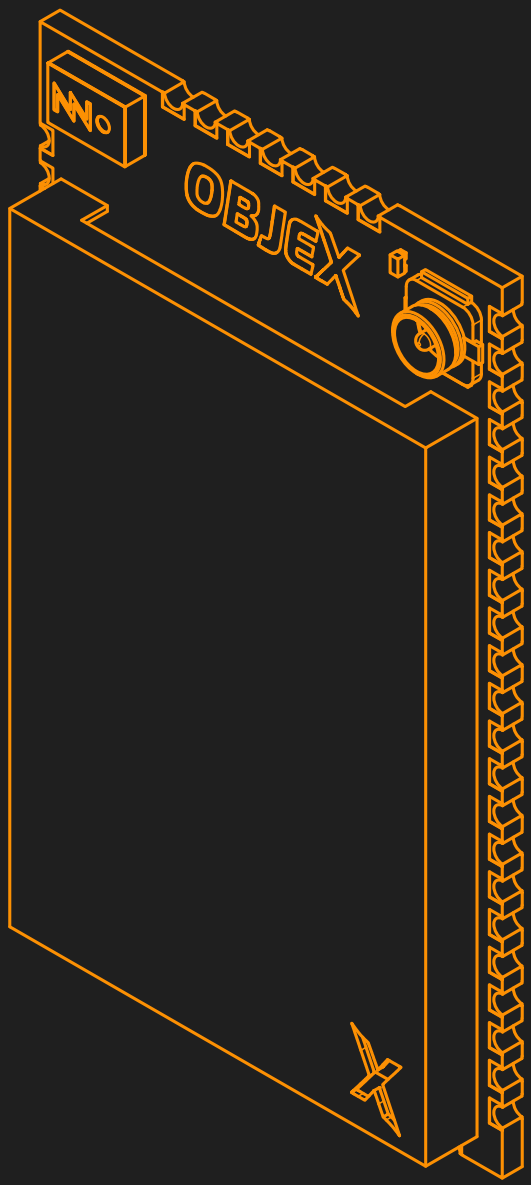
OBJEX

ELPM-S3LW

Extreme Low Power Module based on ESP32-S3FN8 and SX1262

v1.1 LW-B



DATASHEET

OBJEXLABS.COM

Contents

1 | Overview

The ELPM-S3LW is extreme-low power module based on ESP32-S3FN8 and SX1262. Designed for development of IoT devices or generic applications, especially battery-powered devices and energy harvesting applications. The module has a power unit designed to adapt has each specific use, and together with the deep stop mode it is possible to minimize power consumption. ELPM-S3LW is an excellent choice for industrial applications and devices that require high energy efficiency.

1.1 | Features

Microcontroller

- Based on ESP32-S3FN8 (32-bit 240MHz)
- Memory Size: 8MB Flash
- GPIO: 40 available
- WiFi - IEEE 802.11 b/g/n-compliant
- Bluetooth LE: Bluetooth 5, Bluetooth mesh
- Wi-Fi and Bluetooth share the same antenna
- Cryptographic hardware acceleration
- External PSRAM can be installed
- Interfaces: I^2C, I^2S, SPI, UART, USB
- RGB status led (WS2812B)
  - PIN: GPIO48
- Integrated I2C pullup resistors
  - SDA: GPIO8
  - SCL: GPIO9
- Ceramic antenna (WiFi/BLE)
- 50 Ohm pin (WiFi/BLE)

LoRa

- Based on SX1262
- Frequency bands: 862MHz to 928MHz
- Signal power: 22dBm
- Transmission distance: 5km
- Communication interface: SPI
- Air data rate: 0.3kbps to 62.5kbps
- Dissipation Power: 1W
- 50 Ohm U.FL/pad antenna
- Dedicated LDO2 3.3V@300mA
- Dedicated 3.3V pad
- Full compatibility with LoRa and LoRaWAN

Extreme-low power unit

- Dedicated LDO1 3V3@300mA for ESP32
- Rising/falling edge detector (1-30nA)
- Wake and Gate input (1-30nA)
- RTC RV-3028-C7 (45-100nA)
- Dedicated RTC power supply pad
- Battery level circuit (zero leakage current)

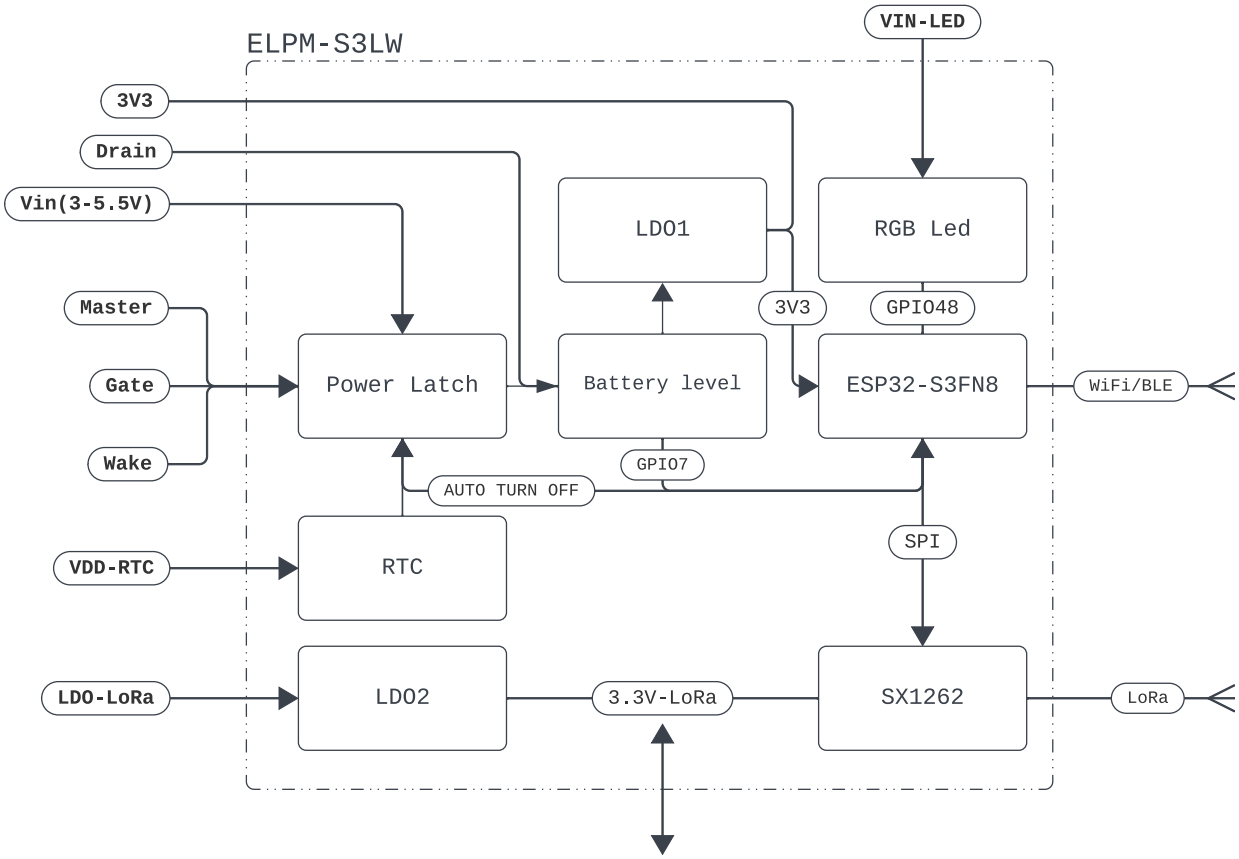
1.2 | Deep Stop Mode (Extreme low power)

An effective alternative to the deep sleep mode of the ESP32-S3FN8. Deep Stop mode consists of a power latch and RTC combined with an smart power management system that controls the ESP32-S3FN8, SX1262, and all loads connected to the 3V3 path. By controlling the power supply to all connected loads, it is possible to minimize power consumption, thus increasing the life of the battery. In addition, an external mosfet can be driven (gate pin) to control multiple loads.

1.3 | Applications

- Battery-powered nodes
- Smart meters
- Assets Tracking
- Smart cities
- Street Lights
- Supply chain
- Building automation
- Energy harvesting
- Smart Agriculture
- Environmental Sensors

2 | Block diagram



3V3: Main power path for the ESP32-S3FN8.

Vin: Power path handled by the power latch, ideal for a battery.

Master: Power latch input: If the status change(0 to 1 || 1 to 0) the P.L. is triggered.

Gate: Power latch input: If low, the power latch is activated instantly.

Wake: Power latch input: If low, the power latch is triggered with a short startup delay.

Drain: Power latch: path between the control block and the input of the primary LDO.

AUTO TURN OFF: Input signal to trigger shutdown by power latch.

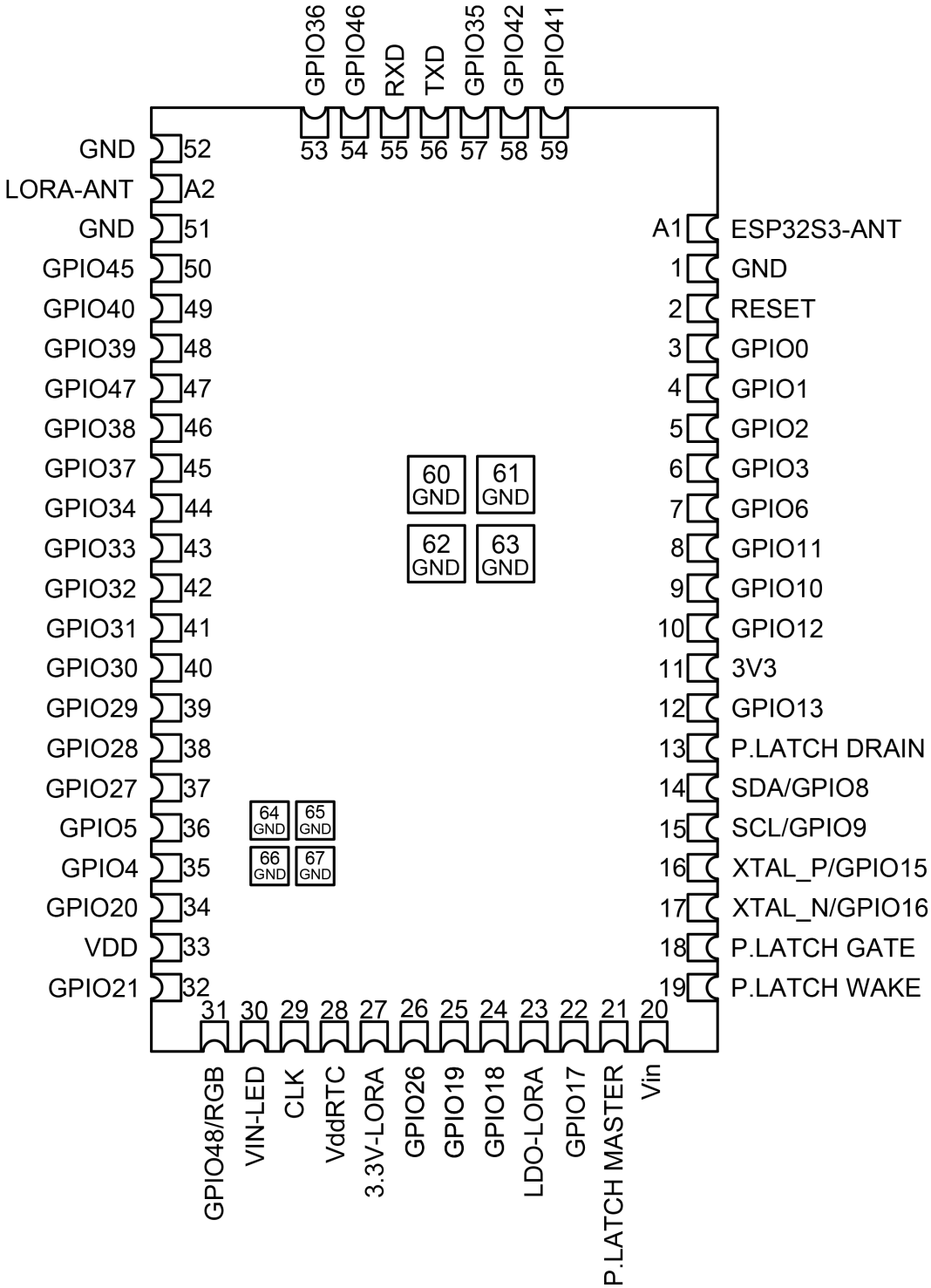
Vdd-RTC: Power path reserved for RV-3028-C7.

LDO-LoRa: LDO input reserved for SX1262.

3.3V-LoRa: Power path reserved for SX1262.

VIN-LED: Power path reserved for the RGB LED.

3 | Pin Definition



3.1 | Pin Overview

The module has 61 pins (castellated holes) and 8 pads (ground only).

No.	Name	Type[a]	Power Domain	Function
A1	ESP32-S3FN8 ANT	RF	-	Antenna interface, impedance 50ohm
A2	SX1262 ANT	RF	-	Antenna interface, impedance 50ohm
1	GND	P	-	Ground
2	RESET	I	3V3-RTC	ESP32-S3FN8 reset pin
3	GPIO0	I/O/T	3V3-RTC	RTC_GPIO0, GPIO0
4	GPIO1	I/O/T	3V3-RTC	RTC_GPIO1, GPIO1, TOUCH1, ADC1_CH0
5	GPIO2	I/O/T	3V3-RTC	RTC_GPIO2, GPIO2, TOUCH2, ADC1_CH1
6	GPIO3	I/O/T	3V3-RTC	RTC_GPIO3, GPIO3, TOUCH3, ADC1_CH2
7	GPIO6	I/O/T	3V3-RTC	RTC_GPIO6, GPIO6, TOUCH6, ADC1_CH5
8	GPIO11	I/O/T	3V3-RTC	RTC_GPIO11, GPIO11, TOUCH11, ADC2_CH0, FSPID
9	GPIO10	I/O/T	3V3-RTC	RTC_GPIO10, GPIO10, TOUCH10, ADC1_CH9, FSPICS0
10	GPIO12	I/O/T	3V3-RTC	RTC_GPIO12, GPIO12, TOUCH12, ADC2_CH1, FSPICLK
11	3V3	P	-	ESP32-S3FN8 Analog power supply
12	GPIO13	I/O/T	3V3-RTC	RTC_GPIO13, GPIO13, TOUCH13, ADC2_CH2, FSPIQ
13	P.LATCH DRAIN	PL	-	Power latch Analog power supply
14	SDA	I/O/T	3V3-RTC	RTC_GPIO8, GPIO8, TOUCH8, ADC1_CH7, SUBSPICS1
15	SCL	I/O/T	3V3-RTC	RTC_GPIO9, GPIO9, TOUCH9, ADC1_CH8, SUBSPIHD
16	XTAL-P	I/O/T	3V3-RTC	RTC_GPIO15, GPIO15, U0RTS, ADC2_CH4
17	XTAL-N	I/O/T	3V3-RTC	RTC_GPIO16, GPIO16, U0CTS, ADC2_CH5
18	P.LATCH GATE	PL	Vin	Power latch input
19	P.LATCH WAKE	PL	Vin	Power latch input
20	Vin	P	-	Power latch Analog power supply
21	P.LATCH MASTER	PL	Vin	Power latch input
22	GPIO17	I/O/T	3V3-RTC	RTC_GPIO17, GPIO17, U1TXD, ADC2_CH6
23	LDO-LORA	P	-	SX1262 Analog power supply
24	GPIO18	I/O/T	3V3-RTC	RTC_GPIO18, GPIO18, U1RXD, ADC2_CH7
25	GPIO19	I/O/T	3V3-RTC	RTC_GPIO19, GPIO19, U1RTS, ADC2_CH8, USB_D-
26	GPIO26	I/O/T	VDD	SPICS1, GPIO26
27	3.3V-LORA	P	-	SX1262 Analog power supply
28	VddRTC	P	-	RTC RV-3028-C7 Analog power supply
29	CLK	O	VddRTC	Clock Output(RV-3028-C7)
30	VIN-LED	P	-	WS2812B Analog power supply
31	GPIO48	I/O/T	3V3-RTC	SPICLK_N_DIFF, GPIO48, SUBSPICLK_N_DIFF
32	GPIO21	I/O/T	3V3-RTC	RTC_GPIO21, GPIO21
33	VDD	P	-	Output power supply: 1.8 V or VDD3P3_RTC
34	GPIO20	I/O/T	3V3-RTC	RTC_GPIO20, GPIO20, ADC2_CH9, CLK_OUT1, USB_D+
35	GPIO4	I/O/T	3V3-RTC	**LORA_RST**
36	GPIO5	I/O/T	3V3-RTC	**LORA_NSS**
37	GPIO27	I/O/T	VDD	SPIHD, GPIO27
38	GPIO28	I/O/T	VDD	SPIWP, GPIO28
39	GPIO29	I/O/T	VDD	SPICS0 , GPIO29
40	GPIO30	I/O/T	VDD	SPICLK, GPIO30
41	GPIO31	I/O/T	VDD	SPIQ, GPIO31
42	GPIO32	I/O/T	VDD	SPID, GPIO32
43	GPIO33	I/O/T	3V3-CPU/VDD	SPII04, GPIO33, FSPIHD, SUBSPIHD
44	GPIO34	I/O/T	3V3-CPU/VDD	SPII05, GPIO34, FSPICS0, SUBSPICS0
45	GPIO37	I/O/T	3V3-CPU/VDD	SPIDQS, GPIO37, FSPIQ, SUBSPIQ
46	GPIO38	I/O/T	3V3-RTC	GPIO38, FSPIWP, SUBSPIWP
47	GPIO47	I/O/T	VDD	**LORA_DIO1**, SPICLK_P_DIFF, GPIO47
48	GPIO39	I/O/T	3V3-CPU	**MTCK**, GPIO39, CLK_OUT3, SUBSPICS1
49	GPIO40	I/O/T	3V3-CPU	**MTDO**, GPIO40, CLK_OUT2
50	GPIO45	I/O/T	3V3-CPU	**LORA_SCK**
51	GND	P	-	Ground
52	GND	P	-	Ground
53	GPIO36	I/O/T	3V3-CPU/VDD	**LORA_MOSI**, SPII07, GPIO36, FSPICLK, SUBSPICLK
54	GPIO46	I/O/T	3V3-CPU	**LORA_BUSY**, GPIO46
55	RXD	I/O/T	3V3-CPU	GPIO44, CLK_OUT2
56	TXD	I/O/T	3V3-CPU	GPIO43, CLK_OUT1

No.	Name	Type[a]	Power Domain	Function
57	GPIO35	I/O/T	3V3-CPU/VDD	**LORA_MISO**, SPIIO6, GPIO35, FSPID, SUBSPID
58	GPIO42	I/O/T	3V3-CPU	**MTMS**, GPIO42
59	GPIO41	I/O/T	3V3-CPU	**MTDI**, GPIO41, CLK_OUT1
60	GND	P	-	Ground
61	GND	P	-	Ground
62	GND	P	-	Ground
63	GND	P	-	Ground
64	GND	P	-	Ground
65	GND	P	-	Ground
66	GND	P	-	Ground
67	GND	P	-	Ground

[a] **P**: power supply; **RF**: radio frequency; **PL**: power latch; **I**: input; **O**: output; **T**: high impedance. Pin functions in **bold** font are the default pin functions.

3.2 | Strapping pins

At each startup or reset, a module requires some initial configuration parameters, such as in which boot mode to load the module, voltage of flash memory, etc. These parameters are passed over via the strapping pins. After reset, the strapping pins operate as regular IO pins. The parameters controlled by the given strapping pins at module reset are as follows:

- Chip boot mode – GPIO0 and GPIO46
- VDD_SPI voltage – GPIO45
- ROM messages printing – GPIO46
- JTAG signal source – GPIO3

GPIO0, GPIO45, and GPIO46 are connected to the chip's internal weak pull-up/pull-down resistors at chip reset. These resistors determine the default bit values of the strapping pins. Also, these resistors determine the bit values if the strapping pins are connected to an external high-impedance circuit.

For more details please read the official documentation of Espressif (ESP32-S3FN8).

3.3 | Pins used

Some GPIOs of the ESP32-S3FN8 are used to handle some module features.

- SDA - GPIO8
- SCL - GPIO9
- Auto turn off - GPIO14
- Battery level – GPIO7

4 | Electrical characteristics

4.1 | Absolute maximum ratings

To ensure the proper operation of the ELPM-S3LW, it is recommended that the ranges of the Operating Conditions be observed.

Symbol	Parameter	Min	Max	Unit
3V3	Power supply voltage	3.0	3.6	V
Vin	Power latch or Battery input	3.0	5.5	V
LDO-LORA	LDO2 input	3.0	5.5	V
3.3V-LORA	3.3V power path for LoRa ICs	3.0	3.5	V
VddRTC	Power path for RTC(RV-3028-C7)	3.0	3.3	V

4.2 | Maximum ESD ratings

Parameter	Min	Max	Unit
ESD immunity	±2	±12*	kV

*Maximum protention guaranteed only on the 3.3V.

4.3 | Operating temperature range

Parameter	Min	Max	Unit
Operating temperature	-40	+85	°C

4.4 | Power consumption

Work mode	Description	Temp	Vin	Min	Typ	Unit
Deep Stop (RTC disabled)	Master, gate/Forceon, wake input triggering the power latch. RTC not powered.	+20°C to +30°C	3.3V	<1	5	nA
Deep Stop with RTC	RTC powered. Multiple interrupt sources.	+20°C to +30°C	3.3V	40	80	nA
Deep sleep	Default deep sleep mode of ESP32-S3.	+20°C to +30°C	3.3V	6	10	µA

4.5 | Startup time

The power latch is capable of booting the whole system in a few milliseconds ensuring high reactivity. It is possible to reduce the startup time of the ESP32-S3 boot by tuning the boot options.

Interrupt mode	Description	typ	Unit
Master Platch	Change of input status.	30	ms
RTC Wake	Programmed awakening.	80	ms
Gate/FORCEON	Gate input to GND.	20	ms
Wake	Wake input to GND.	500	ms

ESP32-S3FN8	typ	Unit
Default startup time	105	ms

5 | Deep Stop

Smart power unit designed to reduce energy waste from the ELPM and all components powered by it. This mode of module operation is not intended to replace the deep sleep mode of the ESP32S3, but is an additional solution that can be tailored during the development of a low-power device.

6 | Deep Stop performance

6.1 | Master/Wake mode

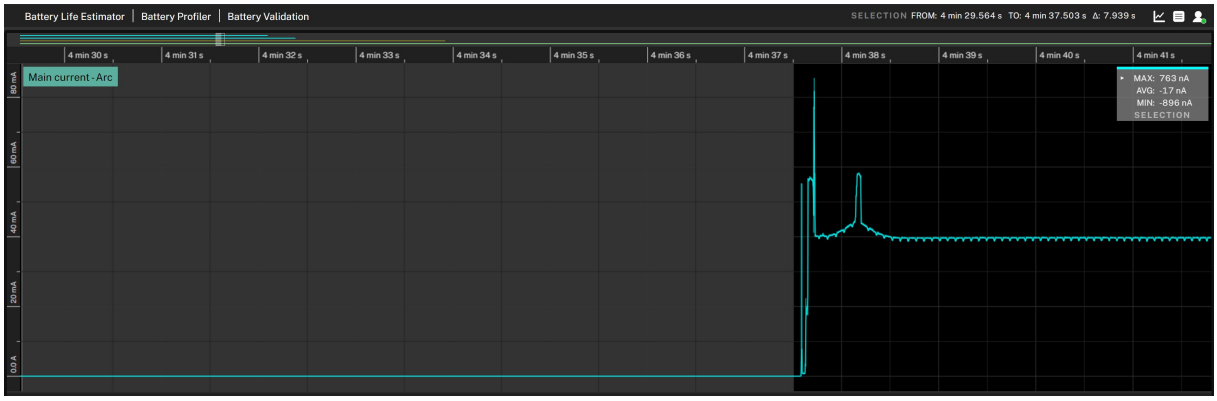


Figure 6.1: Deep stop

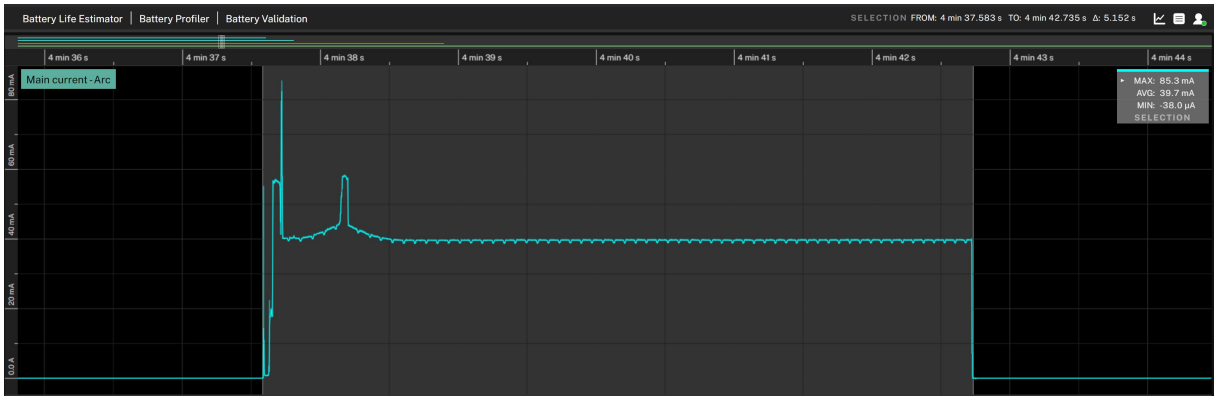


Figure 6.2: System startup

6.2 | RTC mode

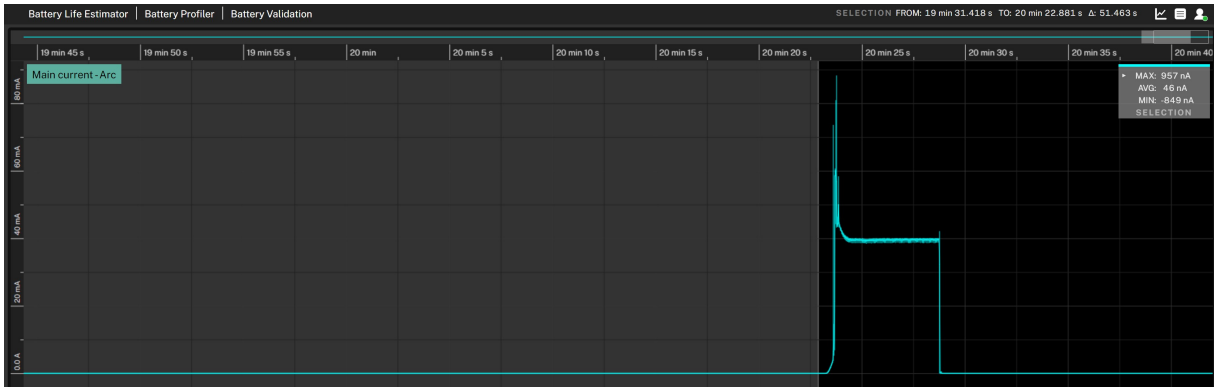


Figure 6.3: Deep stop

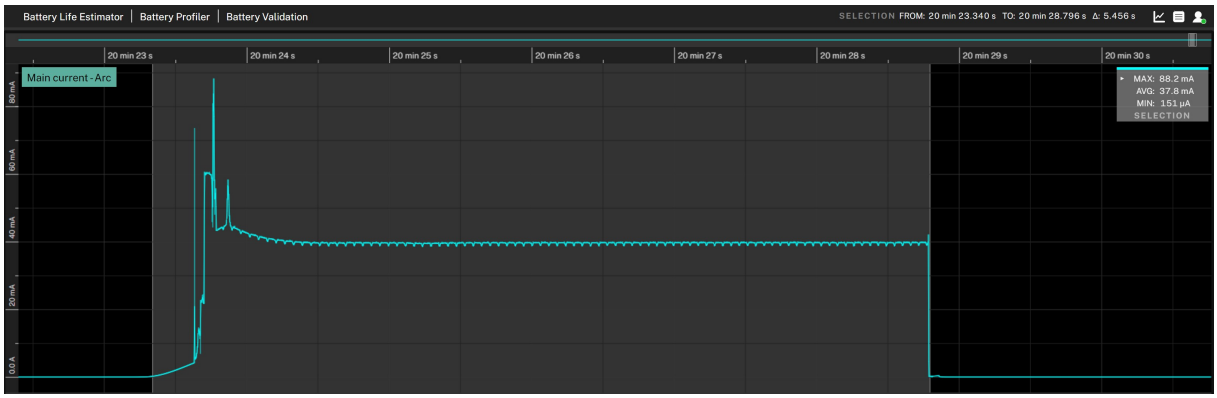


Figure 6.4: System startup

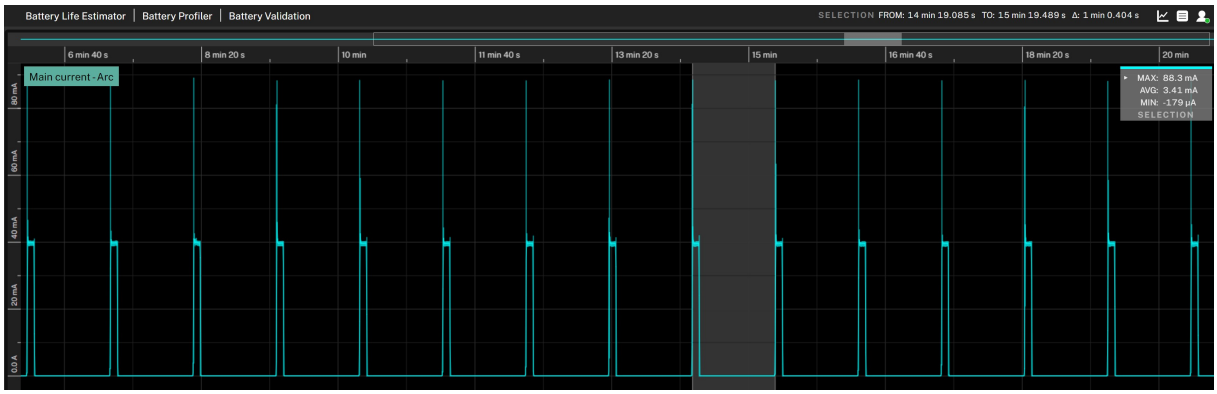


Figure 6.5: Example: Wake-up every minute

7 | Example Firmware

7.1 | Arduino IDE

RGB Led DEMO

```
/*
  RGB Led DEMO

  This code controls the builtin RGB status led of the OBJEX ELPM module.
  It uses the Adafruit_NeoPixel library to manage the LED. The code defines
  a function called "rainbow" that cycles the LED through a rainbow color spectrum.
  The "loop" function continuously calls the "rainbow" function with a delay
  of 10 milliseconds, creating a smooth rainbow animation.
*/

#include <Adafruit_NeoPixel.h>

#define LED_PIN    48   // GPIO connected to RGB status led (WS2812B)
#define LED_COUNT 1

Adafruit_NeoPixel led(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);  // Create a Adafruit_NeoPixel object to
     control the LED

void rainbow(int wait) {
  for(long firstPixelHue = 0; firstPixelHue < 5*65536; firstPixelHue += 256) {
    led.rainbow(firstPixelHue);
    led.show();
    delay(wait);
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("ELPM RGB TEST");

  led.begin();  // Initialize NeoPixel led object
  led.show();   // Turn OFF all pixels
  led.setBrightness(100);
}

void loop() {
  rainbow(10);  // Run the rainbow effect
}
```

Power latch master/wake DEMO

```
/*
  Power latch master/wake DEMO

  The code enables interaction with the Master and Wake pins of the powerlatch
  In the setup phase, it initializes serial communication, configures the power latch to prevent accidental
     shutdown, and sets the initial LED color to a dim green. The loop function simply waits five seconds
     before triggering the power latch to turn off the entire module.
*/

#include <Adafruit_NeoPixel.h>

#define LED_PIN    48   // GPIO connected to RGB status led (WS2812B)
#define NUMPIXELS   1

Adafruit_NeoPixel led(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);  // Create a Adafruit_NeoPixel object to
     control the LED

#define AUTO_TURN_OFF 14  // GPIO connected to power latch
#define BattLVL 7  // GPIO connected to battery level circuit

void setup() {
  // Power latch setup
  pinMode(AUTO_TURN_OFF, OUTPUT);
  digitalWrite(AUTO_TURN_OFF, HIGH);  // Setting the AUTO_TURN_OFF pin HIGH prevents module shutdown. Do this
     quickly, or the power latch may cut off power.
```

```
  // Start serial communication
  Serial.begin(115200);
  Serial.println("system started");

  // Initialize NeoPixel library
  led.begin();  // Initialize NeoPixel led object
  pixels.setPixelColor(0, pixels.Color(0, 50, 0));  // Set a color following RGB standard (R,G,B) ranging from
     0 to 255
  pixels.show();
}

void loop() {
  delay(5000);
  digitalWrite(AUTO_TURN_OFF, LOW);  // Setting the AUTO_TURN_OFF pin LOW will trigger the power latch to turn
     off the module.
}
```

Power latch RTC Interrupt

```
#include <Adafruit_NeoPixel.h>
#include <Melopero_RV3028.h>

Melopero_RV3028 rtc;

#define PIN        48  // GPIO connected to RGB status led (WS2812B)
#define NUMPIXELS   1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);  // Create a Adafruit_NeoPixel object to
     control the LED

#define AUTO_TURN_OFF 14  // GPIO connected to power latch
#define BattLVL 7  // GPIO connected to battery level circuit

void setup() {
  // Power lacth setup
  pinMode(AUTO_TURN_OFF, OUTPUT);
  digitalWrite(AUTO_TURN_OFF, HIGH);  // Setting the AUTO_TURN_OFF pin HIGH prevents module shutdown. Do this
     quickly, or the power latch may cut off power.

  // Start serial communication
  Serial.begin(115200);

  // Initialize NeoPixel library
  led.begin();  // Initialize NeoPixel led object
  pixels.setPixelColor(0, pixels.Color(0, 50, 0));  // Set a color following RGB standard (R,G,B) ranging from
     0 to 255
  pixels.show();

  Wire.begin(8, 9); // SDA | SCL

  rtc.initI2C();

  // Set the device to use the 24hour format (default) instead of the 12 hour format
  rtc.set24HourMode();

  // Set the date and time:
  rtc.setTime(2024, 11, 1, 1, 17, 34, 0);
  // year, month, weekday, date, hour, minute, second
  // Note: time is always set in 24h format
  // Note: month value ranges from 1 (Jan) to 12 (Dec)
  // Note: date value ranges from 1 to 31

  // set the interrupt to trigger every day at 1 minute from now
  // since we set the hour to 15:20 we don't have to check if the
  // minute value exceeds 59...
  rtc.setDateModeForAlarm(false);
  rtc.enableAlarm(
    0, //weekdayOrDate not important
    rtc.getHour(), //hour
    rtc.getMinute() + 1, //minute
    false, //dateAlarm
    true, //hourAlarm
    true, //minuteAlarm
```

```
    true //generateInterrupt
  );

  Serial.print(rtc.getHour());
  Serial.print(":");
  Serial.print(rtc.getMinute());
  Serial.print(":");
  Serial.print(rtc.getSecond());
  Serial.print("   ");
  Serial.print(rtc.getDate());
  Serial.print("/");
  Serial.print(rtc.getMonth());
  Serial.print("/");
  Serial.println(rtc.getYear());

  delay(1000);
  digitalWrite(AUTO_TURN_OFF, LOW);  //Setting the AUTO_TURN_OFF pin LOW will trigger the power latch to turn
     off the module.
}

void loop() {

}
```

LoRaWAN DEMO

```
#include "config.h"

int LORA_SS = 5; // LORA SPI CS
int LORA_SCLK = 45; // LORA SPI CLK
int LORA_MISO = 35; // LORA SPI MISO
int LORA_MOSI = 36; // LORA SPI MOSI

int count = 0;

void setup() {
  // Start serial communication
  Serial.begin(115200);
  Serial.println("LoRaWAN DEMO");

  SPI.begin(LORA_SCLK, LORA_MISO, LORA_MOSI, LORA_SS);

  Serial.println(F("\nSetup ... "));
  Serial.println(F("Initalise the radio"));

  //radio.setDio2AsRfSwitch(true);
  if (radio.setDio2AsRfSwitch() != RADIOLIB_ERR_NONE) {
    Serial.println(F("Failed to set DIO2 as RF switch!"));
    while (true);
  }

  int state = radio.begin(868.0, 125.0, 9, 7, RADIOLIB_SX126X_SYNC_WORD_PRIVATE, 10, 8, 0, false);
  debug(state != RADIOLIB_ERR_NONE, F("Initalise radio failed"), state, true);

  Serial.println(F("Join ('login') to the LoRaWAN Network"));
  state = node.beginOTAA(joinEUI, devEUI, nwkKey, appKey, false);
  debug(state < RADIOLIB_ERR_NONE, F("Join failed"), state, false);
  Serial.println(F("Ready!\n"));
 }

 void loop() {
  Serial.println(F("Sending uplink"));
  Serial.print(F("[LoRaWAN] Sending uplink packet ... "));
  String strUp = "Hello!" + String(count++);
  String strDown;
  int state = node.sendReceive(strUp, 10, strDown);
  debug((state != RADIOLIB_ERR_RX_TIMEOUT) && (state != RADIOLIB_ERR_NONE), F("Error in sendReceive"), state,
     false);

  Serial.print(F("Uplink complete, next in "));
  Serial.print(uplinkIntervalSeconds);
  Serial.println(F(" seconds"));
 }
```

■ Related config.h file

```
#ifndef _CONFIG_H
#define _CONFIG_H

#include <RadioLib.h>

// How often to send an uplink - consider legal & FUP constraints - see notes
const uint32_t uplinkIntervalSeconds = 5UL * 60UL;    // minutes x seconds

// JoinEUI - previous versions of LoRaWAN called this AppEUI
// for development purposes you can use all zeros - see wiki for details
#define RADIOLIB_LORAWAN_JOIN_EUI  0x0000000000000000

// The Device EUI & two keys can be generated on the TTN console
#ifndef RADIOLIB_LORAWAN_DEV_EUI   // Replace with your Device EUI
#define RADIOLIB_LORAWAN_DEV_EUI   0x0000000000000000
#endif
#ifndef RADIOLIB_LORAWAN_APP_KEY   // Replace with your App Key
#define RADIOLIB_LORAWAN_APP_KEY   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
     , 0x00, 0x00, 0x00
#endif
#ifndef RADIOLIB_LORAWAN_NWK_KEY   // Put your Nwk Key here
#define RADIOLIB_LORAWAN_NWK_KEY   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
     , 0x00, 0x00, 0x00
#endif

// Regional choices: EU868, US915, AU915, AS923, IN865, KR920, CN780, CN500
const LoRaWANBand_t Region = EU868;
const uint8_t subBand = 0;  // For US915, change this to 2, otherwise leave on 0

// SX1262  pin order: Module(NSS/CS, DIO1, RESET, BUSY);
SX1262 radio = new Module(5, 47, 4, 46);

// Copy over the EUI's & keys in to the something that will not compile if incorrectly formatted
uint64_t joinEUI =   RADIOLIB_LORAWAN_JOIN_EUI;
uint64_t devEUI  =   RADIOLIB_LORAWAN_DEV_EUI;
uint8_t appKey[] = { RADIOLIB_LORAWAN_APP_KEY };
uint8_t nwkKey[] = { RADIOLIB_LORAWAN_NWK_KEY };

// Create the LoRaWAN node
LoRaWANNode node(&radio, &Region, subBand);

// Helper function to display any issues
void debug(bool isFail, const __FlashStringHelper* message, int state, bool Freeze) {
  if (isFail) {
    Serial.print(message);
    Serial.print("(");
    Serial.print(state);
    Serial.println(")");
    while (Freeze);
  }
}

// Helper function to display a byte array
void arrayDump(uint8_t *buffer, uint16_t len) {
  for (uint16_t c; c < len; c++) {
    Serial.printf("%02X", buffer[c]);
  }
  Serial.println();
}
#endif
```

LoRa Ping-Pong

```
#include <RadioLib.h>

//#define INITIATING_NODE //ANCHOR module 1 with INITIATING_NODE and module 2 with "//INITIATING_NODE"

int LORA_SS = 5;   // LORA SPI CS
int LORA_SCLK = 45;  // LORA SPI CLK
```

```
int LORA_MISO = 35;  // LORA SPI MISO
int LORA_MOSI = 36;  // LORA SPI MOSI

// SX1262 has the following connections:
SX1262 radio = new Module(5, 47, 4, 46);

// save transmission states between loops
int transmissionState = RADIOLIB_ERR_NONE;

//flag to indicate transmission or reception state
bool transmitFlag = false;

// flag to indicate that a packet was sent or received
volatile bool operationDone = false;

// this function is called when a complete packet
// is transmitted or received by the module
// IMPORTANT: this function MUST be 'void' type
//            and MUST NOT have any arguments!
void setFlag(void) {
  // we sent or received a packet, set the flag
  operationDone = true;
}

void setup() {
  // Start serial communication
  Serial.begin(115200);
  Serial.println("Power delivery PPS 3.0 DEMO");

  SPI.begin(LORA_SCLK, LORA_MISO, LORA_MOSI, LORA_SS);

  // initialize SX1262 with default settings
  Serial.print(F("[SX1262] Initializing ... "));
  //int state = radio.begin();
  radio.setDio2AsRfSwitch(true);
  if (radio.setDio2AsRfSwitch() != RADIOLIB_ERR_NONE) {
    Serial.println(F("Failed to set DIO2 as RF switch!"));
    while (true);
  }
  //int state = radio.begin(868.0, 125.0, 9, 7, RADIOLIB_SX126X_SYNC_WORD_PRIVATE, 10, 8, 0, true);
  int state = radio.begin(868.0, 125.0, 9, 7, RADIOLIB_SX126X_SYNC_WORD_PRIVATE, 10, 8, 0, false); //v1.2 -
     more stable

  if (state == RADIOLIB_ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true);
  }

  // set the function that will be called
  // when new packet is received
  radio.setDio1Action(setFlag);

  #if defined(INITIATING_NODE)
    // send the first packet on this node
    Serial.print(F("[SX1262] Sending first packet ... "));
    transmissionState = radio.startTransmit("Hello World!");
    transmitFlag = true;
  #else
    // start listening for LoRa packets on this node
    Serial.print(F("[SX1262] Starting to listen ... "));
    state = radio.startReceive();
    if (state == RADIOLIB_ERR_NONE) {
      Serial.println(F("success!"));
    } else {
      Serial.print(F("failed, code "));
      Serial.println(state);
      while (true);
    }
  #endif
}
```

```
void loop() {
  // check if the previous operation finished
  if(operationDone) {
    // reset flag
    operationDone = false;
    if(transmitFlag) {
      // the previous operation was transmission, listen for response
      // print the result
      if (transmissionState == RADIOLIB_ERR_NONE) {
        // packet was successfully sent
        Serial.println(F("transmission finished!"));
      } else {
        Serial.print(F("failed, code "));
        Serial.println(transmissionState)
      }
      // listen for response
      radio.startReceive();
      transmitFlag = false;
    } else {
      // the previous operation was reception
      // print data and send another packet
      String str;
      int state = radio.readData(str);

      if (state == RADIOLIB_ERR_NONE) {
        // packet was successfully received
        Serial.println(F("[SX1262] Received packet!"));

        // print data of the packet
        Serial.print(F("[SX1262] Data:\t\t"));
        Serial.println(str);

        // print RSSI (Received Signal Strength Indicator)
        Serial.print(F("[SX1262] RSSI:\t\t"));
        Serial.print(radio.getRSSI());
        Serial.println(F(" dBm"));

        // print SNR (Signal-to-Noise Ratio)
        Serial.print(F("[SX1262] SNR:\t\t"));
        Serial.print(radio.getSNR());
        Serial.println(F(" dB"));
      }
      // wait a second before transmitting again
      delay(1000);

      // send another one
      Serial.print(F("[SX1262] Sending another packet ... "));
      transmissionState = radio.startTransmit("Hello World from OBJEX Link!");
      transmitFlag = true;
    }

  }
}
```

7.2 | PlatformIO

Since the code used by PlatformIO is identical to the code provided above for Arduino IDE, we are only attaching the platformio.ini files for the respective examples.

RGB Led DEMO

```
; PlatformIO Project Configuration File

[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
build_flags =
   -DARDUINO_USB_CDC_ON_BOOT=1
monitor_filters = send_on_enter, time, colorize, log2file
```

```
monitor_speed = 115200
monitor_rts = 0
monitor_dtr = 0
lib_deps = adafruit/Adafruit NeoPixel@^1.11.0
```

Power latch master/wake DEMO

```
; PlatformIO Project Configuration File

[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
monitor_filters = send_on_enter, time, colorize, log2file
monitor_speed = 115200
monitor_rts = 0
monitor_dtr = 0
lib_deps =
  adafruit/Adafruit NeoPixel@^1.11.0
```

Power latch RTC Interrupt

```
; PlatformIO Project Configuration File

[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
monitor_filters = send_on_enter, time, colorize, log2file
monitor_speed = 115200
monitor_rts = 0
monitor_dtr = 0
lib_deps =
  melopero/Melopero RV3028@^1.1.0
  adafruit/Adafruit NeoPixel@^1.11.0
  constiko/RTC RV-3028-C7 Arduino Library@^2.1.0
```

LoRaWAN DEMO

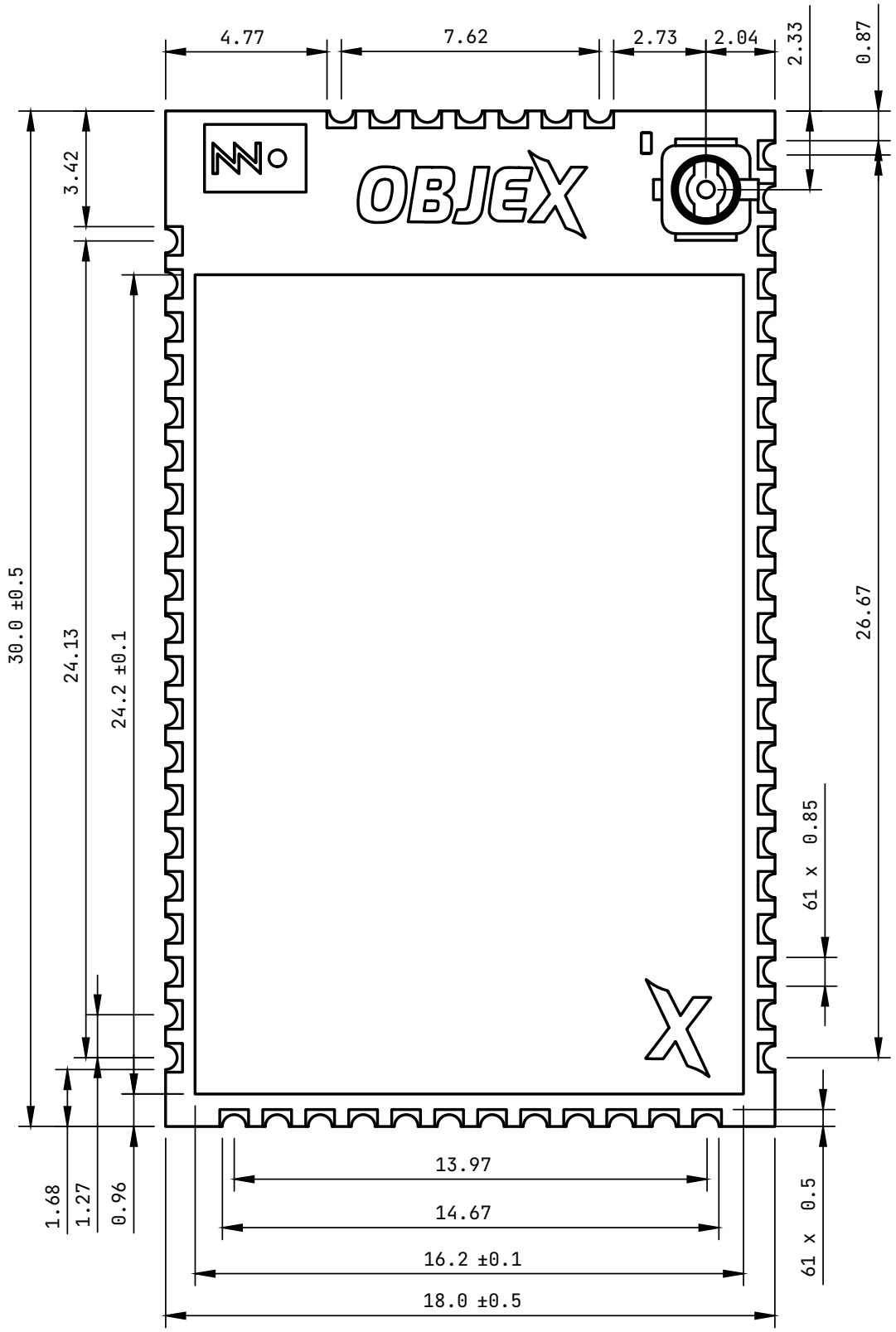
```
; PlatformIO Project Configuration File

[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
monitor_speed = 115200
build_flags = -DLIB_DEBUG=1
monitor_rts = 0
monitor_dtr = 0
lib_deps =
  jgromes/RadioLib@^6.4.2
```

LoRa Ping-Pong

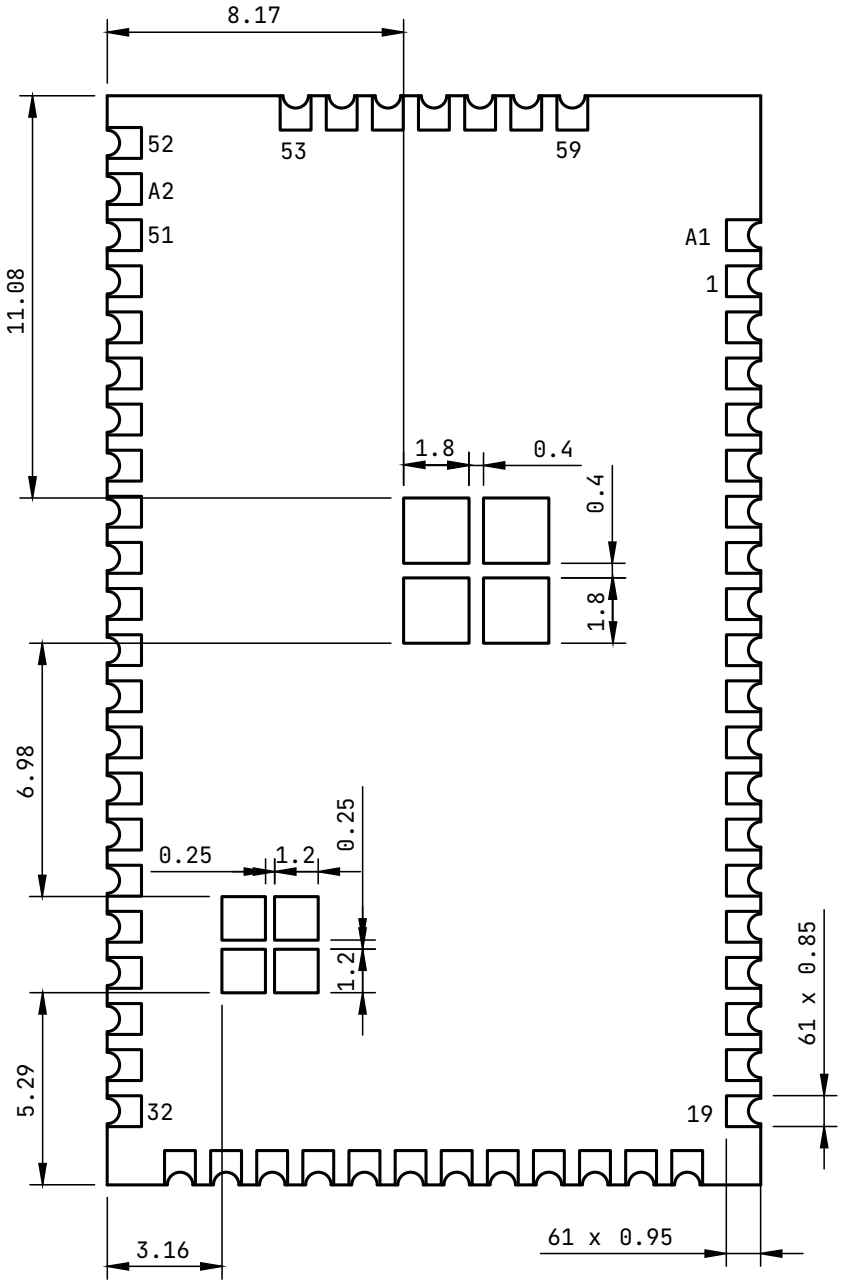
```
; PlatformIO Project Configuration File

[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
monitor_speed = 115200
build_flags = -DLIB_DEBUG=1
monitor_rts = 0
monitor_dtr = 0
lib_deps =
  jgromes/RadioLib@^6.4.2
```

8 | Physical Dimensions

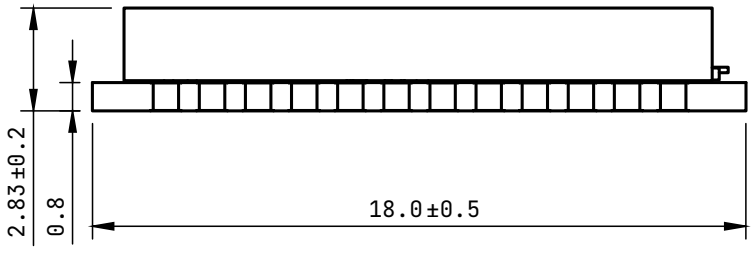
8.1 | Top view



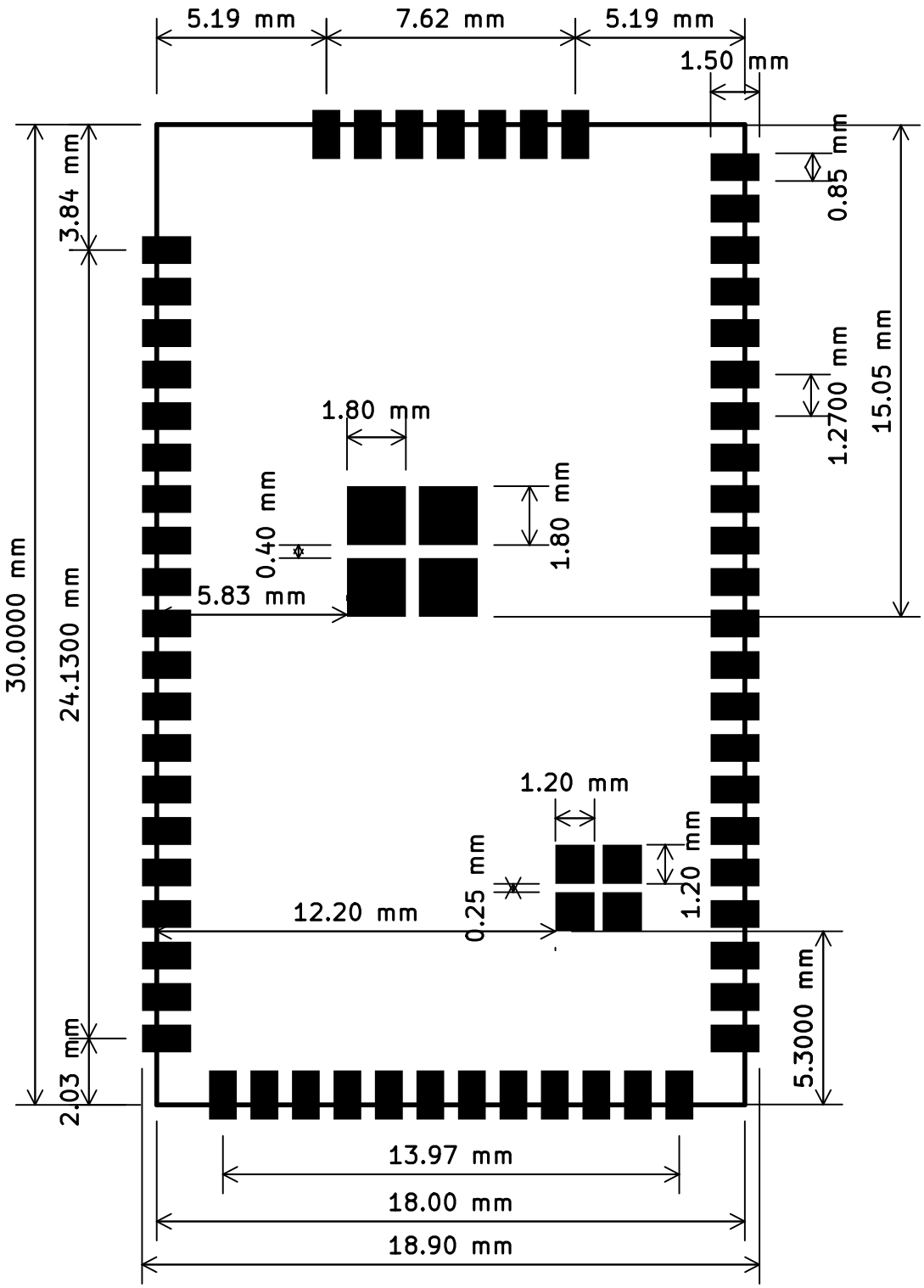
8.2 | Bottom view



8.3 | Side view



8.4 | Recommended PCB footprint



9 | Related documentation

- ESP32-S3 Series - Microcontroller
- SX1262 - LoRa IC
- RV-3028-C7 - Extreme low power ext. RTC

10 | Certification

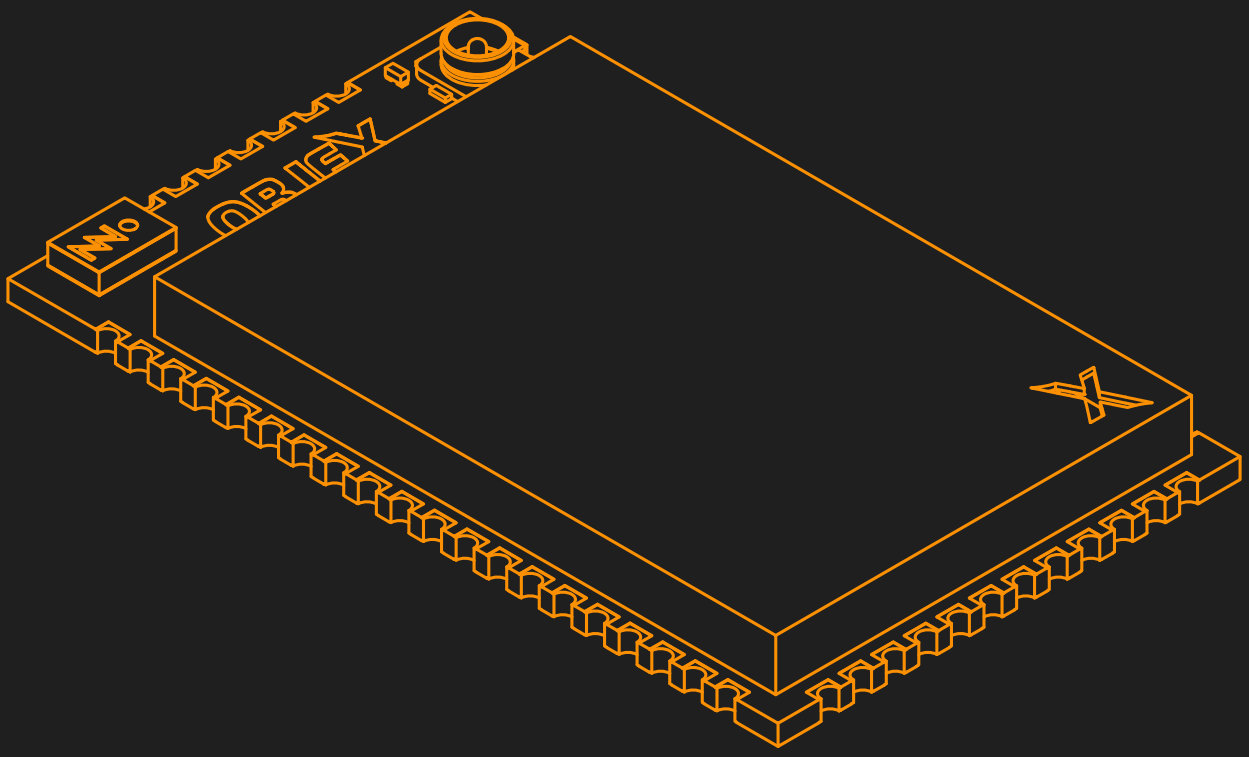
Pending release of the ELPM-S3LW version for mass production.

11 | Revision history

Revision	Date *(mm/dd/yyyy)*	Description
v1.1	07/06/2024	Add example Firmware section
v1.1	04/18/2024	Power consumption graphs, general bugfix (document still being finalized - draft)
v1.0	03/05/2024	Updated the value frequency bands 862MHz to 928MHz
v1.0	02/26/2024	Preliminary version for partners

OBJEX

BEHIND EVERY REVOLUTION



OBJEXLABS.COM