LwPKT

Tilen MAJERLE

Dec 28, 2020

CONTENTS

Welcome to the documentation for version v1.1.0.

LwPKT is a generic packet protocol library optimized for embedded systems.

Download library *Getting started* Open Github Donate

CHAPTER

ONE

FEATURES

- Written in ANSI C99, compatible with `size_t` for size data types
- Platform independent, no architecture specific code
- Uses LwRB library for data read/write operations
- Optimized for embedded systems, allows high optimization for data transfer
- Configurable settings for packet structure and variable data length
- Allows multiple notes in network with *from* and *to* addresses
- Separate optional field for *command* data type
- Variable data length to support theoretically unlimited packet length
- CRC check to handle data transmission errors
- User friendly MIT license

CHAPTER TWO

REQUIREMENTS

- C compiler
- 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/lwpkt` command to clone entire repository, including submodules
  - Run `git clone --recurse-submodules --branch develop https://github.com/MaJerle/lwpkt` to clone *development* branch, including submodules
  - Run `git clone --recurse-submodules --branch master https://github.com/MaJerle/lwpkt` 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 `lwpkt` folder to your project
- Add `lwpkt/src/include` folder to *include path* of your toolchain
- Add `libs/lwrb/src/include` folder to *include path* of your toolchain
- Add source files from `lwpkt/src/` folder to toolchain build
- Add source files from `libs/lwrb/src/` folder to toolchain build
- 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 `lwpkt_opts.h` and its default template looks like the one below.

Note: Default configuration template file location: `lwpkt/src/include/lwpkt/lwpkt_opts_template.h`. File must be renamed to `lwpkt_opts.h` first and then copied to the project directory (or simply renamed in-place) where compiler include paths have access to it by using `#include "lwpkt_opts.h"`.

Tip: Check *Configuration* section for possible configuration settings

Listing 1: Template options file

```
/**
 * \file            lwpkt_opts_template.h
 * \brief           LwPKT configuration file
 */

/*
 * Copyright (c) 2020 Tilen MAJERLE
 *
 * Permission is hereby granted, free of charge, to any person
```

(continues on next page)

(continued from previous page)

```
 * 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 LwPKT - Lightweight packet protocol library.
 *
 * Author:          Tilen MAJERLE <tilen@majerle.eu>
 * Version:         v1.1.0
 */
#ifndef LWPKT_HDR_OPTS_H
#define LWPKT_HDR_OPTS_H

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

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

#endif /* LWPKT_HDR_OPTS_H */
```

5.1.4 Minimal example code

Run below example to test and verify library

Listing 2: Simple demo example to test functionality

```
#include <stdio.h>
#include "lwpkt/lwpkt.h"

/* LwPKT data */
static lwpkt_t pkt;
static lwrb_t pkt_tx_rb, pkt_rx_rb;
static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];

/* Data to read and write */
static const char* data = "Hello World\r\n";

/**
 * \brief           LwPKT example code
```

(continues on next page)

(continued from previous page)

```
 */
void
example_lwpkt(void) {
    lwpktr_t res;
    uint8_t b;

    printf("---\r\nLwPKT default example..\r\n\r\n");

    /*
     * Initialize both ring buffers, for TX and RX operations
     *
     * Initialize LwPKT and link buffers together
     */
    lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
    lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
    lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);

#if LWPKT_CFG_USE_ADDR
    /* Set device address (if feature enabled) */
    lwpkt_set_addr(&pkt, 0x12);
#endif /* LWPKT_CFG_USE_ADDR */

    /*
     * Write packet to the TX ringbuffer,
     * act as device wants to send some data
     */
    res = lwpkt_write(&pkt,
#if LWPKT_CFG_USE_ADDR
        0x11,                           /* End address to whom to send */
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_CMD
        0x85,                           /* Command type */
#endif /* LWPKT_CFG_USE_CMD */
        data, strlen(data));            /* Length of data and actual data */

    /*
     * LwPKT wrote data to pkt_tx_rb ringbuffer
     * Now actually transmit data over your interface
     * (USART for example, ...)
     */

    /*
     * For the purpose of this example, application will
     * fake data transmission by doing reading from TX buffer
     * and writing it to RX buffer
     */
    while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
        lwrb_write(&pkt_rx_rb, &b, 1);
    }

    /*
     * Here we have our data in RX buffer
     * means we received data over network interface
     */

    /* Now read and process packet */
    res = lwpkt_read(&pkt);
```

(continues on next page)

(continued from previous page)

```

    if (res == lwpktVALID) {
        size_t len;

        /* Packet is valid */
        printf("Packet is valid!\r\n");

        /* Print debug messages for packet */
#if LWPKT_CFG_USE_ADDR
        printf("Packet from: 0x%08X\r\n", (unsigned)lwpkt_get_from_addr(&pkt));
        printf("Packet to: 0x%08X\r\n", (unsigned)lwpkt_get_to_addr(&pkt));
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_CMD
        printf("Packet cmd: 0x%02X\r\n", (unsigned)lwpkt_get_cmd(&pkt));
#endif /* LWPKT_CFG_USE_CMD */
        printf("Packet data length: 0x%08X\r\n", (unsigned)lwpkt_get_data_len(&pkt));
        if ((len = lwpkt_get_data_len(&pkt)) > 0) {
            uint8_t* d = lwpkt_get_data(&pkt);
            printf("Packet data: ");
            for (size_t i = 0; i < len; ++i) {
                printf("0x%02X ", (unsigned)d[i]);
            }
            printf("\r\n");
        }

        /* Check who should be dedicated receiver */
#if LWPKT_CFG_USE_ADDR
        if (lwpkt_is_for_me(&pkt)) {
            printf("Packet is for me\r\n");
        } else if (lwpkt_is_broadcast(&pkt)) {
            printf("Packet is broadcast to all devices\r\n");
        } else {
            printf("Packet is for device ID: 0x%08X\r\n", (unsigned)lwpkt_get_to_
↪addr(&pkt));
        }
#endif /* LWPKT_CFG_USE_ADDR */
    } else if (res == lwpktINPROG) {
        printf("Packet is still in progress, did not receive yet all bytes..\r\n");
    } else {
        printf("Packet is not valid!\r\n");
    }
}
```

5.2 User manual

LwPKT protocol library is a simple state-machine parser and raw data generator to allow *2* or more devices in a network to communicate in a structure way.

It is perfectly suitable for communication in embedded systems, suchs as *RS-485*, where multiple devices could be easily connected to one big network.

LwPKT library uses well known and easy implementation of LwRB library for data read and data write. It expects *2* different buffer instances.

Parser is simple state machine that reads and processes every received character from read buffer. When application wants to transmit data, LwPKT library generates raw data and writes them to TX buffer.

Combination of both gives embedded applications freedom to implement communication protocols for TX and RX.

5.2.1 Packet structure

Packet structure consists of several fields, where some are optional and some are mandatory.

Fig. 1: Default packet structure

- `START`: Byte with fixed value to represent start of packet
- `FROM`: Byte(s) from where this packet is coming from. Optional field, can be disabled with *LWPKT_CFG_USE_ADDR*
- `TO`: Byte(s) to where this packet is targeting. Optional field, can be disabled with *LWPKT_CFG_USE_ADDR*
- `CMD`: Byte with optional command field to better align with multiple packets. Optional field, can be disabled with *LWPKT_CFG_USE_CMD*
- `LEN`: Length of *data* part field. This is variable multi-byte length to support data length `>= 256` bytes. Always present
- `DATA`: Optional data field. Number of bytes is as in `LEN` field
- `CRC`: 8-bit CRC of all enabled fields except *START* and *STOP* bytes. Optional field, can be disabled with *LWPKT_CFG_USE_CRC*
- `STOP`: Byte with fixed value to represent stop of packet

Tip: If only `2` devices are communicating and are in the network, considering disabling *LWPKT_CFG_USE_ADDR* to improve data bandwidth and remove unnecessary packet overhead

5.2.2 Data input output

LwPKT library only reads and writes to `2` ringbuffers used for read and write operations. It is up to application to implement how buffers are actually later written for read operation and sent out on the network for write operation.

Warning: LwPKT is platform independant and requires final application to actually take care of data being read/written from/to ringbuffers and transferred further over the network

5.2.3 Variable data length

Some fields implement variable data length feature, to optimize data transfer length. Currently supported fields are:

- `DATA` field is always enabled
- `FROM` and `TO` fields when *LWPKT_CFG_ADDR_EXTENDED* feature is enabled

Variable data length is a feature that uses minimum number of bytes to transfer data. It uses `7 LSB bits` per byte for actual data, and `MSB` bit to indicate if there are more bytes coming after. For example, values between `0x00 - 0x7F` are codified within single byte, while values between `0x80 - 0x3F` require `2` bytes for transfer. To transfer `32-bit` variable, minimum `1-byte` and maximum `5-bytes` are used.

Tip: Data codification is always LSB Byte first.

5.2.4 Event management

LwPKT may operate in event mode, meaning that application receives notifications on different events:

- New packet has been received
- Timeout during packet receive

Timeout function is used when network doesn't transmit all bytes or if data got lost in the middle of transmission. This is to make sure that packet protocol library easily recovers to be able to receive more packets in the future

Warning: To use this feature, application must provide accurate timing in units of milliseconds to be able to properly handle timeout function.

Listing 3: LwPKT example with events

```c
#include <stdio.h>
#include "lwpkt/lwpkt.h"

/* LwPKT data */
static lwpkt_t pkt;
static lwrb_t pkt_tx_rb, pkt_rx_rb;
static uint8_t pkt_tx_rb_data[64], pkt_rx_rb_data[64];

/* Data to read and write */
static const char* data = "Hello World\r\n";

/**
 * \brief           LwPKT application callback
 */
static void
my_lwpkt_evt_fn(lwpkt_t* pkt, lwpkt_evt_type_t type) {
    switch (type) {
        case LWPKT_EVT_PKT: {
            printf("Valid packet received..\r\n");

            /* Packet is valid */
            printf("Packet is valid!\r\n");

            /* Print debug messages for packet */
#if LWPKT_CFG_USE_ADDR
            printf("Packet from: 0x%08X\r\n", (unsigned)lwpkt_get_from_addr(pkt));
            printf("Packet to: 0x%08X\r\n", (unsigned)lwpkt_get_to_addr(pkt));
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_CMD
            printf("Packet cmd: 0x%08X\r\n", (unsigned)lwpkt_get_cmd(pkt));
#endif /* LWPKT_CFG_USE_CMD */
            printf("Packet data length: 0x%08X\r\n", (unsigned)lwpkt_get_data_
↪len(pkt));

            /* Do other thins... */
```

(continues on next page)

(continued from previous page)

```
            break;
        }
        case LWPKT_EVT_TIMEOUT: {
            printf("Timeout detected during read operation..\r\n");
            break;
        }
    }
}

/**
 * \brief           LwPKT example code with event feature
 */
void
example_lwpkt_evt(void) {
    lwpktr_t res;
    uint32_t time;
    uint8_t b;

    printf("---\r\nLwPKT event type..\r\n\r\n");

    /*
     * Initialize both ring buffers, for TX and RX operations
     *
     * Initialize LwPKT and link buffers together
     */
    lwrb_init(&pkt_tx_rb, pkt_tx_rb_data, sizeof(pkt_tx_rb_data));
    lwrb_init(&pkt_rx_rb, pkt_rx_rb_data, sizeof(pkt_rx_rb_data));
    lwpkt_init(&pkt, &pkt_tx_rb, &pkt_rx_rb);

#if LWPKT_CFG_USE_ADDR
    /* Set device address (if feature enabled) */
    lwpkt_set_addr(&pkt, 0x12);
#endif /* LWPKT_CFG_USE_ADDR */

    /*
     * Write packet to the TX ringbuffer,
     * act as device wants to send some data
     */
    res = lwpkt_write(&pkt,
#if LWPKT_CFG_USE_ADDR
        0x11,                       /* End address to whom to send */
#endif /* LWPKT_CFG_USE_ADDR */
#if LWPKT_CFG_USE_CMD
        0x85,                       /* Command type */
#endif /* LWPKT_CFG_USE_CMD */
        data, strlen(data));        /* Length of data and actual data */

    /*
     * LwPKT wrote data to pkt_tx_rb ringbuffer
     * Now actually transmit data over your interface
     * (USART for example, ...)
     */

    /*
     * For the purpose of this example, application will
     * fake data transmission by doing reading from TX buffer
     * and writing it to RX buffer
```

(continues on next page)

(continued from previous page)

```
     */
    while (lwrb_read(&pkt_tx_rb, &b, 1) == 1) {
        lwrb_write(&pkt_rx_rb, &b, 1);
    }

    /*
     * Here we have our data in RX buffer
     * means we received data over network interface
     */

    /* Now call process function instead */
    time = 100; /* Get_current_time_in_milliseconds */
    lwpkt_process(&pkt, time, my_lwpkt_evt_fn);
}
```

5.3 API reference

List of all the modules:

5.3.1 LwPKT

group **LWPKT**

Lightweight packet protocol.

Defines

lwpkt_get_from_addr(*pkt*)

Get address from where packet was sent.

Return Address

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_get_to_addr(*pkt*)

Get address to where packet was sent.

Return Address

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_get_data_len(*pkt*)

Get length of packet.

Return Number of data bytes in packet

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_get_data(*pkt*)
Get pointer to packet data.

Return Pointer to data

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_get_cmd(*pkt*)
Get packet command data field.

Return Command data field

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_is_for_me(*pkt*)
Check if packet `to` field address matches device address.

Return `1` on success, `0` otherwise

Parameters

- `[in] pkt`: LwPKT instance

lwpkt_is_broadcast(*pkt*)
Check if packet was sent to all devices on network.

Return `1` if broadcast, `0` otherwise

Parameters

- `[in] pkt`: LwPKT instance

Typedefs

typedef uint32_t **lwpkt_addr_t**
Device address data type.

typedef void (***lwpkt_evt_fn**)(*lwpkt_t* *pkt, *lwpkt_evt_type_t* type)
LwPKT event function.

Parameters

- `[in] pkt`: LwPKT instance with valid packet
- `[in] type`: Event type

Enums

enum lwpkt_state_t

Packet state enumeration.

Values:

enumerator LWPKT_STATE_START
Packet waits for start byte

enumerator LWPKT_STATE_FROM
Packet waits for "packet from" byte

enumerator LWPKT_STATE_TO
Packet waits for "packet to" byte

enumerator LWPKT_STATE_CMD
Packet waits for "packet cmd" byte

enumerator LWPKT_STATE_LEN
Packet waits for (multiple) data length bytes

enumerator LWPKT_STATE_DATA
Packet waits for actual data bytes

enumerator LWPKT_STATE_CRC
Packet waits for CRC data

enumerator LWPKT_STATE_STOP
Packet waits for stop byte

enum lwpktr_t

Packet result enumeration.

Values:

enumerator lwpktOK
Function returns successfully

enumerator lwpktERR
General error for function status

enumerator lwpktINPROG
Receive is in progress

enumerator lwpktVALID
packet valid and ready to be read as CRC is valid and STOP received

enumerator lwpktERRCRC
CRC integrity error for the packet. Will not wait STOP byte if received

enumerator lwpktERRSTOP
Packet error with STOP byte, wrong character received for STOP

enumerator lwpktWAITDATA
Packet state is in start mode, waiting start byte to start receiving

enum lwpkt_evt_type_t

List of event types.

Values:

enumerator LWPKT_EVT_PKT
Valid packet ready to read

enumerator LWPKT_EVT_TIMEOUT
Timeout on packat, reset event

Functions

lwpktr_t lwpkt_init (lwpkt_t *pkt, LWRB_VOLATILE lwrb_t *tx_rb, LWRB_VOLATILE lwrb_t *r
Initialize packet instance and set device address.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance
- `[in] tx_rb`: TX LwRB instance for data write
- `[in] rx_rb`: RX LwRB instance for data read

lwpktr_t **lwpkt_set_addr** (*lwpkt_t *pkt*, *lwpkt_addr_t addr*)
Set device address for packet instance.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance
- `[in] addr`: New device address

lwpktr_t **lwpkt_read** (*lwpkt_t *pkt*)
Read raw data from RX buffer and prepare packet.

Return *lwpktVALID* when packet valid, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance

lwpktr_t **lwpkt_write** (*lwpkt_t *pkt*, *lwpkt_addr_t to*, uint8_t *cmd*, **const** void **data*, size_t *len*)
Write packet data to TX ringbuffer.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance
- `[in] to`: End device address
- `[in] cmd`: Packet command
- `[in] data`: Pointer to input data. Set to `NULL` if not used
- `[in] len`: Length of input data. Must be set to `0` if `data == NULL`

lwpktr_t **lwpkt_reset** (*lwpkt_t *pkt*)
Reset packet state.

Return *lwpktOK* on success, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance

lwpktr_t **`lwpkt_process`** (*lwpkt_t *pkt*, uint32_t *time*, *lwpkt_evt_fn evt_fn*)
Process packet instance and read new data.

Return *lwpktOK* if processing OK, member of *lwpktr_t* otherwise

Parameters

- `[in] pkt`: Packet instance
- `[in] time`: Current time in units of milliseconds
- `[in] evt_fn`: Event function to be called on events

`struct lwpkt_crc_t`
#include <lwpkt.h> CRC structure for packet.

Public Members

uint8_t **`crc`**
Current CRC value

`struct lwpkt_t`
#include <lwpkt.h> Packet structure.

Public Members

lwpkt_addr_t **`addr`**
Current device address

uint8_t **`data`**[**`LWPKT_CFG_MAX_DATA_LEN`**]
Memory to write received data

`LWRB_VOLATILE lwrb_t * tx_rb`
TX ringbuffer

`LWRB_VOLATILE lwrb_t * rx_rb`
RX ringbuffer

uint32_t **`last_rx_time`**
Last RX time in units of milliseconds

lwpkt_state_t **`state`**
Actual packet state machine

lwpkt_crc_t **`crc`**
Packet CRC byte

lwpkt_addr_t **`from`**
Device address packet is coming from

lwpkt_addr_t **`to`**
Device address packet is intended for

uint8_t **`cmd`**
Command packet

size_t **`len`**
Number of bytes to receive

size_t **`index`**
General index variable for multi-byte parts of packet

`struct` *lwpkt_t*::**[anonymous]** **`m`**
Module that is periodically reset for next packet

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 `lwpkt_opts.h` file.

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

group **`LWPKT_OPT`**
Default configuration setup.

Defines

`LWPKT_CFG_MAX_DATA_LEN`
Maximum length of `data` part of the packet in units of bytes.

`LWPKT_CFG_ADDR_BROADCAST`
Address identifying broadcast message to all devices.

`LWPKT_CFG_USE_ADDR`
Enables `1` or disables `0` `from` and `to` fields in the protocol.

This features is useful if communication is between 2 devices exclusively, without addressing requirements

`LWPKT_CFG_ADDR_EXTENDED`
Enables `1` or disables `0` extended address length.

When enabled, multi-byte addresses are supported with MSB codification. Maximum address is limited to `32-bits`.

When disabled, simple `8-bit` address is fixed with single byte.

Feature is disabled by default to keep architecture compatibility

`LWPKT_CFG_USE_CMD`
Enables `1` or disables `0` `cmd` field in the protocol.

When disabled, command part is not used

`LWPKT_CFG_USE_CRC`
Enables `1` or disables `0` CRC check in the protocol.

`LWPKT_CFG_PROCESS_INPROG_TIMEOUT`
Defines timeout time before packet is considered as not valid when too long time in data-read mode.

Used with *lwpkt_process* function

INDEX