DashLogger

Generated by Doxygen 1.9.4

Chapter 1

PIC32MK Software for DashLogger Display

#Overview

Chapter 2

Todo List

Module CANCOMM

Implement Transmit Functionality

Handle Remote Frames on Recieve

Handle CAN FD Frames on Recieve

File main.c

Make a constant loop time

Implement Watchdog

Class signals_signal_struct

Combine the Different Signal Types to a Union to use less RAM

Chapter 3

Bug List

Module CANCOMM

Only Interface One was tested

File delay.h

Millisecond Deley is inaccurate

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

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

Chapter 7

Module Documentation

7.1 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.

7.1.1 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

Handle Remote Frames on Recieve

Handle CAN FD Frames on Recieve

Bug Only Interface One was tested

7.1.2 Macro Definition Documentation

7.1.2.1 CANCOMM_MAXIMUM_DATA_LENGTH

```
#define CANCOMM_MAXIMUM_DATA_LENGTH 8
```

The Maximum Bytes of Data per CAN Frame.

Definition at line 41 of file cancomm.h.

7.1.2.2 CANCOMM_MAXIMUM_NAME_LENGTH

```
#define CANCOMM_MAXIMUM_NAME_LENGTH 30
```

The Maximum Friendly Name Length of the CAN Messages.

Definition at line 46 of file cancomm.h.

7.1.3 Function Documentation

7.1.3.1 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`

Definition at line 17 of file cancomm.c.

7.2 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.

7.2.1 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.

7.2.2 Function Documentation

7.2.2.1 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.

Definition at line 17 of file command.c.

7.3 CONV

Files

- file conv.h

  This File defines the Prototypes for CONV.
- file conv.c

  This File implements the Prototypes for CONV.

Functions

- float CONV_MinBatVoltage (uint8_t *data)
- float CONV_MaxBatTemp (uint8_t *data)
- float CONV_LapTime (uint8_t *data)
- float CONV_BestLapTime (uint8_t *data)
- uint32_t CONV_FSG_AMI_state (uint8_t *data)
- float **CONV_MaxMotTemp** (uint8_t *data)
- float **CONV_MaxInvTemp** (uint8_t *data)
- float CONV_InverterTemp_FL (uint8_t *data)
- float CONV_InverterTemp_FR (uint8_t *data)
- float CONV_InverterTemp_RL (uint8_t *data)
- float CONV_InverterTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RL (uint8_t *data)
- float CONV_MotorTemp_FL (uint8_t *data)
- float CONV_MotorTemp_FR (uint8_t *data)
- float CONV_MaxMotorTemp (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_MaxInverterTemp (signals_signal *signal_list, uint32_t signal_list_len)
- void CONV_DISP_Motor_Temp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_MinVoltage (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapDelta (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_InverterTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_MaxBatTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_FSG_AMI_State (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- uint32_t CONV_find_string_length (uint8_t *str, uint32_t strlen)
- float CONV_max (float *vals, uint32_t valcount)
- float CONV_min (float *vals, uint32_t valcount)

7.3.1 Detailed Description

In this Module the Callback Functions for the Signals from the SIGNALS Module are defined.

There are Callback Functions for CAN Message Interpretation, for Internal Message Processing and Display Command Generation.

7.3.2 Function Documentation

7.3.2.1 CONV_BestLapTime()

```
float CONV_BestLapTime (
            uint8_t * data )
```

Definition at line 36 of file conv.c.

7.3.2.2 CONV_DISP_FSG_AMI_State()

```
void CONV_DISP_FSG_AMI_State (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 262 of file conv.c.

7.3.2.3 CONV_DISP_InverterTemp()

```
void CONV_DISP_InverterTemp (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 196 of file conv.c.

7.3.2.4 CONV_DISP_LapDelta()

```
void CONV_DISP_LapDelta (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 208 of file conv.c.

7.3.2.5 CONV_DISP_LapTime()

```
void CONV_DISP_LapTime (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 226 of file conv.c.

7.3.2.6 CONV_DISP_LastLapTime()

```
void CONV_DISP_LastLapTime (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 238 of file conv.c.

7.3.2.7 CONV_DISP_MaxBatTemp()

```
void CONV_DISP_MaxBatTemp (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 250 of file conv.c.

7.3.2.8 CONV_DISP_MinVoltage()

```
void CONV_DISP_MinVoltage (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 171 of file conv.c.

7.3.2.9 CONV_DISP_Motor_Temp()

```
void CONV_DISP_Motor_Temp (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            SIGNALS_string * outstring )
```

Definition at line 182 of file conv.c.

7.3.2.10 CONV_find_string_length()

```
uint32_t CONV_find_string_length (
            uint8_t * str,
            uint32_t strlen )
```

Definition at line 295 of file conv.c.

7.3.2.11 CONV_FSG_AMI_state()

```
uint32_t CONV_FSG_AMI_state (
            uint8_t * data )
```

Definition at line 41 of file conv.c.

7.3.2.12 CONV_InverterTemp_FL()

```
float CONV_InverterTemp_FL (
            uint8_t * data )
```

Definition at line 46 of file conv.c.

7.3.2.13 CONV_InverterTemp_FR()

```
float CONV_InverterTemp_FR (
            uint8_t * data )
```

Definition at line 52 of file conv.c.

7.3.2.14 CONV_InverterTemp_RL()

```
float CONV_InverterTemp_RL (
            uint8_t * data )
```

Definition at line 58 of file conv.c.

7.3.2.15 CONV_InverterTemp_RR()

```
float CONV_InverterTemp_RR (
            uint8_t * data )
```

Definition at line 64 of file conv.c.

7.3.2.16 CONV_LapTime()

```
float CONV_LapTime (
            uint8_t * data )
```

Definition at line 31 of file conv.c.

7.3.2.17 CONV_LastLapTime()

```
float CONV_LastLapTime (
            signals_signal * signal_list,
            uint32_t signal_list_len )
```

Definition at line 98 of file conv.c.

7.3.2.18 CONV_max()

```
float CONV_max (
            float * vals,
            uint32_t valcount )
```

Definition at line 316 of file conv.c.

7.3.2.19 CONV_MaxBatTemp()

```
float CONV_MaxBatTemp (
            uint8_t * data )
```

Definition at line 26 of file conv.c.

7.3.2.20 CONV_MaxInverterTemp()

```
float CONV_MaxInverterTemp (
            signals_signal * signal_list,
            uint32_t signal_list_len )
```

Definition at line 116 of file conv.c.

7.3.2.21 CONV_MaxMotorTemp()

```
float CONV_MaxMotorTemp (
            signals_signal * signal_list,
            uint32_t signal_list_len )
```

Definition at line 142 of file conv.c.

7.3.2.22 CONV_min()

```
float CONV_min (
            float * vals,
            uint32_t valcount )
```

Definition at line 306 of file conv.c.

7.3.2.23 CONV_MinBatVoltage()

```
float CONV_MinBatVoltage (
            uint8_t * data )
```

Definition at line 20 of file conv.c.

7.3.2.24 CONV_MotorTemp_FL()

```
float CONV_MotorTemp_FL (
            uint8_t * data )
```

Definition at line 82 of file conv.c.

7.3.2.25 CONV_MotorTemp_FR()

```
float CONV_MotorTemp_FR (
            uint8_t * data )
```

Definition at line 88 of file conv.c.

7.3.2.26 CONV_MotorTemp_RL()

```
float CONV_MotorTemp_RL (
            uint8_t * data )
```

Definition at line 76 of file conv.c.

7.3.2.27 CONV_MotorTemp_RR()

```
float CONV_MotorTemp_RR (
            uint8_t * data )
```

Definition at line 70 of file conv.c.

7.4 DELAY

Files

- file delay.h

  This File defines the Prototypes for DELAY.

Macros

- #define CLOCK_FREQUENCY_CORE 120000000

  The Frequency of the Core in Hz.
- #define MICROSECONDS_IN_SECOND 1000000

  Count of Microseconds in a Second.
- #define MILLISECONDS_IN_SECOND 1000

  Count of Milliseconds in a Second.
- #define TWO_STEPS_DELAY_ADJ 2

Functions

- void DELAY_Milliseconds (uint32_t delay)

  Delay for an amount of Milliseconds.
- void DELAY_Microseconds (uint32_t delay)

  Delay for an amount of Microseconds.

7.4.1 Detailed Description

This Module implements Functions to sleep/delay for a specified Amount of Time.

7.4.2 Macro Definition Documentation

7.4.2.1 CLOCK_FREQUENCY_CORE

```
#define CLOCK_FREQUENCY_CORE 120000000
```

The Frequency of the Core in Hz.

Definition at line 33 of file delay.h.

7.4.2.2 MICROSECONDS_IN_SECOND

```
#define MICROSECONDS_IN_SECOND 1000000
```

Count of Microseconds in a Second.

Definition at line 38 of file delay.h.

7.4.2.3 MILLISECONDS_IN_SECOND

```
#define MILLISECONDS_IN_SECOND 1000
```

Count of Milliseconds in a Second.

Definition at line 43 of file delay.h.

7.4.2.4 TWO_STEPS_DELAY_ADJ

```
#define TWO_STEPS_DELAY_ADJ 2
```

Adjustment of the Delay Functions

Definition at line 48 of file delay.h.

7.4.3 Function Documentation

7.4.3.1 DELAY_Microseconds()

```
void DELAY_Microseconds (
            uint32_t delay )
```

Delay for an amount of Microseconds.

Parameters

delay	The Time to Delay in Microseconds

Definition at line 10 of file delay.c.

References CLOCK_FREQUENCY_CORE, MICROSECONDS_IN_SECOND, and TWO_STEPS_DELAY_ADJ.

7.4.3.2 DELAY_Milliseconds()

```
void DELAY_Milliseconds (
            uint32_t delay )
```

Delay for an amount of Milliseconds.

Parameters

delay	The Time to Delay in Milliseconds

Definition at line 3 of file delay.c.

References DELAY_Microseconds(), and MILLISECONDS_IN_SECOND.

7.5 SHORTPROTOCOL

Files

- file shortprotocol.h

  This File defines the Prototypes for SHORTPROTOCOL.
- file shortprotocol.c

  This File implements the Prototypes for SHORTPROTOCOL.

Data Structures

- struct SHORTPROTOCOL_string

  Struct to combine the length of a String with the String and the Overhead Bytes used by the Shortprotocol in one Data type.
- struct SHORTPROTOCOL_Instance

  Configuration Struct for SHORTPROTOCOL passed to the SHORTPROTOCOL Functions.

Macros

- #define SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH 100

  Maximum Command length, thus maximum payload length.
- #define SHORTPROTOCOL_BEGIN 0x13

  The Begining Byte to indicate a Transmission.
- #define SHORTPROTOCOL_OVERHEAD_BYTES 5

  The Overhead of sending Data with the Shortprotocol.
- #define SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES 3

  The Overhead of sending Data with the Shortprotocol, not counting the two CRC Bytes.
- #define SHORTPROTOCOL_FIRST_BYTE_MASK 0x00FF

  Mask to convert endianess (first Byte is on second place)
- #define SHORTPROTOCOL_SECOND_BYTE_MASK 0xFF00

  Mask to convert endianess (second Byte in on first place)
- #define SHORTPROTOCOL_BYTE_LENGTH 8

  The length of one Byte for endianess conversion.
- #define SHORTPROTOCOL_BEGIN_OFFSET 0

  The Offset of the Begin Byte in the Shortprotocol.
- #define SHORTPROTOCOL_LENGTH_LSB_OFFSET 1

  The Offset of the Least Significant Byte of the Length in the Shortprotocol.
- #define SHORTPROTOCOL_LENGTH_MSB_OFFSET 2

  The Offset of the Most Significant Byte of the Length in the Shortprotocol.
- #define SHORTPROTOCOL_COMMAND_OFFSET 3

  The Offset of the Command, thus the Payload in the Shortprotocol.
- #define SHORTPROTOCOL_CRC_LSB_OFFSET 0

  The Offset of the CRC Least Significant Byte from the Back of the Package.
- #define SHORTPROTOCOL_CRC_MSB_OFFSET 1

  The Offset of the CRC Most Significant Byte from the Back of the Package.

Enumerations

- enum SHORTPROTOCOL_status { **SHORTPROTOCOL_SUCCESS** = 0 , **SHORTPROTOCOL_ERROR** , **SHORTPROTOCOL_AVAILABLE** , **SHORTPROTOCOL_NOT_AVAILABLE** }

  Return Status used by both the Callback Functions of the SHORTPROTOCOL and the Functions.

Functions

- SHORTPROTOCOL_status SHORTPROTOCOL_Send (SHORTPROTOCOL_Instance *inst, uint8_t *data, uint32_t length)

  Function to try to send a new Command using the SHORTPROTOCOL.
- void SHORTPROTOCOL_Update (SHORTPROTOCOL_Instance *inst)

  Function to be called repeadetly in the Main Loop to send Pieces of maximumPackageLength of the Package generated by SHORTPROTOCOL_Send.
- SHORTPROTOCOL_status SHORTPROTOCOL_Available (SHORTPROTOCOL_Instance *inst)

  Returns if a new Command can be sent using the Shortprotocol.
- void SHORTPROTOCOL_Initialize (SHORTPROTOCOL_Instance *inst)

  Initializes an Empty SHORTPROTOCOL_Instance. Callback Functions still have to be added/defined and the maximumPackageLength has to be set.

7.5.1 Detailed Description

This Module handles packing the Commands generated by the SIGNAL Module and sending them using the Shortprotocol defined in the Displays (EA uniTFTs035-ATC) Datasheet. To not block the Main Loop for too long, the Packet is split into smaller Parts defined by SHORTPROTOCOL_Instance.maximumPackageLength. The Module is written in a general Fashion and can be adopted to use SPI, for this the Callback Functions in SHORTPROTOCOL_Instance have to be redefined for SPI.

7.5.2 Macro Definition Documentation

7.5.2.1 SHORTPROTOCOL_BEGIN

```
#define SHORTPROTOCOL_BEGIN 0x13
```

The Begining Byte to indicate a Transmission.

Definition at line 43 of file shortprotocol.h.

7.5.2.2 SHORTPROTOCOL_BEGIN_OFFSET

```
#define SHORTPROTOCOL_BEGIN_OFFSET 0
```

The Offset of the Begin Byte in the Shortprotocol.

Definition at line 74 of file shortprotocol.h.

7.5.2.3 SHORTPROTOCOL_BYTE_LENGTH

```
#define SHORTPROTOCOL_BYTE_LENGTH 8
```

The length of one Byte for endianess conversion.

Definition at line 69 of file shortprotocol.h.

7.5.2.4 SHORTPROTOCOL_COMMAND_OFFSET

```
#define SHORTPROTOCOL_COMMAND_OFFSET 3
```

The Offset of the Command, thus the Payload in the Shortprotocol.

Definition at line 92 of file shortprotocol.h.

7.5.2.5 SHORTPROTOCOL_CRC_LSB_OFFSET

```
#define SHORTPROTOCOL_CRC_LSB_OFFSET 0
```

The Offset of the CRC Least Significant Byte from the Back of the Package.

Definition at line 98 of file shortprotocol.h.

7.5.2.6 SHORTPROTOCOL_CRC_MSB_OFFSET

```
#define SHORTPROTOCOL_CRC_MSB_OFFSET 1
```

The Offset of the CRC Most Significant Byte from the Back of the Package.

Definition at line 104 of file shortprotocol.h.

7.5.2.7 SHORTPROTOCOL_FIRST_BYTE_MASK

```
#define SHORTPROTOCOL_FIRST_BYTE_MASK 0x00FF
```

Mask to convert endianess (first Byte is on second place)

Definition at line 59 of file shortprotocol.h.

7.5.2.8 SHORTPROTOCOL_LENGTH_LSB_OFFSET

```
#define SHORTPROTOCOL_LENGTH_LSB_OFFSET 1
```

The Offset of the Least Significant Byte of the Length in the Shortprotocol.

Definition at line 80 of file shortprotocol.h.

7.5.2.9 SHORTPROTOCOL_LENGTH_MSB_OFFSET

```
#define SHORTPROTOCOL_LENGTH_MSB_OFFSET 2
```

The Offset of the Most Significant Byte of the Length in the Shortprotocol.

Definition at line 87 of file shortprotocol.h.

7.5.2.10 SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH

```
#define SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH 100
```

Maximum Command length, thus maximum payload length.

Definition at line 38 of file shortprotocol.h.

7.5.2.11 SHORTPROTOCOL_OVERHEAD_BYTES

```
#define SHORTPROTOCOL_OVERHEAD_BYTES 5
```

The Overhead of sending Data with the Shortprotocol.

Definition at line 48 of file shortprotocol.h.

7.5.2.12 SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES

```
#define SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES 3
```

The Overhead of sending Data with the Shortprotocol, not counting the two CRC Bytes.

Definition at line 54 of file shortprotocol.h.

7.5.2.13 SHORTPROTOCOL_SECOND_BYTE_MASK

```
#define SHORTPROTOCOL_SECOND_BYTE_MASK 0xFF00
```

Mask to convert endianess (second Byte in on first place)

Definition at line 64 of file shortprotocol.h.

7.5.3 Enumeration Type Documentation

7.5.3.1 SHORTPROTOCOL_status

```
enum SHORTPROTOCOL_status
```

Return Status used by both the Callback Functions of the SHORTPROTOCOL and the Functions.

Definition at line 110 of file shortprotocol.h.

7.5.4 Function Documentation

7.5.4.1 SHORTPROTOCOL_Available()

```
SHORTPROTOCOL_status SHORTPROTOCOL_Available (
            SHORTPROTOCOL_Instance * inst )
```

Returns if a new Command can be sent using the Shortprotocol.

Parameters

inst	SHORTPROTOCOL_Instance

Returns

SHORTPROTOCOL_AVAILABLE if a new Command can be written using the SHORTPROTOCOL_Send Function, SHORTPROTOCOL_NOT_AVAILABLE if the last Command is not yet sent.

Definition at line 104 of file shortprotocol.c.

References SHORTPROTOCOL_Instance::newCommand.

7.5.4.2 SHORTPROTOCOL_Initialize()

```
void SHORTPROTOCOL_Initialize (
            SHORTPROTOCOL_Instance * inst )
```

Initializes an Empty SHORTPROTOCOL_Instance. Callback Functions still have to be added/defined and the maximumPackageLength has to be set.

Parameters

inst	SHORTPROTOCOL_Instance

Definition at line 108 of file shortprotocol.c.

References SHORTPROTOCOL_Instance::newCommand, and SHORTPROTOCOL_Instance::writeCounter.

7.5.4.3 SHORTPROTOCOL_Send()

```
SHORTPROTOCOL_status SHORTPROTOCOL_Send (
            SHORTPROTOCOL_Instance * inst,
            uint8_t * data,
            uint32_t length )
```

Function to try to send a new Command using the SHORTPROTOCOL.

Parameters

inst	SHORTPROTOCOL_Instance
data	The Data to be sent using the Shortprotocol
length	The Length of the Payload (Data) to be sent (without NULL termination)

Returns

SHORTPROTOCOL_NOT_AVAILABLE when the last Command was not yet sent, SHORTPROTOCOL_↩ ERROR when the Command has a length of zero, SHORTPROTOCOL_SUCCESS when the Command was successfully queued to be sent.

Definition at line 17 of file shortprotocol.c.

References SHORTPROTOCOL_Instance::newCommand.

7.5.4.4 SHORTPROTOCOL_Update()

```
void SHORTPROTOCOL_Update (
            SHORTPROTOCOL_Instance * inst )
```

Function to be called repeadetly in the Main Loop to send Pieces of maximumPackageLength of the Package generated by SHORTPROTOCOL_Send.

Parameters

inst	SHORTPROTOCOL_Instance

Definition at line 78 of file shortprotocol.c.

References SHORTPROTOCOL_Instance::newCommand.

7.6 SIGNALS

Files

- file signals.h

  This File defines the Prototypes for SIGNALS.
- file signals.c

  This File implements 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)

7.6.1 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.

7.6.2 Macro Definition Documentation

7.6.2.1 SIGNALS_MAXIMUM_NAME_LENGTH

```
#define SIGNALS_MAXIMUM_NAME_LENGTH 30
```

The maximum friendly name length for a signal.

Definition at line 41 of file signals.h.

7.6.2.2 SIGNALS_STRING_MAXIMUM_LENGTH

```
#define SIGNALS_STRING_MAXIMUM_LENGTH 50
```

The maximum length of the string value of a signal.

Definition at line 46 of file signals.h.

7.6.3 Typedef Documentation

7.6.3.1 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.

Definition at line 105 of file signals.h.

7.6.4 Enumeration Type Documentation

7.6.4.1 signals_data_type

```
enum signals_data_type
```

The Data type of a Signal.

Definition at line 77 of file signals.h.

7.6.4.2 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

Definition at line 51 of file signals.h.

7.6.4.3 signals_signal_type

```
enum signals_signal_type
```

The Signal type of a Signal.

Definition at line 86 of file signals.h.

7.6.5 Function Documentation

7.6.5.1 signals_compare_names()

```
signals_result signals_compare_names (
            uint8_t * first,
            uint32_t firstlen,
            uint8_t * second,
            uint32_t secondlen )
```

Function to compare two Signals Names.

Parameters

first	First Name
firstlen	Length of `first`
second	Second Name
secondlen	Length of `second`

Returns

SIGNALS_MATCH if the Names match, and SIGNALS_NO_MATCH if they dont

Definition at line 100 of file signals.c.

References SIGNALS_MATCH, and SIGNALS_NO_MATCH.

7.6.5.2 signals_find_data()

```
signals_result signals_find_data (
            uint32_t id,
            uint8_t interface,
```

```
cancomm_message * message_list,
uint32_t message_list_len,
uint8_t * data )
```

This function finds the data from a CAN Message and writes it to the Buffer `data`

Parameters

id	The CAN Message ID
interface	The CAN Interface (wich uniquely identifies a CAN Interface)
message_list	List of cancomm_message Structs with the data in it
message_list_len	The Length of `message_list`
data	The Buffer the Data of the requested Message gets written to

Returns

A signals_result. Returns SIGNALS_FOUND if the Data of the Message was found, and SIGNALS_NOT_↩ FOUND if the Message is not in the List.

Definition at line 82 of file signals.c.

7.6.5.3 signals_find_display_signal()

```
signals_signal * signals_find_display_signal (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            uint32_t * dispSignalCount,
            uint32_t needle )
```

Function to return the first, second, third ... display Signal. If needle is for example five, the fifth display Signal is returned. Also, the total Count of Display Signals is written to `dispSignalCount`.

Parameters

signal_list	The signal_list to be searched in.
signal_list_len	The Length of `signal_list`
dispSignalCount	The total Count of Display Signals in `signal_list` is written to this Pointer.
needle	The Display Signal to return

Returns

Pointer to the display Signal in place of `needle`

Definition at line 148 of file signals.c.

7.6.5.4 signals_find_signal()

```
signals_signal * signals_find_signal (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            void(*)(void) Callback )
```

This function finds a singnal from signal_list by its Callback. This is needed for the Conversion Functions of Internal Signals, as they need to read other Signal Values in thier Conversion Function.

Parameters

signal_list	List of signals_signal to be searched in
signal_list_len	The Length of `signal_list`
Callback	The Callback Function of the searched signal

Returns

Pointer to the Signal if it is found, else NULL is returned

Definition at line 127 of file signals.c.

7.6.5.5 SIGNALS_Interpret()

```
void SIGNALS_Interpret (
            signals_signal * signal_list,
            uint32_t signal_list_len,
            cancomm_message * message_list,
            uint32_t message_list_len )
```

This Function takes a signal_list and a message_list and interprets each Signal either from other Signals or from a CAN message. The Callback Functions defined in each Signal are called here.

Parameters

signal_list	A List of signals_signal Structs
signal_list_len	The Length of `signal_list`
message_list	A List of cancomm_message Structs
message_list_len	The Length of `signal_list`

Definition at line 16 of file signals.c.

7.7 UART

Files

- file uart.h

  This File defines the Prototypes for UART.
- file uart.c

  This File Implements the Prototypes for UART.

Functions

- SHORTPROTOCOL_status UART_ReadAvailable (void)
- uint8_t UART_ReadByte (void)
- SHORTPROTOCOL_status UART_WriteAvailable (void)
- void UART_WriteByte (uint8_t byte)

7.7.1 Detailed Description

This Module defines the Interface Functions used by SHORTPROTOCOL.

7.7.2 Function Documentation

7.7.2.1 UART_ReadAvailable()

```
SHORTPROTOCOL_status UART_ReadAvailable (
            void  )
```

Returns wether or not a Byte can be read from UART

Returns

SHORTPROTOCOL_AVAILABLE if a Byte can be read from UART and SHORTPROTOCOL_NOT_↩ AVAILABLE if no Byte can be read from UART

Definition at line 17 of file uart.c.

7.7.2.2 UART_ReadByte()

```
uint8_t UART_ReadByte (
            void  )
```

Reads one Byte from UART

Returns

The Byte read from UART

Definition at line 25 of file uart.c.

7.7.2.3 UART_WriteAvailable()

```
SHORTPROTOCOL_status UART_WriteAvailable (
            void  )
```

Returns wether or not a Byte can be written to UART

Returns

SHORTPROTOCOL_AVAILABLE if a Byte can be written to UART and SHORTPROTOCOL_NOT_↩ AVAILABLE if no Byte can be written to UART

Definition at line 29 of file uart.c.

7.7.2.4 UART_WriteByte()

```
void UART_WriteByte (
            uint8_t byte )
```

Writes one Byte to UART

Definition at line 37 of file uart.c.

Chapter 8

Data Structure Documentation

8.1 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.

8.1.1 Detailed Description

Structure defining a CAN Interface.

Definition at line 52 of file cancomm.h.

8.1.2 Field Documentation

8.1.2.1 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

Definition at line 105 of file cancomm.h.

8.1.2.2 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

Definition at line 83 of file cancomm.h.

8.1.2.3 number

```
uint8_t cancomm_interface::number
```

The Interface Number (Has to be unique)

Definition at line 56 of file cancomm.h.

8.1.2.4 receiveFifo

```
uint8_t cancomm_interface::receiveFifo
```

The FIFO Number used for Recieving Frames.

Definition at line 61 of file cancomm.h.

8.1.2.5 transmitFifo

```
uint8_t cancomm_interface::transmitFifo
```

The FIFO Number used for Transmitting Frames.

Definition at line 66 of file cancomm.h.

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

- cancomm.h

8.2 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.

8.2.1 Detailed Description

Structure defining a CAN Message.

Definition at line 115 of file cancomm.h.

8.2.2 Field Documentation

8.2.2.1 data

`uint8_t cancomm_message::data[CANCOMM_MAXIMUM_DATA_LENGTH]`

The Data of the Message.

Definition at line 134 of file cancomm.h.

8.2.2.2 friendly_name

`uint8_t cancomm_message::friendly_name[CANCOMM_MAXIMUM_NAME_LENGTH]`

A friendly Human Readable Name for the Message.

Definition at line 144 of file cancomm.h.

8.2.2.3 id

`uint32_t cancomm_message::id`

The ID of the Message.

Definition at line 119 of file cancomm.h.

8.2.2.4 interface_number

`uint8_t cancomm_message::interface_number`

The interface number (wich has to be unique to a CAN Interface)

Definition at line 124 of file cancomm.h.

8.2.2.5 length

`uint8_t cancomm_message::length`

The DLC of the Message.

Definition at line 129 of file cancomm.h.

8.2.2.6 timestamp

```
uint32_t cancomm_message::timestamp
```

The timestamp when the Message was recieved.

Definition at line 139 of file cancomm.h.

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

- cancomm.h

8.3 SHORTPROTOCOL_Instance Struct Reference

Configuration Struct for SHORTPROTOCOL passed to the SHORTPROTOCOL Functions.

```
#include <shortprotocol.h>
```

Data Fields

- uint8_t(∗ readByte)(void)

  Callback to read one Byte from the Serial Interface.
- SHORTPROTOCOL_status(∗ readAvailable)(void)

  Callback to check if a Byte is available to be read. If this returns SHORTPROTOCOL_AVAILABLE at least one Byte has to be readable from readByte.
- void(∗ writeByte)(uint8_t)

  Callback to write one Byte to the Serial Interface.
- SHORTPROTOCOL_status(∗ writeAvailable)(void)

  Callbakc to check if a Byte is writeable to the Serial Interface If this returns SHORTPROTOCOL_AVAILABLE at leat one Byte has to be writeable to write Byte.
- uint32_t maximumPackageLength

  The maximum Package Length Packets longer than this will be split into Packets with the maximum Size of maximumPackageLength.
- SHORTPROTOCOL_string command_buffer

  A Buffer where the Command wich is currently sent resides.
- SHORTPROTOCOL_status newCommand

  Status of the SHORTPROTOCOL Instance itself.
- uint32_t writeCounter

  Counter to keep track of what Parts of a Package were already sent when it is split to maximumPackageLength.

8.3.1 Detailed Description

Configuration Struct for SHORTPROTOCOL passed to the SHORTPROTOCOL Functions.

The Callback functions have to all be defined as described in thier Documentaion, also a maximumPackageLength has to be defined.

Definition at line 134 of file shortprotocol.h.

8.3.2 Field Documentation

8.3.2.1 command_buffer

```
SHORTPROTOCOL_string SHORTPROTOCOL_Instance::command_buffer
```

A Buffer where the Command wich is currently sent resides.

Definition at line 169 of file shortprotocol.h.

8.3.2.2 maximumPackageLength

```
uint32_t SHORTPROTOCOL_Instance::maximumPackageLength
```

The maximum Package Length Packets longer than this will be split into Packets with the maximum Size of maximumPackageLength.

Definition at line 164 of file shortprotocol.h.

8.3.2.3 newCommand

```
SHORTPROTOCOL_status SHORTPROTOCOL_Instance::newCommand
```

Status of the SHORTPROTOCOL Instance itself.

Definition at line 175 of file shortprotocol.h.

8.3.2.4 readAvailable

```
SHORTPROTOCOL_status(* SHORTPROTOCOL_Instance::readAvailable) (void)
```

Callback to check if a Byte is available to be read. If this returns SHORTPROTOCOL_AVAILABLE at least one Byte has to be readable from readByte.

Definition at line 145 of file shortprotocol.h.

8.3.2.5 readByte

```
uint8_t(* SHORTPROTOCOL_Instance::readByte) (void)
```

Callback to read one Byte from the Serial Interface.

Definition at line 138 of file shortprotocol.h.

8.3.2.6 writeAvailable

```
SHORTPROTOCOL_status(* SHORTPROTOCOL_Instance::writeAvailable) (void)
```

Callbakc to check if a Byte is writeable to the Serial Interface If this returns SHORTPROTOCOL_AVAILABLE at leat one Byte has to be writeable to write Byte.

Definition at line 157 of file shortprotocol.h.

8.3.2.7 writeByte

```
void(* SHORTPROTOCOL_Instance::writeByte) (uint8_t)
```

Callback to write one Byte to the Serial Interface.

Definition at line 150 of file shortprotocol.h.

8.3.2.8 writeCounter

```
uint32_t SHORTPROTOCOL_Instance::writeCounter
```

Counter to keep track of what Parts of a Package were already sent when it is split to maximumPackageLength.

Definition at line 181 of file shortprotocol.h.

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

- shortprotocol.h

8.4 SHORTPROTOCOL_string Struct Reference

Struct to combine the length of a String with the String and the Overhead Bytes used by the Shortprotocol in one Data type.

```
#include <shortprotocol.h>
```

Data Fields

- uint32_t length
- uint8_t data [SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH+SHORTPROTOCOL_OVERHEAD_BYTES]

8.4.1 Detailed Description

Struct to combine the length of a String with the String and the Overhead Bytes used by the Shortprotocol in one Data type.

Definition at line 121 of file shortprotocol.h.

8.4.2 Field Documentation

8.4.2.1 data

```
uint8_t SHORTPROTOCOL_string::data[SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH+ SHORTPROTOCOL_OVERHEAD_BYTES]
```

Definition at line 123 of file shortprotocol.h.

8.4.2.2 length

```
uint32_t SHORTPROTOCOL_string::length
```

Definition at line 122 of file shortprotocol.h.

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

- shortprotocol.h

8.5 signals_signal_struct Struct Reference

```
#include <signals.h>
```

Data Fields

- uint32_t id
- uint8_t interface_number
- signals_signal_type type
- 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

8.5.1 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.

A Signal is therefore defined by its type and data_type.

Todo Combine the Different Signal Types to a Union to use less RAM

Definition at line 137 of file signals.h.

8.5.2 Field Documentation

8.5.2.1 can_convert_float

```
float(* signals_signal_struct::can_convert_float) (uint8_t *data)
```

Callback Function for type=SIGNALS_CAN_MESSAGE and data_type=SIGNALS_FLOAT_SIGNAL

Definition at line 152 of file signals.h.

8.5.2.2 can_convert_string

```
void(* signals_signal_struct::can_convert_string) (uint8_t *data, SIGNALS_string *string)
```

Callback Function for type=SIGNALS_CAN_MESSAGE and data_type=SIGNALS_STRING_SIGNAL

Definition at line 178 of file signals.h.

8.5.2.3 can_convert_uint32_t

```
uint32_t(* signals_signal_struct::can_convert_uint32_t) (uint8_t *data)
```

Callback Function for type=SIGNALS_CAN_MESSAGE and data_type=SIGNALS_UINT32_T_SIGNAL

Definition at line 165 of file signals.h.

8.5.2.4 data_type

```
signals_data_type signals_signal_struct::data_type
```

Data Type of the Signal

Definition at line 148 of file signals.h.

8.5.2.5 friendly_name

```
uint8_t signals_signal_struct::friendly_name[SIGNALS_MAXIMUM_NAME_LENGTH]
```

A friendly Name of the Signal, only used for Debugging

Definition at line 190 of file signals.h.

8.5.2.6 id

```
uint32_t signals_signal_struct::id
```

The ID of the CAN Message

Definition at line 139 of file signals.h.

8.5.2.7 interface_number

```
uint8_t signals_signal_struct::interface_number
```

The CAN Interface Number (unique Identifier for a CAN Interface)

Definition at line 142 of file signals.h.

8.5.2.8 internal_convert_float

```
float(* signals_signal_struct::internal_convert_float) (signals_signal *signal_list, uint32_t
signal_list_len)
```

Callback Function for type=SIGNALS_INTERNAL_SIGNAL and data_type=SIGNALS_FLOAT_SIGNAL

Definition at line 156 of file signals.h.

8.5.2.9 internal_convert_string

```
void(* signals_signal_struct::internal_convert_string) (signals_signal *signal_list, uint32_t
signal_list_len, SIGNALS_string *string)
```

Callback Function for type=SIGNALS_INTERNAL_SIGNAL or type=SIGNALS_DISPLAY_SIGNAL. and data_type SIGNALS_STRING_SIGNAL

Definition at line 182 of file signals.h.

8.5.2.10 internal_convert_uint32_t

```
uint32_t(* signals_signal_struct::internal_convert_uint32_t) (signals_signal *signal_list,
uint32_t signal_list_len)
```

Callback Function for type=SIGNALS_INTERNAL_SIGNAL and data_type=SIGNALS_UINT32_T_SIGNAL

Definition at line 169 of file signals.h.

8.5.2.11 timestamp

```
uint32_t signals_signal_struct::timestamp
```

The Timestamp of the Signal

Definition at line 193 of file signals.h.

8.5.2.12 type

```
signals_signal_type signals_signal_struct::type
```

Type of the Signal

Definition at line 145 of file signals.h.

8.5.2.13 value_float

```
float signals_signal_struct::value_float
```

The Float Value of this Signal, only used when data_type=SIGNALS_FLOAT_SIGNAL

Definition at line 161 of file signals.h.

8.5.2.14 value_string

```
SIGNALS_string signals_signal_struct::value_string
```

The String Value of this Signal, only used when data_type=SIGNALS_STRING_SIGNAL

Definition at line 187 of file signals.h.

8.5.2.15 value_uint32_t

```
uint32_t signals_signal_struct::value_uint32_t
```

The uint32_t Value of this Signal, only used when data_type=SIGNALS_UINT32_T_SIGNAL

Definition at line 174 of file signals.h.

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

- signals.h

8.6 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]

8.6.1 Detailed Description

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

Definition at line 95 of file signals.h.

8.6.2 Field Documentation

8.6.2.1 data

```
uint8_t SIGNALS_string::data[SIGNALS_STRING_MAXIMUM_LENGTH]
```

Definition at line 97 of file signals.h.

8.6.2.2 length

```
uint32_t SIGNALS_string::length
```

Definition at line 96 of file signals.h.

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

- signals.h

Chapter 9

File Documentation

9.1 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.

9.1.1 Detailed Description

This File Implements the Prototypes for CANCOMM.

Author

Frederic Emmerth

Definition in file cancomm.c.

9.2 cancomm.c

Go to the documentation of this file.

```
00001 /**
00002  * @file cancomm.c
00003  * @brief This File Implements the Prototypes for \ref CANCOMM
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup CANCOMM
00008  *
00009  * \addtogroup CANCOMM
00010  * \{
00011  *
00012  */
00013
00014 #include "cancomm.h"
00015
00016
00017 void CANCOMM_ReadMessages(cancomm_message* message_list,
00018         uint32_t message_list_len,
00019         cancomm_interface* interface_list,
00020         uint8_t interface_list_len){
00021
00022     /* Local Variables where the Frame Data can be temporarily saved */
00023     uint8_t message_recieved = 0;
00024     uint32_t temp_id;
00025     uint8_t temp_length;
00026     uint8_t temp_data[CANCOMM_MAXIMUM_DATA_LENGTH];
00027     CANFD_MSG_RX_ATTRIBUTE temp_msgAttr;
00028     uint32_t temp_timestamp;
00029
00030     /* Loop over every Interface and get the recieved Messages from the FIFOs*/
00031     for(uint8_t current_interface = 0;
00032             current_interface < interface_list_len;
00033             current_interface ++)
00034         {
00035
00036         /* Endless Loop, broken internally when there is no data in the fifo */
00037         while(1){
00038
00039             /* Try to get a Message from FIFO */
00040             message_recieved = interface_list[current_interface].MessageReceive(
00041                     &temp_id,
00042                     &temp_length,
00043                     temp_data,
00044                     &temp_timestamp,
00045                     interface_list[current_interface].receiveFifo,
00046                     &temp_msgAttr);
00047
00048             /* If there was a Message in the FIFO interpret it */
00049             if(message_recieved){
00050                 /* See if the Message is in cancomm_message_list */
00051                 for(uint32_t i=0; i<message_list_len; i++){
00052                     /* If ID and Interface match, copy the data into the message_list*/
00053                     if((message_list[i].id == temp_id)&&
00054                             (message_list[i].interface_number ==
00055                             interface_list[current_interface].number))
00056                     {
00057                         for(uint8_t j=0; j<CANCOMM_MAXIMUM_DATA_LENGTH; j++){
00058                             message_list[i].data[j] = temp_data[j];
00059                         }
00060                         /* Break the Loop after finding the Message */
00061                         break;
00062                     }
00063                 }
00064             /* If there was no Message in the FIFO break Loop */
00065             }else{
00066                 break;
00067             }
00068         }
00069
00070     }
00071
00072 }
00073
00074 /**
00075  * \}
00076  */
```

9.3 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.

9.3.1 Detailed Description

This File defines the Prototypes for CANCOMM.

Author

Frederic Emmerth

Definition in file cancomm.h.

9.4 cancomm.h

Go to the documentation of this file.

```
00001 /**
00002  * @file cancomm.h
00003  * @brief This File defines the Prototypes for \ref CANCOMM
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup CANCOMM
00008  *
00009  */
00010
00011
00012
00013 /**
00014  * \defgroup CANCOMM CANCOMM
00015  * This Module is configured using a list of \ref cancomm_interface Structs.
00016  * These provide the Recieve and Transmit Functions for the CAN Interface. Which
00017  * Messages should be recieved and transmitted is configured using a list of
00018  * \ref cancomm_message Structs.
00019  *
00020  * \todo Implement Transmit Functionality
00021  * \todo Handle Remote Frames on Recieve
00022  * \todo Handle CAN FD Frames on Recieve
00023  * \bug Only Interface One was tested
00024  *
00025  * \addtogroup CANCOMM
00026  * \{
00027  */
00028
00029 #ifndef CANCOMM_H
00030 #define CANCOMM_H
00031
00032 #ifdef  __cplusplus
00033 extern "C" {
00034 #endif
00035
00036 #include "definitions.h"
00037
00038 /**
00039  * @brief The Maximum Bytes of Data per CAN Frame
00040  */
00041 #define CANCOMM_MAXIMUM_DATA_LENGTH     8
00042
00043 /**
00044  * @brief The Maximum Friendly Name Length of the CAN Messages
00045  */
00046 #define CANCOMM_MAXIMUM_NAME_LENGTH     30
00047
00048
00049 /**
00050  * @brief Structure defining a CAN Interface
00051  */
00052 typedef struct{
00053     /**
00054      * @brief The Interface Number (Has to be unique)
00055      */
00056     uint8_t number;
00057
00058     /**
00059      * @brief The FIFO Number used for Recieving Frames
00060      */
00061     uint8_t receiveFifo;
00062
00063     /**
00064      * @brief The FIFO Number used for Transmitting Frames
00065      */
00066     uint8_t transmitFifo;
00067
00068     /**
00069      * @brief Callback Function to Transmit CAN Messages
00070      *
00071      * This Function matches the Protoype generated by MPLAB Harmony, so this
00072      * Function Pointer can be set the the Functions generated by Harmony.
00073      *
00074      * \param id            The ID of the Message
00075      * \param length        The DLC of the Message
00076      * \param data          The Data Bytes the Message should have
00077      * \param fifoQueueNum  The Number of the transmit FIFO
00078      * \param mode          For non CAN-FD Messages this is CANFD_MODE_NORMAL
00079      * \param msgAttr       For a Data Frame this is CANFD_MSG_TX_DATA_FRAME
00080      *
00081      * \return True if the Message was sucessfully added to the transmit FIFO
00082      */
```

```
00083     bool (*MessageTransmit)(uint32_t id,
00084             uint8_t length, uint8_t* data,
00085             uint8_t fifoQueueNum, CANFD_MODE mode,
00086             CANFD_MSG_TX_ATTRIBUTE msgAttr);
00087 
00088     /**
00089      * @brief Callback Function to Recieve CAN Messages
00090      *
00091      * This Function matches the Protoype generated by MPLAB Harmony, so this
00092      * Function Pointer can be set the the Functions generated by Harmony.
00093      *
00094      * \param id            The ID of the Recieved Message
00095      * \param length        The DLC of the Recieved Message
00096      * \param data          The Data of the Recieved Message
00097      * \param timestamp     The Timestamp of the Recieved Message
00098      * \param fifoNum       The FIFO used for recieving Messages
00099      * \param msgAttr       If the recieved Message was a Data Frame, this is
00100      *                      CANFD_MSG_TX_DATA_FRAME
00101      *
00102      * \returns             False if there is no Message in the Recieve FIFO left
00103      *
00104      */
00105     bool (*MessageReceive)(uint32_t *id,
00106             uint8_t *length, uint8_t *data,
00107             uint32_t *timestamp, uint8_t fifoNum,
00108             CANFD_MSG_RX_ATTRIBUTE *msgAttr);
00109 }cancomm_interface;
00110 
00111 
00112 /**
00113  * @brief Structure defining a CAN Message
00114  */
00115 typedef struct{
00116     /**
00117      * @brief The ID of the Message
00118      */
00119     uint32_t id;
00120 
00121     /**
00122      * @brief The interface number (wich has to be unique to a CAN Interface)
00123      */
00124     uint8_t interface_number;
00125 
00126     /**
00127      * @brief The DLC of the Message
00128      */
00129     uint8_t length;
00130 
00131     /**
00132      * @brief The Data of the Message
00133      */
00134     uint8_t data [CANCOMM_MAXIMUM_DATA_LENGTH];
00135 
00136     /**
00137      * @brief The timestamp when the Message was recieved
00138      */
00139     uint32_t timestamp;
00140 
00141     /**
00142      * @brief A friendly Human Readable Name for the Message
00143      */
00144     uint8_t friendly_name [CANCOMM_MAXIMUM_NAME_LENGTH];
00145 }cancomm_message;
00146 
00147 
00148 /**
00149  * @brief This function recieves the CAN Messages defined in a message_list
00150  * using the provided interface_list.
00151  * @param message_list  List of \ref cancom_message Structs
00152  * @param message_list_len  Length of \p message_list
00153  * @param interface_list    List of \ref cancomm_interface Structs
00154  * @param interface_list_len Length of \p interface_list
00155  */
00156 void CANCOMM_ReadMessages(cancomm_message* message_list,
00157         uint32_t message_list_len,
00158         cancomm_interface* interface_list,
00159         uint8_t interface_list_len);
00160 
00161 #ifdef  __cplusplus
00162 }
00163 #endif
00164 
00165 #endif  /* CANCOMM_H */
00166 
00167 /**
00168  * \}
00169  */
```

9.5 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.

9.5.1 Detailed Description

This File implements the Prototypes for COMMAND.

Author

Frederic Emmerth

Definition in file command.c.

9.6 command.c

Go to the documentation of this file.

```
00001 /**
00002  * @file command.c
00003  * @brief This File implements the Prototypes for \ref COMMAND
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup COMMAND
00008  * \addtogroup COMMAND
00009  * \{
00010  *
00011  */
00012
00013
00014
00015 #include "command.h"
00016
00017 void COMMAND_Generate(signals_signal* signal_list, uint32_t signal_list_len,
00018         uint32_t* current_display_signal, SHORTPROTOCOL_Instance* shortProt){
00019
00020     uint32_t display_signal_count;
00021
00022     signals_signal* current_signal;
00023
00024     /* get the count of display signals and if there is one, the current display signal*/
00025     current_signal = signals_find_display_signal(signal_list, signal_list_len,
00026             &display_signal_count, *current_display_signal);
00027
00028     /* Only add a new Command to be sent if the last one was sent */
00029     if(SHORTPROTOCOL_Available(shortProt)==SHORTPROTOCOL_AVAILABLE){
00030
00031         /* Set the Command to be next transmitted by the Shortprotocol */
00032         SHORTPROTOCOL_Send(shortProt, current_signal->value_string.data,
00033                 current_signal->value_string.length);
00034
00035         *current_display_signal = *current_display_signal + 1;
00036
00037         if(*current_display_signal >= display_signal_count){
00038             *current_display_signal = 0;
00039         }
00040
00041     }
00042 }
00043
00044
00045 /**
00046  * \}
00047  */
```

9.7 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.

9.7.1 Detailed Description

This File defines the Prototypes for COMMAND.

Author

Frederic Emmerth

Definition in file command.h.

9.8 command.h

Go to the documentation of this file.

```
00001 /**
00002  * @file command.h
00003  * @brief This File defines the Prototypes for \ref COMMAND
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup COMMAND
00008  *
00009  */
00010
00011 /**
00012  * \defgroup COMMAND COMMAND
00013  * This Module Picks the \ref signals_signal from a signal_list wich are of Type
00014  * SIGNALS_DISPLAY_SIGNAL and Generates a Comand to change the Text Object on
00015  * the Display with ID \ref signals_signal.oject_id to
00016  * \ref signals_signal.string_value.
00017  *
00018  * \addtogroup COMMAND
00019  * \{
00020  */
00021
00022 #ifndef COMMAND_H
00023 #define COMMAND_H
00024
00025 #ifdef  __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 #include "definitions.h"
00030 #include "signals.h"
00031 #include "shortprotocol.h"
00032
00033 /**
00034  * @brief Adds a Command from \p signal_list if \p shortProt is ready.
```

```
00035  *
00036  * \param signal_list A list of \ref signals_signal
00037  * \param signal_list_len The Length of \p signal_list
00038  * \param next_command A pointer to an Integer. This represents the next
00039  *  Command to be Sent. This value should only be changed by this Function.
00040  * \param shortProt A \ref SHORTPROTOCOL_Instance for sending Packets to the
00041  *  Display.
00042  *
00043  */
00044 void COMMAND_Generate(signals_signal* signal_list, uint32_t signal_list_len,
00045         uint32_t* next_command, SHORTPROTOCOL_Instance* shortProt);
00046
00047
00048 #ifdef  __cplusplus
00049 }
00050 #endif
00051
00052 #endif  /* COMMAND_H */
00053
00054 /**
00055  * \}
00056  */
00057
```

9.9 conv.c File Reference

This File implements the Prototypes for CONV.

```
#include "conv.h"
```

Functions

- float CONV_MinBatVoltage (uint8_t *data)
- float CONV_MaxBatTemp (uint8_t *data)
- float CONV_LapTime (uint8_t *data)
- float CONV_BestLapTime (uint8_t *data)
- uint32_t CONV_FSG_AMI_state (uint8_t *data)
- float CONV_InverterTemp_FL (uint8_t *data)
- float CONV_InverterTemp_FR (uint8_t *data)
- float CONV_InverterTemp_RL (uint8_t *data)
- float CONV_InverterTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RL (uint8_t *data)
- float CONV_MotorTemp_FL (uint8_t *data)
- float CONV_MotorTemp_FR (uint8_t *data)
- float CONV_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_MaxInverterTemp (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_MaxMotorTemp (signals_signal *signal_list, uint32_t signal_list_len)
- void CONV_DISP_MinVoltage (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_Motor_Temp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_InverterTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapDelta (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)

- void CONV_DISP_MaxBatTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_FSG_AMI_State (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- uint32_t CONV_find_string_length (uint8_t *str, uint32_t strlen)
- float CONV_min (float *vals, uint32_t valcount)
- float CONV_max (float *vals, uint32_t valcount)

9.9.1 Detailed Description

This File implements the Prototypes for CONV.

Author

Frederic Emmerth

Definition in file conv.c.

9.10 conv.c

Go to the documentation of this file.

```
00001 /**
00002  * @file conv.c
00003  * @brief This File implements the Prototypes for \ref CONV
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup CONV
00008  * \addtogroup CONV
00009  * \{
00010  *
00011  */
00012
00013
00014 #include "conv.h"
00015
00016 /*##############################################################################*/
00017 /*############################## CAN Signals ###################################*/
00018 /*##############################################################################*/
00019
00020 float CONV_MinBatVoltage(uint8_t* data){
00021     uint16_t temp;
00022     temp = (data[7] << 8) | data[6];
00023     return (float)0.0001 * temp;
00024 }
00025
00026 float CONV_MaxBatTemp(uint8_t* data){
00027     int16_t temp = (data[5] << 8) | data[4];
00028     return temp * 0.01;
00029 }
00030
00031 float CONV_LapTime(uint8_t* data){
00032     uint32_t temp = ((data[2]&0x0F)<<16)|(data[1]<<8)|data[0];
00033     return temp * 0.001;
00034 }
00035
00036 float CONV_BestLapTime(uint8_t* data){
00037     uint32_t temp = (data[4]<<12)|(data[3]<<4)|((data[2]&0xF0)>>4);
00038     return temp * 0.001;
00039 }
00040
00041 uint32_t CONV_FSG_AMI_state(uint8_t* data){
00042     uint8_t state = (data[0] & 0xE0)>>5;
00043     return state;
00044 }
00045
00046 float CONV_InverterTemp_FL(uint8_t* data){
00047     int16_t temp;
00048     temp = (data[1]<<8) | data[0];
```

```
00049     return (float)temp * 0.1;
00050 }
00051
00052 float CONV_InverterTemp_FR(uint8_t* data){
00053     int16_t temp;
00054     temp = (data[3]«8) | data[2];
00055     return (float)temp * 0.1;
00056 }
00057
00058 float CONV_InverterTemp_RL(uint8_t* data){
00059     int16_t temp;
00060     temp = (data[5]«8) | data[4];
00061     return (float)temp * 0.1;
00062 }
00063
00064 float CONV_InverterTemp_RR(uint8_t* data){
00065     int16_t temp;
00066     temp = (data[7]«8) | data[6];
00067     return (float)temp * 0.1;
00068 }
00069
00070 float CONV_MotorTemp_RR(uint8_t* data){
00071     int16_t temp;
00072     temp = (data[6] | (data[7]  « 8));
00073     return (float)temp * 0.1;
00074 }
00075
00076 float CONV_MotorTemp_RL(uint8_t* data){
00077     int16_t temp;
00078     temp = (data[4] | (data[5] « 8));
00079     return (float)temp * 0.1;
00080 }
00081
00082 float CONV_MotorTemp_FL(uint8_t* data){
00083     int16_t temp;
00084     temp = (data[0] | (data[1] « 8));
00085     return (float)temp * 0.1;
00086 }
00087
00088 float CONV_MotorTemp_FR(uint8_t* data){
00089     int16_t temp;
00090     temp = (data[2] | (data[3] « 8));
00091     return (float)temp * 0.1;
00092 }
00093
00094 /*#############################################################################*/
00095 /*############################## Internal Signals ##############################*/
00096 /*#############################################################################*/
00097
00098 float CONV_LastLapTime(signals_signal* signal_list, uint32_t signal_list_len){
00099
00100     static float last_reported_value;
00101     static float last_lap;
00102
00103     float reported_lap_value = signals_find_signal(signal_list, signal_list_len,
00104             (void(*)(void))CONV_LapTime)->value_float;
00105
00106     if(reported_lap_value != last_reported_value){
00107         last_lap = last_reported_value;
00108     }
00109
00110
00111     last_reported_value = reported_lap_value;
00112
00113     return last_lap;
00114 }
00115
00116 float CONV_MaxInverterTemp(signals_signal* signal_list,
00117         uint32_t signal_list_len){
00118
00119     signals_signal* rr_temp = signals_find_signal(signal_list, signal_list_len,
00120             (void(*)(void))CONV_InverterTemp_RR);
00121
00122     signals_signal* rl_temp = signals_find_signal(signal_list, signal_list_len,
00123             (void(*)(void))CONV_InverterTemp_FL);
00124
00125     signals_signal* fr_temp = signals_find_signal(signal_list, signal_list_len,
00126             (void(*)(void))CONV_InverterTemp_FR);
00127
00128     signals_signal* fl_temp = signals_find_signal(signal_list, signal_list_len,
00129             (void(*)(void))CONV_InverterTemp_RL);
00130
00131     float temperatureValues[4] =
00132     {
00133         rr_temp->value_float,
00134         rl_temp->value_float,
00135         fr_temp->value_float,
```

```
00136         fl_temp->value_float
00137     };
00138 
00139     return CONV_max(temperatureValues, 4);
00140 }
00141 
00142 float CONV_MaxMotorTemp(signals_signal* signal_list, uint32_t signal_list_len){
00143 
00144     signals_signal* rr_temp = signals_find_signal(signal_list, signal_list_len,
00145             (void(*)(void))CONV_MotorTemp_RR);
00146 
00147     signals_signal* rl_temp = signals_find_signal(signal_list, signal_list_len,
00148             (void(*)(void))CONV_MotorTemp_FL);
00149 
00150     signals_signal* fr_temp = signals_find_signal(signal_list, signal_list_len,
00151             (void(*)(void))CONV_MotorTemp_FR);
00152 
00153     signals_signal* fl_temp = signals_find_signal(signal_list, signal_list_len,
00154             (void(*)(void))CONV_MotorTemp_RL);
00155 
00156     float temperatureValues[4] =
00157     {
00158         rr_temp->value_float,
00159         rl_temp->value_float,
00160         fr_temp->value_float,
00161         fl_temp->value_float
00162     };
00163 
00164     return CONV_max(temperatureValues, 4);
00165 }
00166 
00167 /*#############################################################################*/
00168 /*############################## Display Signals ##############################*/
00169 /*#############################################################################*/
00170 
00171 void CONV_DISP_MinVoltage(signals_signal* signal_list, uint32_t signal_list_len,
00172         SIGNALS_string* outstring){
00173     float minvoltage = signals_find_signal(signal_list,
00174         signal_list_len,(void(*)(void))CONV_MinBatVoltage)->value_float;
00175 
00176     sprintf((char*)outstring->data,"#SSC %d,\"%4.3fV\";\n",9,minvoltage);
00177 
00178     outstring->length = CONV_find_string_length(outstring->data,
00179                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00180 }
00181 
00182 void CONV_DISP_Motor_Temp(signals_signal* signal_list, uint32_t signal_list_len,
00183         SIGNALS_string* outstring){
00184 
00185     float maxtemp = signals_find_signal(signal_list,
00186         signal_list_len,(void(*)(void))CONV_MaxMotorTemp)->value_float;
00187     sprintf((char*)outstring->data,"#SSC %d,\"%4.1fC\";\n",11,maxtemp);
00188 
00189     outstring->length = CONV_find_string_length(outstring->data,
00190                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00191 
00192 }
00193 
00194 
00195 
00196 void CONV_DISP_InverterTemp(signals_signal* signal_list, uint32_t signal_list_len,
00197         SIGNALS_string* outstring){
00198 
00199     float invtemp = signals_find_signal(signal_list,
00200         signal_list_len,(void(*)(void))CONV_MaxInverterTemp)->value_float;
00201     sprintf((char*)outstring->data,"#SSC %d,\"%4.1fC\";\n",8,invtemp);
00202 
00203     outstring->length = CONV_find_string_length(outstring->data,
00204                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00205 
00206 }
00207 
00208 void CONV_DISP_LapDelta(signals_signal* signal_list, uint32_t signal_list_len,
00209         SIGNALS_string* outstring){
00210 
00211     float bestlap = signals_find_signal(signal_list, signal_list_len,
00212             (void(*)(void))CONV_BestLapTime)->value_float;
00213 
00214     float thislap = signals_find_signal(signal_list, signal_list_len,
00215             (void(*)(void))CONV_LapTime)->value_float;
00216 
00217     float timedelta = bestlap - thislap;
00218 
00219     sprintf((char*)outstring->data,"#SSC %d,\"%4.2fs\";\n",5,timedelta);
00220 
00221     outstring->length = CONV_find_string_length(outstring->data,
00222                                 SIGNALS_STRING_MAXIMUM_LENGTH);
```

```
00223
00224 }
00225
00226 void CONV_DISP_LapTime(signals_signal* signal_list, uint32_t signal_list_len,
00227         SIGNALS_string* outstring){
00228     float thislap = signals_find_signal(signal_list, signal_list_len,
00229             (void(*)(void))CONV_LapTime)->value_float;
00230
00231     sprintf((char*)outstring->data,"#SSC %d,\"%4.2fs\";\n",2,thislap);
00232
00233     outstring->length = CONV_find_string_length(outstring->data,
00234                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00235
00236 }
00237
00238 void CONV_DISP_LastLapTime(signals_signal* signal_list, uint32_t signal_list_len,
00239         SIGNALS_string* outstring){
00240     float lastlap = signals_find_signal(signal_list, signal_list_len,
00241             (void(*)(void))CONV_LastLapTime)->value_float;
00242
00243     sprintf((char*)outstring->data,"#SSC %d,\"%4.2fs\";\n",4,lastlap);
00244
00245     outstring->length = CONV_find_string_length(outstring->data,
00246                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00247
00248 }
00249
00250 void CONV_DISP_MaxBatTemp(signals_signal* signal_list, uint32_t signal_list_len,
00251         SIGNALS_string* outstring){
00252     float maxtemp = signals_find_signal(signal_list, signal_list_len,
00253             (void(*)(void))CONV_MaxBatTemp)->value_float;
00254
00255     sprintf((char*)outstring->data,"#SSC %d,\"%4.1fC\";\n",6,maxtemp);
00256
00257     outstring->length = CONV_find_string_length(outstring->data,
00258                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00259
00260 }
00261
00262 void CONV_DISP_FSG_AMI_State(signals_signal* signal_list,
00263         uint32_t signal_list_len, SIGNALS_string* outstring){
00264
00265     uint8_t statenum = signals_find_signal(signal_list, signal_list_len,
00266             (void(*)(void))CONV_FSG_AMI_state)->value_uint32_t;
00267
00268     if(statenum > 7){
00269         statenum = 8;
00270     }
00271
00272     uint8_t statenames [9][15] ={
00273         "Manual",
00274         "Acceleration",
00275         "Skidpad",
00276         "Trackdrive",
00277         "Braketest",
00278         "Inspection",
00279         "Autocross",
00280         "Selecting",
00281         "Unknown"
00282     };
00283
00284     sprintf((char*)outstring->data,"#SSC %d,\"%s\";\n",12,statenames[statenum]);
00285
00286     outstring->length = CONV_find_string_length(outstring->data,
00287                                 SIGNALS_STRING_MAXIMUM_LENGTH);
00288
00289 }
00290
00291 /*#############################################################################*/
00292 /*############################## Helpful Functions ##############################*/
00293 /*#############################################################################*/
00294
00295 uint32_t CONV_find_string_length(uint8_t* str, uint32_t strlen){
00296
00297     uint32_t i=0;
00298     for(; i<strlen; i++){
00299         if(str[i] == 0x00){
00300             break;
00301         }
00302     }
00303     return i;
00304 }
00305
00306 float CONV_min(float* vals, uint32_t valcount){
00307     float minimum = 10000;
00308     for(uint32_t i=0; i<valcount; i++){
00309         if(vals[i] < minimum){
```

```
00310             minimum = vals[i];
00311         }
00312     }
00313     return minimum;
00314 }
00315
00316 float CONV_max(float* vals, uint32_t valcount){
00317     float maximum = -10000;
00318     for(uint32_t i=0; i<valcount; i++){
00319         if(vals[i] > maximum){
00320             maximum = vals[i];
00321         }
00322     }
00323     return maximum;
00324 }
00325
00326 /**
00327  * \}
00328  */
```

9.11 conv.h File Reference

This File defines the Prototypes for CONV.

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

Functions

- float CONV_MinBatVoltage (uint8_t *data)
- float CONV_MaxBatTemp (uint8_t *data)
- float CONV_LapTime (uint8_t *data)
- float CONV_BestLapTime (uint8_t *data)
- uint32_t CONV_FSG_AMI_state (uint8_t *data)
- float **CONV_MaxMotTemp** (uint8_t *data)
- float **CONV_MaxInvTemp** (uint8_t *data)
- float CONV_InverterTemp_FL (uint8_t *data)
- float CONV_InverterTemp_FR (uint8_t *data)
- float CONV_InverterTemp_RL (uint8_t *data)
- float CONV_InverterTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RR (uint8_t *data)
- float CONV_MotorTemp_RL (uint8_t *data)
- float CONV_MotorTemp_FL (uint8_t *data)
- float CONV_MotorTemp_FR (uint8_t *data)
- float CONV_MaxMotorTemp (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len)
- float CONV_MaxInverterTemp (signals_signal *signal_list, uint32_t signal_list_len)
- void CONV_DISP_Motor_Temp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_MinVoltage (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapDelta (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_LastLapTime (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_InverterTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)

- void CONV_DISP_MaxBatTemp (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- void CONV_DISP_FSG_AMI_State (signals_signal *signal_list, uint32_t signal_list_len, SIGNALS_string *outstring)
- uint32_t CONV_find_string_length (uint8_t *str, uint32_t strlen)
- float CONV_max (float *vals, uint32_t valcount)
- float CONV_min (float *vals, uint32_t valcount)

9.11.1 Detailed Description

This File defines the Prototypes for CONV.

Author

Frederic Emmerth

Definition in file conv.h.

9.12 conv.h

Go to the documentation of this file.

```
00001 /**
00002  * @file conv.h
00003  * @brief This File defines the Prototypes for \ref CONV
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup CONV
00008  *
00009  */
00010
00011 /**
00012  * \defgroup CONV CONV
00013  * In this Module the Callback Functions for the Signals from the
00014  * \ref SIGNALS Module are defined.
00015  *
00016  * There are Callback Functions for CAN Message Interpretation, for Internal
00017  * Message Processing and Display Command Generation.
00018  *
00019  * \addtogroup CONV
00020  * \{
00021  */
00022
00023 #ifndef CONV_H
00024 #define CONV_H
00025
00026 #ifdef __cplusplus
00027 extern "C" {
00028 #endif
00029
00030 #include "definitions.h"
00031 #include "signals.h"
00032 #include "stdio.h"
00033
00034 /* User-Defined Callback Functions to Convert the RAW CAN Data to Values */
00035 /* Numbers are Hardcoded here, to make it more Readable */
00036
00037 /* CAN Signals*/
00038 float CONV_MinBatVoltage(uint8_t* data);
00039 float CONV_MaxBatTemp(uint8_t* data);
00040 float CONV_LapTime(uint8_t* data);
00041 float CONV_BestLapTime(uint8_t* data);
00042 uint32_t CONV_FSG_AMI_state(uint8_t* data);
00043 float CONV_MaxMotTemp(uint8_t* data);
00044 float CONV_MaxInvTemp(uint8_t* data);
00045 float CONV_InverterTemp_FL(uint8_t* data);
00046 float CONV_InverterTemp_FR(uint8_t* data);
00047 float CONV_InverterTemp_RL(uint8_t* data);
00048 float CONV_InverterTemp_RR(uint8_t* data);
```

```
00049 float CONV_MotorTemp_RR(uint8_t* data);
00050 float CONV_MotorTemp_RL(uint8_t* data);
00051 float CONV_MotorTemp_FL(uint8_t* data);
00052 float CONV_MotorTemp_FR(uint8_t* data);
00053
00054 /* Internal Signals */
00055 float CONV_MaxMotorTemp(signals_signal* signal_list, uint32_t signal_list_len);
00056 float CONV_LastLapTime(signals_signal* signal_list, uint32_t signal_list_len);
00057 float CONV_MaxInverterTemp(signals_signal* signal_list,
00058         uint32_t signal_list_len);
00059
00060 /* Display Signals */
00061 void CONV_DISP_Motor_Temp(signals_signal* signal_list, uint32_t signal_list_len,
00062         SIGNALS_string* outstring);
00063 void CONV_DISP_MinVoltage(signals_signal* signal_list, uint32_t signal_list_len,
00064         SIGNALS_string* outstring);
00065 void CONV_DISP_LapDelta(signals_signal* signal_list, uint32_t signal_list_len,
00066         SIGNALS_string* outstring);
00067 void CONV_DISP_LapTime(signals_signal* signal_list, uint32_t signal_list_len,
00068         SIGNALS_string* outstring);
00069 void CONV_DISP_LastLapTime(signals_signal* signal_list, uint32_t signal_list_len,
00070         SIGNALS_string* outstring);
00071 void CONV_DISP_InverterTemp(signals_signal* signal_list, uint32_t signal_list_len,
00072         SIGNALS_string* outstring);
00073 void CONV_DISP_MaxBatTemp(signals_signal* signal_list, uint32_t signal_list_len,
00074         SIGNALS_string* outstring);
00075 void CONV_DISP_FSG_AMI_State(signals_signal* signal_list,
00076         uint32_t signal_list_len, SIGNALS_string* outstring);
00077
00078 /* Useful Arithmetic Functions (No Signal Callback Functions)*/
00079 uint32_t CONV_find_string_length(uint8_t* str, uint32_t strlen);
00080 float CONV_max(float* vals, uint32_t valcount);
00081 float CONV_min(float* vals, uint32_t valcount);
00082
00083 #ifdef  __cplusplus
00084 }
00085 #endif
00086
00087 #endif  /* CONV_H */
00088
00089 /**
00090  * \}
00091  */
00092
```

9.13 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)

9.13.1 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

Definition in file crc.c.

9.13.2 Function Documentation

9.13.2.1 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.

Definition at line 74 of file crc.c.

9.13.2.2 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.

Definition at line 61 of file crc.c.

9.14 crc.c

Go to the documentation of this file.

```
00001 /**
00002  * \file
00003  * Functions and types for CRC checks.
00004  *
00005  * Generated on Wed May 11 23:00:19 2022
00006  * by pycrc v0.9.2, https://pycrc.org
00007  * using the configuration:
00008  *  - Width         = 16
00009  *  - Poly          = 0x1021
00010  *  - XorIn         = 0x1d0f
00011  *  - ReflectIn     = False
00012  *  - XorOut        = 0x0000
00013  *  - ReflectOut    = False
00014  *  - Algorithm     = table-driven
00015  */
00016 #include "crc.h"     /* include the header file generated with pycrc */
00017 #include <stdlib.h>
00018 #include <stdint.h>
00019
00020
00021
00022 /**
00023  * Static table used for the table_driven implementation.
00024  */
00025 static const crc_t crc_table[256] = {
00026     0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
00027     0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
00028     0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
00029     0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
00030     0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
00031     0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
00032     0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
00033     0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
00034     0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
00035     0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
00036     0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
00037     0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
00038     0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
00039     0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
00040     0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
00041     0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
00042     0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
00043     0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
00044     0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
00045     0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
00046     0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
00047     0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
00048     0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
00049     0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
00050     0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
00051     0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
00052     0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
00053     0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
00054     0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
00055     0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
00056     0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
00057     0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
00058 };
00059
00060
00061 crc_t crc_update(crc_t crc, const void *data, size_t data_len)
00062 {
00063     const unsigned char *d = (const unsigned char *)data;
00064     unsigned int tbl_idx;
00065
00066     while (data_len--) {
00067         tbl_idx = ((crc » 8) ^ *d) & 0xff;
00068         crc = (crc_table[tbl_idx] ^ (crc « 8)) & 0xffff;
00069         d++;
00070     }
```

```
00071     return crc & 0xffff;
00072 }
00073
00074 uint32_t CRC_Calculate(void* data, uint32_t length){
00075     crc_t crc;
00076     crc = crc_init();
00077     crc = crc_update(crc, data, length);
00078     crc = crc_finalize(crc);
00079     return crc;
00080 }
00081
```

9.15 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)

9.15.1 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);
```

Definition in file crc.h.

9.15.2 Macro Definition Documentation

9.15.2.1 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.

Definition at line 57 of file crc.h.

9.15.3 Typedef Documentation

9.15.3.1 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.

Definition at line 65 of file crc.h.

9.15.4 Function Documentation

9.15.4.1 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.

Definition at line 74 of file crc.c.

9.15.4.2 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.

Definition at line 61 of file crc.c.

9.16 crc.h

Go to the documentation of this file.

```
00001 /**
00002  * \file
00003  * Functions and types for CRC checks.
00004  *
00005  * Generated on Wed May 11 23:00:05 2022
00006  * by pycrc v0.9.2, https://pycrc.org
00007  * using the configuration:
00008  *  - Width         = 16
00009  *  - Poly          = 0x1021
00010  *  - XorIn         = 0x1d0f
00011  *  - ReflectIn     = False
00012  *  - XorOut        = 0x0000
00013  *  - ReflectOut    = False
00014  *  - Algorithm     = table-driven
00015  *
00016  * This file defines the functions crc_init(), crc_update() and crc_finalize().
00017  *
00018  * The crc_init() function returns the inital \c crc value and must be called
```

```
00019  * before the first call to crc_update().
00020  * Similarly, the crc_finalize() function must be called after the last call
00021  * to crc_update(), before the \c crc is being used.
00022  * is being used.
00023  *
00024  * The crc_update() function can be called any number of times (including zero
00025  * times) in between the crc_init() and crc_finalize() calls.
00026  *
00027  * This pseudo-code shows an example usage of the API:
00028  * \code{.c}
00029  * crc_t crc;
00030  * unsigned char data[MAX_DATA_LEN];
00031  * size_t data_len;
00032  *
00033  * crc = crc_init();
00034  * while ((data_len = read_data(data, MAX_DATA_LEN)) > 0) {
00035  *     crc = crc_update(crc, data, data_len);
00036  * }
00037  * crc = crc_finalize(crc);
00038  * \endcode
00039  */
00040 #ifndef CRC_H
00041 #define CRC_H
00042
00043 #include <stdlib.h>
00044 #include <stdint.h>
00045
00046 #ifdef __cplusplus
00047 extern "C" {
00048 #endif
00049
00050
00051 /**
00052  * The definition of the used algorithm.
00053  *
00054  * This is not used anywhere in the generated code, but it may be used by the
00055  * application code to call algorithm-specific code, if desired.
00056  */
00057 #define CRC_ALGO_TABLE_DRIVEN 1
00058
00059
00060 /**
00061  * The type of the CRC values.
00062  *
00063  * This type must be big enough to contain at least 16 bits.
00064  */
00065 typedef uint_fast16_t crc_t;
00066
00067
00068 /**
00069  * Calculate the initial crc value.
00070  *
00071  * \return     The initial crc value.
00072  */
00073 static inline crc_t crc_init(void)
00074 {
00075     return 0xFFFF;
00076 }
00077
00078
00079 /**
00080  * Update the crc value with new data.
00081  *
00082  * \param[in] crc      The current crc value.
00083  * \param[in] data     Pointer to a buffer of \a data_len bytes.
00084  * \param[in] data_len Number of bytes in the \a data buffer.
00085  * \return             The updated crc value.
00086  */
00087 crc_t crc_update(crc_t crc, const void *data, size_t data_len);
00088
00089
00090 /**
00091  * Calculate the final crc value.
00092  *
00093  * \param[in] crc  The current crc value.
00094  * \return     The final crc value.
00095  */
00096 static inline crc_t crc_finalize(crc_t crc)
00097 {
00098     return crc;
00099 }
00100
00101 /**
00102  * Wrapper function to Calculate the CRC over a Array
00103  *
00104  * \param[in] data The Array of Data to calculate the CRC of.
00105  * \param[in] length The Length of the Input Data Array
```

```
00106  *
00107  * \return The CRC Value.
00108  *
00109  */
00110 uint32_t CRC_Calculate(void* data, uint32_t length);
00111
00112
00113 #ifdef __cplusplus
00114 }           /* closing brace for extern "C" */
00115 #endif
00116
00117 #endif      /* CRC_H */
```

9.17 delay.c

```
00001 #include "delay.h"
00002
00003 void DELAY_Milliseconds(uint32_t delay){
00004     uint32_t i=0;
00005     for(;i < delay; i++){
00006         DELAY_Microseconds(MILLISECONDS_IN_SECOND);
00007     }
00008 }
00009
00010 void DELAY_Microseconds(uint32_t delay){
00011     uint32_t cycles;
00012     cycles = (CLOCK_FREQUENCY_CORE /
00013             (MICROSECONDS_IN_SECOND * TWO_STEPS_DELAY_ADJ))
00014              * delay;
00015     uint32_t i=0;
00016     for(; i<cycles; i++){
00017
00018     }
00019 }
```

9.18 delay.h File Reference

This File defines the Prototypes for DELAY.

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

Macros

- #define CLOCK_FREQUENCY_CORE 120000000

  The Frequency of the Core in Hz.
- #define MICROSECONDS_IN_SECOND 1000000

  Count of Microseconds in a Second.
- #define MILLISECONDS_IN_SECOND 1000

  Count of Milliseconds in a Second.
- #define TWO_STEPS_DELAY_ADJ 2

Functions

- void DELAY_Milliseconds (uint32_t delay)

  Delay for an amount of Milliseconds.
- void DELAY_Microseconds (uint32_t delay)

  Delay for an amount of Microseconds.

9.18.1 Detailed Description

This File defines the Prototypes for DELAY.

Author

Frederic Emmerth

Bug Millisecond Deley is inaccurate

Definition in file delay.h.

9.19 delay.h

Go to the documentation of this file.

```
00001 /**
00002  * @file delay.h
00003  * @brief This File defines the Prototypes for \ref DELAY
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup DELAY
00008  *
00009  * @bug Millisecond Deley is inaccurate
00010  *
00011  */
00012
00013 /**
00014  * \defgroup DELAY DELAY
00015  * This Module implements Functions to sleep/delay for a specified Amount of
00016  * Time.
00017  * \addtogroup DELAY
00018  * \{
00019  */
00020
00021 #ifndef DELAY_H
00022 #define DELAY_H
00023
00024 #ifdef  __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #include <stdint.h>
00029
00030 /**
00031  * @brief The Frequency of the Core in Hz.
00032  */
00033 #define CLOCK_FREQUENCY_CORE 120000000
00034
00035 /**
00036  * @brief Count of Microseconds in a Second
00037  */
00038 #define MICROSECONDS_IN_SECOND 1000000
00039
00040 /**
00041  * @brief Count of Milliseconds in a Second
00042  */
00043 #define MILLISECONDS_IN_SECOND 1000
00044
00045 /**
00046  * Adjustment of the Delay Functions
00047  */
00048 #define TWO_STEPS_DELAY_ADJ 2
00049
00050 /**
00051  * @brief Delay for an amount of Milliseconds
00052  * @param delay The Time to Delay in Milliseconds
00053  */
00054 void DELAY_Milliseconds(uint32_t delay);
00055
00056 /**
00057  * @brief Delay for an amount of Microseconds
00058  * @param delay The Time to Delay in Microseconds
```

```
00059  */
00060 void DELAY_Microseconds(uint32_t delay);
00061
00062
00063 #ifdef  __cplusplus
00064 }
00065 #endif
00066
00067 #endif  /* DELAY_H */
00068
00069 /**
00070  * \}
00071  */
00072
```

9.20 main.c File Reference

```
#include <stddef.h>
#include <stdbool.h>
#include <stdlib.h>
#include "definitions.h"
#include "delay.h"
#include "cancomm.h"
#include "signals.h"
#include "conv.h"
#include "shortprotocol.h"
#include "uart.h"
#include "command.h"
```

Functions

- int main (void)

Variables

- cancomm_interface interface_list [ ]
- uint32_t interface_list_len
- cancomm_message message_list [ ]
- uint32_t message_list_len = sizeof(message_list) / sizeof(cancomm_message)
- signals_signal signal_list [ ]
- uint32_t signal_list_len = sizeof(signal_list) / sizeof(signals_signal)
- uint32_t current_command_signal = 0
- SHORTPROTOCOL_Instance shortProt

9.20.1 Detailed Description

In this file, there is the main() function wich is the entry Point of The Program

Author

Frederic Emmerth

Todo Make a constant loop time

Implement Watchdog

Definition in file main.c.

9.20.2 Function Documentation

9.20.2.1 main()

```
int main (
            void  )
```

Definition at line 241 of file main.c.

9.20.3 Variable Documentation

9.20.3.1 current_command_signal

```
uint32_t current_command_signal = 0
```

Definition at line 230 of file main.c.

9.20.3.2 interface_list

```
cancomm_interface interface_list[]
```

Initial value:

```
= {
    {.number=1,
            .receiveFifo=2, .transmitFifo=1,
            .MessageTransmit=CAN1_MessageTransmit,
            .MessageReceive=CAN1_MessageReceive
    },
    {.number=2,
            .receiveFifo=2, .transmitFifo=1,
            .MessageTransmit=CAN2_MessageTransmit,
            .MessageReceive=CAN2_MessageReceive
    },
    {.number=3,
            .receiveFifo=2, .transmitFifo=1,
            .MessageTransmit=CAN3_MessageTransmit,
            .MessageReceive=CAN3_MessageReceive
    },
    {.number=4,
            .receiveFifo=2, .transmitFifo=1,
            .MessageTransmit=CAN4_MessageTransmit,
            .MessageReceive=CAN4_MessageReceive
    }
}
```

Definition at line 28 of file main.c.

9.20.3.3 interface_list_len

```
uint32_t interface_list_len
```

Initial value:

```
= sizeof(interface_list)
                                        / sizeof(cancomm_interface)
```

Definition at line 50 of file main.c.

9.20.3.4 message_list

```
cancomm_message message_list[]
```

Initial value:

```
= {
    {.friendly_name="BCU_Extrem_Voltages",
            .id=0xA2,  .interface_number=1, .length=8
    },
    {.friendly_name="BCU_Extrem_Temperatures",
            .id=0xA3,  .interface_number=1, .length=8
    },
    {.friendly_name="VCU_LapInfo",
            .id=0x711, .interface_number=1, .length=8
    },
    {.friendly_name="FSG_System_status",
            .id=0x502, .interface_number=1, .length=8
    },
    {.friendly_name="BCU_SCstatus",
            .id=0xA4,  .interface_number=1, .length=2
    },
    {.friendly_name="VCU_Motor_Temps",
            .id=0x239, .interface_number=1, .length=8
    },
    {.friendly_name="VCU_Inverter_Temp",
            .id=0x23A, .interface_number=1, .length=8
    }
}
```

Definition at line 54 of file main.c.

9.20.3.5 message_list_len

```
uint32_t message_list_len = sizeof(message_list) / sizeof(cancomm_message)
```

Definition at line 77 of file main.c.

9.20.3.6 shortProt

```
SHORTPROTOCOL_Instance shortProt
```

Initial value:

```
= {
    .readAvailable = UART_ReadAvailable,
    .readByte = UART_ReadByte,
    .writeAvailable = UART_WriteAvailable,
    .writeByte = UART_WriteByte,
    .maximumPackageLength = 5
}
```

Definition at line 233 of file main.c.

9.20.3.7 signal_list

```
signals_signal signal_list[]
```

Definition at line 80 of file main.c.

9.20.3.8 signal_list_len

```
uint32_t signal_list_len = sizeof(signal_list) / sizeof(signals_signal)
```

Definition at line 228 of file main.c.

9.21 main.c

Go to the documentation of this file.

```
00001 /**
00002  * @file main.c
00003  * In this file, there is the main() function wich is the entry Point of
00004  * The Program
00005  *
00006  * @author Frederic Emmerth
00007  *
00008  * @todo Make a constant loop time
00009  * @todo Implement Watchdog
00010  *
00011  */
00012
00013
00014 /* Includes */
00015 #include <stddef.h>
00016 #include <stdbool.h>
00017 #include <stdlib.h>
00018 #include "definitions.h"
00019 #include "delay.h"
00020 #include "cancomm.h"
00021 #include "signals.h"
00022 #include "conv.h"
00023 #include "shortprotocol.h"
00024 #include "uart.h"
00025 #include "command.h"
00026
00027 /* Define List of CAN Interfaces */
00028 cancomm_interface interface_list [] = {
00029     {.number=1,
00030             .receiveFifo=2, .transmitFifo=1,
00031             .MessageTransmit=CAN1_MessageTransmit,
00032             .MessageReceive=CAN1_MessageReceive
00033     },
00034     {.number=2,
00035             .receiveFifo=2, .transmitFifo=1,
00036             .MessageTransmit=CAN2_MessageTransmit,
00037             .MessageReceive=CAN2_MessageReceive
00038     },
00039     {.number=3,
00040             .receiveFifo=2, .transmitFifo=1,
00041             .MessageTransmit=CAN3_MessageTransmit,
00042             .MessageReceive=CAN3_MessageReceive
00043     },
00044     {.number=4,
00045             .receiveFifo=2, .transmitFifo=1,
00046             .MessageTransmit=CAN4_MessageTransmit,
00047             .MessageReceive=CAN4_MessageReceive
00048     }
00049 };
00050 uint32_t interface_list_len = sizeof(interface_list)
00051                                         / sizeof(cancomm_interface);
00052
00053 /* Define all Messages to be interpreted */
00054 cancomm_message message_list [] = {
00055     {.friendly_name="BCU_Extrem_Voltages",
```

```
00056             .id=0xA2,  .interface_number=1, .length=8
00057     },
00058     {.friendly_name="BCU_Extrem_Temperatures",
00059             .id=0xA3,  .interface_number=1, .length=8
00060     },
00061     {.friendly_name="VCU_LapInfo",
00062             .id=0x711, .interface_number=1, .length=8
00063     },
00064     {.friendly_name="FSG_System_status",
00065             .id=0x502, .interface_number=1, .length=8
00066     },
00067     {.friendly_name="BCU_SCstatus",
00068             .id=0xA4,  .interface_number=1, .length=2
00069     },
00070     {.friendly_name="VCU_Motor_Temps",
00071             .id=0x239, .interface_number=1, .length=8
00072     },
00073     {.friendly_name="VCU_Inverter_Temp",
00074             .id=0x23A, .interface_number=1, .length=8
00075     }
00076 };
00077 uint32_t message_list_len = sizeof(message_list) / sizeof(cancomm_message);
00078
00079 /* Define all Signals to be interpreted */
00080 signals_signal signal_list [] = {
00081     {.friendly_name="MinBatVoltage",
00082             .type = SIGNALS_CAN_MESSAGE,
00083             .id=0xA2,
00084             .interface_number=1,
00085             .data_type=SIGNALS_FLOAT_SIGNAL,
00086             .can_convert_float=CONV_MinBatVoltage
00087     },
00088     {.friendly_name="MaxBatTemp",
00089             .type = SIGNALS_CAN_MESSAGE,
00090             .id=0xA3,
00091             .interface_number=1,
00092             .data_type=SIGNALS_FLOAT_SIGNAL,
00093             .can_convert_float=CONV_MaxBatTemp
00094     },
00095     {.friendly_name="LapTime",
00096             .type = SIGNALS_CAN_MESSAGE,
00097             .id=0x711,
00098             .interface_number=1,
00099             .data_type=SIGNALS_FLOAT_SIGNAL,
00100             .can_convert_float=CONV_LapTime
00101     },
00102     {.friendly_name="BestLapTime",
00103             .type=SIGNALS_CAN_MESSAGE,
00104             .id=0x711,
00105             .interface_number=1,
00106             .data_type=SIGNALS_FLOAT_SIGNAL,
00107             .can_convert_float=CONV_BestLapTime
00108     },
00109     {.friendly_name="FSG_AMI_state",
00110             .type = SIGNALS_CAN_MESSAGE,
00111             .id=0x502,
00112             .interface_number=1,
00113             .data_type=SIGNALS_UINT32_T_SIGNAL,
00114             .can_convert_uint32_t=CONV_FSG_AMI_state
00115     },
00116     {.friendly_name="InverterTemp_RR",
00117             .type = SIGNALS_CAN_MESSAGE,
00118             .id = 0x23A,
00119             .interface_number=1,
00120             .data_type=SIGNALS_FLOAT_SIGNAL,
00121             .can_convert_float=CONV_InverterTemp_RR
00122     },
00123     {.friendly_name="InverterTemp_FL",
00124             .type = SIGNALS_CAN_MESSAGE,
00125             .id = 0x23A,
00126             .interface_number=1,
00127             .data_type=SIGNALS_FLOAT_SIGNAL,
00128             .can_convert_float=CONV_InverterTemp_FL
00129     },
00130     {.friendly_name="InverterTemp_FR",
00131             .type = SIGNALS_CAN_MESSAGE,
00132             .id = 0x23A,
00133             .interface_number=1,
00134             .data_type=SIGNALS_FLOAT_SIGNAL,
00135             .can_convert_float=CONV_InverterTemp_FR
00136     },
00137     {.friendly_name="InverterTemp_RL",
00138             .type = SIGNALS_CAN_MESSAGE,
00139             .id = 0x23A,
00140             .interface_number=1,
00141             .data_type=SIGNALS_FLOAT_SIGNAL,
00142             .can_convert_float=CONV_InverterTemp_RL
```

```
00143     },
00144     {.friendly_name="MotorTemp_RR",
00145             .type = SIGNALS_CAN_MESSAGE,
00146             .id=0x239,
00147             .interface_number=1,
00148             .data_type=SIGNALS_FLOAT_SIGNAL,
00149             .can_convert_float=CONV_MotorTemp_RR
00150     },
00151     {.friendly_name="MotorTemp_FL",
00152             .type = SIGNALS_CAN_MESSAGE,
00153             .id=0x239,
00154             .interface_number=1,
00155             .data_type=SIGNALS_FLOAT_SIGNAL,
00156             .can_convert_float=CONV_MotorTemp_FL
00157     },
00158     {.friendly_name="MotorTemp_FR",
00159             .type = SIGNALS_CAN_MESSAGE,
00160             .id=0x239,
00161             .interface_number=1,
00162             .data_type=SIGNALS_FLOAT_SIGNAL,
00163             .can_convert_float=CONV_MotorTemp_FR
00164     },
00165     {.friendly_name="MotorTemp_RL",
00166             .type = SIGNALS_CAN_MESSAGE,
00167             .id=0x239,
00168             .interface_number=1,
00169             .data_type=SIGNALS_FLOAT_SIGNAL,
00170             .can_convert_float=CONV_MotorTemp_RL
00171     },
00172     {.friendly_name="MaxMotorTemp",
00173             .type = SIGNALS_INTERNAL_SIGNAL,
00174             .data_type=SIGNALS_FLOAT_SIGNAL,
00175             .internal_convert_float=CONV_MaxMotorTemp
00176     },
00177     {.friendly_name="MaxInverterTemp",
00178             .type = SIGNALS_INTERNAL_SIGNAL,
00179             .data_type=SIGNALS_FLOAT_SIGNAL,
00180             .internal_convert_float=CONV_MaxInverterTemp
00181     },
00182     {.friendly_name="LastLap",
00183             .type = SIGNALS_INTERNAL_SIGNAL,
00184             .data_type=SIGNALS_FLOAT_SIGNAL,
00185             .internal_convert_float=CONV_LastLapTime
00186     },
00187     {.friendly_name="DISP_Motor_Temp",
00188             .type=SIGNALS_DISPLAY_SIGNAL,
00189             .data_type=SIGNALS_STRING_SIGNAL,
00190             .internal_convert_string=CONV_DISP_Motor_Temp
00191     },
00192     {.friendly_name="DISP_MinVoltage",
00193             .type=SIGNALS_DISPLAY_SIGNAL,
00194             .data_type=SIGNALS_STRING_SIGNAL,
00195             .internal_convert_string=CONV_DISP_MinVoltage
00196     },
00197     {.friendly_name="DISP_LapTime",
00198             .type=SIGNALS_DISPLAY_SIGNAL,
00199             .data_type=SIGNALS_STRING_SIGNAL,
00200             .internal_convert_string=CONV_DISP_LapTime
00201     },
00202     {.friendly_name="DISP_LapDelta",
00203             .type=SIGNALS_DISPLAY_SIGNAL,
00204             .data_type=SIGNALS_STRING_SIGNAL,
00205             .internal_convert_string=CONV_DISP_LapDelta
00206     },
00207     {.friendly_name="DISP_LastLap",
00208             .type=SIGNALS_DISPLAY_SIGNAL,
00209             .data_type=SIGNALS_STRING_SIGNAL,
00210             .internal_convert_string=CONV_DISP_LastLapTime
00211     },
00212     {.friendly_name="DISP_Inverter_Temp",
00213             .type=SIGNALS_DISPLAY_SIGNAL,
00214             .data_type=SIGNALS_STRING_SIGNAL,
00215             .internal_convert_string=CONV_DISP_InverterTemp
00216     },
00217     {.friendly_name="DISP_MaxBatTemp",
00218             .type=SIGNALS_DISPLAY_SIGNAL,
00219             .data_type=SIGNALS_STRING_SIGNAL,
00220             .internal_convert_string=CONV_DISP_MaxBatTemp
00221     },
00222     {.friendly_name="DISP_FSG_AMI_State",
00223             .type=SIGNALS_DISPLAY_SIGNAL,
00224             .data_type=SIGNALS_STRING_SIGNAL,
00225             .internal_convert_string=CONV_DISP_FSG_AMI_State
00226     },
00227 };
00228 uint32_t signal_list_len = sizeof(signal_list) / sizeof(signals_signal);
00229
```

```
00230 uint32_t current_command_signal = 0;
00231
00232 /* Define a Protocol Instance for Shortprotocol Communication with Display */
00233 SHORTPROTOCOL_Instance shortProt = {
00234     .readAvailable = UART_ReadAvailable,
00235     .readByte = UART_ReadByte,
00236     .writeAvailable = UART_WriteAvailable,
00237     .writeByte = UART_WriteByte,
00238     .maximumPackageLength = 5
00239 };
00240
00241 int main ( void )
00242 {
00243     /* Initialize all modules */
00244     SYS_Initialize ( NULL );
00245
00246     /* Give Shortprotocol Parameters initial Values */
00247     SHORTPROTOCOL_Initialize(&shortProt);
00248
00249     /* Recieve CAN Frames for 5ms */
00250     DELAY_Microseconds(5000);
00251
00252     /* Main Loop */
00253     /* The Maximum Loop Time has to be smaller than 2ms, to catch all Messages*80/
00254     /* 15k Frames/s MAX and FIFO Length 32 -> 1/15E3 * 32 = 2ms */
00255     while(1){
00256
00257         /* Read All Messages in the CAN FIFOs */
00258         CANCOMM_ReadMessages(message_list, message_list_len,
00259                 interface_list, interface_list_len);
00260
00261         /* Interpret the RAW CAN Message Data */
00262         SIGNALS_Interpret(signal_list, signal_list_len,
00263                 message_list, message_list_len);
00264
00265         /* Generate the Commands to be sent to the Display */
00266         COMMAND_Generate(signal_list, signal_list_len,
00267                 &current_command_signal, &shortProt);
00268
00269         /* Send Messages to Display */
00270         SHORTPROTOCOL_Update(&shortProt);
00271
00272         /* Wait for constant Loop time */
00273         DELAY_Microseconds(200);
00274
00275         /* (Window WDG Reset)*/
00276
00277
00278     }
00279
00280     /* Execution should not come here during normal operation */
00281 }
00282
00283
```

9.22 shortprotocol.c File Reference

This File implements the Prototypes for SHORTPROTOCOL.

```
#include "shortprotocol.h"
```

Functions

- SHORTPROTOCOL_status SHORTPROTOCOL_Send (SHORTPROTOCOL_Instance *inst, uint8_t *data, uint32_t length)

  Function to try to send a new Command using the SHORTPROTOCOL.
- void SHORTPROTOCOL_Update (SHORTPROTOCOL_Instance *inst)

  Function to be called repeadetly in the Main Loop to send Pieces of maximumPackageLength of the Package generated by SHORTPROTOCOL_Send.
- SHORTPROTOCOL_status SHORTPROTOCOL_Available (SHORTPROTOCOL_Instance *inst)

Returns if a new Command can be sent using the Shortprotocol.

- void SHORTPROTOCOL_Initialize (SHORTPROTOCOL_Instance *inst)

  Initializes an Empty SHORTPROTOCOL_Instance. Callback Functions still have to be added/defined and the maximumPackageLength has to be set.

9.22.1 Detailed Description

This File implements the Prototypes for SHORTPROTOCOL.

Author

Frederic Emmerth

Definition in file shortprotocol.c.

9.23 shortprotocol.c

Go to the documentation of this file.

```
00001 /**
00002  * @file shortprotocol.c
00003  * @brief This File implements the Prototypes for \ref SHORTPROTOCOL
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup SHORTPROTOCOL
00008  *
00009  * \addtogroup SHORTPROTOCOL
00010  * \{
00011  *
00012  */
00013
00014
00015 #include "shortprotocol.h"
00016
00017 SHORTPROTOCOL_status SHORTPROTOCOL_Send(SHORTPROTOCOL_Instance* inst,
00018         uint8_t* data, uint32_t length){
00019     /* Command Length without Zero Termination */
00020     /* This Function generates a Shortprotocol Packet from a Command */
00021
00022     /* If the last Packet is not yet send, cant accept a new Command */
00023     if(inst->newCommand == SHORTPROTOCOL_NOT_AVAILABLE){
00024         return SHORTPROTOCOL_NOT_AVAILABLE;
00025     }
00026
00027     /* Check that the Command will fit in the Packet Buffer */
00028     if(length > SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH){
00029         length = SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH;
00030     }
00031
00032     /* Check that the Command is not zero length */
00033     if(length == 0){
00034         return SHORTPROTOCOL_ERROR;
00035     }
00036
00037     /* First Byte of Packet is Always the Same */
00038     inst->command_buffer.data[SHORTPROTOCOL_BEGIN_OFFSET] = SHORTPROTOCOL_BEGIN;
00039
00040     /* Second and third Byte is the Length of the Command */
00041     inst->command_buffer.data[SHORTPROTOCOL_LENGTH_LSB_OFFSET] =
00042             (length & SHORTPROTOCOL_FIRST_BYTE_MASK);
00043
00044     inst->command_buffer.data[SHORTPROTOCOL_LENGTH_MSB_OFFSET] =
00045             ((length & SHORTPROTOCOL_SECOND_BYTE_MASK)
00046             » SHORTPROTOCOL_BYTE_LENGTH);
00047
00048     /* The Command begins on the fourth byte */
00049     for(uint32_t i=0; i<length; i++){
00050         inst->command_buffer.data[i + SHORTPROTOCOL_COMMAND_OFFSET] = data[i];
00051     }
00052
```

```
00053     /* Calculate the CRC of the Packet */
00054     uint32_t crc_temp;
00055     crc_temp = CRC_Calculate(inst->command_buffer.data,
00056             length + SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES);
00057
00058     /* The last two Bytes are the CRC */
00059     inst->command_buffer.data[SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES +
00060                                     length + SHORTPROTOCOL_CRC_LSB_OFFSET] =
00061             crc_temp & SHORTPROTOCOL_FIRST_BYTE_MASK;
00062
00063     inst->command_buffer.data[SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES +
00064                                     length + SHORTPROTOCOL_CRC_MSB_OFFSET] =
00065             (crc_temp & SHORTPROTOCOL_SECOND_BYTE_MASK)
00066                     » SHORTPROTOCOL_BYTE_LENGTH;
00067
00068     /* Calculate the Length of the Full Packet */
00069     inst->command_buffer.length = length + SHORTPROTOCOL_OVERHEAD_BYTES;
00070
00071     /* Tell the Update Function a Packet is ready */
00072     inst->writeCounter = 0;
00073     inst->newCommand = SHORTPROTOCOL_NOT_AVAILABLE;
00074
00075     return SHORTPROTOCOL_SUCCESS;
00076 }
00077
00078 void SHORTPROTOCOL_Update(SHORTPROTOCOL_Instance* inst){
00079     /*
00080      * If a New Command is not available
00081      * (therefore the current one is not completed)
00082      */
00083     if(inst->newCommand == SHORTPROTOCOL_NOT_AVAILABLE){
00084         /* Counter to control the length of this package being sent */
00085         uint32_t package_length = 0;
00086         while(package_length < inst->maximumPackageLength){
00087
00088             inst->writeByte(inst->command_buffer.data[inst->writeCounter]);
00089
00090             while(inst->writeAvailable() == SHORTPROTOCOL_NOT_AVAILABLE);
00091
00092             if(inst->writeCounter == inst->command_buffer.length - 1){
00093                 /* Command was completly written, a new Command is writable */
00094                 inst->newCommand = SHORTPROTOCOL_AVAILABLE;
00095                 break;
00096             }
00097
00098             inst->writeCounter += 1;
00099             package_length += 1;
00100         }
00101     }
00102 }
00103
00104 SHORTPROTOCOL_status SHORTPROTOCOL_Available(SHORTPROTOCOL_Instance* inst){
00105     return inst->newCommand;
00106 }
00107
00108 void SHORTPROTOCOL_Initialize(SHORTPROTOCOL_Instance* inst){
00109     inst->writeCounter = 0;
00110     inst->newCommand = SHORTPROTOCOL_AVAILABLE;
00111 }
00112
00113 /**
00114  * \}
00115  */
```

9.24 shortprotocol.h File Reference

This File defines the Prototypes for SHORTPROTOCOL.

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

Data Structures

- struct SHORTPROTOCOL_string

Struct to combine the length of a String with the String and the Overhead Bytes used by the Shortprotocol in one Data type.

- struct SHORTPROTOCOL_Instance

  Configuration Struct for SHORTPROTOCOL passed to the SHORTPROTOCOL Functions.

Macros

- #define SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH 100

  Maximum Command length, thus maximum payload length.
- #define SHORTPROTOCOL_BEGIN 0x13

  The Begining Byte to indicate a Transmission.
- #define SHORTPROTOCOL_OVERHEAD_BYTES 5

  The Overhead of sending Data with the Shortprotocol.
- #define SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES 3

  The Overhead of sending Data with the Shortprotocol, not counting the two CRC Bytes.
- #define SHORTPROTOCOL_FIRST_BYTE_MASK 0x00FF

  Mask to convert endianess (first Byte is on second place)
- #define SHORTPROTOCOL_SECOND_BYTE_MASK 0xFF00

  Mask to convert endianess (second Byte in on first place)
- #define SHORTPROTOCOL_BYTE_LENGTH 8

  The length of one Byte for endianess conversion.
- #define SHORTPROTOCOL_BEGIN_OFFSET 0

  The Offset of the Begin Byte in the Shortprotocol.
- #define SHORTPROTOCOL_LENGTH_LSB_OFFSET 1

  The Offset of the Least Significant Byte of the Length in the Shortprotocol.
- #define SHORTPROTOCOL_LENGTH_MSB_OFFSET 2

  The Offset of the Most Significant Byte of the Length in the Shortprotocol.
- #define SHORTPROTOCOL_COMMAND_OFFSET 3

  The Offset of the Command, thus the Payload in the Shortprotocol.
- #define SHORTPROTOCOL_CRC_LSB_OFFSET 0

  The Offset of the CRC Least Significant Byte from the Back of the Package.
- #define SHORTPROTOCOL_CRC_MSB_OFFSET 1

  The Offset of the CRC Most Significant Byte from the Back of the Package.

Enumerations

- enum SHORTPROTOCOL_status { **SHORTPROTOCOL_SUCCESS** = 0 , **SHORTPROTOCOL_ERROR** , **SHORTPROTOCOL_AVAILABLE** , **SHORTPROTOCOL_NOT_AVAILABLE** }

  Return Status used by both the Callback Functions of the SHORTPROTOCOL and the Functions.

Functions

- SHORTPROTOCOL_status SHORTPROTOCOL_Send (SHORTPROTOCOL_Instance *inst, uint8_t *data, uint32_t length)

  Function to try to send a new Command using the SHORTPROTOCOL.
- void SHORTPROTOCOL_Update (SHORTPROTOCOL_Instance *inst)

  Function to be called repeadetly in the Main Loop to send Pieces of maximumPackageLength of the Package generated by SHORTPROTOCOL_Send.
- SHORTPROTOCOL_status SHORTPROTOCOL_Available (SHORTPROTOCOL_Instance *inst)

  Returns if a new Command can be sent using the Shortprotocol.
- void SHORTPROTOCOL_Initialize (SHORTPROTOCOL_Instance *inst)

  Initializes an Empty SHORTPROTOCOL_Instance. Callback Functions still have to be added/defined and the maximumPackageLength has to be set.

9.24.1 Detailed Description

This File defines the Prototypes for SHORTPROTOCOL.

Author

Frederic Emmerth

Definition in file shortprotocol.h.

9.25 shortprotocol.h

Go to the documentation of this file.

```
00001 /**
00002  * @file shortprotocol.h
00003  * @brief This File defines the Prototypes for \ref SHORTPROTOCOL
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup SHORTPROTOCOL
00008  *
00009  */
00010
00011 /**
00012  * \defgroup SHORTPROTOCOL SHORTPROTOCOL
00013  * This Module handles packing the Commands generated by the \ref SIGNAL Module
00014  * and sending them using the Shortprotocol defined in the Displays
00015  * (EA uniTFTs035-ATC) Datasheet. To not block the Main Loop for too long, the
00016  * Packet is split into smaller Parts defined by
00017  * SHORTPROTOCOL_Instance.maximumPackageLength. The Module is written in a
00018  * general Fashion and can be adopted to use SPI, for this the Callback Functions
00019  * in SHORTPROTOCOL_Instance have to be redefined for SPI.
00020  *
00021  * \addtogroup SHORTPROTOCOL
00022  * \{
00023  */
00024
00025 #ifndef SHORTPROTOCOL_H
00026 #define SHORTPROTOCOL_H
00027
00028 #ifdef  __cplusplus
00029 extern "C" {
00030 #endif
00031
00032 #include "definitions.h"
00033 #include "crc.h"
00034
00035 /**
00036  * @brief Maximum Command length, thus maximum payload length
00037  */
00038 #define SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH 100
00039
00040 /**
00041  * @brief The Begining Byte to indicate a Transmission
00042  */
00043 #define SHORTPROTOCOL_BEGIN 0x13
00044
00045 /**
00046  * @brief The Overhead of sending Data with the Shortprotocol
00047  */
00048 #define SHORTPROTOCOL_OVERHEAD_BYTES 5
00049
00050 /**
00051  * @brief The Overhead of sending Data with the Shortprotocol, not counting the
00052  * two CRC Bytes.
00053  */
00054 #define SHORTPROTOCOL_OVERHEAD_WITHOUT_CRC_BYTES 3
00055
00056 /**
00057  * @brief Mask to convert endianess (first Byte is on second place)
00058  */
00059 #define SHORTPROTOCOL_FIRST_BYTE_MASK 0x00FF
00060
00061 /**
00062  * @brief Mask to convert endianess (second Byte in on first place)
```

```
00063  */
00064 #define SHORTPROTOCOL_SECOND_BYTE_MASK 0xFF00
00065
00066 /**
00067  * @brief The length of one Byte for endianess conversion
00068  */
00069 #define SHORTPROTOCOL_BYTE_LENGTH 8
00070
00071 /**
00072  * @brief The Offset of the Begin Byte in the Shortprotocol
00073  */
00074 #define SHORTPROTOCOL_BEGIN_OFFSET 0
00075
00076 /**
00077  * @brief The Offset of the Least Significant Byte of the Length in the
00078  * Shortprotocol
00079  */
00080 #define SHORTPROTOCOL_LENGTH_LSB_OFFSET 1
00081
00082
00083 /**
00084  * @brief The Offset of the Most Significant Byte of the Length in the
00085  * Shortprotocol
00086  */
00087 #define SHORTPROTOCOL_LENGTH_MSB_OFFSET 2
00088
00089 /**
00090  * @brief The Offset of the Command, thus the Payload in the Shortprotocol
00091  */
00092 #define SHORTPROTOCOL_COMMAND_OFFSET 3
00093
00094 /**
00095  * @brief The Offset of the CRC Least Significant Byte from the Back of the
00096  * Package
00097  */
00098 #define SHORTPROTOCOL_CRC_LSB_OFFSET 0
00099
00100 /**
00101  * @brief The Offset of the CRC Most Significant Byte from the Back of the
00102  * Package
00103  */
00104 #define SHORTPROTOCOL_CRC_MSB_OFFSET 1
00105
00106 /**
00107  * @brief Return Status used by both the Callback Functions of the
00108  * \ref SHORTPROTOCOL and the Functions.
00109  */
00110 typedef enum{
00111     SHORTPROTOCOL_SUCCESS = 0,
00112     SHORTPROTOCOL_ERROR,
00113     SHORTPROTOCOL_AVAILABLE,
00114     SHORTPROTOCOL_NOT_AVAILABLE
00115 }SHORTPROTOCOL_status;
00116
00117 /**
00118  * @brief Struct to combine the length of a String with the String and the
00119  * Overhead Bytes used by the Shortprotocol in one Data type.
00120  */
00121 typedef struct{
00122     uint32_t length;
00123     uint8_t data[SHORTPROTOCOL_MAXIMUM_COMMAND_LENGTH +
00124                          SHORTPROTOCOL_OVERHEAD_BYTES];
00125 }SHORTPROTOCOL_string;
00126
00127 /**
00128  * @brief Configuration Struct for \ref SHORTPROTOCOL passed to the
00129  * \ref SHORTPROTOCOL Functions
00130  *
00131  * The Callback functions have to all be defined as described in thier
00132  * Documentaion, also a maximumPackageLength has to be defined.
00133  */
00134 typedef struct{
00135     /**
00136      * @brief Callback to read one Byte from the Serial Interface
00137      */
00138     uint8_t (*readByte)(void);
00139
00140     /**
00141      * @brief Callback to check if a Byte is available to be read.
00142      * If this returns SHORTPROTOCOL_AVAILABLE at least one Byte has to be
00143      * readable from readByte.
00144      */
00145     SHORTPROTOCOL_status (*readAvailable)(void);
00146
00147     /**
00148      * @brief Callback to write one Byte to the Serial Interface
00149      */
```

```
00150     void (*writeByte)(uint8_t);
00151
00152     /**
00153      * @brief Callbakc to check if a Byte is writeable to the Serial Interface
00154      * If this returns SHORTPROTOCOL_AVAILABLE at leat one Byte has to be
00155      * writeable to write Byte.
00156      */
00157     SHORTPROTOCOL_status (*writeAvailable)(void);
00158
00159     /**
00160      * @brief The maximum Package Length
00161      * Packets longer than this will be split into Packets with the maximum Size
00162      * of maximumPackageLength.
00163      */
00164     uint32_t maximumPackageLength;
00165
00166     /**
00167      * @brief A Buffer where the Command wich is currently sent resides.
00168      */
00169     SHORTPROTOCOL_string command_buffer;
00170
00171     /**
00172      * @brief Status of the SHORTPROTOCOL Instance itself.
00173      *
00174      */
00175     SHORTPROTOCOL_status newCommand;
00176
00177     /**
00178      * @brief Counter to keep track of what Parts of a Package were already sent
00179      * when it is split to maximumPackageLength.
00180      */
00181     uint32_t writeCounter;
00182
00183 }SHORTPROTOCOL_Instance;
00184
00185 /**
00186  * @brief Function to try to send a new Command using the SHORTPROTOCOL
00187  *
00188  * \param inst SHORTPROTOCOL_Instance
00189  * \param data The Data to be sent using the Shortprotocol
00190  * \param length The Length of the Payload (Data) to be sent
00191  * (without NULL termination)
00192  *
00193  * @return SHORTPROTOCOL_NOT_AVAILABLE when the last Command was not yet sent,
00194  * SHORTPROTOCOL_ERROR when the Command has a length of zero,
00195  * SHORTPROTOCOL_SUCCESS when the Command was successfully queued to be sent.
00196  */
00197 SHORTPROTOCOL_status SHORTPROTOCOL_Send(SHORTPROTOCOL_Instance* inst,
00198         uint8_t* data, uint32_t length);
00199
00200 /**
00201  * @brief Function to be called repeadetly in the Main Loop to send Pieces of
00202  * maximumPackageLength of the Package generated by SHORTPROTOCOL_Send
00203  * @param inst SHORTPROTOCOL_Instance
00204  */
00205 void SHORTPROTOCOL_Update(SHORTPROTOCOL_Instance* inst);
00206
00207 /**
00208  * @brief Returns if a new Command can be sent using the Shortprotocol
00209  * @param inst SHORTPROTOCOL_Instance
00210  * @return SHORTPROTOCOL_AVAILABLE if a new Command can be written using the
00211  * SHORTPROTOCOL_Send Function, SHORTPROTOCOL_NOT_AVAILABLE if the last
00212  * Command is not yet sent.
00213  */
00214 SHORTPROTOCOL_status SHORTPROTOCOL_Available(SHORTPROTOCOL_Instance* inst);
00215
00216 /**
00217  * @brief Initializes an Empty SHORTPROTOCOL_Instance.
00218  * Callback Functions still have to be added/defined and the
00219  * maximumPackageLength has to be set.
00220  * @param inst SHORTPROTOCOL_Instance
00221  */
00222 void SHORTPROTOCOL_Initialize(SHORTPROTOCOL_Instance* inst);
00223
00224 #ifdef  __cplusplus
00225 }
00226 #endif
00227
00228 #endif  /* SHORTPROTOCOL_H */
00229
00230 /**
00231  * \}
00232  */
00233
```

9.26 signals.c File Reference

This File implements the Prototypes for SIGNALS.

```
#include "signals.h"
```

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_result signals_compare_names (uint8_t *first, uint32_t firstlen, uint8_t *second, uint32_t secondlen)
- signals_signal * signals_find_signal (signals_signal *signal_list, uint32_t signal_list_len, void(*Callback)(void))
- signals_signal * signals_find_display_signal (signals_signal *signal_list, uint32_t signal_list_len, uint32_t *dispSignalCount, uint32_t needle)

9.26.1 Detailed Description

This File implements the Prototypes for SIGNALS.

Author

Frederic Emmerth

Definition in file signals.c.

9.27 signals.c

Go to the documentation of this file.

```
00001 /**
00002  * @file signals.c
00003  * @brief This File implements the Prototypes for \ref SIGNALS
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup SIGNALS
00008  *
00009  * \addtogroup SIGNALS
00010  * \{
00011  *
00012  */
00013
00014 #include "signals.h"
00015
00016 void SIGNALS_Interpret(signals_signal* signal_list, uint32_t signal_list_len,
00017     cancomm_message* message_list, uint32_t message_list_len){
00018
00019     uint8_t databuff[CANCOMM_MAXIMUM_DATA_LENGTH];
00020     for(uint32_t i=0; i<signal_list_len; i++){
00021         if(signal_list[i].type == SIGNALS_CAN_MESSAGE){
00022             if(signals_find_data(signal_list[i].id, signal_list[i].interface_number,
00023                     message_list, message_list_len, databuff) == SIGNALS_FOUND)
00024             {
00025                 switch(signal_list[i].data_type){
00026                     case SIGNALS_FLOAT_SIGNAL:
00027                         signal_list[i].value_float =
00028                                 signal_list[i].can_convert_float(databuff);
00029                         break;
00030
```

```
00031                     case SIGNALS_UINT32_T_SIGNAL:
00032                         signal_list[i].value_uint32_t =
00033                                 signal_list[i].can_convert_uint32_t(databuff);
00034                         break;
00035 
00036                     case SIGNALS_STRING_SIGNAL:
00037                         signal_list[i].can_convert_string(databuff,
00038                                 &(signal_list[i].value_string));
00039                         break;
00040 
00041                     default:
00042                         break;
00043                 }
00044             }
00045         }else if(signal_list[i].type == SIGNALS_INTERNAL_SIGNAL){
00046             switch(signal_list[i].data_type){
00047                     case SIGNALS_FLOAT_SIGNAL:
00048                         signal_list[i].value_float =
00049                                 signal_list[i].internal_convert_float(
00050                                         signal_list,signal_list_len);
00051                         break;
00052 
00053                     case SIGNALS_UINT32_T_SIGNAL:
00054                         signal_list[i].value_uint32_t =
00055                                 signal_list[i].internal_convert_uint32_t(
00056                                         signal_list, signal_list_len);
00057                         break;
00058 
00059                     case SIGNALS_STRING_SIGNAL:
00060                         signal_list[i].internal_convert_string(signal_list,
00061                                 signal_list_len, &(signal_list[i].value_string));
00062                         break;
00063 
00064                     default:
00065                         break;
00066             }
00067         }else if(signal_list[i].type == SIGNALS_DISPLAY_SIGNAL){
00068             switch(signal_list[i].data_type){
00069 
00070                     case SIGNALS_STRING_SIGNAL:
00071                         signal_list[i].internal_convert_string(signal_list,
00072                                 signal_list_len, &(signal_list[i].value_string));
00073                         break;
00074 
00075                     default:
00076                         break;
00077             }
00078         }
00079     }
00080 }
00081 
00082 signals_result signals_find_data(uint32_t id, uint8_t interface,
00083         cancomm_message* message_list, uint32_t message_list_len,
00084         uint8_t* data){
00085 
00086     signals_result result = SIGNALS_NOT_FOUND;
00087     for(uint32_t i=0; i<message_list_len; i++){
00088         if(message_list[i].id == id && message_list[i].interface_number == interface){
00089             result = SIGNALS_FOUND;
00090             for(uint8_t j=0; j<CANCOMM_MAXIMUM_DATA_LENGTH; j++){
00091                 data[j] = message_list[i].data[j];
00092             }
00093             break;
00094         }
00095     }
00096 
00097     return result;
00098 }
00099 
00100 signals_result signals_compare_names(uint8_t* first, uint32_t firstlen,
00101         uint8_t* second, uint32_t secondlen){
00102 
00103     uint32_t searchlen;
00104 
00105     /* Set searchlen to the shorter length */
00106     if(firstlen > secondlen){
00107         searchlen = secondlen;
00108     }else if (secondlen > firstlen){
00109         searchlen = firstlen;
00110     }else{
00111         /* The Strings have equal length */
00112         searchlen = firstlen;
00113     }
00114 
00115     signals_result res = SIGNALS_MATCH;
00116 
00117     for(uint32_t i=0; i<searchlen; i++){
```

```
00118         if(first[i] != second[i]){
00119             res = SIGNALS_NO_MATCH;
00120             break;
00121         }
00122     }
00123
00124     return res;
00125 }
00126
00127 signals_signal* signals_find_signal( signals_signal* signal_list,
00128         uint32_t signal_list_len, void(*Callback)(void)){
00129
00130     for(uint32_t i=0; i<signal_list_len; i++){
00131         if(Callback == (void(*)(void))signal_list[i].can_convert_float        ||
00132            Callback == (void(*)(void))signal_list[i].can_convert_uint32_t     ||
00133            Callback == (void(*)(void))signal_list[i].internal_convert_float   ||
00134            Callback == (void(*)(void))signal_list[i].internal_convert_uint32_t||
00135            Callback == (void(*)(void))signal_list[i].internal_convert_string  ||
00136            Callback == (void(*)(void))signal_list[i].can_convert_string        )
00137         {
00138             return &(signal_list[i]);
00139         }
00140     }
00141
00142     return NULL;
00143 }
00144
00145 /* this function returns the total count of display signals in the list, and */
00146 /* if there is at least one display signal, the nth display signal by needle*/
00147
00148 signals_signal* signals_find_display_signal(signals_signal* signal_list,
00149         uint32_t signal_list_len, uint32_t* dispSignalCount, uint32_t needle){
00150
00151     uint32_t counter = 0;
00152     signals_signal* foundNeedle = NULL;
00153     for(uint32_t i=0; i<signal_list_len; i++){
00154         if(signal_list[i].type == SIGNALS_DISPLAY_SIGNAL){
00155             if(needle == counter){
00156                 foundNeedle = &(signal_list[i]);
00157             }
00158             counter++;
00159         }
00160     }
00161
00162     *dispSignalCount = counter;
00163     return foundNeedle;
00164 }
00165
00166
00167 /**
00168  * \}
00169  */
```

9.28 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)

9.28.1 Detailed Description

This File defines the Prototypes for SIGNALS.

Author

Frederic Emmerth

Definition in file signals.h.

9.29 signals.h

Go to the documentation of this file.

```
00001 /**
00002  * @file signals.h
00003  * @brief This File defines the Prototypes for \ref SIGNALS
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup SIGNALS
00008  *
00009  */
00010
00011 /**
00012  * \defgroup SIGNALS SIGNALS
00013  * This Module Handles the Information Interpretation Handling of All Inputs
00014  * to Outputs. Signals have a Type, wich changes how it is interpreted and used.
```

```
00015  * For more on this, refer to \ref signals_data_type and
00016  * \ref signals_signal_type. Signals can be chained together. For Example, four
00017  * Signals of Type SIGNALS_CAN_MESSAGE could recieve four Temperatures, wich
00018  * would then be interpreted to float Values using thier respective Callback
00019  * Functions. Another Signal of Type SIGNALS_INTERNAL_SIGNAL could then find the
00020  * highest of these Temperatures using its Callback Function. To complete the
00021  * Chain, a Signal of Type SIGNALS_DISPLAY_SIGNAL can be used to convert the
00022  * float value of the previous Maximising Function to a String Command wich can
00023  * be interpreted by a Display.
00024  * \addtogroup SIGNALS
00025  * \{
00026  */
00027
00028 #ifndef SIGNALS_H
00029 #define SIGNALS_H
00030
00031 #ifdef  __cplusplus
00032 extern "C" {
00033 #endif
00034
00035 #include "definitions.h"
00036 #include "cancomm.h"
00037
00038 /**
00039  * @brief The maximum friendly name length for a signal
00040  */
00041 #define SIGNALS_MAXIMUM_NAME_LENGTH     30
00042
00043 /**
00044  * @brief The maximum length of the string value of a signal
00045  */
00046 #define SIGNALS_STRING_MAXIMUM_LENGTH   50
00047
00048 /**
00049  * @brief Definition of the Return Values of some Functions of this Module
00050  */
00051 typedef enum{
00052     /**
00053      * Returned if a Signal is found
00054      */
00055     SIGNALS_FOUND = 0,
00056
00057     /**
00058      * Returned if no Signal is found
00059      */
00060     SIGNALS_NOT_FOUND,
00061
00062     /**
00063      * Returned if the Inputs match
00064      */
00065     SIGNALS_MATCH,
00066
00067     /**
00068      * Returned if the Inputs dont match
00069      */
00070     SIGNALS_NO_MATCH
00071 }signals_result;
00072
00073
00074 /**
00075  * @brief The Data type of a Signal
00076  */
00077 typedef enum{
00078     SIGNALS_FLOAT_SIGNAL = 0,
00079     SIGNALS_UINT32_T_SIGNAL,
00080     SIGNALS_STRING_SIGNAL
00081 }signals_data_type;
00082
00083 /**
00084  * @brief The Signal type of a Signal
00085  */
00086 typedef enum{
00087     SIGNALS_CAN_MESSAGE = 0,
00088     SIGNALS_INTERNAL_SIGNAL,
00089     SIGNALS_DISPLAY_SIGNAL
00090 }signals_signal_type;
00091
00092 /**
00093  * \brief A struct to combine a String with it's length
00094  */
00095 typedef struct{
00096     uint32_t length;
00097     uint8_t data[SIGNALS_STRING_MAXIMUM_LENGTH];
00098 }SIGNALS_string;
00099
00100 /**
00101  * Type Definition of signals_signal_struct to be able to use
```

```
00102  * signals_signal in the definition of signals_signal, as signal structs have
00103  * to store pointers to other signals_signal structs.
00104  */
00105 typedef struct signals_signal_struct signals_signal;
00106
00107 /**
00108  * \struct signals_signal_struct
00109  *
00110  * This Struct defines a Signal. Every Signal should have a friendly Name, to be
00111  * easily identified by a human. Every Signal has to have a
00112  * \ref signals_signal.type and a \ref signals_signal.data_type. Depending on
00113  * the \ref signals_signal.type and \ref signals_signal.data_type other
00114  * members of the struct have to be defined. If a signal is of type
00115  * SIGNALS_CAN_MESSAGE, the id and the interface_number have to be defined.
00116  * The Signal can be either of data_type SIGNALS_FLOAT_SIGNAL,
00117  * SIGNALS_UINT32_T_SIGNAL or SIGNALS_STRING_SIGNAL. The corresponding callback
00118  * function can_convert_float, can_convert_uint32_t or can_convert_string has
00119  * to be defined.
00120  *
00121  * If a signal is of type SIGNALS_INTERNAL_SIGNAL, it can be of data_type
00122  * SIGNALS_FLOAT_SIGNAL, SIGNALS_UINT32_T_SIGNAL or SIGNALS_STRING_SIGNAL. The
00123  * corresponding callback function internal_convert_float,
00124  * internal_convert_uint32_t or internal_convert_string has to be defined.
00125  *
00126  * If a signal is of type SIGNALS_DISPLAY_SIGNAL, it's data_type has to be
00127  * SIGNALS_STRING_SIGNAL, and its callback function internal_convert_string
00128  * has to be defined in a way to produce correct Commands for the Display to
00129  * be interpreted. Signals of this Type are read by \ref COMMAND to be sent of
00130  * to the Display using the \ref SHORTPROTOCOL.
00131  *
00132  * A Signal is therefore defined by its type and data_type.
00133  *
00134  * @todo Combine the Different Signal Types to a Union to use less RAM
00135  *
00136  */
00137 struct signals_signal_struct{
00138     /** The ID of the CAN Message */
00139     uint32_t id;
00140
00141     /** The CAN Interface Number (unique Identifier for a CAN Interface)*/
00142     uint8_t interface_number;
00143
00144     /** Type of the Signal */
00145     signals_signal_type type;
00146
00147     /** Data Type of the Signal */
00148     signals_data_type data_type;
00149
00150     /** Callback Function for type=SIGNALS_CAN_MESSAGE and
00151      * data_type=SIGNALS_FLOAT_SIGNAL */
00152     float(*can_convert_float)(uint8_t* data);
00153
00154     /** Callback Function for type=SIGNALS_INTERNAL_SIGNAL and
00155      * data_type=SIGNALS_FLOAT_SIGNAL */
00156     float(*internal_convert_float)(signals_signal* signal_list,
00157             uint32_t signal_list_len);
00158
00159     /** The Float Value of this Signal, only used when
00160      * data_type=SIGNALS_FLOAT_SIGNAL*/
00161     float value_float;
00162
00163     /** Callback Function for type=SIGNALS_CAN_MESSAGE and
00164      * data_type=SIGNALS_UINT32_T_SIGNAL*/
00165     uint32_t(*can_convert_uint32_t)(uint8_t* data);
00166
00167     /** Callback Function for type=SIGNALS_INTERNAL_SIGNAL and
00168      * data_type=SIGNALS_UINT32_T_SIGNAL*/
00169     uint32_t(*internal_convert_uint32_t)(signals_signal* signal_list,
00170             uint32_t signal_list_len);
00171
00172     /** The uint32_t Value of this Signal, only used when
00173      * data_type=SIGNALS_UINT32_T_SIGNAL*/
00174     uint32_t value_uint32_t;
00175
00176     /** Callback Function for type=SIGNALS_CAN_MESSAGE and
00177      * data_type=SIGNALS_STRING_SIGNAL*/
00178     void (*can_convert_string)(uint8_t* data, SIGNALS_string* string);
00179
00180     /** Callback Function for type=SIGNALS_INTERNAL_SIGNAL or
00181      * type=SIGNALS_DISPLAY_SIGNAL. and data_type SIGNALS_STRING_SIGNAL*/
00182     void (*internal_convert_string)(signals_signal* signal_list,
00183             uint32_t signal_list_len, SIGNALS_string* string);
00184
00185     /** The String Value of this Signal, only used when
00186      * data_type=SIGNALS_STRING_SIGNAL*/
00187     SIGNALS_string value_string;
00188
```

```
00189     /** A friendly Name of the Signal, only used for Debugging */
00190     uint8_t friendly_name[SIGNALS_MAXIMUM_NAME_LENGTH];
00191
00192     /** The Timestamp of the Signal */
00193     uint32_t timestamp;
00194 };
00195
00196 /**
00197  * This Function takes a signal_list and a message_list and interprets each
00198  * Signal either from other Signals or from a CAN message. The Callback
00199  * Functions defined in each Signal are called here.
00200  *
00201  * @param signal_list A List of \ref signals_signal Structs
00202  * @param signal_list_len The Length of \p signal_list
00203  * @param message_list A List of \ref cancomm_message Structs
00204  * @param message_list_len The Length of \p signal_list
00205  */
00206 void SIGNALS_Interpret(signals_signal* signal_list, uint32_t signal_list_len,
00207         cancomm_message* message_list, uint32_t message_list_len);
00208
00209 /**
00210  * This function finds the data from a CAN Message and writes it to the Buffer
00211  * \p data
00212  *
00213  * @param id The CAN Message ID
00214  * @param interface The CAN Interface (wich uniquely identifies a CAN Interface)
00215  * @param message_list List of \ref cancomm_message Structs with the data in it
00216  * @param message_list_len The Length of \p message_list
00217  * @param data The Buffer the Data of the requested Message gets written to
00218  * @return A \ref signals_result. Returns SIGNALS_FOUND if the Data of the
00219  * Message was found, and SIGNALS_NOT_FOUND if the Message is not in the List.
00220  */
00221 signals_result signals_find_data(uint32_t id, uint8_t interface,
00222         cancomm_message* message_list, uint32_t message_list_len,
00223         uint8_t* data);
00224
00225 /**
00226  * This function finds a singnal from signal_list by its Callback. This is
00227  * needed for the Conversion Functions of Internal Signals, as they need to
00228  * read other Signal Values in thier Conversion Function.
00229  *
00230  * @param signal_list List of \ref signals_signal to be searched in
00231  * @param signal_list_len The Length of \p signal_list
00232  * @param Callback The Callback Function of the searched signal
00233  * @return Pointer to the Signal if it is found, else NULL is returned
00234  */
00235 signals_signal* signals_find_signal( signals_signal* signal_list,
00236         uint32_t signal_list_len, void(*Callback)(void));
00237
00238 /**
00239  * Function to compare two Signals Names.
00240  * @param first First Name
00241  * @param firstlen Length of \p first
00242  * @param second Second Name
00243  * @param secondlen Length of \p second
00244  * @return SIGNALS_MATCH if the Names match, and SIGNALS_NO_MATCH if they dont
00245  */
00246 signals_result signals_compare_names(uint8_t* first, uint32_t firstlen,
00247         uint8_t* second, uint32_t secondlen);
00248
00249 /**
00250  * Function to return the first, second, third ... display Signal. If needle is
00251  * for example five, the fifth display Signal is returned. Also, the total Count
00252  * of Display Signals is written to \p dispSignalCount.
00253  *
00254  * @param signal_list The signal_list to be searched in.
00255  * @param signal_list_len The Length of \p signal_list
00256  * @param dispSignalCount The total Count of Display Signals in \p signal_list
00257  * is written to this Pointer.
00258  * @param needle The Display Signal to return
00259  * @return Pointer to the display Signal in place of \p needle
00260  */
00261 signals_signal* signals_find_display_signal(signals_signal* signal_list,
00262         uint32_t signal_list_len, uint32_t* dispSignalCount, uint32_t needle);
00263
00264 #ifdef  __cplusplus
00265 }
00266 #endif
00267
00268 #endif  /* SIGNALS_H */
00269
00270 /**
00271  * \}
00272  */
00273
```

9.30 uart.c File Reference

This File Implements the Prototypes for UART.

```
#include "uart.h"
```

Functions

- SHORTPROTOCOL_status UART_ReadAvailable (void)
- uint8_t UART_ReadByte (void)
- SHORTPROTOCOL_status UART_WriteAvailable (void)
- void UART_WriteByte (uint8_t byte)

9.30.1 Detailed Description

This File Implements the Prototypes for UART.

Author

Frederic Emmerth

Definition in file uart.c.

9.31 uart.c

Go to the documentation of this file.

```
00001 /**
00002  * @file uart.c
00003  * @brief This File Implements the Prototypes for \ref UART
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup UART
00008  *
00009  * \addtogroup UART
00010  * \{
00011  *
00012  */
00013
00014
00015 #include "uart.h"
00016
00017 SHORTPROTOCOL_status UART_ReadAvailable( void ){
00018     if(UART3_ReceiverIsReady()){
00019         return SHORTPROTOCOL_AVAILABLE;
00020     }else{
00021         return SHORTPROTOCOL_NOT_AVAILABLE;
00022     }
00023 }
00024
00025 uint8_t UART_ReadByte( void ){
00026     return UART3_ReadByte();
00027 }
00028
00029 SHORTPROTOCOL_status UART_WriteAvailable( void ){
00030     if(UART3_TransmitterIsReady()){
00031         return SHORTPROTOCOL_AVAILABLE;
00032     }else{
00033         return SHORTPROTOCOL_NOT_AVAILABLE;
00034     }
00035 }
00036
00037 void UART_WriteByte(uint8_t byte){
00038     UART3_WriteByte(byte);
00039 }
00040
00041 /**
00042  * \}
00043  */
```

9.32 uart.h File Reference

This File defines the Prototypes for UART.

```
#include "shortprotocol.h"
```

Functions

- SHORTPROTOCOL_status UART_ReadAvailable (void)
- uint8_t UART_ReadByte (void)
- SHORTPROTOCOL_status UART_WriteAvailable (void)
- void UART_WriteByte (uint8_t byte)

9.32.1 Detailed Description

This File defines the Prototypes for UART.

Author

Frederic Emmerth

Definition in file uart.h.

9.33 uart.h

Go to the documentation of this file.

```
00001 /**
00002  * @file uart.h
00003  * @brief This File defines the Prototypes for \ref UART
00004  *
00005  * @author Frederic Emmerth
00006  *
00007  * \ingroup UART
00008  *
00009  */
00010
00011 /**
00012  * \defgroup UART UART
00013  * This Module defines the Interface Functions used by \ref SHORTPROTOCOL.
00014  *
00015  * \addtogroup UART
00016  * \{
00017  */
00018
00019
00020 #ifndef UART_H
00021 #define UART_H
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00027 #include "shortprotocol.h"
00028
00029 /**
00030  * Returns wether or not a Byte can be read from UART
00031  * @return SHORTPROTOCOL_AVAILABLE if a Byte can be read from UART and
00032  * SHORTPROTOCOL_NOT_AVAILABLE if no Byte can be read from UART
00033  */
00034 SHORTPROTOCOL_status UART_ReadAvailable( void );
00035
00036 /**
```

```
00037  * Reads one Byte from UART
00038  * @return The Byte read from UART
00039  */
00040 uint8_t UART_ReadByte( void );
00041
00042 /**
00043  * Returns wether or not a Byte can be written to UART
00044  * @return SHORTPROTOCOL_AVAILABLE if a Byte can be written to UART and
00045  * SHORTPROTOCOL_NOT_AVAILABLE if no Byte can be written to UART
00046  */
00047 SHORTPROTOCOL_status UART_WriteAvailable( void );
00048
00049 /**
00050  * Writes one Byte to UART
00051  */
00052 void UART_WriteByte(uint8_t byte);
00053
00054 #ifdef  __cplusplus
00055 }
00056 #endif
00057
00058 #endif  /* UART_H */
00059
00060 /**
00061  * \}
00062  */
00063
```

Index