arm

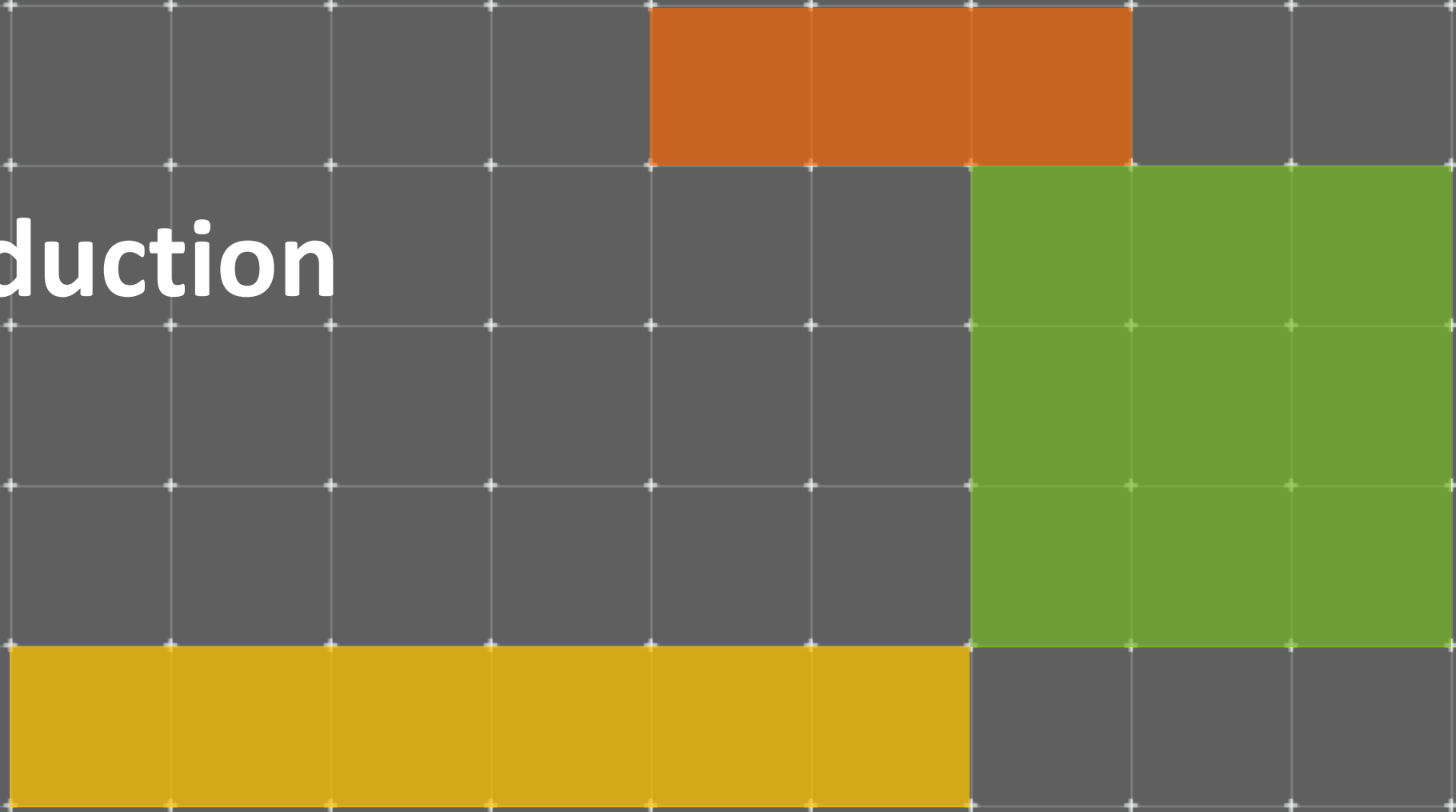
Mbed Silicon Partner Workshop

# HAL Improvements RTC

26th of March 2018

Paris, France

# Introduction



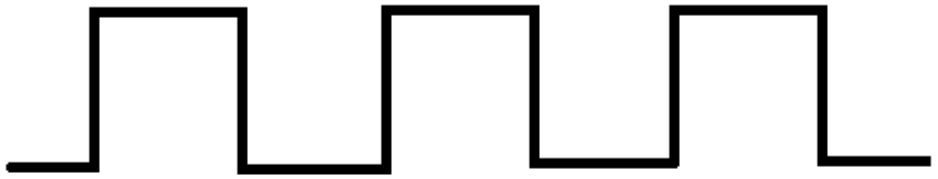


arm

# What is the Real Time Clock?


Low power, low resolution clock

Phrase *real time* is used to differentiate it from a typical digital clock which generates electrical pulses, but doesn't actually keep track of second or minutes.



vs




arm

# What is the Real Time Clock?


- It enables system to track time in human units
- Usually it continues to count over system resets
  - Date and time remains correct even after the system is reset
- RTC is often supplied with a dedicated battery to keep track of time during power loss
- There are two main varieties of RTC clocks
  - Second resolution
  - Sub-second resolution



arm

# Why do we need a Real Time Clock?


- To be able to carry out an action at the same time every day
  - Turn lights on or off
  - Allow access to a building
- Certificate revocation needs to know when the expiry date has passed.



arm

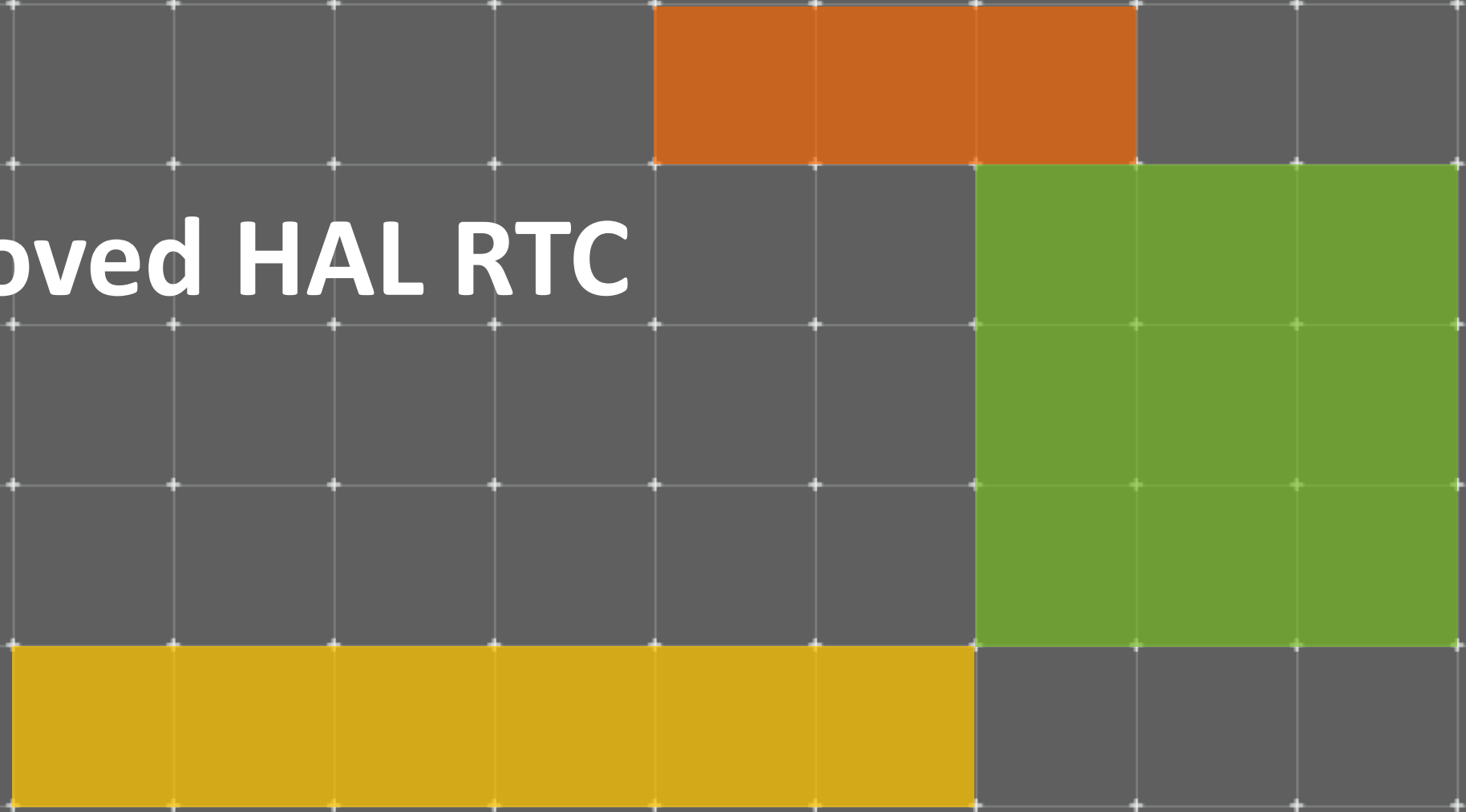
# RTC in Mbed OS


- Mbed OS retargets standard C library time functions
  - `time_t time(time_t *)` - returns number of seconds since the Epoch, 1970-01-01
  - `struct tm *localtime(const time_t *)` – get local time in current timezone
  - `time_t mktime(struct tm *timeptr)` – convert timezone time to seconds since Epoch
- Extended timestamps
  - 32-bit unsigned timestamp
  - 1st of January 1970 - 00:00:00 to 7th of February 2106 at 06:28:15
- Mbed OS APIs
  - https://os.mbed.com/docs/v5.7/reference/rtc.html
  - https://os.mbed.com/docs/v5.7/reference/time.html



arm

# Improved HAL RTC





arm

# Changes to HAL

- No changes to the API
- Updated specification
- Revised porting guide
- New validation test suite




arm

# Changes to HAL

## New specification

RTC has to keep counting through

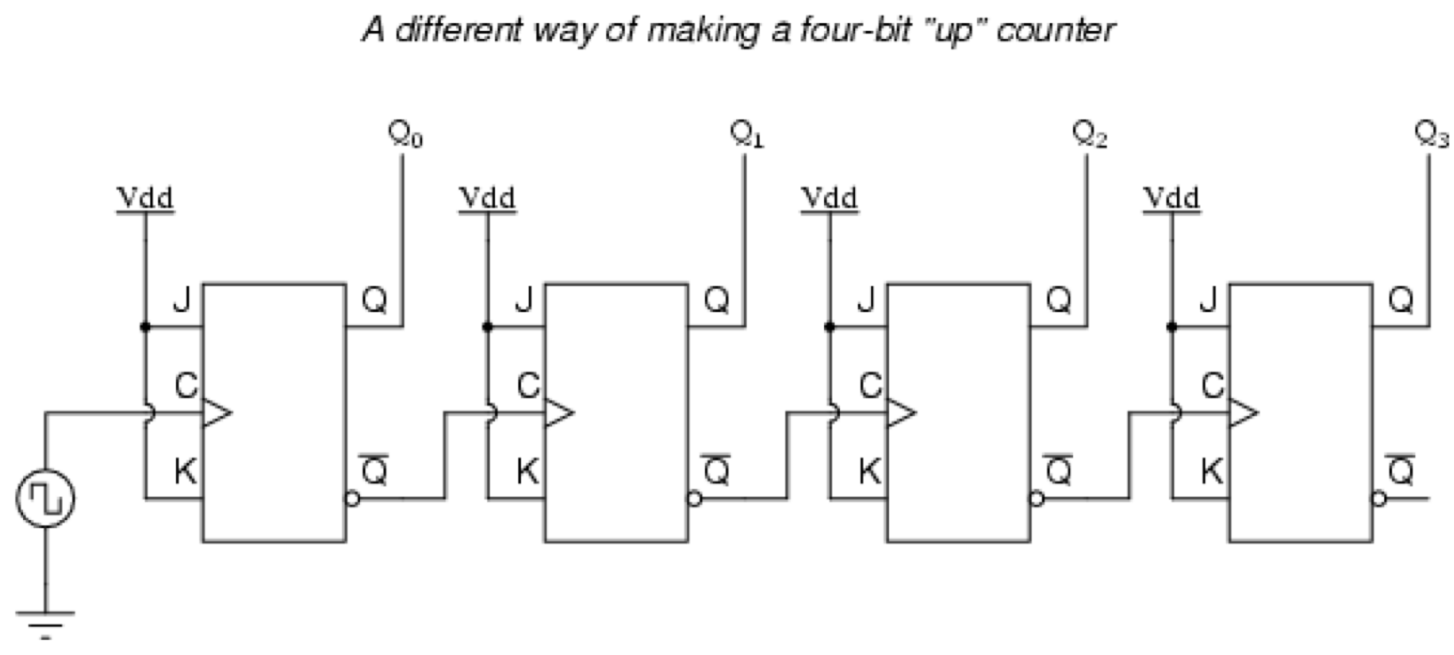
- `rtc_init()` and `rtc_free()` calls
- Software reset
- Sleep modes
- Shutdown mode

RTC accuracy is at least 10%

- The counting error has to be below 100ms for each passing second

# What to look out for? - Async / Ripple counters

- A ripple counter, sometimes referred to as an async counter clocks each subsequent flip-flop with the output of the previous
- Each clock "ripples" through the counter which gives it its name
- Must read the same value twice to ensure you are not in the middle of a "ripple"






arm

# What to look out for? - Overflow handling

- The RTC HAL doesn't specify the starting time of the RTC
- If a user sets the RTC very close to its overflow value the RTC needs to keep counting from 0 after it overflows




arm

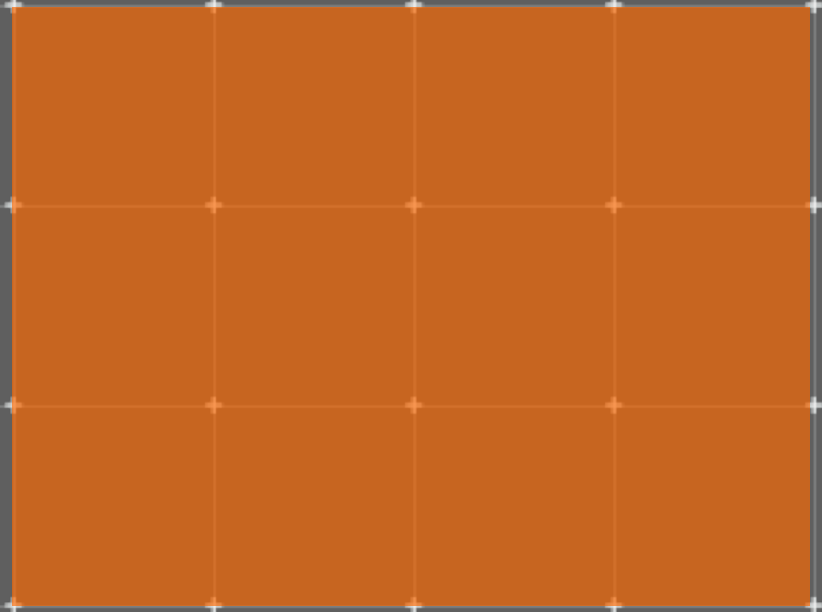
# What to look out for? - mktime and localtime

- Standard C `mktime` and `localtime` use RTOS primitives and can cause problems inside of drivers in critical sections
- Mbed OS provides its own set of `mktime` and `localtime` functions specifically for repurposing the RTC for other uses in the driver




arm

# Porting RTC





arm

# Explaining HAL functions

- Add `RTC` to `device_has` in targets.json

  https://github.com/ARMmbed/mbed-os/blob/master/targets/targets.json

```
"LPC1768": {
        "inherits": ["LPCTarget"],
        "core": "Cortex-M3",
        "extra_labels": ["NXP", "LPC176X", "MBED_LPC1768"],
        "supported_toolchains": ["ARM", ..., "GCC_CR", "IAR"],
        "detect_code": ["1010"],
        "device_has": ["ANALOGIN", ..., "RTC"],
        "release_versions": ["2", "5"],
        "features": ["LWIP"],
        "device_name": "LPC1768",
        "bootloader_supported": true
    },
```

- Next step is to implement the API

  https://github.com/ARMmbed/mbed-os/blob/feature-hal-spec-rtc/hal/rtc_api.h




arm

# Implementing rtc_init and rtc_free

```
void rtc_init(void);
void rtc_free(void);
```

- `rtc_init()`
  - Initialize the RTC peripheral
  - Is safe to call repeatedly
  - Don’t stop RTC from counting
  - Calling any other function before rtc_init() is undefined
  - Does not reset the counter, counter value is assigned by rtc_write()
- `rtc_free()`
  - Deinitialize RTC, it should only affect the CPU domain and not the time keeping logic
  - Dosn’t stop RTC from counting
  - Calling any function other than `rtc_init()` after `rtc_free()` is undefined

# Implementing rtc_read and rtc_write

```
time_t rtc_read(void);
void rtc_write(time_t t);
```

- `rtc_read()` gets the current time from the RTC peripheral in seconds
- `rtc_write()` writes the current time in seconds to the RTC peripheral in seconds

# Implementing rtc_isenabled

```
int rtc_isenabled(void);
```

- Check if the RTC has the time set and is counting

# What tests to pass?

- HAL Sleep validation suite:

```
$ git checkout feature-hal-spec-rtc
$ mbed test -t <toolchain> -m <target> -n "tests-mbed_hal-rtc*"
```

# Hands-on workshop

- RTC
  - Use branch - `feature-hal-spec-rtc`
  - HAL API & Testing - https://os.mbed.com/docs/v5.7/feature-hal-spec-rtc-doxy/group__hal__rtc.html
  - Porting - https://github.com/ARMmbed/mbed-sip-workshop-2018q1/blob/master/rtc.md

- Workshop materials - https://github.com/ARMmbed/mbed-sip-workshop-2018q1



arm

# Example – RTC display

```
#include "mbed.h"

InterruptIn irq(BUTTON1);
void reset() {
    // Set RTC time to Wed, 28 Oct 2009 11:35:37
    set_time(1256729737);
}

int main() {
    irq.fall(reset);
    while (true) {
        time_t seconds = time(NULL);
        printf("Seconds since January 1, 1970 = %d\n",
seconds);
        printf("Current time = %s", ctime(&seconds));

        char buffer[32];
        strftime(buffer, 32, "%I:%M %p\n",
localtime(&seconds));
        printf("Local time = %s", buffer);

        wait(1); // Wait 1 second
    }
}
```

Tasks:

- Bring up RTC on your boards
- Add an alarm?
  - Allow user to set alarm time?
  - Blink LEDs?
  - Make noise?
  - Light board on fire?




arm