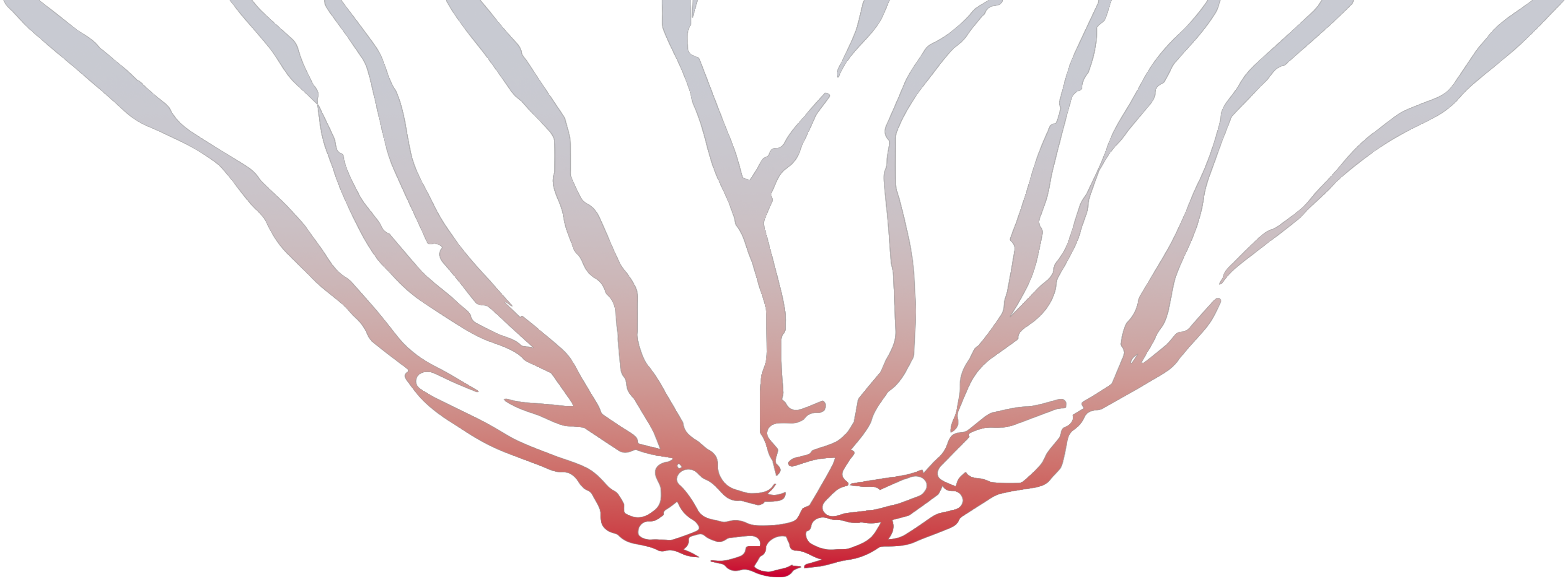


ESPRESSIF

TRAINING

Power Management

Objective

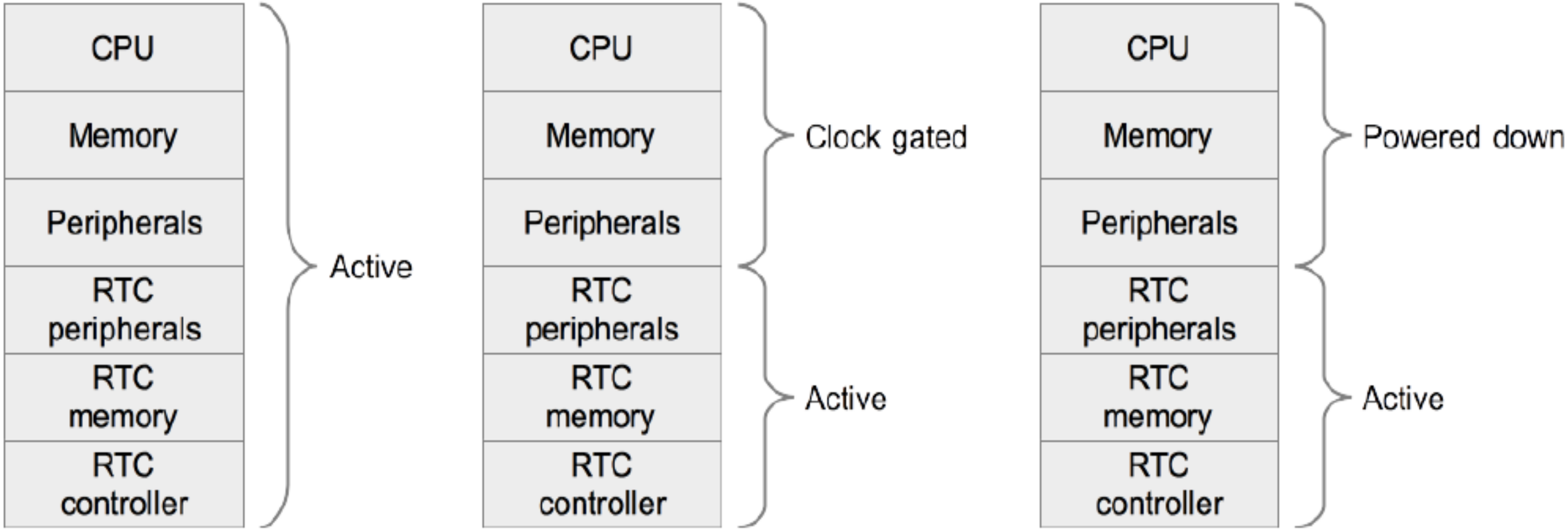
- Sleep Modes
  - Light Sleep Mode
  - Deep Sleep Mode

- Power Management
  - Configuration
  - DFS
  - Automatic Light Sleep

- Low power use cases with ULP

Sleep Modes

Light Sleep vs. Deep Sleep



CPU
Memory
Peripherals
RTC peripherals
RTC memory
RTC controller
Active

Active

CPU
Memory
Peripherals
Clock gated
RTC peripherals
RTC memory
RTC controller
Active

Light sleep

CPU
Memory
Peripherals
Powered down
RTC peripherals
RTC memory
RTC controller
Active

Deep sleep

Light Sleep Features

- Memory Contents and CPU/Peripherals state is preserved
- Current consumption close to 1mA (dominated by leakage, WiFi/BT off)
- Several wakeup sources: timer, GPIO, ULP
- Time is preserved using RTC clock (always on domain)

Deep Sleep Features

- Lowest current consumption (8-9uA)
- On wakeup application starts from scratch
- Several wakeup sources: timer, GPIO, ULP
- Time is preserved using RTC clock

Usage and API

- Configure Wakeup Source

```
esp_err_t esp_sleep_enable_ulp_wakeup();
esp_err_t esp_sleep_enable_timer_wakeup(uint64_t time_in_us);
esp_err_t esp_sleep_enable_touchpad_wakeup();
esp_err_t esp_sleep_enable_ext0_wakeup(gpio_num_t gpio_num, int level);
esp_err_t esp_sleep_enable_ext1_wakeup(uint64_t mask, esp_sleep_ext1_wakeup_mode_t mode);
```

- Enable sleep mode

```
void esp_light_sleep_start();
void esp_deep_sleep_start();
```

https://docs.espressif.com/projects/esp-idf/en/v3.1/api-reference/system/sleep_modes.html

Power Management

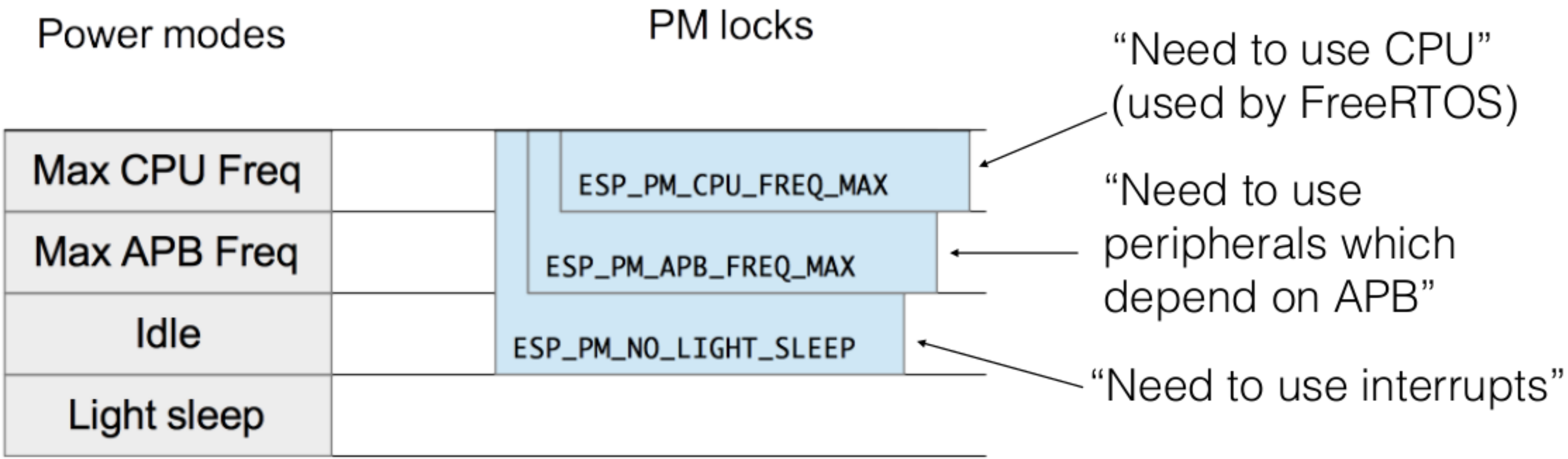
Configuration

- Scale CPU and APB Frequency for power saving
- Optionally enable light sleep mode
- Power management locks drives specific power mode transition

```
typedef enum {
        PM_MODE_LIGHT_SLEEP,//!< Light sleep
        PM_MODE_APB_MIN,      //!< Idle (no CPU frequency or APB frequency locks)
        PM_MODE_APB_MAX,      //!< Maximum APB frequency mode
        PM_MODE_CPU_MAX,      //!< Maximum CPU frequency mode
        PM_MODE_COUNT         //!< Number of items
} pm_mode_t;
```

Power Management Locks

Power management locks are used by components to request specific *power mode.*



Power modes

PM locks

Max CPU Freq

Max APB Freq

Idle

Light sleep

ESP_PM_CPU_FREQ_MAX

ESP_PM_APB_FREQ_MAX

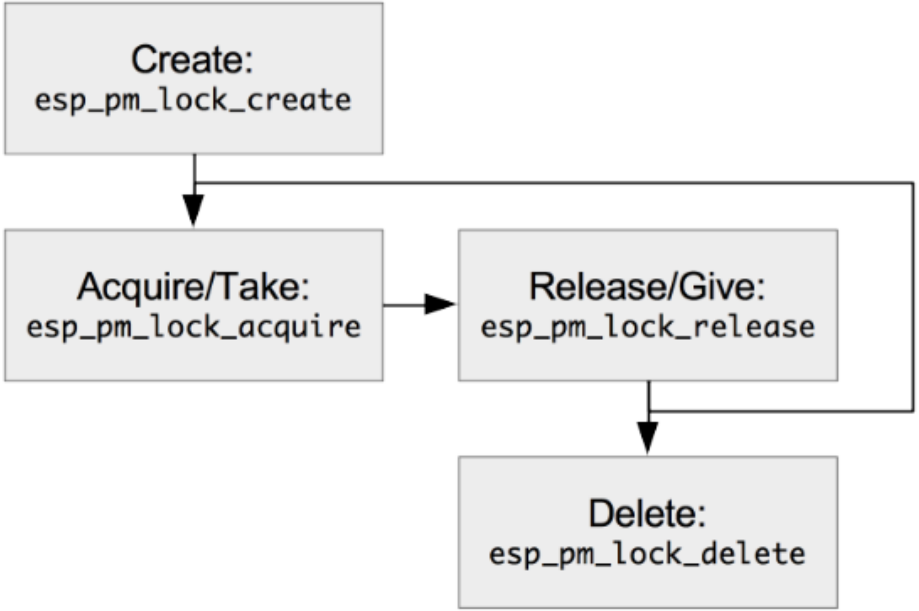
ESP_PM_NO_LIGHT_SLEEP

“Need to use CPU” (used by FreeRTOS)

“Need to use peripherals which depend on APB”

“Need to use interrupts”

Power Management Locks

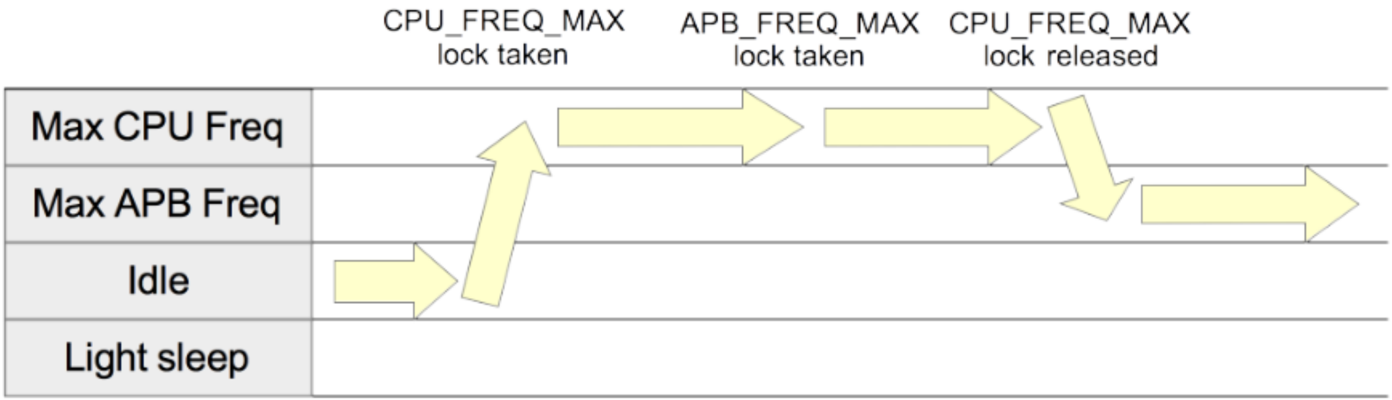


Take/acquire lock: means that component needs certain power mode.

Release/give lock: means that component no longer needs certain power mode.

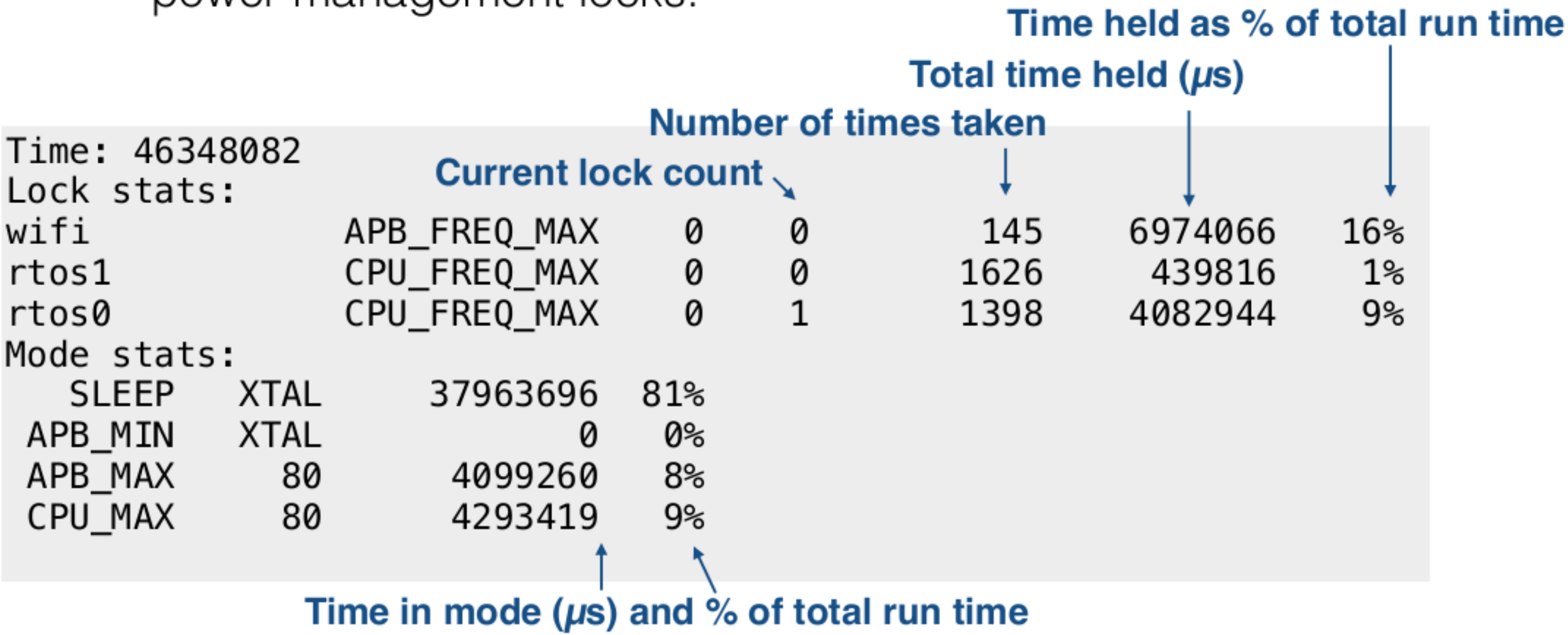
Power Management Locks

- Highest taken PM lock determines the current mode.
- Releasing some lock does not mean that mode will change — there may be other locks in the system.



Power Management Locks: Profiling

Enable CONFIG_PM_PROFILING and call esp_pm_dump function to get information about power management locks:



```
Time: 46348082
Lock stats:
wifi            APB_FREQ_MAX    0    0       145   6974066   16%
rtos1           CPU_FREQ_MAX    0    0      1626    439816    1%
rtos0           CPU_FREQ_MAX    0    1      1398   4082944    9%
Mode stats:
   SLEEP    XTAL      37963696  81%
 APB_MIN    XTAL             0   0%
 APB_MAX      80       4099260   8%
 CPU_MAX      80       4293419   9%
```

Time held as % of total run time

Total time held (μs)

Number of times taken

Current lock count

Time in mode (μs) and % of total run time

Dynamic Frequency Scaling

- CPU frequency gets scaled between min and max values provided
- APB frequency gets automatically scaled (<= CPU frequency)
- Acceptable level of timekeeping accuracy
- PLL is not disabled due to latency considerations

Dynamic Frequency Scaling

- If maximal CPU frequency is 160 MHz:
  - When ESP_PM_CPU_FREQ_MAX is acquired, CPU frequency is set to 160 MHz, and APB frequency to 80 MHz.
  - When ESP_PM_CPU_FREQ_MAX is not acquired, but ESP_PM_APB_FREQ_MAX is, CPU and APB frequencies are set to 80 MHz.
  - Otherwise, frequency will be switched to the minimal value set using esp_pm_configure().

Current Numbers

- Connected WiFi Mode ~110mA (Active Tx ~190mA)
- WiFi Modem Sleep ~40mA (CPU @160MHz)
- Automatic light sleep + Modem sleep ~11mA
- DFS + Automatic light sleep + Modem sleep ~5mA

API And Usage

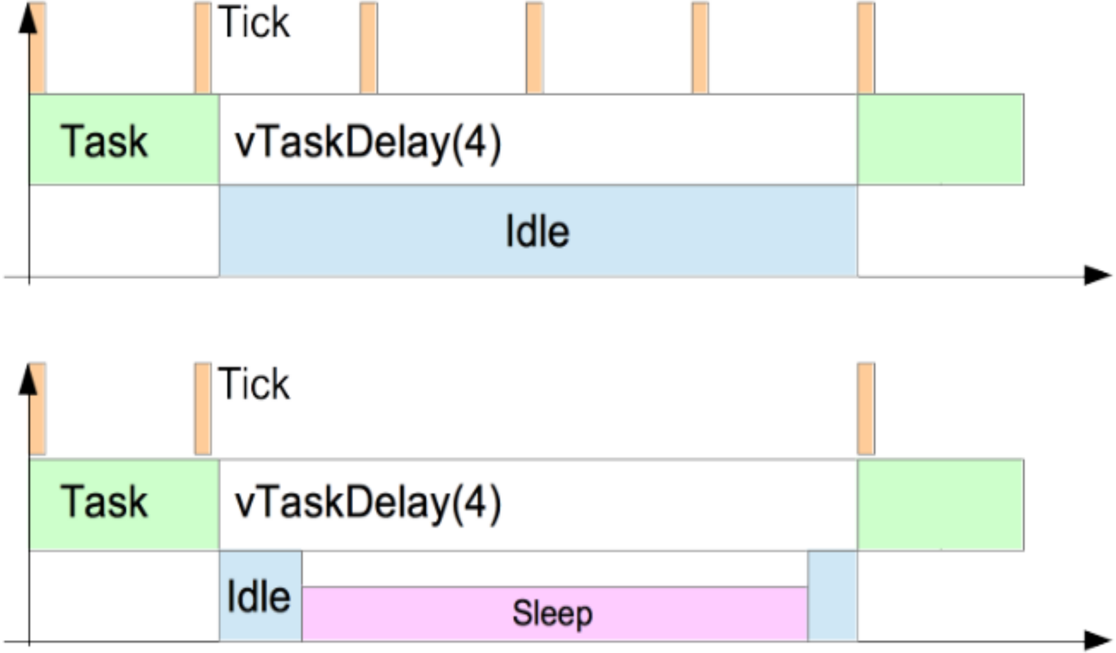
```
#if CONFIG_PM_ENABLE
        esp_pm_config_esp32_t pm_config = {
                .max_cpu_freq = RTC_CPU_FREQ_160M,
                .min_cpu_freq = RTC_CPU_FREQ_XTAL,
#if CONFIG_FREERTOS_USE_TICKLESS_IDLE
                .light_sleep_enable = true
#endif
        };
        esp_pm_configure(&pm_config);
#endif // CONFIG_PM_ENABLE
```

https://docs.espressif.com/projects/esp-idf/en/v3.1/api-reference/system/power_management.html

Automatic Light Sleep

- Automatic light sleep using FreeRTOS Tickless Idle
- Task unblock time, FreeRTOS timers, and esp_timer timers are taken into account when calculating sleep time
- System Tick is adjusted based on sleep time

FreeRTOS Tickless Idle



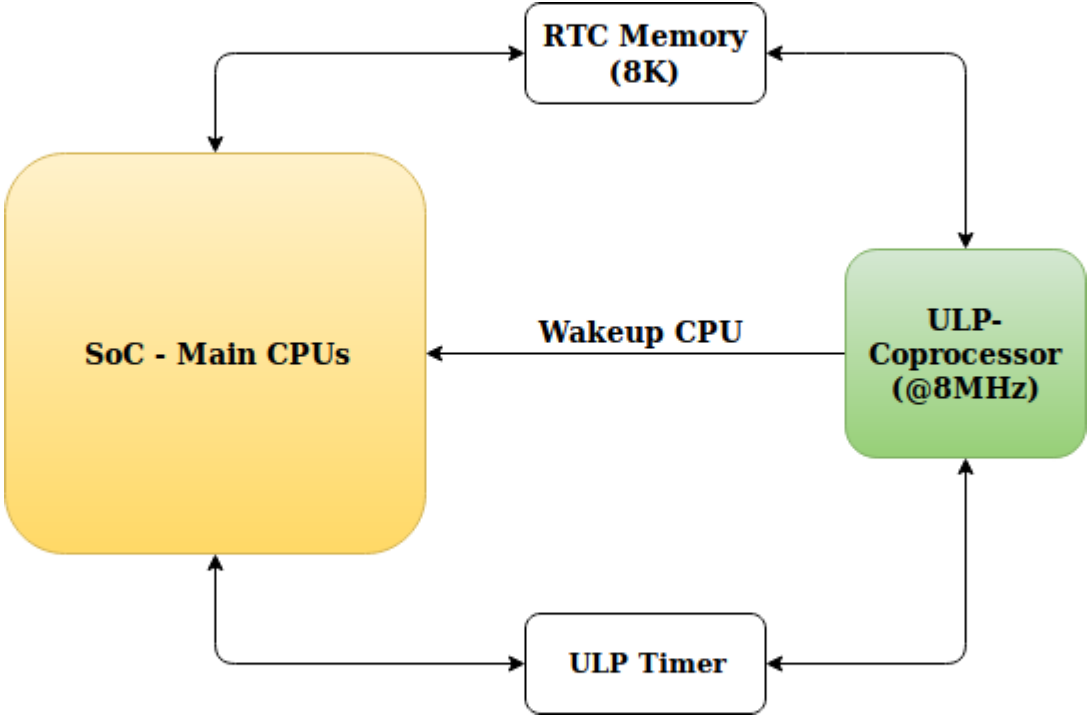
Tickless Idle feature allows RTOS to skip a number of tick interrupts, allowing the application to enter low power mode

ULP Coprocessor

Ultra Low Power Coprocessor

- Coprocessor with its own instruction set
- Upto 8KB RTC slow memory for execution
- Can access peripheral devices GPIO/I2C/ADC
- Current consumption <30uA (based on wakeup period of ULP)

Ultra Low Power Co-processor



ULP API

- esp_err_t **ulp_load_binary**(uint32_t **load_addr**, const uint8_t* **program_binary**, size_t **program_size**);
- esp_err_t **ulp_set_wakeup_period**(size_t **period_index**, uint32_t **period_us**);
- esp_err_t **ulp_run**(uint32_t **entry_point**);

Setting up ULP demo

- ULP requires setting up its own toolchain
- ULP firmware gets embedded in application firmware image

For more details please refer: https://docs.espressif.com/projects/esp-idf/en/v3.1/api-guides/ulp.html

(examples/system/ulp, examples/system/ulp_adc)