black hat
ARSENAL 2022
DECEMBER 7–8
EXCEL LONDON / UK

RFQuack

A Versatile, Modular, RF Security Toolkit

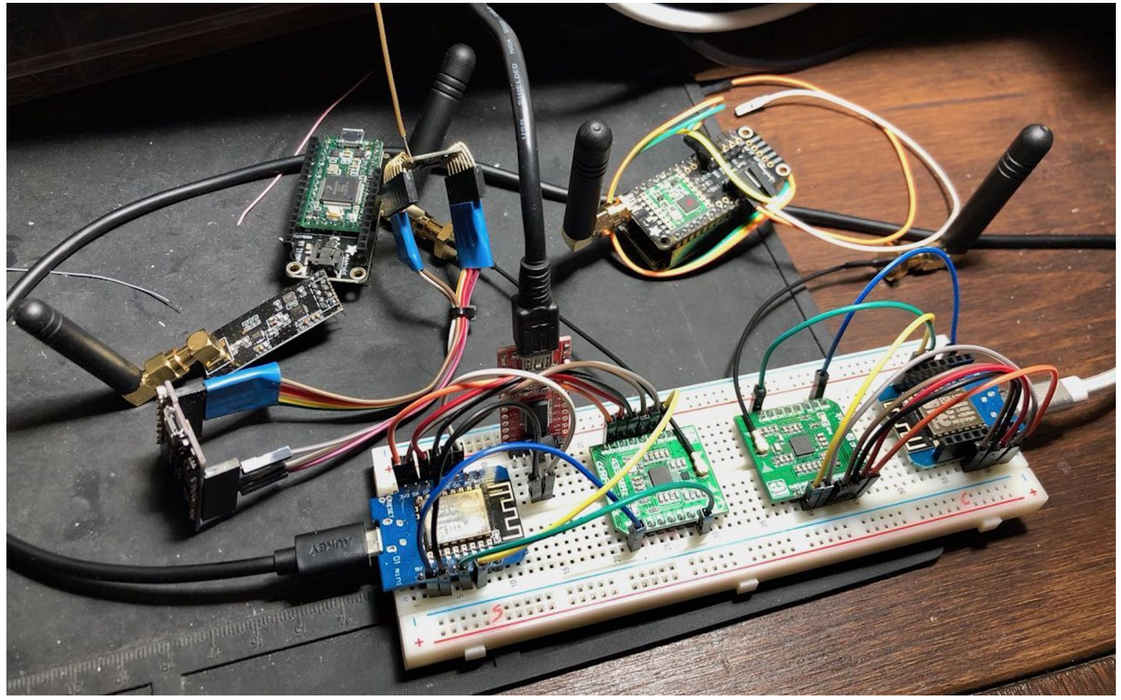
Federico Maggi
federico@maggi.cc
@phretor@infosec.exchange

Pointers

- Source code
  - https://github.com/rfquack/RFQuack
- Documentation
  - https://rfquack.org
- Demo code and instructions
  - https://github.com/rfquack/RFQuack/tree/master/examples/demos

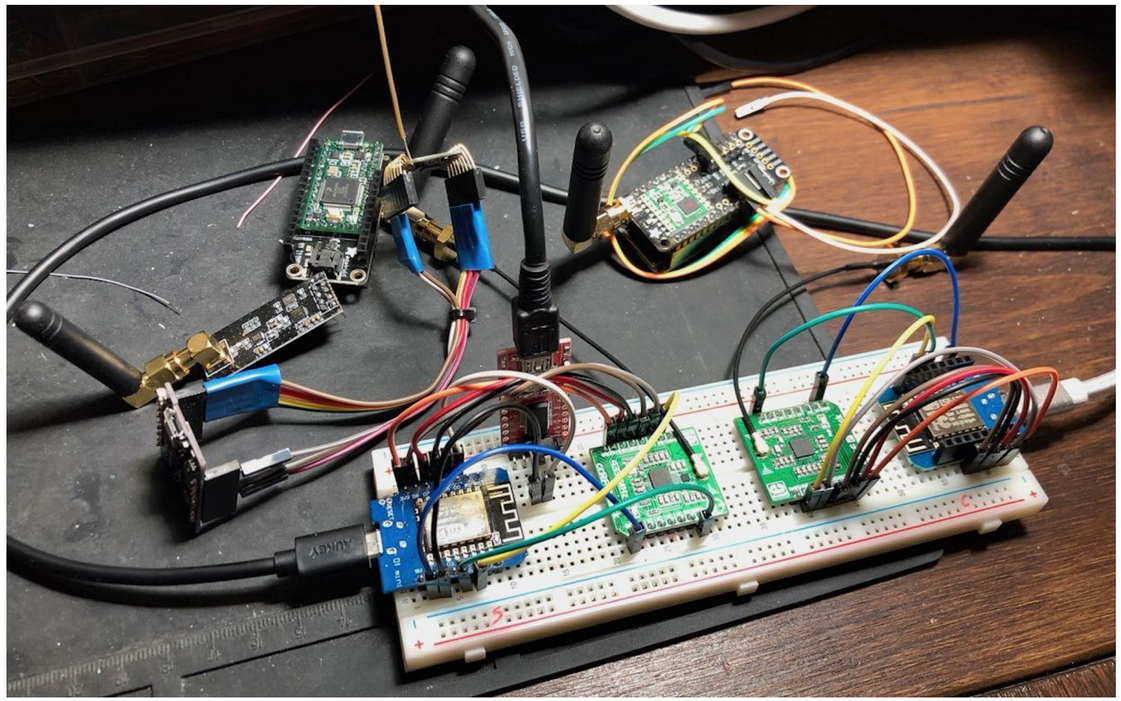
It's been a while...

2019

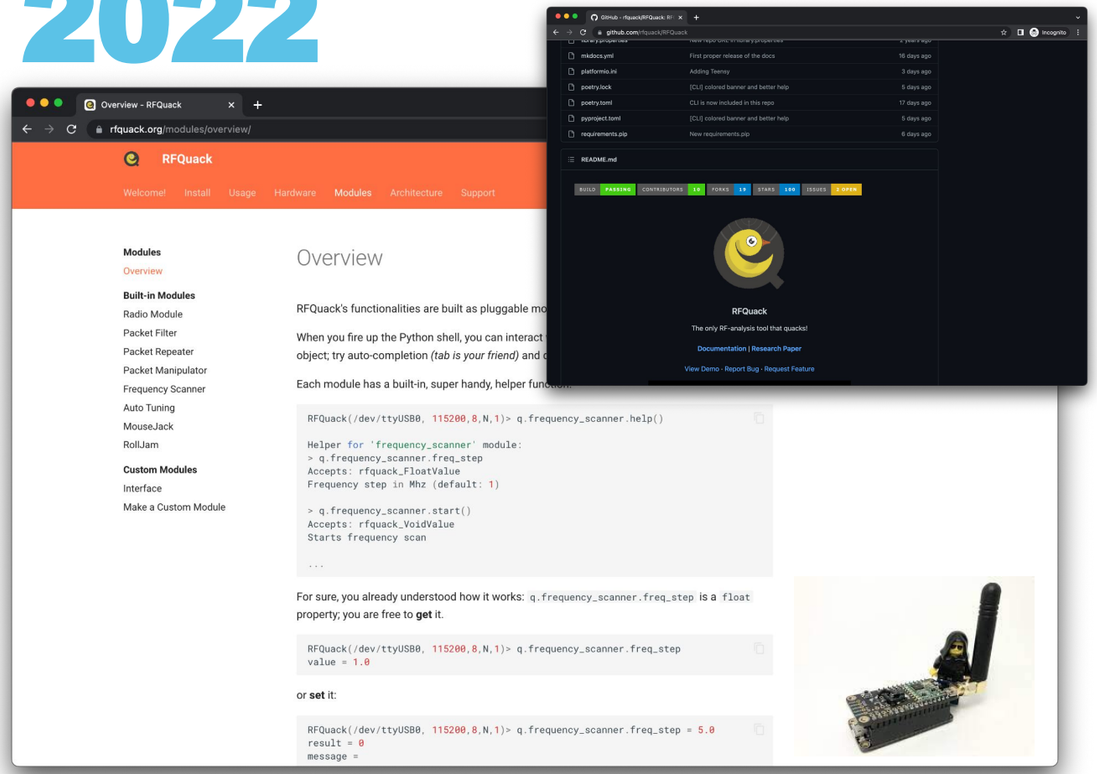


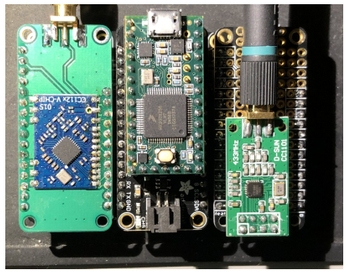
It's been a while...

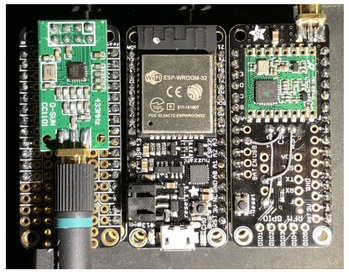
2019

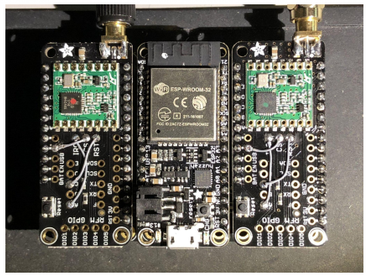


2022









Thanks!

🙏 This work wouldn't have been possible without the initial **support** of my previous employer, **Trend Micro Research**, who welcomed my proposal to **open source** a PoC I used during a project.

That PoC evolved into RFQuack.

I also want to thank the **students** who passionately worked on this project, especially **Andrea Guglielmini**.

Finally, the small but growing **community** is keeping me motivated and provide early testing: Thanks! ❤️

Come join us!

https://rfquack.org/support/community/

~~Quack~~ Quick Agenda

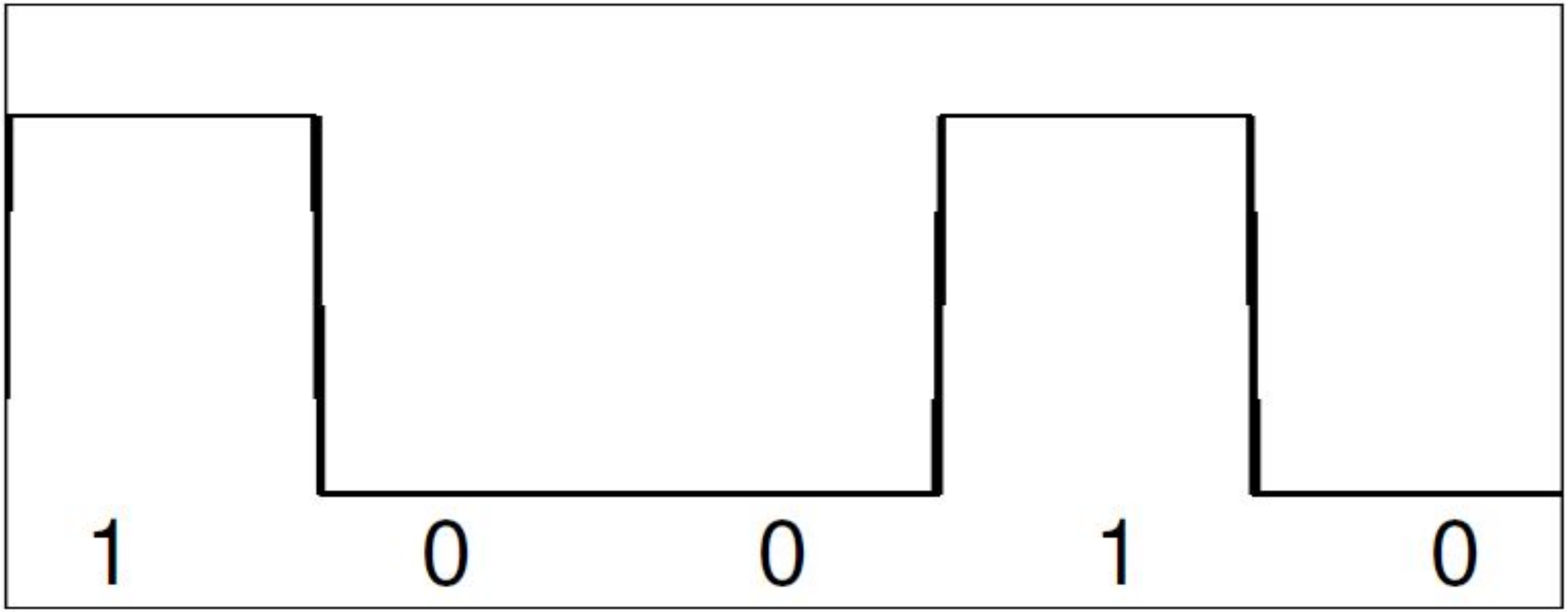
- **Context:** RF analysis 101
  - **DEMO 1:** what RFQuack can do to ease RF analysis
- **Approach:** SDRs vs. RF dongles vs. RFQuack
  - **DEMO 2:** easy firmware customization
- **Internals:** the life of an RF packet, from the air to your Python objects
- **Modules:** hackable firmware with a humanized interface
  - **DEMO 3:** scripting attacks through a micro-iptables inside a chip
- **Beyond RF:** poking SPI registers from the comfort of your terminal
- **Resources:** docs, demo videos
- **Community & Roadmap:** hardware prizes 🥇 for PR contributors!

Radio Frequency Analysis 101

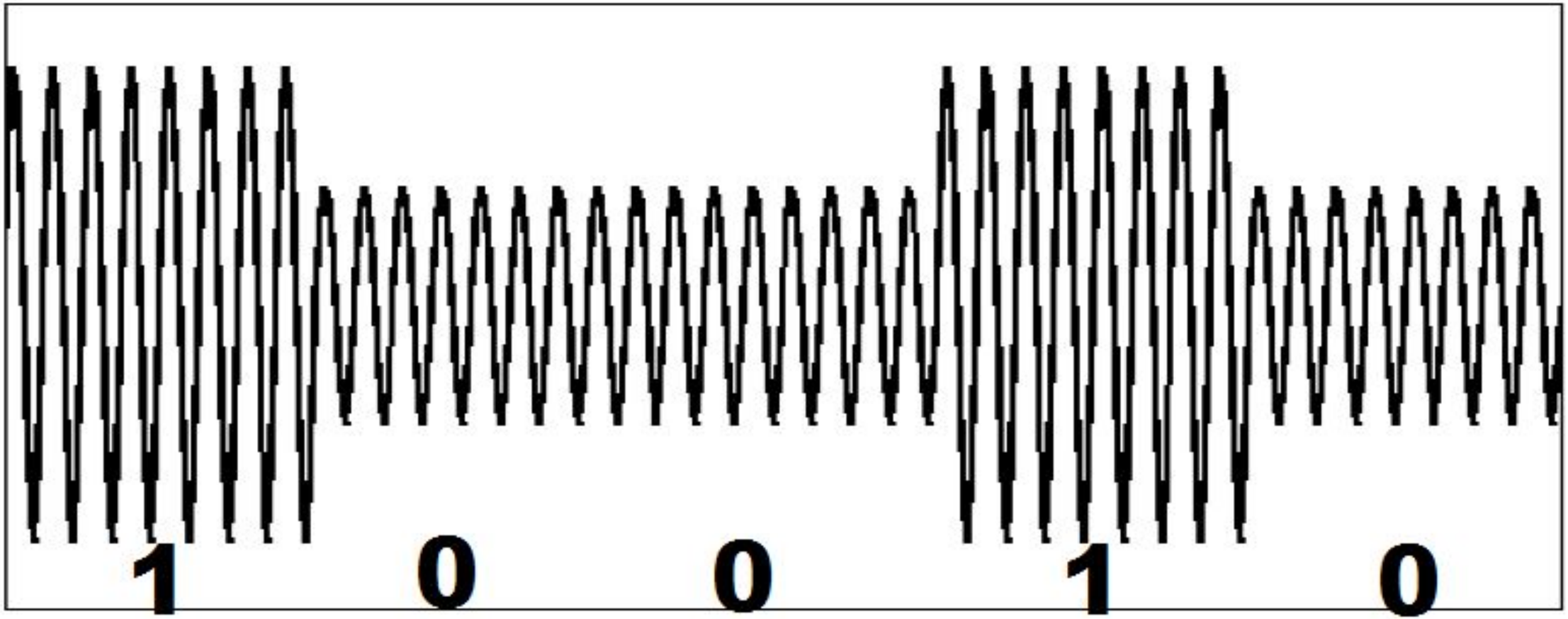
A 5-minutes dive into analyzing digital RF transmissions

- **initial signal reconnaissance**: carrier and bandwidth
- **finding precise radio parameters**: modulation, shape, simbol rate
- **developing a reliable transceiver**: embed into software or hardware

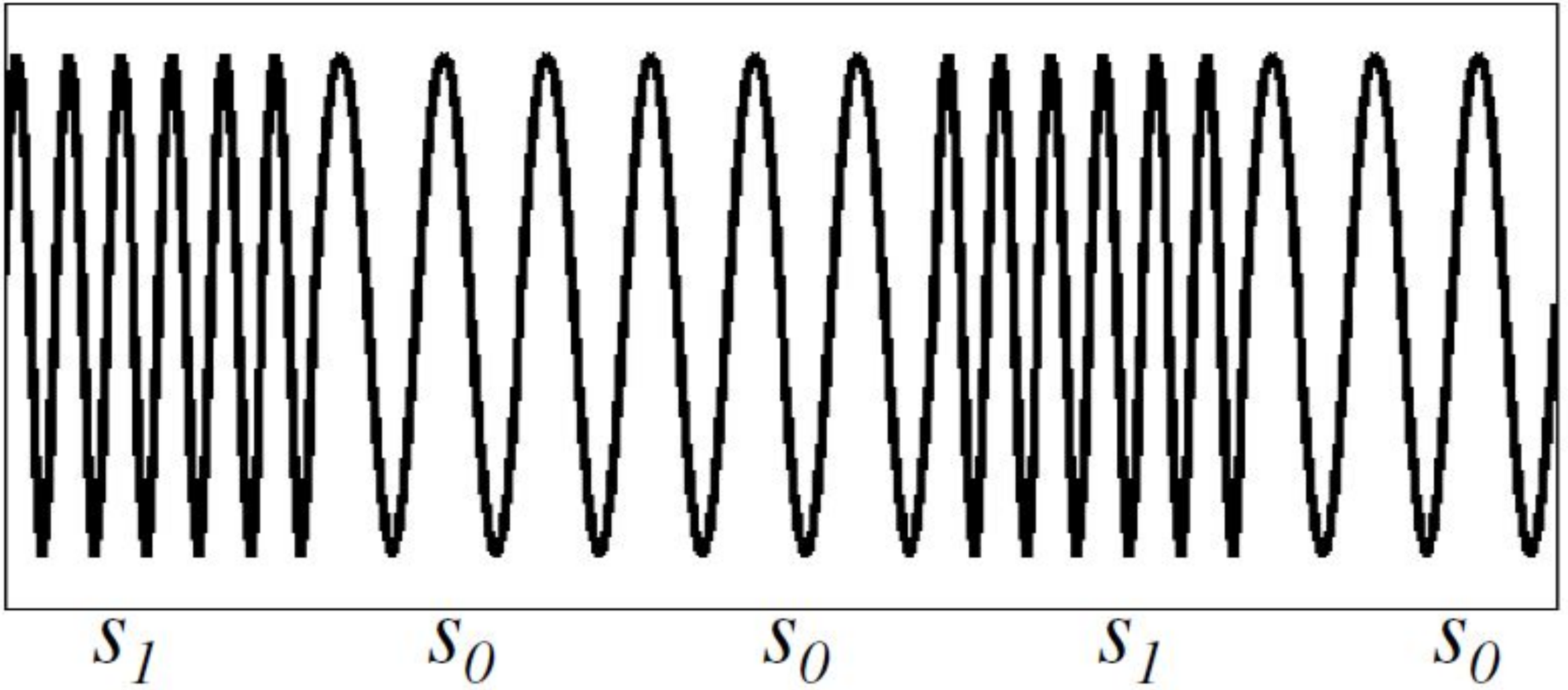
From Symbols to Signal



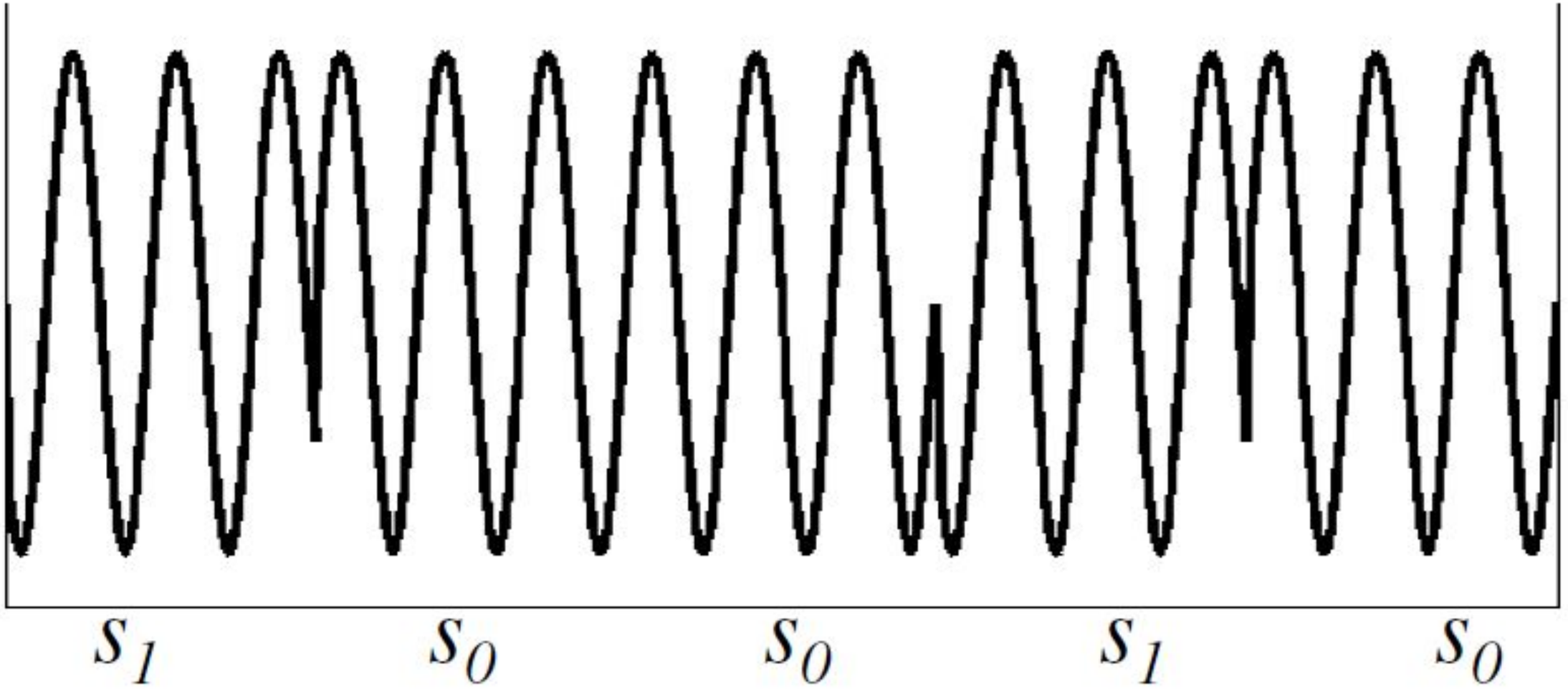
From Symbols to Signal: ASK



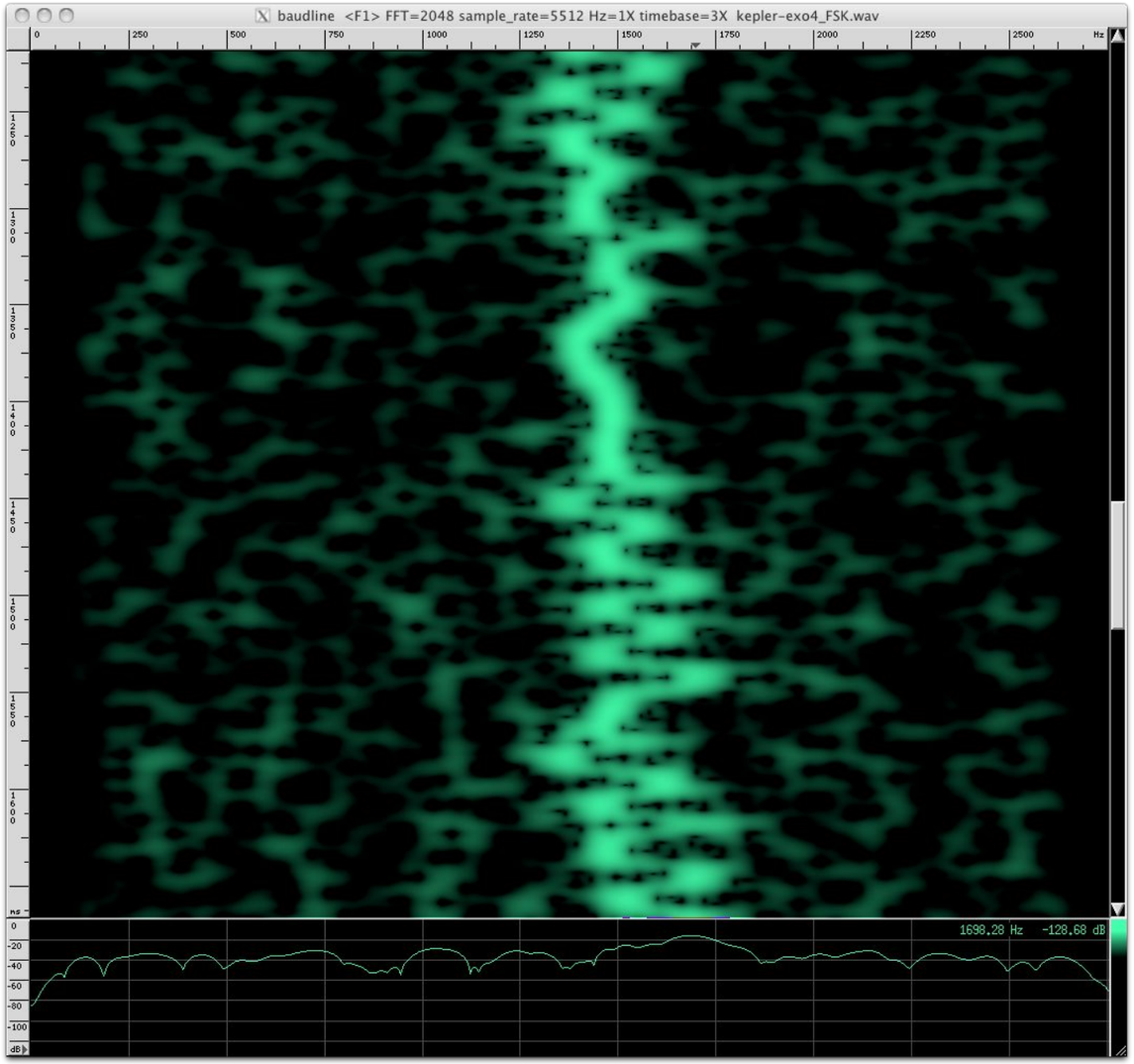
From Symbols to Signal: FSK

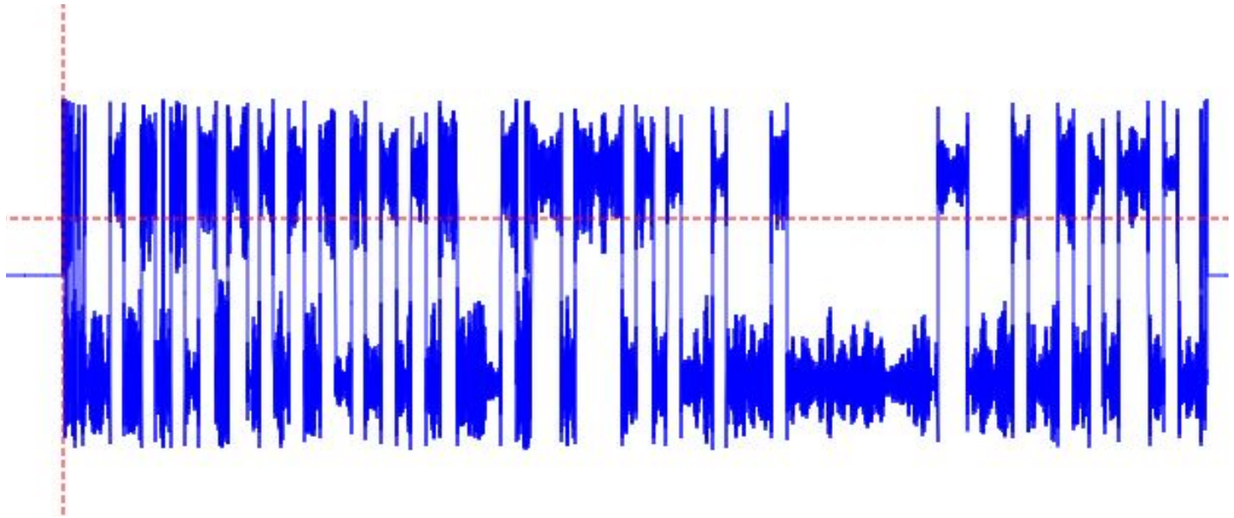


From Symbols to Signal: PSK



Real World FSK Example





We Now Have the Bitstream

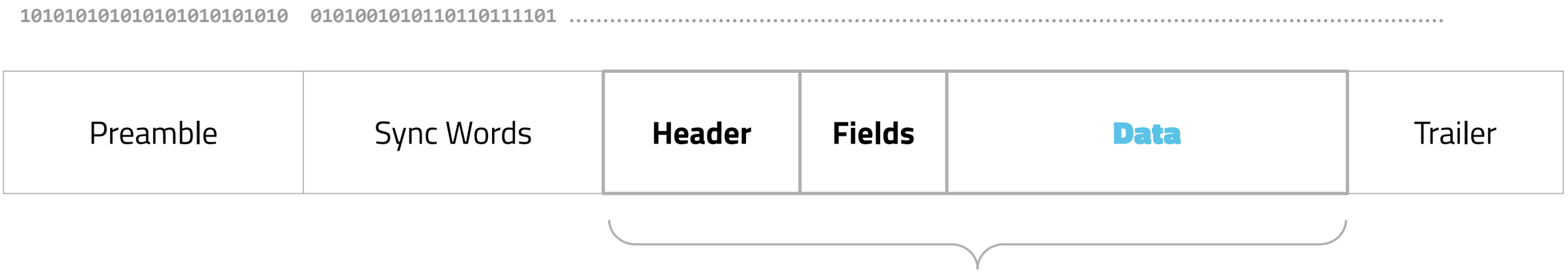
0101011011010101001010110110111101011111010101

The Hard Part is not Over

From Bitstream to Packets

```
10101010101010101010101010  01010010101101101111101 ..........................................................................................
```

From Packets to Data

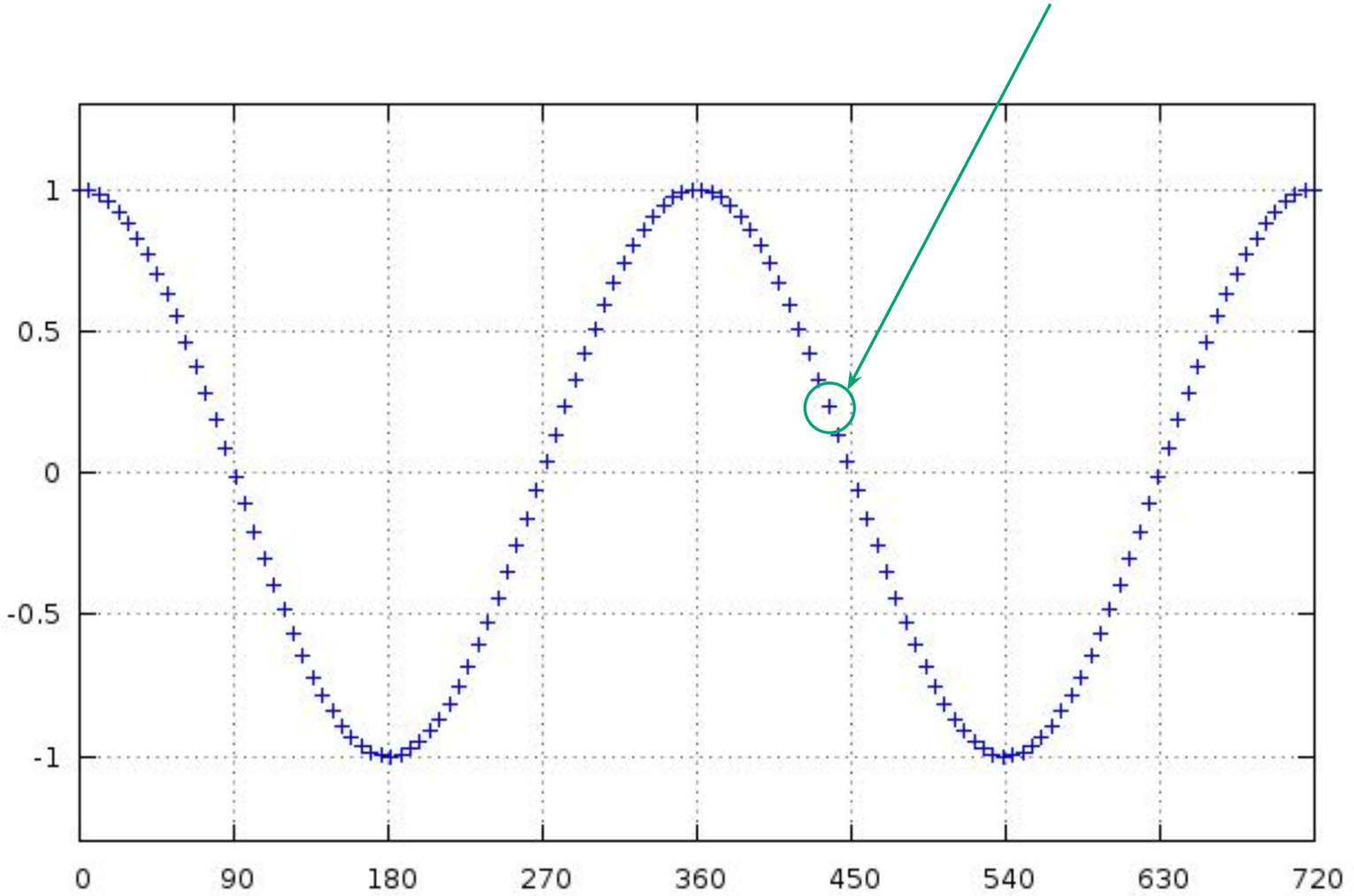


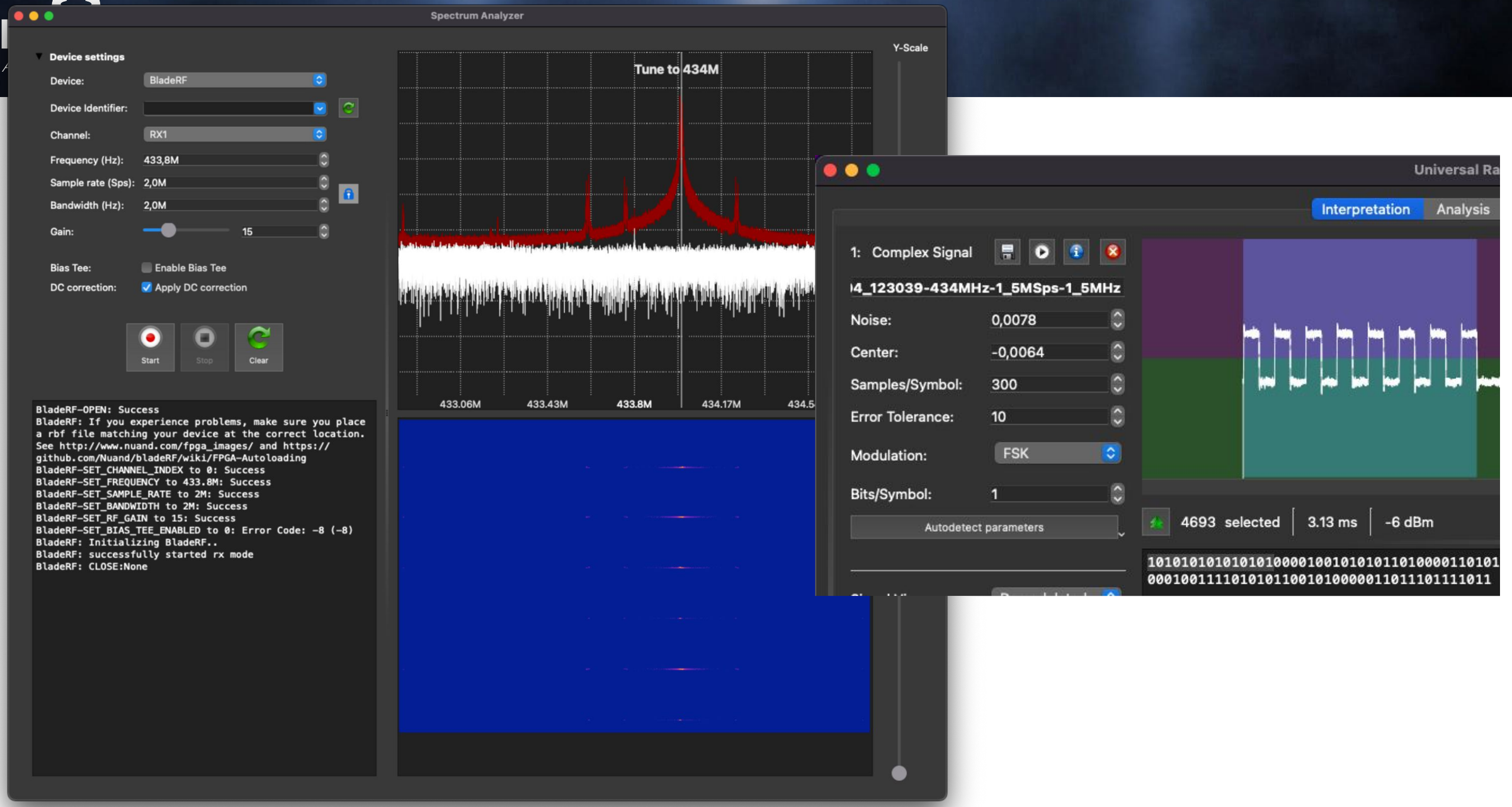
Approaches

Software Defined Radios vs. RF Dongles vs. RFQuack

Software Defined Radios

Software Defined Radios





Software Defined Radios

	SDR		RF Dongles
	GNU Radio	**URH**	
Spectrum Coverage	10^3 to 10^6Hz, continuous		
RF Parameters	Any (custom physical layer)		
Host Interfaces	Wired (USB, Ethernet)		
API Uniformity	High (established project)		
Extensibility	Very high	High	
Main Target Layers	Physical	Logic	
Focus	Research	Research, RE, Fuzzing	
Development Effort	High (DSP)	Medium (UI, Scripting)	
Transceiver Performance	Medium (Latency)		
Hardware cost	$20–15,000		

The World of RF-hacking Dongles

The World of RF-hacking Dongles


```
root@edolin ~/rfcat $ ./rfcat -r
'RfCat, the greatest thing since Frequency Hopping!'

Research Mode: enjoy the raw power of rflib

currently your environment has an object called "d" for dongle.  this is how
you interact with the rfcat dongle:
    >>> d.ping()
    >>> d.setFreq(433000000)
    >>> d.setMdmModulation(MOD_ASK_OOK)
    >>> d.makePktFLEN(250)
    >>> d.RFxmit("HALLO")
    >>> d.RFrecv()
    >>> print d.reprRadioConfig()


In [1]:
```

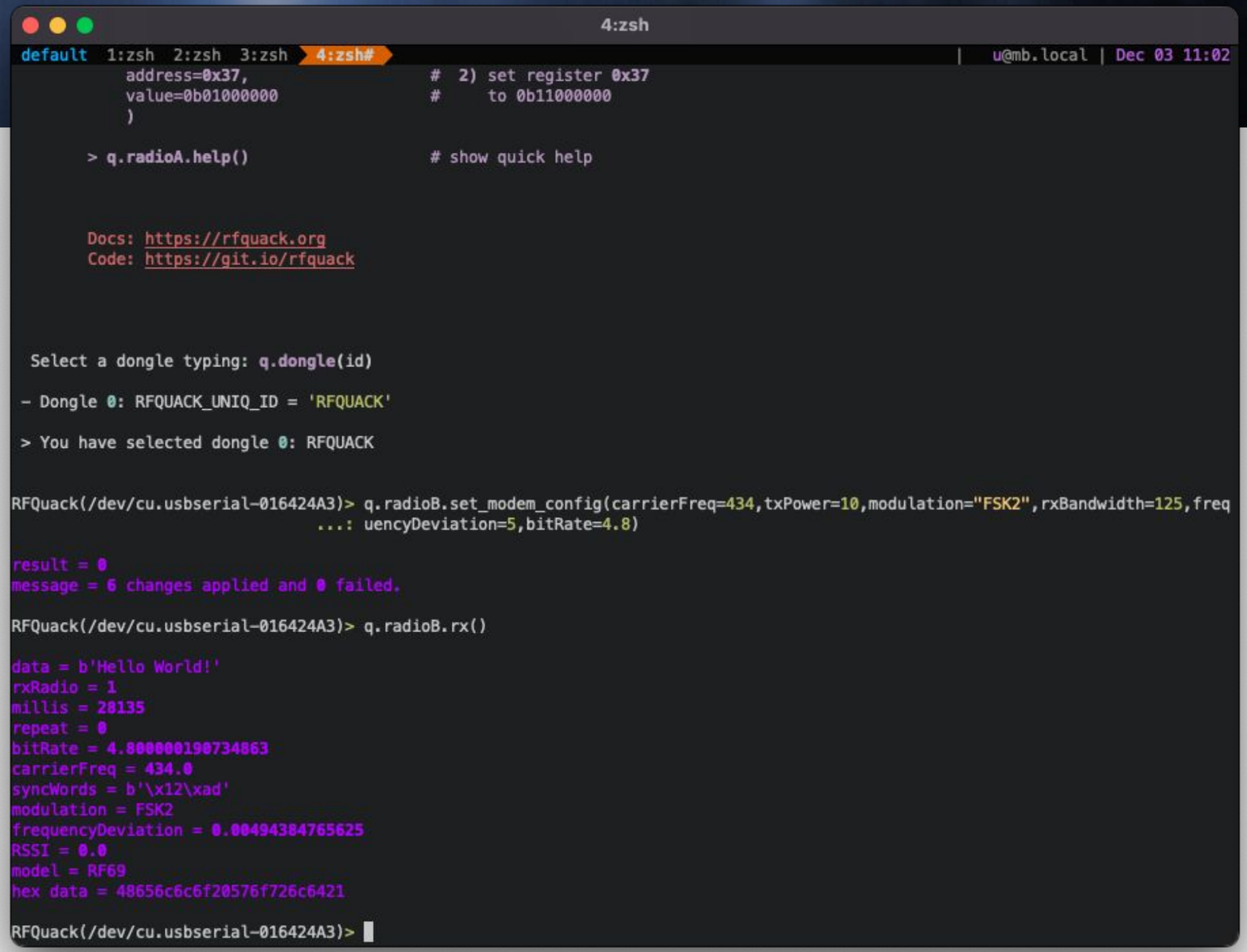
SDRs vs. RF-Hacking Dongles

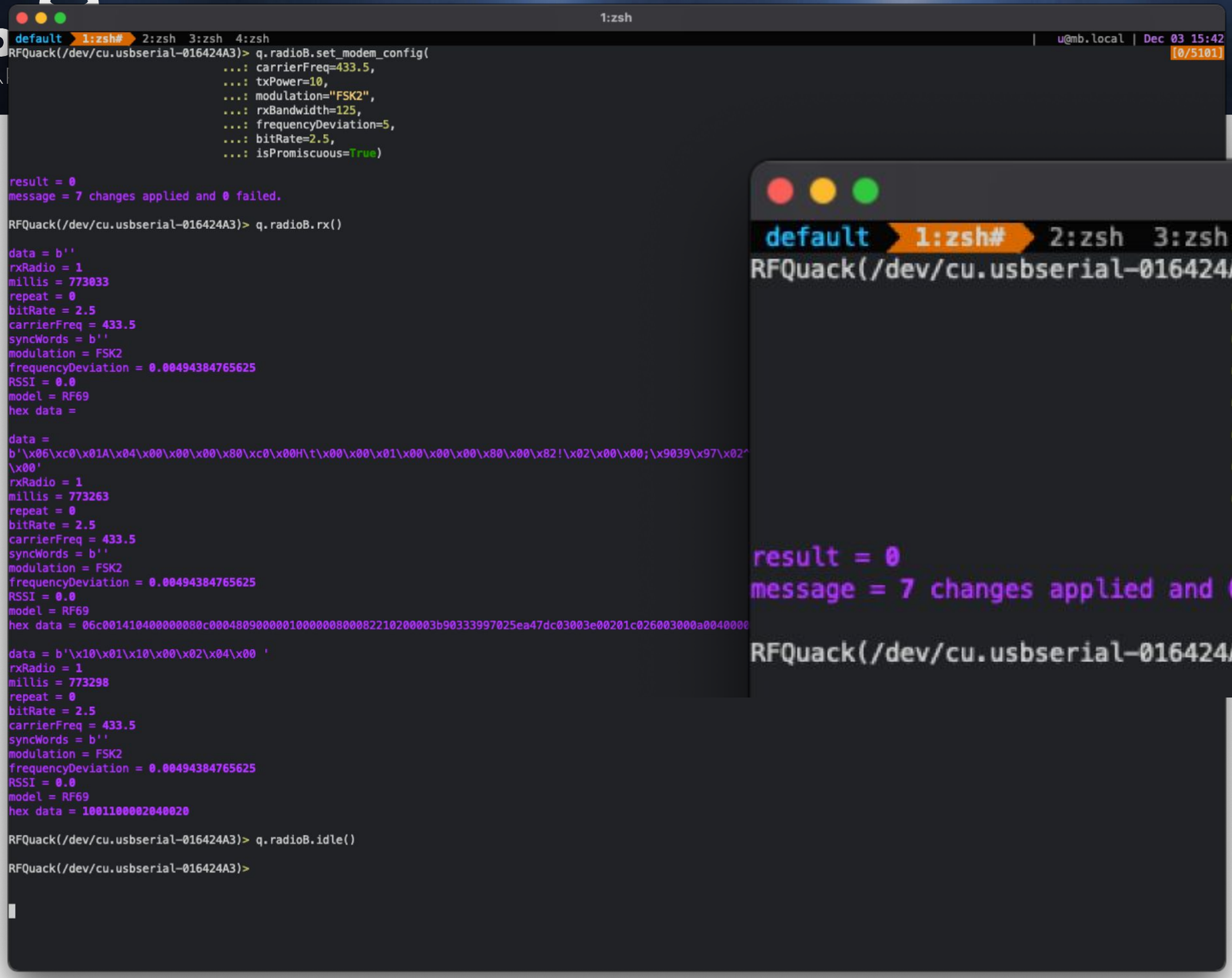


	SDR		RF Dongles	
	GNU Radio	**URH**	**Single Radio**	**Multi Radio**
Spectrum Coverage	10^3 to 10^6Hz, continuous		Only specific bands	
RF Parameters	Any (custom physical layer)		Hardware bound	
Host Interfaces	Wired (USB, Ethernet)		USB, BLE	USB, BLE, WiFi, Cellular
API Uniformity	High (established project)		Very low	
Extensibility	Very high	High	Very low (SW)	
Main Target Layers	Physical	Logic	Logic	Hybrid
Focus	Research	Research, RE, Fuzzing	Red-teaming	Red-teaming
Development Effort	High (DSP)	Medium (UI, Scripting)	Low (Scripting)	Low (UI, Scripting)
Transceiver Performance	Medium (Latency)		Very high (IC)	High (SPI)
Hardware cost	$20–15,000		$10-200	

DEMO 1

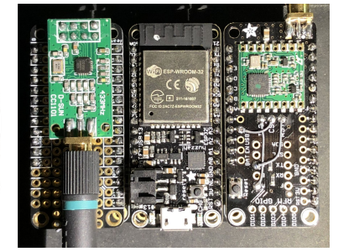
On the surface, RFQuack doesn't appear different from RFCat (CC1101-based dongles)





default 1:zsh# 2:zsh 3:zsh 4:zsh
RFQuack(/dev/cu.usbserial-016424A3)> q.radioB.set_modem_config(
...: carrierFreq=433.5,
...: txPower=10,
...: modulation="FSK2",
...: rxBandwidth=125,
...: frequencyDeviation=5,
...: bitRate=2.5,
...: isPromiscuous=True)
result = 0
message = 7 changes applied and 0 failed.
RFQuack(/dev/cu.usbserial-016424A3)> q.radioB.rx()

SDRs vs. RF-hacking Dongles vs. RFQuack

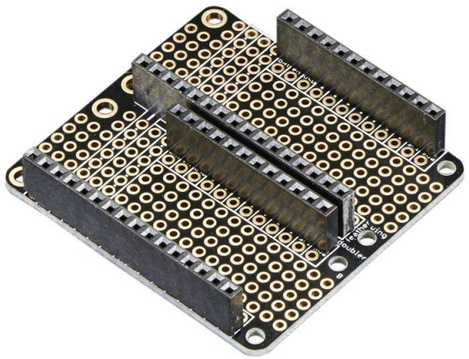


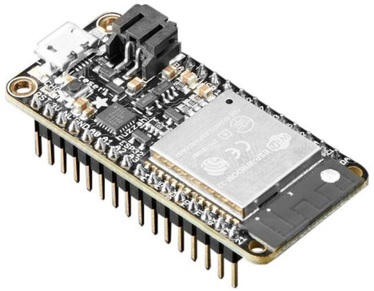
	SDR		RF Dongles		
	GNU Radio	**URH**	**Single Radio**	**Multi Radio**	**RFQuack**
Spectrum Coverage	10^3 to 10^6Hz, continuous		Only specific bands		Arbitrary bands, discrete
RF Parameters	Any (custom physical layer)		Hardware bound		All (modular)
Host Interfaces	Wired (USB, Ethernet)		USB, BLE	USB, BLE, WiFi, Cellular	Any (modular)
API Uniformity	High (established project)		Very low		High
Extensibility	Very high	High	Very low (SW)		High (HW, SW)
Main Target Layers	Physical	Logic	Logic	Hybrid	Hybrid
Focus	Research	Research, RE, Fuzzing	Red-teaming	Red-teaming	Research, RE, Fuzzing
Development Effort	High (DSP)	Medium (UI, Scripting)	Low (Scripting)	Low (UI, Scripting)	Medium (Scripting, C)
Transceiver Performance	Medium (Latency)		Very high (IC)	High (SPI)	High (SPI)
Hardware cost	$20–15,000		$10-200		

DEMO 2

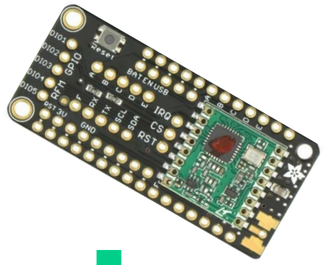
Easy hardware & firmware customization

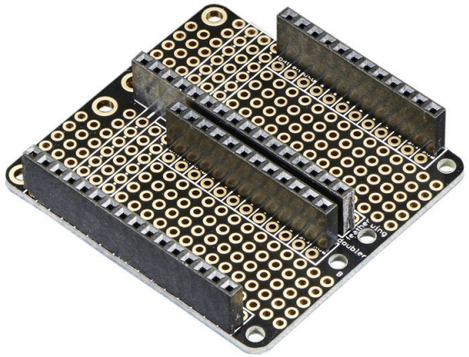
Let's Make a 433MHz Dongle

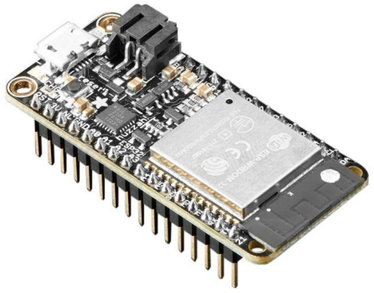


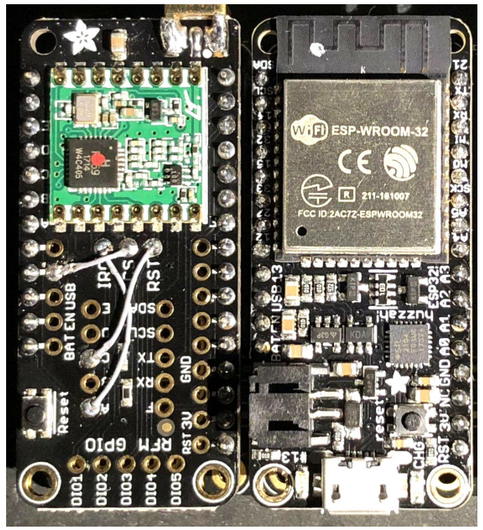


Let's Make a 433MHz Dongle



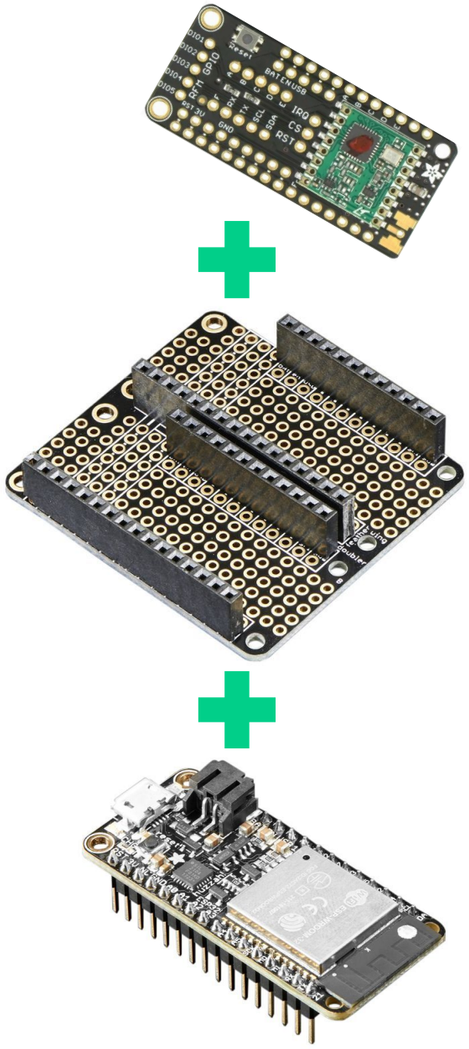


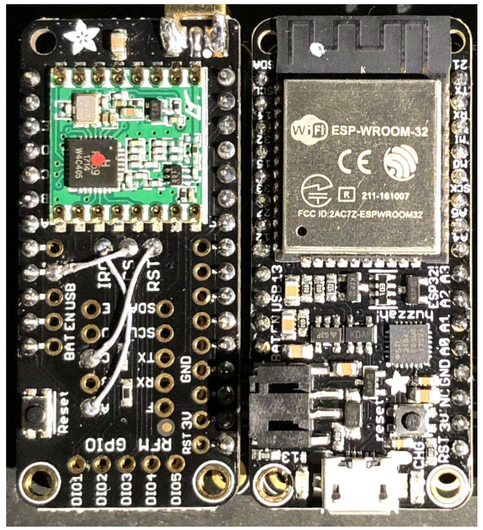




433MHz

Let's Make a 433MHz Dongle

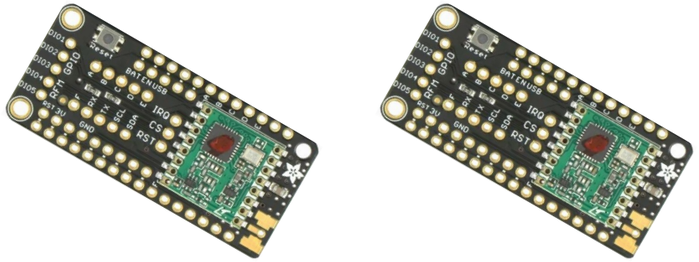




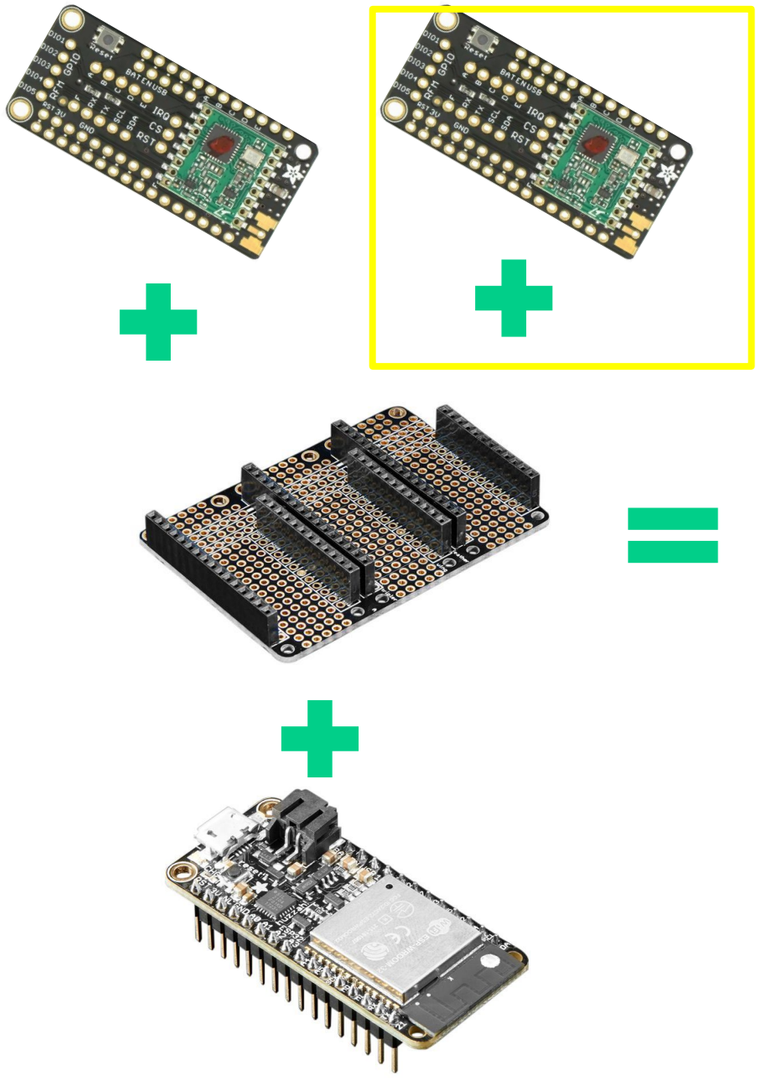
433MHz

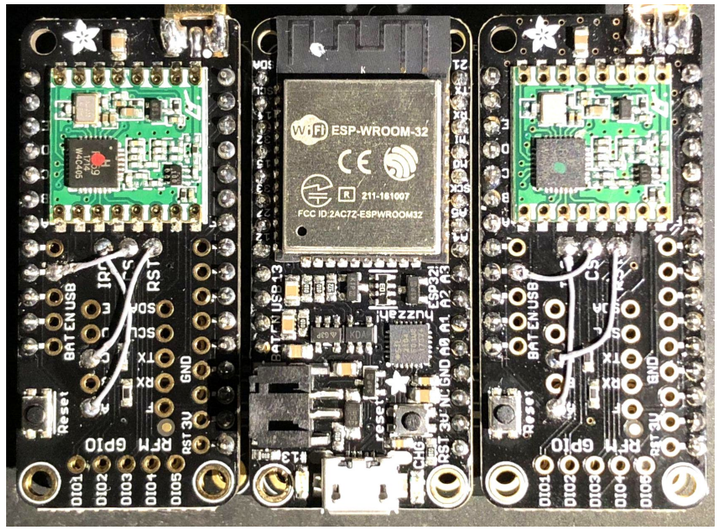
```
$ vim build.env

RADIOA=RF69

RADIOA_CS=13

RADIOA_IRQ=27

LOG_ENABLED=true


$ make flash ⚡
```

"But I need two radios!"



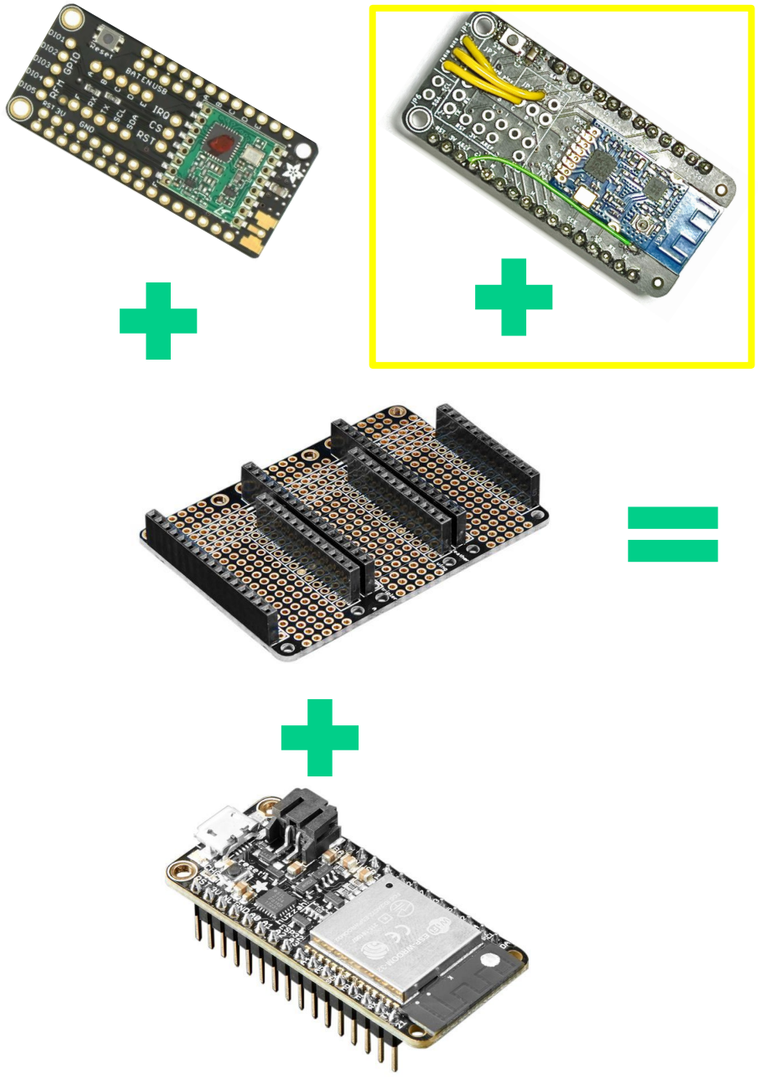
Here: A Dual-radio Firmware

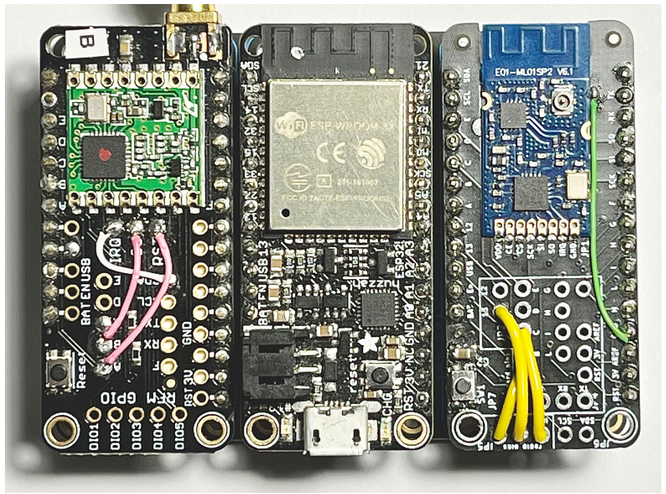





```
$ vim build.env

RADIOA=RF69

RADIOA_CS=13

RADIOA_IRQ=27

RADIOB=RF69

RADIOB_CS=33

RADIOB_IRQ=23


LOG_ENABLED=true


$ make flash ⚡
```

"But I want different radios!"





433MHz 2.4GHz

```
$ vim build.env

RADIOA=nRF24
RADIOA_CS=13
RADIOA_IRQ=15
RADIOA_RST=27

RADIOB=RF69
RADIOB_CS=33
RADIOB_IRQ=23


LOG_ENABLED=true



$ make flash ⚡
```

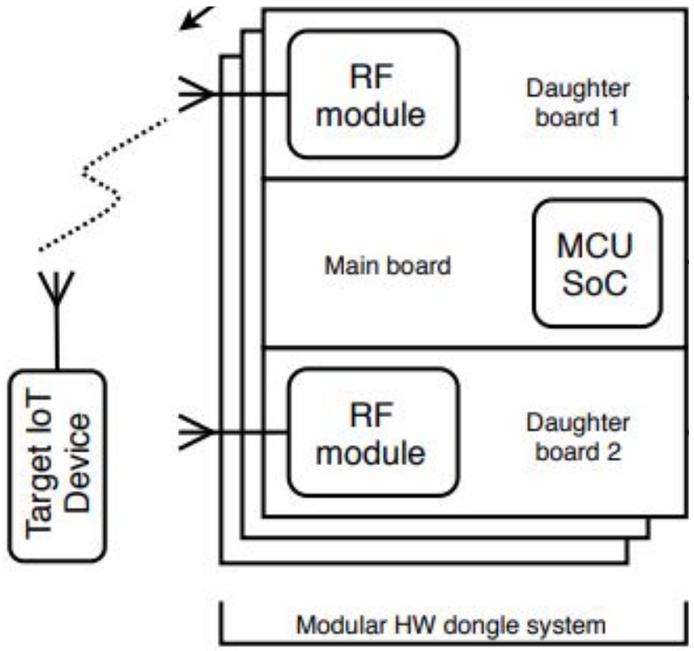
Internals

The life of an RF packet: from the air, through RFQuack, into your Python objects

The Modular Hardware

Shared SPI bus

1 RF module has dedicated:
- CS
- IRQ
- [RST]

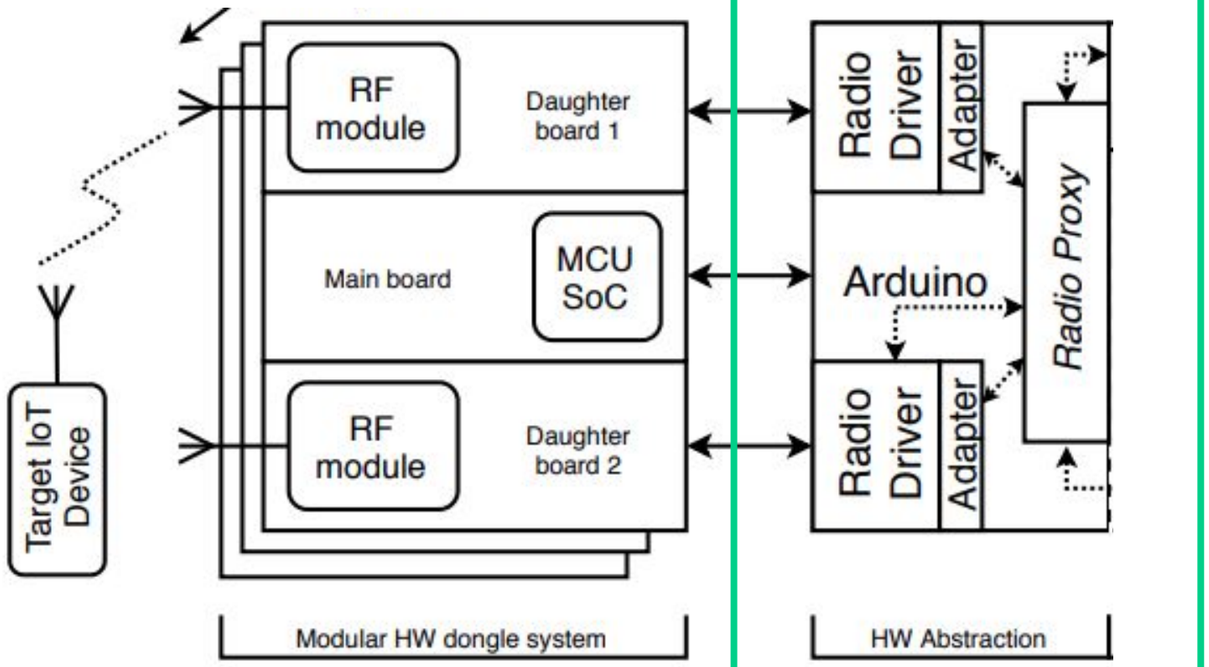


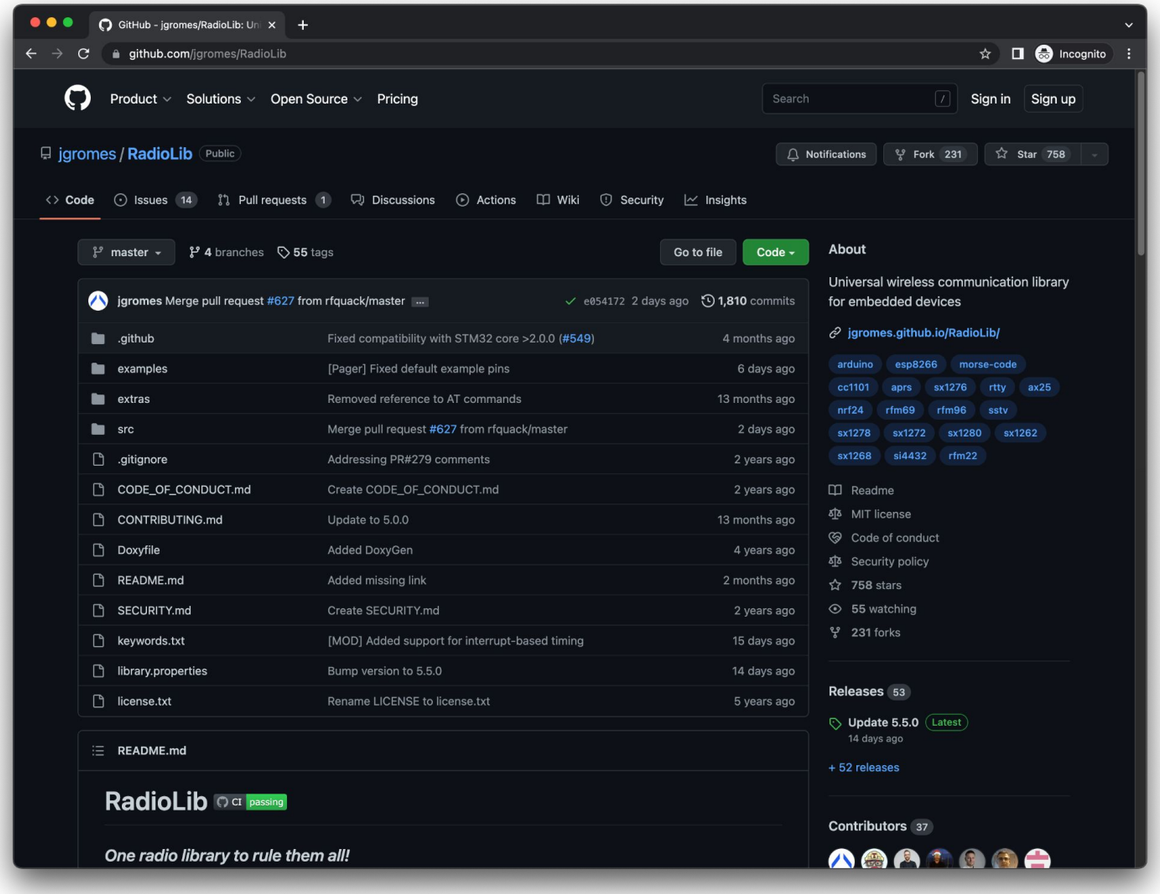
The RF Abstraction

Shared SPI bus

1 RF module has dedicated:
- CS
- IRQ
- [RST]

```
radio
├── RFQCC1101.h
├── RFQRFM69.h
├── RFQnRF24.h
├── ...
├── RadioLibWrapper.h
└── drivers.h
```





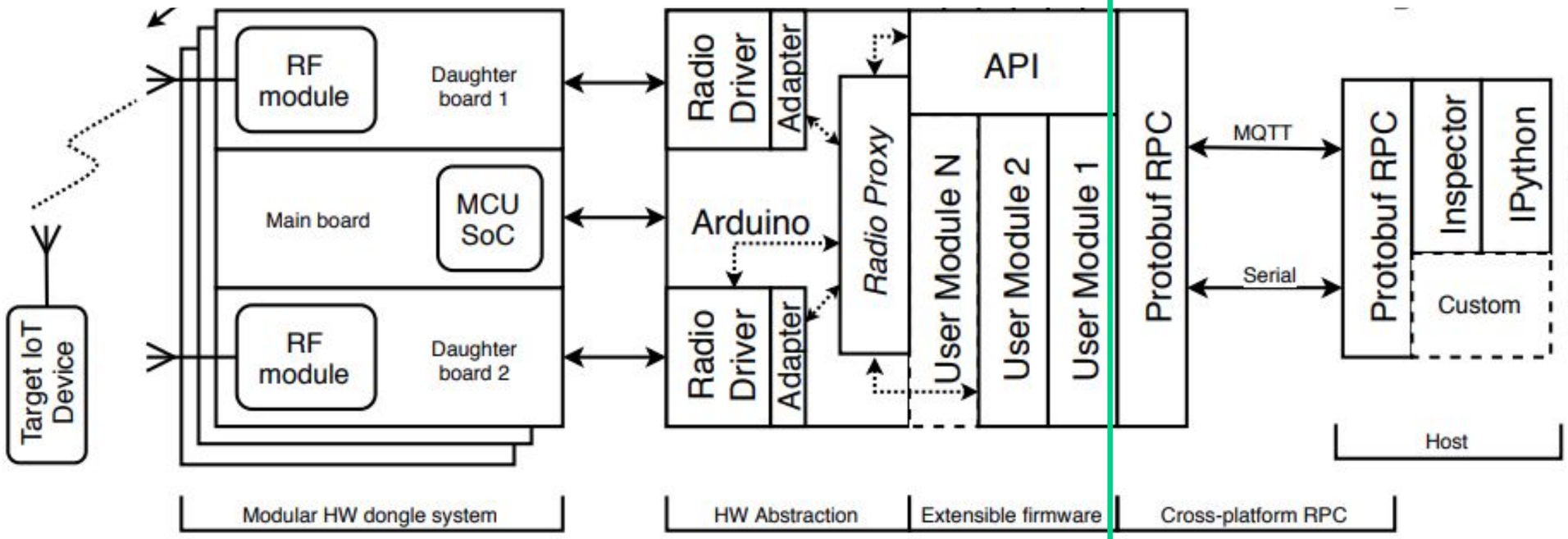
Cross-platform Protocol

Shared SPI bus

1 RF module has dedicated:
- CS
- IRQ
- [RST]

```
├── radio
│   ├── RFQCC1101.h
│   ├── RFQRFM69.h
│   ├── RFQnRF24.h
│   ├── ...
│   ├── RadioLibWrapper.h
│   └── drivers.h
```

```
├── rfquack_common.h
├── rfquack_config.h
├── rfquack_logging.h
├── rfquack_network.h
└── rfquack_transport.h
```



Example Protobuf Message

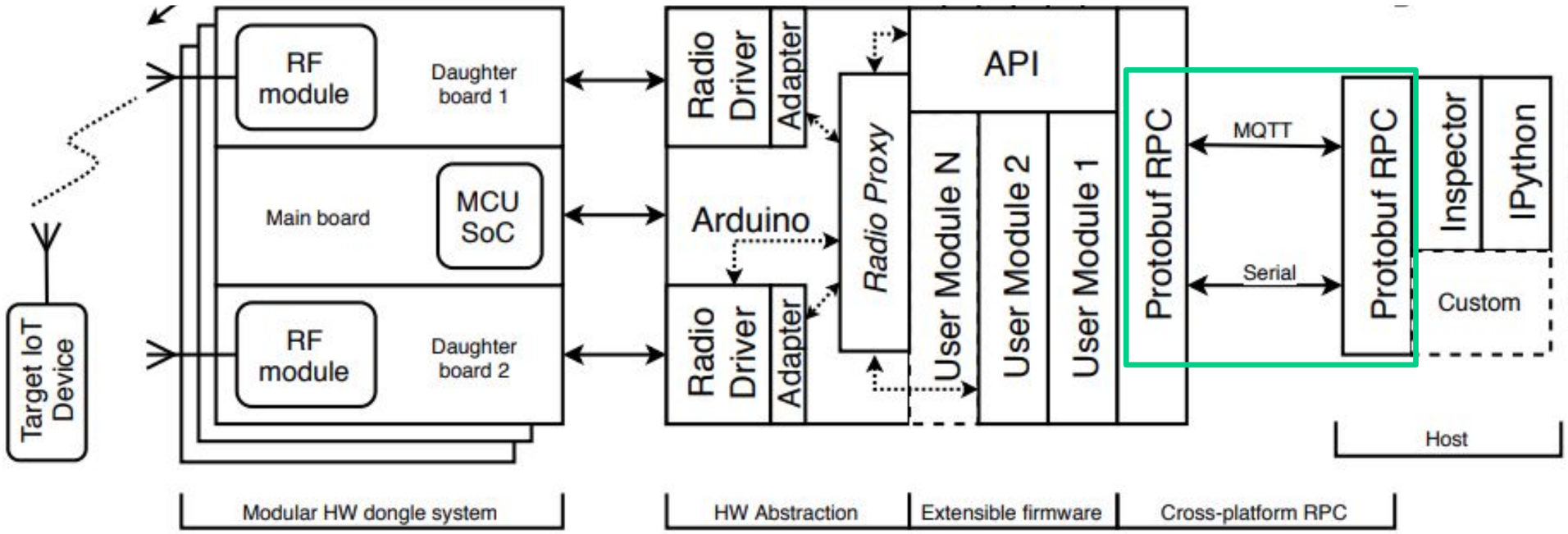
Shared SPI bus

1 RF module has dedicated:
- CS
- IRQ
- [RST]

```
├── radio
│   ├── RFQCC1101.h
│   ├── RFQRFM69.h
│   ├── RFQnRF24.h
│   ├── ...
│   ├── RadioLibWrapper.h
│   └── drivers.h
```

```
├── rfquack_common.h
├── rfquack_config.h
├── rfquack_logging.h
├── rfquack_network.h
└── rfquack_transport.h
```

```
message Packet {
    required bytes data = 1;
    optional WhichRadio rxRadio = 2;
    optional uint64 millis = 3;
    optional uint32 repeat = 4;
    optional float bitRate = 5;
    optional float carrierFreq = 6;
    optional bytes syncWords = 7;
    optional string modulation = 8;
    optional float frequencyDeviation = 9;
    optional float RSSI = 10;
    optional string model = 11;
}
```



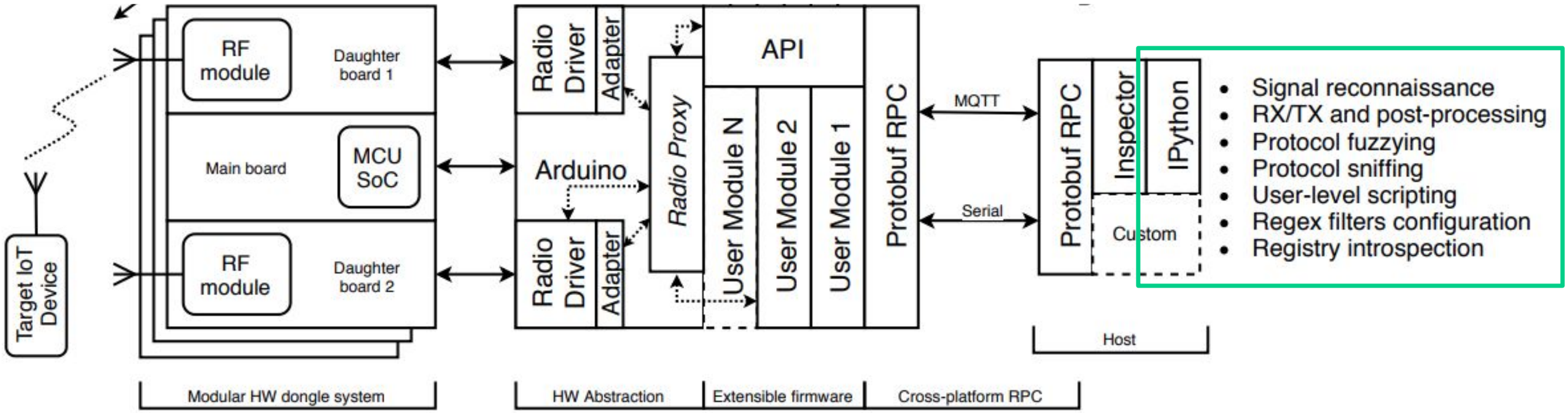
The IPython-based Console

Shared SPI bus

1 RF module has dedicated:
- CS
- IRQ
- [RST]

```
├── radio
│   ├── RFQCC1101.h
│   ├── RFQRFM69.h
│   ├── RFQnRF24.h
│   ├── ...
│   ├── RadioLibWrapper.h
│   └── drivers.h
```

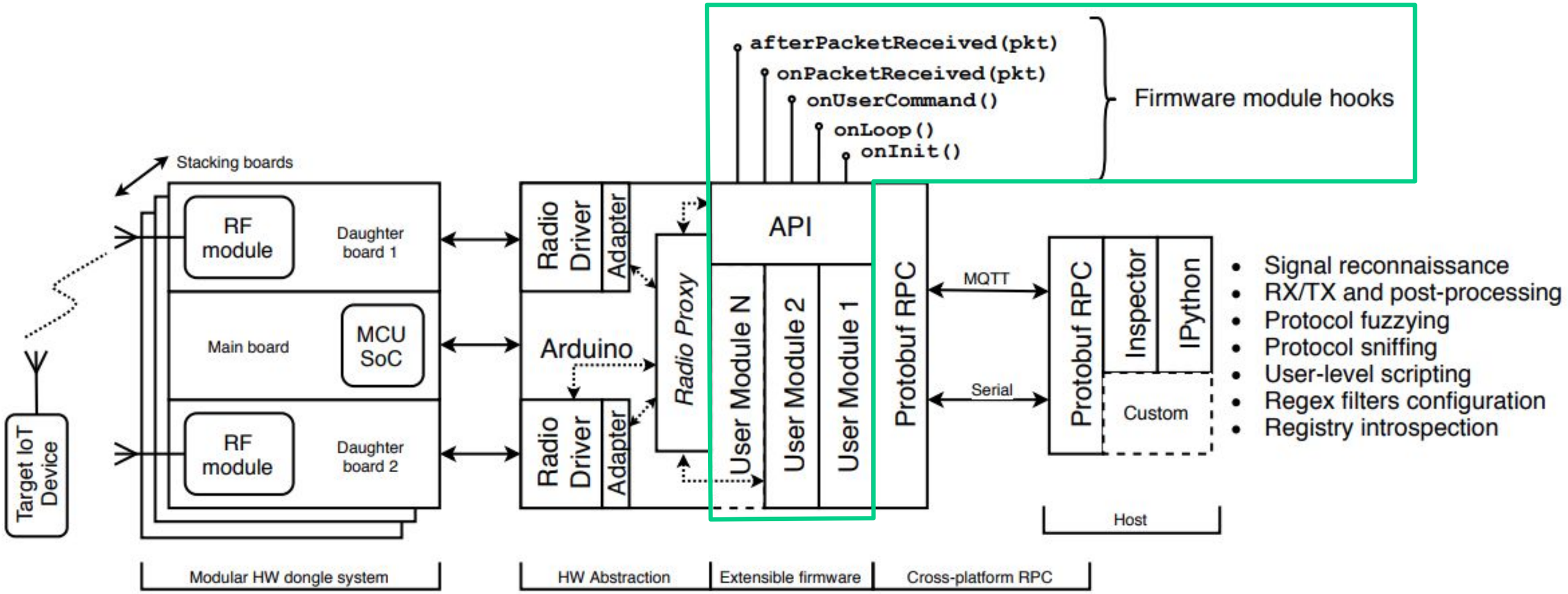
```
├── rfquack_common.h
├── rfquack_config.h
├── rfquack_logging.h
├── rfquack_network.h
└── rfquack_transport.h
```



Modules

Hackable firmware with a humanized interface

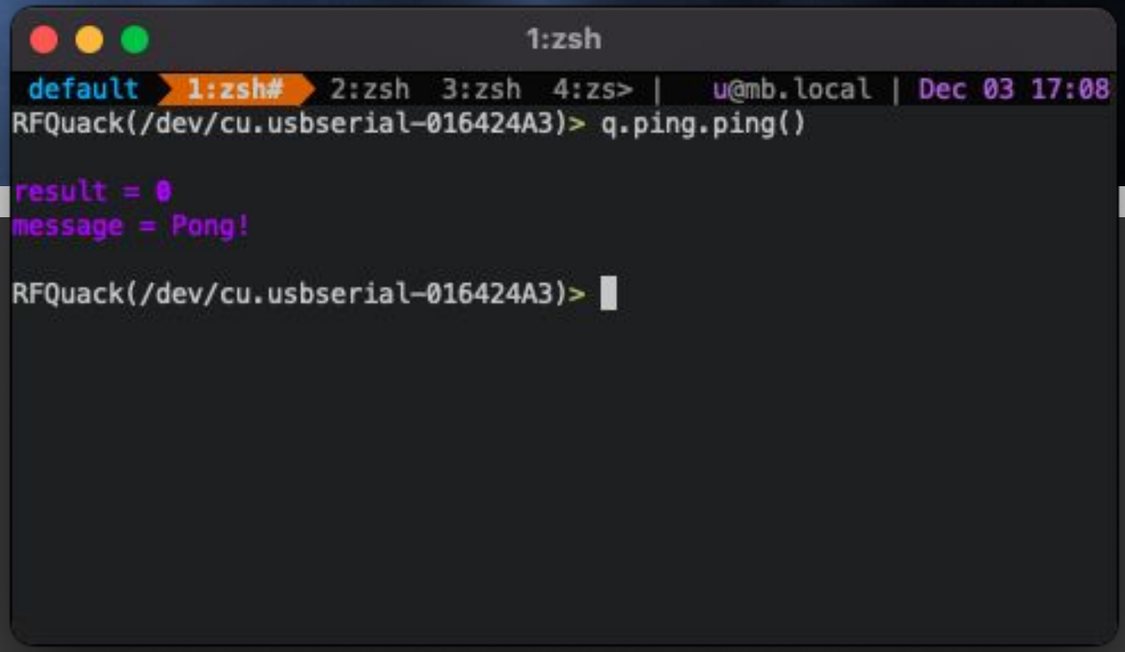
Firmware Modules with Hooks



Enabling Modules

```
$ vim build.env

RADIOA=RF69

...


GUESSING_MODULE=true

FREQ_SCANNER_MODULE=true

MOUSE_JACK_MODULE=true

PACKET_FILTER_MODULE=true

PACKET_MOD_MODULE=true

PACKET_REPEAT_MODULE=true

ROLL_JAM_MODULE=true


$ make flash ⚡
```

```
├── modules
│   ├── ModulesDispatcher.h
│   ├── README.md
│   ├── RFQModule.h
│   ├── defaults
│   │   ├── FrequencyScannerModule.h
│   │   ├── GuessingModule.h
│   │   ├── HelloWorldModule.h
│   │   ├── MouseJackModule.h
│   │   ├── PacketFilterModule.h
│   │   ├── PacketModificationModule.h
│   │   ├── PacketRepeaterModule.h
│   │   ├── PingModule.h
│   │   ├── RadioModule.h
│   │   └── RollJamModule.h
│   └── hooks
│       ├── AfterPacketReceived.h
│       ├── OnLoop.h
│       └── OnPacketReceived.h
```




```
class PingModule : public RFQModule {
public:
    PingModule() : RFQModule("ping") {}

    void onInit() override {
      // Nothing to do :)
    }

    void executeUserCommand(char *verb, char **args, uint8_t argsLen, char *messagePayload,
                            unsigned int messageLen) override {
      // Ping this dongle
      CMD_MATCHES_METHOD_CALL(rfquack_VoidValue, "ping", "Replies to ping", ping(reply))
    }

    void ping(rfquack_CmdReply &reply) {
      setReplyMessage(reply, F("Pong!"), 0);
    }
};
```

NO CLIENT CODE NEEDED 😱

Modules and their methods are **automatically discovered** by the CLI.

Supporting Python code is **generated** on the fly.

Including a short doc string. 🤯

DEMO 3

Scripting attacks: a micro-iptables inside a chip!

DEMO 3

- IoT node that sends random challenge signals
- expects "secret" modified packets in response

- scripting filtering
  - from the host
  - from inside the firmware (without touching the firmware!)

- scripting modified response packet
  - from the host
  - from inside the firmware (without touching the firmware!)

```
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
RX timeout!
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
RX timeout!
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
success!
RX data:        OH OH OH SkKS
🔵 BLINK 🔵 BLINK 🔵 BLINK 🔵 BLINK 🔵 BLINK
         RSSI: -57.00 dBm
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
RX timeout!
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
success!
RX data:        OH OH OH SkKS
🔵 BLINK 🔵 BLINK 🔵 BLINK 🔵 BLINK 🔵 BLINK
         RSSI: -56.50 dBm
------------------------------------ TX @ 434MHz
TX success!
        Data: OH OH OH SJKS
------------------------------------ RX @ 433MHz
RX timeout!
```

1:zsh

default 1:zsh# 2:zsh 3:zsh 4:zsh | u@mb.local | Dec 04 00:02 [39/19039]

```
RFQuack(/dev/cu.usbserial-016424A3)> q.radioA.set_modem_config(carrierFreq=433,txPower=10,modulation="FSK2",rxBandwidth=10.4,frequenc
                                ...: yDeviation=5,bitRate=4.8)

result = 0
message = 6 changes applied and 0 failed.

RFQuack(/dev/cu.usbserial-016424A3)> q.packet_modification.add(position=10, operand=33, operation="XOR")

result = 0
message = Rule added, there are 1 modification rule(s).

RFQuack(/dev/cu.usbserial-016424A3)> q.packet_repeater.enabled = True

result = 0
message =

RFQuack(/dev/cu.usbserial-016424A3)> q.packet_modification.enabled = False

result = 0
message =

RFQuack(/dev/cu.usbserial-016424A3)> q.packet_modification.enabled = True

result = 0
message =

RFQuack(/dev/cu.usbserial-016424A3)> q.radioB.rx()

RFQuack(/dev/cu.usbserial-016424A3)> q.radioA.tx()

RFQuack(/dev/cu.usbserial-016424A3)> q.packet_repeater.repeat = 5

result = 0
message =

data = b'OH OH OH SkKS'
rxRadio = 1
millis = 76313
repeat = 5
bitRate = 4.800000190734863
```

Payload Filtering

Simple filtering done by the radio
Configurable via registers

Same

Complex filtering done by RFQuack
Configurable via regexes

Payload Modification

Content-based byte-level access

```
Payload[indexOf(0x42)] = Payload[indexOf(0x42)] OP Value
```

1 **2** 3 4 5 **6** 7 8 9 10 11 **12** **13** **14** 15

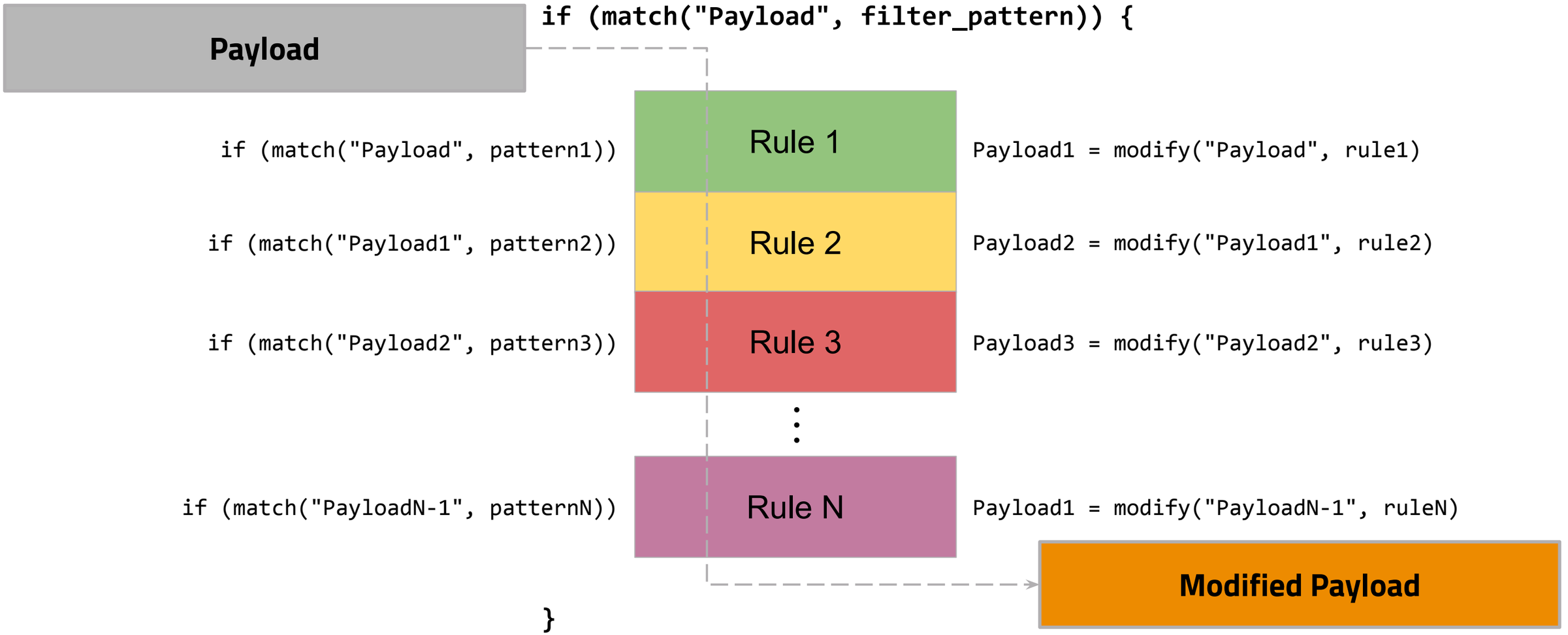
Positional byte-level access

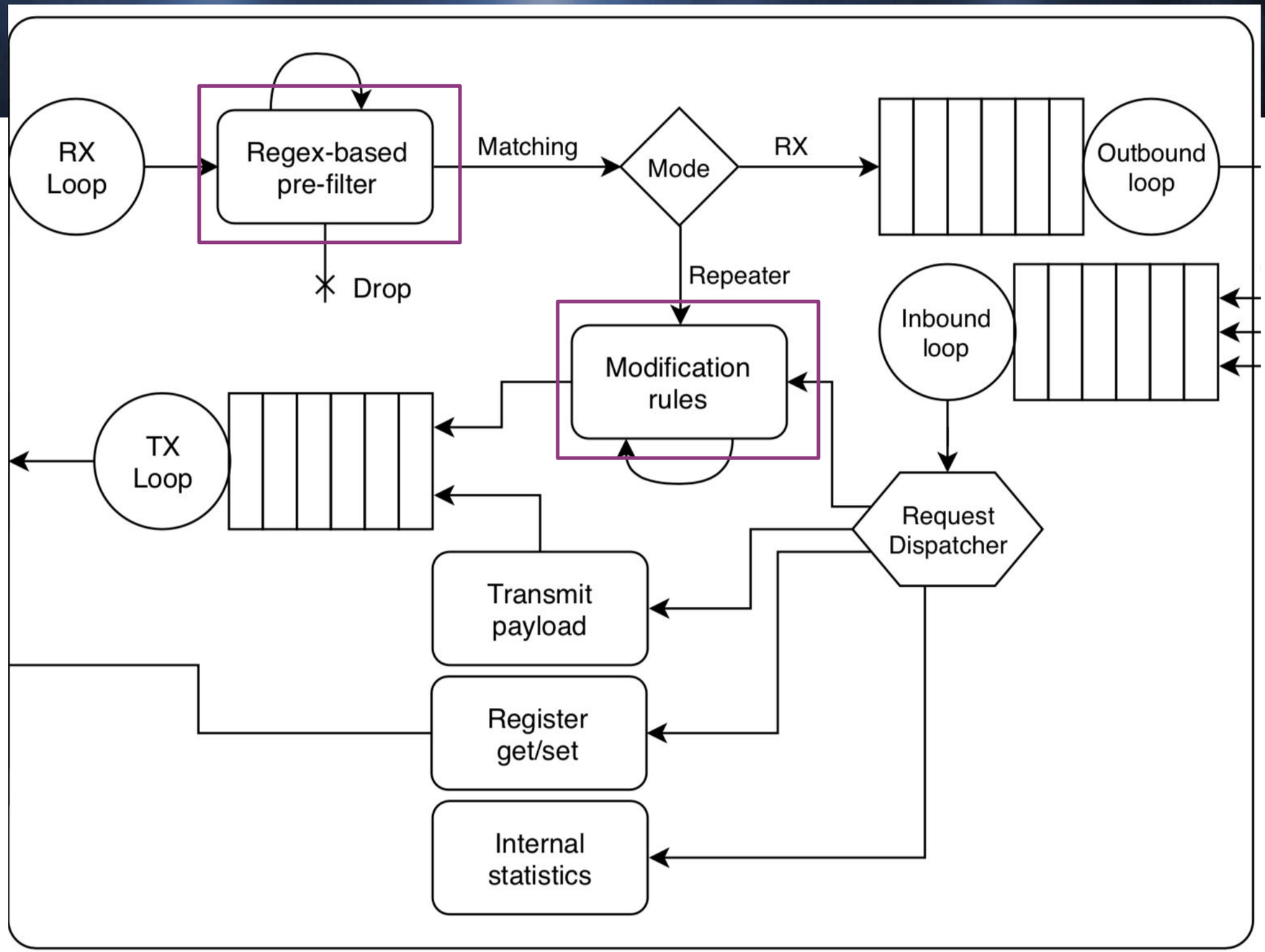
```
Payload[position] = Payload[position] OP Value
```

Operations:

XOR
AND
NOT
>>
<<

Rule-based Payload Modification





Beyond RF

Poking SPI registers from the comfort of your terminal

Poking with SPI Registers

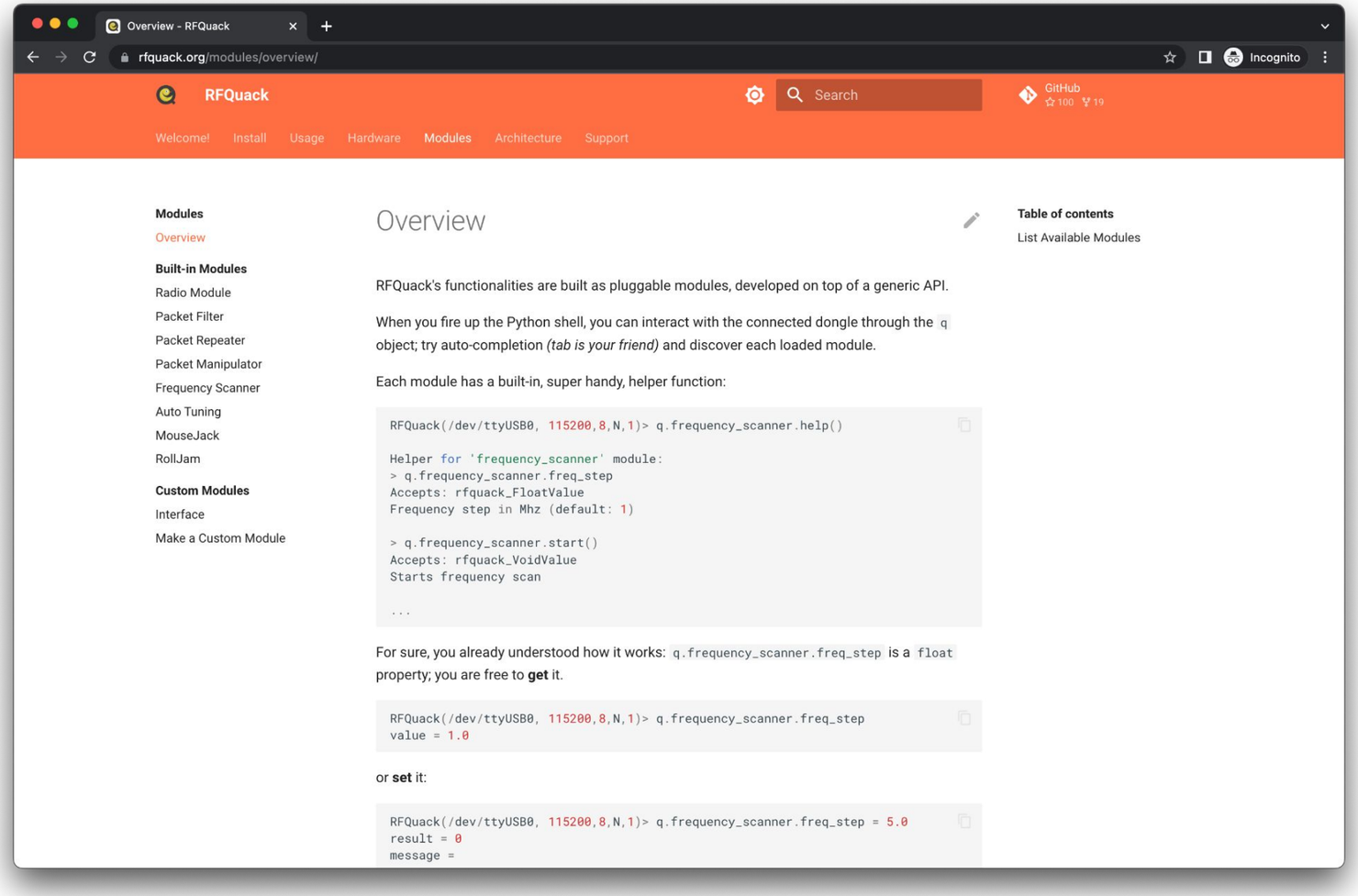
```
q.radioA.set_register(
    0x2e,              # register address (8 or 16 bits)
    fuzz_me(x))        # register value (you can write in HEX or DEC too)
```

- You could **bypass** any (modem) configuration
- You should **study the datasheet** of the radio module (or not!)
- You could easily "hang" the radio and RFQuack (just push reset, or register a successful crash!)

Resources

Documentation, videos, examples

Official Documentation



- getting started
- downloading
- pre-requisites
- configuring
- building
- flashing
- using the CLI
- ...
- ...a bit WIP!

Paper, Code, Videos

RFQuack: A Universal Hardware-Software Toolkit for Wireless Protocol (Security) Analysis and Research

Federico Maggi
Trend Micro Research
federico@maggi.cc

Andrea Guglielmini
Politecnico di Milano
andrea.guglielmini@mail.polimi.it

Abstract

Software-defined radios (SDRs) are indispensable for signal reconnaissance and physical-layer dissection, but despite we have advanced tools like Universal Radio Hacker, SDR-based approaches require substantial effort. Contrarily, RF dongles such as the popular Yard Stick One are easy to use and guarantee a deterministic physical-layer implementation. However, they're not very flexible, as each dongle is a static hardware system with a monolithic firmware.

We present RFQuack, an open-source tool and library firmware that combines the flexibility of a software-based approach with the determinism and performance of embedded RF frontends. RFQuack is based on a multi-radio hardware system with swappable RF frontends, and a firmware that exposes a uniform, hardware-agnostic API. RFQuack focuses on a structured firmware architecture that allows high- and low-level interaction with the RF frontends. It facilitates the development of host-side scripts and firmware plug-ins, to implement efficient data-processing pipelines or interactive protocols, thanks to the multi-radio support. RFQuack has an IPython shell and 9 firmware modules for: spectrum scanning, automatic carrier detection and bitrate estimation, headless operation with remote management, in-flight packet filtering and manipulation, MouseJack, and RollJam (as examples).

We used RFQuack to setup RF hacking contests, analyze industrial-grade devices and key fobs, on which we found and reported 11 vulnerabilities in their RF protocols.

1 Introduction

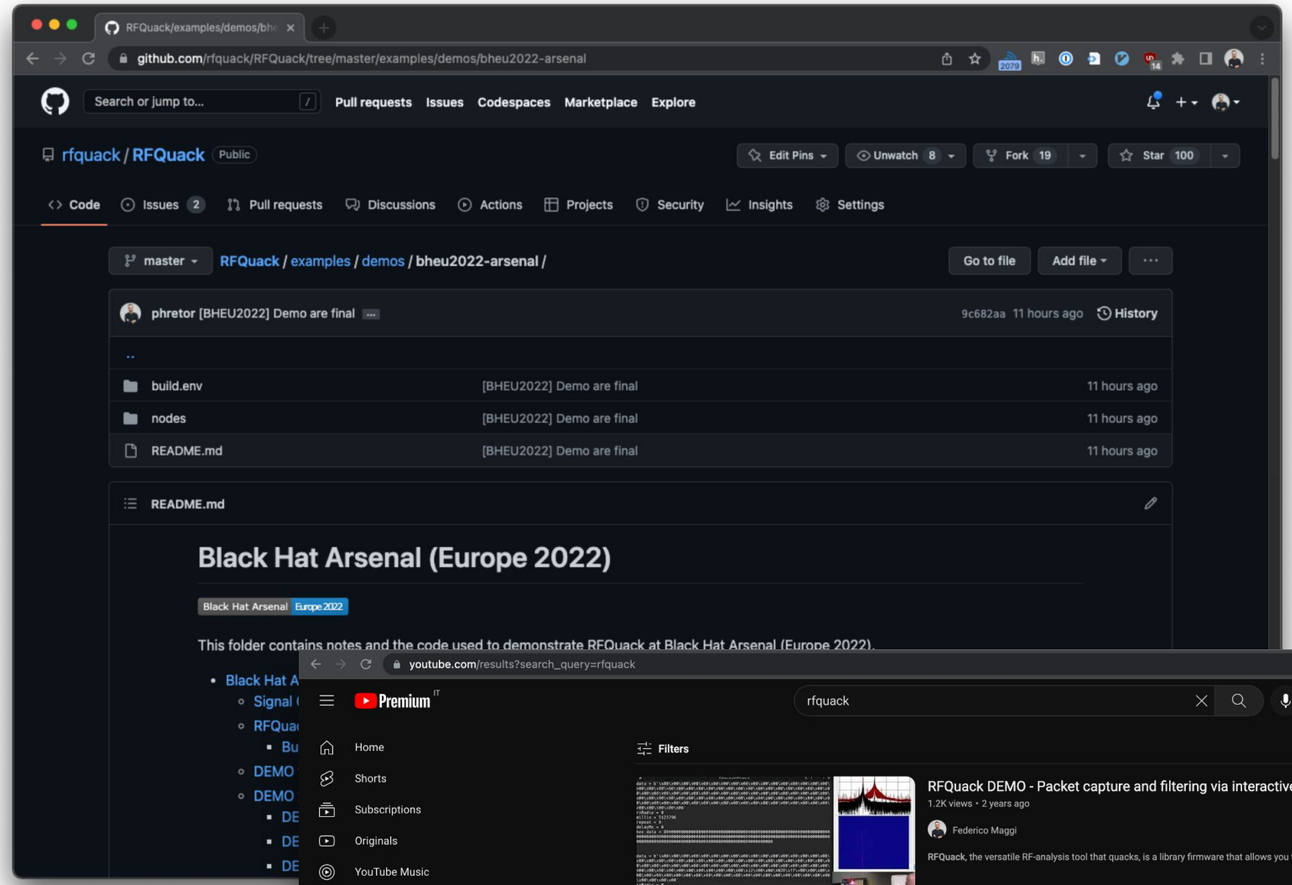
The increased adoption of wireless communication puts security research on the front line. Previous work has showed that both legacy [7] and newer-generation protocols (e.g., LoRaWAN [5]) require in-depth security auditing of the RF protocols. The impact of vulnerabilities on legacy protocols are particularly relevant, because they can affect industrial devices, which have long life spans (decades), and thus may never be patched until replacement.

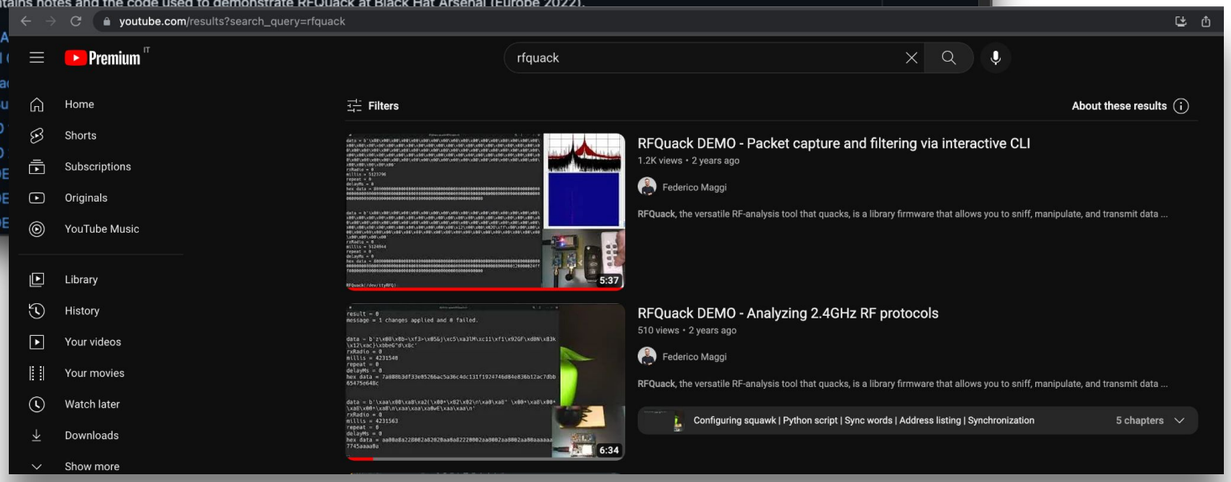
After an almost-mandatory, blind replay-attack test with an SDR, the typical workflow to analyze an unknown wireless protocol begins with a quick assessment using RF dongles (embedded RF transceivers), trying to sniff messages and "play around" with them. If the RF dongle fully supports the lower communication layers of our target protocol, then we have immediate access to the payload for further analysis. However, there exist many low-layer implementations, each with their own peculiarities, especially if we're looking at industrial applications. At this point we need an SDR to capture the signal and analyze it offline, with the goal of reverse engineering most of the lower communication layers. Once we have complete knowledge of the protocol, we can start looking for flaws. At this point it is not uncommon to bring RF dongles back in the game, because when the protocol is fully known, it's much more reliable to use a hardware transmitter to forge messages. Or maybe we need to quickly build a custom dongle, because we need a peculiar transceiver that supports a very uncommon modulation scheme.

Even if offline signal analysis is easier than in the past—thanks to advanced SDR tools like URH [11]—it's still quite challenging and error-prone to develop full, dynamic, precise, reliable transceivers. We wish we had flexible RF dongles that can be quickly reconfigured and adapted to support virtually any protocol, like with SDRs. This is what motivated us to develop and release RFQuack[1], which we like to think of as "the Arduino for RF researchers."

RFQuack is a RF dongle *system* with an extensible hardware and software platform that provides a solid foundation to develop custom RF dongles (see Figure 1) for wireless reverse engineering. While allowing full manual control, RFQuack includes ready-to-use RF-analysis features such as an automatic frequency and bitrate estimation that detects and clamps on transmissions in real time (in under 33 ms, with a 20 kHz accuracy). It supports multiple radios simultaneously, including mixed sub-GHz and 2.4GHz ones, and virtually any embedded radio. Also, RFQuack makes it easy to program interactive RF protocols without changing the firmware: It includes a packet filtering and modification engine that runs on the dongle and

[1] https://github.com/rfquack





Community & Roadmap

Hardware prizes for PR contributors!

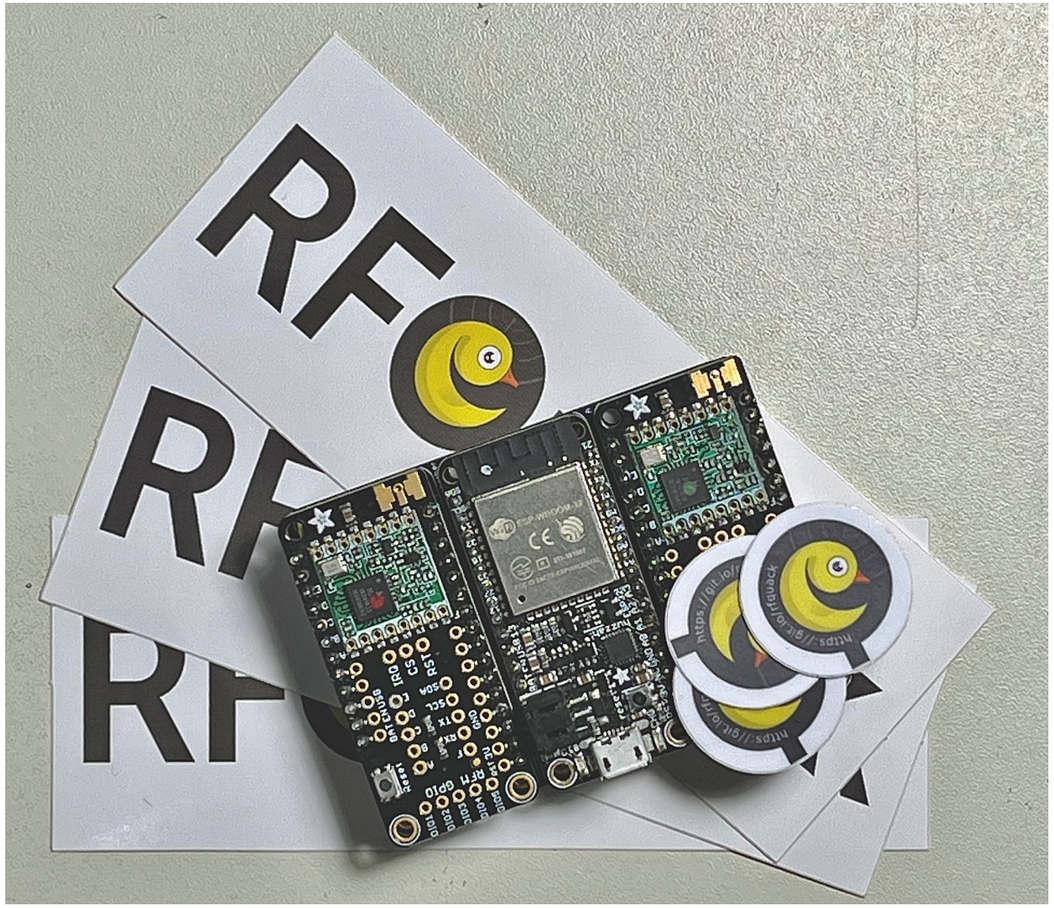
Ideas for the Future (sort of roadmap)

- RadioLib has supports for **11 radios**: RFQuack currently uses 3 drivers 😢
  - Most wanted: **LoRa**
- **Soldering**-based wiring is a huge limitation: **software-based routing of CS/IRQs** 🚀
- **URH** is a great GUI: RFQuack has a very open API 😇 https://github.com/jopohl/urh/discussions/963
- **Autotuning** only implemented in CC1101 (RF69 and nRF24 are waiting)
- Mixing **logging** & **control** data isn't a good idea for performance

Come join us!
https://rfquack.org/support/community/

Pull Requests == Prizes

Dual-LoRa (433MHz & 900MHz) RFQuack kit



- to **the first PR** 🥇 with a testable implementation of RadioLib-based LoRa driver for RFQuack
- must follow RFQuack and RadioLib development guidelines (check `CONTRIBUTING.md`)
- must come from a "legitimate looking" GitHub account
- prize assigned at maintainer's discretion and based on code quality.

https://github.com/rfquack/RFQuack

Other Ways to Contribute

- New hardware shields to adapt other radios (FeatherWing format)
  - compact shield for CC1101

- The KiCAD and Gerbers of existing shields are at https://github.com/rfquack/RFQuack
  - start from those

black hat
ARSENAL 2022

RFQuack

A Versatile, Modular, RF Security Toolkit

https://rfquack.org