microparcel

Vivien Henry

Jan 27, 2022

CONTENTS:

CHAPTER

ONE

MICROPARCEL

Serialize and deserialize structured data.

Designed for use on bare-metal embedded systems (no dynamic allocation, template based for optimisation), And provides implementation in Python. (see [microparcel-python](https://github.com/lukh/microparcel-python))

- Free software: MIT license
- Documentation: https://microparcel.readthedocs.io.

Provide three differents entities.

1.1 Message

The Message is the Payload, transmitted via a serial line (UART, I2C), byte by byte.

The Message object holds a Data buffer (an array of uint8_t), the size is defined by a class template parameter. And provides methods to access a specific part of the message (a specific bitfield in the message): It uses offset, in bits, and a bitmask

```
#include <microparcel/microparcel.h>

int main(){
    // creates a message with a 8 bytes payload
    microparcel::Message<8> msg = microparcel::Message<8>;

    // set the 5th, 6th, 7th bits of the payload (5,6,7 of the first byte) at the value
↪"2"
    msg.set<uint8_t, 5, 3>(2);

    // set the 13th, 14th, 15th bits of the payload (5,6,7 of the second byte) at the␣
↪value "3"
    msg.set<uint8_t, 13, 3>(3);

    // set the 6th, 7th, 8th, 9th bits of the payload; eg:
    // bits 6 and 7 of the first byte, bits 0 and 1 of the second
    // at the value "1"
    msg.set<uint8_t, 6, 4>(1);


    // for bisize higher that 8 (one byte), the offset must be aligned on a byte
```

(continues on next page)

(continued from previous page)

```
    // bitsize is limited to 16; and the rettype should be change to uint16_t
    msg.set<uint16_t, 24, 16>(0xFFAF);


    // getter works in the same way:
    msg.get<uint8_t, 5, 3>();
    msg.get<uint16_t, 24, 16>();

}
```

1.2 Frame

A Frame encapsulate the Message between a StartOfFrame (SOF) and a CheckSum.

The SOF is an arbitrary value (in our case, 0xAA), and the CheckSum is the sum of all bytes, including the SOF, truncated to 8bits.

It allows a lighweight and fast data integrity validation.

1.3 Parser

The Parser takes bytes, and builds up a Message from the data stream.

```
#include <microparcel/microparcel.h>

// a way to get data from a Serial Line (UART ?)
uint8_t getByteFromDataLine();
bool isDataLineEmpty();

int main(){
    // a Parser for Message with a Payload of 6.
    using TParser = microparcel::Parser<6>;

    TParser parser;
    TParser::Message_T msg;
    TParser::Status status;

    // main loop of embedded application
    while(true){
        // continue till the fifo is empty
        while(!isDataLineEmpty()){
            uint8_t byte = getByteFromDataLine();
            status = parser.parse(byte, &msg);
            switch(status){
                // not complete and error could be treated differently...
                // error means mainly that the checksum is not valid; transmission␣
↪failed.
                case TParser::eNotComplete:
                case TParser::eError:
```

(continues on next page)

(continued from previous page)

```
                    break;

                case TParser::eComplete:
                    // msg is complete, handle it
                    // HANDLE_MSG(msg);
                    break;
            }
        }

    }
}
```

The Parser also encodes Message into Frames for sending data

```
#include <microparcel/microparcel.h>

// prototype to send data
void send(uint8_t *data, uint8_t datasize);

int main(){
    // a Parser for Message with a Payload of 6.
    using TParser = microparcel::Parser<6>;

    TParser::Message_T msg;

    // fill the message
    msg.set<uint8_t, 4, 8>(60);
    msg.set<uint8_t, 0, 4>(0xC);
    //...

    // builds the frame, with SOF and checksum
    TParser::Frame_T frame = TParser.encode(msg);

    // send over physical layer of choice
    send((uint8_t*)&inFrame, TFrame::FrameSize);

}
```

CHAPTER

TWO

LIBRARY API

2.1 Class Hierarchy

2.2 File Hierarchy

2.3 Full API

2.3.1 Namespaces

Namespace microparcel

Contents

Classes

2.3.2 Classes and Structs

Template Class Frame

- Defined in file_include_microparcel_microparcel.h

Class Documentation

template<uint8_t **MsgSize**>

class `microparcel::Frame`

Public Members

uint8_t **SOF**

Message<*MsgSize*> **message**

uint8_t **checksum**

Public Static Attributes

static const uint8_t **kSOF** = 0xAA

static const uint8_t **FrameSize** = *MsgSize* + 2

Template Class Message

- Defined in file_include_microparcel_microparcel.h

Class Documentation

template<uint8_t **Size**>

class `microparcel::Message`
a *Message* class that provides API to access specific fields of a data payload

tparam Size the Byte Size of the *Message*

Public Functions

template<typename **T**, uint8_t **Offset**, uint8_t **Bitsize**>
inline *T* **get**()
returns bitfields of Bitsize located at Offset in a uint8_t data chunk Can return field from 1 to 16 bits

Template Parameters

- **T** – the return type
- **Offset** – the offset, in bits (from 0 to 8*Size)
- **Bitsize** – the bitsize of the returned field (defines the mask)

template<typename **T**, uint8_t **Offset**, uint8_t **Bitsize**>
inline void **set**(*T* field)
sets a bitfield of Bitsize located at Offset in a uint8_t data chunk

Template Parameters

- **T** – the field type
- **Offset** – the offset, in bits (from 0 to 8*Size)
- **Bitsize** – the bitsize of the returned field (defines the mask)

Parameters **field** – the data to set

Public Members

uint8_t **data**[*Size*]

Public Static Attributes

static const uint8_t **kSize** = *Size*

Template Class MsgProcessor

- Defined in file_include_microparcel_microparcel.h

Inheritance Relationships

Base Type

- public Router

Class Documentation

template<typename **Implementation**, typename **Router**, typename **MsgType**>

class `microparcel::MsgProcessor` : public *Router*

A hardware abstracted implementation of a MessageProcessor. parse byte from hardware to route the message to the generated microparcel Router. (processXYZ(...)) a send method makes the frame from message and send bytes via pure virtual sendFrame

sendFrame must be implemented, in Implementation. It sends data on the bus in microparcel format. all the processXYZ methods provided by the router must be implemented, in Implementation Implementation can also provide a way to poll data from a stream (eg UART) and call parse, in a run() for example

Usage: class ZeProcessor: public microparcel::MsgProcessor<ZeProcessor, ZeRouter, ZeMessage >{ virtual void sendFrame(uint8_t *buffer, uint8_t buffer_size){ // send data to the Bus, UART, etc... }

void run(){ parse(uart::getchar()); }

}

Public Functions

inline void **send**(const *MsgType* &inMsg)

Send a message generated via a Router::makeXYZ

inline void **parse**(uint8_t inByte)

interface for sending frame's bytes Parse a byte with *microparcel::Parser*, and process it with the given Router

Template Class Parser

- Defined in file_include_microparcel_microparcel.h

Class Documentation

template<uint8_t **MsgSize**>

class `microparcel::Parser`

Public Types

enum **Status**

Values:

enumerator **eComplete**

enumerator **eNotComplete**

enumerator **eError**

using **Message_T** = *Message*<*MsgSize*>

using **Frame_T** = *Frame*<*MsgSize*>

Public Functions

inline **Parser**()

inline *Status* **parse**(uint8_t in_byte, *Message_T* *out_msg)

Public Static Functions

static inline *Frame_T* **encode**(const *Message_T* &in_msg)

Protected Functions

inline bool **isCheckSumValid**()

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

M