EVOMIN transport protocol

Procedure

1. Enable transport layer (low-level), i.e. I2C, SPI

2. Use evoMin_Init() to initialize a passed instance of evoMin_Interface

3. evoMin_RXHandler() needs to be called from within the low-level transport routine (i.e IRQHandler for SPI/I2C/UART) with the received byte (if you want to receive data, otherwise just skip this method)

4. Whenever a new, valid frame is received, the evoMin_Handler_FrameRecvd() callback gets called from evoMIN

5. To send data, one must implement a TX callback method (low-level implementation for the send transport layer, i.e. I2C, SPI) and assign the implemented callback to the interface by using evoMin_SetTXHandler()

If the device is receive-only, you can disable sending by compiling with the EVOMIN_TX_DISABLE statement.

To eventually send a frame, use the evoMin_sendFrame() method

Buffer status management

The buffer of each individual frame can be checked against overflows etc. Therefor, use the EVOMIN_BUF_STATUS_MASK_xxx masks.

CRC8

The checksum CRC8 is only calculated over the payload bytes EXCLUDING the stuff bytes and also EXCLUDING the frame header bytes, but INCLUDING the command and the payload length byte!

0	command byte
1	payload length byte
2..n	payload bytes

Callbacks / Application-dependent handlers

(see evomin.h)

```
/* -- Custom handlers, must be implemented on the application side -- */

/* evoMin_RXHandler must be called through the low-level byte receive
method, i.e. the SPI RX IRQHandler
   whenever a byte is received on the line */
```

Last edit: 27.11.2018 16:19:00

```
void evoMin_RXHandler(struct evoMin_Interface* interface, uint8_t cByte);

/* Application dependent CRC8 calculation, must be implemented! */
uint8_t evoMin_CRC8(uint8_t* bytes, uint32_t bLen);

/* evoMin_Handler_FrameRecvd callback gets called by evoMIN whenever a new,
valid frame has been received
   It contains a pointer to the received frame, including the command, it's
payload length and the payload itself
   in a buffer */
void evoMin_Handler_FrameRecvd(struct evoMin_Frame* frame);
```

You must implement above listed handlers / callcacks.

Use case

Senden von Daten:

```
/* Test sending of a frame */
const uint32_t sendBufferLen = 9;
uint8_t sendBuffer[] = {
      0xAA,
      0xAA,
      0xAA,
      0xBB,
      0xAA,
      0xAA,
      0xAA,
      0xCC,
      0xDD
};
evoMin_sendFrame(&comInterface, EVOMIN_CMD_CHIP, sendBuffer, sendBufferLen);
```

Resultiert in:

```
Send byte: 170 (AA)
Send byte: 170 (AA)
Send byte: 170 (AA)
Send byte: 15 (F)
Send byte: 9 (9)
Send byte: 170 (AA)
Send byte: 170 (AA)
Send byte: 85 (55)
Send byte: 170 (AA)
Send byte: 187 (BB)
Send byte: 170 (AA)
Send byte: 170 (AA)
```

Send byte: 85 (55)
Send byte: 170 (AA)
Send byte: 204 (CC)
Send byte: 221 (DD)
Send byte: 141 (8D)
Send byte: 85 (55)

Jedes Byte verursacht einen Call des `evoMin_comTXImplementation` Callbacks.

Was man hier schön sieht, ist das automatische Einfügen eines Stopfbytes (0x55 = 85) nach zwei aufeinanderfolgenden 0xAA Bytes innerhalb der Payload.

Test software (Python)

You can use the following script to generate test data to be send or received via evoMIN.

```
from random import randint

n = 20

j_data = ''

for i in range(0, n):

  payload_length = 4

  payload = ''
  for r in range(0, payload_length):
    payload += '{0},'.format(randint(0x00, 0xFE))

  payload = payload[:-1]


  msg = '''uint8_t testData{0}[] =
{1}0xAA,0xAA,0xAA,{5},{2},{3},0xCC,0x55{4};'''.format(i,'{', payload_length,
payload, '}', i)

  j_data += 'testData{0},'.format(i)

  # Print the single rows
  print(msg)
```

```
j_data = j_data[:-1]

jagged_array = 'uint8_t* testData[] = {0} {1} {2};'.format('{', j_data, '}')

# Print the jagged array
print(jagged_array)
```