Power Measurement and Optimization

Michael Gerndt
Technische Universität München

Background

- Sensors are frequently used without access to the power grid.
- **Goal**: reduce power consumption to extend battery lifetime.
- Sensors operate in two modes: sleep mode and duty cycle
  - Duty cycle: sensor is active
  - Sleep mode: sensor is inactive
- Power aspects
  - Length of the duty cycle
  - Power during the duty cycle and the sleep mode
- What effects these two aspects?
  - Duty cycle length?
  - Duty cycle power?
  - Sleep mode power?

Duty Cycle Length

- The application
- Latency of computation
  - Clock frequency
  - Code optimization
  - Suitability of selected instructions
  - Memory latency

Power Consumption of Duty Cycle

- Clock frequency: Higher frequency -> more power
  - Why?
  - Dynamic Voltage and Frequency Scaling (DVFS), ESP32 only DFS
- Number of powered up components
  - RTC, two cores, RAM, FLASH, peripherals
  - Two types of power saving:
    - Clock gating
    - Power gating

Sleep Modes

- Switches to sleep mode by executing a special instruction.
- Return from sleep mode by a wakeup signal.
- Power consumption during sleep mode depends on:
    - Powered up components
    - Infrastructure: e.g., power stabilizers
    - Peripherals: LEDs, Displays, Sensors
- Processors might support different sleep modes depending on which components are powered up.
    - ESP: light and deep sleep

Power vs Energy

- Power:
  - Instantaneous Voltage * Current
  - Measured in Watt (W), e.g. 3.3 V * 100 mA = 0.33 W
  - During optimization focus on current if power is constant.
- Energy
  - Power integrated over time
  - Measured in Wh or Joule (1 Joule = 1 Watt for one second)
  - 0.33 W for 10 seconds = 3.3 Ws = 0,9 mWh
- Typical LiPo battery

ESP Power Saving Features

- Light and deep sleep
- Automatic light sleep
- Power saving for Wifi
- Different clock frequencies
- Dynamic Frequency Scaling

Sleep Modes

- Sleep Modes reduce the power consumption through clock and power gating.
  - Light sleep:
    - clock gating of Xtensa cores
    - Peripherals are clock gated and interrupts are not generated
    - Continues from the point where light sleep was started
  - Deep sleep:
    - power gating Xtensa cores and RAM
    - RTC slow memory by default powered on if variables are placed there (RTC_NOINIT_ATTR).
    - RTC fast memory by default powered off. Can be overwritten by esp_sleep_pd_config()
    - Reboots after deep sleep
- Wakeup from sleep modes through wakeup sources
  - Different wakeup sources for different sleep modes

Switch to Sleep Mode

```
#include "esp_sleep.h"

int sleep_sec = 15;
ESP_ERROR_CHECK(esp_sleep_enable_timer_wakeup(1000000LL * sleep_sec));
ESP_LOGI(TAG,"Starting light sleep");
esp_light_sleep_start();
ESP_LOGI(TAG,"Woke up from light sleep");

printLocalTime();

ESP_LOGI(TAG,"Starting deep sleep");
esp_deep_sleep(1000000LL * sleep_sec);
```

- Deep sleep without specifying the sleep time

```
esp_deep_sleep();
```

Wake Up from Sleep Modes

- Wake up is possible through the Real Time Clock (RTC) module which is a low-power subsystem consisting of
  - Timer, Ultra Low Power co-Processor (ULP) and RTC slow and fast memory.
- Wake up sources
  - Build in timer: configured via **esp_sleep_enable_timer_wakeup()**
  - Touch pad: **esp_sleep_enable_touchpad_wakeup()**
  - External Wakeup (ext0): single pin; **esp_sleep_enable_ext0_wakeup()**
  - External Wakeup (ext1): multiple pins; **esp_sleep_enable_ext1_wakeup()**
  - ULP coprocessor wake up: **esp_sleep_enable_ulp_wakeup()**
  - GPIO wakeup (only light sleep): **esp_sleep_enable_gpio_wakeup()**
  - UART wakeup (only light sleep): **esp_sleep_enable_uart_wakeup()**
- Configuration of a wakeup source ensures that the components are powered on during deep sleep
  - E.g. ULP wake up powers up the ULP coprocessor and RTC slow memory

RTC GPIO wake up from Deep Sleep

- External Wakeup (ext0)
  - Can be connected to a single RTC GPIO pin

```
esp_err_t esp_sleep_enable_ext0_wakeup(gpio_num_t gpio_num,
                                                   int level)
    //input level which will trigger wakeup (0=low, 1=high)
```

- External Wakeup (ext1)
  - Can be connected to multiple RTC GPIO pins
  - Can be triggered by all low or any high

```
esp_err_t esp_sleep_enable_ext1_wakeup(uint64_t mask,
                                       esp_sleep_ext1_wakeup_mode_t mode)
    //ESP_EXT1_WAKEUP_ALL_LOW or ... ANY_HIGH
```

RTC GPIO

- Only some of the GPIOs can be used in the RTC module
- The API routines use the GPIO pin numbers

GPIO	Analog Function	RTC GPIO	Comments
GPIO0	ADC2_CH1	RTC_GPIO11	Strapping pin
GPIO1			TXD
GPIO2	ADC2_CH2	RTC_GPIO12	Strapping pin
GPIO3			RXD
GPIO4	ADC2_CH0	RTC_GPIO10	
GPIO5			Strapping pin
GPIO6			SPI0/1
GPIO7			SPI0/1
GPIO8			SPI0/1
GPIO9			SPI0/1
GPIO10			SPI0/1
GPIO11			SPI0/1
GPIO12	ADC2_CH5	RTC_GPIO15	Strapping pin; JTAG
GPIO13	ADC2_CH4	RTC_GPIO14	JTAG
GPIO14	ADC2_CH6	RTC_GPIO16	JTAG
GPIO15	ADC2_CH3	RTC_GPIO13	Strapping pin; JTAG
GPIO16			SPI0/1
GPIO17			SPI0/1
GPIO18			
GPIO19			
GPIO20			This pin is only available on ESP32-PICO-V3 chip package
GPIO21			
GPIO22			
GPIO23			
GPIO25	ADC2_CH8	RTC_GPIO6	
GPIO26	ADC2_CH9	RTC_GPIO7	
GPIO27	ADC2_CH7	RTC_GPIO17	
GPIO32	ADC1_CH4	RTC_GPIO9	
GPIO33	ADC1_CH5	RTC_GPIO8	
GPIO34	ADC1_CH6	RTC_GPIO4	GPI
GPIO35	ADC1_CH7	RTC_GPIO5	GPI
GPIO36	ADC1_CH0	RTC_GPIO0	GPI
GPIO37	ADC1_CH1	RTC_GPIO1	GPI
GPIO38	ADC1_CH2	RTC_GPIO2	GPI
GPIO39	ADC1_CH3	RTC_GPIO3	GPI

Automatic Light Sleep

- ESP32 supports an automatic switch to light sleep.
- It is based on the tickless mode of FreeRTOS.
  - It has to be enabled in menuconfig->component config->FreeRTOS.
  - The minimum number of ticks to switch to light sleep can be set.
- Power management must be enabled
  - menuconfig->component config->power management
- The ESP32 automatically switches to light sleep mode when no locks are acquired.

```
#include "esp_pm.h"

void configPM(){
    esp_pm_config_esp32_t pm_config = {
        .max_freq_mhz = 160,
        .min_freq_mhz = 160,         //DFS, enable in menucofig in Power Management
        .light_sleep_enable = true   //automatic light sleep, enable via menuconfig in FreeRTOS
    };
    ESP_ERROR_CHECK(esp_pm_configure(&pm_config));

}
```

Power Saving for Wifi

- AP is sending beacon every 102400 µs.

```
I (1030) my WIFI: wifi connected!
I (1080) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (1660) esp_netif_handlers: sta ip: 192.168.178.77, mask: 255.255.255.0, gw: 19
2.168.178.1
I (1660) my WIFI: got ip:192.168.178.77
```

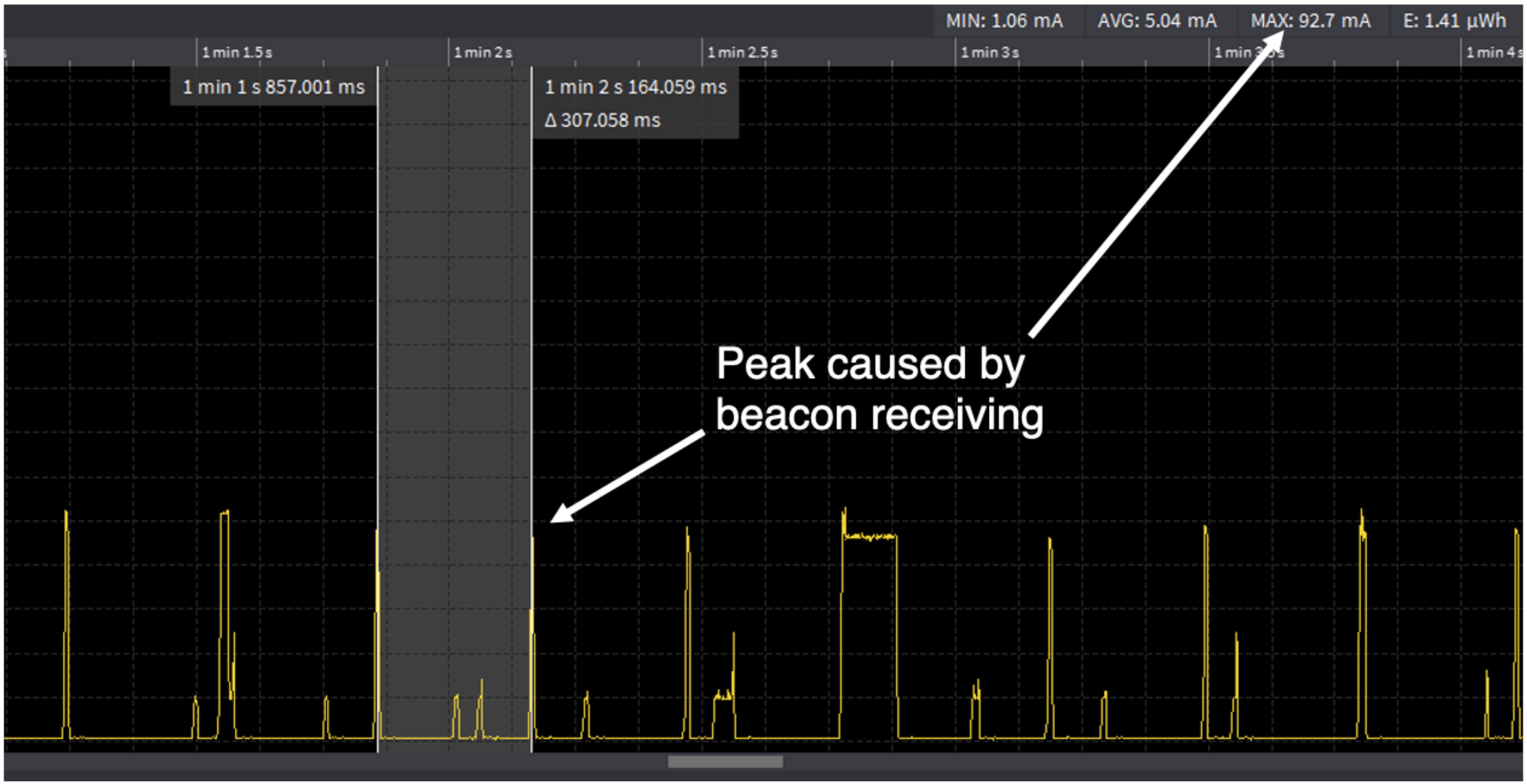
- We don’t need to listen every one of them.

```
wifi_config_t wifi_config = {
  .sta = {
        .ssid = EXAMPLE_ESP_WIFI_SSID,
        .password = EXAMPLE_ESP_WIFI_PASS,
        .listen_interval = 3, //factor of intervals between beacons
  },
};
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config) );
ESP_ERROR_CHECK(esp_wifi_start() );

ESP_LOGI(TAG, "esp_wifi_set_ps().");
esp_wifi_set_ps(WIFI_PS_MAX_MODEM); //Required to specify list_interval
```

Power Saving for Wifi

Effect of listen interval = 3: $3 \times 102 \approx 306ms$



Select Clock Frequency

- Standard clock frequencies are 80 MHz, 160 MHz, 240 MHz
- These can be selected via
  - menuconfig->ESP32-specific->CPU frequency
- Higher clock frequency -> higher power consumption (due to more frequent transistor switching) and faster computation
  - Efficiency of the code determines whether it is worth going faster.
- Other frequencies (10, 20, 40) can be selected programmatically
  - XTAL frequency: crossTAL -> crystal
  - The external crystal frequency is 40 MHz
  - Wifi and bluetooth require at least 80 MHz

```
#include "esp_pm.h"

void configPM(){
    esp_pm_config_esp32_t pm_config = {
        .max_freq_mhz = 80,
        .min_freq_mhz = 10,         //DFS, enable in menuconfig in Power Management
        .light_sleep_enable = true   //automatic light sleep, enable via menuconfig in FreeRTOS
    };
    ESP_ERROR_CHECK(esp_pm_configure(&pm_config));

}
```

Dynamic Frequency Scaling

- The CPU adapts the clock frequency within a specified range automatically.
- The selection of the min or max frequency depends on whether a task other than idle is able to run.
  - If other tasks are ready->max frequency
  - If idle is the only ready task-> min frequency
- Automatic profiling of the frequency enabled with
  - menuconfig->component config->power management->enable profiling counters
- Dump profiling counters via
  - `esp_pm_dump_locks(stdout);`

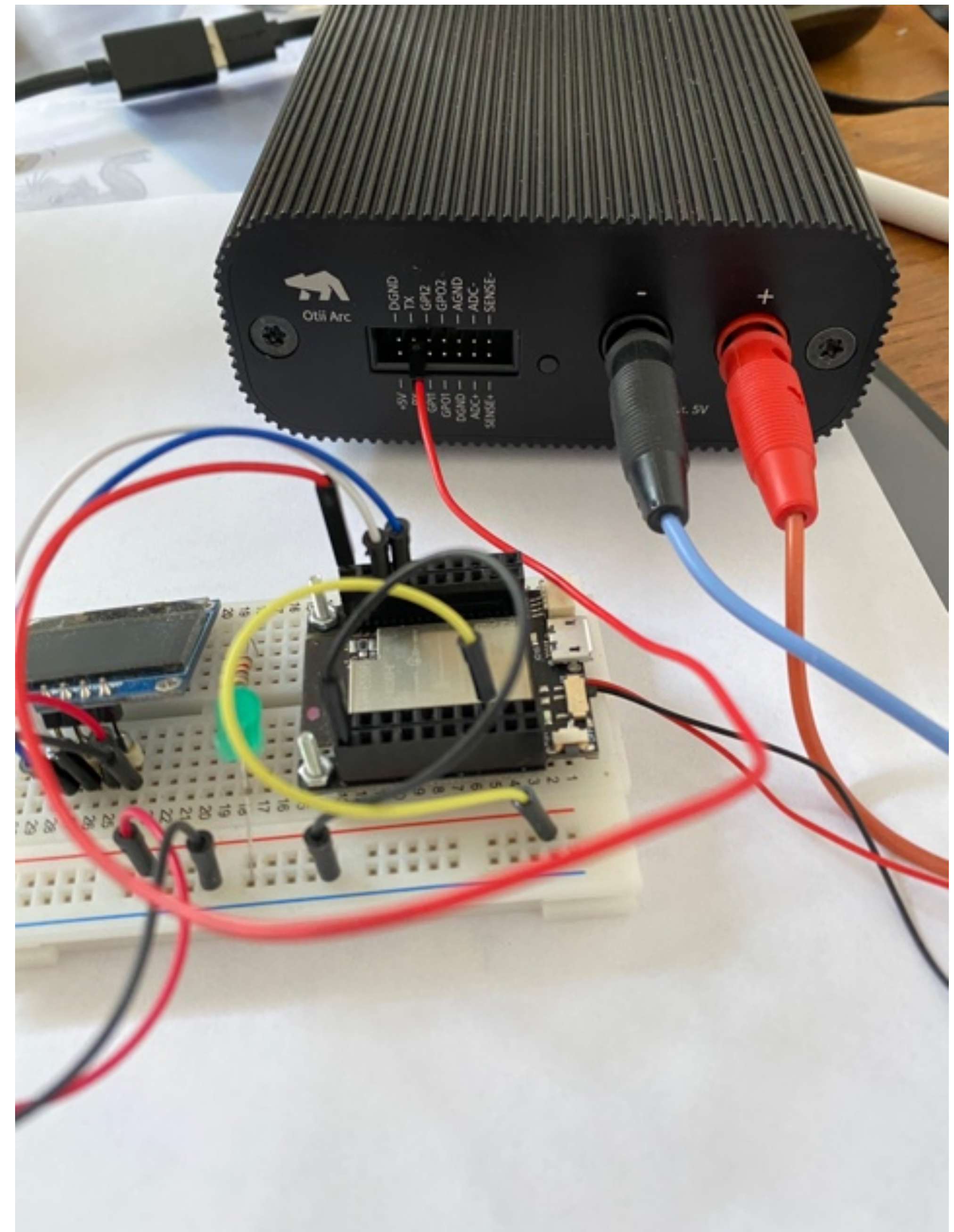
Measurement Infrastructure

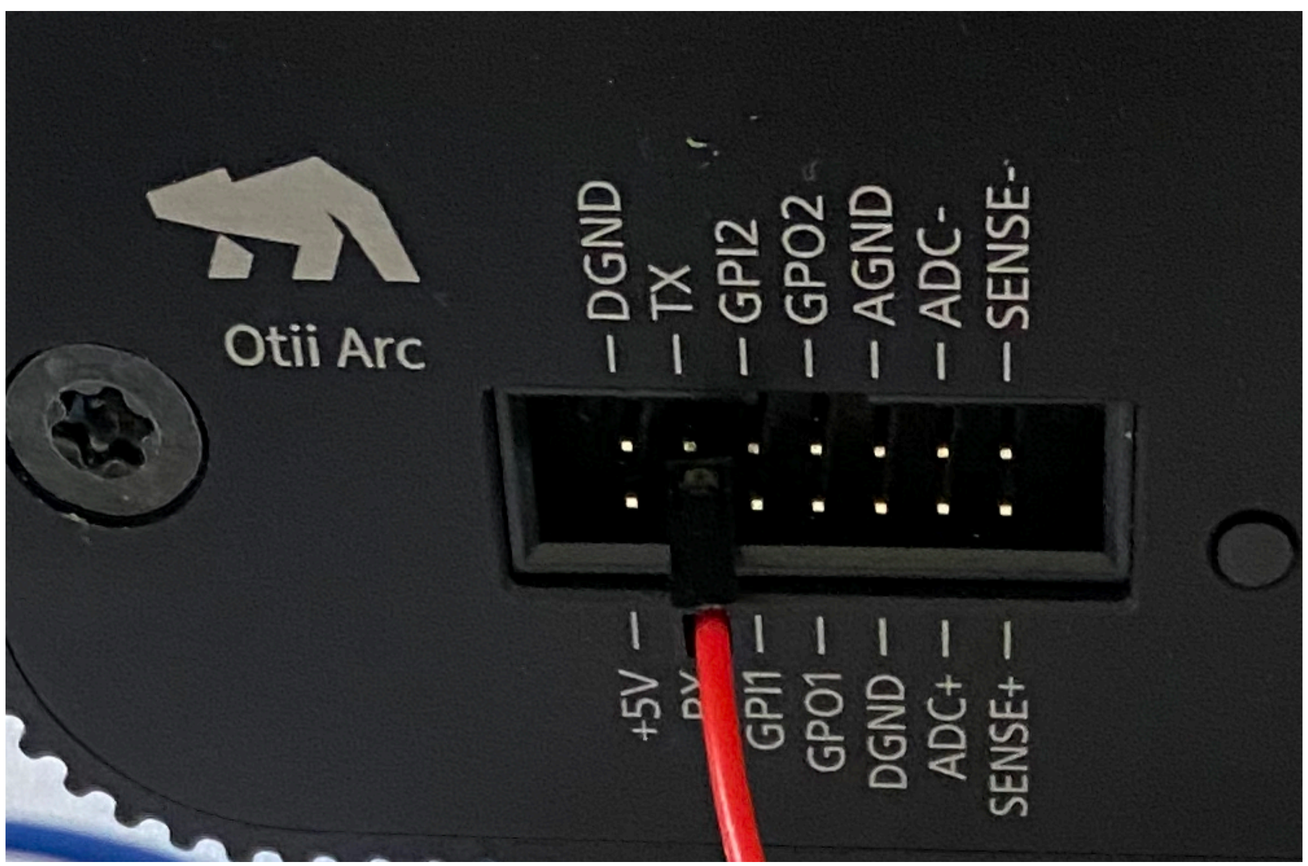
- Otii Arc

- Connect the device to the computer by USB. The device will power up the board and measure the current.
- Connect the board to the device

Connect Otii Arc and Board

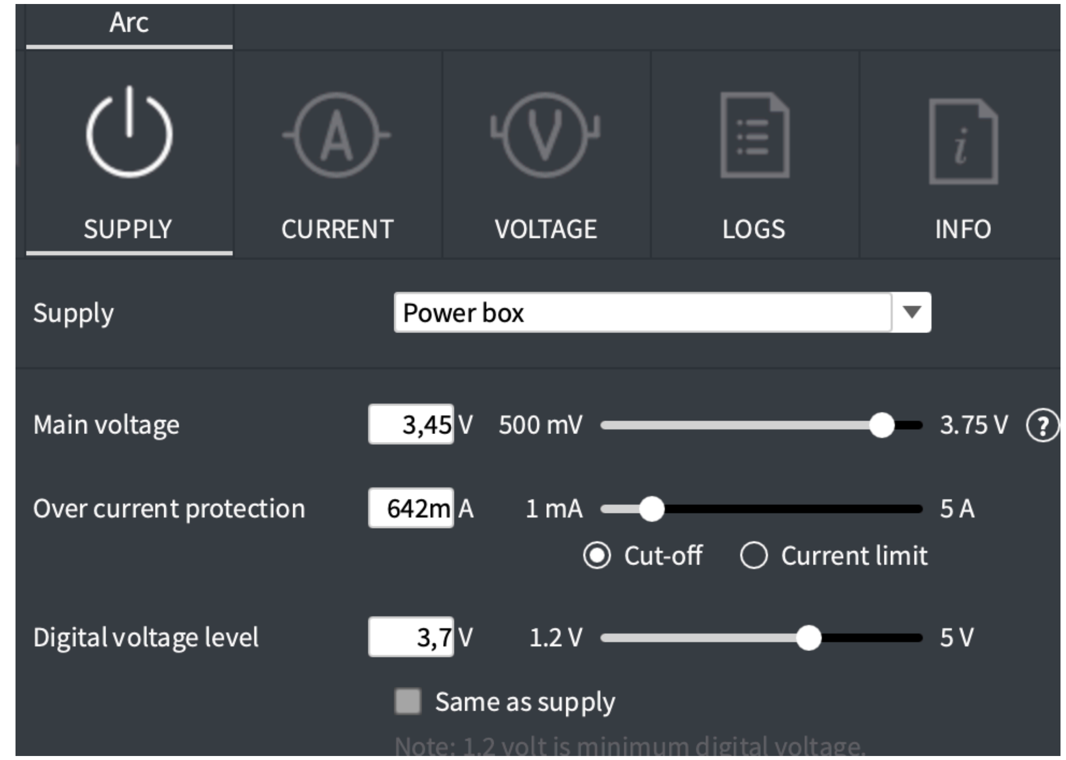
- Power outlets -> JST connector for battery
- RX -> board serial connector TX



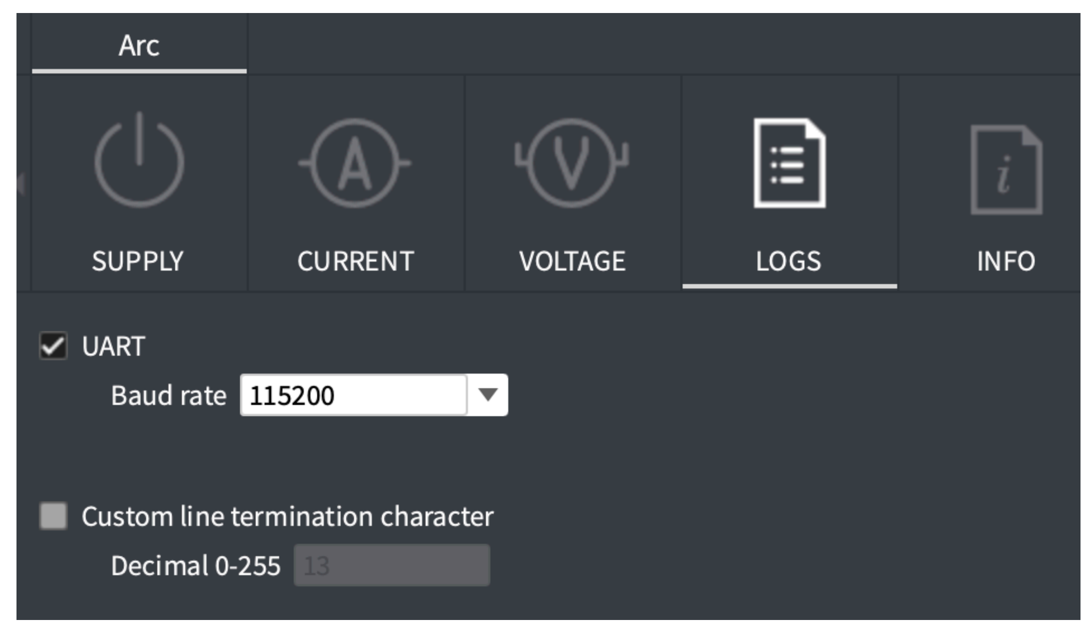


Define a Project

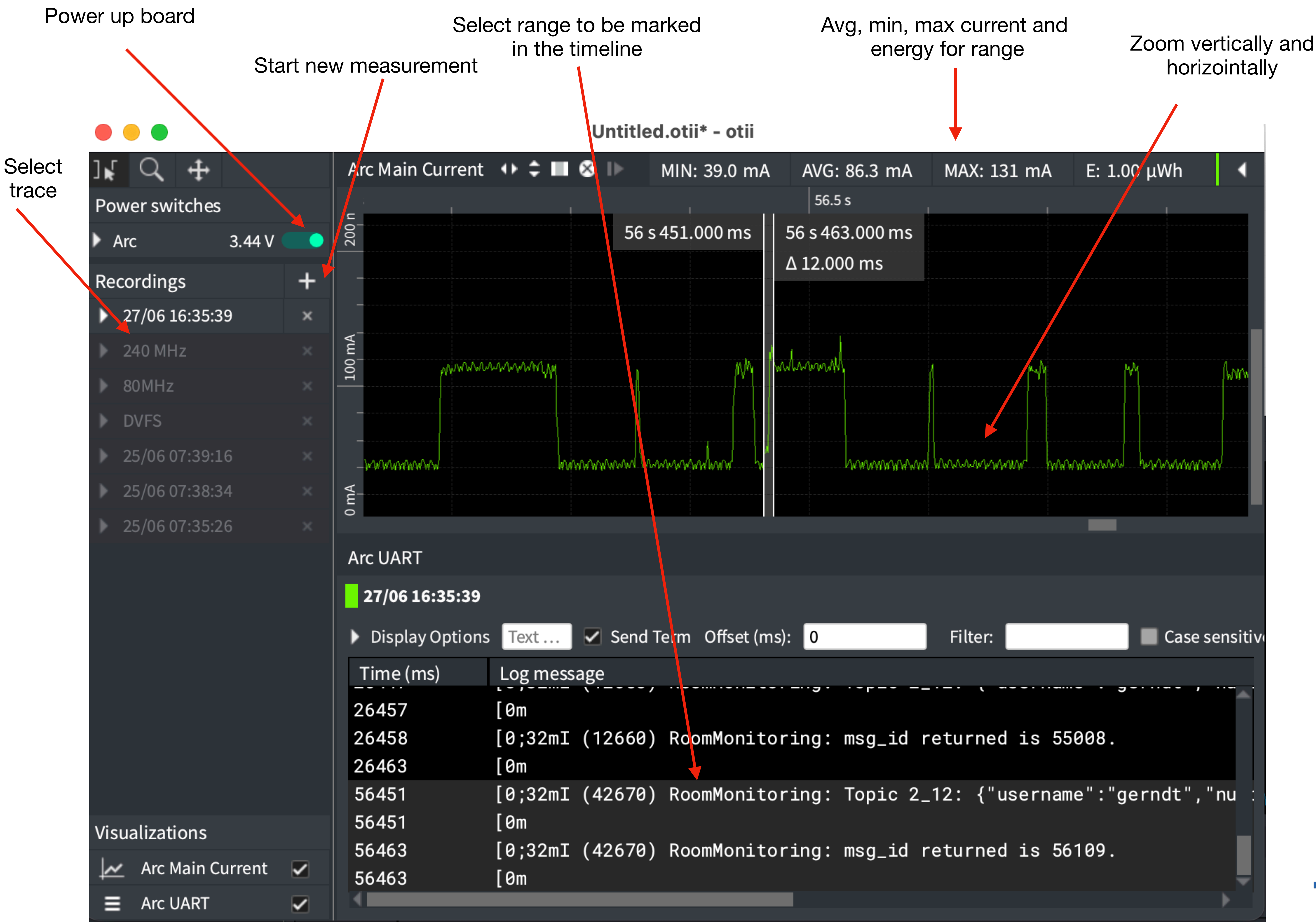
- Set voltage to 3.45
- Set current >400 mA



- Check UART to record message on the serial connection



Record Input Current



Warning

- Do not connect the board through USB for flashing while it is powered up through the measurement device.
  - Switch off the power in the GUI before.
- Reason
  - Your measurements will be wrong, because the board tries to charge the battery.
  - You might break the board.

Assignment 6

- Perform the same measurements separately for
  - initial application phase (Wifi/SNTP)
  - counting phase (without triggering interrupts, with and without MQTT for counts and predictions)
  - What is the best setting to reduce power or energy for each phase?
- Run measurements
  - Test 80/160/240 MHz
  - Test automatic light sleep
  - Test DFS 80 - 240 MHz
  - Test beacon optimization
- Study the effect of the OLED Display. Try switching the display on and off
- Calculate the lifetime of a 1000 mAh battery for your best settings. Assuming that the display is off.
- Determine the power consumption in
  - Light sleep and deep sleep based on timer wakeup