DashLogger

Generated by Doxygen 1.9.4

Chapter 1

Todo List

Module CANCOMM

Implement Transmit Functionality

Chapter 2

Bug List

Module CANCOMM

Only Interface One was tested

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Module Documentation

6.1 CANCOMM

Files

- file cancomm.h

  This File defines the Prototypes for CANCOMM.
- file cancomm.c

  This File Implements the Prototypes for CANCOMM.

Data Structures

- struct cancomm_interface

  Structure defining a CAN Interface.
- struct cancomm_message

  Structure defining a CAN Message.

Macros

- #define **CANCOMM_MAXIMUM_DATA_LENGTH** 8

  The Maximum Bytes of Data per CAN Frame.
- #define **CANCOMM_MAXIMUM_NAME_LENGTH** 30

  The Maximum Friendly Name Length of the CAN Messages.

Functions

- void CANCOMM_ReadMessages (cancomm_message *message_list, uint32_t message_list_len, cancomm_interface *interface_list, uint8_t interface_list_len)

  This function recieves the CAN Messages defined in a message_list using the provided interface_list.

6.1.1 Detailed Description

This Module is configured using a list of cancomm_interface Structs. These provide the Recieve and Transmit Functions for the CAN Interface. Which Messages should be recieved and transmitted is configured using a list of cancomm_message Structs.

Todo Implement Transmit Functionality

Bug Only Interface One was tested

6.1.2 Function Documentation

6.1.2.1 CANCOMM_ReadMessages()

```
void CANCOMM_ReadMessages (
            cancomm_message * message_list,
            uint32_t message_list_len,
            cancomm_interface * interface_list,
            uint8_t interface_list_len )
```

This function recieves the CAN Messages defined in a message_list using the provided interface_list.

Parameters

message_list	List of cancom_message Structs
message_list_len	Length of `message_list`
interface_list	List of cancomm_interface Structs
interface_list_len	Length of `interface_list`

6.2 COMMAND

Files

- file command.h

  This File defines the Prototypes for COMMAND.
- file command.c

  This File implements the Prototypes for COMMAND.

Functions

- void COMMAND_Generate (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *next_command, SHORTPROTOCOL_Instance *shortProt)

  Adds a Command from `signal_list` if `shortProt` is ready.

6.2.1 Detailed Description

This Module Picks the signals_signal from a signal_list wich are of Type SIGNALS_DISPLAY_SIGNAL and Generates a Comand to change the Text Object on the Display with ID signals_signal.oject_id to signals_signal.string_↩ value.

6.2.2 Function Documentation

6.2.2.1 COMMAND_Generate()

```
void COMMAND_Generate (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            uint32_t * next_command,
            SHORTPROTOCOL_Instance * shortProt )
```

Adds a Command from `signal_list` if `shortProt` is ready.

Parameters

signal_list	A list of signals_signal
signal_list_len	The Length of `signal_list`
next_command	A pointer to an Integer. This represents the next Command to be Sent. This value should only be changed by this Function.
shortProt	A SHORTPROTOCOL_Instance for sending Packets to the Display.

6.3 SIGNALS

Files

- file signals.h

  This File defines the Prototypes for SIGNALS.

Data Structures

- struct SIGNALS_string

  A struct to combine a String with it's length.
- struct signals_signal_struct

Macros

- #define **SIGNALS_MAXIMUM_NAME_LENGTH** 30

  The maximum friendly name length for a signal.
- #define **SIGNALS_STRING_MAXIMUM_LENGTH** 50

  The maximum length of the string value of a signal.

Typedefs

- typedef struct signals_signal_struct signals_signal

Enumerations

- enum signals_result { SIGNALS_FOUND = 0 , SIGNALS_NOT_FOUND , SIGNALS_MATCH , SIGNALS_NO_MATCH }

  Definition of the Return Values of some Functions of this Module.
- enum signals_data_type { **SIGNALS_FLOAT_SIGNAL** = 0 , **SIGNALS_UINT32_T_SIGNAL** , **SIGNALS_STRING_SIGNAL** }

  The Data type of a Signal.
- enum signals_signal_type { **SIGNALS_CAN_MESSAGE** = 0 , **SIGNALS_INTERNAL_SIGNAL** , **SIGNALS_DISPLAY_SIGNAL** }

  The Signal type of a Signal.

Functions

- void **SIGNALS_Interpret** (signals_signal *signal_list, uint32_t signal_list_len, cancomm_message *message_list, uint32_t message_list_len)
- signals_result **signals_find_data** (uint32_t id, uint8_t interface, cancomm_message *message_list, uint32_t message_list_len, uint8_t *data)
- signals_signal * **signals_find_signal** (signals_signal *signal_list, uint32_t signal_list_len, void(*Callback)(void))
- signals_result **signals_compare_names** (uint8_t *first, uint32_t firstlen, uint8_t *second, uint32_t secondlen)
- signals_signal * **signals_find_display_signal** (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *dispSignalCount, uint32_t needle)

6.3.1 Detailed Description

This Module Handles the Information Interpretation Handling of All Inputs to Outputs. Signals have a Type, wich changes how it is interpreted and used. For more on this, refer to signals_data_type and signals_signal_type. Signals can be chained together. For Example, four Signals of Type SIGNALS_CAN_MESSAGE could recieve four Temperatures, wich would then be interpreted to float Values using thier respective Callback Functions. Another Signal of Type SIGNALS_INTERNAL_SIGNAL could then find the highest of these Temperatures using its Callback Function. To complete the Chain, a Signal of Type SIGNALS_DISPLAY_SIGNAL can be used to convert the float value of the previous Maximising Function to a String Command wich can be interpreted by a Display.

6.3.2 Typedef Documentation

6.3.2.1 signals_signal

```
typedef struct signals_signal_struct signals_signal
```

Type Definition of signals_signal_struct to be able to use signals_signal in the definition of signals_signal, as signal structs have to store pointers to other signals_signal structs.

6.3.3 Enumeration Type Documentation

6.3.3.1 signals_result

```
enum signals_result
```

Definition of the Return Values of some Functions of this Module.

Enumerator

SIGNALS_FOUND	Returned if a Signal is found
SIGNALS_NOT_FOUND	Returned if no Signal is found
SIGNALS_MATCH	Returned if the Inputs match
SIGNALS_NO_MATCH	Returned if the Inputs dont match

Chapter 7

Data Structure Documentation

7.1 cancomm_interface Struct Reference

Structure defining a CAN Interface.

`#include <cancomm.h>`

Data Fields

- uint8_t **number**

  The Interface Number (Has to be unique)
- uint8_t **receiveFifo**

  The FIFO Number used for Recieving Frames.
- uint8_t **transmitFifo**

  The FIFO Number used for Transmitting Frames.
- bool(∗ MessageTransmit)(uint32_t id, uint8_t length, uint8_t ∗data, uint8_t fifoQueueNum, CANFD_MODE mode, CANFD_MSG_TX_ATTRIBUTE msgAttr)

  Callback Function to Transmit CAN Messages.
- bool(∗ MessageReceive)(uint32_t ∗id, uint8_t ∗length, uint8_t ∗data, uint32_t ∗timestamp, uint8_t fifoNum, CANFD_MSG_RX_ATTRIBUTE ∗msgAttr)

  Callback Function to Recieve CAN Messages.

7.1.1 Detailed Description

Structure defining a CAN Interface.

7.1.2 Field Documentation

7.1.2.1 MessageReceive

```
bool(* cancomm_interface::MessageReceive) (uint32_t *id, uint8_t *length, uint8_t *data, uint32↩
_t *timestamp, uint8_t fifoNum, CANFD_MSG_RX_ATTRIBUTE *msgAttr)
```

Callback Function to Recieve CAN Messages.

This Function matches the Protoype generated by MPLAB Harmony, so this Function Pointer can be set the the Functions generated by Harmony.

Parameters

id	The ID of the Recieved Message
length	The DLC of the Recieved Message
data	The Data of the Recieved Message
timestamp	The Timestamp of the Recieved Message
fifoNum	The FIFO used for recieving Messages
msgAttr	If the recieved Message was a Data Frame, this is CANFD_MSG_TX_DATA_FRAME

Returns

False if there is no Message in the Recieve FIFO left

7.1.2.2 MessageTransmit

```
bool(* cancomm_interface::MessageTransmit) (uint32_t id, uint8_t length, uint8_t *data, uint8↩
_t fifoQueueNum, CANFD_MODE mode, CANFD_MSG_TX_ATTRIBUTE msgAttr)
```

Callback Function to Transmit CAN Messages.

This Function matches the Protoype generated by MPLAB Harmony, so this Function Pointer can be set the the Functions generated by Harmony.

Parameters

id	The ID of the Message
length	The DLC of the Message
data	The Data Bytes the Message should have
fifoQueueNum	The Number of the transmit FIFO
mode	For non CAN-FD Messages this is CANFD_MODE_NORMAL
msgAttr	For a Data Frame this is CANFD_MSG_TX_DATA_FRAME

Returns

True if the Message was sucessfully added to the transmit FIFO

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

- cancomm.h

7.2 cancomm_message Struct Reference

Structure defining a CAN Message.

```
#include <cancomm.h>
```

Data Fields

- uint32_t **id**

  The ID of the Message.
- uint8_t **interface_number**

  The interface number (wich has to be unique to a CAN Interface)
- uint8_t **length**

  The DLC of the Message.
- uint8_t **data** [CANCOMM_MAXIMUM_DATA_LENGTH]

  The Data of the Message.
- uint32_t **timestamp**

  The timestamp when the Message was recieved.
- uint8_t **friendly_name** [CANCOMM_MAXIMUM_NAME_LENGTH]

  A friendly Human Readable Name for the Message.

7.2.1 Detailed Description

Structure defining a CAN Message.

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

- cancomm.h

7.3 SHORTPROTOCOL_Instance Struct Reference

Data Fields

- uint8_t(∗ **readByte**)(void)
- SHORTPROTOCOL_status(∗ **readAvailable**)(void)
- void(∗ **writeByte**)(uint8_t)
- SHORTPROTOCOL_status(∗ **writeAvailable**)(void)
- uint32_t **maximumPackageLength**
- SHORTPROTOCOL_string **command_buffer**
- SHORTPROTOCOL_status **newCommand**
- uint32_t **writeCounter**

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

- shortprotocol.h

7.4 SHORTPROTOCOL_string Struct Reference

Data Fields

- uint32_t **length**
- uint8_t **data** [SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH+SHORTPROTOCOL_OVERHEAD↩ _BYTES]

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

- shortprotocol.h

7.5 signals_signal_struct Struct Reference

`#include <signals.h>`

Data Fields

- uint32_t **id**
- uint8_t **interface_number**
- signals_data_type **data_type**
- float(∗ **can_convert_float**)(uint8_t ∗data)
- float(∗ **internal_convert_float**)(signals_signal ∗signal_list, uint32_t signal_list_len)
- float **value_float**
- uint32_t(∗ **can_convert_uint32_t**)(uint8_t ∗data)
- uint32_t(∗ **internal_convert_uint32_t**)(signals_signal ∗signal_list, uint32_t signal_list_len)
- uint32_t **value_uint32_t**
- void(∗ **can_convert_string**)(uint8_t ∗data, SIGNALS_string ∗string)
- void(∗ **internal_convert_string**)(signals_signal ∗signal_list, uint32_t signal_list_len, SIGNALS_string ∗string)
- SIGNALS_string **value_string**
- uint8_t **friendly_name** [SIGNALS_MAXIMUM_NAME_LENGTH]
- uint32_t **timestamp**
- signals_signal_type **type**
- uint32_t **object_id**

7.5.1 Detailed Description

This Struct defines a Signal. Every Signal should have a friendly Name, to be easily identified by a human. Every Signal has to have a signals_signal.type and a signals_signal.data_type. Depending on the signals_signal.type and signals_signal.data_type other members of the struct have to be defined. If a signal is of type SIGNALS↩_CAN_MESSAGE, the id and the interface_number have to be defined. The Signal can be either of data_type SIGNALS_FLOAT_SIGNAL, SIGNALS_UINT32_T_SIGNAL or SIGNALS_STRING_SIGNAL. The corresponding callback function can_convert_float, can_convert_uint32_t or can_convert_string has to be defined.

If a signal is of type SIGNALS_INTERNAL_SIGNAL, it can be of data_type SIGNALS_FLOAT_SIGNAL, SIGNALS↩_UINT32_T_SIGNAL or SIGNALS_STRING_SIGNAL. The corresponding callback function internal_convert_float, internal_convert_uint32_t or internal_convert_string has to be defined.

If a signal is of type SIGNALS_DISPLAY_SIGNAL, it's data_type has to be SIGNALS_STRING_SIGNAL, and its callback function internal_convert_string has to be defined in a way to produce correct Commands for the Display to be interpreted. Signals of this Type are read by COMMAND to be sent of to the Display using the SHORTPROTOCOL.

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

- signals.h

7.6 SIGNALS_string Struct Reference

A struct to combine a String with it's length.

`#include <signals.h>`

Data Fields

- uint32_t **length**
- uint8_t **data** [SIGNALS_STRING_MAXIMUM_LENGTH]

7.6.1 Detailed Description

A struct to combine a String with it's length.

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

- signals.h

Chapter 8

File Documentation

8.1 cancomm.c File Reference

This File Implements the Prototypes for CANCOMM.

```
#include "cancomm.h"
```

Functions

- void CANCOMM_ReadMessages (cancomm_message *message_list, uint32_t message_list_len, cancomm_interface *interface_list, uint8_t interface_list_len)

  This function recieves the CAN Messages defined in a message_list using the provided interface_list.

8.1.1 Detailed Description

This File Implements the Prototypes for CANCOMM.

Author

Frederic Emmerth

8.2 cancomm.h File Reference

This File defines the Prototypes for CANCOMM.

```
#include "definitions.h"
```

Data Structures

- struct cancomm_interface

  Structure defining a CAN Interface.
- struct cancomm_message

  Structure defining a CAN Message.

Macros

- #define **CANCOMM_MAXIMUM_DATA_LENGTH** 8

  The Maximum Bytes of Data per CAN Frame.
- #define **CANCOMM_MAXIMUM_NAME_LENGTH** 30

  The Maximum Friendly Name Length of the CAN Messages.

Functions

- void CANCOMM_ReadMessages (cancomm_message *message_list, uint32_t message_list_len, cancomm_interface *interface_list, uint8_t interface_list_len)

  This function recieves the CAN Messages defined in a message_list using the provided interface_list.

8.2.1 Detailed Description

This File defines the Prototypes for CANCOMM.

Author

Frederic Emmerth

8.3 cancomm.h

Go to the documentation of this file.

```
1
27 #ifndef CANCOMM_H
28 #define CANCOMM_H
29
30 #ifdef __cplusplus
31 extern "C" {
32 #endif
33
34 #include "definitions.h"
35
39 #define CANCOMM_MAXIMUM_DATA_LENGTH     8
40
44 #define CANCOMM_MAXIMUM_NAME_LENGTH     30
45
46
50 typedef struct{
54     uint8_t number;
55
59     uint8_t receiveFifo;
60
64     uint8_t transmitFifo;
65
81     bool (*MessageTransmit)(uint32_t id,
82             uint8_t length, uint8_t* data,
83             uint8_t fifoQueueNum, CANFD_MODE mode,
84             CANFD_MSG_TX_ATTRIBUTE msgAttr);
85
103     bool (*MessageReceive)(uint32_t *id,
104              uint8_t *length, uint8_t *data,
105              uint32_t *timestamp, uint8_t fifoNum,
106              CANFD_MSG_RX_ATTRIBUTE *msgAttr);
107 }cancomm_interface;
108
109
113 typedef struct{
117     uint32_t id;
118
122     uint8_t interface_number;
123
127     uint8_t length;
128
132     uint8_t data [CANCOMM_MAXIMUM_DATA_LENGTH];
```

```
133
137     uint32_t timestamp;
138
142     uint8_t friendly_name [CANCOMM_MAXIMUM_NAME_LENGTH];
143 }cancomm_message;
144
145
154 void CANCOMM_ReadMessages(cancomm_message* message_list,
155         uint32_t message_list_len,
156         cancomm_interface* interface_list,
157         uint8_t interface_list_len);
158
159 #ifdef  __cplusplus
160 }
161 #endif
162
163 #endif  /* CANCOMM_H */
164
```

8.4 command.c File Reference

This File implements the Prototypes for COMMAND.

```
#include "command.h"
```

Functions

- void COMMAND_Generate (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *next_command, SHORTPROTOCOL_Instance *shortProt)

  Adds a Command from `signal_list` if `shortProt` is ready.

8.4.1 Detailed Description

This File implements the Prototypes for COMMAND.

Author

Frederic Emmerth

8.5 command.h File Reference

This File defines the Prototypes for COMMAND.

```
#include "definitions.h"
#include "signals.h"
#include "shortprotocol.h"
```

Functions

- void COMMAND_Generate (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *next_command, SHORTPROTOCOL_Instance *shortProt)

  Adds a Command from `signal_list` if `shortProt` is ready.

8.5.1 Detailed Description

This File defines the Prototypes for COMMAND.

Author

Frederic Emmerth

8.6 command.h

Go to the documentation of this file.

```
1
22 #ifndef COMMAND_H
23 #define COMMAND_H
24
25 #ifdef  __cplusplus
26 extern "C" {
27 #endif
28
29 #include "definitions.h"
30 #include "signals.h"
31 #include "shortprotocol.h"
32
44 void COMMAND_Generate(signals_signal* signal_list, uint32_t signal_list_len,
45         uint32_t* next_command, SHORTPROTOCOL_Instance* shortProt);
46
47
48 #ifdef  __cplusplus
49 }
50 #endif
51
52 #endif  /* COMMAND_H */
53
```

8.7 conv.h

```
1
2
3 #ifndef CONV_H
4 #define CONV_H
5
6 #ifdef  __cplusplus
7 extern "C" {
8 #endif
9
10 #include "definitions.h"
11 #include "signals.h"
12 #include "stdio.h"
13
14 /* User-Defined Callback Functions to Convert the RAW CAN Data to Values */
15 /* Numbers are Hardcoded here, to make it more Readable */
16
17 float CONV_MinVoltage(uint8_t* data);
18 float CONV_MaxTemp(uint8_t* data);
19 float CONV_LapTime(uint8_t* data);
20 float CONV_BestLapTime(uint8_t* data);
21 uint32_t CONV_FSG_AMI_state(uint8_t* data);
22 float CONV_MaxMotTemp(uint8_t* data);
23 float CONV_MaxInvTemp(uint8_t* data);
24 float CONV_MotorTemp_RR(uint8_t* data);
25 float CONV_MotorTemp_RL(uint8_t* data);
26 float CONV_MotorTemp_FL(uint8_t* data);
27 float CONV_MotorTemp_FR(uint8_t* data);
28 float CONV_MaxMotorTemp(signals_signal* signal_list, uint32_t signal_list_len);
29 void CONV_DISP_Motor_Temp(signals_signal* signal_list, uint32_t signal_list_len,
30         SIGNALS_string* outstring);
31 void CONV_DISP_MinVoltage(signals_signal* signal_list, uint32_t signal_list_len,
32         SIGNALS_string* outstring);
33 uint32_t CONV_find_string_length(uint8_t* str, uint32_t strlen);
34
35 float CONV_max(float* vals, uint32_t valcount);
36 float CONV_min(float* vals, uint32_t valcount);
37
```

```
38 #ifdef  __cplusplus
39 }
40 #endif
41
42 #endif  /* CONV_H */
43
```

8.8 crc.c File Reference

```
#include "crc.h"
#include <stdlib.h>
#include <stdint.h>
```

Functions

- crc_t crc_update (crc_t crc, const void *data, size_t data_len)
- uint32_t CRC_Calculate (void *data, uint32_t length)

8.8.1 Detailed Description

Functions and types for CRC checks.

Generated on Wed May 11 23:00:19 2022 by pycrc v0.9.2, https://pycrc.org using the configuration:

- Width = 16
- Poly = 0x1021
- XorIn = 0x1d0f
- ReflectIn = False
- XorOut = 0x0000
- ReflectOut = False
- Algorithm = table-driven

8.8.2 Function Documentation

8.8.2.1 CRC_Calculate()

```
uint32_t CRC_Calculate (
            void * data,
            uint32_t length )
```

Wrapper function to Calculate the CRC over a Array

Parameters

in	*data*	The Array of Data to calculate the CRC of.
in	*length*	The Length of the Input Data Array

Returns

The CRC Value.

8.8.2.2 crc_update()

```
crc_t crc_update (
            crc_t crc,
            const void * data,
            size_t data_len )
```

Update the crc value with new data.

Parameters

in	*crc*	The current crc value.
in	*data*	Pointer to a buffer of *data_len* bytes.
in	*data_len*	Number of bytes in the *data* buffer.

Returns

The updated crc value.

8.9 crc.h File Reference

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

Macros

- #define CRC_ALGO_TABLE_DRIVEN 1

Typedefs

- typedef uint_fast16_t crc_t

Functions

- crc_t crc_update (crc_t crc, const void ∗data, size_t data_len)
- uint32_t CRC_Calculate (void ∗data, uint32_t length)

8.9.1 Detailed Description

Functions and types for CRC checks.

Generated on Wed May 11 23:00:05 2022 by pycrc v0.9.2, https://pycrc.org using the configuration:

- Width = 16
- Poly = 0x1021
- XorIn = 0x1d0f
- ReflectIn = False
- XorOut = 0x0000
- ReflectOut = False
- Algorithm = table-driven

This file defines the functions crc_init(), crc_update() and crc_finalize().

The crc_init() function returns the inital `crc` value and must be called before the first call to crc_update(). Similarly, the crc_finalize() function must be called after the last call to crc_update(), before the `crc` is being used. is being used.

The crc_update() function can be called any number of times (including zero times) in between the crc_init() and crc_finalize() calls.

This pseudo-code shows an example usage of the API:

```
crc_t crc;
unsigned char data[MAX_DATA_LEN];
size_t data_len;
crc = crc_init();
while ((data_len = read_data(data, MAX_DATA_LEN)) > 0) {
    crc = crc_update(crc, data, data_len);
}
crc = crc_finalize(crc);
```

8.9.2 Macro Definition Documentation

8.9.2.1 CRC_ALGO_TABLE_DRIVEN

```
#define CRC_ALGO_TABLE_DRIVEN 1
```

The definition of the used algorithm.

This is not used anywhere in the generated code, but it may be used by the application code to call algorithm-specific code, if desired.

8.9.3 Typedef Documentation

8.9.3.1 crc_t

```
typedef uint_fast16_t crc_t
```

The type of the CRC values.

This type must be big enough to contain at least 16 bits.

8.9.4 Function Documentation

8.9.4.1 CRC_Calculate()

```
uint32_t CRC_Calculate (
            void * data,
            uint32_t length )
```

Wrapper function to Calculate the CRC over a Array

Parameters

in	*data*	The Array of Data to calculate the CRC of.
in	*length*	The Length of the Input Data Array

Returns

The CRC Value.

8.9.4.2 crc_update()

```
crc_t crc_update (
            crc_t crc,
            const void * data,
            size_t data_len )
```

Update the crc value with new data.

Parameters

in	*crc*	The current crc value.
in	*data*	Pointer to a buffer of *data_len* bytes.
in	*data_len*	Number of bytes in the *data* buffer.

Returns

The updated crc value.

8.10 crc.h

Go to the documentation of this file.

```
1
40 #ifndef CRC_H
41 #define CRC_H
42
43 #include <stdlib.h>
44 #include <stdint.h>
45
46 #ifdef __cplusplus
47 extern "C" {
48 #endif
49
50
57 #define CRC_ALGO_TABLE_DRIVEN 1
58
59
65 typedef uint_fast16_t crc_t;
66
67
73 static inline crc_t crc_init(void)
74 {
75     return 0xFFFF;
76 }
77
78
87 crc_t crc_update(crc_t crc, const void *data, size_t data_len);
88
89
96 static inline crc_t crc_finalize(crc_t crc)
97 {
98     return crc;
99 }
100
110 uint32_t CRC_Calculate(void* data, uint32_t length);
111
112
113 #ifdef __cplusplus
114 }           /* closing brace for extern "C" */
115 #endif
116
117 #endif      /* CRC_H */
```

8.11 delay.h

```
1 /*
2  * File:   delay.h
3  * Author: Frederic
4  *
5  * Created on 11. Mai 2022, 17:54
6  */
7
8 #ifndef DELAY_H
9 #define DELAY_H
10
11 #ifdef  __cplusplus
12 extern "C" {
13 #endif
14
15 #include <stdint.h>
16
17 #define CLOCK_FREQUENCY_CORE 120000000
18 #define MICROSECONDS_IN_SECOND 1000000
19 #define MILLISECONDS_IN_SECOND 1000
20 #define TWO_STEPS_DELAY_ADJ 2
21
22 void DELAY_Milliseconds(uint32_t delay);
23 void DELAY_Microseconds(uint32_t delay);
24
25
26 #ifdef  __cplusplus
27 }
```

```
28 #endif
29
30 #endif  /* DELAY_H */
31
```

8.12 shortprotocol.h

```
1 /*
2  * File:   shortprotocol.h
3  * Author: Frederic
4  *
5  * Created on 19. Mai 2022, 13:37
6  */
7
8 #ifndef SHORTPROTOCOL_H
9 #define SHORTPROTOCOL_H
10
11 #ifdef  __cplusplus
12 extern "C" {
13 #endif
14
15 #include "definitions.h"
16 #include "crc.h"
17
18 #define SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH 100
19 #define SHORTPROTOCOL_BEGIN 0x13
20 #define SHORTPROTOCOL_OVERHEAD_BYTES 5
21 #define SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES 3
22 #define SHORTPROTOCOL_FIRST_BYTE_MASK 0x00FF
23 #define SHORTPROTOCOL_SECOND_BYTE_MASK 0xFF00
24 #define SHORTPROTOCOL_BYTE_LENGTH 8
25 #define SHORTPROTOCOL_BEGIN_OFFSET 0
26 #define SHORTPROTOCOL_LENGTH_LSB_OFFSET 1
27 #define SHORTPROTOCOL_LENGTH_MSB_OFFSET 2
28 #define SHORTPROTOCOL_COMMAND_OFFSET 3
29 #define SHORTPROTOCOL_CRC_LSB_OFFSET 0
30 #define SHORTPROTOCOL_CRC_MSB_OFFSET 1
31
32 typedef enum{
33     SHORTPROTOCOL_SUCCESS = 0,
34     SHORTPROTOCOL_ERROR,
35     SHORTPROTOCOL_AVAILABLE,
36     SHORTPROTOCOL_NOT_AVAILABLE
37 }SHORTPROTOCOL_status;
38
39 typedef struct{
40     uint32_t length;
41     uint8_t data[SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH +
42                          SHORTPROTOCOL_OVERHEAD_BYTES];
43 }SHORTPROTOCOL_string;
44
45 typedef struct{
46     uint8_t (*readByte)(void);
47     SHORTPROTOCOL_status (*readAvailable)(void);
48     void (*writeByte)(uint8_t);
49     SHORTPROTOCOL_status (*writeAvailable)(void);
50     uint32_t maximumPackageLength;
51     SHORTPROTOCOL_string command_buffer;
52     SHORTPROTOCOL_status newCommand;
53     uint32_t writeCounter;
54
55 }SHORTPROTOCOL_Instance;
56
57 SHORTPROTOCOL_status SHORTPROTOCOL_Send(SHORTPROTOCOL_Instance* inst,
58         uint8_t* data, uint32_t length);
59
60 void SHORTPROTOCOL_Update(SHORTPROTOCOL_Instance* inst);
61
62 SHORTPROTOCOL_status SHORTPROTOCOL_Available(SHORTPROTOCOL_Instance* inst);
63
64 void SHORTPROTOCOL_Initialize(SHORTPROTOCOL_Instance* inst);
65
66 #ifdef  __cplusplus
67 }
68 #endif
69
70 #endif  /* SHORTPROTOCOL_H */
71
```

8.13 signals.h File Reference

This File defines the Prototypes for SIGNALS.

```
#include "definitions.h"
#include "cancomm.h"
```

Data Structures

- struct SIGNALS_string

  A struct to combine a String with it's length.
- struct signals_signal_struct

Macros

- #define **SIGNALS_MAXIMUM_NAME_LENGTH** 30

  The maximum friendly name length for a signal.
- #define **SIGNALS_STRING_MAXIMUM_LENGTH** 50

  The maximum length of the string value of a signal.

Typedefs

- typedef struct signals_signal_struct signals_signal

Enumerations

- enum signals_result { SIGNALS_FOUND = 0 , SIGNALS_NOT_FOUND , SIGNALS_MATCH , SIGNALS_NO_MATCH }

  Definition of the Return Values of some Functions of this Module.
- enum signals_data_type { **SIGNALS_FLOAT_SIGNAL** = 0 , **SIGNALS_UINT32_T_SIGNAL** , **SIGNALS_STRING_SIGNAL** }

  The Data type of a Signal.
- enum signals_signal_type { **SIGNALS_CAN_MESSAGE** = 0 , **SIGNALS_INTERNAL_SIGNAL** , **SIGNALS_DISPLAY_SIGNAL** }

  The Signal type of a Signal.

Functions

- void **SIGNALS_Interpret** (signals_signal *signal_list, uint32_t signal_list_len, cancomm_message *message_list, uint32_t message_list_len)
- signals_result **signals_find_data** (uint32_t id, uint8_t interface, cancomm_message *message_list, uint32_t message_list_len, uint8_t *data)
- signals_signal * **signals_find_signal** (signals_signal *signal_list, uint32_t signal_list_len, void(*Callback)(void))
- signals_result **signals_compare_names** (uint8_t *first, uint32_t firstlen, uint8_t *second, uint32_t secondlen)
- signals_signal * **signals_find_display_signal** (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *dispSignalCount, uint32_t needle)

8.13.1 Detailed Description

This File defines the Prototypes for SIGNALS.

Author

Frederic Emmerth

8.14 signals.h

Go to the documentation of this file.

```
1
30 #ifndef SIGNALS_H
31 #define SIGNALS_H
32
33 #ifdef  __cplusplus
34 extern "C" {
35 #endif
36
37 #include "definitions.h"
38 #include "cancomm.h"
39
43 #define SIGNALS_MAXIMUM_NAME_LENGTH     30
44
48 #define SIGNALS_STRING_MAXIMUM_LENGTH   50
49
53 typedef enum{
57     SIGNALS_FOUND = 0,
58
62     SIGNALS_NOT_FOUND,
63
67     SIGNALS_MATCH,
68
72     SIGNALS_NO_MATCH
73 }signals_result;
74
75
79 typedef enum{
80     SIGNALS_FLOAT_SIGNAL = 0,
81     SIGNALS_UINT32_T_SIGNAL,
82     SIGNALS_STRING_SIGNAL
83 }signals_data_type;
84
88 typedef enum{
89     SIGNALS_CAN_MESSAGE = 0,
90     SIGNALS_INTERNAL_SIGNAL,
91     SIGNALS_DISPLAY_SIGNAL
92 }signals_signal_type;
93
97 typedef struct{
98     uint32_t length;
99     uint8_t data[SIGNALS_STRING_MAXIMUM_LENGTH];
100 }SIGNALS_string;
101
107 typedef struct signals_signal_struct signals_signal;
108
135 struct signals_signal_struct{
136     uint32_t id;
137     uint8_t interface_number;
138     signals_data_type data_type;
139
140     float(*can_convert_float)(uint8_t* data);
141     float(*internal_convert_float)(signals_signal* signal_list,
142             uint32_t signal_list_len);
143     float value_float;
144
145     uint32_t(*can_convert_uint32_t)(uint8_t* data);
146     uint32_t(*internal_convert_uint32_t)(signals_signal* signal_list,
147             uint32_t signal_list_len);
148     uint32_t value_uint32_t;
149
150     void (*can_convert_string)(uint8_t* data, SIGNALS_string* string);
151     void (*internal_convert_string)(signals_signal* signal_list,
152             uint32_t signal_list_len, SIGNALS_string* string);
153     SIGNALS_string value_string;
154
155     uint8_t friendly_name[SIGNALS_MAXIMUM_NAME_LENGTH];
```

```
156     uint32_t timestamp;
157     signals_signal_type type;
158 
159     uint32_t object_id;
160 };
161 
162 void SIGNALS_Interpret(signals_signal* signal_list, uint32_t signal_list_len,
163         cancomm_message* message_list, uint32_t message_list_len);
164 
165 signals_result signals_find_data(uint32_t id, uint8_t interface,
166         cancomm_message* message_list, uint32_t message_list_len,
167         uint8_t* data);
168 
169 signals_signal* signals_find_signal( signals_signal* signal_list,
170         uint32_t signal_list_len, void(*Callback)(void));
171 
172 signals_result signals_compare_names(uint8_t* first, uint32_t firstlen,
173         uint8_t* second, uint32_t secondlen);
174 
175 signals_signal* signals_find_display_signal(signals_signal* signal_list,
176         uint32_t signal_list_len, uint32_t* dispSignalCount, uint32_t needle);
177 
178 #ifdef  __cplusplus
179 }
180 #endif
181 
182 #endif  /* SIGNALS_H */
183 
```

8.15 uart.h

```
1 /*
2  * File:   uart.h
3  * Author: Frederic
4  *
5  * Created on 23. Mai 2022, 17:10
6  */
7 
8 #ifndef UART_H
9 #define UART_H
10 
11 #ifdef  __cplusplus
12 extern "C" {
13 #endif
14 
15 #include "shortprotocol.h"
16 
17 SHORTPROTOCOL_status UART_ReadAvailable( void );
18 uint8_t UART_ReadByte( void );
19 SHORTPROTOCOL_status UART_WriteAvailable( void );
20 void UART_WriteByte(uint8_t byte);
21 
22 #ifdef  __cplusplus
23 }
24 #endif
25 
26 #endif  /* UART_H */
27 
```

Index