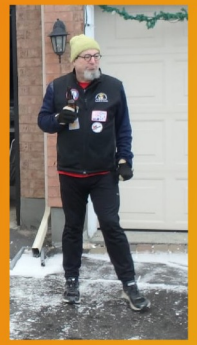
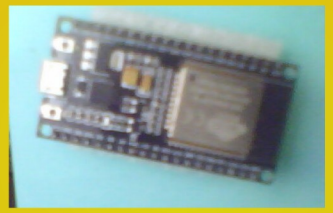
Building and Integrating Rust libraries for RIOT-OS, with Micropython support

(and FreeRTOS)

Michael Richardson
<mcr@sandelman.ca>
Ottawa, Canada



ESP32



Yoichi J. Nakaguro
<ynaka96@gmail.com>
Thailand

Overview of Talk

Goals

Technologies

Target Audience

RUST Challenges

Micropyhon Challenges

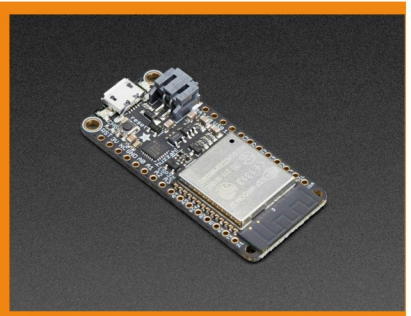
Demo github(s)

Goals of Project

1) get some secure onboarding upstream into RIOT-OS and FreeRTOS
2) Rust-based RFC8366 voucher library for embedded (no_std) and regular (std) use
3) Integrate with micropython as the top-level control loop, so that policy is in the end-user space
4) The next revolutionary product will come from amateurs, get them good tools

Technologies

- RFC8366 (vouchers)
- RFC8995 (BRSKI)
  - See https://brski.org/
- RUST
  - For memory safety,
  - Thread safety
  - It’s the future



- ESP32
  - https://www.adafruit.com/product/3405
- Specs:
  - 240 MHz dual core Tensilica LX6 microcontroller with 600 DMIPS
  - Integrated 520 KB SRAM
  - Integrated 802.11b/g/n HT40 Wi-Fi transceiver, baseband, stack and LWIP
  - Integrated dual mode Bluetooth (classic and BLE)
  - 4 MByte flash include in the WROOM32 module
- FreeRTOS
  - Because it’s there
- RIOT-OS
  - Because it’s cool

Target Audience

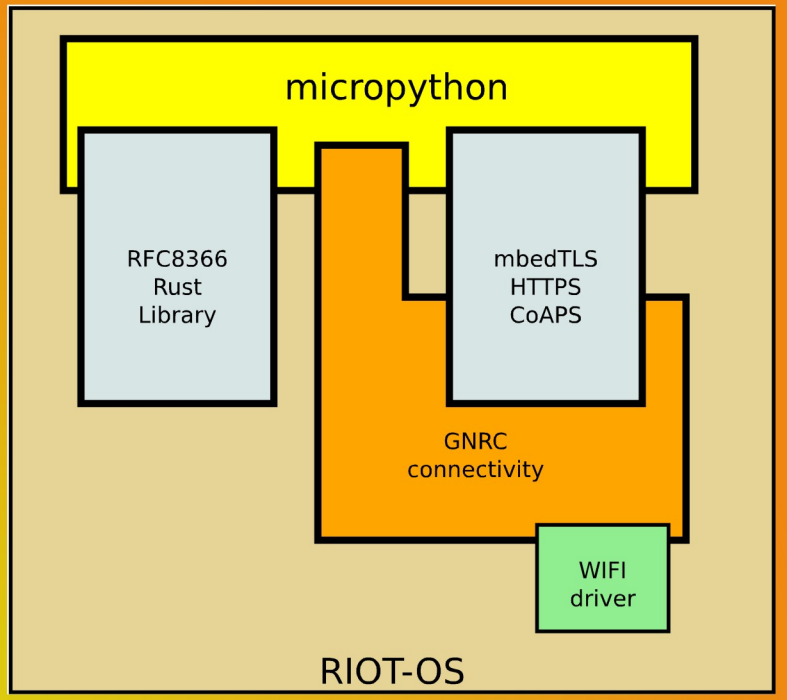
- The “Hello World” application
  - Batteries included security
  - Onboarding included
- Needs to be upstreamed everywhere, always available, always tested, compiled.

- The garage/small enterprise IoT experimenter
  - Doing a “proof of concept”
  - Winds up shipping/in production

- Target system code must be Royalty-Free (RF) license to reduce friction
- Seats, support, Registrar, onboarding, provisioning systems will cost money
  - But, open standards, people can roll their own, if they do not like ours.

Project Architecture

- Provide a series of libraries to do heavy lifting
- Keep top-level policy in python, where end developers can see and change the policy
- Provide reference python code to do onboarding
  - Provide to call home, softAP, and other fallback methods



ESP32 challenges (opportunities)

- Provisioning of IDevID into devices.
  - Looking for standardized way to store private key, certificates
  - Trust Anchor for SUIT
- Needs to be compatible with industry trends for provisioning.
  - Many different methods, but needs to result in same layout and content
- First firmware burn should be SUIT based.
- Provisioned anchors and configuration space perhaps should be CBOR?
- Xtensa has some secured/measured boot mechanisms which need further examination

Project Status

The Good

- RUST voucher library 90%
- Front-end-loaded **integration** to FreeRTOS and RIOT-OS
- Many libraries adapted and made no_std
- Using QEMU-Xtensa, but also RIOT-OS native build for testing

The Bad

- Micropython not connected to gnrc network on RIOT-OS
  - (yes on FreeRTOS)
- Cargo build not integrated into RIOT-OS, done by linking in Cargo “staticlib”, that is, a lib.a

RUST integration challenges

- Dependencies (no_std!). Many useful crates in Crates.io are **not** no_std aware; we often need to adapt them with a clear scope.

- Looking for good compromise between no_std and std: **“semi_std”** for e.g. dynamic containers, smart pointers, basic IO operations.

- We made available an interface crate “mcu-if” (details in Case Study)

```
#![no_std]

use mcu_if::println;
use mcu_if::alloc::{boxed::Box, vec, vec::Vec};
use mcu_if::core2::io::{self as io, Cursor, Seek, SeekFrom, Write};
```

- Cargo builds a static library, and then RIOT-OS links from this.

```
INCLUDES += -I$(CURDIR)/../include
ARCHIVES += $(CURDIR)/../target/xtensa-esp32-none-elf/release/librustmod.a
```

RIOT-OS vs FreeRTOS

ANIMA Minerva perspective

	RIOT-OS xtensa	FreeRTOS xtensa
Framework	RIOT 2021.07	ESP-IDF 4.2 (by Espressif)
Build type	esp32, **native**	esp32
Networking	**GNRC**	LwIP
in micropython	**We plan to working on this!**	Works (WIFI, LAN)
esp32.bin	~100 KB (small and build is fast)	~300 KB
C libraries (libc)	Compatible: io – printf, mem – malloc/free, etc.	
C libraries (crypto)	mbedtls coming	Has mbedtls
Rust	**Compatible!** We verified no_std/”semi_std” Rust works fine	

RIOT ‘native’ board and testing

MCU emulators

RIOT board type	esp32 	esp32	native
Arch	xtensa	xtensa	x86
MCU	real device	qemu-xtensa	32-bit Linux process
Memory emulation	~300 KB	~300 KB; **OK**	?
SPI emulation	Perfect (of course)	**OK (dev/debug)**	N/A
dev/debug cycle	**SLOW (need flashing)**	OK	**Great**
Notes	- net if: WIFI, (LAN) - CI: near impossible - Production test: **Yes!**	- net if: LAN - net debug tap: OK - Rust toolchain setup: Hard - CI: **Great**	- net if: LAN - net debug tap: **Great** - GNRC experiments: **Yes!** - Rust toolchain: **Easy** - CI: **Great**

Case study: 0) reference implementation repo

Name will probably change

iot-rust-module-studio

Robust IoT development with Rust and RIOT-OS.

Languages

Rust 60.0% Makefile 32.7%

Shell 7.3%

Repository map

```
/
  README.md
  crates/                 .... 💡 currently supports mcu's specific to esp32 (and Linux native) only
    mcu-emu               .... emulator runner (`qemu-system-xtensa` or RIOT native board binary)
    mcu-if                .... "semi_std" interface on top of bare `no_std`
  examples/
    esp32-no_std          .... bare `no_std` firmware with a Rust module
    xbd-base              .... cross-`BOARD` (esp32/native) firmware with minimal 'librustmod.a'
    xbd-micropython       .... cross-`BOARD` firmware featuring MicroPython with 'libvoucher.a'
    ...
```

1 2 3

Environments

Ubuntu 20.04 is supported and also being used for CI.

https://github.com/AnimaGUS-minerva/iot-rust-module-studio

Case study: 0) clone and init the repo

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

1

Getting started

After cloning the repo, first, set up the pre-configured RIOT/ESP32 toolchain:

```
$ make init
```

2

3

CI #54

Test (linux)

succeeded 18 hours ago in 7m 54s

Set up job	2s
Run actions/checkout@v1	1s
Install rustup	2s
make init-rust-xtensa	2m 3s
make init-esp-idf	36s
make init-riot-xtensa	14s
make init-qemu-xtensa	0s
make init-rust-i686-nightly	23s
make init-rust-x86_64-nightly	10s
Run test	4m 23s
Complete job	0s

Case study: 1) examples/esp32-no_std - structure

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── rustmod.h
├── riot
│   ├── Makefile
│   └── main.c
└── src
    └── lib.rs
```

```toml
[package]
name = "esp32-no_std"
version = "0.1.0"
edition = "2018"
authors = ["ANIMA Minerva toolkit"]

[lib]
name = "rustmod"
crate-type = ["staticlib"]
```

1

```
APPLICATION = riot
BOARD ?= native
RIOTBASE ?= $(CURDIR)/../../../RIOT

INCLUDES += -I$(CURDIR)/../include
ARCHIVES += $(CURDIR)/../target/xtensa-esp32-none-elf/release/librustmod.a

include $(RIOTBASE)/Makefile.include
```

2

```c
#include <stdio.h>
#include "rustmod.h"

int main(void)
{
    int input = 4;

    printf("riot: before calling rustmod\n");
    int output = square(input);
    printf("riot: after calling rustmod\n");

    printf("riot: square(%d) -> %d\n", input, output);

    return 0;
}
```

4

```rust
#![no_std]

#[no_mangle]
pub extern fn square(input: i32) -> i32 {
    input * input
}

extern "C" {
    fn abort() -> !;
}

#[panic_handler]
fn panic(_info: &core::panic::PanicInfo) -> ! {
    unsafe { abort(); }
}
```

3

To be refactored into “mcu-if” crate (later)

Case study: 1) examples/esp32-no_std – build and run `esp32` firmware

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── rustmod.h
├── riot
│   ├── Makefile
│   └── main.c
└── src
    └── lib.rs
```

1

make

make run

```
   text    data     bss     dec     hex filename
  46452    4404    6312   57168    df50 /home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/esp32-no_std/riot/bin/esp32-wroom-32/riot.elf
make[4]: Leaving directory '/home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/esp32-no_std/riot'
esptool.py v3.1-dev
Merged 1 ELF section
-rw-r--r-- 1 runner docker 89136 Sep  5 10:12 ../riot.esp32.bin
```

We can flash this onto the real ESP32 device too!

2

```
I (745) boot: Partition Table:
I (745) boot: ## Label            Usage          Type ST Offset   Length
I (746) boot:  0 nvs              WiFi data        01 02 00009000 00006000
I (747) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (747) boot:  2 factory          factory app      00 00 00010000 00100000
I (753) boot: End of partition table
I (756) esp_image: segment 0: paddr=0x00010020 vaddr=0x3f400020 size=0x01610 (  5648) map
I (765) esp_image: segment 1: paddr=0x00011638 vaddr=0x3ffb0000 size=0x01134 (  4404) load
I (776) esp_image: segment 2: paddr=0x00012774 vaddr=0x40080000 size=0x04310 ( 17168) load
I (789) esp_image: segment 3: paddr=0x00016a8c vaddr=0x400c0000 size=0x00064 (   100) load
I (796) esp_image: segment 4: paddr=0x00016af8 vaddr=0x00000000 size=0x09518 ( 38168)
I (811) esp_image: segment 5: paddr=0x00020018 vaddr=0x400d0018 size=0x05bf4 ( 23540) map
I (833) boot: Loaded app from partition at offset 0x10000
I (833) boot: Disabling RNG early entropy source...

main(): This is RIOT! (Version: 2021.04)
riot: before calling rustmod
riot: after calling rustmod
riot: square(4) -> 16
```

3

Case study: 2) examples/xbd-base - structure

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── rustmod.h
├── riot
│   ├── Makefile
│   └── main.c
└── src
    └── lib.rs
```

1

```rust
#![no_std]
#![feature(panic_info_message)]
#![feature(alloc_error_handler)]

pub use libc_print::libc_println as println;
pub use core2;

extern "C" {
    fn abort() -> !;
    fn malloc(sz: u32) -> *mut core::ffi::c_voi;
    fn free(ptr: *mut core::ffi::c_void);
}
```

2

```toml
[package]
name = "xbd-base"
version = "0.1.0"
edition = "2018"
authors = ["ANIMA Minerva toolkit"]

[lib]
name = "rustmod"
crate-type = ["staticlib"]

[dependencies]
mcu-if = { path = "../../crates/mcu-if" }
```

`mcu-if` implements "*semi_std*" features

- `#[panic_handler]`
- `#[global_allocator]`
- `#[alloc_error_handler]`
- `println!()`
- `core2::io::*` (substitute of `std::io::*`)

3

```rust
#![no_std]

use mcu_if::println;
use mcu_if::alloc::{boxed::Box, vec, vec::Vec};
use mcu_if::core2::io::{self as io, Cursor, Seek, SeekFrom, Write};

#[no_mangle]
pub extern fn square(input: i32) -> i32 {
    println!("[src/lib.rs] square(): input: {}", input);

    demo_alloc();
    demo_io();

    input * input
}

//

fn demo_alloc() {
    println!("Box::new(42): {:?}", Box::new(42));
    println!("Box::new([0; 10]): {:?}", Box::new([0; 10]));
    println!("Vec::from([0, 1, 2]): {:?}", Vec::from([0, 1, 2]));
    println!("vec![0, 1, 2]: {:?}", vec![0, 1, 2]);
}
```

4

Case study: 2) examples/xbd-base – build and run `native` firmware

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── rustmod.h
├── riot
│   ├── Makefile
│   └── main.c
└── src
    └── lib.rs
```

1

make build-native

```
   text    data     bss     dec     hex filename
  53329    1460   47792  102581   190b5 /home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/xbd-base/riot/bin/native/riot.elf
make[5]: Leaving directory '/home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/xbd-base/riot'
ldd ./riot/bin/native/riot.elf && file ./riot/bin/native/riot.elf
        linux-gate.so.1 (0xf7f97000)
        libdl.so.2 => /lib32/libdl.so.2 (0xf7f5e000)
        libgcc_s.so.1 => /lib32/libgcc_s.so.1 (0xf7f3f000)
        libc.so.6 => /lib32/libc.so.6 (0xf7d54000)
        /lib/ld-linux.so.2 (0xf7f99000)
./riot/bin/native/riot.elf: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=bf1a758cc79a77f4adffb7277c6835fe3aa09b57, for GNU/Linux 3.2.0, with debug_info, not stripped
```

2

make run-native

```
RIOT native interrupts/signals initialized.
LED_RED_OFF
LED_GREEN_ON
RIOT native board initialized.
RIOT native hardware initialization complete.

main(): This is RIOT! (Version: 2021.04)
riot: RIOT_BOARD: native
riot: RIOT_MCU: native
riot: before calling rustmod
[src/lib.rs] square(): input: 4
Box::new(42): 42
Box::new([0; 10]): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Vec::from([0, 1, 2]): [0, 1, 2]
vec![0, 1, 2]: [0, 1, 2]
buff: Cursor { inner: [0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9], pos: 15 }
check: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
riot: after calling rustmod
riot: square(4) -> 16
```

3

Case study: 2) examples/xbd-base - build and run `esp32` firmware

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── rustmod.h
├── riot
│   ├── Makefile
│   └── main.c
└── src
    └── lib.rs
```

1

make build-esp32

make run-esp32

```
   text    data     bss     dec     hex filename
  65261    4404    6312   75977   128c9 /home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/xbd-base/riot/bin/esp32-wroom-32/riot.elf
make[5]: Leaving directory '/home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/xbd-base/riot'
esptool.py v3.1-dev
Merged 1 ELF section
-rw-r--r-- 1 runner docker 103104 Sep  5 10:14 ../riot.esp32.bin
```

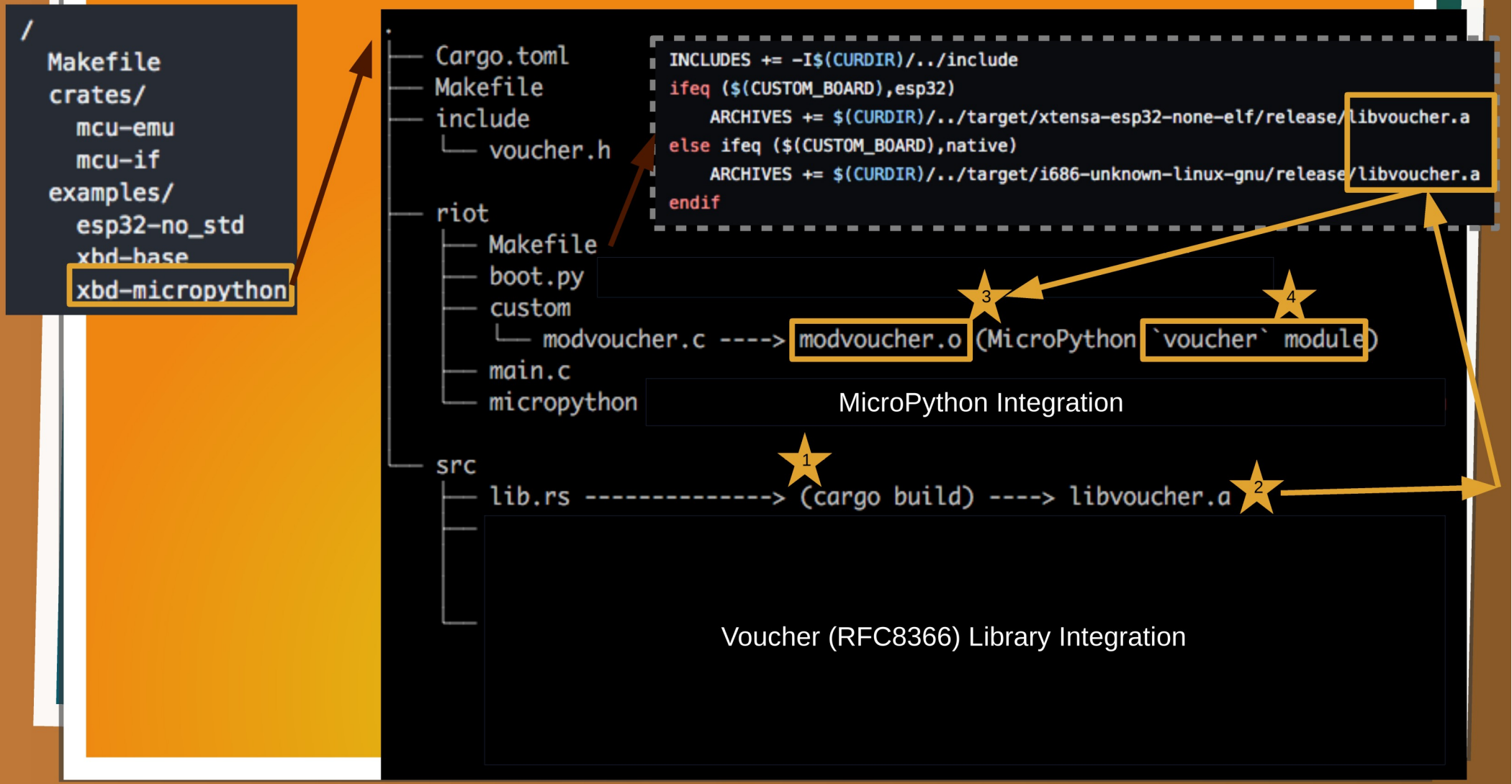
2

```
I (747) boot: End of partition table
I (749) esp_image: segment 0: paddr=0x00010020 vaddr=0x3f400020 size=0x028e0 ( 10464) map
I (760) esp_image: segment 1: paddr=0x00012908 vaddr=0x3ffb0000 size=0x01134 (  4404) load
I (769) esp_image: segment 2: paddr=0x00013a44 vaddr=0x40080000 size=0x04328 ( 17192) load
I (782) esp_image: segment 3: paddr=0x00017d74 vaddr=0x400c0000 size=0x00064 (   100) load
I (788) esp_image: segment 4: paddr=0x00017de0 vaddr=0x00000000 size=0x08230 ( 33328)
I (805) esp_image: segment 5: paddr=0x00020018 vaddr=0x400d0018 size=0x09284 ( 37508) map
I (828) boot: Loaded app from partition at offset 0x10000
I (829) boot: Disabling RNG early entropy source...

main(): This is RIOT! (Version: 2021.04)
riot: RIOT_BOARD: esp32-wroom-32
riot: RIOT_MCU: esp32
riot: before calling rustmod
[src/lib.rs] square(): input: 4
Box::new(42): 42
Box::new([0; 10]): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Vec::from([0, 1, 2]): [0, 1, 2]
vec![0, 1, 2]: [0, 1, 2]
buff: Cursor { inner: [0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9], pos: 15 }
check: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
riot: after calling rustmod
riot: square(4) -> 16
```

3

Case study: 3) examples/xbd-micropython - structure



Case study: 3) examples/xbd-micropython – MicroPython integration

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
main.c
micropython .......... fork of https://github.com/kaspar030/micropython
src
```

MicroPython status and our actions

- [upstream] micropython/**ports/riot** is missing; while [kaspar030] micropython/**ports/riot** is under dev

```
>>> Updated kaspar's branch for latest v1.16 micropython
>>> Extended kaspar's branch to be multi-arch (esp32 tested)
>>> Probably PR to kaspar and help merge ports/riot into
    upstream micropython!
```

- Recent v1.16 release has ESP32 ethernet support that works well in FreeRTOS with ESP-IDF v4.2

```
>>> Planning to connect RIOT's GNRC layer to Micropython's
    ethernet LAN api
>>> For this to happen, we made sure our codebase is compatible
    with both esp32 AND native.
    RIOT's native board is awesome for dev/debug purpose!
```

Case study: 3) examples/xbd-micropython – Voucher (RFC8366) Library integration

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
$ cargo tree

xbd-micropython
├── cose v0.1.4 (fork of franziskuskiefer/cose-rust 🔗 )
│   └── moz_cbor v0.1.2 (fork of franziskuskiefer/cbor-rust 🔗 )
│       └── core2 v0.3.0-alpha.1 (fork of technocreatives/core2 🔗 )
│           └── memchr v2.4.1
│
└── mcu-if ☺
    ├── core2 v0.3.0-alpha.1 (fork of technocreatives/core2 🔗 )
    │   └── memchr v2.4.1
    └── libc-print v0.1.16 (fork of mmastrac/rust-libc-print 🔗 )
```

💡 minimize # of deps, while...

- using **awesome 3rd-party crates** ❤
- retaining necessary std-like features ☺

```
src
├── lib.rs --------------> (cargo build) ----> libvoucher.a
├── tests
│   ├── mod.rs
│   └── voucher.rs
└── voucher ................. Rust-based RFC8366 Voucher lib prototype
    ├── cose_data.rs ............ COSE/CBOR voucher decoder
    ├── mod.rs
    ├── validate.rs ............. mbedtls-based ECDSA validator (WIP)
    └── validate_std.rs ......... mbedtls-based ECDSA debug with `std`
```

Case study: 3) examples/xbd-micropython – run test and REPL (native build)

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
.
├── Cargo.toml
├── Makefile
├── include
│   └── voucher.h
└── riot
    ├── Makefile
    ├── boot.py
    ├── custom
    │   └── modvou
    └── main.c
```

1

```
main(): This is RIOT! (Version: 2021.04)
sizeof(mp_heap): 131072
-- Executing boot.py
dir(voucher): ['__name__', 'debug', 'demo', 'get_masa_pem_F2_00
get_voucher_jada', 'test_ffi', 'validate']
[custom/modvoucher.c] mod_demo(): ^^
voucher.demo(): None
[test] bytes from list : ✅
[test] list from bytes : ✅
mod_test_ffi(): ^^
input: 4 output: 16
sizeof(tuple): 28
[test] voucher.test_ffi : ✅
[test] voucher.get_voucher_jada : ✅
[test] voucher.get_voucher_F2_00_02 : ✅
[test] voucher.get_masa_pem_F2_00_02 : ✅
[test] no MemoryError for simple ops : ✅
```

2

3 Debug in REPL

```
>>> vch = voucher.get_voucher_F2_00_02()
vch = voucher.get_voucher_F2_00_02()
>>> pem = voucher.get_masa_pem_F2_00_02()
pem = voucher.get_masa_pem_F2_00_02()
>>> vch
vch
b'\xd2\x84C\xa1\x01&\xa0Y\x02\xb7\xa1\x19\t\x93\xa5\x01flogged\x02\xc1\x1a_V\xd1w\x0bq00-D0-E5
-F2-00-02\x07vXSyF4LLIiqU2-OGk6lFCAg\x08y\x02tMIIB0TCCAVagAwIBAgIBAjAKBggqhkjOPQQDAzBxMRIwEAYK
CZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xQDA+BgNVBAMMNyM8U3lzdGVtVmFyaWFibGU6MH
gwMDAwMDAwNGY5MTFhMD4gVW5zdHJ1bmcgRm91bnRhaW4gQ0EwHhcNMTcxMTA3MjM0NTI4WhcNMTkxMTA3MjM0NTI4WjBD
MRIwEAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xEjAQBgNVBAMMCWxvY2FsaG9zdDBZMB
MGByqGSM49AgEGCCqGSM49AwEHA0IABJZlUHI0up/l3eZf9vCBb+lInoEMEgc7Ro+XZCtjAI0CD1fJfJR/hIyyDmHWyYiN
FbRCH9fyarfkzgX4p0zTizqjDTALMAkGA1UdEwQCMAAwCgYIKoZIzj0EAwMDaQAwZgIxALQMNurf8tv50lROD5DQXHEOJJ
NW3QV2g9QEdDSk2MY+AoSrBSmGSNjh4olEOhEuLgIxAJ4nWfNw+BjbZmKiIiUEcTwHMhGVXaMHY/F7n39wwKcBBSOndNPq
CpOELl6bq3CZqQ==X@c\xcc\x82:4\xb9d\xad\xc85\xb5\x8e.\xe1\xe7\xe3\x00\x88\xad\xe6\x89o\x94\xb1:
\xc70d>\x96`\xb5\xa94$\xf3\xc9\xd8\xa0\x9a\xb5z\x01\x13\xa4\x06rx\x84v:*\xd0KO\xabOo\xb8\xbc\x
b3.\xfaG'
>>> voucher.validate(vch, pem)
voucher.validate(vch, pem)
@@ validating raw_voucher with pem: [len=771] [len=684]
⚠ECDSA verification under `no_std` is WIP!  It's `std` prototype (validate_std.rs) works thou
gh; try `make test-std-debug`.
False
>>> voucher.debug(vch)
voucher.debug(vch)
======== cose dump
  signature_type: ES256
  signature: [len=64] [99, 204, 130, 58, 52, 185, 100, 173, 200, 53, 181, 142, 46, 225, 231, 2
27, 0, 136, 173, 230, 137, 111, 148, 177, 58, 199, 48, 100, 62, 150, 96, 181, 169, 52, 83, 243
, 201, 216, 160, 154, 181, 122, 1, 19, 164, 6, 114, 120, 132, 118, 58, 42, 208, 75, 79, 171, 7
9, 111, 184, 188, 179, 46, 250, 71]
  signer_cert: [len=0] []
  to_verify: [len=715] [132, 106, 83, 105, 103, 110, 97, 116, 117, 114, 101, 49, 67, 161, 1, 3
8, 64, 89, 2, 183, 161, 25, 9, 147, 165, 1, 102, 108, 111, 103, 103, 101, 100, 2, 193, 26, 95,
```

4 Rust-based COSE decoder in action!

Case study: 3) examples/xbd-micropython – run test (esp32 build)

```
/
  Makefile
  crates/
    mcu-emu
    mcu-if
  examples/
    esp32-no_std
    xbd-base
    xbd-micropython
```

```
   text    data     bss     dec     hex filename
 229683    4688  143320  377691   5c35b /home/runner/work/iot-rust-module-studio/iot-rust-module-
studio/examples/xbd-micropython/riot/bin/esp32-wroom-32/riot.elf
make[5]: Leaving directory '/home/runner/work/iot-rust-module-studio/iot-rust-module-studio/examples/xbd-
micropython/riot'
esptool.py v3.1-dev
Merged 1 ELF section
-rw-r--r-- 1 runner docker 234496 Sep  5 10:15 ../riot.esp32.bin
```

We can flash this onto the real ESP32 device too!

1

```
I (728
I (730
I (750
I (763
I (777
I (827) esp_image: segment 4: paddr=0x0004751c vaddr=0x40083020 size=0x01c48 (  7240) load
I (837) esp_image: segment 5: paddr=0x0004936c vaddr=0x400c0000 size=0x00064 (   100) load
I (856) boot: Loaded app from partition at offset 0x10000
I (856) boot: Disabling RNG early entropy source...

main(): This is RIOT! (Version: 2021.04)
sizeof(mp_heap): 131072
-- Executing boot.py
dir(voucher): ['__name__', 'debug', 'demo', 'get_masa_pem_F2_00_02', 'get_voucher_F2_00_02',
'get_voucher_jada', 'test_ffi', 'validate']
[custom/modvoucher.c] mod_demo(): ^^
voucher.demo(): None
[test] bytes from list : ✅
[test] list from bytes : ✅
mod_test_ffi(): ^^
input: 4 output: 16
sizeof(tuple): 28
[test] voucher.test_ffi : ✅
[test] voucher.get_voucher_jada : ✅
[test] voucher.get_voucher_F2_00_02 : ✅
[test] voucher.get_masa_pem_F2_00_02 : ✅
[test] no MemoryError for simple ops : ✅
```

2

Conclusions

- Ongoing work

  - Connect GNRC to micropython in `native` (then `esp32`)

  - Compact no_std Rust interface/binding to the *latest* mbedtls library (we recently saw mbedtls pkg coming to RIOT!)

- Please expect pull requests
- Looking for discussion of security architecture
  - How much can we do on multiple boards?
  - Can we do variations with some abstraction API?
  - Some boards have Bootloaders, others do not