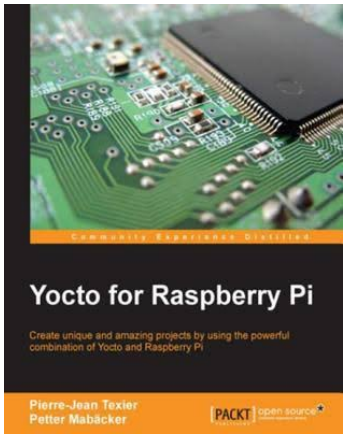
Architecture hétérogène au service de l'IoT industriel ?

Pierre-Jean Texier

$ whoami => Pierre-Jean Texier

- Ingénieur Linux Embarqué **Lafon Technologies**

- Free software enthusiast
- Co-auteur de **"Yocto for Raspberry Pi"** et auteur dans GNU/Linux magazine et Open silicium



Au service du paiement ...

4 personnes, diverses activités au sein du BE :

- Linux Device Driver (i2c, mtd, ...)
- Board Support Package (`OpenEmbedded`) =>`ARM` & `MIPS`
- Interface Graphique (Framework Qt)
- Firmware MCU => `EFM32` & `STM32`
- ...

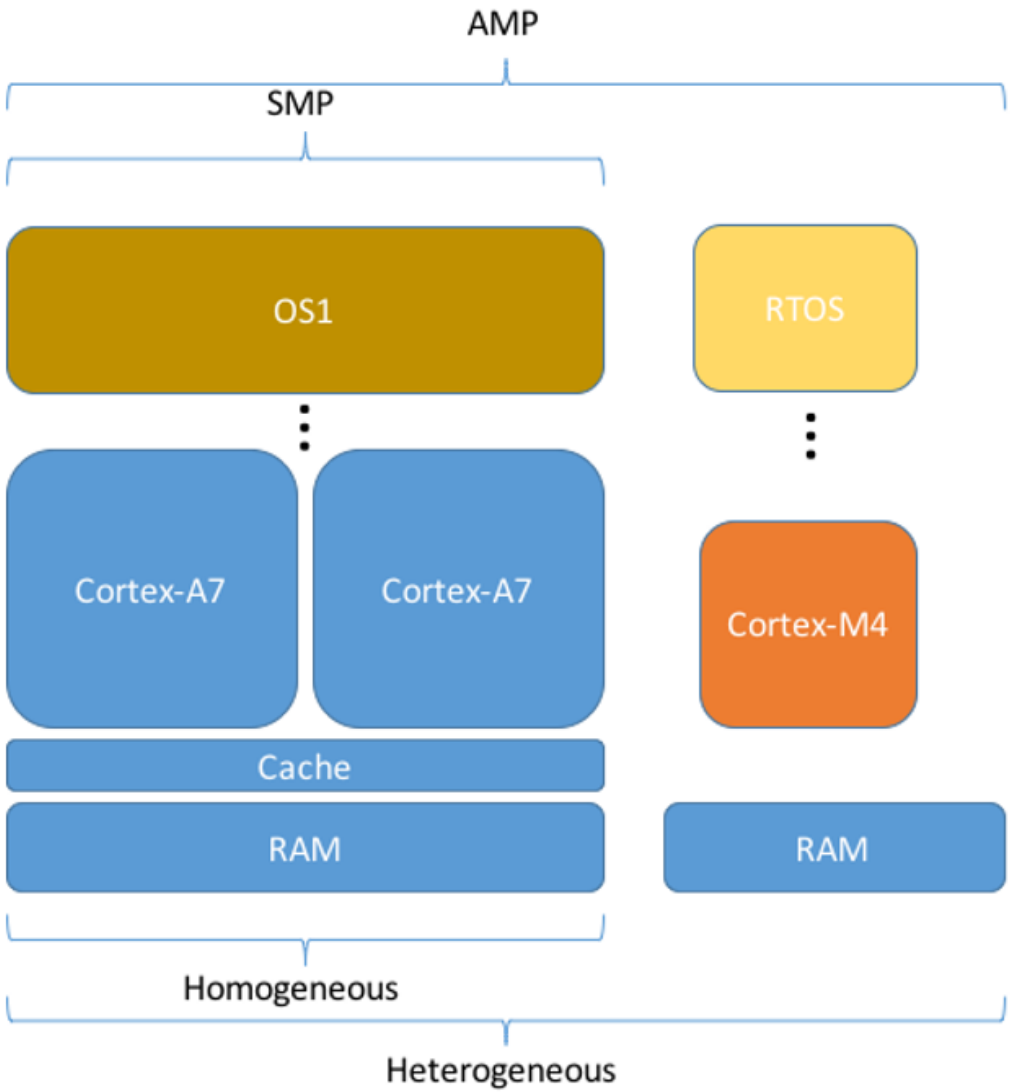
APL3

I.MX7

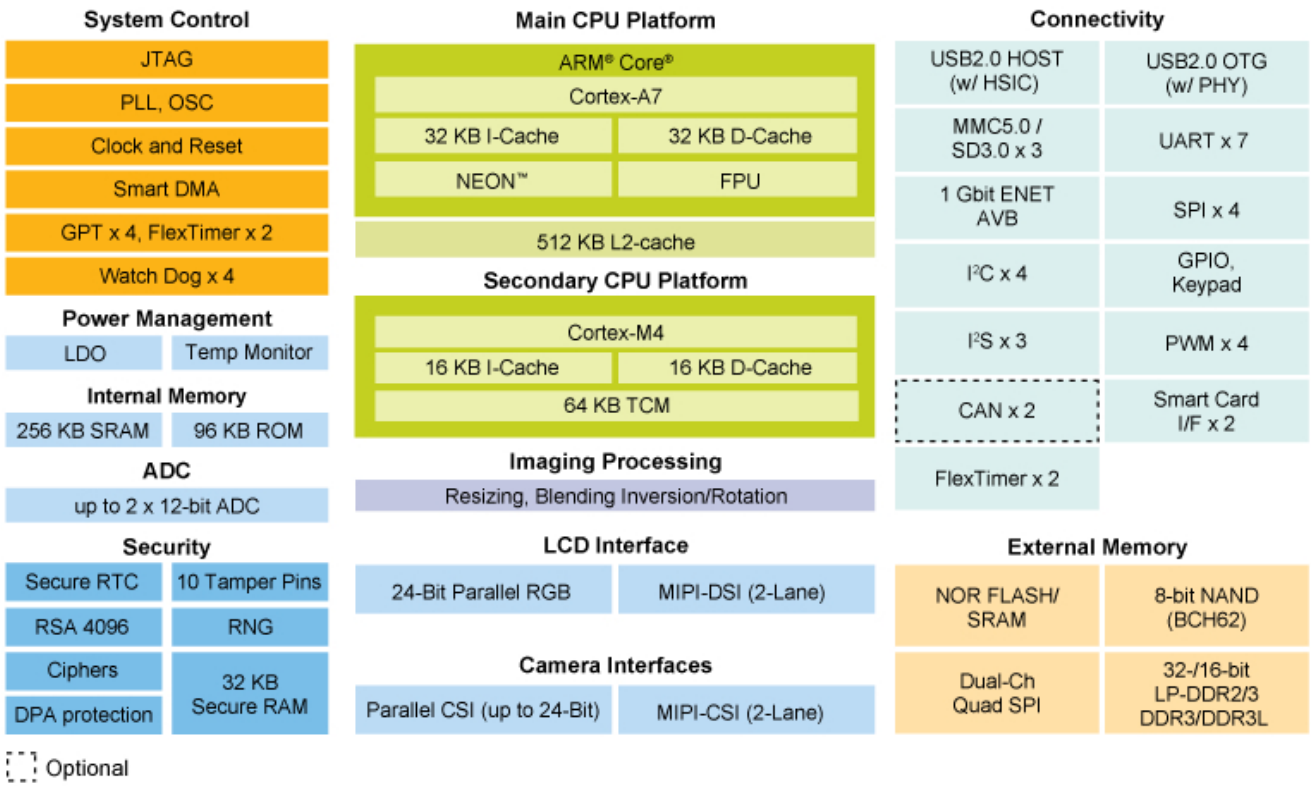
En quelques mots

- Architecture hétérogène :
  - Cortex A7 (1 pour l'`i.MX7s`, 2 pour l'`i.MX7d`)
  - Cortex M4
- Conçu pour :
  - être efficace en energie
  - avoir une "expérience" temps-réel
  - faire tourner des applications complexes
  - faire du multimédia
  - ...

AMP, SMP, Hétérogène, ...



Architecture



GPU Less

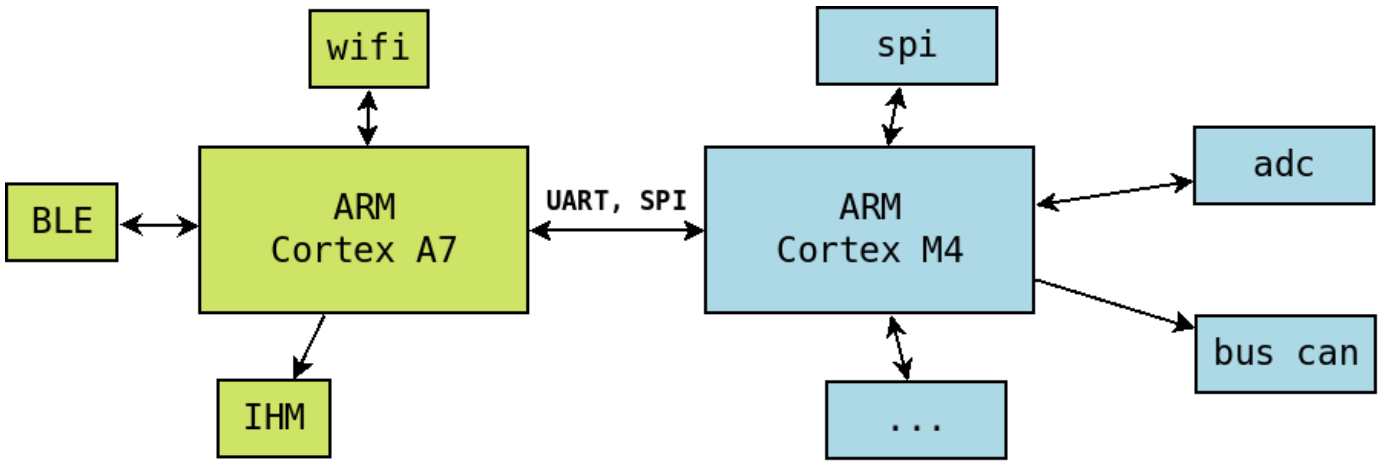
Il est quand même possible de faire tourner des applications !

- Utilisation de Qt Quick 2D Renderer => Depuis la version 5.7 de Qt
- Niveau framebuffer => `linuxfb`
- Il faudra configurer l'environnement :

```
$ export QT_QPA_PLATFORM=linuxfb:fb=/dev/fb0
$ export QMLSCENE_DEVICE=softwarecontext
```

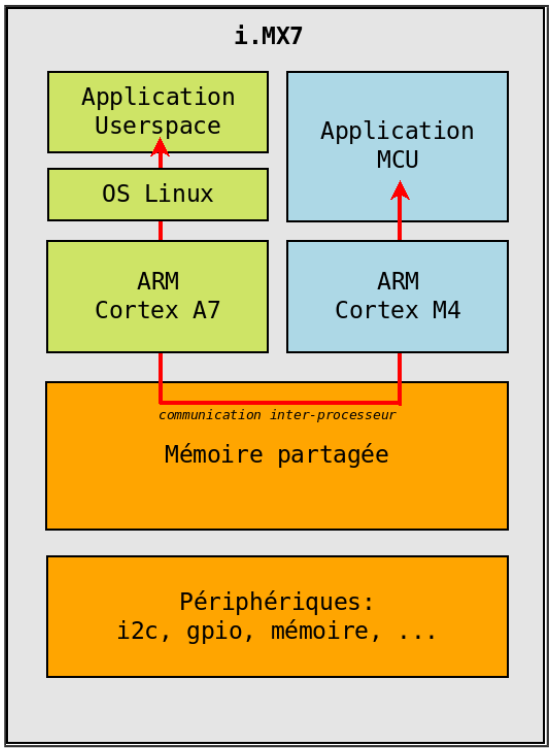
L'approche traditionelle

- MCU et MPU séparés
- Communication au travers un bus de communication (i2c ou SPI)
  - `LxMCU` ?
- Séparation de l'architecture logicielle
  - Le MCU pour les acquisitions (temps-réel)
  - Le MPU pour l'affichage/connectivités



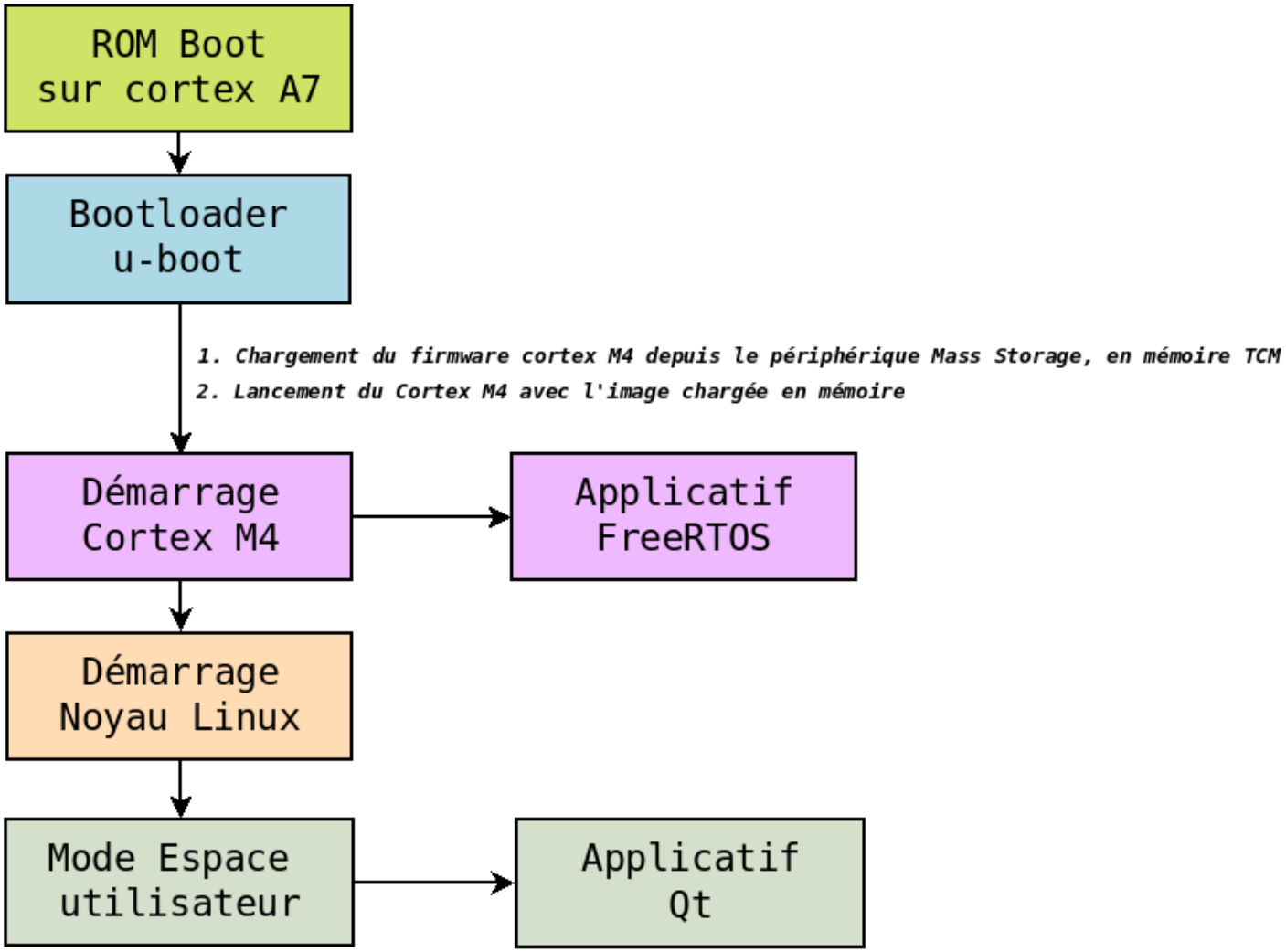
L'approche HMP

- Partagent les mêmes ressources matérielles (i2c, GPIO, ...)
- Communication en mémoire partagée: DDR3
- Avantages de l'architecture :
  - Plusieurs OS sur une même puce
  - Mise à jour plus simple (`mender.io`, `SWUpdate`, `rauc`)
  - Communication + rapide
  - Réduction du coût de la BOM
  - ...



Cortex M4: boot, mémoire, ...

Cortex M4


ROM Boot
sur cortex A7
Bootloader
u-boot
1. Chargement du firmware cortex M4 depuis le périphérique Mass Storage, en mémoire TCM
2. Lancement du Cortex M4 avec l'image chargée en mémoire
Démarrage
Cortex M4
Applicatif
FreeRTOS
Démarrage
Noyau Linux
Mode Espace
utilisateur
Applicatif
Qt

- Mémoire:
  - Plusieurs types de mémoires: `OCRAM`, `DDR3`, `TCM`
  - `TCM` locale au cortex M4, donc plus performante
  - Aussi accessible depuis le cortex A7
  - Utile pour le chargement de l'image depuis u-boot :

    ```
    => fatload mmc 0:1 0x7F8000 main.bin
    ```

  - Commande `bootaux`, pour démarrer à une adresse donnée:

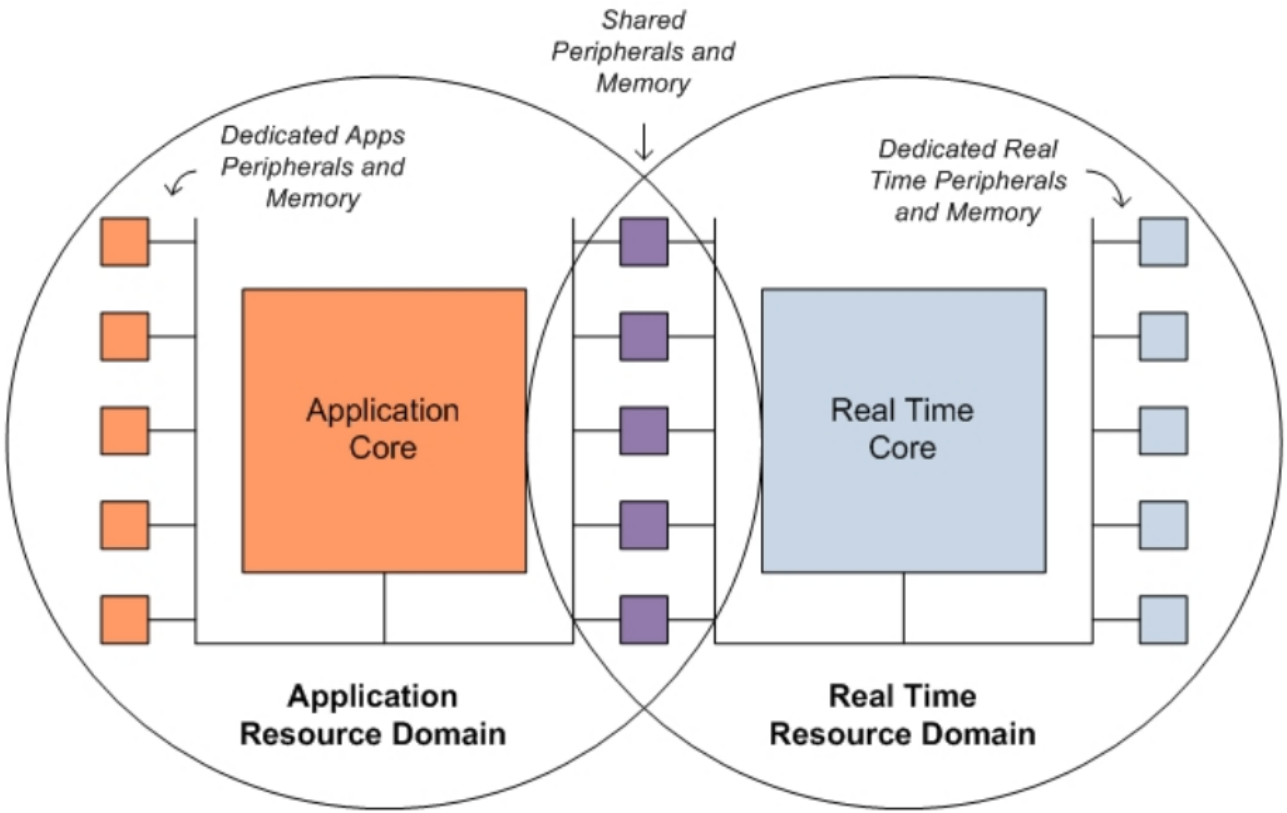
    ```
    => bootaux 0x7F8000
    ```

- Le cortex A7 est maître :
  - Démarre les horloges
  - Charge le code du cortex M4 à l'adresse TCM (TCML)
  - Démarre le coeur M4 (out of reset)

Resource Domain Controller

Resource Domain Controller

L'élément central du System on Chip



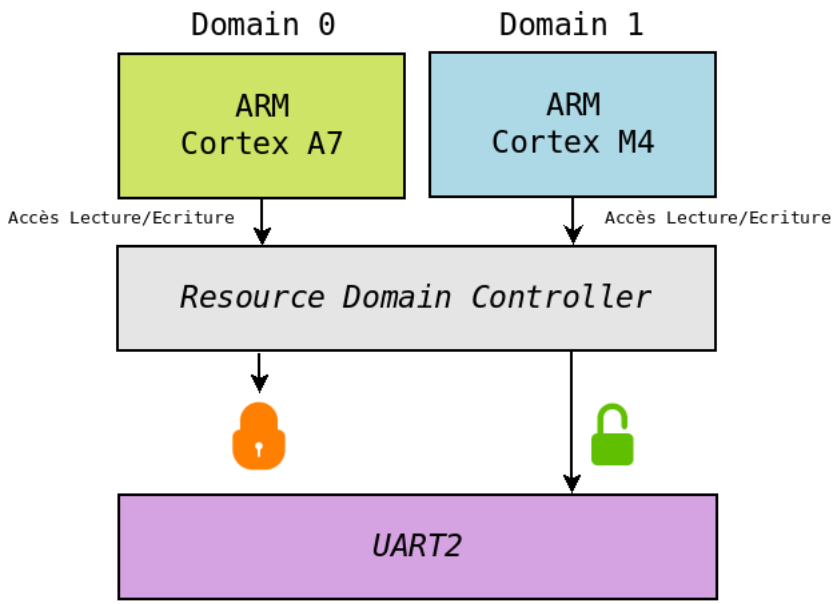
Resource Domain Controller: le garant (1/3)

- Sous-système pour le partionnement (domaine, périphérique, mémoire)
- Nécessaire pour éviter les accès concurrents !
- DomainID :
  - 4 domaines pour les bus maîtres (A7, M4, SDMA, ...)
  - Notion de MDA (Master Domain Assignment)
  - = 0 au boot
  - Dans notre cas, seul le cortex M4 est sur un domaine différent
  - Exemple :

```
RDC_SetDomainID(RDC,rdcMdaM4,1,false);
```

Resource Domain Controller: le garant (2/3)

- PDAP: Peripheral Domain Access Permission
  - Gestion des permissions des périphériques en fonction du `DomainID`
  - via la fonction `RDC_SetPdapAccess()`
  - Un exemple :



Resource Domain Controller: le garant (3/3)

- Pour réaliser cette implémentation :

```
72      /* Set debug uart for M4 core domain access only */
73      RDC_SetPdapAccess(RDC, BOARD_DEBUG_UART_RDC_PDAP, 3 << (BOARD_DOMAIN_ID * 2), false, false);
```

- Où:
  - Le 3ème paramètre de la fonction permet de remplir le registe PDAP
  - Dans notre cas, registre = 00001100 (R/W sur domaine 1)
  - Le 4ème paramètre, permet de sélectionner le bit SREQ pour l'utilisation des sémaphores matériels

3.2.5.6 Peripheral Domain Access Permissions (RDC_PDAP*n*)

Address: 303D_0000h base + 400h offset + (4d × i), where i=0d to 117d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	LCK	SREQ	Reserved													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved								D3R	D3W	D2R	D2W	D1R	D1W	D0R	D0W
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

RPMsg

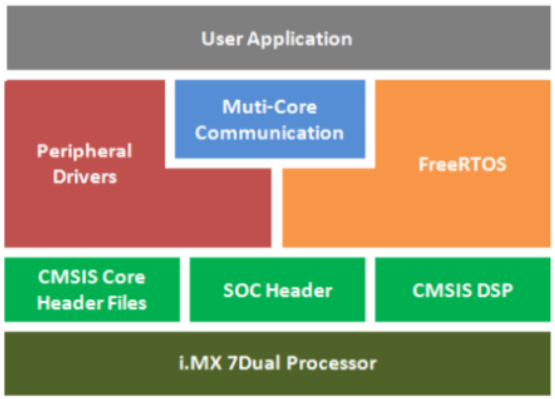
Explications

- RPMsg est un bus de communication basé sur la couche de transport VirtIO
  - Permet la gestion des IPC (Inter Processor Communication)
  - Architecture Client/Serveur
- VirtIO : une couche d'abstraction transport basée sur la mémoire partagée
  - Fournit une API Virtqueue pour la communication avec le processeur distant
  - Architecture à buffer circulaire -> VRING

Simplement :

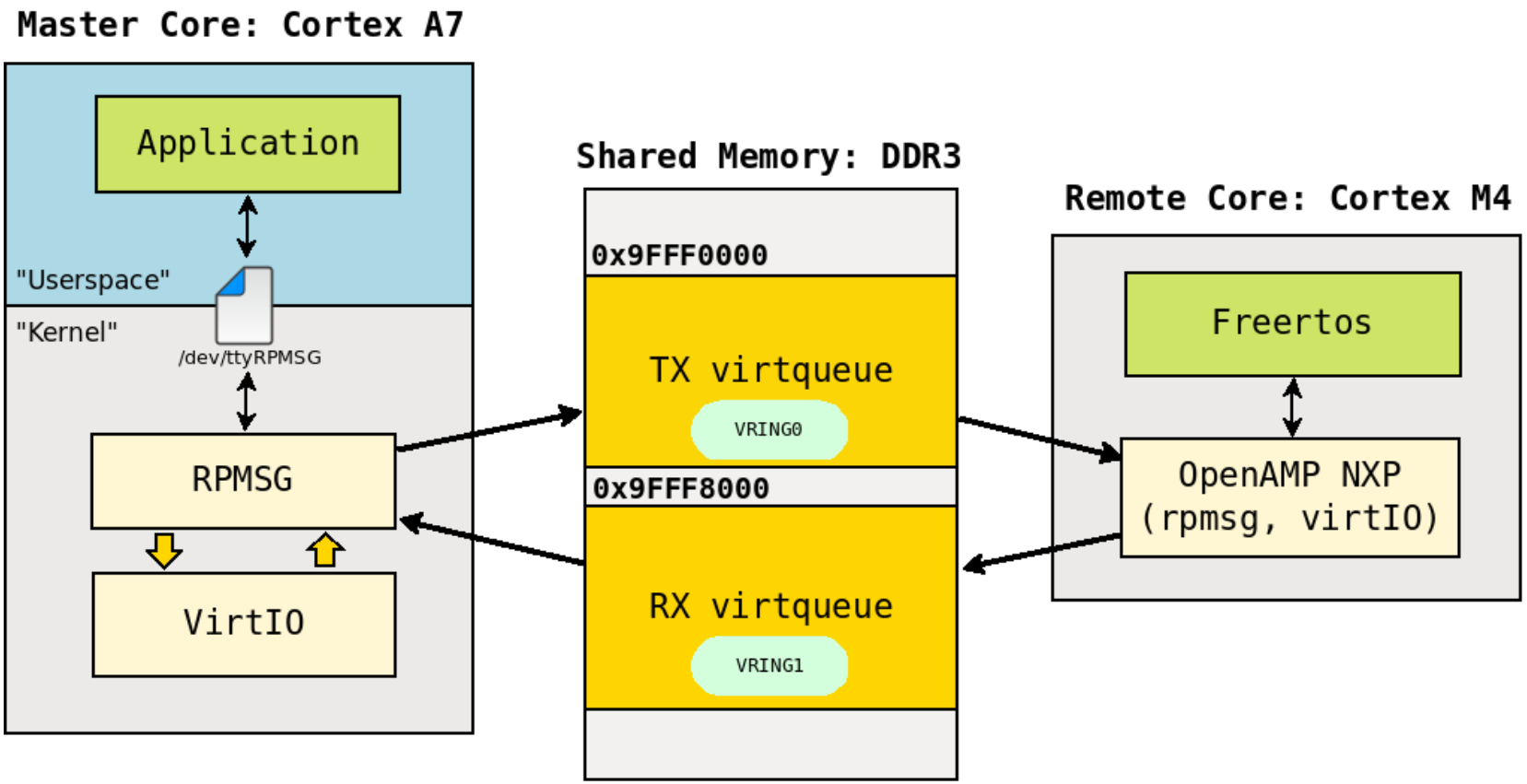
- 2 buffers VirtIO -> TX et RX
- 512 Octects chacun
- Emplacement VRING hard-coded (GNU/Linux & FreeRTOS)

- FreeRTOS
  - Basé sur OpenAMP
  - Portage réalisé par NXP (et la communauté !)



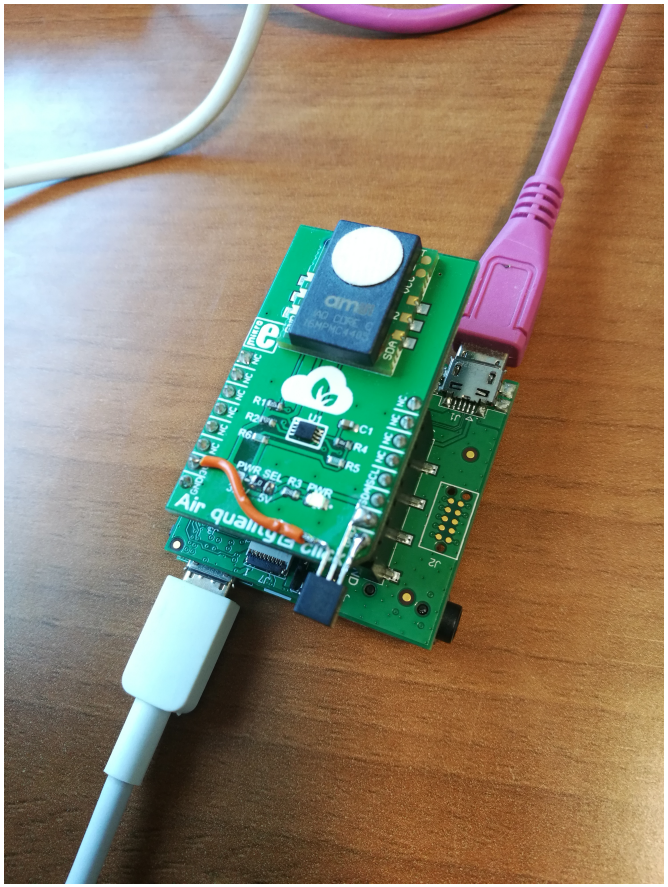
- GNU/Linux
  - Au travers une ligne série virtuelle (`/dev/ttyRPMSG`)
  - Driver fournit par NXP: `imx-rpmsg-tty`

Pour résumer ...



Etude de cas

Object connecté (LIBRE!) pour la mesure de la qualité de l'air



Les besoins

- La carte : WaRP7 (Wearable Reference Platform)

Les besoins

- La carte : WaRP7 (Wearable Reference Platform)
- FreeRTOS côté cortex M4
  - Acquisitions capteur + communication avec cortex A7
  - Utilisation de la socket MikroBUS

Les besoins

- La carte : WaRP7 (Wearable Reference Platform)
- FreeRTOS côté cortex M4
  - Acquisitions capteur + communication avec cortex A7
  - Utilisation de la socket MikroBUS

- Yocto/OpenEmbedded côté cortex A7
  - Une application Qt pour la récupération des données RPMsg
  - Une application Qt pour les échanges en BLE
  - `QSharedMemory` entre les 2 applications.

air-quality-click : *Acquisition + Dialogue A7*

- Tâche iaqDataTask
  - Acquisitions i2c
  - Mise à jour des champs de la structure de données (co2, TVOC, ...)
  - Utilisation de `vTaskDelay()`

air-quality-click : *Acquisition + Dialogue A7*

- Tâche iaqDataTask
  - Acquisitions i2c
  - Mise à jour des champs de la structure de données (co2, TVOC, ...)
  - Utilisation de `vTaskDelay()`
- Tâche commandTask
  - Création du canal RPMsg
  - Gestion des échanges avec le coeur distant

ServiceRPMSG : *Dialogue A7 <-> M4*

- Utilisation de la ligne série virtuelle `/dev/ttyRPMSG`
  - Configuration du descripteur de fichier associé (via `termios`)
  - Utilisation des primitives `read()` et `write()`
  - Protocole simple, ex:

    ```
    $ echo "?getAirQuality" > /dev/ttyRPMSG
    ```

- Mise à disposition des données dans un segment mémoire

ServiceGateway : *Connectivité sans-fil*

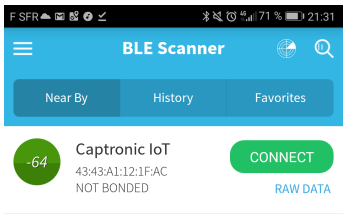
- Récupération des données du segment mémoire
- Utilisation de l'API Qt Bluetooth et `bluez5`
  - nom du périphérique : `Captronic IoT`
  - Utilisation du Service GATT `Automation I/O` ▸ doc
  - Avec la caractéristique GATT `Analog` ▸ doc
- Gestion:
  - des read -> "Notify"
  - des write -> `!printTVOC`

Démo sur cible

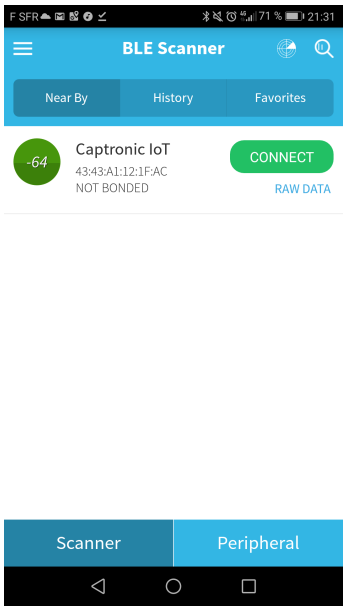
Démarrage de la cible

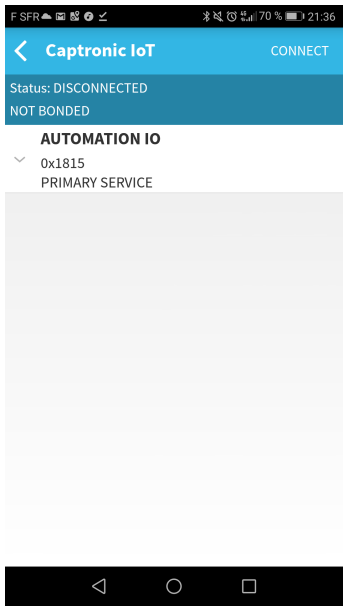
1. Démarrage du cortex M4
2. Démarrage du cortex A7
3. Insertion du driver pour la gestion RPMsg
4. Configuration de l'interface BLE => `hciconfig`
5. Démarrage des applications

- **BLE Scanner: découverte du périphérique**

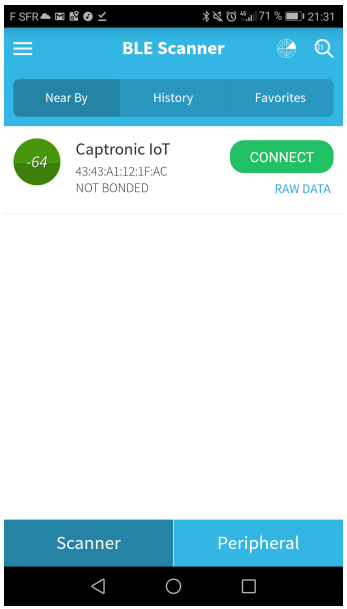


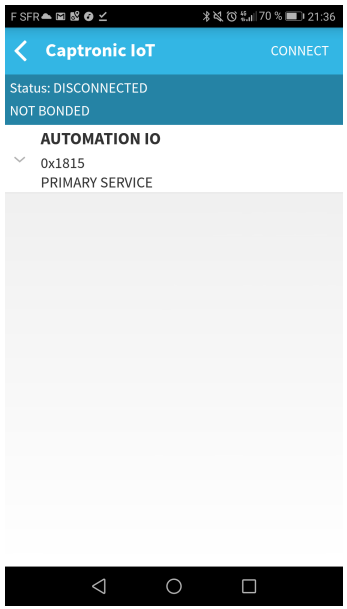
- **BLE Scanner: découverte du périphérique**
- **BLE Scanner: découverte du service**

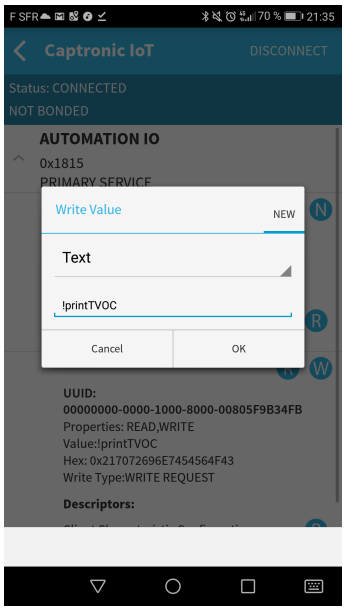




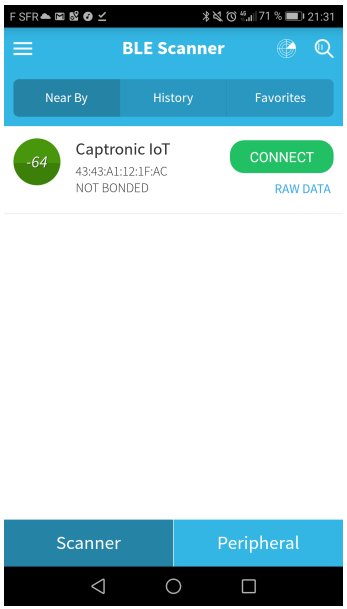
- BLE Scanner: découverte du périphérique
- BLE Scanner: découverte du service
- BLE Scanner: Gestion du write

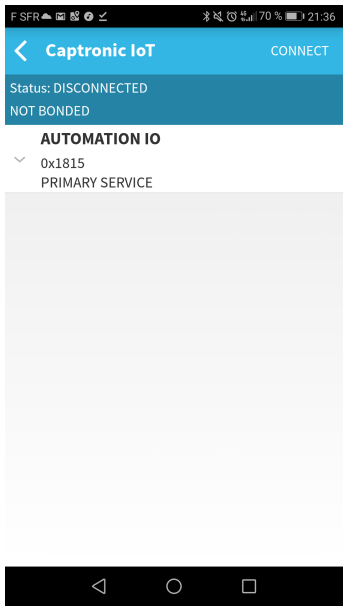


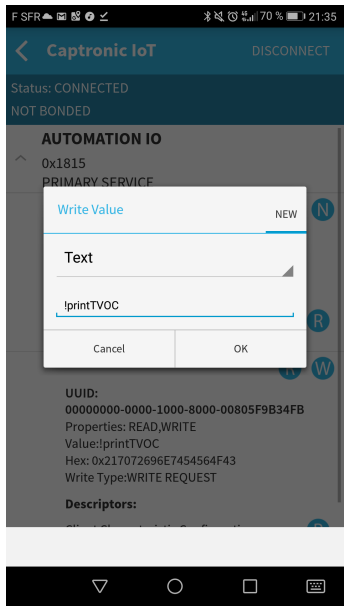


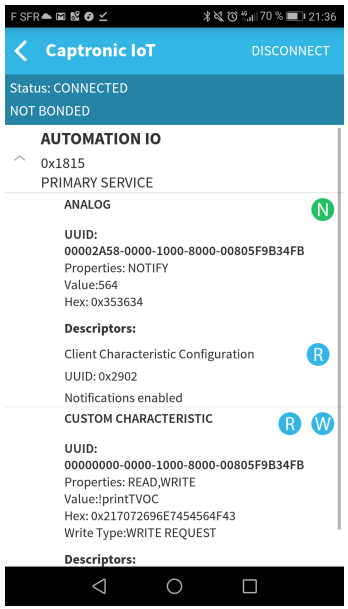


- BLE Scanner: découverte du périphérique
- BLE Scanner: découverte du service
- BLE Scanner: Gestion du write
- BLE Scanner: Gestion du read









may the (open) IoT be with you !

Questions?

texier.pj2@gmail.com

twitter: pjtexier