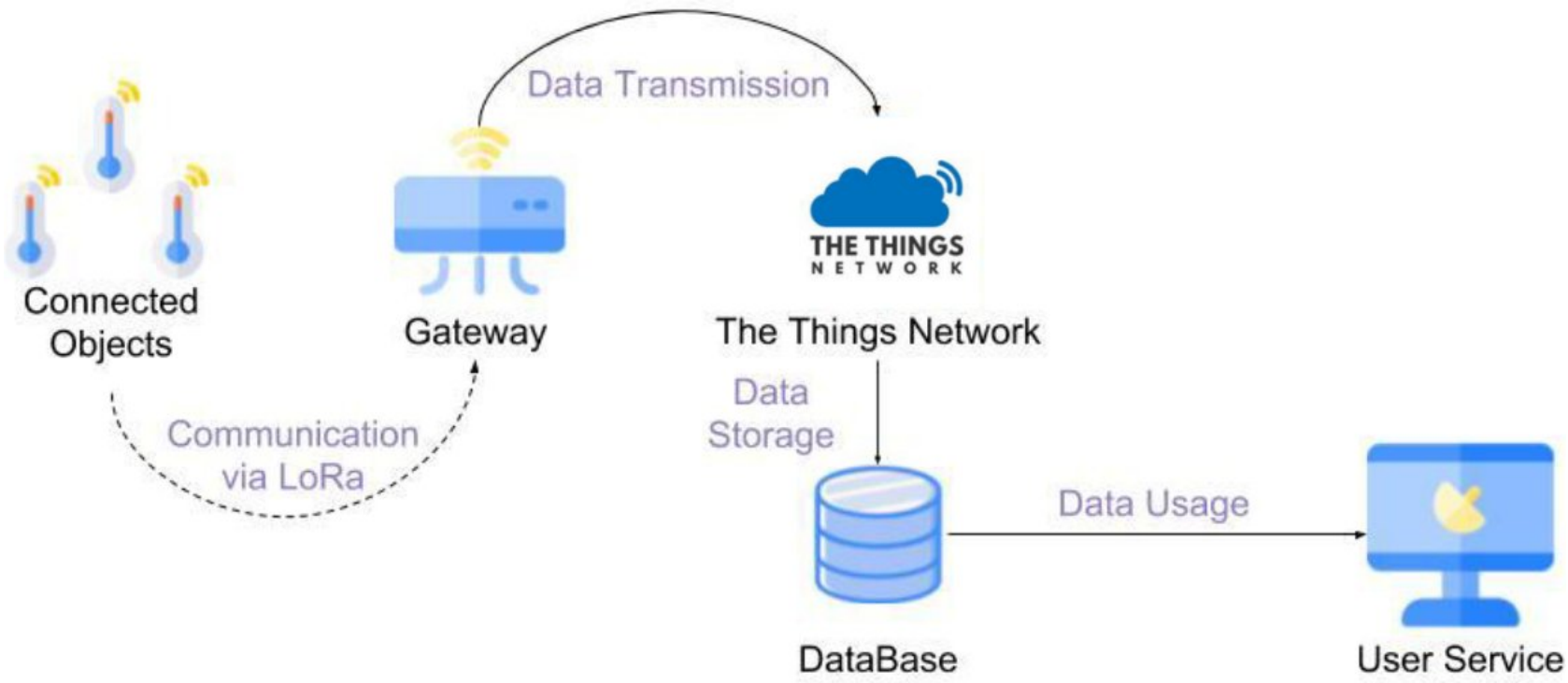
- C'est plus simple à mettre en œuvre mais cela reste moins solide pour la sécurité globale du réseau LoRaWAN.

TTN

- TTN (*The Things Network*) est « *un réseau LoRaWAN communautaire et open source pour l'Internet des Objets. Actuellement TTN se compose de plus de 40000 contributeurs regroupés en plus de 400 communautés dans 90 pays ayant déployé plus de 4000 passerelles. Il est possible pour les particuliers, universités, entreprises, ou encore les communes de contribuer au déploiement ou d'utiliser gratuitement The Things Network* ».

- Le réseau communautaire TTN est donc une infrastructure LoRaWAN pour la partie serveur réseau et serveur d'applications.
- www.thethingsnetwork.org

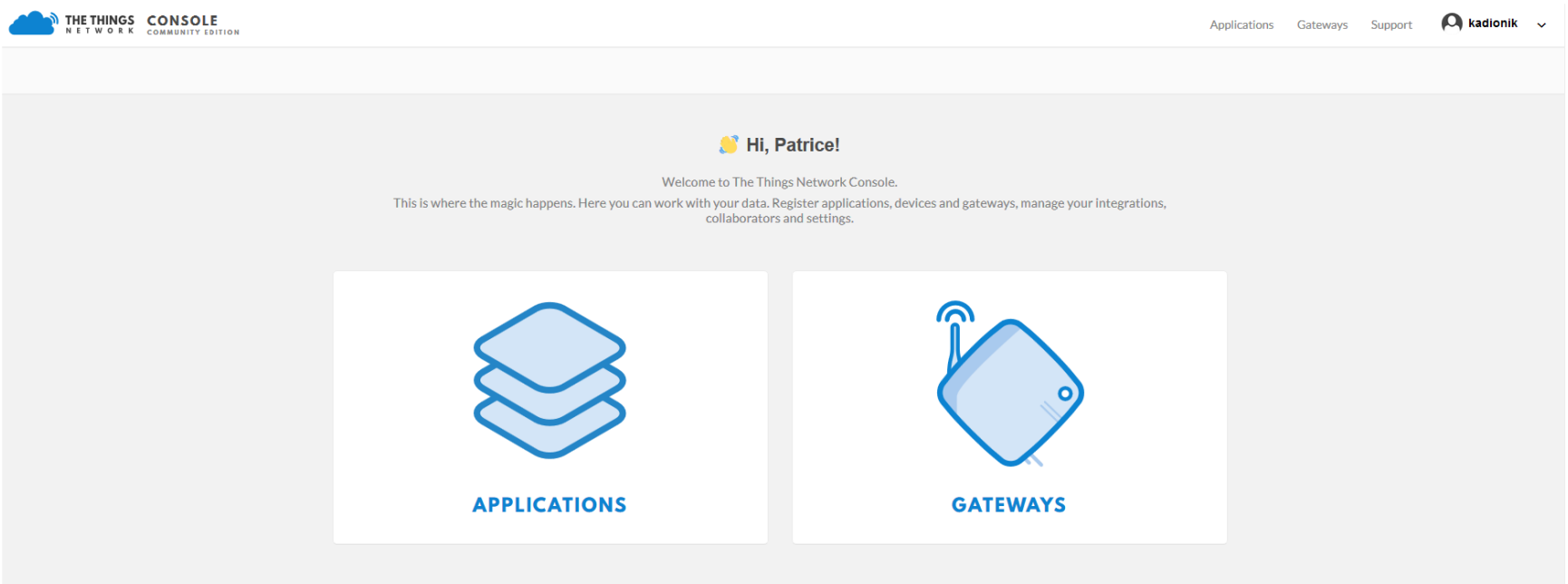
TTN


Data Transmission
Connected Objects
Gateway
THE THINGS NETWORK
The Things Network
Data Storage
Communication via LoRa
Data Usage
DataBase
User Service

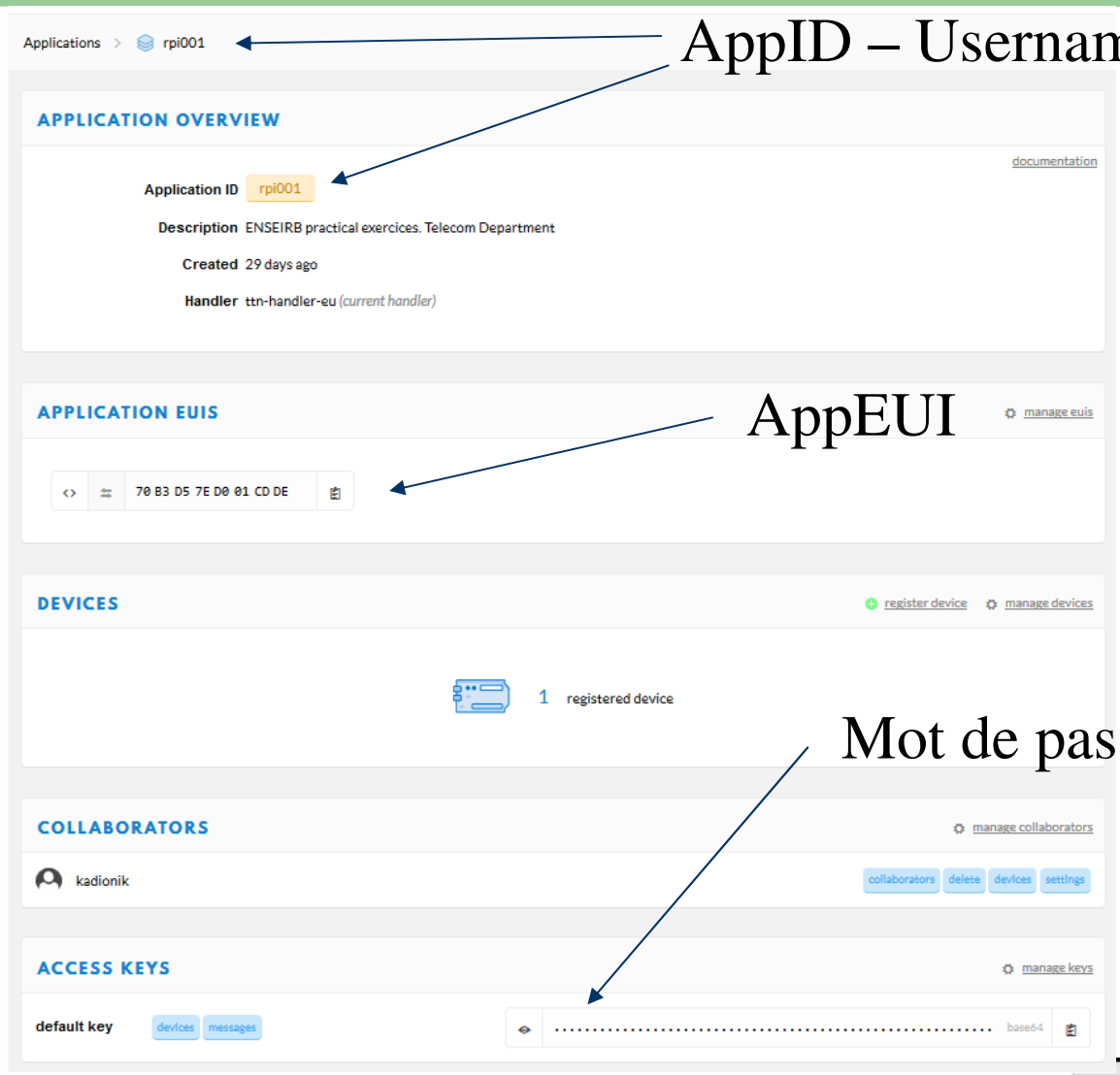
TTN

- La console de gestion d'un compte TTN (console.thethingsnetwork.org) permet de déclarer :
  - Des équipements terminaux LoRaWAN.
  - Des passerelles LoRaWAN.
  - Des applications LoRaWAN.

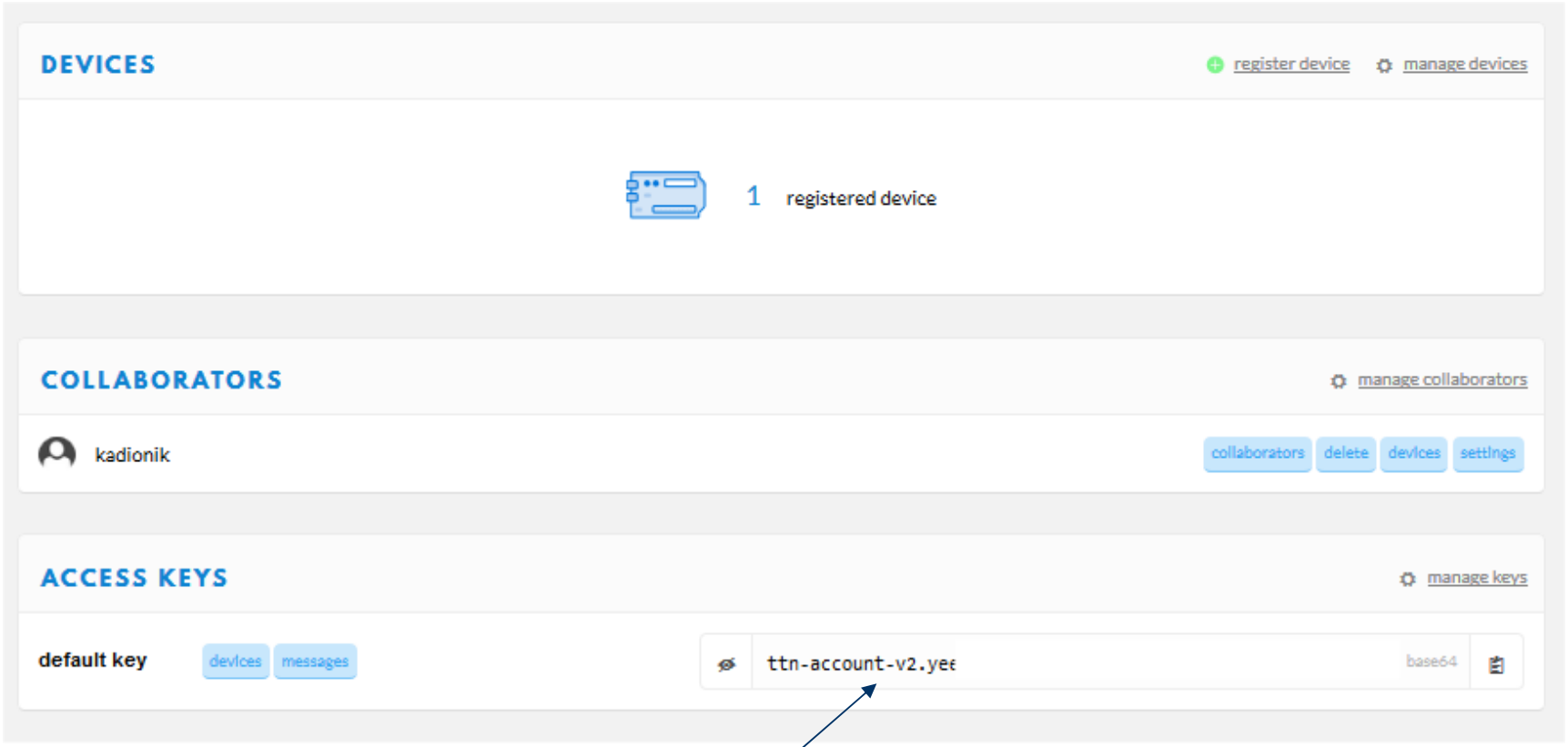
TTN


THE THINGS NETWORK CONSOLE COMMUNITY EDITION
Applications
Gateways
Support
kadionik
Hi, Patrice!
Welcome to The Things Network Console.
This is where the magic happens. Here you can work with your data. Register applications, devices and gateways, manage your integrations, collaborators and settings.
APPLICATIONS
GATEWAYS

TTN

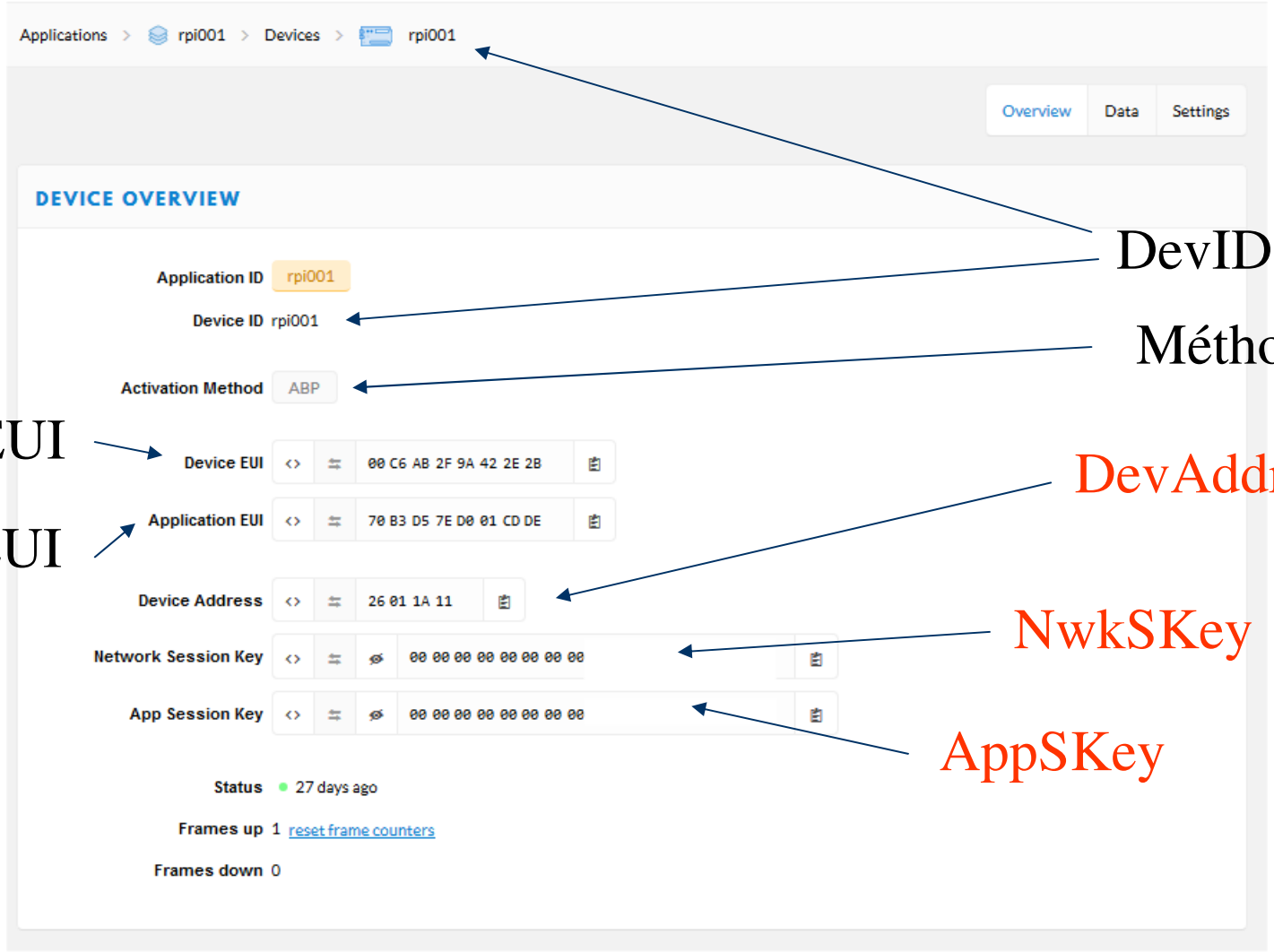

Applications
rpi001
AppID – Username MQTT
APPLICATION OVERVIEW
documentation
Application ID
rpi001
Description
ENSEIRB practical exercices. Telecom Department
Created
29 days ago
Handler
ttn-handler-eu (current handler)
APPLICATION EUIS
manage euis
AppEUI
70 B3 D5 7E D0 01 CD DE
DEVICES
register device
manage devices
1
registered device
Mot de passe MQTT
COLLABORATORS
manage collaborators
kadionik
collaborators
delete
devices
settings
ACCESS KEYS
manage keys
default key
devices
messages
base64

TTN

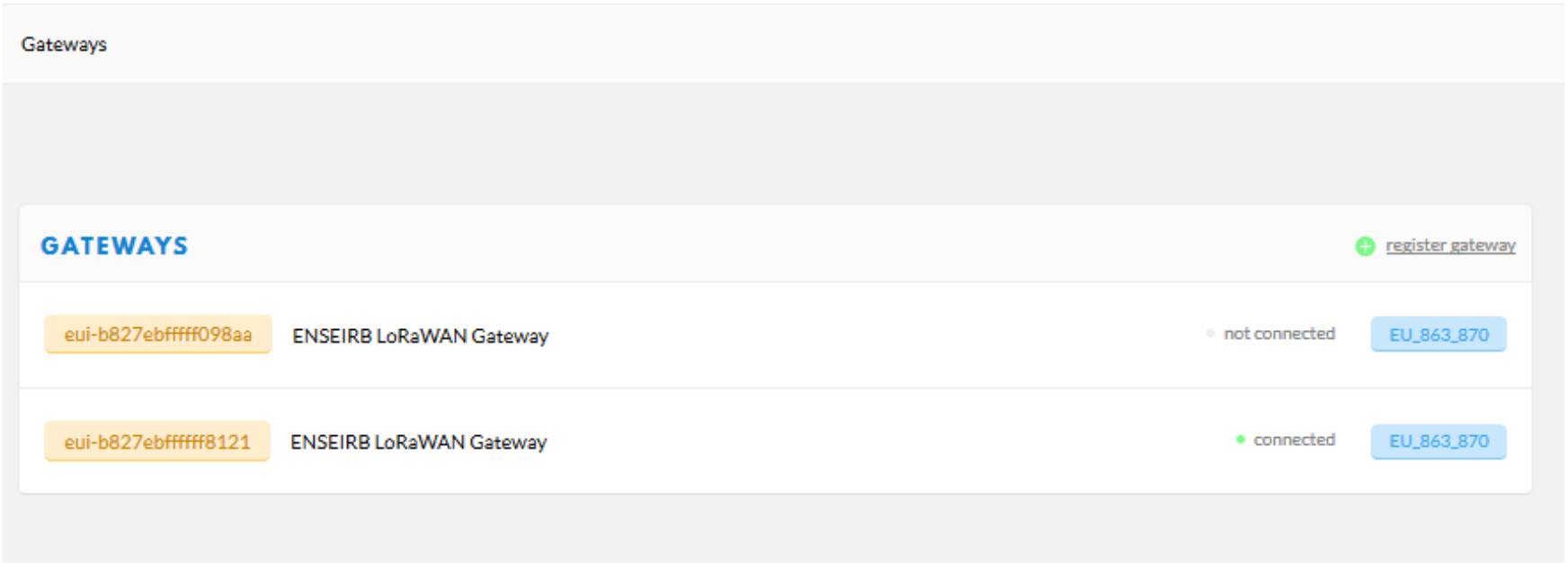


Mot de passe MQTT

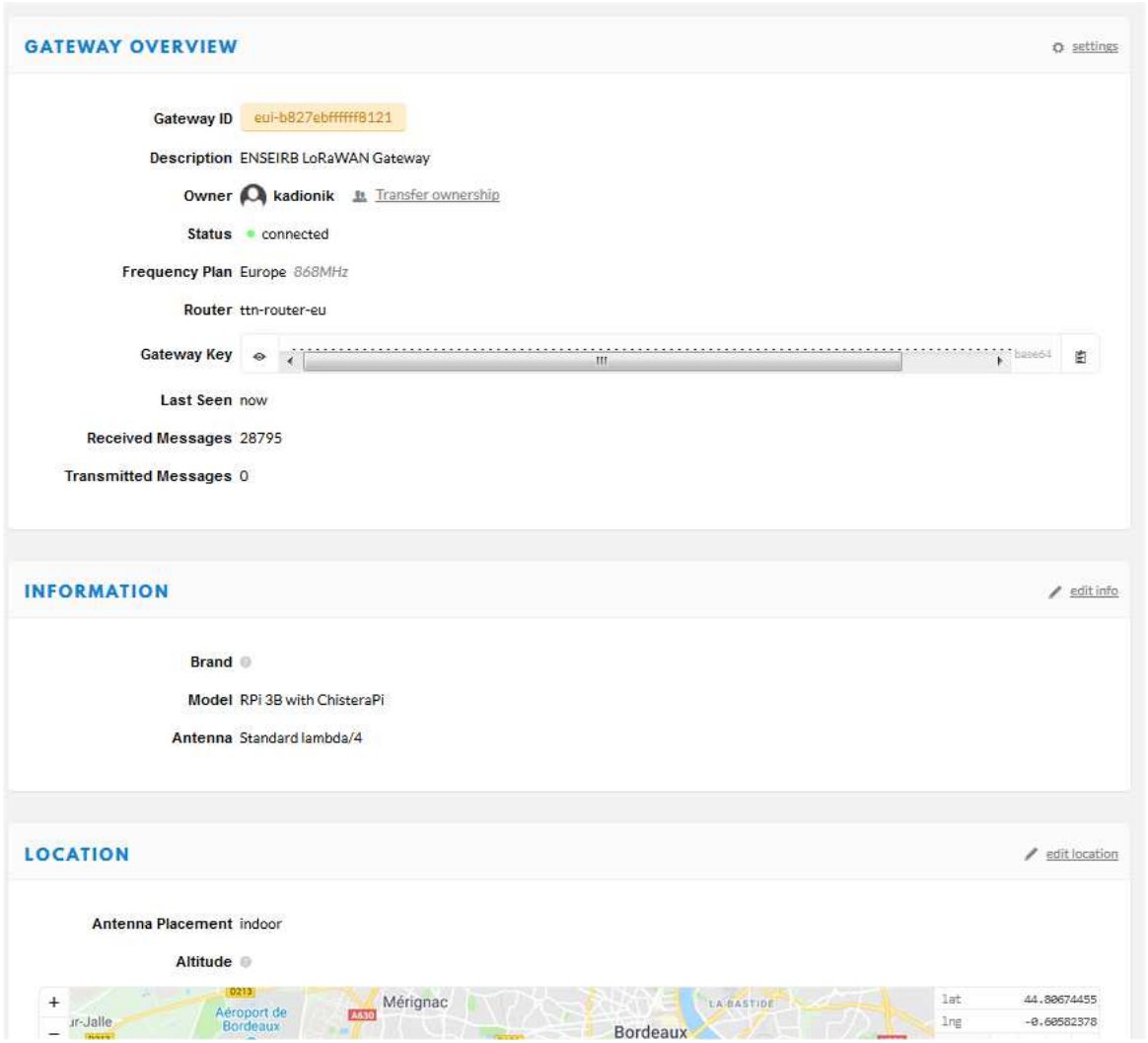
TTN


Applications > rpi001 > Devices > rpi001
Overview
Data
Settings
DEVICE OVERVIEW
Application ID rpi001
Device ID rpi001
Activation Method ABP
Device EUI 00 C6 AB 2F 9A 42 2E 2B
Application EUI 70 B3 D5 7E D0 01 CD DE
Device Address 26 01 1A 11
Network Session Key 00 00 00 00 00 00 00 00
App Session Key 00 00 00 00 00 00 00 00
Status 27 days ago
Frames up 1 reset frame counters
Frames down 0
DevID
Méthode ABP
DevEUI
AppEUI
DevAddr
NwkSKey
AppSKey

TTN


Gateways
GATEWAYS
register gateway
eui-b827ebffffff098aa
ENSEIRB LoRaWAN Gateway
not connected
EU_863_870
eui-b827ebffffff8121
ENSEIRB LoRaWAN Gateway
connected
EU_863_870

TTN


GATEWAY OVERVIEW
settings
Gateway ID
eui-b827ebffffff8121
Description
ENSEIRB LoRaWAN Gateway
Owner
kadionik
Transfer ownership
Status
connected
Frequency Plan
Europe 868MHz
Router
ttn-router-eu
Gateway Key
base64
Last Seen
now
Received Messages
28795
Transmitted Messages
0
INFORMATION
edit info
Brand
Model
RPi 3B with ChisteraPi
Antenna
Standard lambda/4
LOCATION
edit location
Antenna Placement
indoor
Altitude
Aéroport de Bordeaux
Mérignac
Bordeaux
LA BASTIDE
lat
44.80674455
lng
-0.60582378

TTN

- L'infrastructure LoRaWAN utilisée en TP est composée :
  - Les équipements terminaux : il s'agit de cartes RPi équipées pour ce TP d'un *HAT* qui possède un émetteur/récepteur LoRa (carte Draguino) dans la bande ISM 868 MHz.
  - Une passerelle LoRaWAN : il s'agit d'une carte RPi équipée d'un *HAT* possédant un émetteur/récepteur LoRa (carte ChisteraPi) exécutant un programme de passerelle vers le réseau TTN. Elle est monocanal pour LoRa et on ne pourra utiliser que la méthode d'activation ABP.
  - Un serveur réseau et aussi serveur d'applications : c'est le réseau TTN.

ENVIRONNEMENT LMIC

- La bibliothèque IBM LMIC permet d'utiliser facilement le composant LoRa du *HAT* Draguino de la carte RPi simulant l'équipement terminal LoRaWAN.

- La structure générale d'un programme C utilisant la bibliothèque LMIC est la suivante :

```
void main () {
  osjob_t initjob;
  // initialize run-time env
  os_init();
  // setup initial job
  os_setCallback(&initjob, initfunc);
  // execute scheduled jobs and events
  os_runloop();// (not reached)
}
```

ENVIRONNEMENT LMIC

- L'environnement d'exécution est initialisé par la fonction `os_init()`.
- La fonction `os_runloop()` permet de lancer l'ordonnanceur de jobs à exécuter (pas de retour de cette fonction).
- Afin de démarrer le système, un job initial doit être généré. Ce job initial est programmé par la fonction `os_setCallback()`.
- Le code du job initial défini dans la fonction `initfunc()` permet ci-après d'initialiser l'interface LoRa et de joindre ici le réseau LoRAWAN par la méthode OTAA (ou ABP).

ENVIRONNEMENT LMIC

- La fonction `initfunc()` retourne immédiatement et la fonction de *callback* `onEvent()` sera invoquée par l'ordonnanceur d'événements sur la réception des événements OTAA `EV_JOINING`, `EV_JOINED` ou `EV_JOIN_FAILED`...

```
// initial job
static void initfunc (osjob_t* j) {
  // reset MAC state
  LMIC_reset();
  // start joining
  LMIC_startJoining();
  // init done -onEvent() callback will be invoked...
}
```

ENCODAGE CAYENNE

- Pour envoyer vers TTN les données mesurées par la carte RPi, on n'utilisera pas un encodage des données de type XML ou JSON trop verbeux pour un réseau LPWAN.
- On utilisera plutôt l'encodage Cayenne qui est supporté par TTN.
- Cayenne permet d'optimiser la taille des données.
- Il est du type (Canal, Type, Valeur) et reprend la philosophie de l'encodage binaire TLV (Type, Longueur, Valeur) que l'on utilise dans les réseaux de télécommunications.

ENCODAGE CAYENNE

- Par exemple, Type vaut 0x67 pour la température et 0x73 pour la pression.

- Si la température correspond au canal de données 1 et la pression au canal de données 2, alors :
  - La température de 27,2 °C sera encodée en Cayenne (multiple de 0.1 °C signé sur 2 octets pour Valeur) comme : 0x01 0x67 0x0110.
  - La pression de 1020 hPa sera encodée en Cayenne (multiple de 0.1 hPa non signé sur 2 octets pour Valeur) comme : 0x02 0x73 0x27D8.

ENCODAGE CAYENNE

- On utilise une bibliothèque Cayenne écrite en langage C++.

- Les méthodes intéressantes de cette bibliothèque sont :
  - `CayenneLPP lpp(uint8_t size)` : taille du buffer utilisé par Cayenne en octets.
  - `lpp.reset()` : mise à zéro du buffer.
  - `lpp.getSize()` : taille consommée dans le buffer.
  - `lpp.getBuffer()` : pointeur sur le début du buffer.
  - `lpp.addTemperature(uint8_t channel, float celsius)` : ajout d'une température celcius dans le canal channel.
  - `lpp.addBarometricPressure(uint8_t channel, float hpa)` : ajout d'une pression hpa dans le canal channel.

ENCODAGE CAYENNE

- Un exemple d'usage de Cayenne est donné ci-après :

```
CayenneLPP lpp(51);
lpp.reset();

lpp.addTemperature(1, 27.2);
lpp.addBarometricPressure(2, 1020);

LMIC_setTxData2(1, (xref2u1_t)lpp.getBuffer(),
    lpp.getSize(), 0);  // Envoi données sur interface LoRa
```

OPTIONS GLRT ET RSC

Voir le cours « Internet des objets et Middleware »

CHAPITRE 9 :
CONCEPTION
D'OBJETS CONNECTES POUR L'IoT

INTRODUCTION

- L'IoT avec ses objets connectés est en plein développement.
- L'IoT met en œuvre des technologies éprouvées afin de proposer de nouveaux services.
- L'importance dans l'IoT est donc plus le service offert que les technologies mises en œuvre.

INTRODUCTION

- L'IoT comme l'embarqué est un marché de niche.
- Dans un marché de niche, il est important d'être le premier à proposer un nouveau service le plus rapidement possible même s'il n'est pas parfait car la concurrence aura toujours du mal à rattraper l'avance ainsi prise.
- Il est donc nécessaire de développer rapidement et… bien.
- Produire un prototype rapidement est donc impératif car il servira de *Proof of Concept* (*PoC*). Cela facilitera par exemple une levée de fonds…

INTRODUCTION

- Créer un prototype rapidement correspond à un prototypage rapide.
- Le prototypage rapide est donc une bonne approche pour l'IoT.
- Le prototypage rapide concerne :
  - Le matériel.
  - Les logiciels embarqués.
  - Le langage de développement.
- Il convient de balayer les technologies matérielles et logicielles intéressantes pour le prototypage rapide…

TECHNOLOGIES MATERIELLES

- Les technologies matérielles pour le prototypage rapide privilégient l'usage de matériels à bas coût.
- Il s'agit généralement de matériels libres.
- On peut citer :
  - Carte Raspberry Pi.
  - Carte BeagleBone.
  - Carte Arduino.

TECHNOLOGIES MATERIELLES

- En cas de besoin d'un circuit FPGA, on a à disposition des solutions :
  - Carte Armadeus.
  - Cartes Digilent ZedBoard, Zybo. Cartes à base de processeurs FPGA Xilinx Zynq.
  - Carte Digilent Pynq : carte à base de FPGA Zynq programmée en langage Python.

- Il est toujours possible de réaliser sa propre carte ou acheter une carte COTS…

TECHNOLOGIES LOGICIELLES

- Les technologies logicielles pour le prototypage rapide privilégient l'usage de logiciels libres.

- On retrouve les logiciels libres :
  - Pour la compilation des sources.
  - Pour la mise en œuvre d'un OS Temps Réel ou pas.
  - Pour la mise en œuvre de projets libres (pile BlueTooth…).

- On peut citer :
  - Langages : C, Java, Python.
  - OS : Linux embarqué, machine virtuelle Java.
  - OS Temps Réel : Xenomai pour Linux, FreeRTOS.

CONCEPTION D'UN OBJET CONNECTE

- L'idée est ici de concevoir un objet connecté par prototypage rapide.

- L'exemple donné ici correspond à un objet connecté à Internet bien sûr qui gère en autonome des capteurs et que l'on peut interroger par le Web (protocole HTTP). C'est une base assez générique de ce qu'est un objet connecté aujourd'hui.

CONCEPTION D'UN OBJET CONNECTE

- On utilisera dans ce cadre :
  - Du point de vue matériel : une carte Raspberry Pi (RPi) et une carte d'E/S métier (maison) connectée sur le bus d'E/S de la carte RPi.
  - Du point de vue logiciel : Linux embarqué (distribution Raspberry Pi OS) et Python pour le développement.

- L'objet connecté réalisé est appelé carte rpi-python.

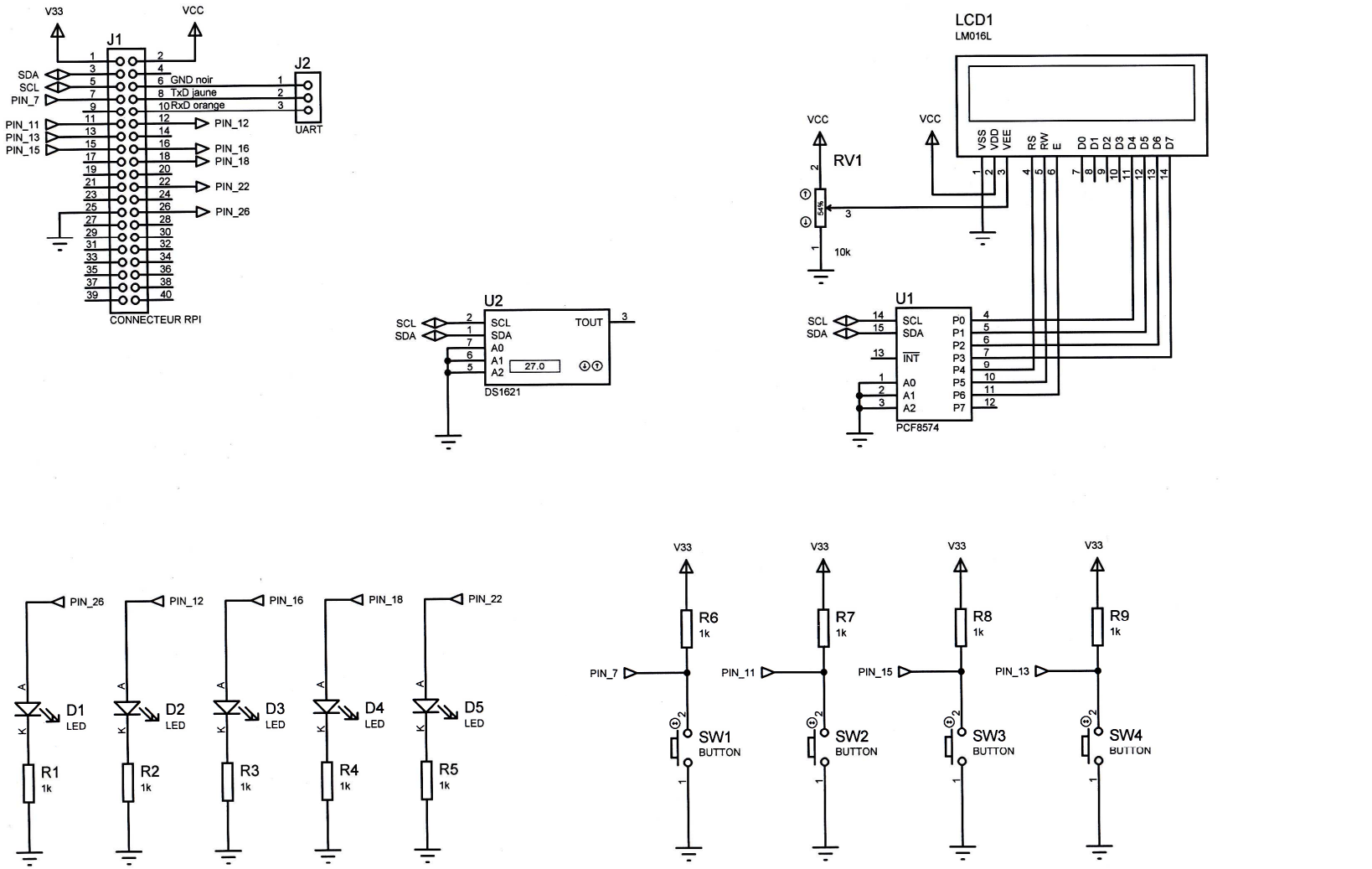
CONCEPTION D'UN OBJET CONNECTE

- La carte cible rpi-python est construite autour d'une carte Raspberry Pi 3 B.
- La carte rpi-python est donc l'association de la carte RPi et de la carte d'E/S.
- Elle est complétée d'une carte d'entrées/sorties (E/S) connectée à l'aide d'un connecteur 2x20 broches au connecteur d'E/S de la carte RPi.

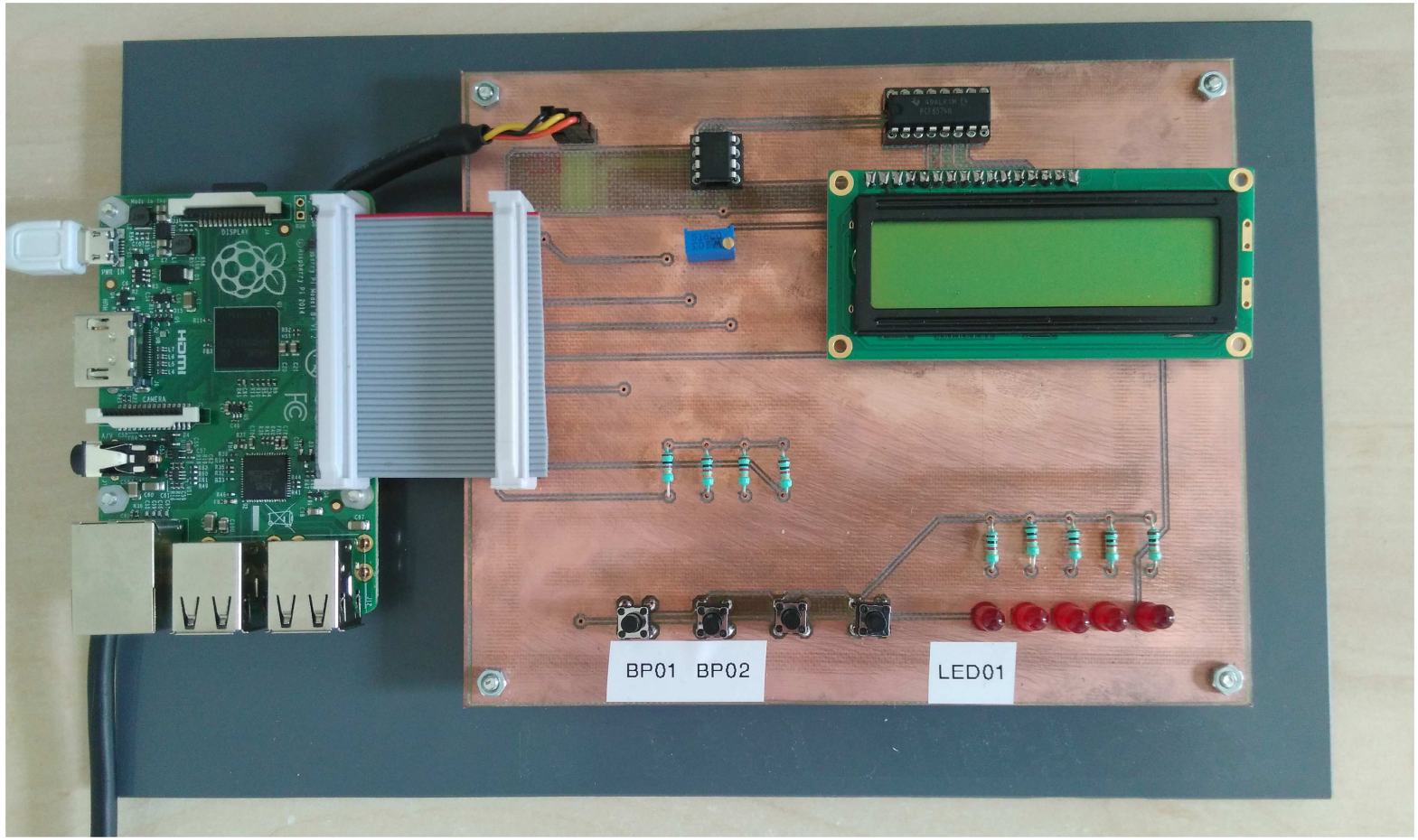
CONCEPTION D'UN OBJET CONNECTE

- La carte d'E/S rpi-python possède :
  - 5 leds : LED1 à LED5.
  - 4 boutons poussoirs : BP1 à BP4.
  - Un afficheur LCD 2x16 caractères interfacé sur le bus I2C de la carte RPi.
  - Un capteur de température numérique DS1624 interfacé sur le bus I2C de la carte RPi.

CONCEPTION D'UN OBJET CONNECTE



CONCEPTION D'UN OBJET CONNECTE



CONCEPTION D'UN OBJET CONNECTE

- On notera sur le tableau suivant la correspondance entre le numéro de broche du connecteur 2x20 broches de la carte RPi (BOARD) :

CONCEPTION D'UN OBJET CONNECTE

Numéro de broche BOARD	Numéro de broche BCM	Fonction
1		3,3 V
2		5 V
3	2	Bus I2C
5	3	Bus I2C
6		RS232
7	4	BP1
8	14	RS232
10	15	RS232
11	17	BP2
12	18	LED2
13	27	BP4
15	22	BP3
16	23	LED3
18	24	LED4
22	25	LED5
25		Masse
26	7	LED1

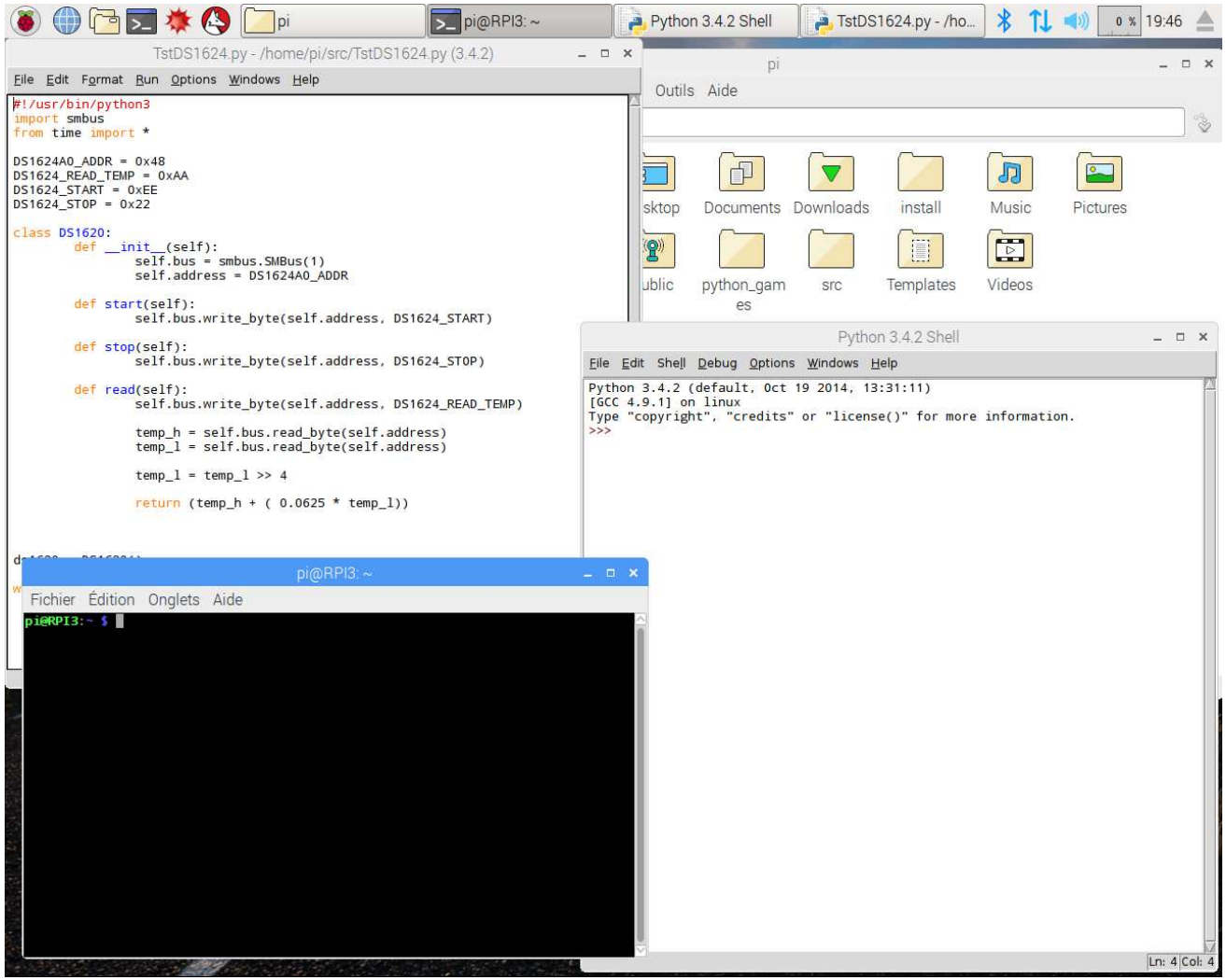
CONCEPTION D'UN OBJET CONNECTE

- La distribution Raspbian est installée sur la carte SD de la carte Raspberry Pi.

- L'environnement de développement est natif, c'est à dire que l'environnement hôte de développement et la cible se confondent. Cela facilitera le développement et le test.

- On utilisera l'environnement graphique fourni par Raspbian en branchant son écran au connecteur HDMI de la carte RPi ainsi qu'un clavier et une souris USB.

CONCEPTION D'UN OBJET CONNECTE



CONCEPTION D'UN OBJET CONNECTE

- On ajustera le fichier de configuration */boot/config.txt* pour activer le bus I2C :

```
rpi-python% cat /boot/config.txt
. . .
# Uncomment some or all of these to enable the optional
hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on
. . .
```

- Il est possible d'utiliser l'outil raspi-config pour l'activation du bus I2C :

```
rpi-python% sudo raspi-config
```

CONCEPTION D'UN OBJET CONNECTE

- On ajustera le fichier de configuration */etc/modules* pour charger les modules Linux I2C :

```
rpi-python% cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that
  should be loaded
# at boot time, one per line. Lines beginning with "#"
  are ignored.
i2c-dev
i2c_bcm2708
```

CONCEPTION D'UN OBJET CONNECTE

- On pourra vérifier que le bus I2C est bien actif après redémarrage de la carte RPi :

```
rpi-python% lsmod|grep i2c
i2c_bcm2708 4834 0
i2c_dev 5859 0
rpi-python% dmesg|grep i2c
[ 1.971246] i2c /dev entries driver
[ 1.986707] bcm2708_i2c 3f804000.i2c: BSC1 Controller at
    0x3f804000 (irq 83) (baudrate 100000
```

CONCEPTION D'UN OBJET CONNECTE

- On pourra aussi vérifier que les 2 périphériques de la carte rpi-python sont bien visibles :

```
rpi-python% i2cdetect -y 1
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

CONCEPTION D'UN OBJET CONNECTE

- On vérifiera que les paquetages Python RPi.GPIO et CharLCD sont bien installés (RPi.GPIO l'est par défaut). Si ce n'est pas le cas, on les installera par la commande :

```
rpi-python% sudo pip3 install RPi.GPIO
rpi-python% sudo pip3 install CharLCD
```

- La bibliothèque Python RPi.GPIO permet d'accéder aux E/S de la carte RPi donc aux leds et boutons poussoirs de la carte.

- La bibliothèque Python CharLCD permet d'accéder à l'afficheur LCD de la carte d'E/S.

LANGAGE PYTHON

- On programmera notre objet connecté en langage Python.
- Python est un langage de haut niveau orienté objet.
- Il est très facile d'apprentissage surtout si l'on connaît un langage de programmation comme le langage C (apprentissage en un jour).
- Python est vu en CPGE pour apprendre la programmation.
- Y a plus ka !

ACCES PYTHON AUX GPIO

- Nous utiliserons pour cela la bibliothèque Python RPi.GPIO.

- L'importation de la bibliothèque se fera par la directive :

```
import RPi.GPIO as GPIO
```

ACCES PYTHON AUX GPIO

- On a à disposition les principales fonctions suivantes :
  - `GPIO.setmode(GPIO.BOARD)` : numérotation des broches en mode BOARD.
  - `GPIO.setmode(GPIO.BCM)` : numérotation des broches en mode BCM.
  - `GPIO.setup(x, GPIO.IN)` : configuration de la broche x en entrée.
  - `GPIO.setup(x, GPIO.OUT)` : configuration de la broche x en sortie.
  - `GPIO.input(x)` : lecture de l'état courant de l'entrée x.
  - `GPIO.output(x, GPIO.LOW)` : positionnement à l'état bas (0) de la sortie x.
  - `GPIO.output(x, GPIO.HIGH)` : positionnement à l'état haut (1) de la sortie x.

ACCES PYTHON AUX GPIO

- La réalisation d'un chenillard à la « K2000 » sur la carte rpi-python se fera très simplement avec le code source Python suivant :

```
#!/usr/bin/python3
import RPi.GPIO as GPIO
from time import *

LED1 = 26
LED2 = 12
LED3 = 16
LED4 = 18
LED5 = 22
```

ACCES PYTHON AUX GPIO

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

GPIO.setup(LED1, GPIO.OUT)
GPIO.setup(LED2, GPIO.OUT)
GPIO.setup(LED3, GPIO.OUT)
GPIO.setup(LED4, GPIO.OUT)
GPIO.setup(LED5, GPIO.OUT)

while True:
   GPIO.output(LED1, GPIO.LOW)
   GPIO.output(LED2, GPIO.HIGH)
   sleep(0.1)


   GPIO.output(LED2, GPIO.LOW)
   GPIO.output(LED3, GPIO.HIGH)
   sleep(0.1)
   . . .
```

ACCES PYTHON A L'I2C

- Deux périphériques I2C sont connectés sur le bus I2C de la carte rpi-python. Nous trouvons ainsi :
  - Un afficheur LCD 2x16 caractères en mode 4 bits par l'intermédiaire d'un circuit adaptateur I2C/GPIO (identificateur I2C 0x20).
  - Un capteur de température numérique DS1624 (identificateur I2C 0x48).

- L'accès au capteur de température se fait très simplement avec la bibliothèque I2C native de Python.

- Il faudra au préalable importer la bibliothèque smbus par la directive :

```
import smbus
```

ACCES PYTHON A L'I2C

- On a alors à disposition les principales fonctions suivantes :
  - `bus = smbus.SMBus(x)` : choix du bus I2C numéro `x`.
  - `bus.write_byte(address, command)` : écriture I2C de la commande command pour le périphérique de numéro I2C `address`.
  - `bus.read_byte(address)` : lecture I2C de la donnée 8 bits du périphérique de numéro I2C `address`.

ACCES PYTHON A L'I2C

- Un exemple d'usage du capteur de température est donné par le code source Python suivant :

```
#!/usr/bin/python3
import smbus
from time import *

DS1624A0_ADDR = 0x48
DS1624_READ_TEMP = 0xAA
DS1624_START = 0xEE
DS1624_STOP = 0x22

class DS1624:
   def __init__(self):
     self.bus = smbus.SMBus(1)
     self.address = DS1624A0_ADDR
```

ACCES PYTHON A L'I2C

```
    def start(self):
      self.bus.write_byte(self.address, DS1624_START)
    def stop(self):
      self.bus.write_byte(self.address, DS1624_STOP)
    def read_temp(self):
      self.bus.write_byte(self.address, DS1624_READ_TEMP)
      temp_h = self.bus.read_byte(self.address)
      temp_l = self.bus.read_byte(self.address)
      temp_l = temp_l >> 4
      return (temp_h + (0.0625 * temp_l))

ds = DS1624()
while True :
    ds.start()
    sleep(1)
    temperature = ds.read_temp()
    print ("T1=%02.1f" % temperature)
```

ACCES PYTHON A L'I2C

- Le pilotage de l'afficheur LCD se fait à l'aide de la bibliothèque Python CharLCD.

- Il faudra au préalable importer les bibliothèques RPi.GPIO et charlcd par les directives :

```
import RPi.GPIO as GPIO
from charlcd import direct as lcd
from charlcd.drivers.gpio import Gpio
from charlcd.drivers.i2c import I2C
```

ACCES PYTHON A L'I2C

- La bibliothèque Python CharLCD permet de piloter un afficheur LCD en mode 8 bits ou 4 bits, un afficheur I2C ou bien via un circuit adaptateur I2C/GPIO PCF8574 comme utilisé par la carte rpi-python.

- Elle fournit pour notre cas les objets Python I2C et CharLCD qu'il suffit d'instancier avec les bons paramètres.

ACCES PYTHON A L'I2C

- Pour la carte rpi-python, nous avons un afficheur 2x16 caractères à l'adresse I2C 0x20 sur le bus I2C numéro 1. Cela donne l'initialisation suivante :

```
i2c = I2C(0x20, 1)
i2c.pins = {
'RS': 4,
'E': 6,
'E2': None,
'DB4': 0,
'DB5': 1,
'DB6': 2,
'DB7': 3
}
lcd = lcd.CharLCD(16, 2, i2c)
```

ACCES PYTHON A L'I2C

- Nous avons alors à disposition les principales méthodes suivantes :
  - `init()` : initialisation de l'affichage.
  - `write(string)` : écriture d'une chaîne de caractères `string` à la position courante.
  - `write(string, li, co)` : écriture d'une chaîne de caractères string à la ligne `li` et colonne `co`.
  - `lcd.set_xy(li, co)` : positionnement à la ligne `li` et colonne `co`.

ACCES PYTHON A L'I2C

- Un exemple d'usage de l'afficheur LCD est donné par le code source Python suivant :

```
#!/usr/bin/python3
import RPi.GPIO as GPIO
from charlcd import direct as lcd
from charlcd.drivers.gpio import Gpio
from charlcd.drivers.i2c import I2C

GPIO.setmode(GPIO.BCM)
```

ACCES PYTHON A L'I2C

```
i2c = I2C(0x20, 1)
i2c.pins = {
'RS': 4,
'E': 6,
'E2': None,
'DB4': 0,
'DB5': 1,
'DB6': 2,
'DB7': 3
}

lcd = lcd.CharLCD(16, 2, i2c)
lcd.init()

lcd.write("Hello world!")
lcd.set_xy(0, 1)
lcd.write("Raspberry Pi")
```

PROGRAMMATION DE THREADS PYTHON

- On peut implanter un *thread* sous forme d'un objet Python implémentant l'interface *threading.Thread*.

- Il faudra au préalable importer la bibliothèque *threading* par la directive :

```
import threading
```

PROGRAMMATION DE THREADS PYTHON

- Un exemple de création de *threads* est donné par le code source Python suivant :

```
import threading
import time
class Affiche(threading.Thread):
   def __init__(self, nom = ''):
     threading.Thread.__init__(self)
     self.nom = nom
     self.Terminated = False
   def run(self):
     while not self.Terminated:
     print (self.nom)
     time.sleep(2.0)
   def stop(self):
     self.Terminated = True
```

PROGRAMMATION DE THREADS PYTHON

```
a = Affiche("Thread A")
b = Affiche("Thread B")

a.start()
b.start()

time.sleep(10)

a.stop()
b.stop()
```

API SOCKETS PYTHON

- Si l'on veut contrôler à distance son objet connecté, au plus simple, il faut pouvoir créer un serveur TCP.

- Python fournit une bibliothèque *socket* qui permet d'implémenter des clients et des serveurs. Elle ressemble bien sûr dans sa syntaxe et son usage à l'API *socket* en langage C sous *NIX…

- Il faudra au préalable importer la bibliothèque *socket* par la directive :

```
import socket
```

API SOCKETS PYTHON

- Un exemple de miniserveur Web itératif écoutant sur le port 8080 est donné par le code source Python suivant :

```
import socket

socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
socket.bind(('', 8080))
socket.listen(1)

while True:
  client, address = socket.accept()
  request = client.recv(1024)

  client.send(str.encode("HTTP/1.1 200 OK\n"))
  client.send(str.encode("Content-Type: text/html\n"))
  client.send(str.encode("\n"))
```

API SOCKETS PYTHON

```
client.send(str.encode("<meta http-equiv=\"Refresh\"
content=\"1\">"))

client.send(str.encode("<CENTER><H1>Welcome to the rpi-
python Web Server</H1></CENTER>"))

client.send(str.encode("Hello World!"))

client.close()
```

API SOCKETS PYTHON

- Le serveur itératif renvoie à chaque requête HTTP la même réponse HTTP suivante :

```
HTTP/1.1 200 OK
Content-Type: text/html

<meta http-equiv="Refresh" content=1>
<CENTER><H1>Welcome to the rpi-python
  WebServer</H1></CENTER>
Hello World!
```

PROGRAMMATION DE L'OBJET CONNECTE

- A des fins d'illustration, notre objet connecté représenté par la carte rpi-python possède les fonctionnalités suivantes :
  - Il est interrogeable à distance et intègre un serveur Web qui fournit la température courante et l'heure courante.
  - Il affiche localement sur l'afficheur LCD de la carte rpi-python la température courante et l'heure courante.
  - Il réalise un chenillard sur les leds de la carte.

PROGRAMMATION DE L'OBJET CONNECTE

- On créera 3 *threads* Python :
  - *Thread* task_led : gestion des leds de la carte rpi-python réaliser un chenillard à la K2000.
  - *Thread* task_lcd : gestion de l'afficheur LCD avec affichage de l'heure et de la température courantes de la carte rpi-python.
  - *Thread* task_www : implémentation d'un serveur Web sur le port d'écoute 8080 qui renvoie l'heure courante et la température courante. La page Web est rafraîchie toutes les secondes.

- En forgeant la requête Web @IP_carte_rpi-python:8080, on accède avec un navigateur Web à la page d'accueil « *Welcome to the rpi-python Web Server* » qui fournit l'heure courante et la température courante de notre objet connecté.

PROGRAMMATION DE L'OBJET CONNECTE

```
#!/usr/bin/python3
import smbus
import threading
import socket


from time import *


import RPi.GPIO as GPIO


from charlcd import direct as lcd
from charlcd.drivers.gpio import Gpio
from charlcd.drivers.i2c import I2C


DS1624A0_ADDR = 0x48
DS1624_READ_TEMP = 0xAA
DS1624_START = 0xEE
DS1624_STOP = 0x22
```

PROGRAMMATION DE L'OBJET CONNECTE

```
# Notation BCM
LED1 = 7
LED2 = 18
LED3 = 23
LED4 = 24
LED5 = 25


temp_www = 0


class DS1624:
    def __init__(self):
      self.bus = smbus.SMBus(1)
      self.address = DS1624A0_ADDR


    def start(self):
      self.bus.write_byte(self.address, DS1624_START)
    def stop(self):
      self.bus.write_byte(self.address, DS1624_STOP)
```

PROGRAMMATION DE L'OBJET CONNECTE

```
    def read_temp(self):
      self.bus.write_byte(self.address, DS1624_READ_TEMP)
      temp_h = self.bus.read_byte(self.address)
      temp_l = self.bus.read_byte(self.address)
      temp_l = temp_l >> 4
      return (temp_h + (0.0625 * temp_l))

class task_led(threading.Thread):
    def __init__(self, nom = ''):
      threading.Thread.__init__(self)
      self.nom = nom
      self.Terminated = False
      GPIO.setup(LED1, GPIO.OUT)
      GPIO.setup(LED2, GPIO.OUT)
      GPIO.setup(LED3, GPIO.OUT)
      GPIO.setup(LED4, GPIO.OUT)
      GPIO.setup(LED5, GPIO.OUT)

    def run(self):
      while not self.Terminated:
                GPIO.output(LED1, GPIO.LOW)
                GPIO.output(LED2, GPIO.HIGH)
                sleep(0.1)
```

PROGRAMMATION DE L'OBJET CONNECTE

```
    def stop(self):
      self.Terminated = True

class task_lcd(threading.Thread):
    def __init__(self, nom = ''):
      threading.Thread.__init__(self)
      self.nom = nom
      self.Terminated = False

    def run(self):
      global temp_www
      while not self.Terminated:
                temp = 100.0

                ds = DS1624()

                lcd.init()

                while (True):
                          ds.start()
                          ds.stop()
                          temp = ds.read_temp()
                          temp_www = temp
```

PROGRAMMATION DE L'OBJET CONNECTE

```
                        lcd.set_xy(0, 0)
                        t = strftime('%H:%M:%S')
                        lcd.write(t)

                        lcd.set_xy(0, 1)
                        lcd.write("TEMP=%02.1f oC" % temp)
                        sleep(0.01)


    def stop(self):
      self.Terminated = True

class task_www(threading.Thread):
    def __init__(self, nom = ''):
      threading.Thread.__init__(self)
      self.nom = nom
      self.Terminated = False

      self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      self.socket.bind(('', 8080))
      self.socket.listen(1)
```

PROGRAMMATION DE L'OBJET CONNECTE

```
def run(self):
  global temp_www
  while not self.Terminated:
            while (True):
                      client, address = self.socket.accept()

                      request = client.recv(1024)
                      client.send(str.encode("HTTP/1.1 200 OK\n"))
                      client.send(str.encode("Content-Type: text/html\n"))
                      client.send(str.encode("\n"))

                      client.send(str.encode("<meta http-equiv=\"Refresh\"
content=\"1\">"))
                      client.send(str.encode("<CENTER><H1>Welcome to the rpi-
python Web Server</H1></CENTER>"))

                      t = strftime('%H:%M:%S')
                      client.send(str.encode(t))

                      client.send(str.encode("<BR>"))

                      client.send(str.encode("TEMP=%02.1f °C" % temp_www))
                      client.close()
```

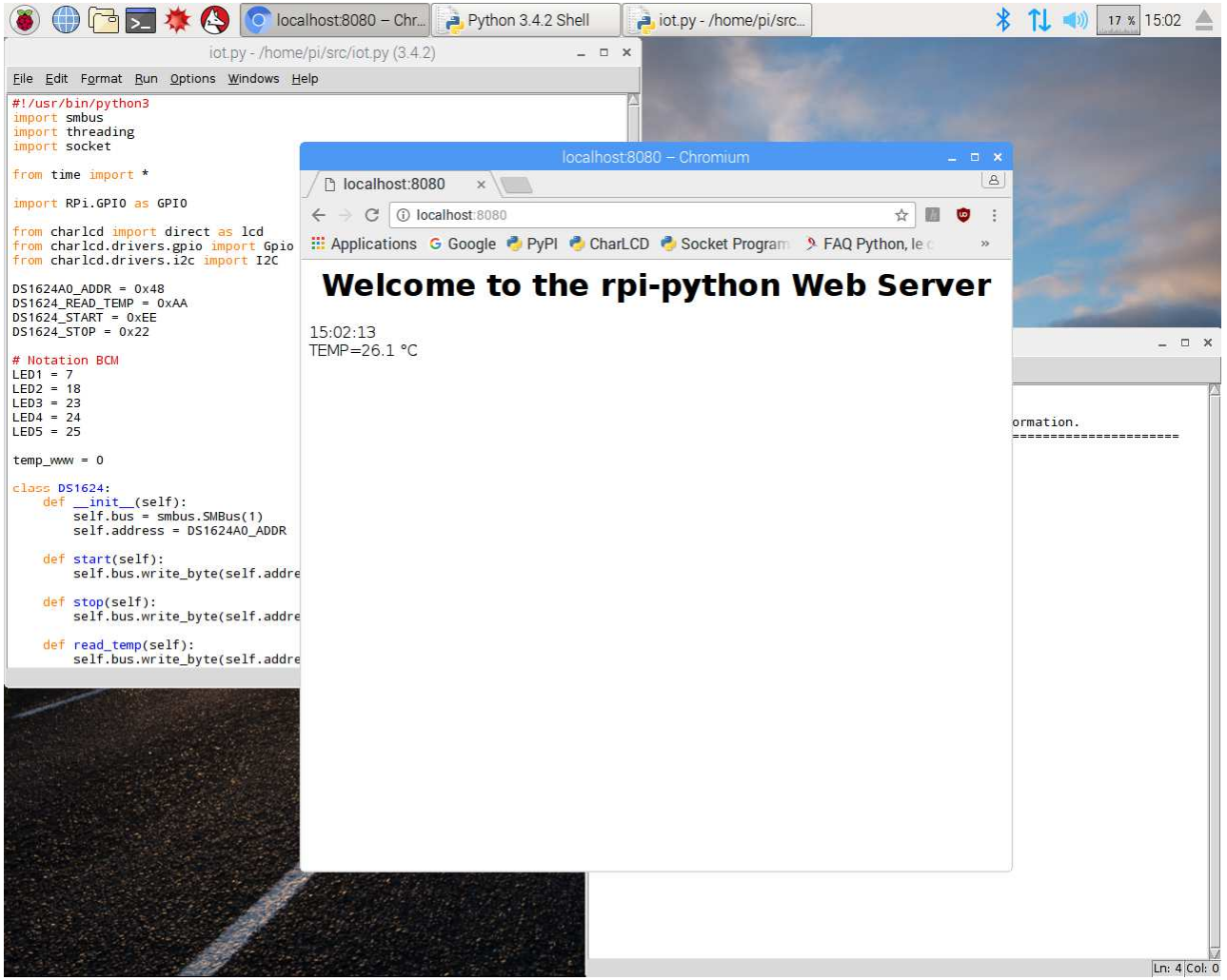
PROGRAMMATION DE L'OBJET CONNECTE

```
    def stop(self):
      self.Terminated = True

####
# Debut
####

# Initialisation GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

# Initialisation LCD
i2c = I2C(0x20, 1)
i2c.pins = {
'RS': 4,
'E': 6,
'E2': None,
'DB4': 0,
'DB5': 1,
'DB6': 2,
'DB7': 3
}
lcd = lcd.CharLCD(16, 2, i2c)
```

PROGRAMMATION DE L'OBJET CONNECTE

```
# Lancement threads
t1 = task_lcd("task_lcd")
t1.start()
t2 = task_led("task_led")
t2.start()
t3 = task_www("task_www")
t3.start()
```

PROGRAMMATION DE L'OBJET CONNECTE



INTERET DU PROTOTYPAGE RAPIDE

- Nous avons pu voir la conception d'un objet connecté. Le prototypage rapide est utilisé et a pris :
  - 1 j pour réaliser la carte.
  - 1 j pour configurer et programmer la carte en langage Python.

- La réalisation a été ainsi accélérée grâce :
  - Au choix matériel : carte RPi.
  - Aux choix logiciels : Linux embarqué.
  - Langage : Python.

- Le prototypage rapide est une nécessité dans l'IoT : être le premier sur un marché de niche.

INTERET DU PROTOTYPAGE RAPIDE

- Nous avons vu un exemple de prototypage rapide et cela peut être appliqué plus généralement dans l'IoT :
  - Matériels : RPi, Arduino, BeagleBone, Zybo ou carte COTS.
  - Logiciels : Linux embarqué, Xenomai, FreeRTOS.
  - Langages : C, Java, Python.

- Les choix se feront en fonction de l'objet connecté à réaliser…

OPTION SE

Voir le cours « Internet des objets et Middleware »

CONCLUSION

CONCLUSION

- Nous avons pu voir ce qu'est un système embarqué et ses principales caractéristiques.
- Les systèmes embarqués complexes mettent en oeuvre aujourd'hui le *codesign*.
- Il existe généralement des contraintes temporelles à respecter et il faut alors utiliser des systèmes Temps Réel dur ou Temps Réel mou suivant la QoS recherchée pour la gestion du temps.
- L'IoT est un domaine important de l'embarqué. Nous avons pu dresser une liste des technologies utilisées.

CONCLUSION

- Dans le monde de l'embarqué, le logiciel libre et le matériel libre sont de plus en plus utilisés. Le noyau Linux pour l'embarqué ou plus simplement Linux embarqué est un allié de poids.

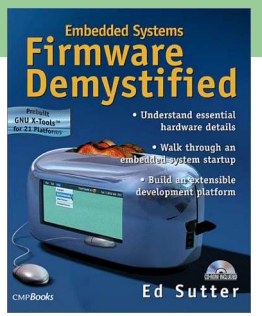
- Il existe des solutions Temps Réel dur sous Linux (Xenomai) et des solutions Temps Réel mou sous Linux (PREEMPT-RT).

- L'IoT met en œuvre des protocoles d'échanges de données spécifiques (MQTT, HTTP…) ainsi que des réseaux LPWAN performants (LoRaWAN). Nous avons pu voir aussi l'importance du prototypage rapide dans la conception des objets connectés pour l'IoT.

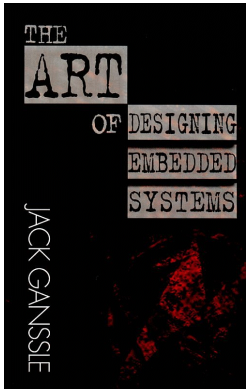
BIBLIOGRAPHIE

REFERENCES BIBLIOGRAPHIQUES

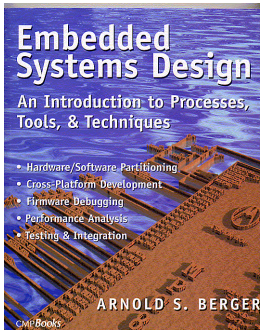
- Embedded Systems. Firmware Demystified. E. Sutter. Editions CMP Books



- The Art of Designing embedded Systems. J. Ganssle. Editions Butterworth-Heinemann



- Embedded Systems Design. A S. Berger. Editions CMP Books



REFERENCES BIBLIOGRAPHIQUES

- Linux embarqué. P. Ficheux. Editions Eyrolles.

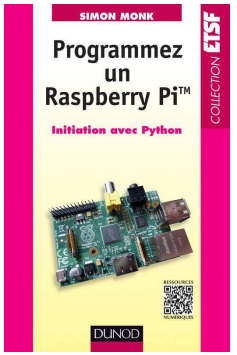
  LA REFERENCE !

- Solutions Temps Réel sous Linux. C. Blaess. Editions Eyrolles.

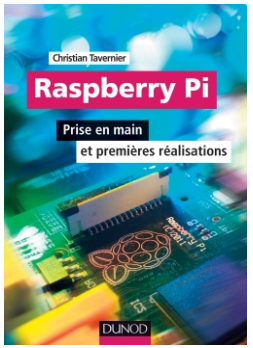
  LA REFERENCE !

REFERENCES BIBLIOGRAPHIQUES

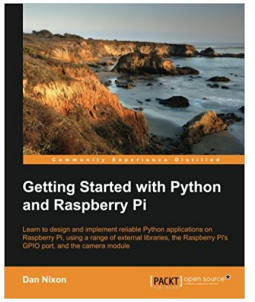
- Programmez un Raspberry Pi. Initiation avec Python. S. Monk. Editions Dunod



- Raspberry Pi. Prise en main. C. Tavernier. Editions Dunod

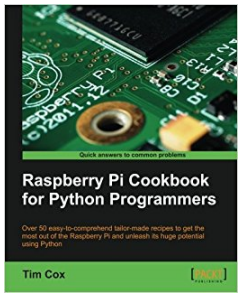


- Getting Started with Python and Raspberry Pi. D. Nixon. Editions Packt Publishing

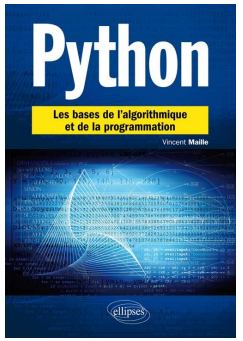


REFERENCES BIBLIOGRAPHIQUES

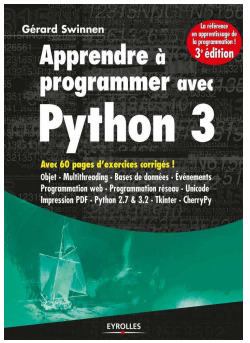
- Raspberry Pi Cookbook for Python Programmers. T. Cox. Editions Packt Publishing



- Python. Les bases de l'algorithmique et de la programmation. V. Maille. Editions Ellipses



- Apprendre à programmer avec Python 3. Gérard Swinnen. Editiond Eyrolles



REFERENCES BIBLIOGRAPHIQUES

- EETimes Embedded Markets Study 2019
- IoT Developer Survey Results. 2018
- Energy Efficiency across Programming Languages. How Do Energy, Time, and Memory Relate? R. Pereira and al. ACM SIGPLAN International Conference on Software Language Engineering. 2017
- Introduction à LoRa. Arnaud Pecoraro
- Introduction à LoRa. Ali Benfattoum
- Réseau TTN : www.thethingsnetwork.org
- Wikipedia : fr.wikipedia.org