OneWire UART

Tilen MAJERLE

Feb 16, 2020

CONTENTS

Welcome to the documentation for version latest-develop.

OneWire-UART is lightweight, platform independent library for Onewire protocol for embedded systems.

Download library · *Getting started* · Open Github

CHAPTER

ONE

FEATURES

- Written in ANSI C99
- Platform independent, uses custom low-level layer for device drivers
- 1-Wire protocol fits UART specifications at `9600` and `115200` bauds
- Hardware is responsible for timing characteristics
  - Allows DMA on the high-performance microcontrollers
- Different device drivers included
  - DS18x20 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) 2020 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

5.1.1 Download library

Library is primarly hosted on Github.

- Download latest release from releases area on Github
- Clone *develop* branch for latest development

Download from releases

All releases are available on Github releases area.

Clone from Github

First-time clone

- Download and install `git` if not already
- Open console and navigate to path in the system to clone repository to. Use command `cd your_path`
- Clone repository with one of available `3` options
  - Run `git clone --recurse-submodules https://github.com/MaJerle/onewire-uart` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/onewire-uart` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch master https://github.com/MaJerle/onewire-uart` 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 resources repository is. Use command `cd your_path`
- Run `git pull origin master --recurse-submodules` command to pull latest changes and to fetch latest changes from submodules
- Run `git submodule foreach git pull origin master` to update & merge all submodules

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.

- Copy `onewire_uart` folder to your project
- Add `onewire_uart/src/include` folder to *include path* of your toolchain
- Add source files from `onewire_uart/src/` folder to toolchain build
- Copy `onewire_uart/src/include/ow/ow_config_template.h` to project folder and rename it to `ow_config.h`
- Implement device drivers for UART hardware
- Build the project

5.1.3 Configuration file

Library comes with template config file, which can be modified according to needs. This file shall be named `ow_config.h` and its default template looks like the one below:

Tip: Check *OW Configuration* section for possible configuration settings

Listing 1: Config template file

```
/**
 * \file            ow_config_template.h
 * \brief           OneWire configuration file
 */

/*
 * Copyright (c) 2020 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,
```

(continues on next page)

(continued from previous page)

```
 * 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 OneWire-UART library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */
#ifndef OW_HDR_CONFIG_H
#define OW_HDR_CONFIG_H

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

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

/* After user configuration, call default config to merge config together */
#include "ow/ow_config_default.h"

#endif /* OW_HDR_CONFIG_H */
```

5.2 User manual

5.2.1 How it works

OneWire-UART 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.

Tip: To enable thread-safety support, parameter `OW_CFG_OS` must be set to `1`. Please check *OW 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 `onewire_uart/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            ow_sys_cmsis_os.c
 * \brief           System functions for CMSIS-OS based operating system
 */

/*
 * Copyright (c) 2020 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 OneWire-UART library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */
```

(continues on next page)

(continued from previous page)

```c
#include "ow/ow.h"

#if OW_CFG_OS && !__DOXYGEN__

#include "cmsis_os.h"

uint8_t
ow_sys_mutex_create(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    const osMutexAttr_t attr = {
        .attr_bits = osMutexRecursive
    };

    *mutex = osMutexNew(&attr);                 /* Create new mutex */
    OW_UNUSED(arg);
    return 1;
}

uint8_t
ow_sys_mutex_delete(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    OW_UNUSED(arg);
    osMutexDelete(*mutex);                      /* Delete mutex */
    return 1;
}

uint8_t
ow_sys_mutex_wait(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    if (osMutexAcquire(*mutex, osWaitForever) != osOK) {
        return 0;
    }
    OW_UNUSED(arg);
    return 1;
}

uint8_t
ow_sys_mutex_release(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    if (osMutexRelease(*mutex) != osOK) {
        return 0;
    }
    OW_UNUSED(arg);
    return 1;
}

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

5.2.3 Hardware 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 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.

5.2.5 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 `ow_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 *OW_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 `ow_sys_` name.

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

- *ow_sys_mutex_create()* function to create new mutex
- *ow_sys_mutex_delete()* function to delete existing mutex
- *ow_sys_mutex_wait()* function to wait for mutex to be available
- *ow_sys_mutex_release()* function to release (give) mutex back

Warning: Application must define *OW_CFG_OS_MUTEX_HANDLE* for mutex type. This shall be done in `ow_config.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

```
/**
 * \file            ow_ll_win32.c
 * \brief           UART implementation for WIN32
 */

/*
 * Copyright (c) 2020 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
```

(continues on next page)

(continued from previous page)

```
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * This file is part of OneWire-UART library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */
#include "ow/ow.h"
#include "windows.h"
#include <stdio.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 */
const ow_ll_drv_t
ow_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);

    /* Open virtual file as read/write */
    com_port = CreateFile(L"\\\\.\\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)) {
```

(continues on next page)

(continued from previous page)

```
            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 COM port info\r\n");
    }

    return 1;
}

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

    return 1;
}

uint8_t
set_baudrate(uint32_t baud, void* 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;

    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
         */
```

(continues on next page)

(continued from previous page)

```
        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;
                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            ow_ll_stm32.c
 * \brief           Generic UART implementation for STM32 MCUs
 */

/*
 * Copyright (c) 2020 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 OneWire-UART library.
```

(continues on next page)

(continued from previous page)

```
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */

/*
 * How it works
 *
 * https://docs.majerle.eu/projects/onewire-uart/en/latest/user-manual/hw_connection.html#
 */
#include "ow/ow.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 ow_ll_drv_t
ow_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;
```

(continues on next page)

(continued from previous page)

```
    LL_GPIO_Init(ONEWIRE_RX_PORT, &gpio_init);

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

    OW_UNUSED(arg);

    return 1;
}

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

#endif /* !__DOXYGEN__ */
```

Example: System functions for WIN32

Listing 5: Actual implementation of system functions for WIN32

```c
/**
 * \file            ow_sys_win32.c
 * \brief           System functions for WIN32
 */

/*
 * Copyright (c) 2020 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 OneWire-UART library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */
#include "ow/ow.h"
#include "windows.h"

#if OW_CFG_OS && !__DOXYGEN__

uint8_t
ow_sys_mutex_create(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    *mutex = CreateMutex(NULL, 0, NULL);
    return 1;
}

uint8_t
ow_sys_mutex_delete(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    CloseHandle(*mutex);
    *mutex = NULL;
    return 1;
}

uint8_t
ow_sys_mutex_wait(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
```

(continues on next page)

(continued from previous page)

```
    return WaitForSingleObject(*mutex, INFINITE) == WAIT_OBJECT_0;
}

uint8_t
ow_sys_mutex_release(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    return ReleaseMutex(*mutex);
}

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

Example: System functions for CMSIS-OS

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

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

/*
 * Copyright (c) 2020 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 OneWire-UART library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v2.0.0
 */
#include "ow/ow.h"

#if OW_CFG_OS && !__DOXYGEN__

#include "cmsis_os.h"

uint8_t
ow_sys_mutex_create(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
```

(continues on next page)

(continued from previous page)

```
    const osMutexAttr_t attr = {
        .attr_bits = osMutexRecursive
    };

    *mutex = osMutexNew(&attr);                 /* Create new mutex */
    OW_UNUSED(arg);
    return 1;
}

uint8_t
ow_sys_mutex_delete(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    OW_UNUSED(arg);
    osMutexDelete(*mutex);                      /* Delete mutex */
    return 1;
}

uint8_t
ow_sys_mutex_wait(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    if (osMutexAcquire(*mutex, osWaitForever) != osOK) {
        return 0;
    }
    OW_UNUSED(arg);
    return 1;
}

uint8_t
ow_sys_mutex_release(OW_CFG_OS_MUTEX_HANDLE* mutex, void* arg) {
    if (osMutexRelease(*mutex) != osOK) {
        return 0;
    }
    OW_UNUSED(arg);
    return 1;
}

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

5.3 API reference

List of all the modules:

5.3.1 OneWire-UART

group **OW**

OneWire API.

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

Defines

OW_UNUSED(x)
Unused variable macro

OW_ASSERT(msg, c)
Assert check function.

OW_ASSERT0(msg, c)
Assert check function with return `0`

OW_ARRAYSIZE(x)
Get size of statically declared array.

Return Number of array elements

Parameters

- `[in] x`: Input array

OW_CMD_RSCRATCHPAD
Read scratchpad command for 1-Wire devices

OW_CMD_WSCRATCHPAD
Write scratchpad command for 1-Wire devices

OW_CMD_CPYSCRATCHPAD
Copy scratchpad command for 1-Wire devices

OW_CMD_RECEEPROM
Read EEPROM command

OW_CMD_RPWRSUPPLY
Read power supply command

OW_CMD_SEARCHROM
Search ROM command

OW_CMD_READROM
Read ROM command

OW_CMD_MATCHROM
Match ROM command. Select device with specific ROM

OW_CMD_SKIPROM
Skip ROM, select all devices

Typedefs

typedef owr_t (***ow_search_cb_fn**)(*ow_t* ***const** ow, **const** *ow_rom_t* ***const** rom_id, size_t index, void *arg)
Search callback function implementation.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: Rom address when new device detected. Set to `NULL` when search finished

- `[in] index`: Current device index When `rom_id = NULL`, value indicates number of total devices found
- `[in] arg`: Custom user argument

Enums

enum owr_t
1-Wire result enumeration

Values:

owOK = 0x00
Device returned OK

owERRPRESENCE = -1
Presence was not successful

owERRNODEV = -2
No device connected, maybe device removed during scan?

owPARERR = -3
Parameter error

owERR
General-Purpose error

Functions

owr_t **ow_init** (*ow_t* ***const** *ow*, **const** *ow_ll_drv_t* ***const** *ll_drv*, void **arg*)
Initialize OneWire instance.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[in] ow`: OneWire instance
- `[in] ll_drv`: Low-level driver
- `[in] arg`: Custom argument

void **ow_deinit** (*ow_t* ***const** *ow*)
Deinitialize OneWire instance.

Parameters

- `[in] ow`: OneWire instance

owr_t **ow_protect** (*ow_t* ***const** *ow*, **const** uint8_t *protect*)
Protect 1-wire from concurrent access.

Note Used only for OS systems

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] protect`: Set to `1` to protect core, `0` otherwise

owr_t **ow_unprotect** (*ow_t* ***const** *ow*, **const** uint8_t *protect*)
Unprotect 1-wire from concurrent access.

Note Used only for OS systems

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] protect`: Set to `1` to protect core, `0` otherwise

owr_t **ow_reset_raw** (*ow_t* ***const** *ow*)
Reset 1-Wire bus and set connected devices to idle state.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle

owr_t **ow_reset** (*ow_t* ***const** *ow*)
Reset 1-Wire bus and set connected devices to idle state.

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle

uint8_t **ow_write_byte_raw** (*ow_t* ***const** *ow*, **const** uint8_t *b*)
Write byte over 1-wire protocol.

Return Received byte over 1-wire protocol

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] b`: Byte to write

uint8_t **ow_write_byte** (*ow_t* ***const** *ow*, **const** uint8_t *b*)
Write byte over 1-wire protocol.

Return Received byte over 1-wire protocol

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] b`: Byte to write

uint8_t **ow_read_byte_raw** (*ow_t* ***const** *ow*)
Read next byte on 1-Wire.

Return Byte read over 1-Wire

Parameters

- `[inout] ow`: 1-Wire handle

uint8_t **ow_read_byte**(*ow_t* ***const** *ow*)
Read next byte on 1-Wire.

Return Byte read over 1-Wire

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle

uint8_t **ow_read_bit_raw**(*ow_t* ***const** *ow*)
Read single bit on 1-Wire network.

Return Bit value

Parameters

- `[inout] ow`: 1-Wire handle

uint8_t **ow_read_bit**(*ow_t* ***const** *ow*)
Read single bit on 1-Wire network.

Return Bit value

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle

owr_t **ow_search_reset_raw**(*ow_t* ***const** *ow*)
Reset search.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle

owr_t **ow_search_reset**(*ow_t* ***const** *ow*)
Reset search.

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle

owr_t **ow_search_raw**(*ow_t* ***const** *ow*, *ow_rom_t* ***const** *rom_id*)
Search for devices on 1-wire bus.

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

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle
- `[out] rom_id`: Pointer to ROM structure to save ROM

owr_t **ow_search**(*ow_t* ***const** *ow*, *ow_rom_t* ***const** *rom_id*)

Search for devices on 1-wire bus.

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

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle
- `[out] rom_id`: Pointer to ROM structure to save ROM

owr_t **ow_search_with_command_raw**(*ow_t* ***const** *ow*, **const** uint8_t *cmd*, *ow_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 *ow_search_reset* function

Return *owOK* on success, member of *owr_t* otherwise

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] cmd`: command to use for search operation
- `[out] rom_id`: Pointer to ROM structure to store address

owr_t **ow_search_with_command**(*ow_t* ***const** *ow*, **const** uint8_t *cmd*, *ow_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 *ow_search_reset* function

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

- `[inout] ow`: 1-Wire handle
- `[in] cmd`: command to use for search operation
- `[out] rom_id`: Pointer to ROM structure to store address

owr_t **ow_search_with_command_callback**(*ow_t* ***const** *ow*, **const** uint8_t *cmd*, size_t ***const** *roms_found*, **const** *ow_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

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

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

owr_t **ow_search_with_callback**(*ow_t* ***const** *ow*, size_t ***const** *roms_found*, **const** *ow_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

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

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

owr_t **ow_search_devices_with_command_raw**(*ow_t* ***const** *ow*, **const** uint8_t *cmd*, *ow_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.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

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

owr_t **ow_search_devices_with_command**(*ow_t* ***const** *ow*, **const** uint8_t *cmd*, *ow_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.

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

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

owr_t **ow_search_devices_raw**(*ow_t* ***const** *ow*, *ow_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.

Return *owOK* on success, member of *owr_t* otherwise

Parameters

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

owr_t **ow_search_devices**(*ow_t* ***const** *ow*, *ow_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.

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

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

uint8_t **ow_match_rom_raw**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)

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

Return 1 on success, 0 otherwise

Parameters

- [in] ow: 1-Wire handle
- [in] rom_id: 1-Wire device address to match device

uint8_t **ow_match_rom**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)

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

Return 1 on success, 0 otherwise

Note This function is thread-safe

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: 1-Wire device address to match device

uint8_t **ow_skip_rom_raw**(*ow_t* ***const** *ow*)
Skip ROM address and select all devices on the network.

Return `1` on success, `0` otherwise

Parameters

- `[in] ow`: 1-Wire handle

uint8_t **ow_skip_rom**(*ow_t* ***const** *ow*)
Skip ROM address and select all devices on the network.

Return `1` on success, `0` otherwise

Note This function is thread-safe

Parameters

- `[in] ow`: 1-Wire handle

uint8_t **ow_crc**(**const** void ***const** *in*, **const** size_t *len*)
Calculate CRC-8 of input data.

Return Calculated CRC

Note This function is reentrant

Parameters

- `[in] in`: Input data
- `[in] len`: Number of bytes

struct ow_rom_t
#include <*ow.h*> ROM structure.

Public Members

uint8_t **rom**[8]
8-bytes ROM address

struct ow_t
#include <*ow.h*> 1-Wire structure

Public Members

ow_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 *ow_ll_drv_t* ***ll_drv**
Low-level functions driver

OW_CFG_OS_MUTEX_HANDLE **mutex**
Mutex handle

5.3.2 OW 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 `ow_config.h` file.

Note: Check *Getting started* to create configuration file.

group **OW_CONFIG**
Configuration for OneWire library.

Defines

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

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

OW_CFG_OS_MUTEX_HANDLE
Mutex handle type.

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

5.3.3 Platform specific

List of all the modules:

Low-level driver

group **OW_LL**

Low-level device dependant functions.

struct ow_ll_drv_t

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

Public Members

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

Initialize low-level driver.

Return `1` on success, `0` otherwise

Parameters

- `[in] arg`: Custom argument passed to *ow_init* function

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

De-initialize low-level driver.

Return `1` on success, `0` otherwise

Parameters

- `[in] arg`: Custom argument passed to *ow_init* function

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

Set UART baudrate.

Return `1` on success, `0` otherwise

Parameters

- `[in] baud`: Baudrate to set in units of bauds, normally `9600` or `115200`
- `[in] arg`: Custom argument passed to *ow_init* function

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

Return `1` on success, `0` otherwise

Parameters

- `[in] tx`: Data to transmit over UART
- `[out] rx`: Array to write received data to
- `[in] len`: Number of bytes to exchange
- `[in] arg`: Custom argument passed to *ow_init* function

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 **OW_SYS**
System functions when used with operating system.

Functions

uint8_t **ow_sys_mutex_create**(OW_CFG_OS_MUTEX_HANDLE *_mutex_, void *_arg_)
Create a new mutex and assign value to handle.

Return `1` on success, `0` otherwise

Parameters

- `[out] mutex`: Output variable to save mutex handle
- `[in] arg`: User argument passed on *ow_init* function

uint8_t **ow_sys_mutex_delete**(OW_CFG_OS_MUTEX_HANDLE *_mutex_, void *_arg_)
Delete existing mutex and invalidate mutex variable.

Return `1` on success, `0` otherwise

Parameters

- `[in] mutex`: Mutex handle to remove and invalidate
- `[in] arg`: User argument passed on *ow_init* function

uint8_t **ow_sys_mutex_wait**(OW_CFG_OS_MUTEX_HANDLE *_mutex_, void *_arg_)
Wait for a mutex until ready (unlimited time)

Return `1` on success, `0` otherwise

Parameters

- `[in] mutex`: Mutex handle to wait for
- `[in] arg`: User argument passed on *ow_init* function

uint8_t **ow_sys_mutex_release**(OW_CFG_OS_MUTEX_HANDLE *_mutex_, void *_arg_)
Release already locked mutex.

Return `1` on success, `0` otherwise

Parameters

- `[in] mutex`: Mutex handle to release
- `[in] arg`: User argument passed on *ow_init* function

5.3.4 Device drivers

List of all supported device drivers

DS18x20 temperature sensor

group **OW_DEVICE_DS18x20**
Device driver for DS18x20 temperature sensor.

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

Defines

OW_DS18X20_ALARM_DISABLE
Disable alarm temperature

OW_DS18X20_ALARM_NOCHANGE
Do not modify current alarm settings

OW_DS18X20_TEMP_MIN
Minimum temperature

OW_DS18X20_TEMP_MAX
Maximal temperature

Functions

uint8_t **ow_ds18x20_start_raw**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)
Start temperature conversion on specific (or all) devices.

Return `1` on success, `0` otherwise

Parameters

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

uint8_t **ow_ds18x20_start**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)
Start temperature conversion on specific (or all) devices.

Return `1` on success, `0` otherwise

Note This function is thread-safe

Parameters

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

uint8_t **ow_ds18x20_read_raw**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*, float ***const** *t*)
Read temperature previously started with *ow_ds18x20_start*.

Return `1` on success, `0` otherwise

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: 1-Wire device address to read data from
- `[out] t`: Pointer to output float variable to save temperature

uint8_t **ow_ds18x20_read**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*, float ***const** *t*)
Read temperature previously started with *ow_ds18x20_start*.

Return `1` on success, `0` otherwise

Note This function is thread-safe

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: 1-Wire device address to read data from
- `[out] t`: Pointer to output float variable to save temperature

uint8_t **ow_ds18x20_set_resolution_raw**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*, **const** uint8_t *bits*)
Set resolution for `DS18B20` sensor.

Note `DS18S20` has fixed `9-bit` resolution

Return `1` on success, `0` otherwise

Parameters

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

uint8_t **ow_ds18x20_set_resolution**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*, **const** uint8_t *bits*)
Set resolution for `DS18B20` sensor.

Note `DS18S20` has fixed `9-bit` resolution

Return `1` on success, `0` otherwise

Note This function is thread-safe

Parameters

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

uint8_t **ow_ds18x20_get_resolution_raw**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)
Get resolution for `DS18B20` device.

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

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: 1-Wire device address to get resolution from

uint8_t **ow_ds18x20_get_resolution**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)

Get resolution for `DS18B20` device.

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

Note This function is thread-safe

Parameters

- `[in] ow`: 1-Wire handle
- `[in] rom_id`: 1-Wire device address to get resolution from

uint8_t **ow_ds18x20_set_alarm_temp_raw**(*ow_t* ***const** *ow*, **const** *ow_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
ow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
//Set alarm temperature; low = disable, high = no change
ow_ds18x20_set_alarm_temp(&ow, dev_id, OW_DS18X20_ALARM_DISABLE, OW_DS18X20_
↪ALARM_NOCHANGE);
//Set alarm temperature; low = no change, high = disable
ow_ds18x20_set_alarm_temp(&ow, dev_id, OW_DS18X20_ALARM_NOCHANGE, OW_DS18X20_
↪ALARM_DISABLE);
//Set alarm temperature; low = 10°C, high = 30°C
ow_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
- *OW_DS18X20_ALARM_DISABLE* to disable temperature alarm (either high or low)
- *OW_DS18X20_ALARM_NOCHANGE* to keep current alarm temperature (either high or low)

Return `1` on success, `0` otherwise

Parameters

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

uint8_t **ow_ds18x20_set_alarm_temp**(*ow_t* ***const** *ow*, **const** *ow_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
ow_ds18x20_set_alarm_temp(&ow, dev_id, 10, 30);
//Set alarm temperature; low = disable, high = no change
ow_ds18x20_set_alarm_temp(&ow, dev_id, OW_DS18X20_ALARM_DISABLE, OW_DS18X20_
↪ALARM_NOCHANGE);
//Set alarm temperature; low = no change, high = disable
ow_ds18x20_set_alarm_temp(&ow, dev_id, OW_DS18X20_ALARM_NOCHANGE, OW_DS18X20_
↪ALARM_DISABLE);
//Set alarm temperature; low = 10°C, high = 30°C
ow_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
- *OW_DS18X20_ALARM_DISABLE* to disable temperature alarm (either high or low)
- *OW_DS18X20_ALARM_NOCHANGE* to keep current alarm temperature (either high or low)

Return `1` on success, `0` otherwise

Note This function is thread-safe

Parameters

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

owr_t **ow_ds18x20_search_alarm_raw**(*ow_t* ***const** *ow*, *ow_rom_t* ***const** *rom_id*)
Search for `DS18x20` devices with alarm flag.

Note To reset search, use *ow_search_reset* function

Return *owOK* on success, member of *owr_t* otherwise

Parameters

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

owr_t **ow_ds18x20_search_alarm**(*ow_t* ***const** *ow*, *ow_rom_t* ***const** *rom_id*)
Search for `DS18x20` devices with alarm flag.

Note To reset search, use *ow_search_reset* function

Return *owOK* on success, member of *owr_t* otherwise

Note This function is thread-safe

Parameters

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

uint8_t **ow_ds18x20_is_b**(*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)
Check if ROM address matches `DS18B20` device.

Return 1 on success, 0 otherwise

Note This function is reentrant

Parameters

- [in] ow: 1-Wire handle
- [in] rom_id: 1-Wire device address to test against DS18B20

uint8_t **ow_ds18x20_is_s** (*ow_t* ***const** *ow*, **const** *ow_rom_t* ***const** *rom_id*)
Check if ROM address matches DS18S20 device.

Return 1 on success, 0 otherwise

Note This function is reentrant

Parameters

- [in] ow: 1-Wire handle
- [in] rom_id: 1-Wire device address to test against DS18S20

5.4 Examples and demos

Various examples are provided for fast library evaluation on embedded systems. These are optimized 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 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.

OW 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

OW OS

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

INDEX

R

S

T