LwOW

Tilen MAJERLE

Mar 19, 2024

CONTENTS

1 Features **3**

2 Requirements **5**

3 Contribute **7**

4 License **9**

5 Table of contents **11**
- 5.1 Getting started . . . 11
- 5.2 User manual . . . 14
- 5.3 API reference . . . 35
- 5.4 Examples and demos . . . 55
- 5.5 Changelog . . . 57
- 5.6 Authors . . . 59

Index **61**

Welcome to the documentation for version latest-develop.

LwOW is lightweight, platform independent library for Onewire protocol for embedded systems. Its primary focus is UART hardware for physical communication for sensors and other slaves.

Download library Getting started Open Github Donate

CHAPTER

ONE

FEATURES

- Written in C (C11)
- Platform independent, uses custom low-level layer for device drivers
- 1-Wire protocol fits UART specifications at `9600` and `115200` bauds
- Allows standard one-wire single-gpio manual control (when UARTs are no more available by the system)
- Hardware is responsible for timing characteristics * Allows DMA on the high-performance microcontrollers
- Different device drivers included * DS18B20 temperature sensor is natively supported
- Works with operating system due to hardware timing management * Separate thread-safe API is available
- API for device scan, reading and writing single bits
- User friendly MIT license

CHAPTER

TWO

REQUIREMENTS

- C compiler
- Platform dependent drivers
- Few *kB* of non-volatile memory

CHAPTER

THREE

CONTRIBUTE

Fresh contributions are always welcome. Simple instructions to proceed:

1. Fork Github repository
2. Respect C style & coding rules used by the library
3. Create a pull request to `develop` branch with new features or bug fixes

Alternatively you may:

1. Report a bug
2. Ask for a feature request

CHAPTER

FOUR

LICENSE

```
MIT License

Copyright (c) 2024 Tilen MAJERLE

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Getting started

Getting started may be the most challenging part of every new library. This guide is describing how to start with the library quickly and effectively

5.1.1 Download library

Library is primarly hosted on Github.

You can get it by:

- Downloading latest release from releases area on Github
- Cloning `main` branch for latest stable version
- Cloning `develop` branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

This is used when you do not have yet local copy on your machine.

- Make sure `git` is installed.
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available options below
  - Run `git clone --recurse-submodules https://github.com/MaJerle/lwow` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwow` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch main https://github.com/MaJerle/lwow` to clone *latest stable* branch, including submodules
- Navigate to `examples` directory and run favourite example

Update cloned to latest version

- Open console and navigate to path in the system where your repository is located. Use command `cd your_path`
- Run `git pull origin main` command to get latest changes on `main` branch
- Run `git pull origin develop` command to get latest changes on `develop` branch
- Run `git submodule update --init --remote` to update submodules to latest version

Note: This is preferred option to use when you want to evaluate library and run prepared examples. Repository consists of multiple submodules which can be automatically downloaded when cloning and pulling changes from root repository.

5.1.2 Add library to project

At this point it is assumed that you have successfully download library, either cloned it or from releases page. Next step is to add the library to the project, by means of source files to compiler inputs and header files in search path.

CMake is the main supported build system. Package comes with the `CMakeLists.txt` and `library.cmake` files, both located in the `lwow` directory:

- `CMakeLists.txt`: Is a wrapper and only includes `library.cmake` file. It is used if target application uses `add_subdirectory` and then uses `target_link_libraries` to include the library in the project
- `library.cmake`: It is a fully configured set of variables. User must use `include(path/to/library.cmake)` to include the library and must manually add files/includes to the final target

Tip: Open `library.cmake` file and manually analyze all the possible variables you can set for full functionality.

If you do not use the *CMake*, you can do the following:

- Copy `lwow` folder to your project, it contains library files
- Add `lwow/src/include` folder to *include path* of your toolchain. This is where *C/C++* compiler can find the files during compilation process. Usually using `-I` flag
- Add source files from `lwow/src/` folder to toolchain build. These files are built by *C/C++* compiler. CMake configuration comes with the library, allows users to include library in the project as **subdirectory** and **library**.
- Copy `lwow/src/include/lwow/lwow_opts_template.h` to project folder and rename it to `lwow_opts.h`
- Build the project

5.1.3 Configuration file

Configuration file is used to overwrite default settings defined for the essential use case. Library comes with template config file, which can be modified according to the application needs. and it should be copied (or simply renamed in-place) and named `lwow_opts.h`

Tip: If you are using *CMake* build system, define the variable `LWOW_OPTS_FILE` before adding library's directory to the *CMake* project. Variable must contain the path to the user options file. If not provided and to avoid build error, one will be generated in the build directory.

Note: Default configuration template file location: `lwow/src/include/lwow/lwow_opts_template.h`. File must be renamed to `lwow_opts.h` first and then copied to the project directory where compiler include paths have access to it by using `#include "lwow_opts.h"`.

Configuration options list is available available in the *Configuration* section. If any option is about to be modified, it should be done in configuration file

Listing 1: Template configuration file

```
/**
 * \file            lwow_opts_template.h
 * \brief           LwOW application configuration
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#ifndef LWOW_OPTS_HDR_H
#define LWOW_OPTS_HDR_H

/* Rename this file to "lwow_opts.h" for your application */

/*
 * Open "include/lwow/lwow_opt.h" and
 * copy & replace here settings you want to change values
 */

```

(continues on next page)

(continued from previous page)

```
#endif /* LWOW_OPTS_HDR_H */
```

Note: If you prefer to avoid using configuration file, application must define a global symbol `LWOW_IGNORE_USER_OPTS`, visible across entire application. This can be achieved with `-D` compiler option.

5.2 User manual

5.2.1 How it works

LwOW library tends to use *UART* hardware of any microcontroller/embedded system, to generate timing clock for OneWire signals.

Nowaways embedded systems allow many UART ports, usually many more vs requirements for the final application. OneWire protocol needs precise timings and embedded systems (in 99.9%) do not have specific hardware to handle communication of this type.

Fortunately, OneWire timings match with UART timings at `9600` and `115200` bauds.

Note: Check detailed tutorial from Maxim for more information.

5.2.2 Thread safety

With default configuration, library is *not* thread safe. This means whenever it is used with operating system, user must resolve it with care.

Library has locking mechanism support for thread safety, which needs to be enabled manually.

Tip: To enable thread-safety support, parameter `LWOW_CFG_OS` must be set to `1`. Please check *Configuration* for more information about other options.

After thread-safety features has been enabled, it is necessary to implement `4` low-level system functions.

Tip: System function template example is available in `lwow/src/system/` folder.

Example code for `CMSIS-OS V2`

Note: Check *System functions* section for function description

Listing 2: System functions for CMSIS-OS based operating system

```
/**
 * \file            lwow_sys_cmsis_os.c
 * \brief           System functions for CMSIS-OS based operating system
 */
```

(continues on next page)

(continued from previous page)

```c

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#include "system/lwow_sys.h"

#if LWOW_CFG_OS && !__DOXYGEN__

#include "cmsis_os.h"

uint8_t
lwow_sys_mutex_create(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    const osMutexAttr_t attr = {
        .attr_bits = osMutexRecursive,
        .name = "lwow_mutex",
    };
    return (*m = osMutexNew(&attr)) != NULL;
}

uint8_t
lwow_sys_mutex_delete(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexDelete(*m) == osOK;
}

uint8_t
```

(continues on next page)

(continued from previous page)

```
lwow_sys_mutex_wait(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexAcquire(*m, osWaitForever) == osOK;
}

uint8_t
lwow_sys_mutex_release(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexRelease(*m) == osOK;
}

#endif /* LWOW_CFG_OS && !__DOXYGEN__ */
```

5.2.3 Hardware UART connection with sensor

To be able to successfully use sensors and other devices with embedded systems, these needs to be physically wired with embedded system (or PC).

Target devices (usually sensors or memory devices) are connected to master host device using single wire (from here protocol name *One Wire*) for communication only. There are also voltage and ground lines, marked as *VCC* and *GND*, respectively.

At this point, we assume you are familiar with UART protocol and you understand it has 2 independent lines, one for transmitting data (*TX*) and second to receive data (*RX*).

For successful communication with sensors, bi-directional support is necessary to be implemented, but there is only 1 wire available to do so. It might sound complicated at this point.

OneWire data line is by default in *open-drain* mode. This means that:

- Any device connected to data line can at any time pull line to *GND* without fear of short circuit
- None of the devices are allowed to force high state on the line. Application must use external *pull-up* resistor to do so.

How to send data over *TX* pin if application cannot force high level on the line? There are 2 options:

- Configure UART TX pin to *open-drain* mode
- Use *push-pull* to *open-drain* converter using *2 mosfets and 1 resistor*

Fig. 1: Push-pull to open-drain converter

Since many latest embedded systems allow you to configure *TX* pin to open-drain mode natively, you may consider second option instead.

Fig. 2: Embedded system with native open-drain TX pin support

Warning: Application must assure that *TX* pin is always configured to open-drain mode, either with *push-pull to open-drain* converter or directly configured in the system.

TX and RX pins

Every communication starts by master initiating it. To transfer data over UART, application uses TX pin and RX pin is used to read data. With 1-Wire protocol, application needs to transfer data and read them back in real-time. This is also called *loop-back* mode.

Let's take reset sequence as an example. By specifications, UART has to be configured in `9600` bauds and master needs to send single UART byte with value of `0xF0`. If there is any slave connected, slave must pull line to GND during transmision 0f `0xF` part of byte. Master needs to identify this by using *RX* pin of the UART.

Note: Please check official document on Maxim website to understand why `0xF0` and `9600` bauds.

5.2.4 Hardware connection with single pin

As the protocol name suggests, one-wire is a single wire protocol. When UART is not available in the system, and when timing constraints may not be a problem, it is possible to use library in classic mode with single GPIO manipulation.

Pin shall be in open-drain mode (when available) and include external pull-up resistor.

5.2.5 UART and 1-Wire timing relation

This part is explaining how UART and 1-Wire timings are connected together and what is important to take into consideration for stable and reliable communication.

1-Wire protocol specification match UART protocol specification when baudrate is configured at `115200` bauds. Going into the details about 1-Wire protocol, we can identify that:

- To send `1` logical *bit* at 1-Wire level, application needs to transmit `1` byte at UART level with speed of `115200` bauds
- To send `1` logical *byte* at 1-Wire level, application must transmit `8` bytes at UART level with speed of `115200` bauds

Fig. 3: UART byte time is equivalent to 1 bit at 1-Wire level

Timing for each bit is very clearly defined by 1-Wire specification (not purpose to go into these details) and needs to respect all low and high level states for reliable communication. Each bit at 1-Wire level starts with master pulling line low for specific amount of time. Until master initiates communication, line is in *idle* mode.

Image above shows relation between UART and 1-Wire timing. It represents transmission of `3` bits on 1-Wire level or `3` bytes at UART level. *Green* and *blue* rectangles show different times between ending of one bit and start of new bit.

Note: By 1-Wire specification, it is important to match bit timing. It is less important to match idle timings as these are not defined. Effectively this allows master to use UART to initiate byte transfer where UART takes care of proper timing.

Different timings (*green* vs *blue*) may happen if application uses many interrupts, but uses UART in polling mode to transmit data. This is very important for operating systems where context switch may disable interrupts. Fortunately, it is not a problem for reliable communication due to:

- When UART starts transmission, hardware takes care of timing

- If application gets preempted with more important task, 1-Wire line will be in idle state for longer time. This is not an issue by 1-Wire specification

More advanced embedded systems implement DMA controllers to support next level of transfers.

5.2.6 Porting guide

Implement low-level driver

Implementation of low-level driver is an essential part. It links middleware with actual hardware design of the device.

Its implementation must provide 4 functions:

- To open/configure UART hardware
- To set UART baudrate on the fly
- To transmit/receive data over UART
- To close/de-init UART hardware

After these functions have been implemented (check below for references), driver must link these functions to single driver structure of type *lwow_ll_drv_t*, later used during instance initialization.

Tip: Check *Low-level driver* for function prototypes.

Implement system functions

System functions are required only if operating system mode is enabled, with *LWOW_CFG_OS*.

Its implementation structure is not the same as for low-level driver, customer needs to implement fixed functions, with pre-defined name, starting with `lwow_sys_` name.

System function must only support OS mutex management and has to provide:

- *lwow_sys_mutex_create()* function to create new mutex
- *lwow_sys_mutex_delete()* function to delete existing mutex
- *lwow_sys_mutex_wait()* function to wait for mutex to be available
- *lwow_sys_mutex_release()* function to release (give) mutex back

Warning: Application must define *LWOW_CFG_OS_MUTEX_HANDLE* for mutex type. This shall be done in `lwow_opts.h` file.

Tip: Check *System functions* for function prototypes.

Example: Low-level driver for WIN32

Example code for low-level porting on *WIN32* platform. It uses native *Windows* features to open *COM* port and read/write from/to it.

Listing 3: Actual implementation of low-level driver for WIN32

```c
/**
 * \file            lwow_ll_win32.c
 * \brief           UART implementation for WIN32
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#include <stdio.h>
#include "lwow/lwow.h"
#include "windows.h"

#if !__DOXYGEN__

/* Function prototypes */
static uint8_t init(void* arg);
static uint8_t deinit(void* arg);
static uint8_t set_baudrate(uint32_t baud, void* arg);
static uint8_t transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg);

/* Win 32 LL driver for OW */
```

(continues on next page)

(continued from previous page)

```
const lwow_ll_drv_t lwow_ll_drv_win32 = {
    .init = init,
    .deinit = deinit,
    .set_baudrate = set_baudrate,
    .tx_rx = transmit_receive,
};

static HANDLE com_port;
static DCB dcb = {0};

static uint8_t
init(void* arg) {
    dcb.DCBlength = sizeof(dcb);
    LWOW_UNUSED(arg);

    /* Open virtual file as read/write */
    com_port = CreateFileA("\\\\.\\COM4", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_
↪EXISTING, 0, NULL);

    /* First read current values */
    if (GetCommState(com_port, &dcb)) {
        COMMTIMEOUTS timeouts;

        dcb.BaudRate = 115200;
        dcb.ByteSize = 8;
        dcb.Parity = NOPARITY;
        dcb.StopBits = ONESTOPBIT;

        /* Try to set com port data */
        if (!SetCommState(com_port, &dcb)) {
            printf("Cannot get COM port..\r\n");
            return 0;
        }

        if (GetCommTimeouts(com_port, &timeouts)) {
            /* Set timeout to return immediatelly from ReadFile function */
            timeouts.ReadIntervalTimeout = MAXDWORD;
            timeouts.ReadTotalTimeoutConstant = 0;
            timeouts.ReadTotalTimeoutMultiplier = 0;
            if (!SetCommTimeouts(com_port, &timeouts)) {
                printf("Cannot set COM PORT timeouts..\r\n");
            }
            GetCommTimeouts(com_port, &timeouts);
        } else {
            printf("Cannot get communication timeouts..\r\n");
            return 0;
        }
    } else {
        printf("Cannot get COM port info..\r\n");
        return 0;
    }

```

(continues on next page)

(continued from previous page)

```
    return 1;
}

uint8_t
deinit(void* arg) {
    /* Disable UART peripheral */
    LWOW_UNUSED(arg);

    return 1;
}

uint8_t
set_baudrate(uint32_t baud, void* arg) {
    LWOW_UNUSED(arg);
    /* Configure UART to selected baudrate */
    dcb.BaudRate = baud;

    /* Try to set com port data */
    if (!SetCommState(com_port, &dcb)) {
        printf("Cannot set COM port baudrate to %u bauds\r\n", (unsigned)baud);
        return 0;
    }

    return 1;
}

uint8_t
transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg) {
    /* Perform data exchange */
    size_t read = 0;
    DWORD br;
    LWOW_UNUSED(arg);

    if (com_port != NULL) {
        /*
         * Flush any data in RX buffer.
         * This helps to reset communication in case of on-the-fly device management
         * if one-or-more device(s) are added or removed.
         *
         * Any noise on UART level could start byte and put it to Win buffer,
         * preventing to read aligned data from it
         */
        PurgeComm(com_port, PURGE_RXCLEAR | PURGE_RXABORT);

        /* Write file and send data */
        WriteFile(com_port, tx, len, &br, NULL);
        FlushFileBuffers(com_port);

        /* Read same amount of data as sent previously (loopback) */
        do {
            if (ReadFile(com_port, rx, (DWORD)(len - read), &br, NULL)) {
                read += (size_t)br;
```

(continues on next page)

(continued from previous page)

```
                rx += (size_t)br;
            }
        } while (read < len);
    }

    return 1;
}

#endif /* !__DOXYGEN__ */
```

Example: Low-level driver for STM32

Example code for low-level porting on *STM32* platform.

Listing 4: Actual implementation of low-level driver for STM32

```
/**
 * \file            lwow_ll_stm32.c
 * \brief           Generic UART implementation for STM32 MCUs
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */

/*
```

(continues on next page)

(continued from previous page)

```
 * How it works
 *
 * https://docs.majerle.eu/projects/lwow/en/latest/user-manual/hw_connection.html#
 */
#include "lwow/lwow.h"

#if !__DOXYGEN__

static uint8_t init(void* arg);
static uint8_t deinit(void* arg);
static uint8_t set_baudrate(uint32_t baud, void* arg);
static uint8_t transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg);

/* STM32 LL driver for OW */
const lwow_ll_drv_t lwow_ll_drv_stm32 = {
    .init = init,
    .deinit = deinit,
    .set_baudrate = set_baudrate,
    .tx_rx = transmit_receive,
};

static LL_USART_InitTypeDef usart_init;

static uint8_t
init(void* arg) {
    LL_GPIO_InitTypeDef gpio_init;

    /* Peripheral clock enable */
    ONEWIRE_USART_CLK_EN;
    ONEWIRE_TX_PORT_CLK_EN;
    ONEWIRE_RX_PORT_CLK_EN;

    /* Configure GPIO pins */
    LL_GPIO_StructInit(&gpio_init);
    gpio_init.Mode = LL_GPIO_MODE_ALTERNATE;
    gpio_init.Speed = LL_GPIO_SPEED_FREQ_HIGH;
    gpio_init.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
    gpio_init.Pull = LL_GPIO_PULL_UP;

    /* TX pin */
    gpio_init.Alternate = ONEWIRE_TX_PIN_AF;
    gpio_init.Pin = ONEWIRE_TX_PIN;
    LL_GPIO_Init(ONEWIRE_TX_PORT, &gpio_init);

    /* RX pin */
    gpio_init.Alternate = ONEWIRE_RX_PIN_AF;
    gpio_init.Pin = ONEWIRE_RX_PIN;
    LL_GPIO_Init(ONEWIRE_RX_PORT, &gpio_init);

    /* Configure UART peripherals */
    LL_USART_DeInit(ONEWIRE_USART);
    LL_USART_StructInit(&usart_init);
```

(continues on next page)

(continued from previous page)

```
    usart_init.BaudRate = 9600;
    usart_init.DataWidth = LL_USART_DATAWIDTH_8B;
    usart_init.StopBits = LL_USART_STOPBITS_1;
    usart_init.Parity = LL_USART_PARITY_NONE;
    usart_init.TransferDirection = LL_USART_DIRECTION_TX_RX;
    usart_init.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
    usart_init.OverSampling = LL_USART_OVERSAMPLING_16;
    LL_USART_Init(ONEWIRE_USART, &usart_init);
    LL_USART_ConfigAsyncMode(ONEWIRE_USART);

    LWOW_UNUSED(arg);

    return 1;
}

static uint8_t
deinit(void* arg) {
    LL_USART_DeInit(ONEWIRE_USART);
    LWOW_UNUSED(arg);
    return 1;
}

static uint8_t
set_baudrate(uint32_t baud, void* arg) {
    usart_init.BaudRate = baud;
    LL_USART_Init(ONEWIRE_USART, &usart_init);
    LL_USART_ConfigAsyncMode(ONEWIRE_USART);
    LWOW_UNUSED(arg);

    return 1;
}

static uint8_t
transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg) {
    const uint8_t* t = tx;
    uint8_t* r = rx;

    /* Send byte with polling */
    LL_USART_Enable(ONEWIRE_USART);
    for (; len > 0; --len, ++t, ++r) {
        LL_USART_TransmitData8(ONEWIRE_USART, *t);
        while (!LL_USART_IsActiveFlag_TXE(ONEWIRE_USART)) {
            ;
        }
        while (!LL_USART_IsActiveFlag_RXNE(ONEWIRE_USART)) {
            ;
        }
        *r = LL_USART_ReceiveData8(ONEWIRE_USART);
    }
    while (!LL_USART_IsActiveFlag_TC(ONEWIRE_USART)) {}
    LL_USART_Disable(ONEWIRE_USART);
    LWOW_UNUSED(arg);
```

(continues on next page)

(continued from previous page)

```
    return 1;
}

#endif /* !__DOXYGEN__ */
```

Example: System functions for WIN32

Listing 5: Actual implementation of system functions for WIN32

```
/**
 * \file            lwow_sys_win32.c
 * \brief           System functions for WIN32
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#include "system/lwow_sys.h"
#include "windows.h"

#if LWOW_CFG_OS && !__DOXYGEN__

uint8_t
lwow_sys_mutex_create(LWOW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    LWOW_UNUSED(arg);
    *mutex = CreateMutex(NULL, 0, NULL);
```

(continues on next page)

(continued from previous page)

```
    return 1;
}

uint8_t
lwow_sys_mutex_delete(LWOW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    LWOW_UNUSED(arg);
    CloseHandle(*mutex);
    *mutex = NULL;
    return 1;
}

uint8_t
lwow_sys_mutex_wait(LWOW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    LWOW_UNUSED(arg);
    return WaitForSingleObject(*mutex, INFINITE) == WAIT_OBJECT_0;
}

uint8_t
lwow_sys_mutex_release(LWOW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    LWOW_UNUSED(arg);
    return ReleaseMutex(*mutex);
}

#endif /* LWOW_CFG_OS && !__DOXYGEN__ */
```

Example: System functions for CMSIS-OS

Listing 6: Actual implementation of system functions for CMSIS-OS

```
/**
 * \file            lwow_sys_cmsis_os.c
 * \brief           System functions for CMSIS-OS based operating system
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
```

(continues on next page)

(continued from previous page)

```
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#include "system/lwow_sys.h"

#if LWOW_CFG_OS && !__DOXYGEN__

#include "cmsis_os.h"

uint8_t
lwow_sys_mutex_create(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    const osMutexAttr_t attr = {
        .attr_bits = osMutexRecursive,
        .name = "lwow_mutex",
    };
    return (*m = osMutexNew(&attr)) != NULL;
}

uint8_t
lwow_sys_mutex_delete(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexDelete(*m) == osOK;
}

uint8_t
lwow_sys_mutex_wait(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexAcquire(*m, osWaitForever) == osOK;
}

uint8_t
lwow_sys_mutex_release(LWOW_CFG_OS_MUTEX_HANDLE* m, void* arg) {
    LWOW_UNUSED(arg);
    return osMutexRelease(*m) == osOK;
}

#endif /* LWOW_CFG_OS && !__DOXYGEN__ */
```

Low-Level driver for STM32 with STM32CubeMX

Specific low-level driver has been implemented for STM32 series of microcontrollers, to allow easy and simple link of LwOW library with projects generated with STM32CubeMX or STM32CubeIDE development tools.

Driver is based on HAL (Hardware Abstraction Layer) and it uses interrupt configuration to transmit/receive data. When customer starts a new project using CubeMX, it must:

- Configure specific UART IP as async mode both directions
- UART must have enabled global interrupts, to allow transmitting/receiving data using interrupts
- Application must pass pointer to UART handle when calling `lwow_init` function

---

Tip: Special example has been developed to demonstrate how can application use multiple OneWire instances on multiple UART ports at the same time. It uses custom argument to determine which UART handle shall be used for data transmit. Check `/examples/stm32/` folder for actual implementation.

Listing 7: Actual implementation of low-level driver for STM32 with HAL drivers

```
/**
 * \file            lwow_ll_stm32_hal.c
 * \brief           UART driver implementation for STM32 with HAL code
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
```

(continues on next page)

(continued from previous page)

```

/*
 * How it works (general)
 *
 * https://docs.majerle.eu/projects/lwow/en/latest/user-manual/hw_connection.html#
 *
 * This specific driver is optimized for proejcts generated by STM32CubeMX or␣
↪STM32CubeIDE with HAL drivers
 * It can be used w/ or w/o operating system and it uses interrupts & polling for data␣
↪receive and data transmit.
 *
 * Application must pass pointer to UART handle as argument to ow_init function in order
 * to link OW instance with actual UART hardware used for OW instance.
 *
 * To use this driver, application must:
 * - Enable interrupt in CubeMX to allow HAL_UART_Receive_IT functionality
 * - Use pointer to UART handle when initializing ow with ow_init
 */
#include "lwow/lwow.h"
#include "main.h" /* Generated normally by CubeMX */

#if !__DOXYGEN__

static uint8_t init(void* arg);
static uint8_t deinit(void* arg);
static uint8_t set_baudrate(uint32_t baud, void* arg);
static uint8_t transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg);

/* STM32 LL driver for OW */
const lwow_ll_drv_t lwow_ll_drv_stm32_hal = {
    .init = init,
    .deinit = deinit,
    .set_baudrate = set_baudrate,
    .tx_rx = transmit_receive,
};

static uint8_t
init(void* arg) {
    UART_HandleTypeDef* huart = arg;

    LWOW_ASSERT0("arg != NULL", arg != NULL);

    /* Initialize UART */
    HAL_UART_DeInit(huart);
    return HAL_UART_Init(huart) == HAL_OK;
}

static uint8_t
deinit(void* arg) {
    UART_HandleTypeDef* huart = arg;

    LWOW_ASSERT0("arg != NULL", arg != NULL);
```

(continues on next page)

(continued from previous page)

```

    return HAL_UART_DeInit(huart);
}

static uint8_t
set_baudrate(uint32_t baud, void* arg) {
    UART_HandleTypeDef* huart = arg;

    LWOW_ASSERT0("arg != NULL", arg != NULL);

    huart->Init.BaudRate = baud;
    return init(huart);
}

static uint8_t
transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg) {
    UART_HandleTypeDef* huart = arg;
    uint32_t start;

    LWOW_ASSERT0("arg != NULL", arg != NULL);

    /* Get current HAL tick */
    start = HAL_GetTick();

    /* Start RX in interrupt mode */
    HAL_UART_Receive_IT(huart, rx, len);

    /* Process TX in polling mode */
    HAL_UART_Transmit(huart, (void*)tx, len, 100);

    /* Wait RX to finish */
    while (huart->RxState != HAL_UART_STATE_READY) {
        if (HAL_GetTick() - start > 100) {
            return 0;
        }
    }

    return 1;
}

#endif /* !__DOXYGEN__ */
```

Low-Level driver for manual GPIO control

it is possible to use LwOW library even without available UARTs in the device (or if UARTs are being used for something else). Demo driver, that manipulates GPIO toggling is available in the repository.

Listing 8: LwOW low-level driver for manual GPIO control without UART

```c
/**
 * \file            lwow_ll_stm32_single_gpio_driver.c
 * \brief           Driver for non-UART use, with single GPIO
 */

/*
 * Copyright (c) 2024 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge,
 * publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be
 * included in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 * AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of LwOW - Lightweight onewire library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v3.0.2
 */
#include "stm32l4xx_hal.h"
/* And all other includes */

#if !__DOXYGEN__

/* Pin setting */
#define GPIO_CLK_EN     LL_AHB2_GRP1_PERIPH_GPIOB
#define GPIO_PORT       GPIOB
#define GPIO_PIN        LL_GPIO_PIN_13
#define OW_PIN_LOW      LL_GPIO_ResetOutputPin(GPIO_PORT, GPIO_PIN)
#define OW_PIN_HIGH     LL_GPIO_SetOutputPin(GPIO_PORT, GPIO_PIN)
#define OW_PIN_INPUT    LL_GPIO_SetPinMode(GPIO_PORT, GPIO_PIN, LL_GPIO_MODE_INPUT)
```

(continues on next page)

(continued from previous page)

```
#define OW_PIN_OUTPUT   LL_GPIO_SetPinMode(GPIO_PORT, GPIO_PIN, LL_GPIO_MODE_OUTPUT)

/* Macros for irq lock */
#define IRQ_LOCK_DEFINE uint32_t primask = __get_PRIMASK()
#define IRQ_LOCK        __disable_irq()
#define IRQ_UNLOCK      __set_PRIMASK(primask)

/* Function prototypes for driver */
static uint8_t prv_init(void* arg);
static uint8_t prv_deinit(void* arg);
static uint8_t prv_set_baudrate(uint32_t baud, void* arg);
static uint8_t prv_transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void*␣
↪arg);

/* Global driver structure for application use */
const lwow_ll_drv_t ow_driver_gpio = {
    .init = prv_init, .deinit = prv_deinit, .set_baudrate = prv_set_baudrate, .tx_rx =␣
↪prv_transmit_receive};
static uint32_t baudrate; /* Expected baudrate set by the application */

/**
 * \brief           Actual data exchange function
 *
 * This is the demo for STM32L4xx; with slight modifications it will work on any other␣
↪architecture.
 *
 * Requirements to be provided for application
 * - microseconds timing function (example below uses 16-bit timer as source)
 * - interrupt locking mechanism
 * - GPIO manupulation features (open-drain mode, pull, etc)
 *
 * \param           low_init_pulse_time: Time in microseconds for initial low pulse width
 * \param           pre_sample_time: Time in us to wait after pin release.
 *                      Bus is samples after time expiration
 * \param           post_sample_time: Time in us to wait after sample has been completed
 * \return          Bus value, `1 == high`, `0 == low`
 */
static uint8_t
prv_exch(uint16_t low_init_pulse_time, uint16_t pre_sample_time, uint16_t post_sample_
↪time) {
    uint8_t b = 0;
    uint16_t time, start_time;
    IRQ_LOCK_DEFINE;

    /* Lock interrupts and start execution */
    IRQ_LOCK;
    time = timebase_get_us_tick();

    /* Initiate start low pulse */
    start_time = time;
    OW_PIN_LOW;
    OW_PIN_OUTPUT;
```

(continues on next page)

(continued from previous page)

```
    while ((uint16_t)((time = timebase_get_us_tick()) - start_time) < low_init_pulse_
↪time) {}

    /* Release line and wait for mid pulse */
    start_time = time;
    OW_PIN_INPUT;
    while ((uint16_t)((time = timebase_get_us_tick()) - start_time) < pre_sample_time) {}

    /* Read pin state */
    b = LL_GPIO_IsInputPinSet(GPIO_PORT, GPIO_PIN);

    /* Interrupts can now be enabled from this point */
    IRQ_UNLOCK;

    /* Wait remaining time */
    start_time = time;
    while ((uint16_t)((time = timebase_get_us_tick()) - start_time) < post_sample_time)
↪{}

    return b;
}

/*******************************************/
/* LwOW driver interface functions         */
/*******************************************/
static uint8_t
prv_init(void* arg) {
    LL_GPIO_InitTypeDef gpio_init;

    /* Peripheral clock enable */
    LL_AHB2_GRP1_EnableClock(GPIO_CLK_EN);

    /* Configure GPIO pin with open-drain mode */
    LL_GPIO_StructInit(&gpio_init);
    gpio_init.Mode = LL_GPIO_MODE_INPUT;
    gpio_init.Speed = LL_GPIO_SPEED_FREQ_HIGH;
    gpio_init.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
    gpio_init.Pull = LL_GPIO_PULL_UP;
    gpio_init.Pin = GPIO_PIN;
    LL_GPIO_Init(GPIO_PORT, &gpio_init);

    LWOW_UNUSED(arg);
    return 1;
}

static uint8_t
prv_deinit(void* arg) {
    LWOW_UNUSED(arg);
    return 1;
}

static uint8_t
```

(continues on next page)

(continued from previous page)

```
prv_set_baudrate(uint32_t baud, void* arg) {
    LWOW_UNUSED(arg);
    baudrate = baud;
    return 1;
}

static uint8_t
prv_transmit_receive(const uint8_t* tx, uint8_t* rx, size_t len, void* arg) {
    const uint8_t* t = tx;
    uint8_t* r = rx;

    /*
     * For baudrate set at 9600 - by UART definition
     * this fits timing only for reset sequence at onewire level
     *
     * Length must always be zero, or error is returned
     */
    if (baudrate == 9600) {
        if (len == 1) {
            uint8_t v = prv_exch(480, 70, 410);
            *r = v ? *t : 0x01;
        } else {
            return 0;
        }
    } else if (baudrate == 115200) {
        /*
         * Regular transmission process
         *
         * Exchange values and set timings for different events,
         * according to the byte value to be transmitted
         */
        for (size_t i = 0; i < len; ++i, ++r, ++t) {
            uint8_t v = prv_exch(*t ? 6 : 60, *t ? 9 : 0, *t ? 55 : 10);

            /*
             * Set value as 0xFF in case of positive reading, 0x00 otherwise.
             * This is to be compliant with LwOW UART expectations
             */
            *r = v ? 0xFF : 0x00;
        }
    } else {
        return 0;
    }
    LWOW_UNUSED(arg);
    return 1;
}

#endif /* !__DOXYGEN__ */
```

5.3 API reference

List of all the modules:

5.3.1 LwOW

group **LWOW**

Lightweight onewire.

Note: Functions with `_raw` suffix do no implement locking mechanism when used with operating system.

Defines

LWOW_UNUSED(x)

Unused variable macro

LWOW_ASSERT(msg, c)

Assert check function.

It returns *lwowERRPAR* if condition check fails

Parameters

- **msg** – **[in]** Optional message parameter to print on failure
- **c** – **[in]** Condition to check for

LWOW_ASSERT0(msg, c)

Assert check function with return `0`

It returns `0` if condition check fails

Parameters

- **msg** – **[in]** Optional message parameter to print on failure
- **c** – **[in]** Condition to check for

LWOW_ARRAYSIZE(x)

Get size of statically declared array.

Parameters

- **x** – **[in]** Input array

Returns

Number of array elements

LWOW_CMD_RSCRATCHPAD

Read scratchpad command for 1-Wire devices

LWOW_CMD_WSCRATCHPAD

Write scratchpad command for 1-Wire devices

LWOW_CMD_CPYSCRATCHPAD

Copy scratchpad command for 1-Wire devices

LWOW_CMD_RECEEPROM

Read EEPROM command

LWOW_CMD_RPWRSUPPLY

Read power supply command

LWOW_CMD_SEARCHROM

Search ROM command

LWOW_CMD_READROM

Read ROM command

LWOW_CMD_MATCHROM

Match ROM command. Select device with specific ROM

LWOW_CMD_SKIPROM

Skip ROM, select all devices

Typedefs

typedef *lwowr_t* (***lwow_search_cb_fn**)(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, size_t index, void *arg)

Search callback function implementation.

Param ow
[in] 1-Wire handle

Param rom_id
[in] Rom address when new device detected. Set to `NULL` when search finished

Param index
[in] Current device index When `rom_id = NULL`, value indicates number of total devices found

Param arg
[in] Custom user argument

Return
lwowOK on success, member of *lwowr_t* otherwise

Enums

enum **lwowr_t**

1-Wire result enumeration

Values:

enumerator **lwowOK** = 0x00

Device returned OK

enumerator **lwowERRPRESENCE**

Presence was not successful

enumerator **lwowERRNODEV**

No device connected, maybe device removed during scan?

enumerator **lwowERRTXRX**

Error while exchanging data

enumerator **lwowERRBAUD**

Error setting baudrate

enumerator **lwowERRPAR**

Parameter error

enumerator **lwowERR**

General-Purpose error

Functions

lwowr_t **lwow_init**(*lwow_t* *const owobj, const *lwow_ll_drv_t* *const ll_drv, void *arg)

Initialize OneWire instance.

Parameters

- **owobj** – **[in]** OneWire instance
- **ll_drv** – **[in]** Low-level driver
- **arg** – **[in]** Custom argument

Returns

lwowOK on success, member of *lwowr_t* otherwise

void **lwow_deinit**(*lwow_t* *const ow)

Deinitialize OneWire instance.

Parameters

owobj – **[in]** OneWire instance

lwowr_t **lwow_protect**(*lwow_t* *const owobj, const uint8_t protect)

Protect 1-wire from concurrent access.

Note: Used only for OS systems

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **protect** – **[in]** Set to `1` to protect core, `0` otherwise

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_unprotect**(*lwow_t* *const owobj, const uint8_t protect)

Unprotect 1-wire from concurrent access, previously protected with *lwow_protect*.

Note: Used only for OS systems

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **protect** – **[in]** Set to `1` to unprotect core, `0` otherwise

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_reset_raw**(*lwow_t* *const owobj)

Reset 1-Wire bus and set connected devices to idle state.

Parameters

owobj – **[inout]** 1-Wire handle

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_reset**(*lwow_t* *const owobj)

Reset 1-Wire bus and set connected devices to idle state.

Note: This function is thread-safe

Parameters

owobj – **[inout]** 1-Wire handle

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_write_byte_ex_raw**(*lwow_t* *const owobj, const uint8_t btw, uint8_t *const byr)

Write byte over OW and read its response.

Parameters

- **owobj** – **[inout]** 1-Wire handle

- **btw** – **[in]** Byte to write
- **byr** – **[out]** Pointer to read value. Set to `NULL` if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_write_byte_ex**(*lwow_t* *const owobj, const uint8_t btw, uint8_t *const byr)
Write byte over OW and read its response.

Note: This function is thread-safe

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **btw** – **[in]** Byte to write
- **byr** – **[out]** Pointer to read value. Set to `NULL` if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_read_byte_ex_raw**(*lwow_t* *const owobj, uint8_t *const byr)
Read byte from OW device.

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **byr** – **[out]** Pointer to save read value

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_read_byte_ex**(*lwow_t* *const owobj, uint8_t *const byr)
Read byte from OW device.

Note: This function is thread-safe

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **byr** – **[out]** Pointer to save read value

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_read_bit_ex_raw**(*lwow_t* *const owobj, uint8_t *const byr)
Read sinle bit from OW device.

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **br** – **[out]** Pointer to save read value, either `1` or `0`

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_read_bit_ex**(*lwow_t* *const owobj, uint8_t *const byr)

Read sinle bit from OW device.

Note: This function is thread-safe

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **br** – **[out]** Pointer to save read value, either `1` or `0`

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_reset_raw**(*lwow_t* *const ow)

Reset search.

Parameters

owobj – **[inout]** 1-Wire handle

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_reset**(*lwow_t* *const ow)

Reset search.

Note: This function is thread-safe

Parameters

owobj – **[inout]** 1-Wire handle

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_raw**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id)

Search for devices on 1-wire bus.

Note: To reset search and to start over, use *lwow_search_reset* function

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **rom_id** – **[out]** Pointer to ROM structure to save ROM

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id)

Search for devices on 1-wire bus.

Note: To reset search and to start over, use *lwow_search_reset* function

Note: This function is thread-safe

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **rom_id** – **[out]** Pointer to ROM structure to save ROM

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_with_command_raw**(*lwow_t* *const owobj, const uint8_t cmd, *lwow_rom_t* *const rom_id)

Search for devices on 1-wire bus with custom search command.

Note: To reset search and to start over, use *lwow_search_reset* function

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **cmd** – **[in]** command to use for search operation
- **rom_id** – **[out]** Pointer to ROM structure to store address

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_with_command**(*lwow_t* *const owobj, const uint8_t cmd, *lwow_rom_t* *const rom_id)

Search for devices on 1-wire bus with custom search command.

Note: To reset search and to start over, use *lwow_search_reset* function

Note: This function is thread-safe

Parameters

- **owobj** – **[inout]** 1-Wire handle
- **cmd** – **[in]** command to use for search operation
- **rom_id** – **[out]** Pointer to ROM structure to store address

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_with_command_callback**(*lwow_t* *const owobj, const uint8_t cmd, size_t *const roms_found, const *lwow_search_cb_fn* func, void *const arg)

Search devices on 1-wire network by using callback function and custom search command.

When new device is detected, callback function `func` is called to notify user

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **cmd** – **[in]** 1-Wire search command
- **roms_found** – **[out]** Output variable to save number of found devices. Set to `NULL` if not used
- **func** – **[in]** Callback function to call for each device
- **arg** – **[in]** Custom user argument, used in callback function

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_with_callback**(*lwow_t* *const owobj, size_t *const roms_found, const *lwow_search_cb_fn* func, void *const arg)

Search devices on 1-wire network by using callback function and `SEARCH_ROM` 1-Wire command.

When new device is detected, callback function `func` is called to notify user

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **roms_found** – **[out]** Output variable to save number of found devices. Set to `NULL` if not used
- **func** – **[in]** Callback function to call for each device
- **arg** – **[in]** Custom user argument, used in callback function

Returns

lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_devices_with_command_raw**(*lwow_t* *const owobj, const uint8_t cmd, *lwow_rom_t* *const rom_id_arr, const size_t rom_len, size_t *const roms_found)

Search for devices on 1-Wire network with command and store ROM IDs to input array.

Parameters

- **owobj** – **[in]** 1-Wire handle
- **cmd** – **[in]** 1-Wire search command
- **rom_id_arr** – **[in]** Pointer to output array to store found ROM IDs into
- **rom_len** – **[in]** Length of input ROM array

- **roms_found** – **[out]** Output variable to save number of found devices. Set to NULL if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_devices_with_command**(*lwow_t* *const owobj, const uint8_t cmd, *lwow_rom_t* *const rom_id_arr, const size_t rom_len, size_t *const roms_found)

Search for devices on 1-Wire network with command and store ROM IDs to input array.

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **cmd** – **[in]** 1-Wire search command
- **rom_id_arr** – **[in]** Pointer to output array to store found ROM IDs into
- **rom_len** – **[in]** Length of input ROM array
- **roms_found** – **[out]** Output variable to save number of found devices. Set to NULL if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_devices_raw**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id_arr, const size_t rom_len, size_t *const roms_found)

Search for devices on 1-Wire network with default command and store ROM IDs to input array.

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id_arr** – **[in]** Pointer to output array to store found ROM IDs into
- **rom_len** – **[in]** Length of input ROM array
- **roms_found** – **[out]** Output variable to save number of found devices. Set to NULL if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_search_devices**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id_arr, const size_t rom_len, size_t *const roms_found)

Search for devices on 1-Wire network with default command and store ROM IDs to input array.

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id_arr** – **[in]** Pointer to output array to store found ROM IDs into

- **rom_len** – **[in]** Length of input ROM array
- **roms_found** – **[out]** Output variable to save number of found devices. Set to `NULL` if not used

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_match_rom_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Select device on 1-wire network with exact ROM number.

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to match device

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_match_rom**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Select device on 1-wire network with exact ROM number.

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to match device

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_match_or_skip_rom_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Select specific device or send skip ROM command, depending on the `rom_id` parameter.

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to match device or `NULL` to skip the match

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_match_or_skip_rom**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Select specific device or send skip ROM command, depending on the `rom_id` parameter.

Note: This function is thread-safe

Parameters

- **owobj** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to match device or `NULL` to skip the match

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_skip_rom_raw**(*lwow_t* *const owobj)

Skip ROM address and select all devices on the network.

Parameters
owobj – **[in]** 1-Wire handle

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_skip_rom**(*lwow_t* *const owobj)

Skip ROM address and select all devices on the network.

Note: This function is thread-safe

Parameters
owobj – **[in]** 1-Wire handle

Returns
lwowOK on success, member of *lwowr_t* otherwise

uint8_t **lwow_crc**(const void *const in, const size_t len)

Calculate CRC-8 of input data.

Note: This function is reentrant

Parameters

- **inp** – **[in]** Input data
- **len** – **[in]** Number of bytes

Returns
Calculated CRC

struct **lwow_rom_t**

#include <*lwow.h*> ROM structure.

Public Members

uint8_t **rom**[8]

8-bytes ROM address

struct **lwow_t**

#include <*lwow.h*> 1-Wire structure

Public Members

lwow_rom_t **rom**

ROM address of last device found. When searching for new devices, we always need last found address, to be able to decide which way to go next time during scan.

uint8_t **disrepancy**

Disrepancy value on last search

void ***arg**

User custom argument

const *lwow_ll_drv_t* ***ll_drv**

Low-level functions driver

LWOW_CFG_OS_MUTEX_HANDLE **mutex**

Mutex handle

5.3.2 Configuration

This is the default configuration of the middleware. When any of the settings shall be modified, it shall be done in dedicated application config `lwow_opts.h` file.

Note: Check *Getting started* for guidelines on how to create and use configuration file.

group **LWOW_OPT**

OW options.

Defines

LWOW_CFG_OS

Enables `1` or disables `0` operating system support in the library.

Note: When `LWOW_CFG_OS` is enabled, user must implement functions in *System functions* group.

LWOW_CFG_OS_MUTEX_HANDLE

Mutex handle type.

Note: This value must be set in case *LWOW_CFG_OS* is set to `1`. If data type is not known to compiler, include header file with definition before you define handle type

LWOW_MEMSET(dst, val, len)

Memory set function.

Note: Function footprint is the same as memset

LWOW_MEMCPY(dst, src, len)

Memory copy function.

Note: Function footprint is the same as memcpy

5.3.3 Platform specific

List of all the modules:

Low-level driver

group **LWOW_LL**

Low-level device dependant functions.

struct **lwow_ll_drv_t**

#include <*lwow.h*> 1-Wire low-level driver structure

Public Members

uint8_t (***init**)(void *arg)

Initialize low-level driver.

Param arg

[in] Custom argument passed to *lwow_init* function

Return

1 on success, `0` otherwise

uint8_t (***deinit**)(void *arg)

De-initialize low-level driver.

Param arg

[in] Custom argument passed to *lwow_init* function

Return

1 on success, `0` otherwise

uint8_t (***set_baudrate**)(uint32_t baud, void *arg)

Set UART baudrate.

Param baud

[in] Baudrate to set in units of bauds, normally `9600` or `115200`

Param arg

[in] Custom argument passed to *lwow_init* function

Return
1 on success, `0` otherwise

uint8_t (***tx_rx**)(const uint8_t *tx, uint8_t *rx, size_t len, void *arg)

Transmit and receive bytes over UART hardware (or custom implementation)

Bytes array for `tx` is already prepared to be directly transmitted over UART hardware, no data manipulation is necessary.

At the same time, library must read received data on RX port and put it to `rx` data array, one by one, up to `len` number of bytes

Param tx
[in] Data to transmit over UART
Param rx
[out] Array to write received data to
Param len
[in] Number of bytes to exchange
Param arg
[in] Custom argument passed to *lwow_init* function
Return
1 on success, `0` otherwise

System functions

System function are used in conjunction with thread safety. These are required when operating system is used and multiple threads want to access to the same OneWire instance.

Tip: Check *Thread safety* and *Porting guide* for instructions on how to port.

Below is a list of function prototypes and its implementation details.

group **LWOW_SYS**

System functions when used with operating system.

Functions

uint8_t **lwow_sys_mutex_create**(LWOW_CFG_OS_MUTEX_HANDLE *mutex, void *arg)

Create a new mutex and assign value to handle.

Parameters

- **mutex** – **[in]** Output variable to save mutex handle
- **arg** – **[in]** User argument passed on *lwow_init* function

Returns
1 on success, `0` otherwise

uint8_t **lwow_sys_mutex_delete**(LWOW_CFG_OS_MUTEX_HANDLE *mutex, void *arg)

Delete existing mutex and invalidate mutex variable.

Parameters

- **mutex** – **[in]** Mutex handle to remove and invalidate

- **arg** – **[in]** User argument passed on *lwow_init* function

Returns
1 on success, 0 otherwise

uint8_t **lwow_sys_mutex_wait**(LWOW_CFG_OS_MUTEX_HANDLE *mutex, void *arg)

Wait for a mutex until ready (unlimited time)

Parameters

- **mutex** – **[in]** Mutex handle to wait for
- **arg** – **[in]** User argument passed on *lwow_init* function

Returns
1 on success, 0 otherwise

uint8_t **lwow_sys_mutex_release**(LWOW_CFG_OS_MUTEX_HANDLE *mutex, void *arg)

Release already locked mutex.

Parameters

- **mutex** – **[in]** Mutex handle to release
- **arg** – **[in]** User argument passed on *lwow_init* function

Returns
1 on success, 0 otherwise

5.3.4 Device drivers

List of all supported device drivers

DS18x20 temperature sensor

group **LWOW_DEVICE_DS18x20**

Device driver for DS18x20 temperature sensor.

Note: Functions with _raw suffix do no implement locking mechanism when used with operating system.

Defines

LWOW_DS18X20_ALARM_DISABLE

Disable alarm temperature

LWOW_DS18X20_ALARM_NOCHANGE

Do not modify current alarm settings

LWOW_DS18X20_TEMP_MIN

Minimum temperature

LWOW_DS18X20_TEMP_MAX

Maximal temperature

LWOW_DS18X20_CMD_ALARM_SEARCH

Alarm Search Command

LWOW_DS18X20_CMD_CONVERT_T

Convert T Command

Functions

uint8_t **lwow_ds18x20_start_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Start temperature conversion on specific (or all) devices.

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to start measurement for. Set to `NULL` to start measurement on all devices at the same time

Returns

1 on success, `0` otherwise

uint8_t **lwow_ds18x20_start**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Start temperature conversion on specific (or all) devices.

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to start measurement for. Set to `NULL` to start measurement on all devices at the same time

Returns

1 on success, `0` otherwise

uint8_t **lwow_ds18x20_read_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, float *const temp_out)

Read temperature previously started with *lwow_ds18x20_start*.

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to read data from
- **temp_out** – **[out]** Pointer to output float variable to save temperature

Returns

1 on success, `0` otherwise

uint8_t **lwow_ds18x20_read**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, float *const temp_out)

Read temperature previously started with *lwow_ds18x20_start*.

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to read data from
- **temp_out** – **[out]** Pointer to output float variable to save temperature

Returns

1 on success, 0 otherwise

uint8_t **lwow_ds18x20_set_resolution_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, const uint8_t bits)

Set resolution for DS18B20 sensor.

Note: DS18S20 has fixed 9-bit resolution

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to set resolution
- **bits** – **[in]** Number of resolution bits. Possible values are 9 - 12

Returns

1 on success, 0 otherwise

uint8_t **lwow_ds18x20_set_resolution**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, const uint8_t bits)

Set resolution for DS18B20 sensor.

Note: DS18S20 has fixed 9-bit resolution

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to set resolution
- **bits** – **[in]** Number of resolution bits. Possible values are 9 - 12

Returns

1 on success, 0 otherwise

uint8_t **lwow_ds18x20_get_resolution_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Get resolution for `DS18B20` device.

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to get resolution from

Returns

Resolution in units of bits (`9 - 12`) on success, `0` otherwise

uint8_t **lwow_ds18x20_get_resolution**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Get resolution for `DS18B20` device.

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to get resolution from

Returns

Resolution in units of bits (`9 - 12`) on success, `0` otherwise

uint8_t **lwow_ds18x20_set_alarm_temp_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, int8_t temp_l, int8_t temp_h)

Set/clear temperature alarm high/low levels in units of degree Celcius.

Example usage would look something similar to:

```
//Set alarm temperature; low = 10°C, high = 30°C
lwow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
//Set alarm temperature; low = disable, high = no change
lwow_ds18x20_set_alarm_temp(&ow, dev_id, LWOW_DS18X20_ALARM_DISABLE, LWOW_
↪DS18X20_ALARM_NOCHANGE);
//Set alarm temperature; low = no change, high = disable
lwow_ds18x20_set_alarm_temp(&ow, dev_id, LWOW_DS18X20_ALARM_NOCHANGE, LWOW_
↪DS18X20_ALARM_DISABLE);
//Set alarm temperature; low = 10°C, high = 30°C
lwow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
```

Note: `temp_h` and `temp_l` are high and low temperature alarms and can accept different values:

- `-55 % 125`, valid temperature range
- *LWOW_DS18X20_ALARM_DISABLE* to disable temperature alarm (either high or low)
- *LWOW_DS18X20_ALARM_NOCHANGE* to keep current alarm temperature (either high or low)

Parameters

- **ow** – **[in]** 1-Wire handle

- **rom_id** – **[in]** 1-Wire device address
- **temp_l** – **[in]** Alarm low temperature
- **temp_h** – **[in]** Alarm high temperature

Returns
1 on success, `0` otherwise

uint8_t **lwow_ds18x20_set_alarm_temp**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, int8_t temp_l, int8_t temp_h)

Set/clear temperature alarm high/low levels in units of degree Celcius.

Example usage would look something similar to:

```
//Set alarm temperature; low = 10°C, high = 30°C
lwow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
//Set alarm temperature; low = disable, high = no change
lwow_ds18x20_set_alarm_temp(&ow, dev_id, LWOW_DS18X20_ALARM_DISABLE, LWOW_
↪DS18X20_ALARM_NOCHANGE);
//Set alarm temperature; low = no change, high = disable
lwow_ds18x20_set_alarm_temp(&ow, dev_id, LWOW_DS18X20_ALARM_NOCHANGE, LWOW_
↪DS18X20_ALARM_DISABLE);
//Set alarm temperature; low = 10°C, high = 30°C
lwow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
```

Note: `temp_h` and `temp_l` are high and low temperature alarms and can accept different values:

- `-55 % 125`, valid temperature range
- *LWOW_DS18X20_ALARM_DISABLE* to disable temperature alarm (either high or low)
- *LWOW_DS18X20_ALARM_NOCHANGE* to keep current alarm temperature (either high or low)

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address
- **temp_l** – **[in]** Alarm low temperature
- **temp_h** – **[in]** Alarm high temperature

Returns
1 on success, `0` otherwise

uint8_t **lwow_ds18x20_get_alarm_temp_raw**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, int8_t *temp_l, int8_t *temp_h)

Get the low and high temperature triggers for the alarm configuration.

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address
- **temp_l** – **[out]** Pointer to output variable to write low temperature alarm trigger
- **temp_h** – **[out]** Pointer to output variable to write high temperature alarm trigger

Returns
1 on success, 0 otherwise

uint8_t **lwow_ds18x20_get_alarm_temp**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id, int8_t *temp_l, int8_t *temp_h)

Get the low and high temperature triggers for the alarm configuration.

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address
- **temp_l** – **[out]** Pointer to output variable to write low temperature alarm trigger
- **temp_h** – **[out]** Pointer to output variable to write high temperature alarm trigger

Returns
1 on success, 0 otherwise

lwowr_t **lwow_ds18x20_search_alarm_raw**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id)

Search for DS18x20 devices with alarm flag.

Note: To reset search, use *lwow_search_reset* function

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[out]** Pointer to 8-byte long variable to save ROM

Returns
lwowOK on success, member of *lwowr_t* otherwise

lwowr_t **lwow_ds18x20_search_alarm**(*lwow_t* *const owobj, *lwow_rom_t* *const rom_id)

Search for DS18x20 devices with alarm flag.

Note: To reset search, use *lwow_search_reset* function

Note: This function is thread-safe

Parameters

- **ow** – **[in]** 1-Wire handle

- **rom_id** – **[out]** Pointer to 8-byte long variable to save ROM

Returns
lwowOK on success, member of *lwowr_t* otherwise

uint8_t **lwow_ds18x20_is_b**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Check if ROM address matches `DS18B20` device.

Note: This function is reentrant

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to test against `DS18B20`

Returns
1 on success, `0` otherwise

uint8_t **lwow_ds18x20_is_s**(*lwow_t* *const owobj, const *lwow_rom_t* *const rom_id)

Check if ROM address matches `DS18S20` device.

Note: This function is reentrant

Parameters

- **ow** – **[in]** 1-Wire handle
- **rom_id** – **[in]** 1-Wire device address to test against `DS18S20`

Returns
1 on success, `0` otherwise

uint16_t **lwow_ds18x20_get_temp_conversion_time**(uint8_t resolution, uint8_t is_b)

Get temperature conversion time in units of milliseconds for a specific resolution.

Parameters

- **resolution** – Resolution in bits
- **is_b** – **[in]** Set to 1 for DS18B20, `0` otherwise

Returns
uint16_t

5.4 Examples and demos

Various examples are provided for fast library evaluation on embedded systems. These are prepared and maintained for 2 platforms, but could be easily extended to more platforms:

- WIN32 examples, prepared as Visual Studio Community projects
- ARM Cortex-M examples for STM32, prepared as STM32CubeIDE GCC projects

Warning: Library is platform independent and can be used on any platform.

5.4.1 Example architectures

There are many platforms available today on a market, however supporting them all would be tough task for single person. Therefore it has been decided to support (for purpose of examples) 2 platforms only, *WIN32* and *STM32*.

WIN32

Examples for *WIN32* are prepared as Visual Studio Community projects. You can directly open project in the IDE, compile & debug.

To run examples on this architecture, external *USB to UART* converted would be necessary. Application opens *COM port* and sends/receives data directly to there.

Tip: Push-pull to open-drain external converter might be necessary. Check *Hardware UART connection with sensor* for more information.

STM32

Embedded market is supported by many vendors and STMicroelectronics is, with their STM32 series of microcontrollers, one of the most important players. There are numerous amount of examples and topics related to this architecture.

Examples for *STM32* are natively supported with STM32CubeIDE, an official development IDE from STMicroelectronics.

You can run examples on one of official development boards, available in repository examples.

Table 1: Supported development boards

Board name	Onewire settings			Debug settings		
	UART	MTX	MRX	UART	MDTX	MDRX
STM32L496G-Discovery	USART1	PB6	PG10	USART2	PA2	PD6
STM32F429ZI-Nucleo	USART1	PA9	PA10	USART3	PD8	PD9

Pins to connect to 1-Wire sensor:

- *MTX*: MCU TX pin, connected to 1-Wire network data pin (together with MCU RX pin)
- *MRX*: MCU RX pin, connected to 1-Wire network data pin (together with MCU TX pin)
  - *TX* pin is configured as open-drain and can be safely connected directly with *RX* pin

Other pins are for your information and are used for debugging purposes on board.

- MDTX: MCU Debug TX pin, connected via on-board ST-Link to PC
- MDRX: MCU Debug RX pin, connected via on-board ST-Link to PC
- Baudrate is always set to `921600` bauds

5.4.2 Examples list

Here is a list of all examples coming with this library.

Tip: Examples are located in `/examples/` folder in downloaded package. Check *Download library* section to get your package.

LwOW bare-metal

Simple example, not using operating system, showing basic configuration of the library. It can be also called *bare-metal* implementation for simple applications.

LwOW OS

LwOW library as an example when multiple threads want to access to single LwOW core.

5.5 Changelog

```
# Changelog

## Develop

- Move system functions to `system/lwow_sys.h` file
- Add support for Platform.IO
- Split CMakeLists.txt files between library and executable
- Change license year to 2022
- Update code style with astyle
- Add demo driver for manual GPIO toggle
- Add `.clang-format` draft
- Add `lwow_match_or_skip_rom`
- Add `lwow_ds18x20_get_alarm_temp` and `lwow_ds18x20_get_temp_conversion_time`
- Remove deprecated functions, prepare for version `4.0.0`

## v3.0.2

- Update CMSIS OS driver to support FreeRTOS aware kernel

## v3.0.1

- Change configuration options from _CONFIG_ to _OPT
- Apply code style settings with Artistic style options
- Update docs

## v3.0.0

- Break compatibility vs `2.0.0`
- New name of library is now LwOW - Lightweight onewire
- Added `_ex` functions to read/write bytes or bits
```

(continues on next page)

(continued from previous page)

```
- Added drivers features as set of callback functions with drv pointer
- Rename private functions to `prv_` prefix

## v2.0.0

- LL drivers are now passed as custom structure to allow multiple blocks as separate␣
↪drivers
- Added first sphinx documentation
- Updated examples

## v1.2.0

- Automatically set result to OK when search function finds at least 1 connected device
- Upgrade examples to CMSIS OS V2
- Use pre-increment/decrement
- Other C code style fixes

## v1.1.0

- Added assert for all API functions
- Added new owPARERR enumerator for wrong parameters
- Added `ow_deinit` function implementation

## v1.0.0

- Use separate ROM structure instead of byte array
- Add option to search devices with command and save result directly to input array
- Make all variables as local instead of using user pointers

## v0.2.0

- Removed `protect` parameter in API functions
- Added `_raw` functions to provide non-thread-safe implementation for operating systems
- Separate API functions with operating system protection

## v0.1.0

- Support for UART interface instead of software emulation on microcontroller or PC␣
↪application
- Support for operating systems (including non-real-time)
- Added API driver for *DS18x20* temperature sensor
```

5.6 Authors

List of authors and contributors to the library

```
Tilen Majerle <tilen.majerle@gmail.com>
Tilen Majerle <tilen@majerle.eu>
Holden <holden@zenithaerotech.com>
```

INDEX

L

LWOW_ARRAYSIZE (*C macro*), 35
LWOW_ASSERT (*C macro*), 35
LWOW_ASSERT0 (*C macro*), 35
LWOW_CFG_OS (*C macro*), 46
LWOW_CFG_OS_MUTEX_HANDLE (*C macro*), 46
LWOW_CMD_CPYSCRATCHPAD (*C macro*), 35
LWOW_CMD_MATCHROM (*C macro*), 36
LWOW_CMD_READROM (*C macro*), 36
LWOW_CMD_RECEEPROM (*C macro*), 36
LWOW_CMD_RPWRSUPPLY (*C macro*), 36
LWOW_CMD_RSCRATCHPAD (*C macro*), 35
LWOW_CMD_SEARCHROM (*C macro*), 36
LWOW_CMD_SKIPROM (*C macro*), 36
LWOW_CMD_WSCRATCHPAD (*C macro*), 35
lwow_crc (*C++ function*), 45
lwow_deinit (*C++ function*), 37
LWOW_DS18X20_ALARM_DISABLE (*C macro*), 49
LWOW_DS18X20_ALARM_NOCHANGE (*C macro*), 49
LWOW_DS18X20_CMD_ALARM_SEARCH (*C macro*), 50
LWOW_DS18X20_CMD_CONVERT_T (*C macro*), 50
lwow_ds18x20_get_alarm_temp (*C++ function*), 54
lwow_ds18x20_get_alarm_temp_raw (*C++ function*), 53
lwow_ds18x20_get_resolution (*C++ function*), 52
lwow_ds18x20_get_resolution_raw (*C++ function*), 51
lwow_ds18x20_get_temp_conversion_time (*C++ function*), 55
lwow_ds18x20_is_b (*C++ function*), 55
lwow_ds18x20_is_s (*C++ function*), 55
lwow_ds18x20_read (*C++ function*), 50
lwow_ds18x20_read_raw (*C++ function*), 50
lwow_ds18x20_search_alarm (*C++ function*), 54
lwow_ds18x20_search_alarm_raw (*C++ function*), 54
lwow_ds18x20_set_alarm_temp (*C++ function*), 53
lwow_ds18x20_set_alarm_temp_raw (*C++ function*), 52
lwow_ds18x20_set_resolution (*C++ function*), 51
lwow_ds18x20_set_resolution_raw (*C++ function*), 51
lwow_ds18x20_start (*C++ function*), 50
lwow_ds18x20_start_raw (*C++ function*), 50
LWOW_DS18X20_TEMP_MAX (*C macro*), 49
LWOW_DS18X20_TEMP_MIN (*C macro*), 49
lwow_init (*C++ function*), 37
lwow_ll_drv_t (*C++ struct*), 47
lwow_ll_drv_t::deinit (*C++ member*), 47
lwow_ll_drv_t::init (*C++ member*), 47
lwow_ll_drv_t::set_baudrate (*C++ member*), 47
lwow_ll_drv_t::tx_rx (*C++ member*), 48
lwow_match_or_skip_rom (*C++ function*), 44
lwow_match_or_skip_rom_raw (*C++ function*), 44
lwow_match_rom (*C++ function*), 44
lwow_match_rom_raw (*C++ function*), 44
LWOW_MEMCPY (*C macro*), 47
LWOW_MEMSET (*C macro*), 46
lwow_protect (*C++ function*), 37
lwow_read_bit_ex (*C++ function*), 40
lwow_read_bit_ex_raw (*C++ function*), 39
lwow_read_byte_ex (*C++ function*), 39
lwow_read_byte_ex_raw (*C++ function*), 39
lwow_reset (*C++ function*), 38
lwow_reset_raw (*C++ function*), 38
lwow_rom_t (*C++ struct*), 45
lwow_rom_t::rom (*C++ member*), 45
lwow_search (*C++ function*), 40
lwow_search_cb_fn (*C++ type*), 36
lwow_search_devices (*C++ function*), 43
lwow_search_devices_raw (*C++ function*), 43
lwow_search_devices_with_command (*C++ function*), 43
lwow_search_devices_with_command_raw (*C++ function*), 42
lwow_search_raw (*C++ function*), 40
lwow_search_reset (*C++ function*), 40
lwow_search_reset_raw (*C++ function*), 40
lwow_search_with_callback (*C++ function*), 42
lwow_search_with_command (*C++ function*), 41
lwow_search_with_command_callback (*C++ function*), 41
lwow_search_with_command_raw (*C++ function*), 41
lwow_skip_rom (*C++ function*), 45
lwow_skip_rom_raw (*C++ function*), 45

lwow_sys_mutex_create (*C++ function*), 48
lwow_sys_mutex_delete (*C++ function*), 48
lwow_sys_mutex_release (*C++ function*), 49
lwow_sys_mutex_wait (*C++ function*), 49
lwow_t (*C++ struct*), 45
lwow_t::arg (*C++ member*), 46
lwow_t::disrepancy (*C++ member*), 46
lwow_t::ll_drv (*C++ member*), 46
lwow_t::mutex (*C++ member*), 46
lwow_t::rom (*C++ member*), 46
lwow_unprotect (*C++ function*), 38
LWOW_UNUSED (*C macro*), 35
lwow_write_byte_ex (*C++ function*), 39
lwow_write_byte_ex_raw (*C++ function*), 38
lwowr_t (*C++ enum*), 37
lwowr_t::lwowERR (*C++ enumerator*), 37
lwowr_t::lwowERRBAUD (*C++ enumerator*), 37
lwowr_t::lwowERRNODEV (*C++ enumerator*), 37
lwowr_t::lwowERRPAR (*C++ enumerator*), 37
lwowr_t::lwowERRPRESENCE (*C++ enumerator*), 37
lwowr_t::lwowERRTXRX (*C++ enumerator*), 37
lwowr_t::lwowOK (*C++ enumerator*), 37