Rétro-ingénierie et détournement de piles protocolaires embarquées

Romain Cayre, Damien Cauquil

Qui sommes-nous ?

Romain Cayre, EURECOM

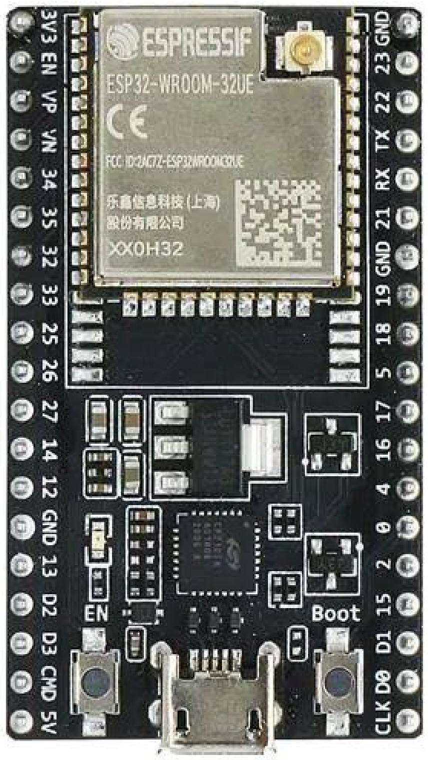
- mainteneur de *Mirage*, un couteau-suisse pour le protocole BLE
- amateur d'attaques inter-protocolaires (*Wazabee*)

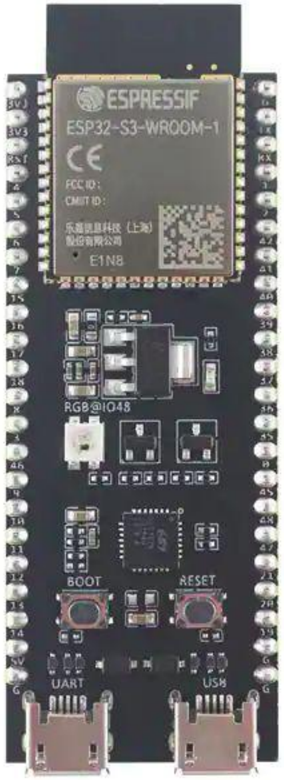
Damien Cauquil, Quarkslab

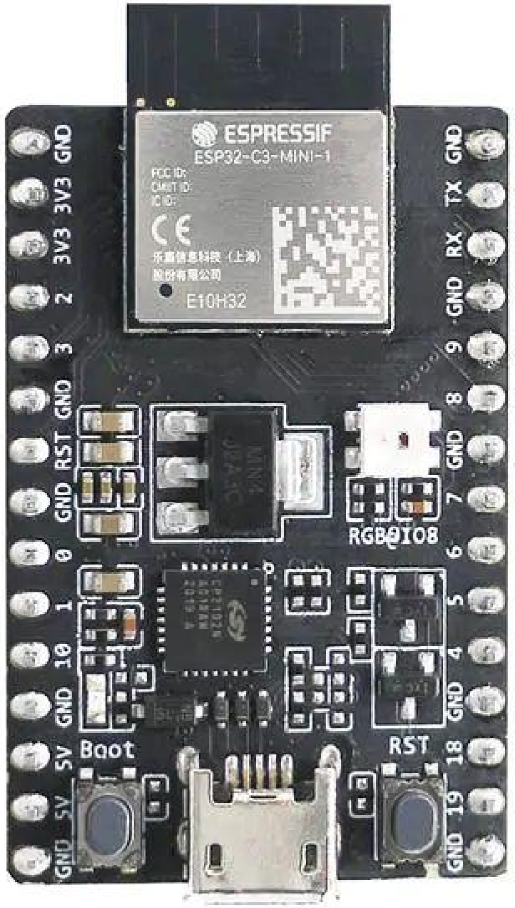
- mainteneur de *Btlejack*, un autre couteau-suisse pour BLE
- aime reverser des trucs, principalement de l'embarqué

Introduction

L'ESP32 et ses copains



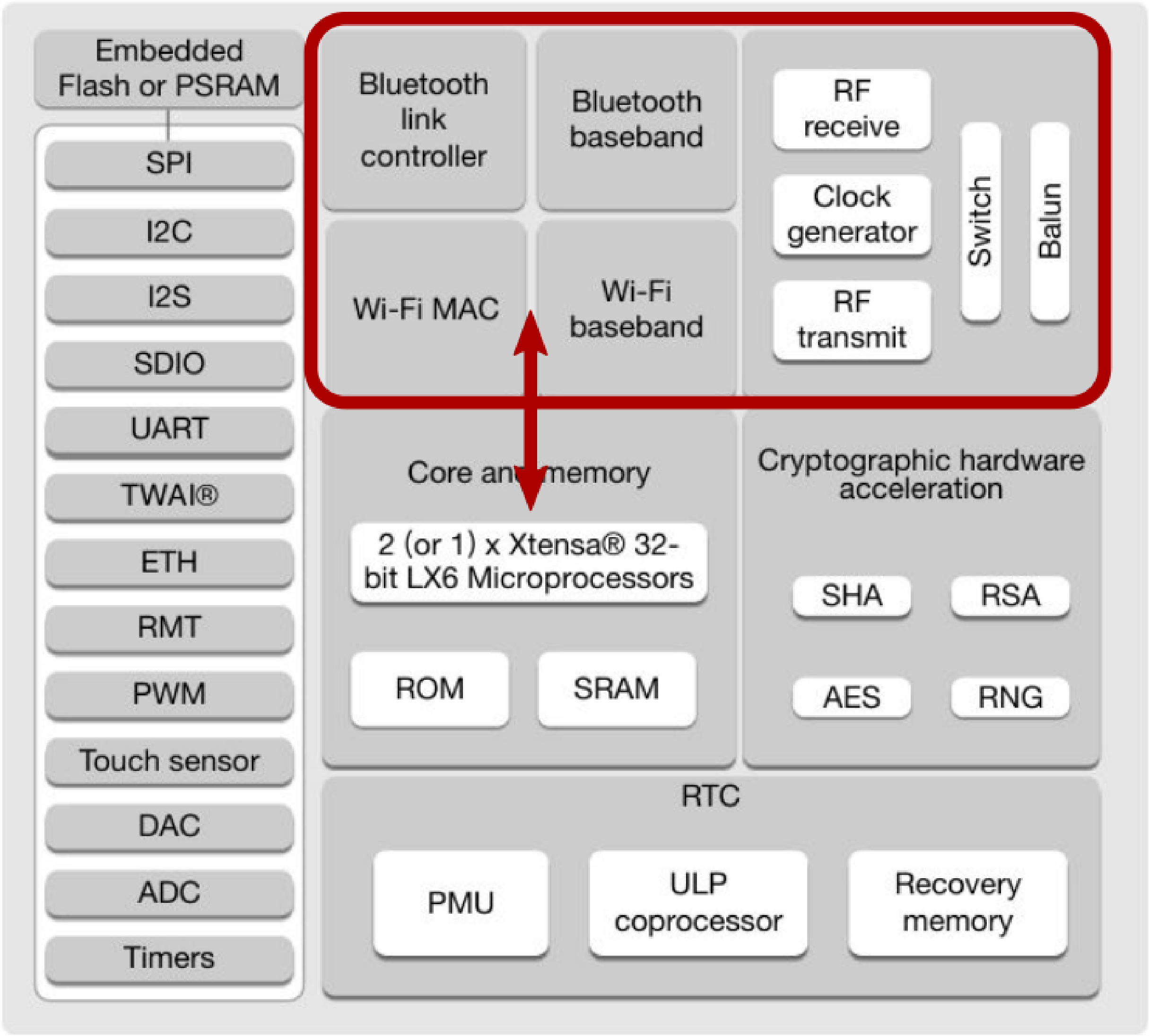




L'ESP32 et ses copains

- SoCs **bon marché** et **légers**
- **Très répandu** dans le monde de l'IoT
- Supporte les protocoles **WiFi**, **Bluetooth Low Energy / Bluetooth BR/EDR**
- Architectures **Tensilica Xtensa** (ESP32, ESP32-S3) et **RISC-V** (ESP32-C3)

ESP32: architecture globale



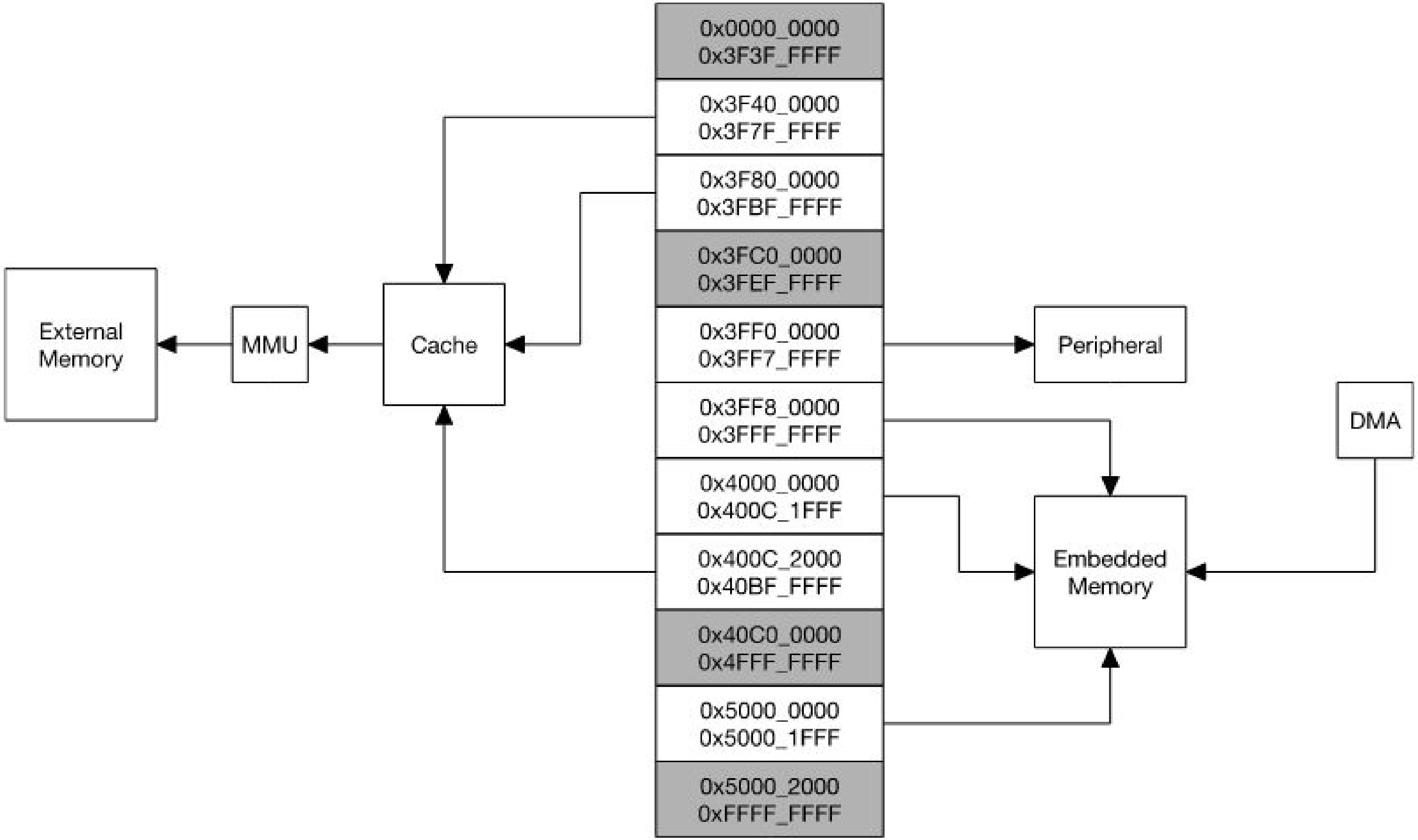
Problématique(s)

Est-il possible de:

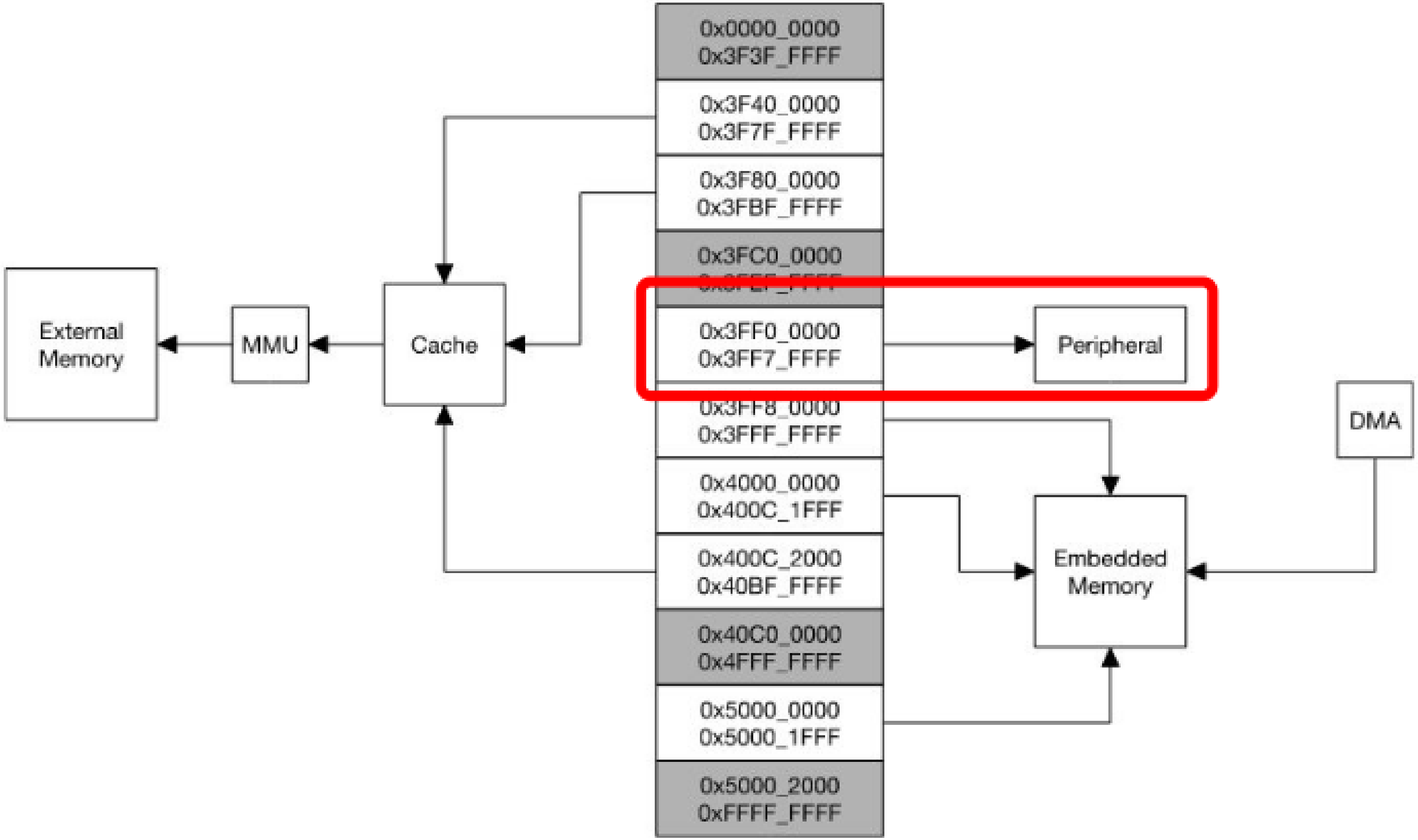
- **sniffer** les communications BLE ?
- **injecter** un paquet BLE arbitraire ?
- **détourner la couche PHY** à des fins offensives ?
- Interagir avec **d'autres protocoles sans-fil** ?
- Faire de l'ESP32 une **plate-forme d'attaque sans-fil** ?

Les entrailles de l'ESP32

Mapping mémoire



Mapping mémoire



Périphériques matériels

Bon nombre d'entre eux sont présents dans la documentation:

- `UART0`
- `GPIO`
- `SPI1`
- ...

Quid du contrôleur BLE ?

Un air de déjà vu ?

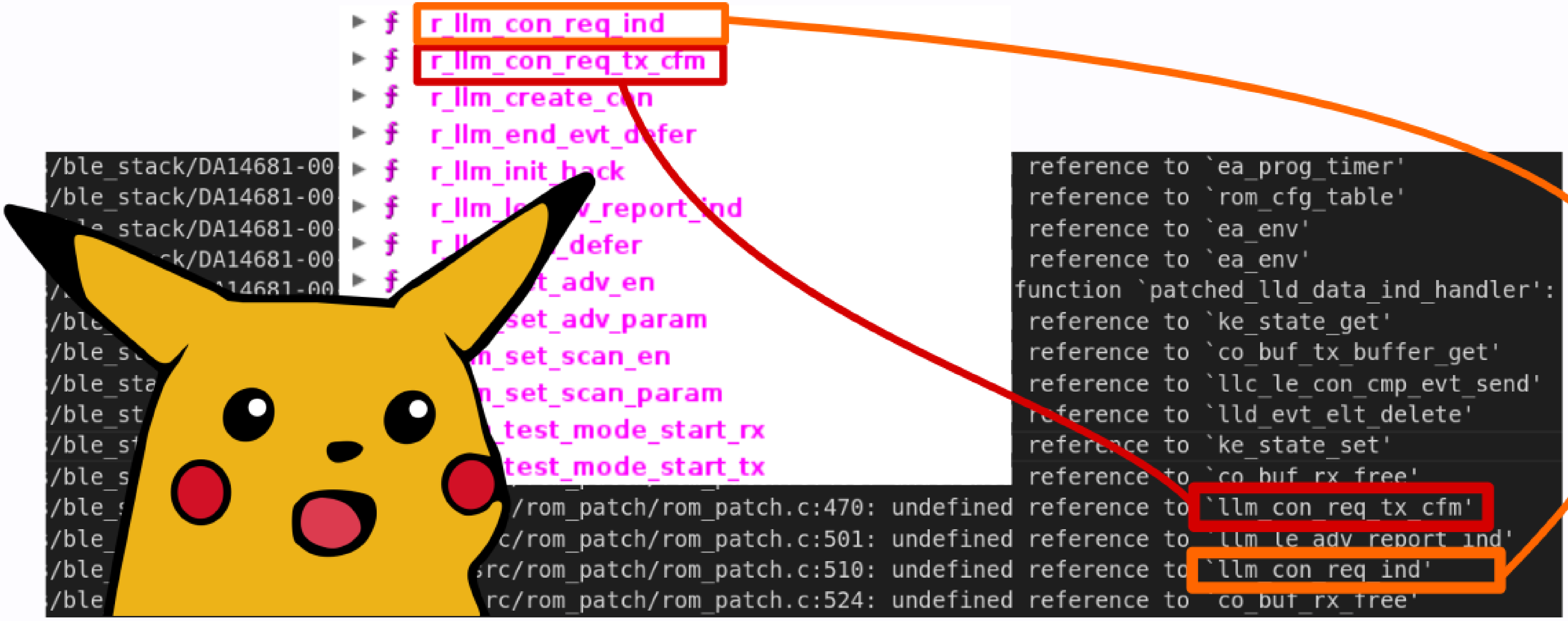
- f r_llm_con_req_ind
- f r_llm_con_req_tx_cfm
- f r_llm_create_con
- f r_llm_end_evt_defer
- f r_llm_init_hack
- f r_llm_le_adv_report_ind
- f r_llm_pdu_defer
- f r_llm_set_adv_en
- f r_llm_set_adv_param
- f r_llm_set_scan_en
- f r_llm_set_scan_param
- f r_llm_test_mode_start_rx
- f r_llm_test_mode_start_tx

Un air de déjà vu ?

Trouvé dans un pastebin sur Internet:

```
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:337: undefined reference to `ea_prog_timer'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:349: undefined reference to `rom_cfg_table'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:349: undefined reference to `ea_env'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:349: undefined reference to `ea_env'
/ble_stack/DA14681-00-Debug/libble_stack_da14681_00.a(rom_patch.o): In function `patched_lld_data_ind_handler':
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:381: undefined reference to `ke_state_get'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:427: undefined reference to `co_buf_tx_buffer_get'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:430: undefined reference to `llc_le_con_cmp_evt_send'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:433: undefined reference to `lld_evt_elt_delete'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:436: undefined reference to `ke_state_set'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:461: undefined reference to `co_buf_rx_free'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:470: undefined reference to `llm_con_req_tx_cfm'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:501: undefined reference to `llm_le_adv_report_ind'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:510: undefined reference to `llm_con_req_ind'
/ble_stack/DA14681-00-Debug/../src/rom_patch/rom_patch.c:524: undefined reference to `co_buf_rx_free'
```

Un air de déjà vu !



DA14681

DA14681

Bluetooth Low Energy 4.2 SoC

FINAL

Table 92: Register map BLE

Address	Port	Description
0x400000D8	BLE_TXMICVAL_REG	AES / CCM plain MIC value
0x400000DC	BLE_RXMICVAL_REG	AES / CCM plain MIC value
0x400000E0	BLE_RFTESTCNTL_REG	RF Testing Register
0x400000E4	BLE_RFTESTTXSTAT_REG	RF Testing Register
0x400000E8	BLE_RFTESTRXSTAT_REG	RF Testing Register
0x400000F0	BLE_TIMGENCNTL_REG	Timing Generator Register
0x400000F4	BLE_GROSSTIMTGT_REG	Gross Timer Target value
0x400000F8	BLE_FINETIMTGT_REG	Fine Timer Target value
0x400000FC	BLE_SAMPLECLK_REG	Samples the Base Time Counter
0x40000100	BLE_COEXIFCNTL0_REG	Coexistence interface Control 0 Register
0x40000104	BLE_COEXIFCNTL1_REG	Coexistence interface Control 1 Register
0x40000108	BLE_BLEMPRIO0_REG	Coexistence interface Priority 0 Register
0x4000010C	BLE_BLEMPRIO1_REG	Coexistence interface Priority 1 Register
0x40000110	BLE_BLEPRIOSCHARB_REG	Priority Scheduling Arbiter Control Register
0x40000200	BLE_CNTL2_REG	BLE Control Register 2
0x40000208	BLE_EM_BASE_REG	Exchange Memory Base Register
0x4000020C	BLE_DIAGCNTL2_REG	Debug use only
0x40000210	BLE_DIAGCNTL3_REG	Debug use only

Initialisation de la couche liaison

EURECOM Sophia Antipolis Quarkslab

```c
void r_lld_init()
{   [...]
    SYNCL = 0xbed6;
    DAT_3ffb0442 = 0x8e89;
    SYNCH = 0x8e89;
    DAT_3ffb0444 = 0x5555;
    CRCINIT0 = 0x5555;
    DAT_3ffb0446 = 0x55;
    CRCINIT1 = 0x55;
}
```

0x8e89bed6: L'**access address** utilisée pour les paquets d'annonce

0x555555: Graîne d'initialisation du CRC

Contrôleur BLE de l'ESP32

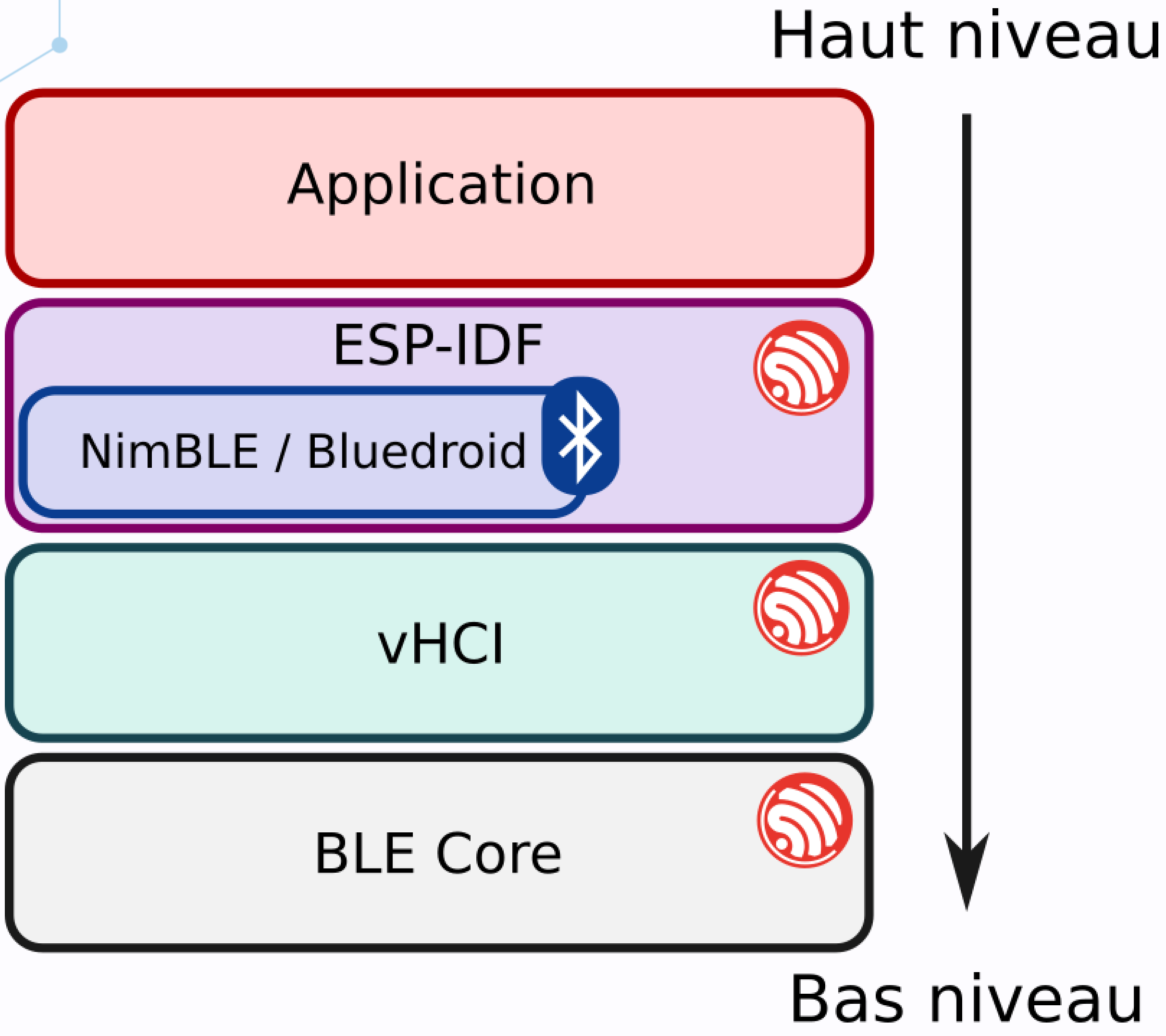
- **registres 16 bits**, identiques au DA14681
- **Même ordre des registres** dans le firmware ESP32
- Le SoC Dialog DA14681 est **intégralement documenté**

BLE hacking

Ce que l'on aimerait faire

- **Sniffer** des connexions BLE
- **Injecter** des paquets arbitraires
- Réaliser des attaques de type **Man-in-the-Middle**

Vue d'ensemble de la pile BLE

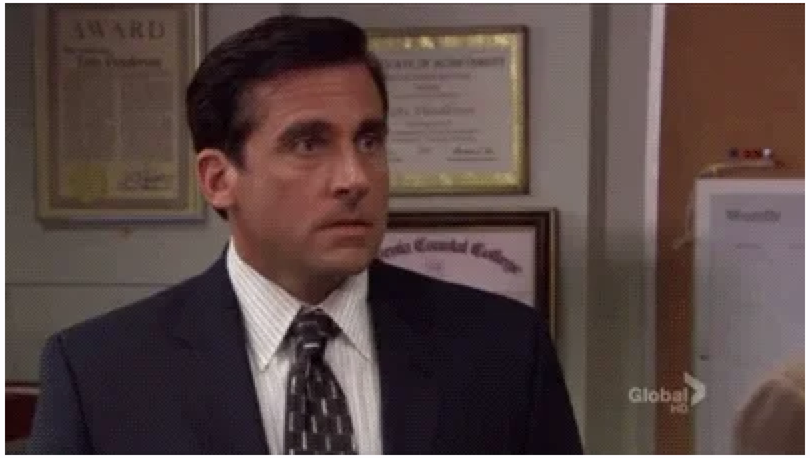


BLE Core (DA14681 ?)

- Couche **physique (PHY)**
- Gère les mécanismes **RF**
- **Périphérique** piloté par un *cerveau*

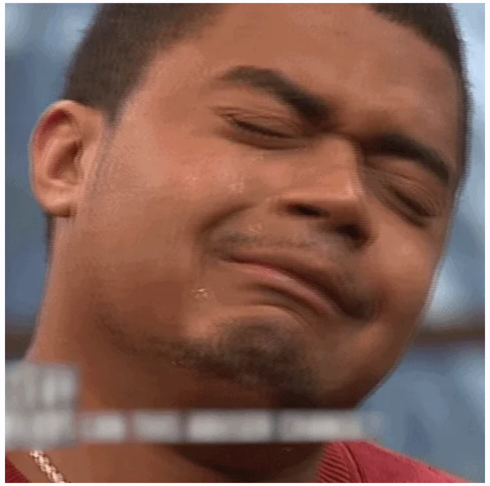
vHCI

- **V**irtual **H**ost **C**ontroller **I**nterface
- **Interface** avec les **couches supérieures** des piles BLE (NimBLE / Bluedroid)
- Messages standardisés, **pas d'injection ni d'écoute possible**



ESP-IDF

- Fournit des implémentations compatibles de **NimBLE** et **Bluedroid**
- Basé sur des **bibliothèques statiques**
- Peut être mis à jour en même temps que *ESP-IDF*
- **Trop dépendant** de la version d'*ESP-IDF*



Il reste peut-être un endroit ...

EURECOM Sophia Antipolis Quarkslab

Category	Target	Start Address	End Address	Size
Embedded Memory	Internal ROM 0	0x4000_0000	0x4005_FFFF	384 KB
	Internal ROM 1	0x3FF9_0000	0x3FF9_FFFF	64 KB
	Internal SRAM 0	0x4007_0000	0x4009_FFFF	192 KB
	Internal SRAM 1	0x3FFE_0000	0x3FFF_FFFF	128 KB
		0x400A_0000	0x400B_FFFF	
	Internal SRAM 2	0x3FFA_E000	0x3FFD_FFFF	200 KB
	RTC FAST Memory	0x3FF8_0000	0x3FF8_1FFF	8 KB
		0x400C_0000	0x400C_1FFF	
	RTC SLOW Memory	0x5000_0000	0x5000_1FFF	8 KB
External Memory	External Flash	0x3F40_0000	0x3F7F_FFFF	4 MB
		0x400C_2000	0x40BF_FFFF	11 MB+248 KB
	External RAM	0x3F80_0000	0x3FBF_FFFF	4 MB

ROMs internes de l'ESP32

- 2 **zones mémoires spécifiques** (lecture seule)
- Ces zones contiennent du **code et des données**
- **API bas-niveau** pour contrôler le BLE Core
- **Nouveau problème**: comment intercepter ces fonctions ?

Interception des fonctions en ROM

- Les fonctions ROM sont appelées via `r_ip_funcs_p`
- `r_ip_funcs_p` est un **tableau de pointeurs de fonctions** stocké en **RAM**

```
400ea86a 41 df e8      l32r      a4,->r_ip_funcs_p
400ea86d 48 04         l32i.n    a4=>r_ip_funcs_p,a4,0x0
400ea86f 42 d4 0a      addmi     a4,a4,0xa00
400ea872 42 24 2f      l32i      a4,a4,0xbc
400ea875 e0 04 00      callx8    a4
```

Correction de fonctions ROM

```
undefined4 config_lld_funcs_reset(undefined4 param_1)

{
  int iVar1;
  code **ppcVar2;

  iVar1 = r_ip_funcs_p;
  ppcVar2 = (code **)(r_ip_funcs_p + 0x900);
  *(code **)(r_ip_funcs_p + 0x8f0) = r_lld_init;
  *(code **)(iVar1 + 0x8f8) = r_lld_adv_start;
  *(code **)(iVar1 + 0x8fc) = r_lld_adv_stop_hack;
  *ppcVar2 = r_lld_scan_start_hack;
  *(code **)(iVar1 + 0x904) = r_lld_scan_stop_hack;
  *(code **)(iVar1 + 0x908) = r_lld_con_start;
  *(code **)(iVar1 + 0x90c) = r_lld_move_to_master_hack;
  *(code **)(iVar1 + 0x928) = r_lld_move_to_slave_hack;
  *(code **)(iVar1 + 0x924) = r_lld_get_mode;
  *(code **)(iVar1 + 0x914) = r_lld_con_update_after_param_req;
  return param_1;
}
```

Interception de fonctions

```c
// Define a similar function pointer
typedef int (*F_r_lld_pdu_rx_handler)(void* evt, uint32_t nb_rx_desc);

// Original function backup
F_r_lld_pdu_rx_handler old_r_lld_pdu_rx_handler = NULL;

// Hooking function
int rx_custom_callback(void* evt, uint32_t nb_rx_desc) {
    esp_rom_printf("Hooked :)\n");
    return old_r_lld_pdu_rx_handler(evt,nb_rx_desc);
}

void setup_hooks() {

    /* Setup RX callback hook */
    // Save the old function pointer to old_r_lld_pdu_rx_handler
    old_r_lld_pdu_rx_handler = (*r_ip_funcs_p)[603];

    // Inject our rx_custom_callback hook
    (*r_ip_funcs_p)[RX_SCAN_CALLBACK_INDEX] = (void*)rx_custom_callback;
}
```

Capture de paquets BLE

Deux fonctions principales à intercepter:

- `r_lld_pdu_rx_handler()` : appelée quand un **paquet BLE est reçu**
- `r_lld_pdu_data_tx_push()` : utilisée pour **envoyer un paquet BLE**

La capture de paquets est possible, mais seulement pour les connexions établies par notre ESP32 🙁

Injection de paquet BLE

EURECOM Sophia Antipolis Quarkslab

- On utilise `r_lld_pdu_data_tx_push()` pour envoyer un paquet BLE
- **Pas de vérification**, on peut envoyer des **paquets de données et de contrôle** ! 🤣
- Nécessite **quelques ajustements**, mais fonctionne parfaitement

Attaques de type Man-in-the-Middle

EURECOM Sophia Antipolis
Quarkslab

- On a besoin d'un **contrôle complet** du BLE core
- Ou *a minima* de pouvoir gérer **deux connexions simultanées**
- Cela **semble impossible** à réaliser avec un ESP32

Quelques hacks cools 👌

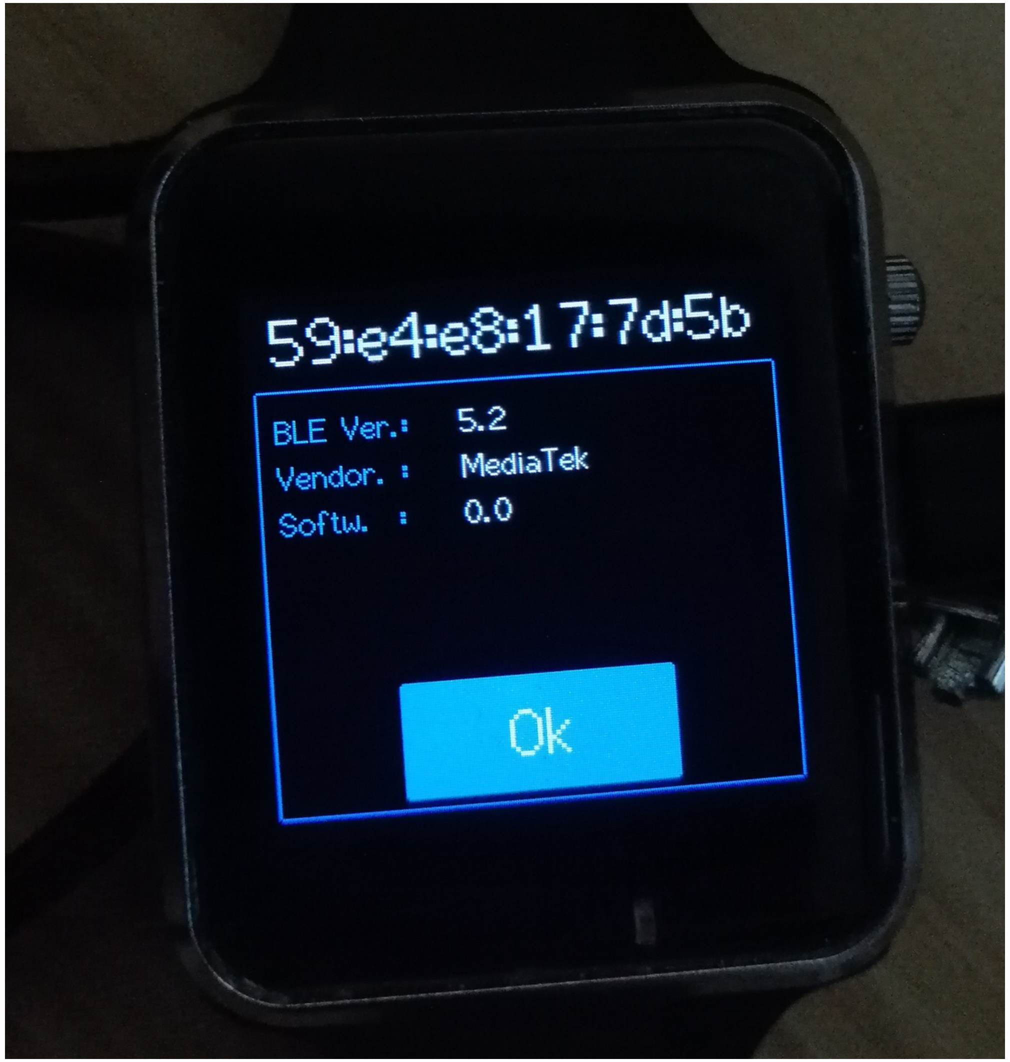
Injection d'un LL_VERSION_IND

```
def send_version(self):
    """Send LL_VERSION_IND PDU.
    """
    if not self.__version_sent:
        # Mark version as sent
        self.__version_sent = True

        # Send LL_VERSION_IND PDU
        self.send_control(
            BTLE_CTRL() / LL_VERSION_IND(
                version=self.__llm.stack.bt_version,
                company=self.__llm.stack.manufacturer_id,
                subversion=self.__llm.stack.bt_sub_version
            )
        )
```

Injection d'un LL_VERSION_IND

Prise d'empreinte active



Détournement de la couche physique

Attaques inter-protocolaires

Est-il possible de détourner la couche PHY de l'ESP32 pour interagir avec d'autres protocoles ?

- Le protocole BLE repose sur une modulation **Gaussian Frequency Shift Keying** (GFSK)
- ... comme des **dizaines de protocoles propriétaires**:
  ANT+ / ANT-FS, Riitek, MosArt, Logitech Unifying, Microsoft...
  → **primitives de réception et d'émission GFSK arbitraires**
- **WazaBee:** équivalence modulation O-QPSK (802.15.4) et GFSK à 2 Mbps (BLE 2M)

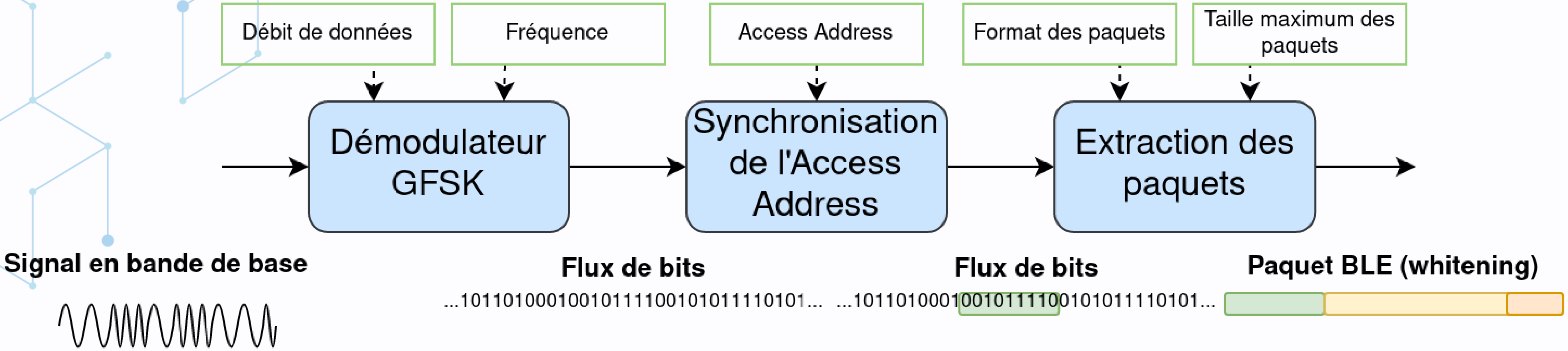
Attaques inter-protocolaires

EURECOM Sophia Antipolis Quarkslab

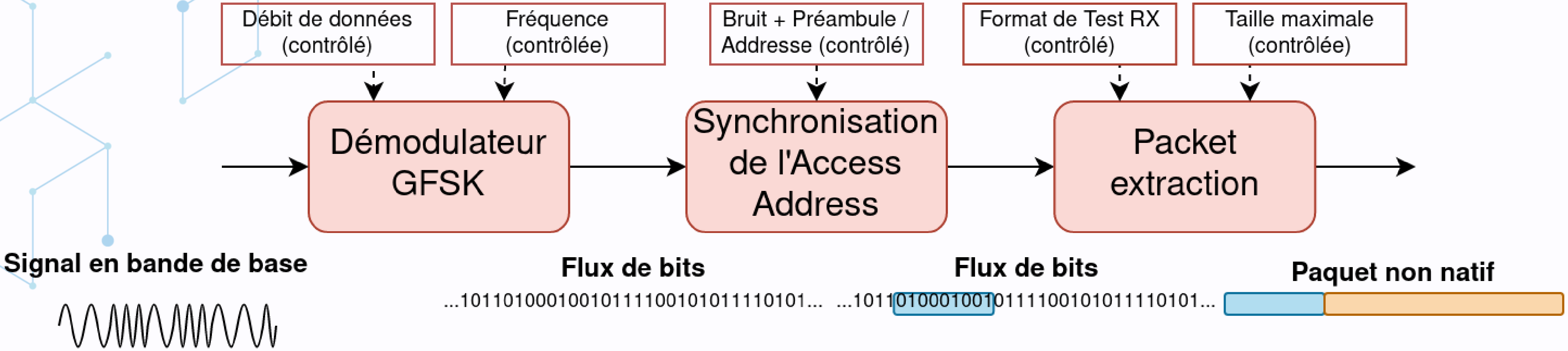
On doit contrôler les **paramètres radio bas-niveau**:

- contrôle du CRC
- fréquence centrale
- débit de données (*datarate*)
- mot de synchronisation
- transformation numérique: *whitening / dewhitening*
- flux de bits en entrée et sortie de modulation/démodulation

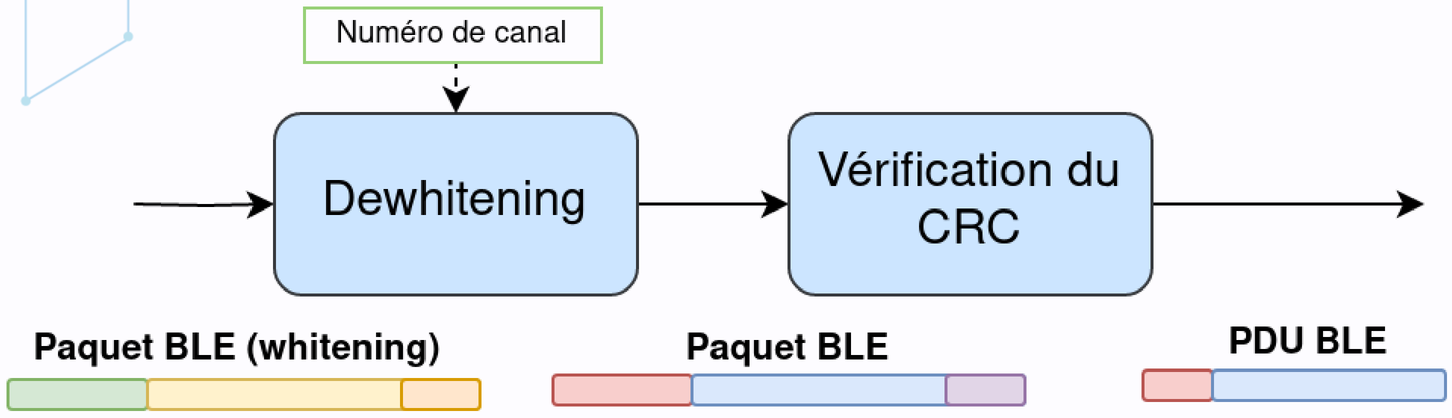
Synchronisation BLE en réception



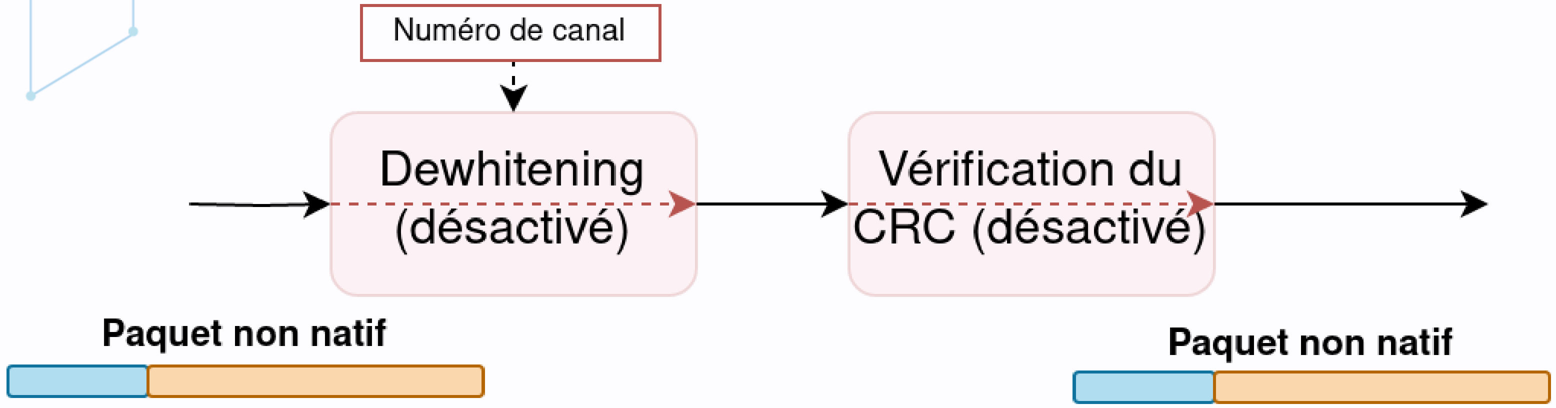
Détournement de la synchronisation



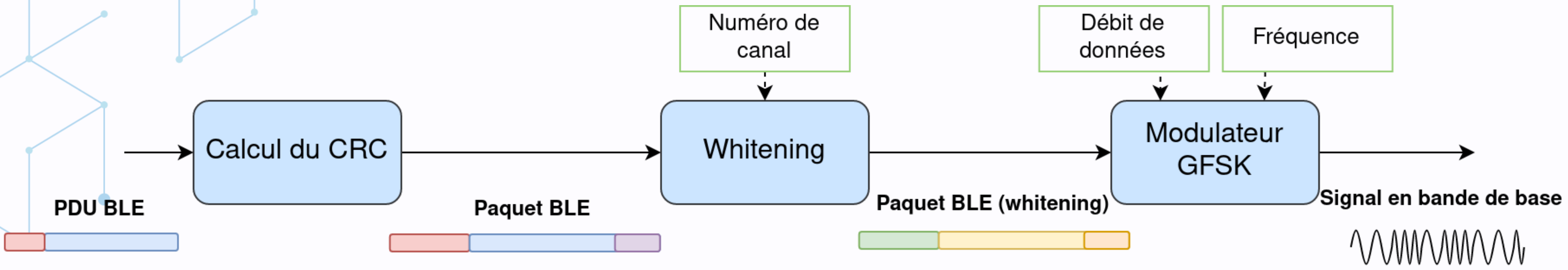
Traitement de paquet



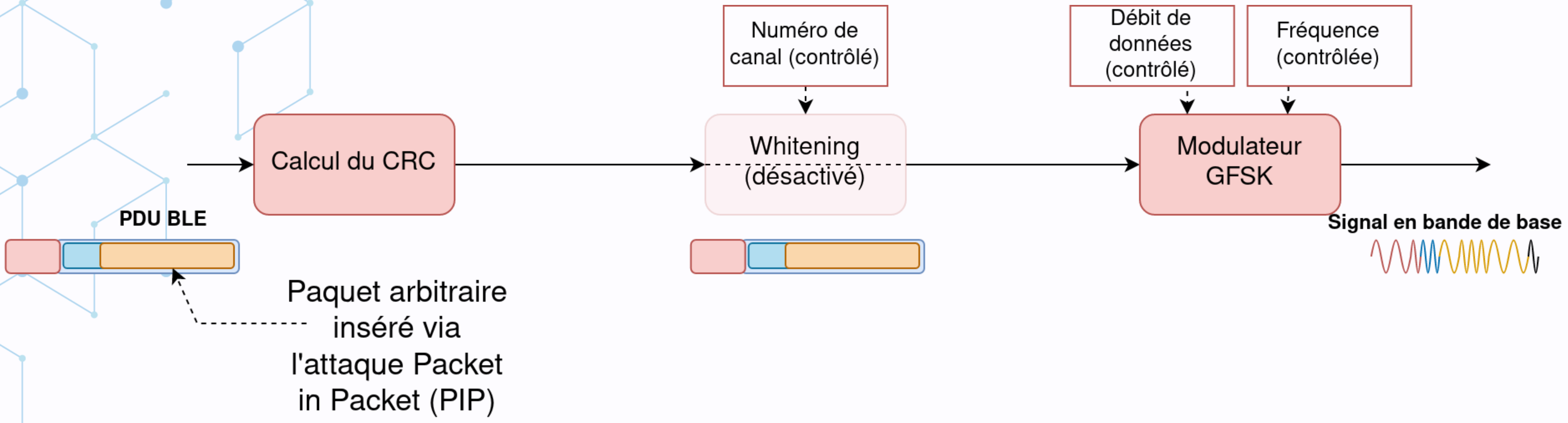
Contournement du traitement



Transmission de paquet BLE

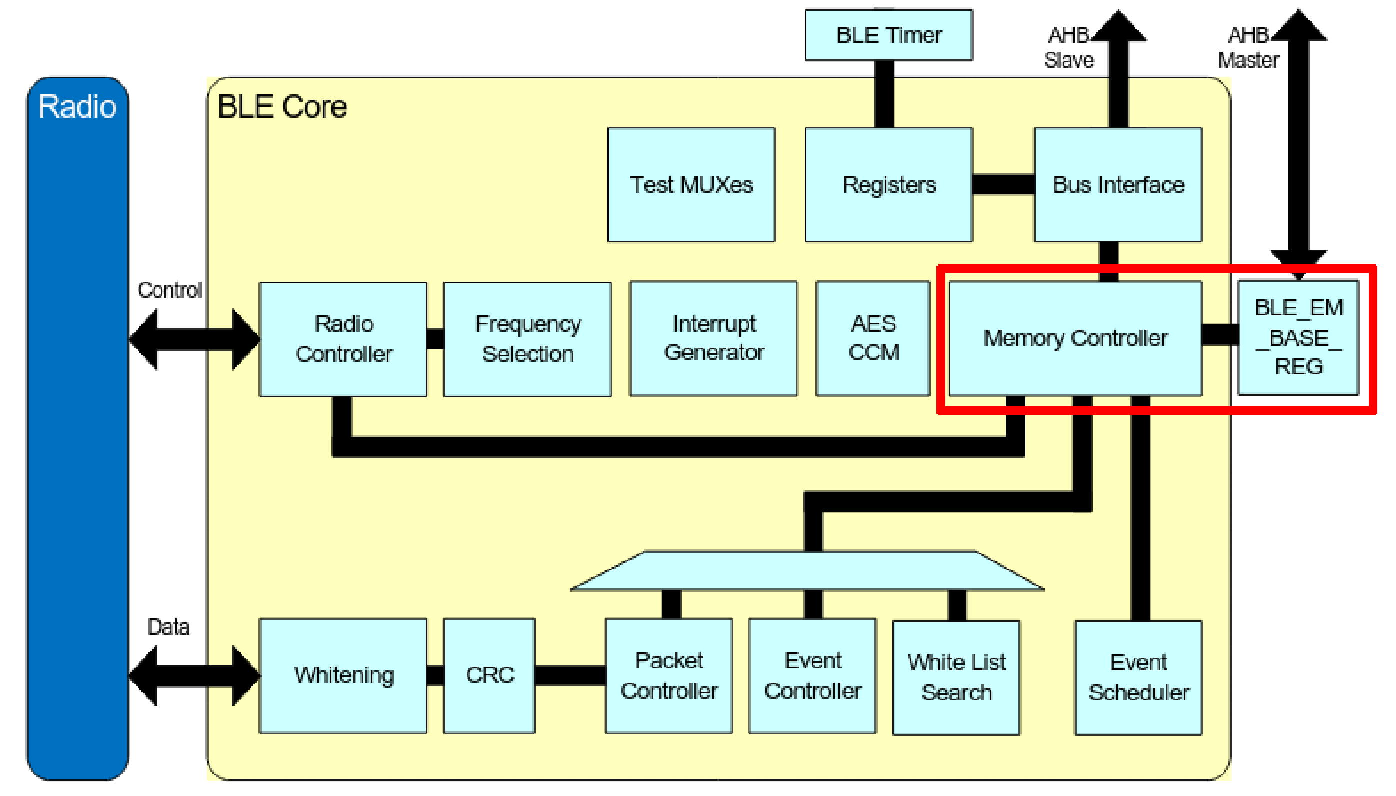


Packet in Packet



Interaction avec le BLE Core

Le BLE Core est programmé grâce à une zone mémoire partagée dénommée *Exchange Memory*.



Exchange Memory

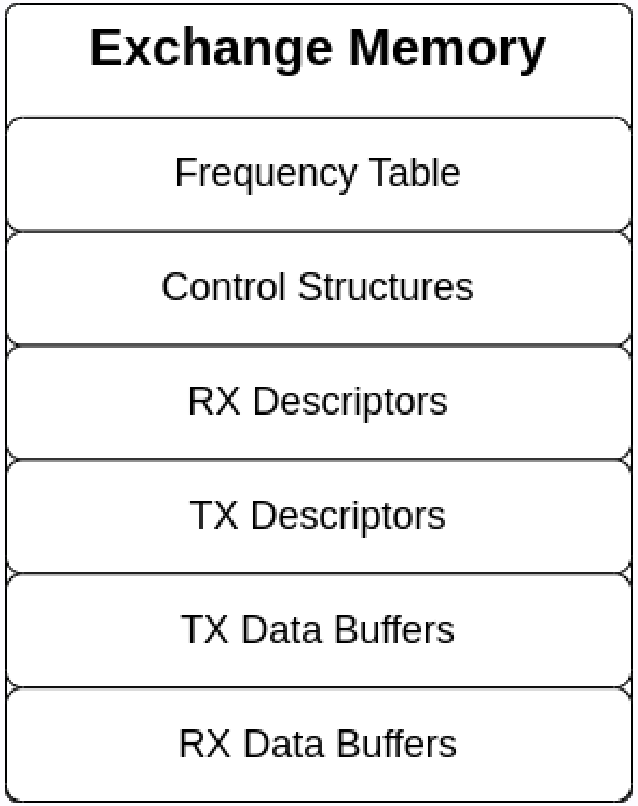
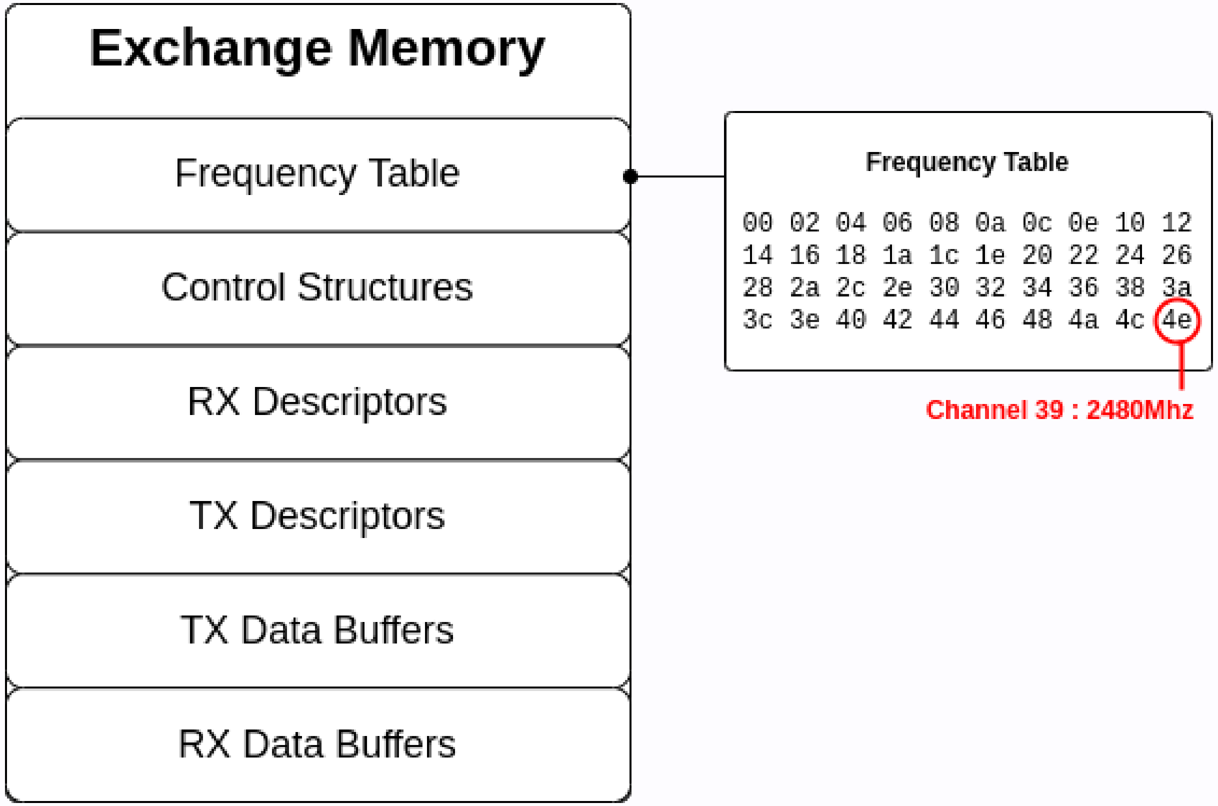
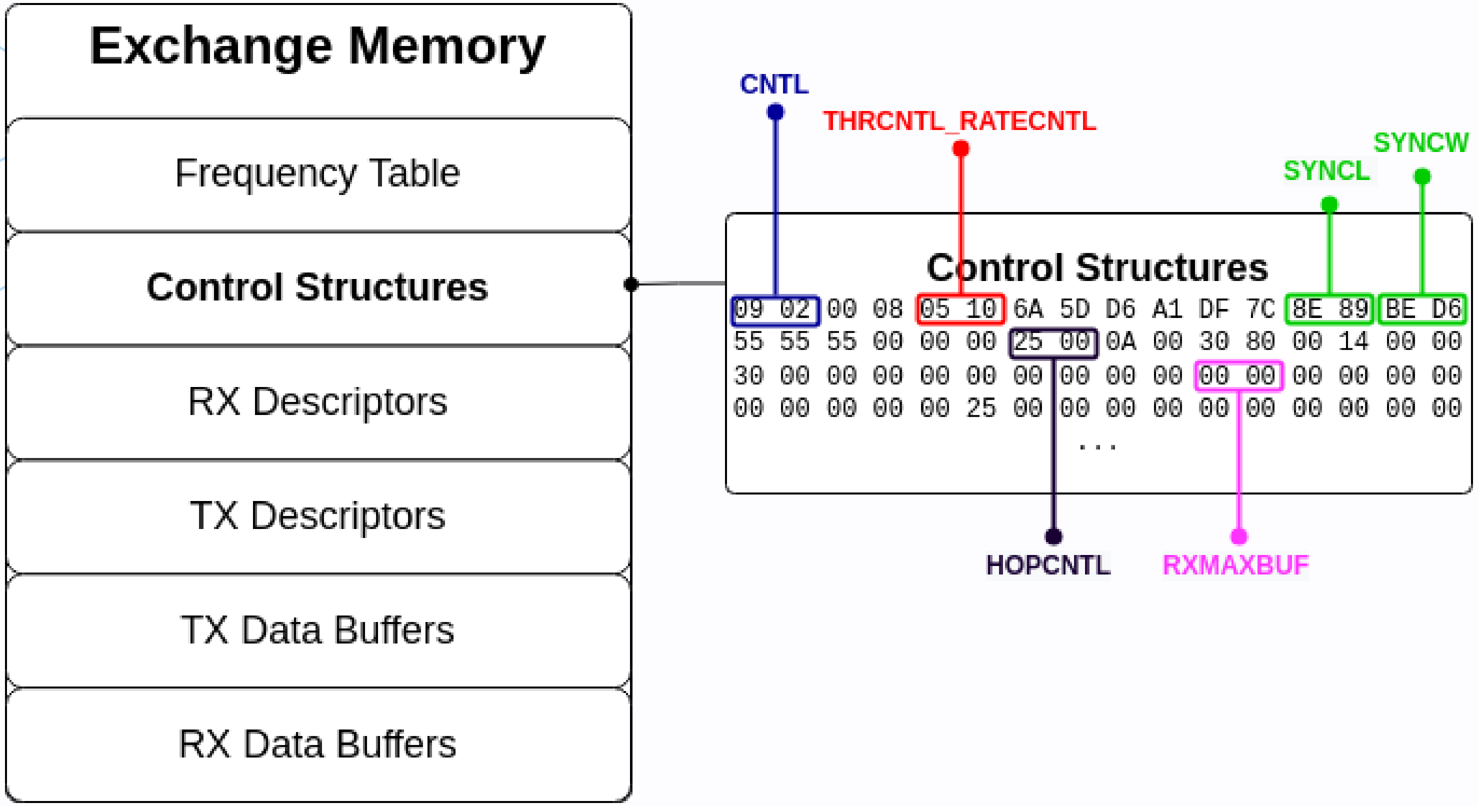


Tableau des fréquences

EURECOM Sophia Antipolis
Quarkslab



Structures de contrôle



Registres BLE

```
RWBLECNTL Register Definition

  Bits             Field Name   Reset Value
 -----    ------------------   -----------
    31        MASTER_SOFT_RST   0
    30      MASTER_TGSOFT_RST   0
    29           REG_SOFT_RST   0
    28              SWINT_REQ   0
    26           RFTEST_ABORT   0
    25           ADVERT_ABORT   0
    24             SCAN_ABORT   0
    22                 MD_DSB   0
    21                 SN_DSB   0
    20               NESN_DSB   0
    19              CRYPT_DSB   0
    18               WHIT_DSB   0
    17                CRC_DSB   0
    16          HOP_REMAP_DSB   0
    09          ADVERTFILT_EN   0
    08               RWBLE_EN   0
 07:04             RXWINSZDEF   0x0
 02:00                SYNCERR   0x0
```

Primitive de réception arbitraire

Interception de `r_llm_start_scan_en()` et **modification des paramètres radio**:

- **Tableau des fréquences:** modification de la fréquence du canal n°39
- **Structures de contrôle:** on force le canal 39, désactive le saut de canal, configure le mot de synchronisation et le *datarate*
- **Registres globaux:** désactivation du *whitening* et du CRC

Interception de `r_lld_pdu_rx_handler()` pour **extraire les paquets**

Primitive d'émission arbitraire

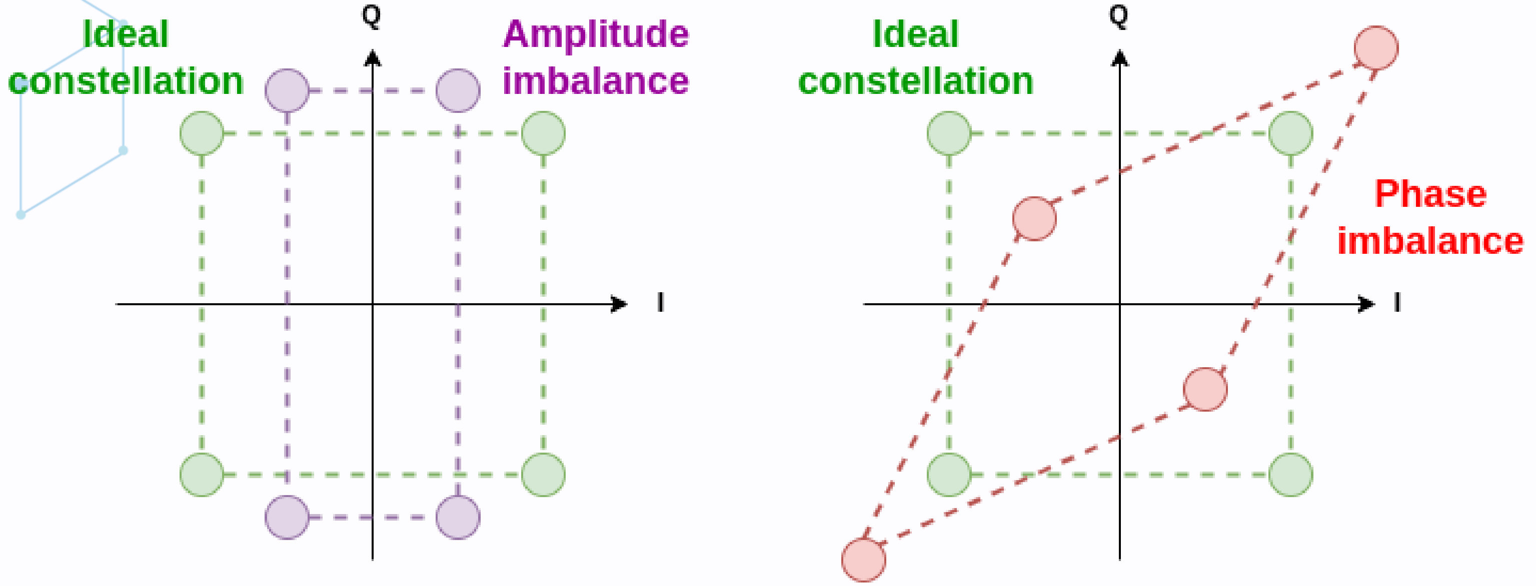
- **Interception** de `r_lld_pdu_tx_push` et **modification des paramètres radio**
- **On trouve le buffer d'émission** dans l'*Exchange Memory* et on **injecte notre paquet dans le *payload* BLE** (attaque *PIP*)
- On démarre la radio en **mode test**

Démo !

Peut-on aller plus loin ?

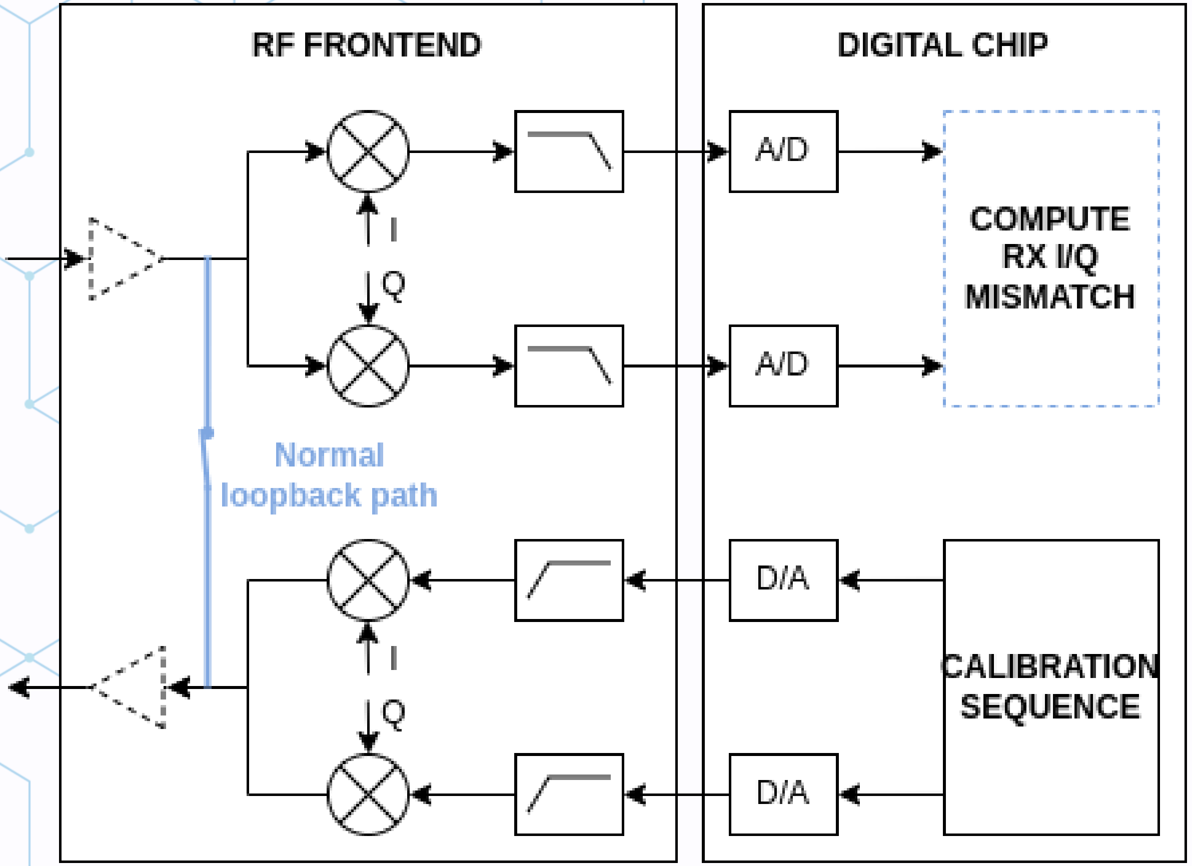
Imperfections RF

L'architecture du *transceiver* introduit des **imperfections RF**:



Incohérence entre les chemins en phase (I) et en quadrature (Q)

Processus de calibration



Les imperfections sont corrigées à l'aide d'une technique de **calibration numérique**:

Boucle entre TX et RX pour estimer et compenser l'incohérence I/Q.

→ **Comment détourner ce processus de calibration ?**

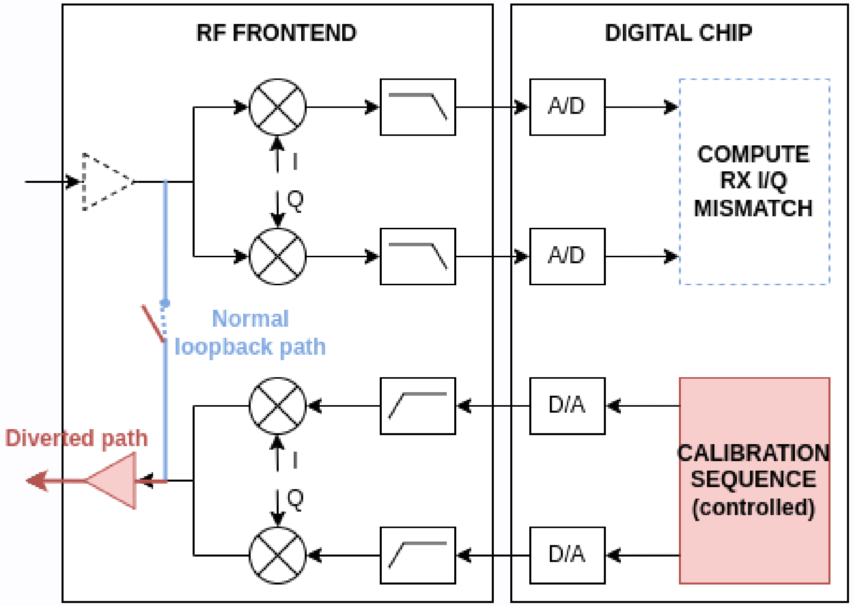
Interception des fonctions PHY

EURECOM Sophia Antipolis Quarkslab

```
                        g_phyFuns_instance
3ffae0c4 6c 2f 00 40     addr       rom_phy_disable_agc
3ffae0c8 88 2f 00 40     addr       rom_phy_enable_agc
3ffae0cc a4 2f 00 40     addr       rom_disable_agc
3ffae0d0 cc 2f 00 40     addr       rom_enable_agc
3ffae0d4 00 30 00 40     addr       rom_phy_disable_cca
3ffae0d8 2c 30 00 40     addr       rom_phy_enable_cca
3ffae0dc 44 30 00 40     addr       rom_pow_usr
3ffae0e0 3c 3e 00 40     addr       rom_gen_rx_gain_table
3ffae0e4 60 30 00 40     addr       rom_set_loopback_gain
3ffae0e8 b8 30 00 40     addr       rom_set_cal_rxdc
3ffae0ec f8 30 00 40     addr       rom_loopback_mode_en
3ffae0f0 2c 31 00 40     addr       rom_get_data_sat
3ffae0f4 a4 31 00 40     addr       rom_set_pbus_mem
3ffae0f8 8c 34 00 40     addr       rom_write_gain_mem
3ffae0fc 1c 35 00 40     addr       rom_rx_gain_force
```

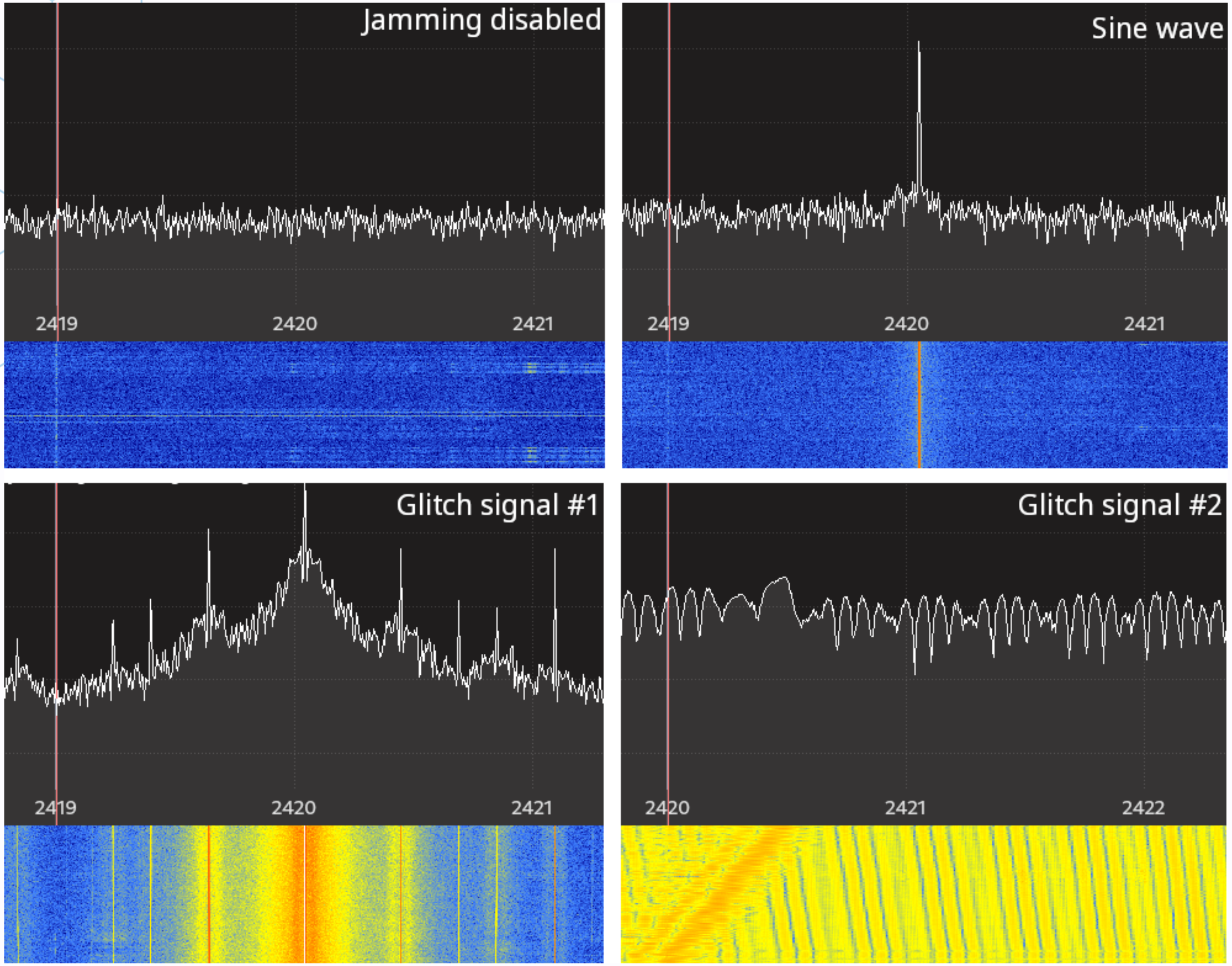
- Les fonctions PHY sont dans un **tableau de pointeurs de fonctions spécifique**: `g_phyFuns` → **stratégie d'interception**
- Fonction **d'activation du loopback**: `rom_loopback_mode_en`

Détournement de la calibration

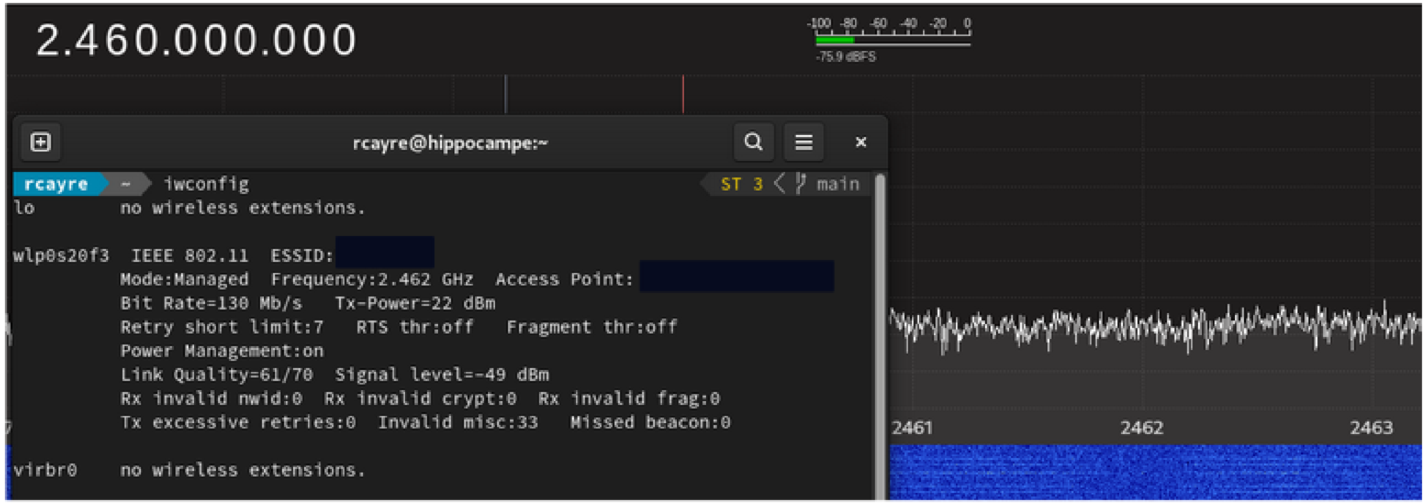


1. Désactivation du contrôle de fréquence matériel
2. Interception de `rom_loopback_mode_en` → **Boucle infinie**
3. Manipulation du signal via des fonctions bas-niveau (fréquence, gain) !

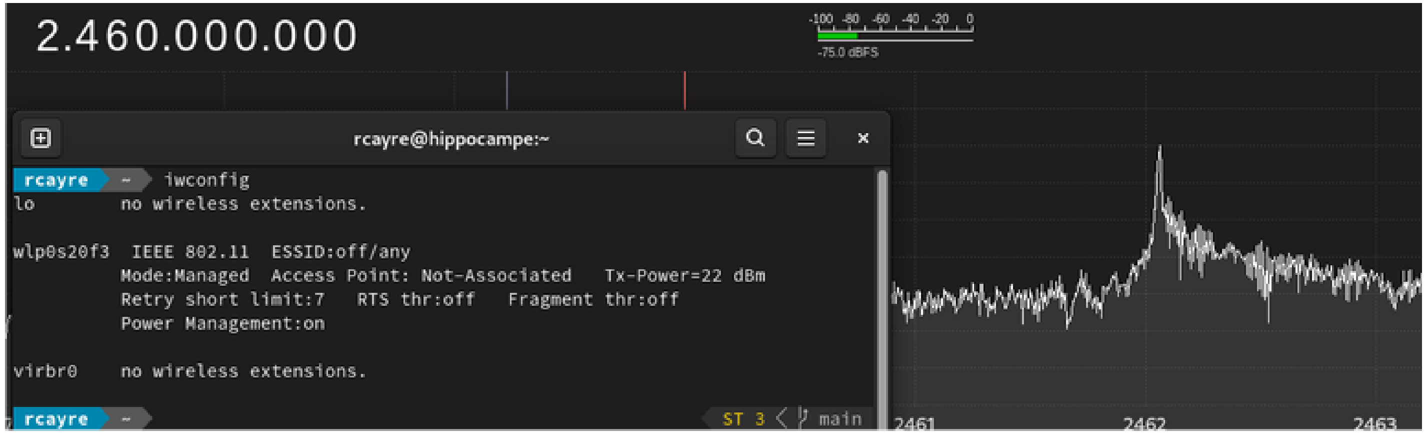
Manipulation du signal



Brouillage WiFi

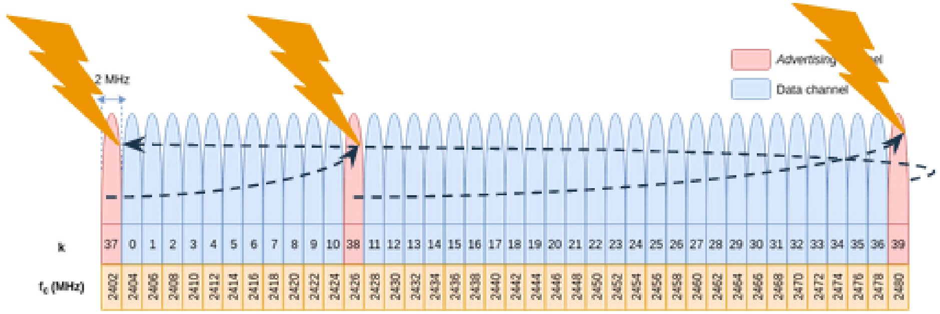


Jamming disabled



Jamming enabled

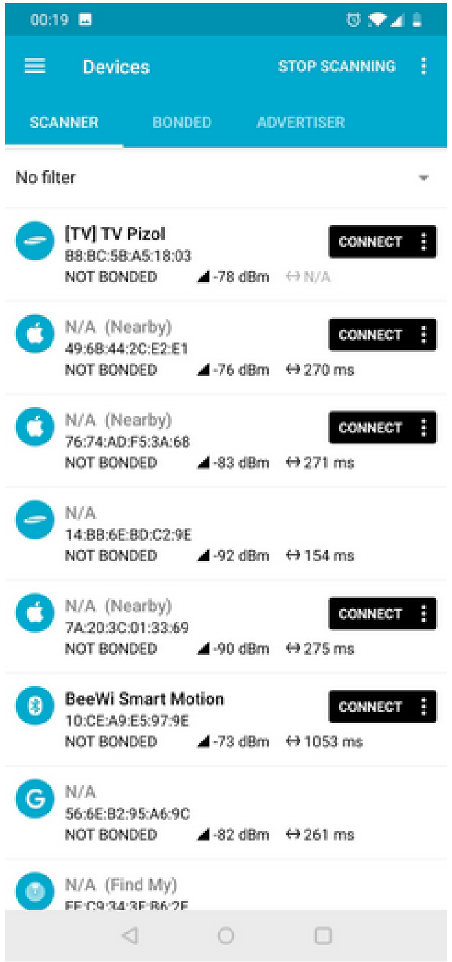
Brouillage BLE



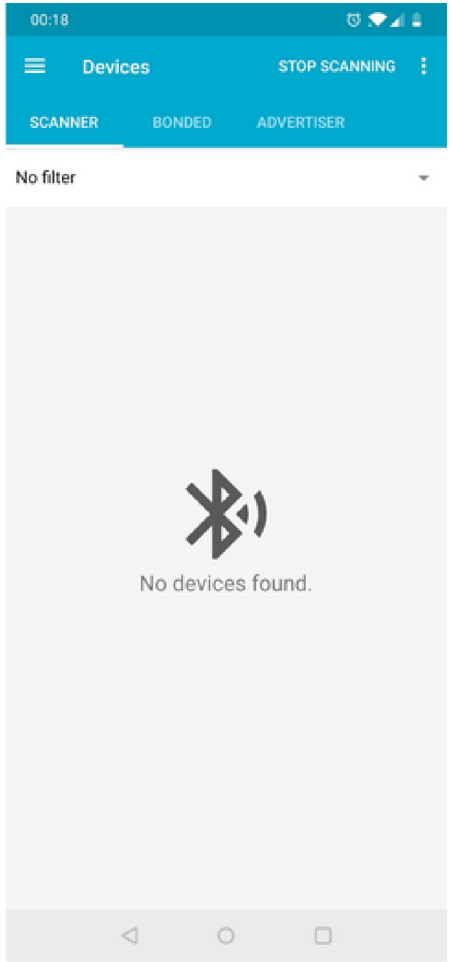

```
while (jammer) {
        // Set frequency to 2402 MHz (channel 37)
        set_chan_freq_sw_start(2,0,0);
        // Alter the parameters
        ram_start_tx_tone(1,0,10,0,0,0);

        // Set frequency to 2426 MHz (channel 38)
        set_chan_freq_sw_start(26,0,0);
        ram_start_tx_tone(1,0,10,0,0,0)

        // Set frequency to 2480 MHz (channel 39)
        set_chan_freq_sw_start(80,0,0);
        ram_start_tx_tone(1,0,10,0,0,0);
}
```



Jamming disabled



Jamming enabled

Conclusion

EURECOM Sophia Antipolis Quarkslab

- **La pile BLE de l'ESP32 peut être détournée** pour:
  - **capturer, modifier et injecter** à la volée des paquets BLE,
  - réaliser des **attaques inter-protocolaires**,
  - **brouiller des canaux multiples** et établir un **canal de communication caché**

- **Et après ?**
  - Contrôle directe de la radio: **émission / réception d'IQ** ?
  - Analyse de la **pile WiFi**

Conclusion

- **Les risques liés à la co-existence de protocoles sans-fil**:
  - L'attaquant peut exploiter des **similitudes de la couche physique**,
  - **Absence de sécurité**, **sécurité par l'obscurité,**
  - Déploiement massif d'équipements BLE ➔ **nouvelle surface d'attaque**

Questions ?

Merci de votre attention !