SerialProtocol

Generated by Doxygen 1.9.3

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 Buffer Class Reference

Public Member Functions

- Buffer (uint8_t bufLen, uint8_t *buf)
  Buffer constructor.
- void putElem (uint8_t ui8_data)
  Puts one byte into the buffer.
- uint8_t readBuf (uint8_t **pui8_target)
  Buffer read operation.
- void flushBuf (void)
  Empties the buffer.
- bool getNextFreeBufSpace (uint8_t **pui8_target)
  Sets the input pointer to the next free buffer address.
- bool increaseBufIdx (uint8_t ui8_size)
  Increases the buffer index.
- int16_t getActualIdx (void)
  Returns the actual (last written) buffer index.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Buffer()

```
Buffer::Buffer (
            uint8_t bufLen,
            uint8_t * buf )
```

Buffer constructor.

Parameters

size	Size of the externally provided buffer
∗*buf*	Pointer to the first byte of the buffer

3.1.2 Member Function Documentation

3.1.2.1 flushBuf()

```
void Buffer::flushBuf (
            void  )
```

Empties the buffer.

Sets the buffer index to -1 (start value) and the actual buffer Space to the buffer length. The buffer contents hence are "invalidated".

3.1.2.2 getActualIdx()

```
int16_t Buffer::getActualIdx (
            void  )
```

Returns the actual (last written) buffer index.

Returns

actual buffer index.

3.1.2.3 getNextFreeBufSpace()

```
bool Buffer::getNextFreeBufSpace (
            uint8_t ** pui8_target )
```

Sets the input pointer to the next free buffer address.

Parameters

∗∗*pui8_target*	Pointer address.

Returns

True if there was free buffer space available, false otherwise

3.1.2.4 increaseBufIdx()

```
bool Buffer::increaseBufIdx (
            uint8_t ui8_size )
```

Increases the buffer index.

Parameters

ui8_size	counts about which to increase the index.

Returns

True if the operation was successful, false otherwise.

3.1.2.5 putElem()

```
void Buffer::putElem (
            uint8_t ui8_data )
```

Puts one byte into the buffer.

Parameters

data	Data byte

3.1.2.6 readBuf()

```
uint8_t Buffer::readBuf (
            uint8_t ** pui8_target )
```

Buffer read operation.

This routine receives the address of a pointer variable, which gets moved to the start of the buffer.

Parameters

∗∗*pui8_target*	Pointer address.

Returns

Size of the stored data in bytes.

The documentation for this class was generated from the following files:

- Buffer.h
- Buffer.cpp

3.2 COMMAND Struct Reference

Command structure declaration.

`#include <Commands.h>`

Public Attributes

- int16_t i16_num
- float f_val
- COMMAND_TYPE e_cmdType

3.2.1 Detailed Description

Command structure declaration.

3.2.2 Member Data Documentation

3.2.2.1 e_cmdType

```
COMMAND_TYPE COMMAND::e_cmdType
```

Command Type.

3.2.2.2 f_val

```
float COMMAND::f_val
```

Command Value.

3.2.2.3 i16_num

```
int16_t COMMAND::i16_num
```

ID Number.

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

- Commands.h

3.3 RESPONSE Struct Reference

Response structure declaration.

```
#include <Commands.h>
```

Public Attributes

- bool b_valid
- int16_t i16_num
- float f_val
- COMMAND_TYPE e_cmdType

3.3.1 Detailed Description

Response structure declaration.

3.3.2 Member Data Documentation

3.3.2.1 b_valid

```
bool RESPONSE::b_valid
```

Flags if response is valid and can be sent.

3.3.2.2 e_cmdType

```
COMMAND_TYPE RESPONSE::e_cmdType
```

Response type inherited from Command type.

3.3.2.3 f_val

```
float RESPONSE::f_val
```

Response value.

3.3.2.4 i16_num

```
int16_t RESPONSE::i16_num
```

ID Number. (Reflects Command ID number).

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

- Commands.h

3.4 SerialCommands Class Reference

Public Member Functions

- **SerialCommands** (void)
  SerialCommands c'tor.
- RESPONSE executeCmd (COMMAND cmd)
  Executes the incoming command.

Public Attributes

- uint8_t ui8_cmdCBStructLength
- COMMAND_CB * p_cmdCBStruct
- VarAccess varAccess = VarAccess()

3.4.1 Member Function Documentation

3.4.1.1 executeCmd()

```
RESPONSE SerialCommands::executeCmd (
            COMMAND cmd )
```

Executes the incoming command.

Parameters

cmd	Holds the command information from the parsed command.

Returns

Response structure.

3.4.2 Member Data Documentation

3.4.2.1 p_cmdCBStruct

```
COMMAND_CB* SerialCommands::p_cmdCBStruct
```

Command callback structure.

3.4.2.2 ui8_cmdCBStructLength

```
uint8_t SerialCommands::ui8_cmdCBStructLength
```

Remembers the length of the command callback structure.

3.4.2.3 varAccess

```
VarAccess SerialCommands::varAccess = VarAccess()
```

Variable structure access methods.

The documentation for this class was generated from the following files:

- Commands.h
- Commands.cpp

3.5 SerialProtocol Class Reference

Public Member Functions

- SerialProtocol ()

  SerialProtocol c'tor.
- void setupCallbacks (TX_CB transmit_cb, READEEPROM_CB readEEPROM_cb, WRITEEEPROM_CB writeEEPROM_cb)

  Take and save the function pointers to the user-defined callbacks.
- void setupVariableStructure (VAR *p_varStruct, uint8_t ui8_structLen)

  Store the variable structure address and length.
- void setupCommandStructure (COMMAND_CB *p_cmdStruct, uint8_t ui8_structLen)

  Store the command structure address and length.
- void statemachine (void)

  Protocol state machine.
- void receive (uint8_t ui8_data)

  Receive method.

Public Attributes

- struct {
  PROTOCOL_STATE e_state
  bool b_error
  bool b_sent
  DEBUG_ACTIVATION_STATE e_dbgActState
  DBG_FCN_CB debugFcnArray [10] = {nullptr}
  } **control**

- uint8_t txRxBuffer [TXRX_BUFFER_LENGTH] = {0}
- Buffer rxBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer)
- Buffer txBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer)
- TX_CB txCallback = nullptr

3.5.1 Constructor & Destructor Documentation

3.5.1.1 SerialProtocol()

```
SerialProtocol::SerialProtocol ( )
```

SerialProtocol c'tor.

Parameters

transmitCallback	Callback for transmitting data

3.5.2 Member Function Documentation

3.5.2.1 receive()

```
void SerialProtocol::receive (
            uint8_t ui8_data )
```

Receive method.

Parameters

ui8_data	Received data byte to be processed within the proocol.

3.5.2.2 setupCallbacks()

```
void SerialProtocol::setupCallbacks (
            TX_CB transmit_cb,
            READEEPROM_CB readEEPROM_cb,
            WRITEEEPROM_CB writeEEPROM_cb )
```

Take and save the function pointers to the user-defined callbacks.

Parameters

transmit_cb	Transmission callback function pointer.
readEEPROM_cb	EEPROM read callback function pointer.
writeEEPROM_cb	EEPROM write callback function pointer.

3.5.2.3 setupCommandStructure()

```
void SerialProtocol::setupCommandStructure (
            COMMAND_CB * p_cmdStruct,
            uint8_t ui8_structLen )
```

Store the command structure address and length.

Parameters

∗p_cmdStruct	pointer to the command structure.
ui8_structLen	Length of the variable structure.

3.5.2.4 setupVariableStructure()

```
void SerialProtocol::setupVariableStructure (
            VAR * p_varStruct,
            uint8_t ui8_structLen )
```

Store the variable structure address and length.

Parameters

∗p_varStruct	pointer to the variable structure.
ui8_structLen	Length of the variable structure.

3.5.2.5 statemachine()

```
void SerialProtocol::statemachine (
            void  )
```

Protocol state machine.

This function must be called in a cyclic manner for proper operation of the serial protocol.

3.5.3 Member Data Documentation

3.5.3.1 b_error

```
bool SerialProtocol::b_error
```

Error flag.

3.5.3.2 b_sent

```
bool SerialProtocol::b_sent
```

Data sent flag.

3.5.3.3 debugFcnArray

```
DBG_FCN_CB SerialProtocol::debugFcnArray[10] = {nullptr}
```

Function pointer array to the debug command functions.

3.5.3.4 e_dbgActState

```
DEBUG_ACTIVATION_STATE SerialProtocol::e_dbgActState
```

State of the received symbols for debug function activation.

3.5.3.5 e_state

```
PROTOCOL_STATE SerialProtocol::e_state
```

Actual protocol state.

3.5.3.6 rxBuffer

```
Buffer SerialProtocol::rxBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer)
```

RX buffer handler.

3.5.3.7 txBuffer

```
Buffer SerialProtocol::txBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer)
```

TX buffer handler.

3.5.3.8 txCallback

```
TX_CB SerialProtocol::txCallback = nullptr
```

Transmission callback function.

3.5.3.9 txRxBuffer

```
uint8_t SerialProtocol::txRxBuffer[TXRX_BUFFER_LENGTH] = {0}
```

Combined TX/RX buffer.

The documentation for this class was generated from the following files:

- SerialProtocol.h
- SerialProtocol.cpp

3.6 VAR Struct Reference

Variable struct member declaration.

```
#include <Variables.h>
```

Public Attributes

- void * val
- TYPE vartype
- DTYPE datatype
- struct {
  uint16_t ui16_eeAddress
  uint8_t ui8_byteLength
  } **runtime**

3.6.1 Detailed Description

Variable struct member declaration.

3.6.2 Member Data Documentation

3.6.2.1 datatype

```
DTYPE VAR::datatype
```

Datatype of the linked variable.

3.6.2.2 ui16_eeAddress

```
uint16_t VAR::ui16_eeAddress
```

EEPROM address of the variable value.

3.6.2.3 ui8_byteLength

```
uint8_t VAR::ui8_byteLength
```

byte length of the variable depending on data type.

3.6.2.4 val

```
void* VAR::val
```

Pointer to the RAM variable the structure item links to.

3.6.2.5 vartype

```
TYPE VAR::vartype
```

Storage type of the variable (RAM or EEPROM).

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

- Variables.h

3.7 VarAccess Class Reference

Public Member Functions

- **VarAccess** ()

  constructor
- bool initVarstruct ()

  Initializes the variable structure.
- bool readValFromVarStruct (int16_t i16_varNum, float *pf_val)

  Performs a variable read operation through the variable structure.
- bool writeValToVarStruct (int16_t i16_varNum, float f_val)

  Performs a variable write operation through the variable structure.
- bool readEEPROMValueIntoVarStruct (int16_t i16_varNum)

  Reads a value from the EEPROM into the Variable structure.
- bool writeEEPROMwithValueFromVarStruct (int16_t i16_varNum)

  Writes the EEPROM by the value read out from the variable structure.

Public Attributes

- uint8_t ui8_varStructLength
- VAR * p_varStruct
- WRITEEEPROM_CB writeEEPROM_cb = nullptr
- READEEPROM_CB readEEPROM_cb = nullptr

3.7.1 Member Function Documentation

3.7.1.1 initVarstruct()

```
bool VarAccess::initVarstruct ( )
```

Initializes the variable structure.

This function sets up the "partition table" for EEPROM accesses and reads writes the values currently stored in the EEPROM to the variable structure.

Returns

Success indicator.

3.7.1.2 readEEPROMValueIntoVarStruct()

```
bool VarAccess::readEEPROMValueIntoVarStruct (
            int16_t i16_varNum )
```

Reads a value from the EEPROM into the Variable structure.

Parameters

i16_varNum	Variable structure number.

Returns

Success indicator.

3.7.1.3 readValFromVarStruct()

```
bool VarAccess::readValFromVarStruct (
            int16_t i16_varNum,
            float * pf_val )
```

Performs a variable read operation through the variable structure.

Parameters

i16_varNum	Variable number (deduced from ID number) to access.
∗pf_val	Address to the variable to which the value gets written.

Returns

Success indicator.

3.7.1.4 writeEEPROMwithValueFromVarStruct()

```
bool VarAccess::writeEEPROMwithValueFromVarStruct (
            int16_t i16_varNum )
```

Writes the EEPROM by the value read out from the variable structure.

Parameters

i16_varNum	Variable structure number.

Returns

Success indicator.

3.7.1.5 writeValToVarStruct()

```
bool VarAccess::writeValToVarStruct (
            int16_t i16_varNum,
            float f_val )
```

Performs a variable write operation through the variable structure.

Parameters

i16_varNum	Variable number (deduced from ID number) to access.
f_val	Value to write.

Returns

Success indicator.

3.7.2 Member Data Documentation

3.7.2.1 p_varStruct

```
VAR* VarAccess::p_varStruct
```

Remembers the address of the variable structure.

3.7.2.2 readEEPROM_cb

```
READEEPROM_CB VarAccess::readEEPROM_cb = nullptr
```

Gets called in case of a EEPROM variable has been read by command.

3.7.2.3 ui8_varStructLength

```
uint8_t VarAccess::ui8_varStructLength
```

Remembers the length of the variable structure.

3.7.2.4 writeEEPROM_cb

```
WRITEEEPROM_CB VarAccess::writeEEPROM_cb = nullptr
```

Gets called in case of a EEPROM variable has been writen by command.

The documentation for this class was generated from the following files:

- Variables.h
- VarAccess.cpp

Chapter 4

File Documentation

4.1 Buffer.cpp File Reference

Definitions for the Buffer module.

```
#include "Buffer.h"
```

4.1.1 Detailed Description

Definitions for the Buffer module.

Author

Roman Holderried

History

- 2022-01-13 - File creation

4.2 Buffer.h File Reference

Functions for controlling data traffic from and into a memory space.

```
#include <stdint.h>
#include <stdbool.h>
```

Classes

- class Buffer

4.2.1 Detailed Description

Functions for controlling data traffic from and into a memory space.

Author

Roman Holderried

Needs an externally defined buffer space (array), to which the address must be passed to the constructor.

History

- 2022-01-13 - File creation

4.3 Buffer.h

Go to the documentation of this file.

```
1 /*******************************************************************************/
14 #ifndef _BUFFER_H_
15 #define _BUFFER_H_
16
17 /*******************************************************************************
18  * Includes
19  *******************************************************************************/
20
21 #include <stdint.h>
22 #include <stdbool.h>
23
24 /*******************************************************************************
25  * Class declarations
26  *******************************************************************************/
27 class Buffer
28 {
29
30     private:
31
32     uint8_t     *pui8_bufPtr;
33     int16_t     i16_bufIdx;
34     uint8_t     ui8_bufLen;
35     uint8_t     ui8_bufSpace;
36     bool        b_ovfl;
38     public: // methods
39
45     Buffer(uint8_t bufLen, uint8_t *buf);
46
51     void putElem(uint8_t ui8_data);
52
61     uint8_t readBuf (uint8_t **pui8_target);
62
68     void flushBuf (void);
69
75     bool getNextFreeBufSpace(uint8_t **pui8_target);
76
82     bool increaseBufIdx(uint8_t ui8_size);
83
88     int16_t getActualIdx(void);
89 };
90
91
92
93 #endif
```

4.4 Commands.cpp File Reference

Definitions for the SerialCommands module.

```
#include <stdint.h>
#include "Commands.h"
#include "Variables.h"
```

4.4.1 Detailed Description

Definitions for the SerialCommands module.

Author

Roman Holderried

History

- 2022-01-13 - File creation

4.5 Commands.h File Reference

Command evaluation and variable structure access.

```
#include <stdint.h>
#include <stdbool.h>
#include "Variables.h"
#include "CommandStucture.h"
```

Classes

- struct COMMAND

  Command structure declaration.
- struct RESPONSE

  Response structure declaration.
- class SerialCommands

Macros

- #define **COMMAND_DEFAULT** {0, 0.0, eCOMMAND_TYPE_NONE}
- #define **RESPONSE_DEFAULT** {false, 0, 0.0, eCOMMAND_TYPE_NONE}

Enumerations

- enum COMMAND_TYPE { **eCOMMAND_TYPE_NONE** = -1 , **eCOMMAND_TYPE_GETVAR** = 0 , **eCOMMAND_TYPE_SETVAR** = 1 , **eCOMMAND_TYPE_COMMAND** = 2 }

  Command type enumeration.

4.5.1 Detailed Description

Command evaluation and variable structure access.

Author

Roman Holderried

History

- 2022-01-13 - File creation

4.6 Commands.h

Go to the documentation of this file.

```
1 /*****************************************************************************/
10 #ifndef _COMMANDS_H_
11 #define _COMMANDS_H_
12
13 /*****************************************************************************
14  * Includes
15  *****************************************************************************/
16 #include <stdint.h>
17 #include <stdbool.h>
18 #include "Variables.h"
19 #include "CommandStucture.h"
20
21 /*****************************************************************************
22  * Type definitions
23  *****************************************************************************/
24
26 typedef enum
27 {
28     eCOMMAND_TYPE_NONE      =   -1,
29     eCOMMAND_TYPE_GETVAR    =   0,
30     eCOMMAND_TYPE_SETVAR    =   1,
31     eCOMMAND_TYPE_COMMAND   =   2
32 }COMMAND_TYPE;
33
34
36 typedef struct
37 {
38     int16_t         i16_num;
39     float           f_val;
40     COMMAND_TYPE    e_cmdType;
41 }COMMAND;
42
43 #define COMMAND_DEFAULT         {0, 0.0, eCOMMAND_TYPE_NONE}
44
46 typedef struct
47 {
48     bool            b_valid;
49     int16_t         i16_num;
50     float           f_val;
51     COMMAND_TYPE    e_cmdType;
52 }RESPONSE;
53
54 #define RESPONSE_DEFAULT         {false, 0, 0.0, eCOMMAND_TYPE_NONE}
55
56 /*****************************************************************************
57  * Class declarations
58  *****************************************************************************/
59 class SerialCommands
60 {
61     public:
62
63     uint8_t ui8_cmdCBStructLength;
64     COMMAND_CB *p_cmdCBStruct;
66     VarAccess varAccess = VarAccess();
69     SerialCommands(void);
70
76     RESPONSE    executeCmd (COMMAND cmd);
77 };
78
79 #endif //_COMMANDS_H_
```

4.7 CommandStucture.h

```
1 /*****************************************************************************/
11 #ifndef _COMMANDSTRUCTURE_H_
12 #define _COMMANDSTRUCTURE_H_
13
14 /*****************************************************************************
15  * Includes
16  *****************************************************************************/
17 #include <stdint.h>
18 #include <stdbool.h>
19
20 /*****************************************************************************
21  * Type definitions
22  *****************************************************************************/
29 typedef bool(*COMMAND_CB)(float* pf_valArray, uint8_t ui8_valArrayLen);
30
31 #endif // _COMMANDSTRUCTURE_H_
```

4.8 Helpers.cpp File Reference

Definitions of the Helpers modules.

```
#include <stdint.h>
#include "Helpers.h"
```

Functions

- uint8_t ftoa (uint8_t *pui8_resBuf, float val, uint8_t ui8_maxAfterpoint, bool b_round)

  Float to ASCII string conversion.

Variables

- const uint32_t **ui32_pow10** [10] = { 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000 }

4.8.1 Detailed Description

Definitions of the Helpers modules.

Author

Roman Holderried

History

- 2022-01-17 - File creation

4.8.2 Function Documentation

4.8.2.1 ftoa()

```
uint8_t ftoa (
            uint8_t * pui8_resBuf,
            float val,
            uint8_t maxAfterPoint = 5,
            bool b_round = true )
```

Float to ASCII string conversion.

This function features an automatic value range detection, a user defined afterpoint length definition, as well es an optional value rounding functionality. Trailing zeros are detected and ignored.

Parameters

∗pui8_resBuf	Pointer to the buffer which will be holding the result.
val	Float value to be converted.
maxAfterPoint	Maximum digits after the decimal point that will be accounted for. Defaults to 5.
b_round	Value gets rounded according to the after point digits or not.

Returns

Output string size in bytes.

4.9 Helpers.h File Reference

Helper functions that can be generically used.

```
#include <stdint.h>
#include <stdbool.h>
```

Functions

- uint8_t ftoa (uint8_t ∗pui8_resBuf, float val, uint8_t maxAfterPoint=5, bool b_round=true)
  Float to ASCII string conversion.

4.9.1 Detailed Description

Helper functions that can be generically used.

Author

Roman Holderried

History

- 2022-01-14 - File creation

4.9.2 Function Documentation

4.9.2.1 ftoa()

```
uint8_t ftoa (
            uint8_t * pui8_resBuf,
            float val,
            uint8_t maxAfterPoint = 5,
            bool b_round = true )
```

Float to ASCII string conversion.

This function features an automatic value range detection, a user defined afterpoint length definition, as well es an optional value rounding functionality. Trailing zeros are detected and ignored.

Parameters

∗pui8_resBuf	Pointer to the buffer which will be holding the result.
val	Float value to be converted.
maxAfterPoint	Maximum digits after the decimal point that will be accounted for. Defaults to 5.
b_round	Value gets rounded according to the after point digits or not.

Returns

Output string size in bytes.

4.10 Helpers.h

Go to the documentation of this file.

```
1 /*****************************************************************************/
10 #ifndef _HELPERS_H_
11 #define _HELPERS_H_
12
13 /*****************************************************************************
14  * Includes
15  *****************************************************************************/
16 #include <stdint.h>
17 #include <stdbool.h>
18
19 // #define FTOA_MAX_AFTERPOINT 5
20 /*****************************************************************************
21  * Function declarations
22  *****************************************************************************/
23
36 uint8_t ftoa (uint8_t *pui8_resBuf, float val, uint8_t maxAfterPoint = 5, bool b_round = true);
37
38
39
40 #endif // _HELPERS_H_
```

4.11 SerialProtocol.cpp File Reference

Definitions for the SerialProtocol module.

```
#include "SerialProtocol.h"
#include <stdint.h>
#include "Commands.h"
#include <string.h>
#include <stdlib.h>
#include "Helpers.h"
#include "Variables.h"
#include "Buffer.h"
```

4.11.1 Detailed Description

Definitions for the SerialProtocol module.

Author

Roman Holderried

History

- 2022-01-13 - File creation

4.12 SerialProtocol.h File Reference

Request - Response protocol functionality.

```
#include <stdint.h>
#include <stdbool.h>
#include "Commands.h"
#include "Variables.h"
#include "Buffer.h"
#include "CommandStucture.h"
```

Classes

- class SerialProtocol

Macros

- #define **TRANSMIT_BUFFER_LENGTH** 64
- #define **TXRX_BUFFER_LENGTH** TRANSMIT_BUFFER_LENGTH
- #define **STX** 0x02
- #define **ETX** 0x03
- #define **GETVAR_IDENTIFIER** '?'
- #define **SETVAR_IDENTIFIER** '!'
- #define **COMMAND_IDENTIFIER** ':'
- #define **DEBUG_FUNCTIONS**
- #define **PROTOCOL_STATE_DEFAULT** ePROTOCOL_IDLE

Typedefs

- typedef bool(∗ **TX_CB**) (uint8_t ∗, uint8_t)
- typedef void(∗ **DBG_FCN_CB**) (void)

Enumerations

- enum **PROTOCOL_STATE** { **ePROTOCOL_IDLE** , **ePROTOCOL_RECEIVING** , **ePROTOCOL_EVALUATING** , **ePROTOCOL_SENDING** , **ePROTOCOL_DEBUG** }
- enum **DEBUG_ACTIVATION_STATE** { **eDEBUG_ACTIVATION_NONE** , **eDEBUG_ACTIVATION_S1** , **eDEBUG_ACTIVATION_S2** , **eDEBUG_ACTIVATION_FINAL** }

4.12.1 Detailed Description

Request - Response protocol functionality.

Author

Roman Holderried

This serial protocol has been initially written for the MMX heater controller module. It provides data read/write access and a command interface to the application.

History

- 2022-01-13 - File creation

4.13 SerialProtocol.h

Go to the documentation of this file.

```
1 /*****************************************************************************/
15 #ifndef _SERIALPROTOCOL_H_
16 #define _SERIALPROTOCOL_H_
17
18 /*****************************************************************************
19  * Includes
20  *****************************************************************************/
21 #include <stdint.h>
22 #include <stdbool.h>
23 #include "Commands.h"
24 #include "Variables.h"
25 #include "Buffer.h"
26 #include "CommandStucture.h"
27
28 /*****************************************************************************
29  * Defines
30  *****************************************************************************/
31 #define TRANSMIT_BUFFER_LENGTH  64
32 // #define NUMBER_OF_CONTROL_BYTES 2
33 #define TXRX_BUFFER_LENGTH      TRANSMIT_BUFFER_LENGTH
34
35 #define STX 0x02
36 #define ETX 0x03
37
38 #define GETVAR_IDENTIFIER   '?'
39 #define SETVAR_IDENTIFIER   '!'
40 #define COMMAND_IDENTIFIER  ':'
41
42 #define DEBUG_FUNCTIONS
43
44 /*****************************************************************************
45  * Type definitions
46  *****************************************************************************/
47 typedef bool(*TX_CB)(uint8_t*, uint8_t);
48 typedef void(*DBG_FCN_CB)(void);
49
50 typedef enum
51 {
52     ePROTOCOL_IDLE,
53     ePROTOCOL_RECEIVING,
54     ePROTOCOL_EVALUATING,
55     ePROTOCOL_SENDING,
56     ePROTOCOL_DEBUG
57 }PROTOCOL_STATE;
58
59 #define PROTOCOL_STATE_DEFAULT ePROTOCOL_IDLE
60
61 typedef enum
62 {
63     eDEBUG_ACTIVATION_NONE,
64     eDEBUG_ACTIVATION_S1,
65     eDEBUG_ACTIVATION_S2,
66     eDEBUG_ACTIVATION_FINAL
67 }DEBUG_ACTIVATION_STATE;
68
69
70 /*****************************************************************************
71  * Class declarations
72  *****************************************************************************/
73 class SerialProtocol
74 {
75     public:
76
77     struct {
78         PROTOCOL_STATE  e_state;
79         bool            b_error;
80         bool            b_sent;
82         #ifdef DEBUG_FUNCTIONS
83         DEBUG_ACTIVATION_STATE e_dbgActState;
84         DBG_FCN_CB  debugFcnArray[10] = {nullptr};
85         #endif
86     } control;
87
88     uint8_t txRxBuffer[TXRX_BUFFER_LENGTH] = {0};
91     Buffer rxBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer);
92     Buffer txBuffer = Buffer(TXRX_BUFFER_LENGTH, this->txRxBuffer);
94     TX_CB       txCallback = nullptr;
96     // Public methods
97
102     SerialProtocol();
103
110     void setupCallbacks(TX_CB transmit_cb, READEEPROM_CB readEEPROM_cb, WRITEEEPROM_CB writeEEPROM_cb);
```

```
111
117     void setupVariableStructure(VAR *p_varStruct, uint8_t ui8_structLen);
118
124     void setupCommandStructure(COMMAND_CB *p_cmdStruct, uint8_t ui8_structLen);
125
131     void statemachine   (void);
132
137     void receive        (uint8_t ui8_data);
138
139     private:
140
141     SerialCommands cmdModule = SerialCommands();
149     COMMAND commandParser(uint8_t *pui8_buf, uint8_t ui8_stringSize);
150
160     uint8_t responseBuilder(uint8_t *pui8_buf, RESPONSE response);
161
162 };
163
164 #endif //_SERIALPROTOCOL_H_
```

4.14 VarAccess.cpp File Reference

Definitions for the VarAccess module.

```
#include <stdint.h>
#include <stdbool.h>
#include "Variables.h"
```

Variables

- const uint8_t **ui8_byteLength** [7] = {1,1,2,2,4,4,4}

4.14.1 Detailed Description

Definitions for the VarAccess module.

Author

Roman Holderried

History

- 2022-01-18 - File creation

4.15 Variables.h File Reference

Declarations for the variable structure.

```
#include "stdint.h"
```

Classes

- struct VAR

  Variable struct member declaration.
- class VarAccess

Macros

- #define **EEPROM_BYTE_ADRESSABLE** 1
- #define **EEPROM_WORD_ADRESSABLE** 2
- #define **EEPROM_LONG_ADRESSABLE** 4
- #define **EEPROM_ADDRESSTYPE_DEFAULT** EEPROM_BYTE_ADRESSABLE
- #define **EEPROM_ADDRESSTYPE** EEPROM_ADDRESSTYPE_DEFAULT
- #define **ADDRESS_OFFSET_DEFAULT** 0
- #define **ADDRESS_OFFET** ADDRESS_OFFSET_DEFAULT

Typedefs

- typedef bool(∗ **WRITEEEPROM_CB**) (uint32_t ui32_val, uint16_t ui16_address)

  EEPROM write user callback.
- typedef bool(∗ **READEEPROM_CB**) (uint32_t ∗ui32_val, uint16_t ui16_address)

  EEPROM read user callback.

Enumerations

- enum TYPE { eVARTYPE_NONE , eVARTYPE_EEPROM , eVARTYPE_RAM }

  Reflects the storage type of the linked variable.
- enum DTYPE {
  eDTYPE_UINT8 = 0 , **eDTYPE_INT8** = 1 , **eDTYPE_UINT16** = 2 , **eDTYPE_INT16** = 3 ,
  eDTYPE_UINT32 = 4 , **eDTYPE_INT32** = 5 , **eDTYPE_F32** = 6 }

  Reflects the data type of the linked variable.

4.15.1 Detailed Description

Declarations for the variable structure.

Author

Roman Holderried

Variale definitions take place in an external, user defined source file. The variable structure must be based on the types defined by this header.

History

- 2022-01-14 - File creation

4.15.2 Enumeration Type Documentation

4.15.2.1 TYPE

`enum TYPE`

Reflects the storage type of the linked variable.

Enumerator

eVARTYPE_NONE	Unknown storage type. Shouldn't be used.
eVARTYPE_EEPROM	EEPROM variable. Command execution will call EEPROM write/read functions on setVar/getVar.
eVARTYPE_RAM	RAM variable. Just the linked variable will be accessed, no EEPROM read/write.

4.16 Variables.h

Go to the documentation of this file.

```
1 /*****************************************************************************/
13 #ifndef _VARIABLES_H_
14 #define _VARIABLES_H_
15
16 /*****************************************************************************
17  * Includes
18  *****************************************************************************/
19 #include "stdint.h"
20
21 /*****************************************************************************
22  * defines
23  *****************************************************************************/
24 #define EEPROM_BYTE_ADRESSABLE      1
25 #define EEPROM_WORD_ADRESSABLE      2
26 #define EEPROM_LONG_ADRESSABLE      4
27
28 #define EEPROM_ADDRESSTYPE_DEFAULT    EEPROM_BYTE_ADRESSABLE
29
30 #ifndef EEPROM_ADDRESSTYPE
31 #define EEPROM_ADDRESSTYPE EEPROM_ADDRESSTYPE_DEFAULT
32 #endif
33
34 #define ADDRESS_OFFSET_DEFAULT  0
35 #ifndef ADDRESS_OFFET
36 #define ADDRESS_OFFET ADDRESS_OFFSET_DEFAULT
37 #endif
38
39 /*****************************************************************************
40  * Type definitions
41  *****************************************************************************/
42
44 typedef enum
45 {
46     eVARTYPE_NONE,
47     eVARTYPE_EEPROM,
48     eVARTYPE_RAM
49 }TYPE;
50
52 typedef enum
53 {
54     eDTYPE_UINT8  = 0,
55     eDTYPE_INT8   = 1,
56     eDTYPE_UINT16 = 2,
57     eDTYPE_INT16  = 3,
58     eDTYPE_UINT32 = 4,
59     eDTYPE_INT32  = 5,
60     eDTYPE_F32    = 6
61 }DTYPE;
62
64 typedef struct
65 {
66     void    *val;
67     TYPE    vartype;
68     DTYPE   datatype;
70     struct
71     {
72         uint16_t    ui16_eeAddress;
73         uint8_t     ui8_byteLength;
74     }runtime;
75 }VAR;
76
78 typedef bool(*WRITEEEPROM_CB)(uint32_t ui32_val, uint16_t ui16_address);
80 typedef bool(*READEEPROM_CB)(uint32_t *ui32_val, uint16_t ui16_address);
81
82 class VarAccess
83 {
```

```
84      public:
85
86      uint8_t ui8_varStructLength;
87      VAR     *p_varStruct;
90      VarAccess();
91
99      bool initVarstruct();
100
101      WRITEEEPROM_CB  writeEEPROM_cb = nullptr;
102      READEEPROM_CB   readEEPROM_cb  = nullptr;
110      bool readValFromVarStruct(int16_t i16_varNum, float *pf_val);
111
118      bool writeValToVarStruct(int16_t i16_varNum, float f_val);
119
125      bool readEEPROMValueIntoVarStruct(int16_t i16_varNum);
126
132      bool writeEEPROMwithValueFromVarStruct(int16_t i16_varNum);
133 };
134
135 #endif //_VARIABLES_H_
```