Sercomm

0.4

Generated by Doxygen 1.7.4

Fri Aug 10 2012 13:30:27

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 sercomm Struct Reference

Sercomm configuration.

`#include <sercomm.h>`

Data Fields

- uint8_t cmd_bytes
- uint8_t ts_bytes
- void(* ts)(void *ts)
- uint8_t len_bytes
- uint8_t hash_bytes
- void(* hash)(unsigned char *hashptr, unsigned char *msg, int mlen)
- uint8_t comm_ctrl_bytes
- uint8_t frame_start_bytes
- unsigned char reset_byte
- uint8_t reset_bytes
- void(* reset)(void)
- unsigned char * buffer
- sc_size_t buffer_size
- sc_size_t message_valid_len
- sc_size_t message_max_len
- void * priv
- sc_size_t buffer_len
- sc_size_t message_len
- uint8_t buffer_reset_bytes
- unsigned char frame_start []

3.1.1 Detailed Description

Sercomm configuration.

This struct sets the header configuration of the serial messages, and it is used for internal storage, while parsing messages.

To use tiny use (smaller struct sercomm size) define SERCOMM_USE_TINY_SC!

Example:

```
static unsigned char comm_buffer[COMM_BUFFER_SIZE];
static struct sercomm sc = {
    .frame_start = { 0x00, 0x01, 0x02, 0x03 },
    .frame_start_bytes = 4,
    .cmd_bytes = 1,
    .ts_bytes = 4,
    .ts = add_timestamp,
    .len_bytes = 1,
    .hash_bytes = 32,
    .hash = gen_crc_hash,
    .comm_ctrl_bytes = 1,
    .message_max_len = 8,
    .message_valid_len = 8,
    .reset_byte = 0xFF,
    .reset_bytes = 51,
    .reset = do_reset,
    .buffer = comm_buffer,
    .buffer_size = COMM_BUFFER_SIZE,
};
```

Message format:

```
+----------------+--------------+
|                |              |
| Sercomm header | Message body |
|                |              |
+----------------+--------------+
```

The deatils of the Sercomm header are below.

Sercomm header format:

```
+-------------+---------+-----------+----------------+----------------+------+
|             |         |           |                |                |      |
| Frame start | Command | Timestamp | Message length | Comm. controll | Hash |
|             |         |           |                |                |      |
+-------------+---------+-----------+----------------+----------------+------+
```

- Frame start: Start sequency
- Command: Message type / Command code
- Timestamp: Message timestamp or sequence number (optional)
- Message length: The length of the message without the header
- Communication controll: Additional comm. controll command, i.e., Close connection (optional)

- Hash: Hash or integrity check, i.e., CRC

Before the hash generation, the hash field is be zero!

Minimal Sercomm header format with only the required fields:

3.1.2 Field Documentation

3.1.2.1 unsigned char* sercomm::buffer

Buffer. It should to be an enogh big array. Use an array which could store at least two messages

3.1.2.2 sc_size_t sercomm::buffer_len

Internal usage: The current number of bytes in the buffer

3.1.2.3 uint8_t sercomm::buffer_reset_bytes

Internal usage: The number of the received reset bytes

3.1.2.4 sc_size_t sercomm::buffer_size

The size of the Buffer

3.1.2.5 uint8_t sercomm::cmd_bytes

3.1.2.6 uint8_t sercomm::comm_ctrl_bytes

The length of the communication controll field (number of bytes). Zero to omit.

3.1.2.7 unsigned char sercomm::frame_start[]

Array of the frame start bytes. See the example

3.1.2.8 uint8_t sercomm::frame_start_bytes

The length of the frame start field (number of bytes).

3.1.2.9 void(∗ sercomm::hash)(unsigned char ∗hashptr, unsigned char ∗msg, int mlen)

Hash callback: hashptr points to the beginning of the Hash field, msg and mlen are the message and its length

3.1.2.10 uint8_t sercomm::hash_bytes

The length of the Hash field (number of bytes). Zero to omit.

3.1.2.11 uint8_t sercomm::len_bytes

The length of the Message length field (number of bytes).

3.1.2.12 sc_size_t sercomm::message_len

Internal usge: The message (body) length of the currently parsed message

3.1.2.13 sc_size_t sercomm::message_max_len

For message validation: The maximum length of a message (without the header).

3.1.2.14 sc_size_t sercomm::message_valid_len

For message validition: If all of the messages have he same size, use it instead of message_max_len. To ommit this check: SERCOMM_IGNORE_MSG_VALID_LENGTH

3.1.2.15 void∗ sercomm::priv

Last priv argument of command callback (fn) in struct sercomm_msg

3.1.2.16 void(∗ sercomm::reset)(void)

Reset callback. It will be called if a reset sequence received. It could be use to reset any message processing mechanisms

3.1.2.17 unsigned char sercomm::reset_byte

The reset byte. A sequence of reset_bytes number of it will be call the reset function

3.1.2.18 uint8_t sercomm::reset_bytes

The number of the reset_byte byte. A sequence of reset_bytes number of reset_byte will be call the reset function

3.1.2.19 void(∗ sercomm::ts)(void ∗ts)

Timestamp callback: ts arguments points to the beginning of the Timestamp field.

3.1.2.20 uint8_t sercomm::ts_bytes

The length of the Timestamp field (number of bytes). Zero to omit.

The documentation for this struct was generated from the following file:

- sercomm.h

3.2 sercomm_msg Struct Reference

Sercomm message and command definition.

`#include <sercomm.h>`

Data Fields

- sc_cmd_t cmd
- void(∗ fn)(unsigned char ∗ts, sc_size_t mlen, unsigned char ∗msg, sc_cctrl_t comm_ctrl, void ∗priv)

3.2.1 Detailed Description

Sercomm message and command definition.

Use this struct to define each message commands and their parser functions. If a message successfully received and validated the fn callbacck will be called for further processing.

The last entry of the array should to be {0, NULL}!

Example usage:

```
static struct sercomm_msg sms[] = {
        { MSG_COMMAND_PRESENT,      cmd_present },
        { MSG_COMMAND_ALARM,        cmd_alarm },
        { MSG_COMMAND_CALIBRATE,    cmd_calibrate },
        { MSG_COMMAND_ALARM_CLEAR,  cmd_alarm_clear },
        { MSG_COMMAND_VALIDATE_COMM,cmd_validate_comm },
        { MSG_COMMAND_BEEP,         cmd_do_beep },
        { MSG_COMMAND_ACK1,         cmd_ack1 },
```

```
        { MSG_COMMAND_ACK2,          cmd_ack2 },
        { 0,         NULL }
};
```

3.2.2 Field Documentation

3.2.2.1 sc_cmd_t sercomm_msg::cmd

Message command value

3.2.2.2 void(* sercomm_msg::fn)(unsigned char *ts, sc_size_t mlen, unsigned char *msg, sc_cctrl_t comm_ctrl, void *priv)

Processing callback. The first argument is the beginning of the timestamp field; the second is the message length; the third is the beginning of the message; the fourth is the value of the comm. control field; and the last is the priv field of struct sercomm

The documentation for this struct was generated from the following file:

- sercomm.h

Chapter 4

File Documentation

4.1 sercomm.h File Reference

```
#include <inttypes.h>
#include <stddef.h>
```

Data Structures

- struct sercomm

  Sercomm configuration.
- struct sercomm_msg

  Sercomm message and command definition.

Defines

- #define SERCOMM_IGNORE_MSG_VALID_LENGTH UINT32_MAX

  Ignore message validity check. Use in message_max_len in struct sercomm.
- #define SERCOMM_OMIT_RESET UINT8_MAX

  Omit reset sequency usage. Use in reset_bytes in struct sercomm.
- #define SIZEOF_SC(frame_start_length) (offsetof(struct sercomm, frame_start) + (frame_start_length) ∗ sizeof ((struct sercomm ∗)0)->frame_start[0])

  The size of the struct sercomm with the dynamic frame_start part.

Functions

- sc_size_t sc_make_message (struct sercomm ∗sc, sc_cmd_t cmd, sc_cctrl_t cctrl, unsigned char ∗msg, sc_size_t mlen, unsigned char ∗output, sc_size_t olen)

  Create a message with sercomm header.

- void sc_get_message (struct sercomm *sc, struct sercomm_msg *sm, unsigned char byte)

  Get and parse a message.

4.1.1 Define Documentation

4.1.1.1 #define SERCOMM_IGNORE_MSG_VALID_LENGTH UINT32_MAX

Ignore message validity check. Use in message_max_len in struct sercomm.

4.1.1.2 #define SERCOMM_OMIT_RESET UINT8_MAX

Omit reset sequency usage. Use in reset_bytes in struct sercomm.

4.1.1.3 #define SIZEOF_SC(*frame_start_length*) (offsetof(struct **sercomm**, frame_start) + (frame_start_length) * sizeof ((struct **sercomm** *)0)->frame_start[0])

The size of the struct sercomm with the dynamic frame_start part.

Parameters

frame_start_length	The length of the frame_start array

4.1.2 Function Documentation

4.1.2.1 void sc_get_message (struct **sercomm** * *sc,* struct **sercomm_msg** * *sm,* unsigned char *byte*)

Get and parse a message.

This function gets the message byte to byte. It build the entire message from the received bytes. If the message is valid, it calls the callback of the command.

It searches the beginng of the message. It shoudl to be the frame start sequence. The first validation will be proceeded after the receiving of the message hader. The next, after the receiving of the full message.

Example:

```
static void main_get_message(uint8_t byte)
{
        sc_get_message(&sc, sms, byte);
}
```

Parameters

sc	The main struct sercomm
sm	The struct sercomm_msg array
byte	The received byte

4.1.2.2 sc_size_t sc_make_message (struct sercomm * *sc*, sc_cmd_t *cmd*, sc_cctrl_t *cctrl*, unsigned char * *msg*, sc_size_t *mlen*, unsigned char * *output*, sc_size_t *olen*)

Create a message with sercomm header.

This function cretates a message with proper header configuration. After return the message is ready for sending.

Example usage:

```
uint8_t tmp;
tmp = sc_make_message(&sc, MSG_COMMAND_ALARM, MSG_CCTRL_NONE, message_body, mess
     age_body_len, comm_buffer, COMM_BUFFER_SIZE);
uart_send_message(comm_buffer, tmp);
```

Parameters

sercomm	The main struct sercom
cmd	Message command value
cctrl	The value of the comm. control field
msg	Message body
mlen	The length of the message body
output	The output buffer. The message with the header will be generated here.
olen	The size of the output buffer

Returns

The length of the message with the header, or zero if error occured

Index