LoRa Driver Documentation

This document provides detailed information about the LoRa driver implementation for STM32 using the HAL library. The driver facilitates communication with a LoRa module via UART, handling initialization, sending, and receiving data.

Files

File Name	Description
`lora.hpp`	Header file with function declarations and constants for LoRa communication.
`lora.cpp`	Source file with function implementations for LoRa module operations.

Header File: `lora.hpp`

Overview

The `lora.hpp` file defines the interface for the LoRa driver, including constants, variables, and function prototypes for interacting with the LoRa module.

Constants

Constant	Type	Description
`band`	`uint8_t[]`	Sets the LoRa frequency band to 868 MHz (`AT+BAND=868000000\r\n`).
`password`	`uint8_t[]`	Sets the LoRa module password (`AT+CPIN=LoRaSecure1234\r\n`).
`getid`	`uint8_t[]`	Command to retrieve the network ID (`AT+NETWORKID\r\n`).
`getadd`	`uint8_t[]`	Command to retrieve the device address (`AT+ADDRESS\r\n`).

PROF

Variables

Variable	Type	Description
`uart_rx_buffer`	`uint8_t[]`	Buffer for storing received UART data (100 bytes).
`huart2`	`UART_HandleTypeDef`	UART handle for LoRa communication.
`my_id`	`uint8_t`	Stores the device's network ID.
`my_addr`	`uint8_t`	Stores the device's address.
`receiver_id`	`uint8_t`	Stores the receiver's network ID.
`receiver_addr`	`uint8_t`	Stores the receiver's address.

Function Prototypes

Function	Parameters	Return Type	Description
`get_id()`	None	`uint8_t`	Retrieves the network ID, returns ID + 1.
`get_address()`	None	`uint8_t`	Retrieves the device address, returns 1.
`lora_init()`	None	`void`	Initializes the LoRa module.
`lora_send_char()`	`uint8_t data`	`void`	Sends a single character over LoRa.
`lora_sendinit()`	`uint8_t *pdata`	`void`	Sends initialization data over LoRa.
`lora_recive()`	None	`uint8_t`	Receives data from the LoRa module.
`process_message()`	`const char *message`	`uint8_t`	Processes a received LoRa message.
`receive_message()`	None	`uint8_t`	Receives and processes a LoRa message.
`lora_send()`	None	`void`	Placeholder for sending data (not implemented).

Source File: `lora.cpp`

Overview

The `lora.cpp` file contains the implementation of the LoRa driver functions, handling UART communication with the LoRa module using the STM32 HAL library.

Functions

Function	Parameters	Return Type	Description
`lora_init()`	None	`void`	Initializes the LoRa module by setting the band and password, and retrieving receiver ID and address.
`lora_sendinit()`	`uint8_t *pdata`	`void`	Transmits initialization data via UART.
`lora_recive()`	None	`uint8_t`	Receives data by calling `receive_message()`.
`receive_message()`	None	`uint8_t`	Receives UART data, checks for `+RCV=` prefix, and processes the message.
`process_message()`	`const char *message`	`uint8_t`	Parses a received message in the format `+RCV=<Address>,<Length>,<Data>,<RSSI>,<SNR>` and extracts data.

Function	Parameters	Return Type	Description
`lora_send_char()`	`uint8_t data`	`void`	Formats and sends a single character in the format `AT+SEND=<addr>,1,<data>\r\n`.
`get_id()`	None	`uint8_t`	Sends `AT+NETWORKID` command, receives response, and returns the network ID + 1.
`get_address()`	None	`uint8_t`	Sends `AT+ADDRESS` command, receives response, and returns the address set to 1.

Dependencies

- **STM32 HAL Library**: For UART communication (`HAL_UART_Transmit`, `HAL_UART_Receive`).
- **Standard C Libraries**: `<cstring>` for string operations, `<cstdio>` for `sprintf` and `sscanf`.

Known Issues

Issue Description	Location
Backslash in `getid` definition may cause compilation error.	`lora.hpp`
`myid` used in `get_id()` instead of `my_id`, causing undefined variable error.	`lora.cpp`
`lora_sendinit()` uses `sizeof(pdata)`, which measures pointer size, not data length.	`lora.cpp`
`get_address()` ignores `sscanf` result and returns `my_addr = 1`.	`lora.cpp`
`process_message()` uses incorrect `sscanf` format `%[^,]` for `uint8_t` data.	`lora.cpp`
`lora_send_char()` sends entire 50-byte buffer instead of actual string length.	`lora.cpp`

Notes

PROF

- The driver assumes a specific LoRa module command set (e.g., `AT+BAND`, `AT+CPIN`, `AT+SEND`).
- UART communication is blocking with a 100ms timeout.
- The `lora_send()` function is declared but not implemented.

Alternative Documentation Tools

While Doxygen is used here, other tools may offer modern features:

Tool	Description
Sphinx	Supports C++ via Breathe, offers customizable HTML output, ideal for multi-language projects.
ClangDoc	Leverages Clang for accurate C++ parsing, integrates with LLVM-based workflows.
Doc++	Lightweight, simple documentation for C++, with clean HTML output.

Tool	Description
CxxDoc	Minimalistic C++ documentation tool, less feature-rich but easy to use.

Doxygen remains robust for C++ due to its widespread use and IDE support.