EURECOM Sophia Antipolis

Quarkslab

Weaponizing ESP32 RF Stacks

Romain Cayre, Damien Cauquil

Who are we ?

Romain Cayre, EURECOM

- maintainer of *Mirage*, a popular BLE swiss-army tool
- loves *cross-protocol attacks* (*Wazabee*)

Damien Cauquil, Quarkslab

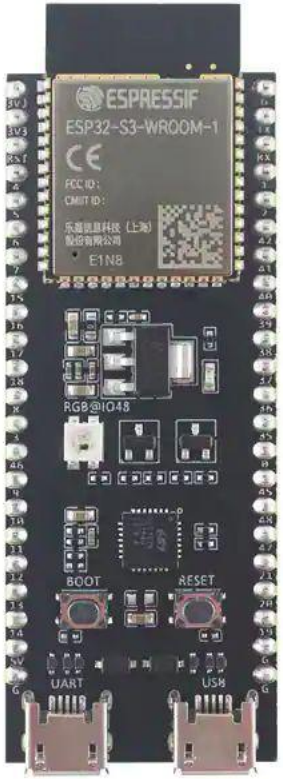
- maintainer of *Btlejack*, another BLE swiss-army tool
- loves reversing stuff, including *embedded systems*

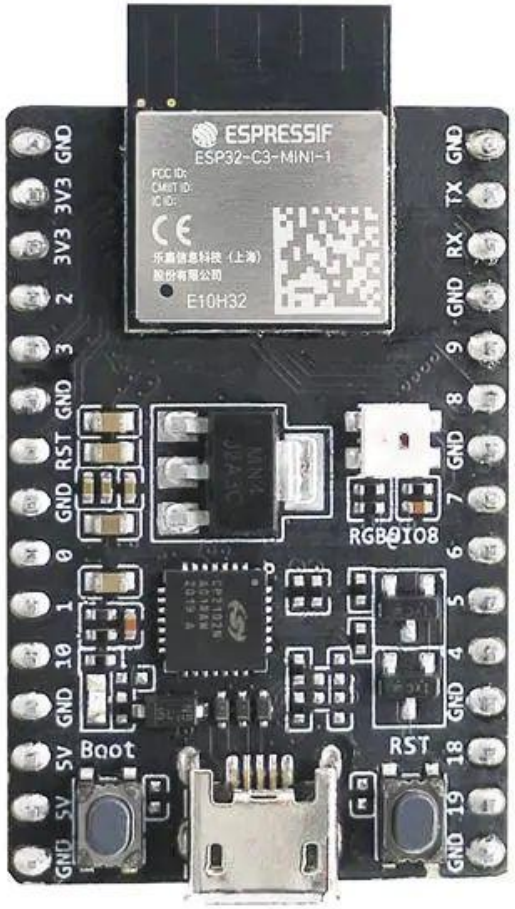
EURECOM Sophia Antipolis

Quarkslab

Introduction

Enter the ESP32 world !



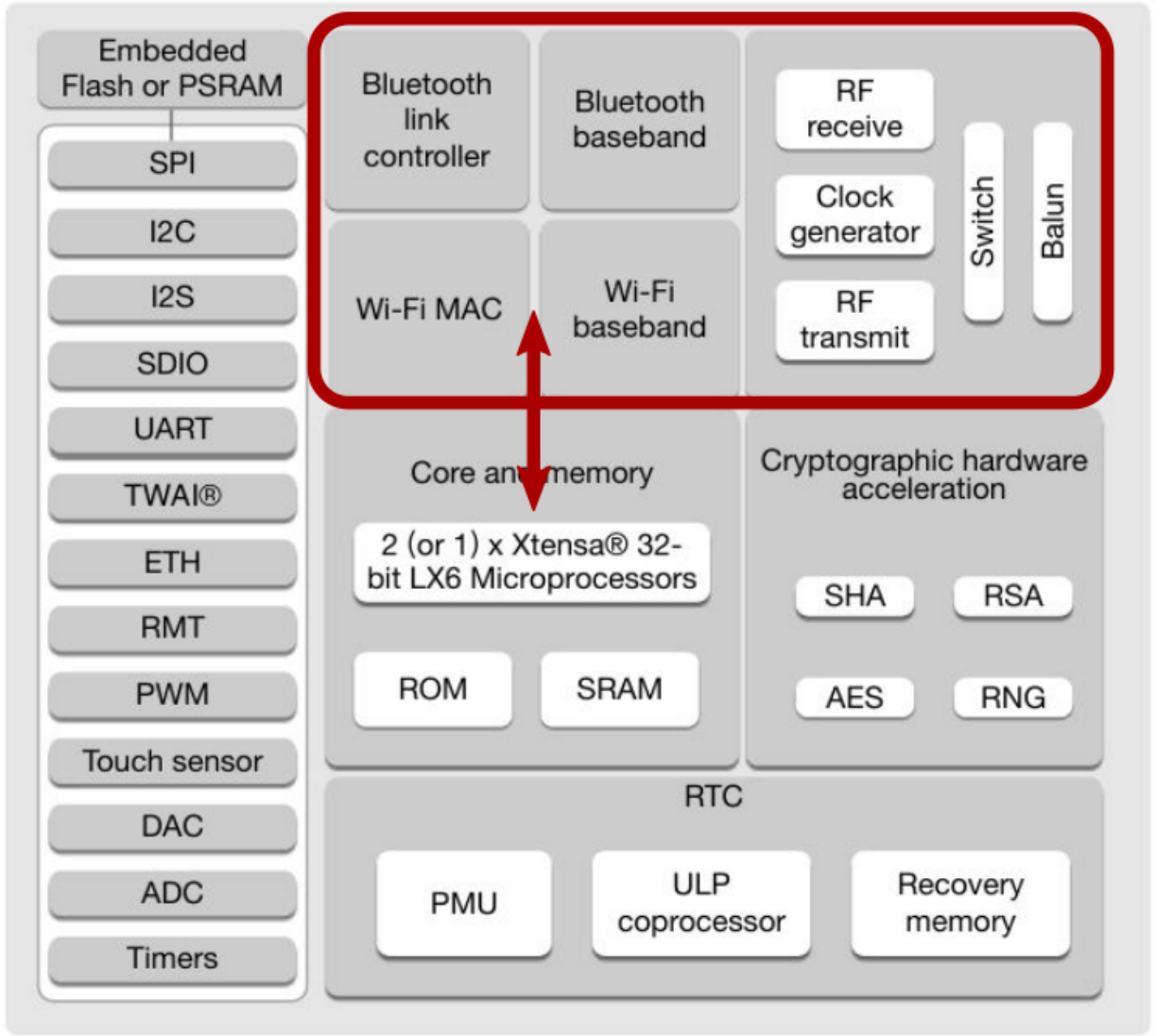


Enter the ESP32 world !

EURECOM Sophia Antipolis Quarkslab

- **Cheap** and **lightweight** SoCs
- Commonly used for **IoT devices** (and *DIY* projects)
- Provides **WiFi**, **Bluetooth Low Energy** / **Bluetooth BR/EDR** 🤩
- **Tensilica Xtensa** (ESP32, ESP32-S3) and **RISC-V** (ESP-C3)

ESP32 global architecture



Lots of questions ...

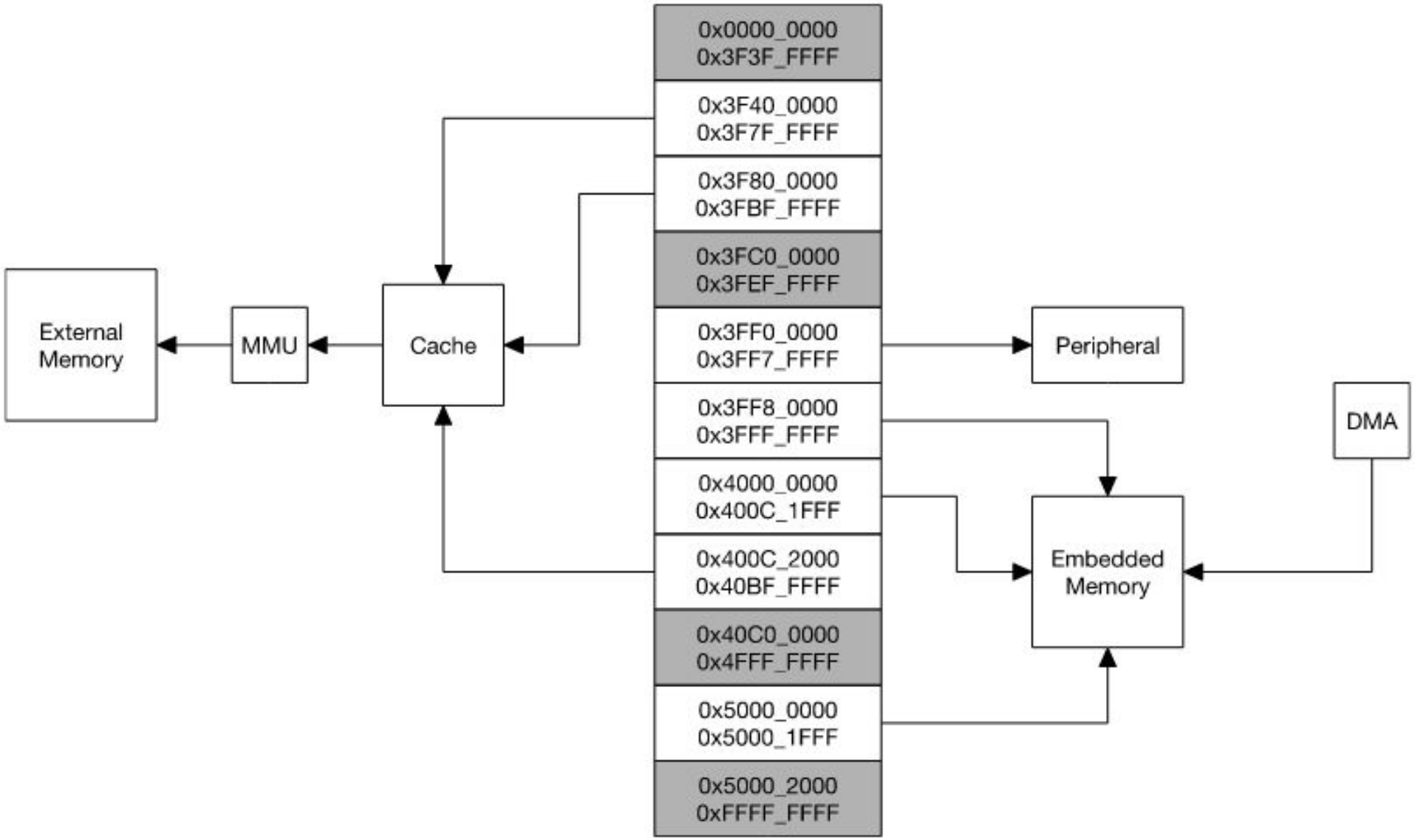
Is it possible to:

- **sniff** BLE communications ?
- **inject** an arbitrary BLE PDU ?
- **divert the radio PHY** to do *nasty* things ?
- support **other wireless protocols** ?
- turn any ESP32 into a **wireless hacking tool** ?

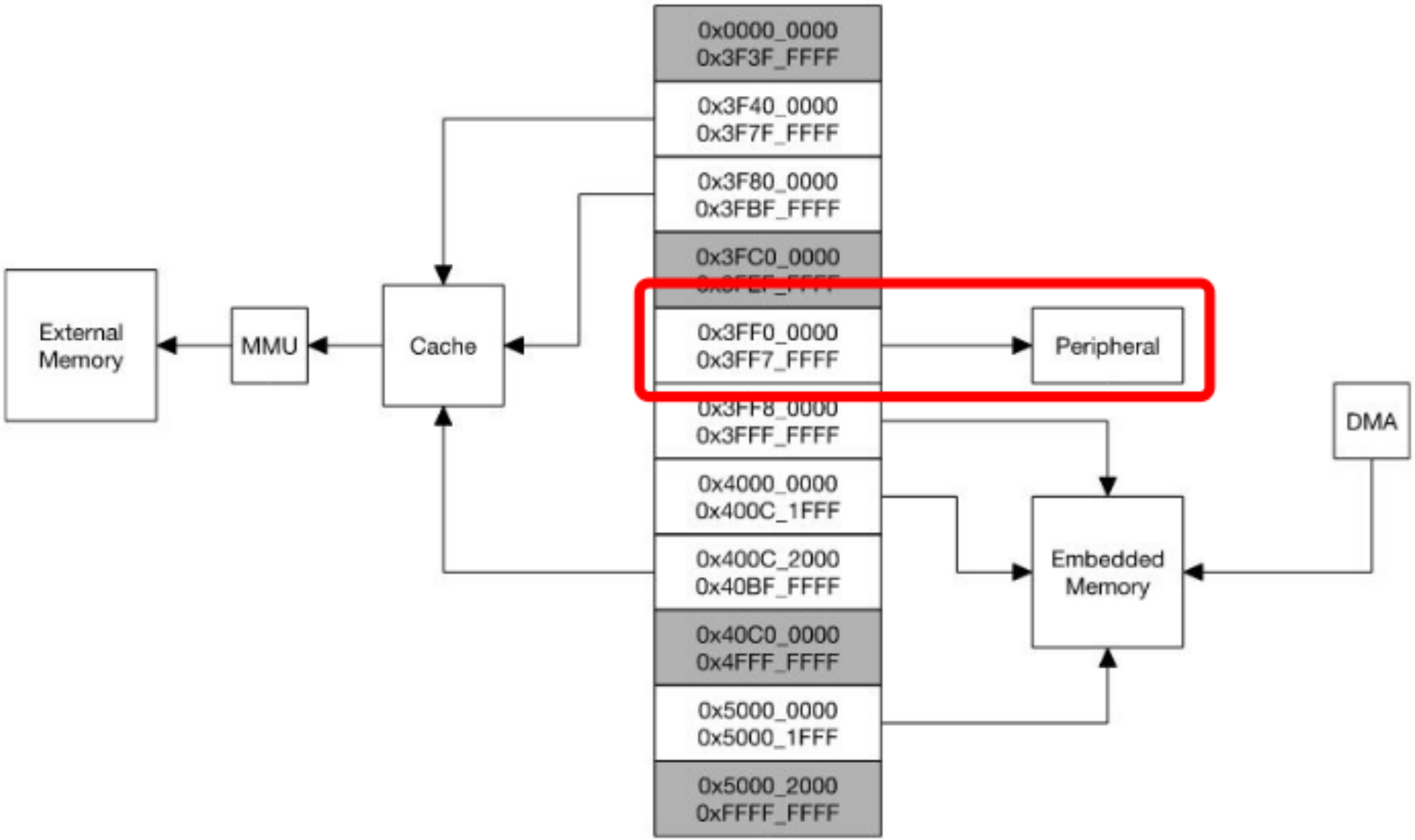
EURECOM Sophia Antipolis Quarkslab

ESP32 internals

Memory Map


0x0000_0000
0x3F3F_FFFF
0x3F40_0000
0x3F7F_FFFF
0x3F80_0000
0x3FBF_FFFF
0x3FC0_0000
0x3FEF_FFFF
0x3FF0_0000
0x3FF7_FFFF
0x3FF8_0000
0x3FFF_FFFF
0x4000_0000
0x400C_1FFF
0x400C_2000
0x40BF_FFFF
0x40C0_0000
0x4FFF_FFFF
0x5000_0000
0x5000_1FFF
0x5000_2000
0xFFFF_FFFF
External Memory
MMU
Cache
Peripheral
DMA
Embedded Memory

Memory Map


0x0000_0000
0x3F3F_FFFF
0x3F40_0000
0x3F7F_FFFF
0x3F80_0000
0x3FBF_FFFF
0x3FC0_0000
0x3FEF_FFFF
0x3FF0_0000
0x3FF7_FFFF
0x3FF8_0000
0x3FFF_FFFF
0x4000_0000
0x400C_1FFF
0x400C_2000
0x40BF_FFFF
0x40C0_0000
0x4FFF_FFFF
0x5000_0000
0x5000_1FFF
0x5000_2000
0xFFFF_FFFF
External Memory
MMU
Cache
Peripheral
DMA
Embedded Memory

Hardware peripherals

Lots of them are mentionned in the ESP32 datasheet:

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

BLE controller ?

Playing hide and seek

EURECOM Sophia Antipolis Quarkslab

- 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

Playing hide and seek

Found in a pastebin somewhere:

```
/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'
```

Playing hide and seek

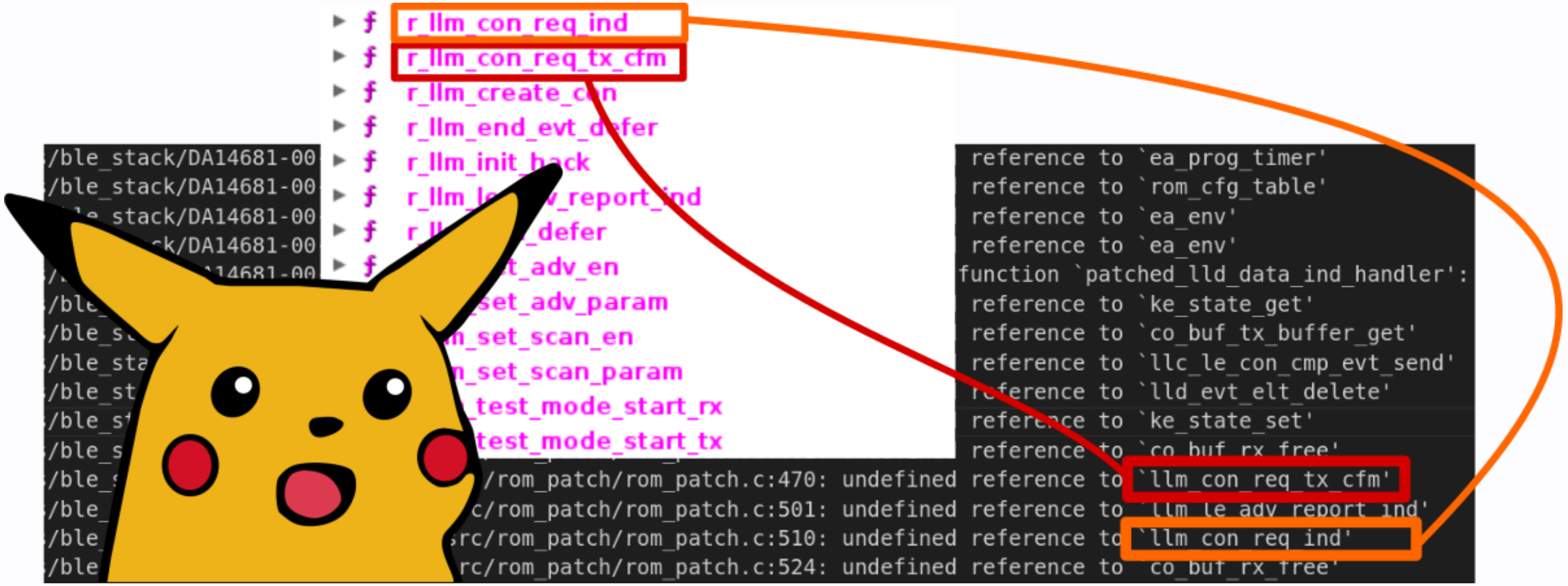


Table 92: Register map BLE

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

BLE Link-Layer initialization

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

0x8e89bed6: the advertising **access address**

ESP32 BLE controller

EURECOM Sophia Antipolis Quarkslab

- **16-bit registers**, like the DA14681
- **Same register order** in ESP32 firmware
- Dialog DA14681 is **fully documented** 😛

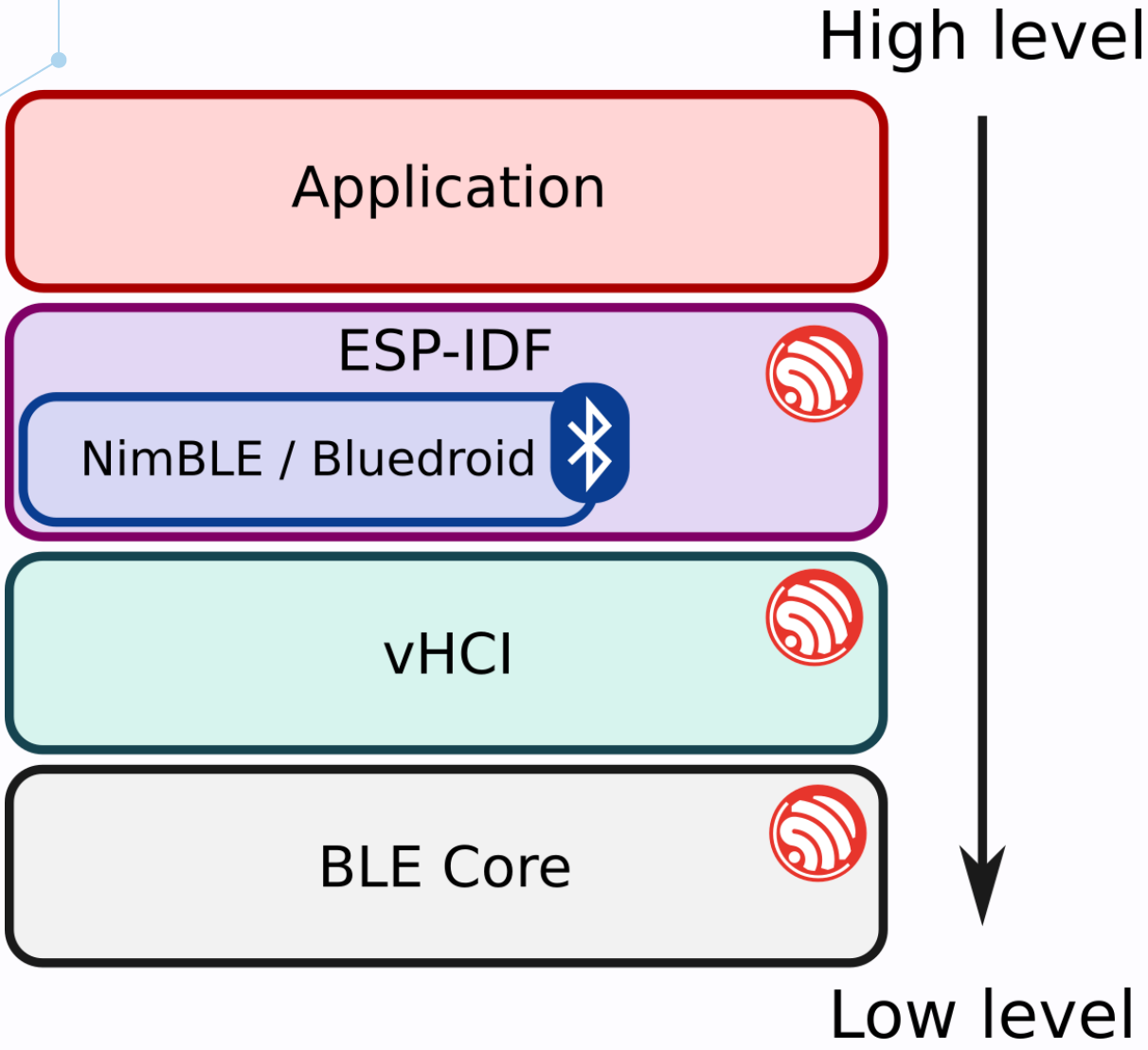
EURECOM Sophia Antipolis Quarkslab

Fun with BLE

What are we looking for ?

- BLE **sniffing**
- Packet **injection**
- **Man-in-the-Middle** attacks

BLE stack overview



BLE Core (DA14681 ?)

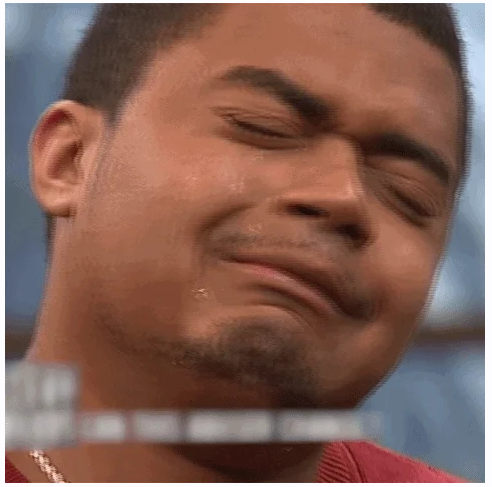
- **PHY** layer
- Handles **all RF-layer operations**
- **Peripheral** driven by a *brain*

vHCI

- **V**irtual **H**ost **C**ontroller **I**nterface
- Used by **upper BLE stacks** (NimBLE or Bluedroid)
- Standardized messages, **no sniffing nor injection** possible

ESP-IDF

- Provides a compatible implementation of **NimBLE** and **Bluedroid**
- Shipped as **static libraries**
- Can be updated when *ESP-IDF* is updated
- **Too dependent** of a specific *ESP-IDF* version



There is maybe a place ...

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

ESP32 Internal ROMs

EURECOM Sophia Antipolis Quarkslab

- 2 **specific ROM** regions
- These regions contain some **code and data**
- **Low-level API functions** to drive the BLE core 🥳
- **New problem**: how to hook these functions? 🤔

Hooking ROM functions

- ROM functions are called through `r_ip_funcs_p`
- `r_ip_funcs_p` is a **table of function pointers in 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
```

Hacked functions (legit)

```
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;
}
```

Let's hook !

```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;
}
```

PDU sniffing

Two main functions to hook:

- `r_lld_pdu_rx_handler()` : called when a **PDU is received**
- `r_lld_pdu_data_tx_push()` : used to **send a PDU**

Sniffing possible, but only for connections related to our ESP32

PDU injection

- Use `r_lld_pdu_data_tx_push()` to send PDU
- Can send **data** and **control PDU** ! 🤣
- Needs some wrapping to work properly

Man-in-the-Middle

- We need a **deep control** over the BLE core
- Or at least to handle **two different connections** at the same time
- **Seems impossible** to achieve with an ESP32

Some cool hacks 👌

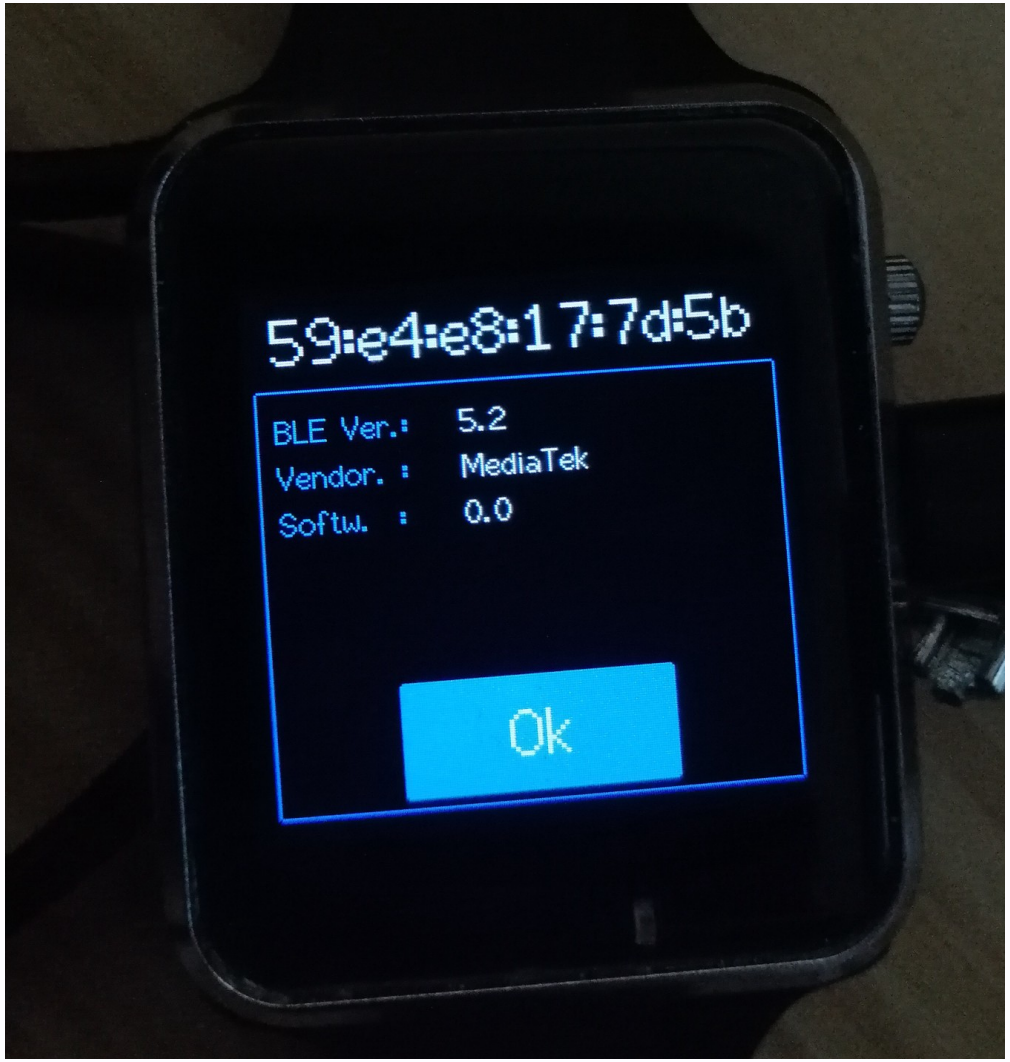
LL_VERSION_IND injection

EURECOM Sophia Antipolis Quarkslab

```
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
            )
        )
```

LL_VERSION_IND injection

Remote BLE stack fingerprinting !



Hacking the physical layer

Cross-protocol attacks

Can ESP32 radio be diverted to interact with other protocols ?

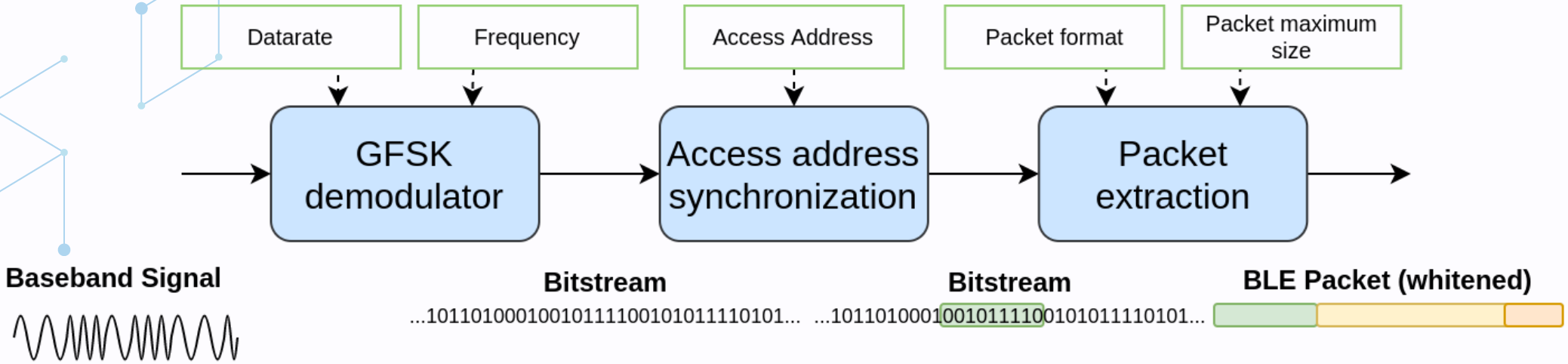
- BLE uses **Gaussian Frequency Shift Keying** (GFSK) modulation…
- … like dozens of proprietary protocols in the same band ! (ANT+ / ANT-FS, Riitek, MosArt, Logitech Unifying, Microsoft…)
- **WazaBee:** equivalence between O-QSPK (802.15.4) and 2Mbps GFSK (BLE 2M) → **ESP32-S3 / ESP32-C3 only**

Cross-protocol attacks

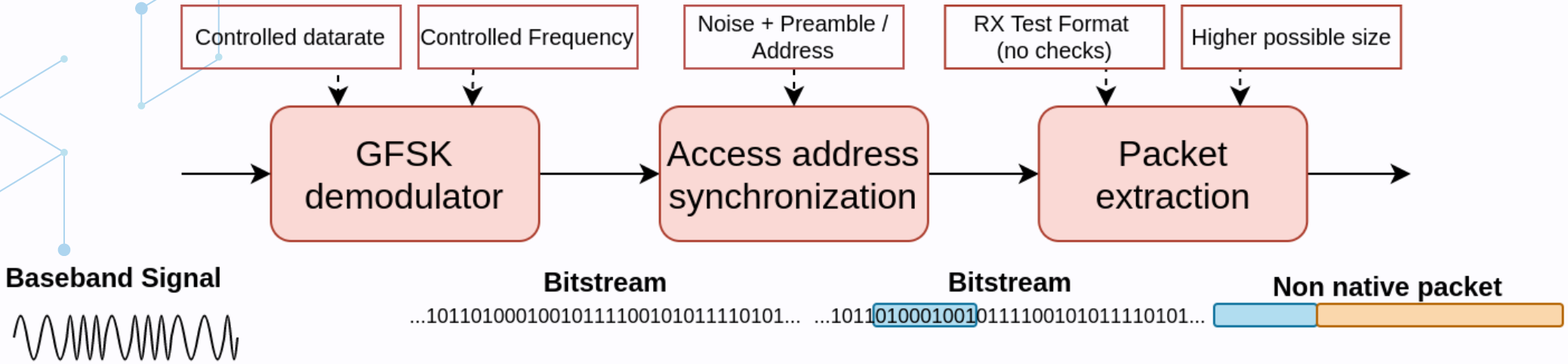
We need to control **low level radio parameters**:

- CRC verification
- frequency
- datarate
- synchronization word
- whitening / dewhitening
- input and output bitstreams

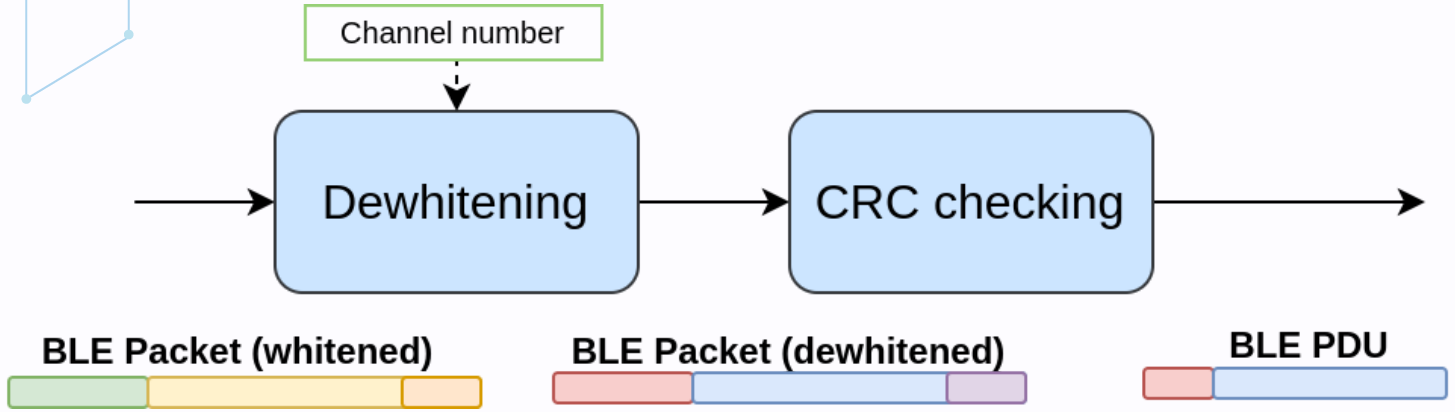
BLE RX synchronization



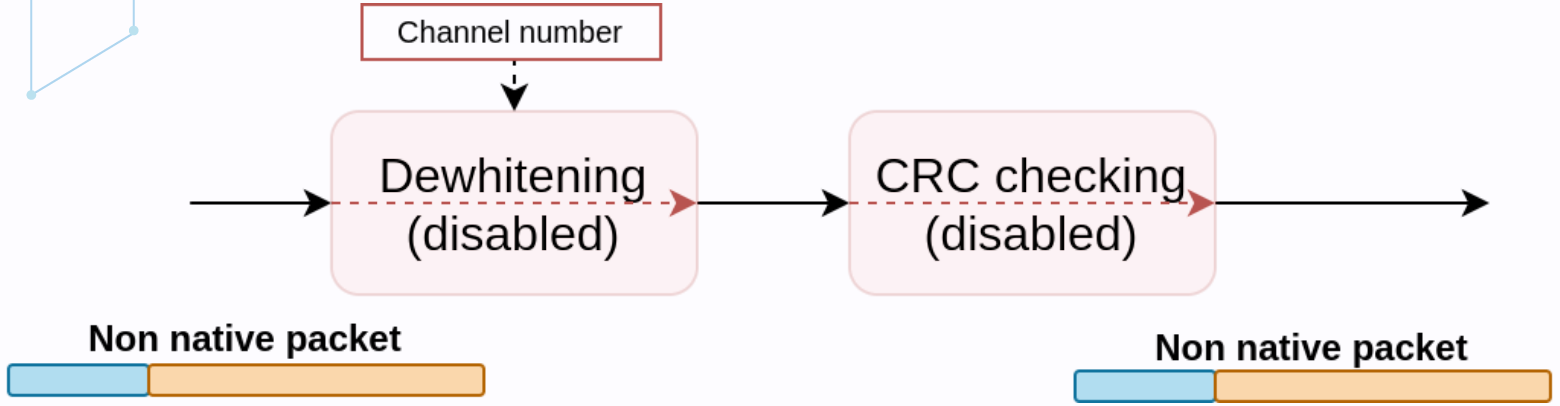
Hacking synchronization



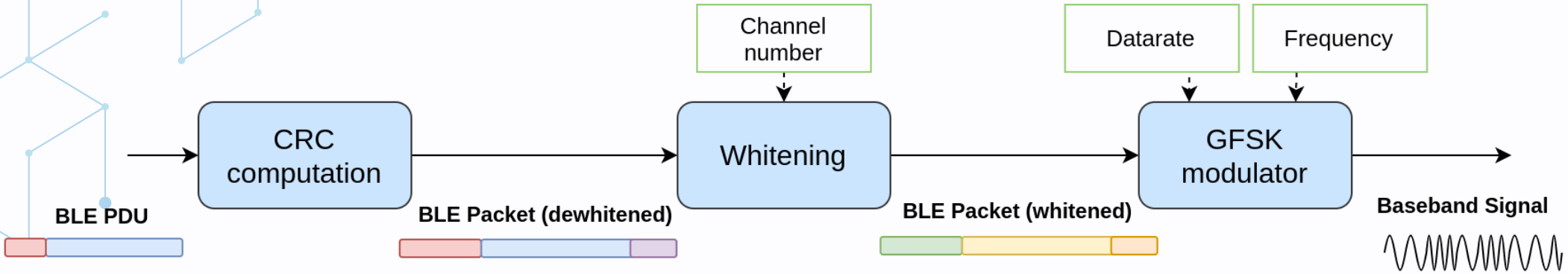
Packet Processing



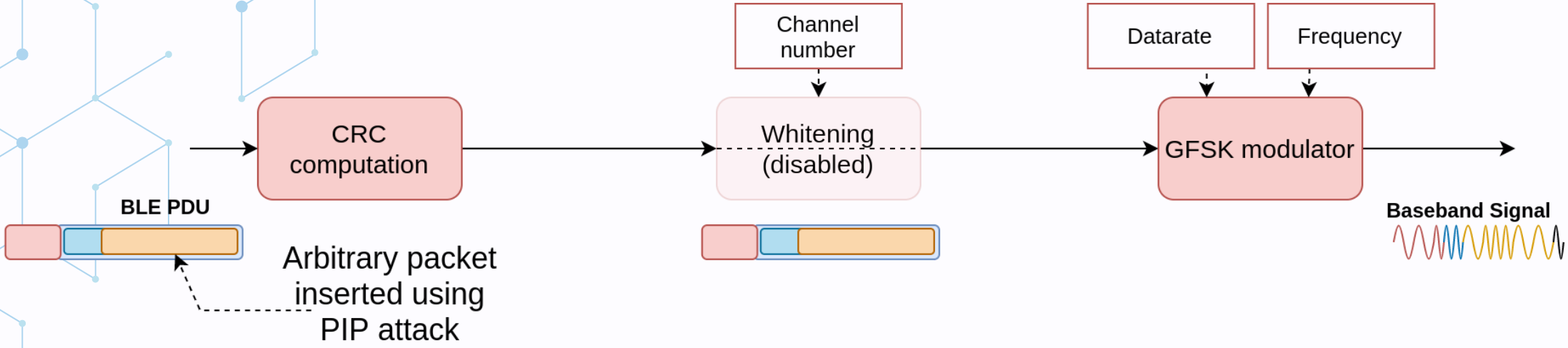
Bypassing Packet Processing



BLE Packet Transmission

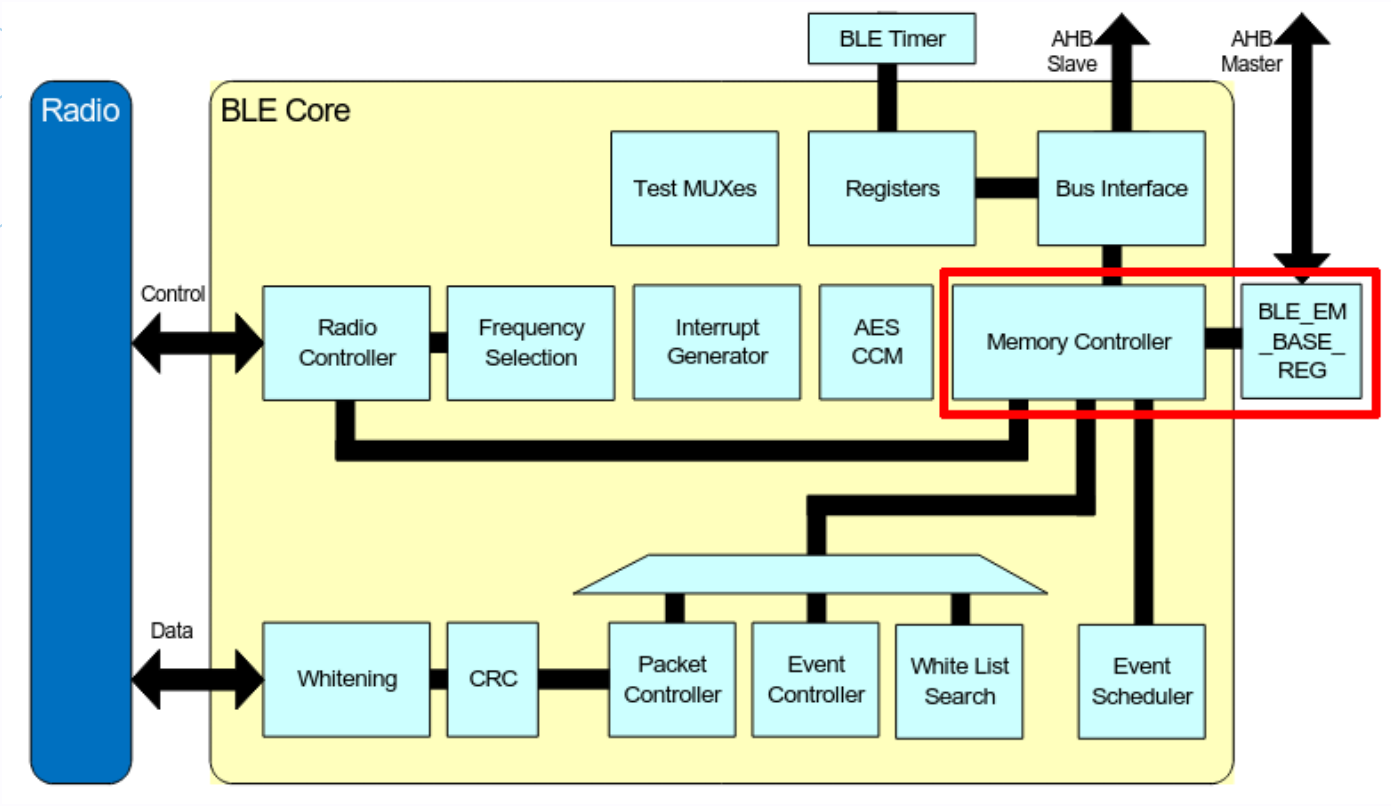


Packet in Packet

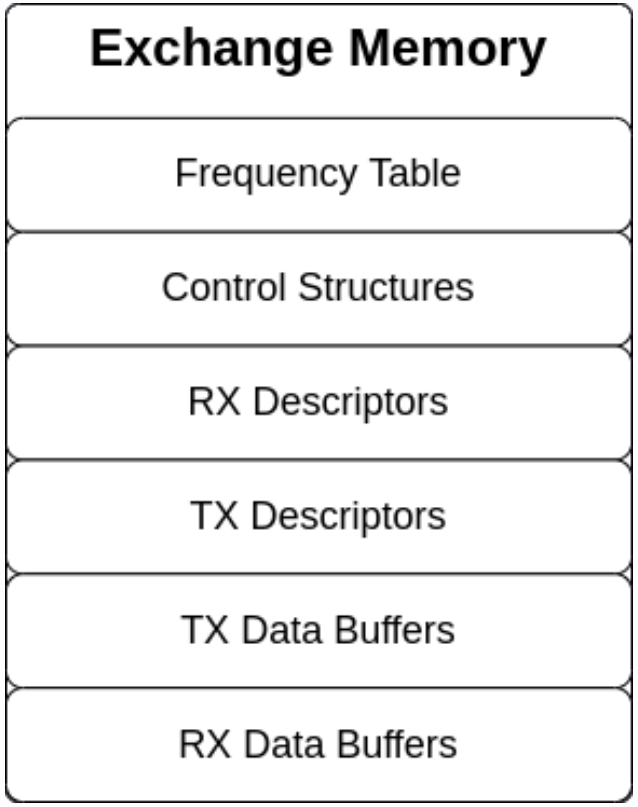


Let's dive deeper

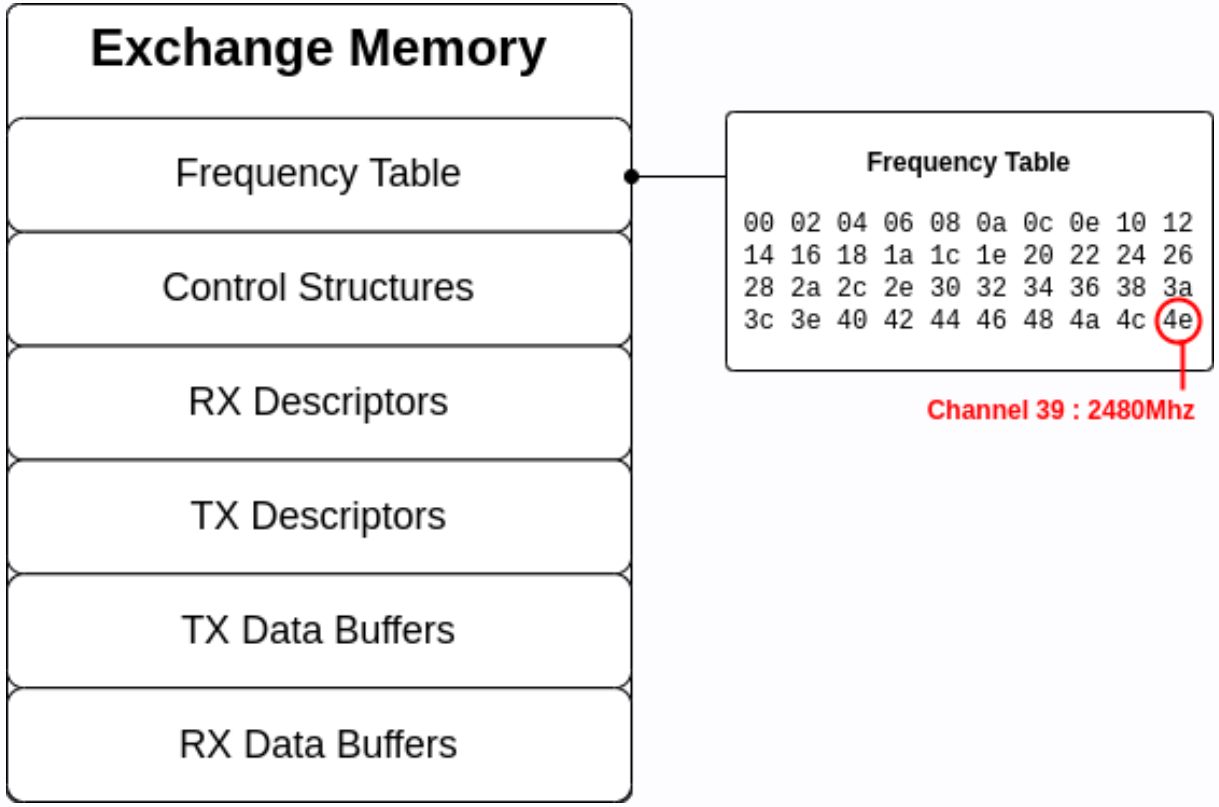
BLE Core is programmed using a shared memory zone named *Exchange Memory*.



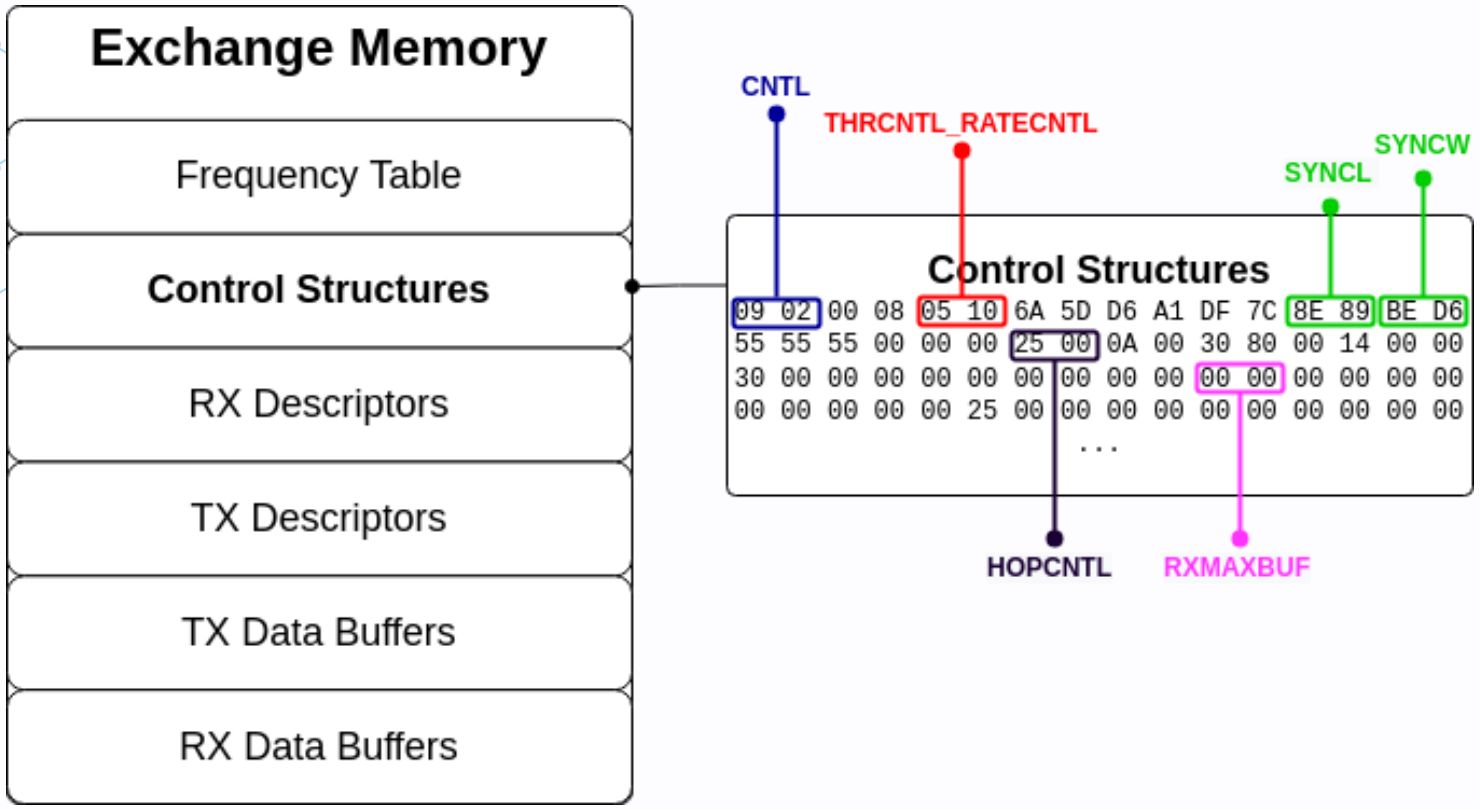
Exchange Memory



Frequency table



Control structures



BLE registers

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

Arbitrary reception primitive

Hook `r_llm_start_scan_en()` and **modify RF parameters**:

- **Frequency Table:** alter frequency of channel 39
- **Control Structure:** force channel 39, disable channel hopping, configure synchronization word, datarate and test format
- **Global Registers:** disable whitening and CRC

Reuse `r_lld_pdu_rx_handler()` hook to **extract packets**

Arbitrary transmission primitive

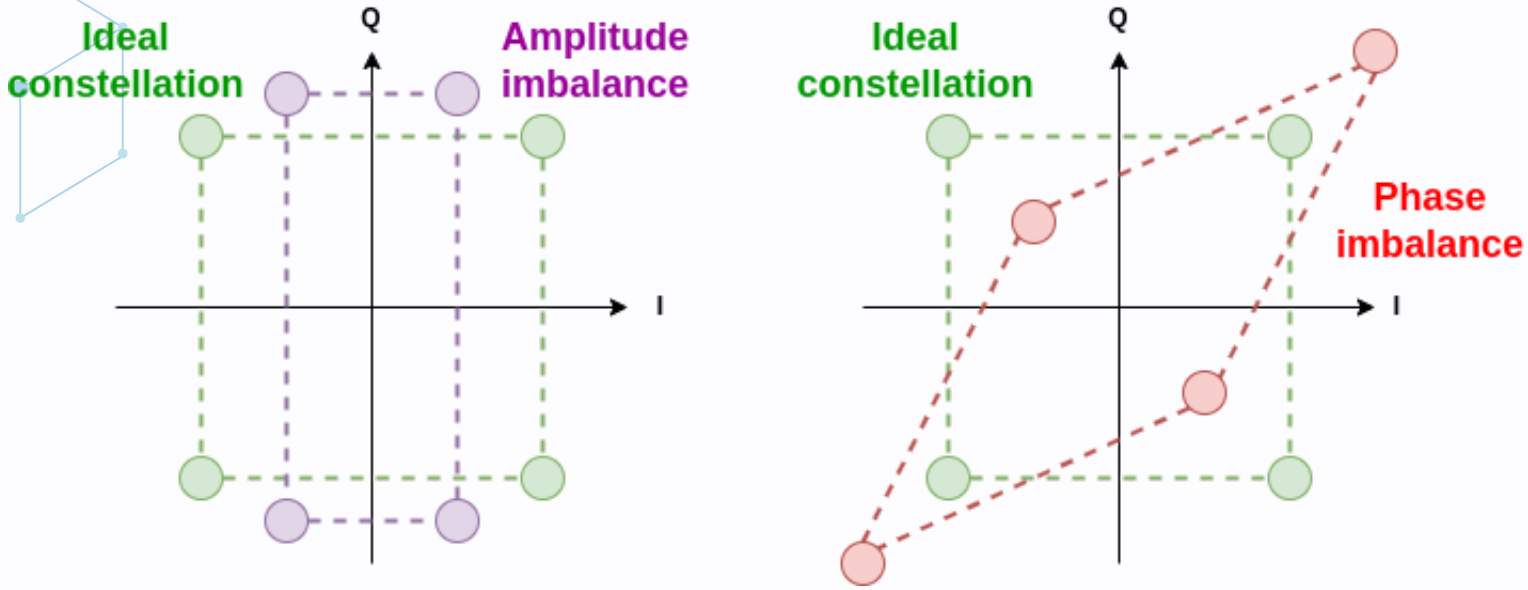
- **Hook** `r_lld_pdu_tx_push` and **modify RF parameters**
- **Find the TX buffer** in EM and write a packet (PIP attack)
- Start radio in **TX test mode**

Demo time !

Can we go *deeper* ?

RF imperfections

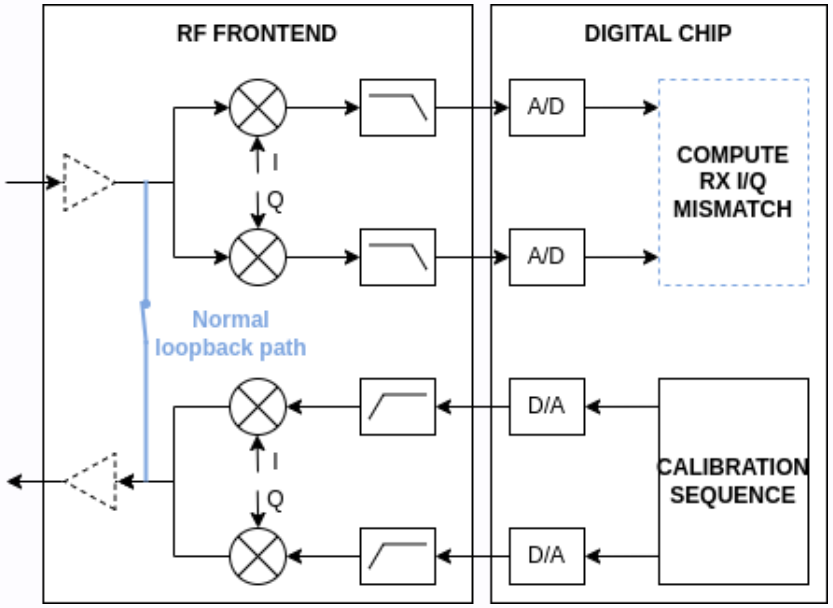
RF imperfections related to transceiver architecture



Mismatch between In-phase (I) and In-quadrature (Q) paths

Calibration process

Imperfections corrected using **digital calibration technique**:



Loopback between TX and RX path to estimate and compensate I/Q mismatch.

Forcing a full calibration

Calibration data stored in **Non Volatile Memory (NVM)**.

Full calibration only triggered if **calibration data can't be found** !

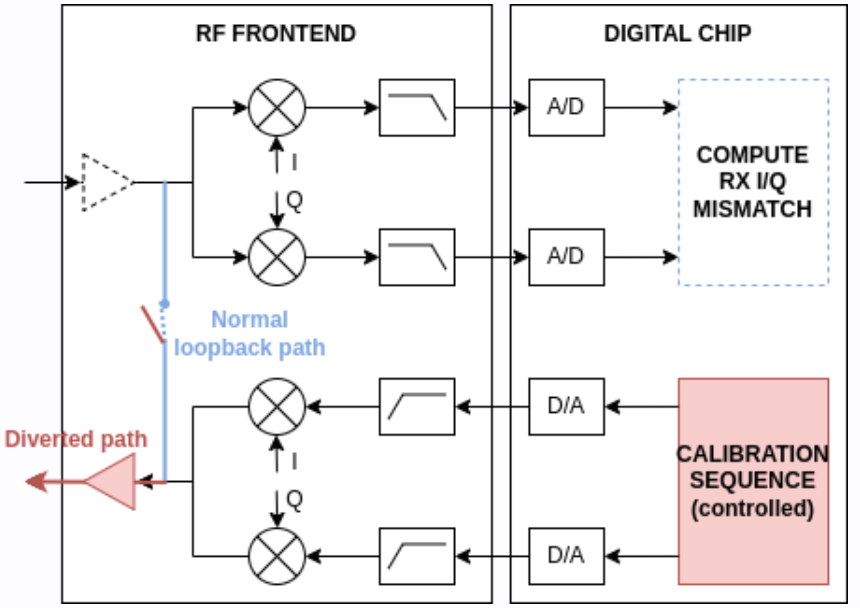
```
// Disable both Bluetooth and Radio Module
esp_bt_controller_shutdown();

// Flush calibration data
esp_phy_erase_cal_data_in_nvs();

// Enabling Radio Module to force a new calibration
esp_phy_enable();
```

Hooking PHY functions

```
                    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
```

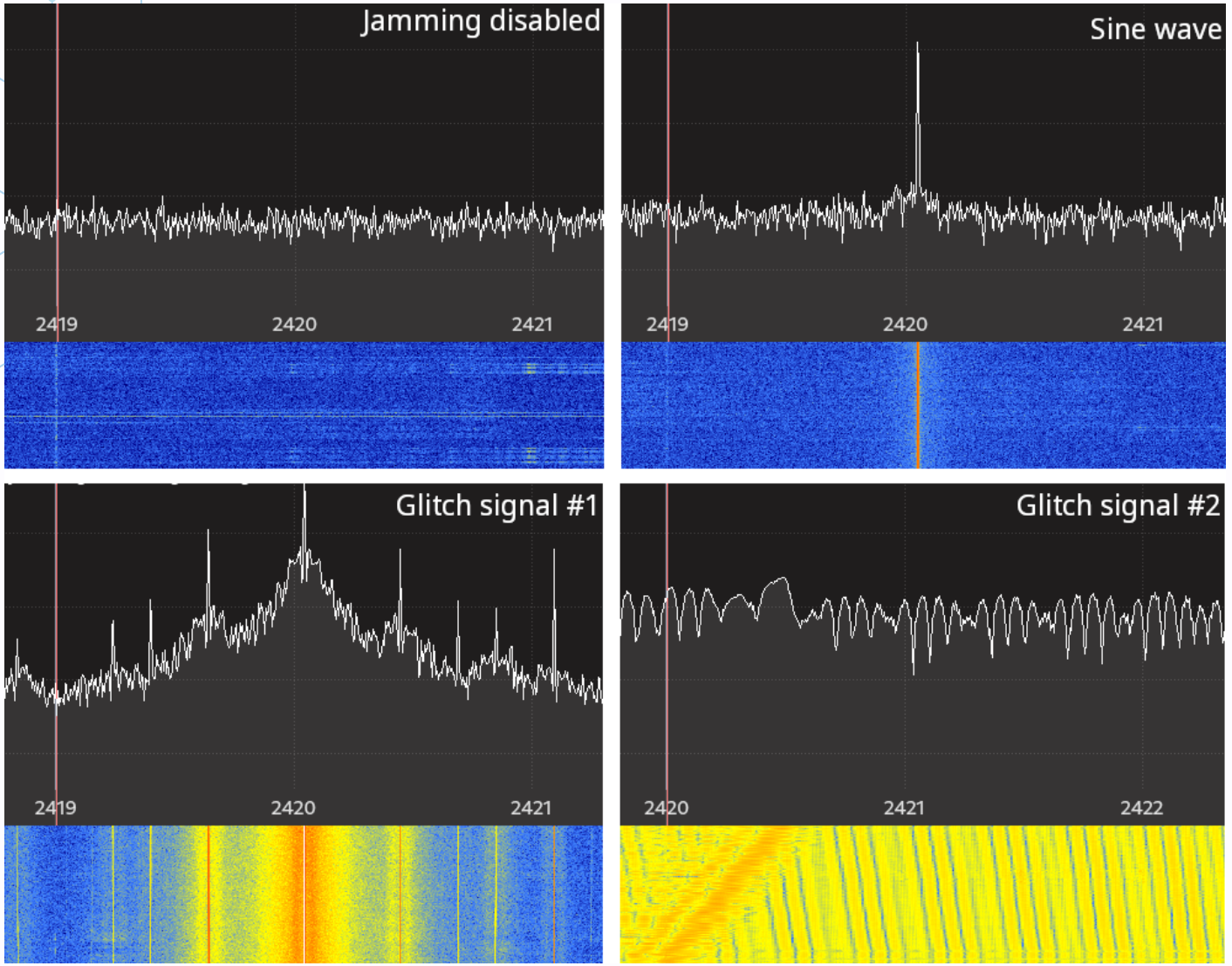
PHY functions stored in a specific function pointers array: `g_phyFuns` (pointer returned by `phy_get_romfuncs()`)

Diverting calibration process

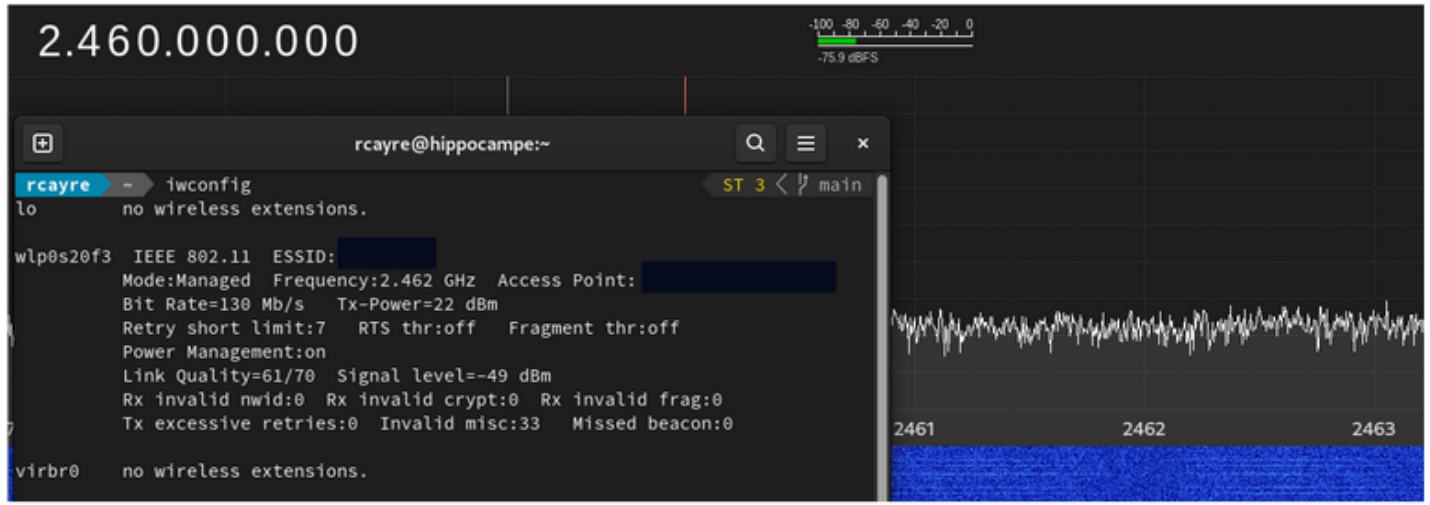


1. Disable HW frequency control (`phy_dis_hw_set_freq`)
2. Infinite loop when `rom_loopback_mode_en` is called
3. Have fun with signal control functions (frequency, gain) !

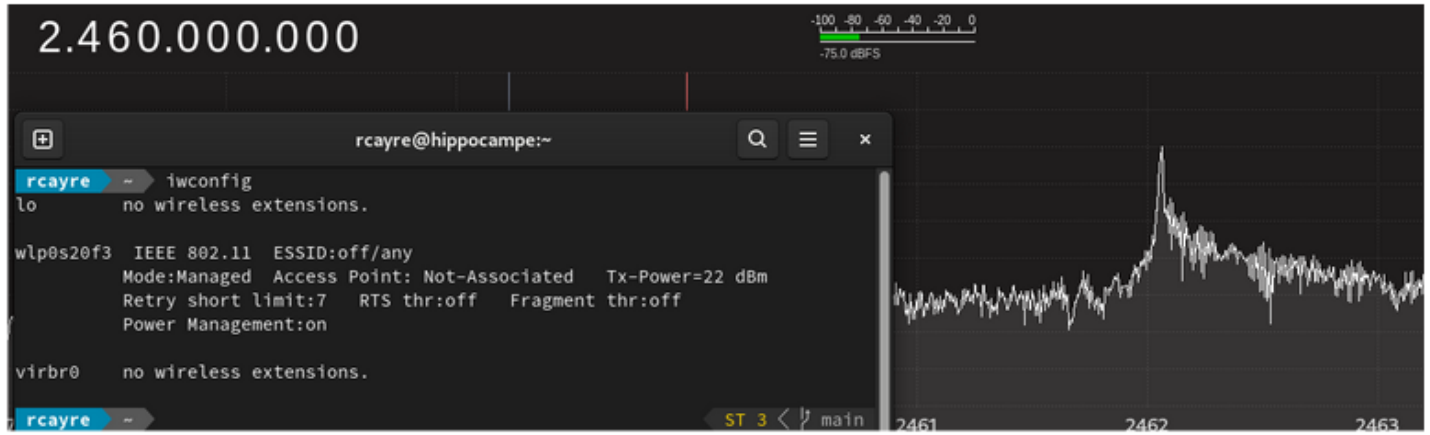
Fun with signals



WiFi Jamming

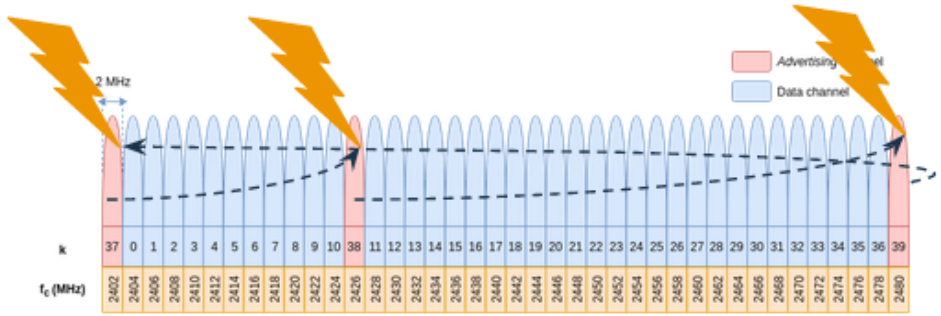


Jamming disabled

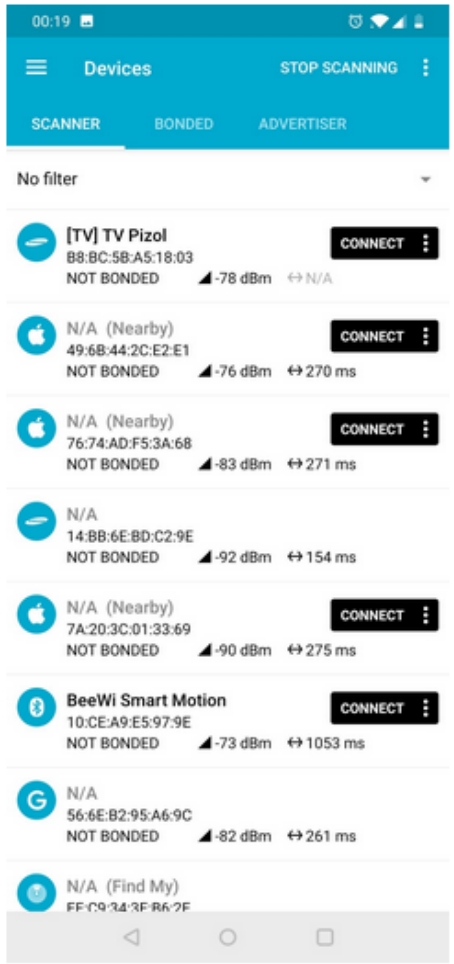


Jamming enabled

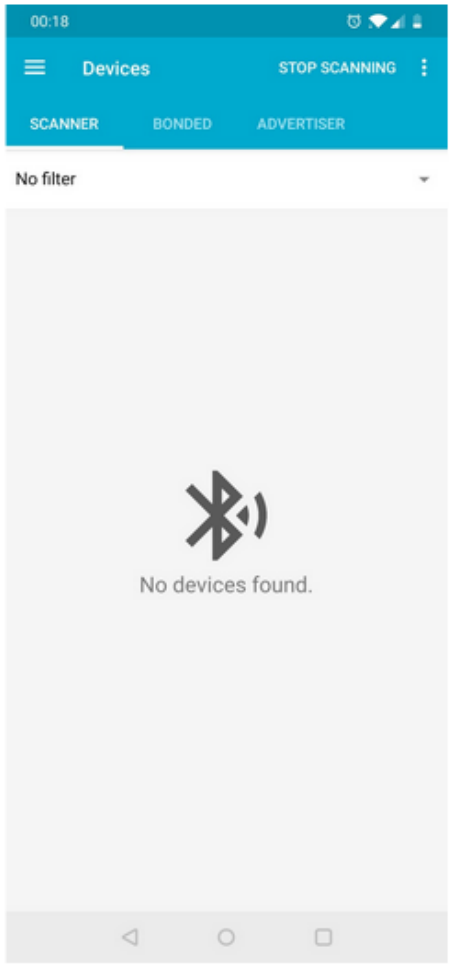
BLE Jamming




```
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

BLE Jamming - demo

Conclusion

Takeaways

- on the fly BLE PDU monitoring, modification & injection
- cross-protocol reception and transmission
- multi channel jamming / covert channel

What's next ?

- Direct control over RF: raw IQ reception / transmission ?
- Replacing NimBLE / Bluedroid: vHCI interface only

EURECOM Sophia Antipolis

Quarkslab

Q/A time

EURECOM Sophia Antipolis

Quarkslab

Thank you !