Tock on Armv8-M architecture + Wi-Fi

Darius Andrei Jipa - darius.jipa@oxidos.io

Irina Nita - irina.nita@oxidos.io

Agenda

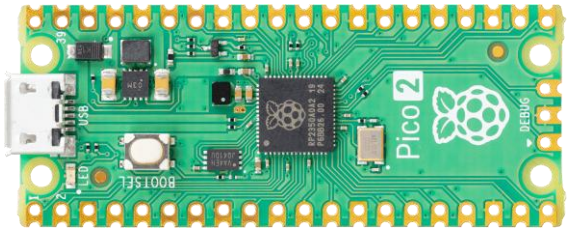
- Motivation for an Armv8-M port
- Porting Tock OS to the Raspberry Pi Pico 2
- Pico 2 Metadata Header
- Armv7-M vs Armv8-M MPU
- Designing the Armv8-M MPU driver
- WiFi

Motivation for an Armv8-M port

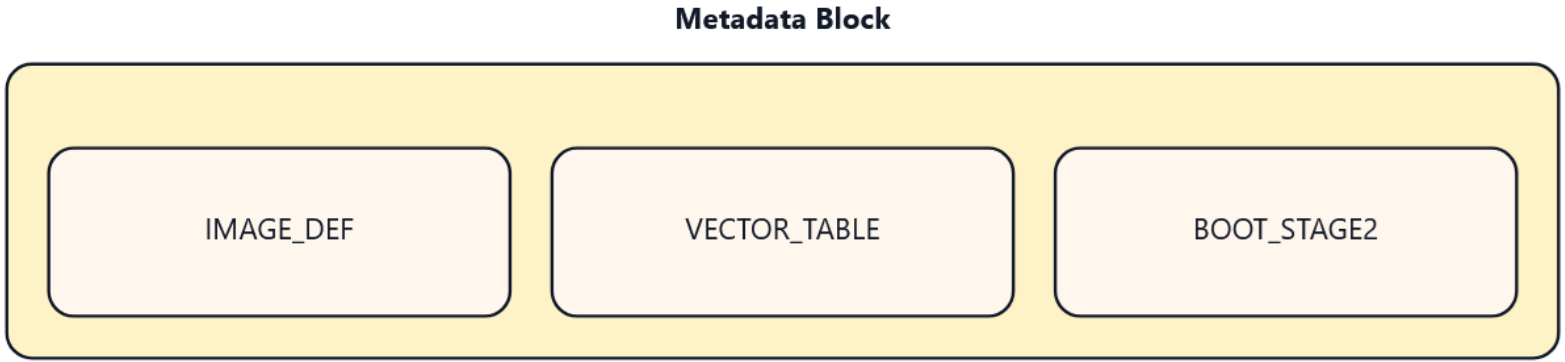
- Built-in support for Armv6-M and Armv7-M
- Next milestone: Armv8-M
  - Security additions:
    - TrustZone: hardware-enforced secure vs. non-secure partitions
    - Updated Memory Protection Unit (MPU): fine-grained region controls
  - Rapidly gaining traction
    - Raspberry Pi Pico RP2350: Dual-core Cortex-M33
    - NXP LPC5500 series: Cortex-M33 MCUs already shipping in volume

Porting Tock OS to the Raspberry Pi Pico 2

- Why Pico 2?
  - The Pico 2 has a dual-core Cortex-M33 (RP2350)
  - Cheap (from $5)
- What is implemented
  - Core boot sequence & runtime setup
  - “Everyday” peripherals: UART, GPIO, Timers, SPI, I²C
- Key challenges
  - RP2350’s mandatory metadata block at flash start
  - Rewriting MPU support for Armv8-M



Pico 2 Metadata Header



Armv7-M vs Armv8-M MPU

- Region shape
  - v7-M = power-of-two size, base aligned to size
  - v8-M = base/limit model; start & end 32-byte aligned; any size
- Overlap
  - v7-M allows overlaps (higher region number wins)
  - v8-M disallows overlaps
- Attributes model
  - v7-M sets attributes in RASR
  - v8-M uses AttrIndx → MAIR0/MAIR1 entries
- Subregions
  - v7-M has 8 subregions (SRD) per region
  - v8-M has no subregions

Designing the Armv8-M MPU Driver

- Add the Armv8-M MPU registers to the code and make a struct over them
- Implement the MPU trait over said struct
- Write the memory regions calculations leveraging the Armv8-M MPU flexibility

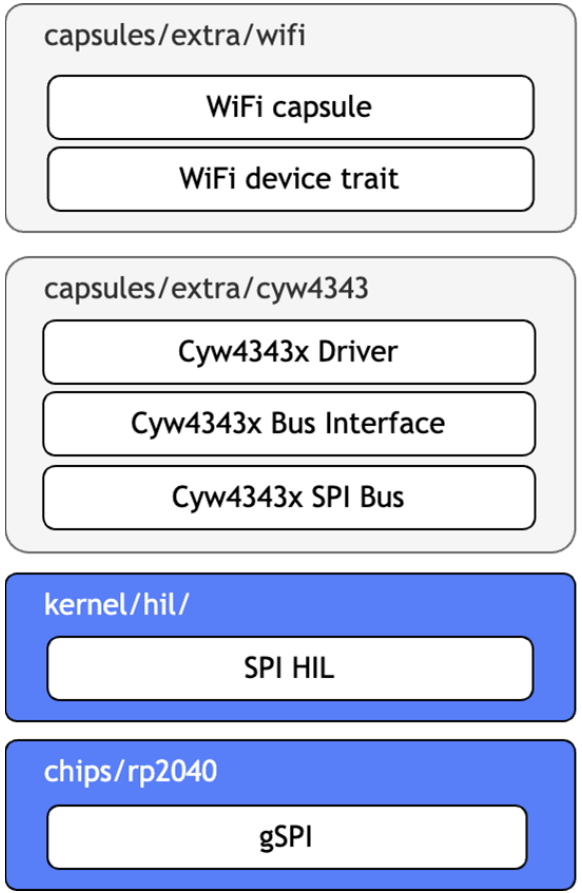
Tock and Wi-Fi

- Wi-Fi was a long awaited feature for Tock Networking
- Support for Ethernet already available, Wi-Fi was the next logical step
- Work has started on introducing a Wi-Fi capsule

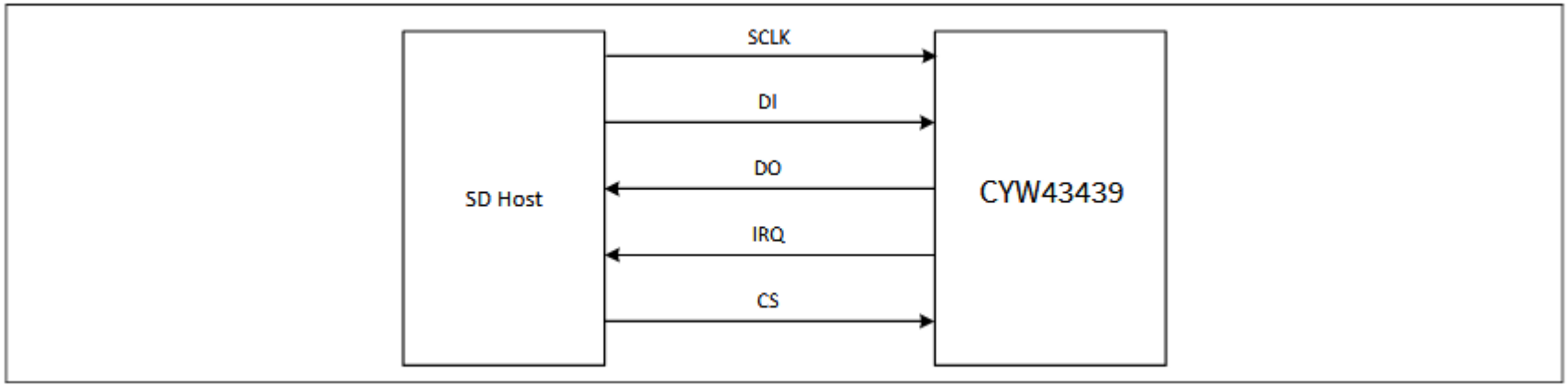
Raspberry Pi Pico W and the CYW43439 chip

- Raspberry Pi Pico W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW43439
- The communication between the RP2040 chip and the CYW43439 chip is done through gSPI

gSPI support



The gSPI protocol



CYW43439 datasheet

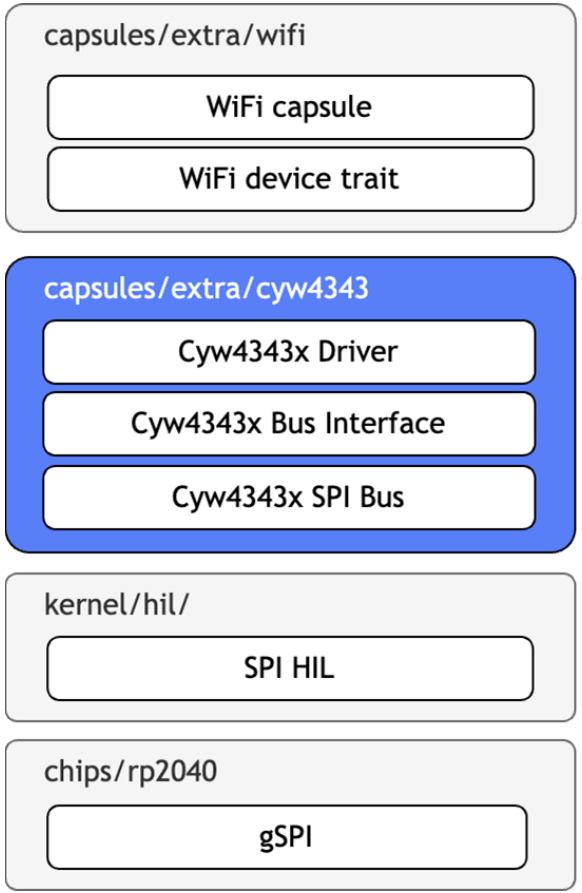
The PIO solution

- RP2040 introduces the PIO (Programmable Input/Output) peripheral
- It consists of 2 blocks with 4 state machines each that function independent of the CPU
- Programs are written with **pioasm** which is an assembly-like language
- Using PIO, the gSPI is implemented and it works like a separate peripheral

The gSPI PIO program

```
out     x, 32           side 0
out     y, 32           side 0
set     pindirs, 1      side 0
.wrap_target
out     pins, 1         side 0
jmp     x--, 3          side 1
set     pindirs, 0      side 0
nop                     side 0
in      pins, 1         side 1
jmp     y--, 7          side 0
wait    1 pin, 0        side 0
irq     nowait 0        side 0
.wrap
```

CYW4343x driver



CYW4343x bus

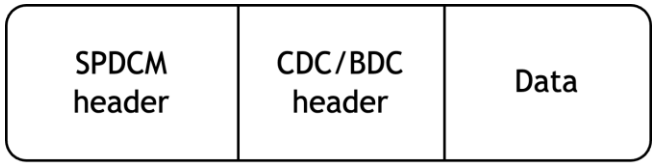
- Bus implementations sit on top of low-level protocols (SPI, SDIO) HIL implementations
- The HILs are not enough for defining a bus
  - Register configuration and command formats are different per protocol (but specific to the CYW4343x chip)
  - Inner state machines differ
- Chip-agnostic, but not protocol agnostic

CYW4343x bus

- Handle configuration and initialisation
- Provide read/write methods for:
  - Bus function, for accessing registers *(mostly protocol-dependent)*
  - Backplane function (interfacing with the inner SoC's address space, e.g. writing the firmware)
  - WLAN function (data packets)

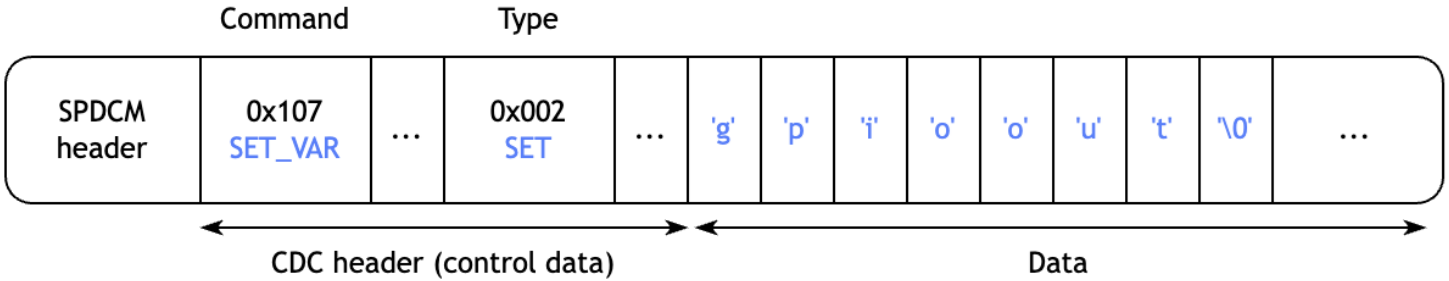
CYW4343x driver

- Sits on top of the generic CYW4343x bus
- Handles initialisation, including writing the firmware
- Constructs and parses *Cypress headers* over the bus WLAN function packets:
  - *SPDCM header* for sequencing packets
    - *BDC header* for data (Ethernet packets) and events
    - *CDC header* for control packets (IOCTL/IOVAR)

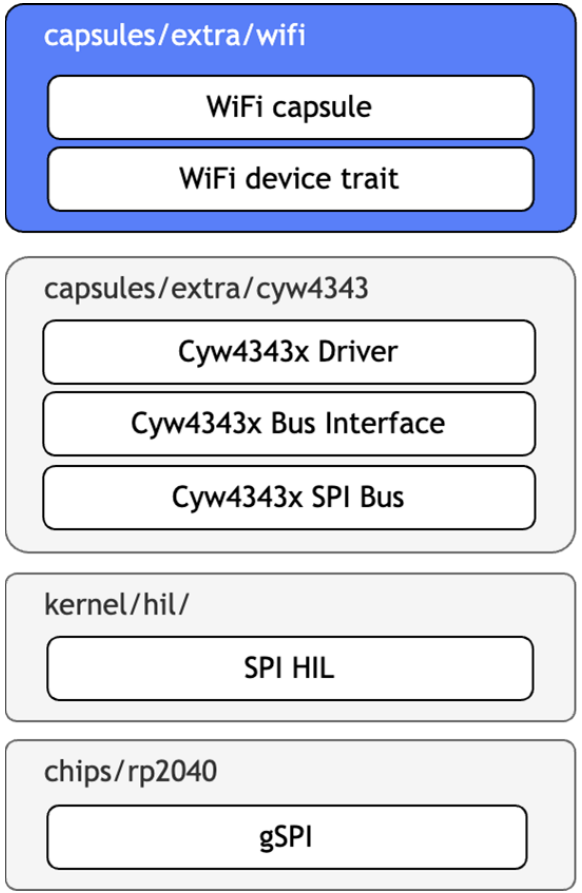


CYW4343x driver

- IOCTLs are used in configuring the WiFi chip
- Example of IOVAR packet:



WiFi capsule



WiFi capsule 🚧

- Initialising
- Network scanning with asynchronous updates
- Access point mode (AP)
- Station mode (STA)

Small bumps

- Proprietary firmware from Infineon
  - How do we include blobs in the Tock ecosystem?
- The first working iteration
  - Was based on a WiFi HIL implemented at chip driver level
  - For multiple chip support it would have required duplicating code
  - A lot of unnecessary unsafe code

Future plans for Wi-Fi

- Add drivers for the Infineon CYW4343W chip
  - Tock has code for a board containing said chip (CY8CPROTO-062-4343W)
  - It communicates with the CPU through the SDIO (Secure Digital Input Output)

Questions time!

Thank you for your attention!