Firmware documentation

v. 6.1.5

Generated by Doxygen 1.8.13

Contents

Chapter 1

Firmware

This is the firmware of the SoftHand board.

Version

6.1.5

This is the firmware of the SoftHand board. It can control a motor and read its encoder. Also can read and convert analog measurements connected to the PSoC microcontroller.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 st_calib Struct Reference

Hand calibration structure.

```
#include <globals.h>
```

Data Fields

- uint8 **enabled**
- uint8 **direction**
- int16 **speed**
- int16 **repetitions**

4.1.1 Detailed Description

Hand calibration structure.

4.1.2 Field Documentation

4.1.2.1 direction

```
uint8 direction
```

Direction of motor winding.

4.1.2.2 enabled

```
uint8 enabled
```

Calibration enabling flag.

4.1.2.3 repetitions

```
int16 repetitions
```

Number of cycles of hand closing/opening.

4.1.2.4 speed

```
int16 speed
```

Speed of hand opening/closing.

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

- **globals.h**

4.2 st_data Struct Reference

Data sent/received structure.

```
#include <globals.h>
```

Data Fields

- uint8 **buffer** [128]
- int16 **length**
- int16 **ind**
- uint8 **ready**

4.2.1 Detailed Description

Data sent/received structure.

4.2.2 Field Documentation

4.2.2.1 buffer

```
uint8 buffer[128]
```

Data buffer [CMD | DATA | CHECKSUM].

4.2.2.2 ind

```
int16 ind
```

Data buffer index.

4.2.2.3 length

```
int16 length
```

Data buffer length.

4.2.2.4 ready

```
uint8 ready
```

Data buffer flag to see if the data is ready.

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

- **globals.h**

4.3 st_dev Struct Reference

Device related structure.

```
#include <globals.h>
```

Data Fields

- int32 **tension**
- float **tension_conv_factor**
- int8 **pwm_limit**
- uint8 **tension_valid**

4.3.1 Detailed Description

Device related structure.

4.3.2 Field Documentation

4.3.2.1 pwm_limit

```
int8 pwm_limit
```

Limit on pwm value driven to the motor.

4.3.2.2 tension

```
int32 tension
```

Power supply tension.

4.3.2.3 tension_conv_factor

```
float tension_conv_factor
```

Used to calculate input tension.

4.3.2.4 tension_valid

```
uint8 tension_valid
```

Flag checked if the power supply has a valid value.

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

- **globals.h**

4.4 st_meas Struct Reference

Measurements structure.

```
#include <globals.h>
```

Data Fields

- int32 **pos** [**NUM_OF_SENSORS**]
- int32 **curr** [**NUM_OF_MOTORS**]
- int8 **rot** [**NUM_OF_SENSORS**]
- int32 **emg** [**NUM_OF_EMGS**]
- int32 **vel** [**NUM_OF_SENSORS**]
- int32 **acc** [**NUM_OF_SENSORS**]

4.4.1 Detailed Description

Measurements structure.

4.4.2 Field Documentation

4.4.2.1 acc

`int32 acc[ NUM_OF_SENSORS]`

Encoder rotational acceleration.

4.4.2.2 curr

`int32 curr[ NUM_OF_MOTORS]`

Motor current and current estimation.

4.4.2.3 emg

`int32 emg[ NUM_OF_EMGS]`

EMG sensors values.

4.4.2.4 pos

`int32 pos[ NUM_OF_SENSORS]`

Encoder sensor position.

4.4.2.5 rot

`int8 rot[ NUM_OF_SENSORS]`

Encoder sensor rotations.

4.4.2.6 vel

`int32 vel[ NUM_OF_SENSORS]`

Encoder rotational velocity.

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

- **globals.h**

4.5 st_mem Struct Reference

EEPROM stored structure.

```
#include <globals.h>
```

Data Fields

- uint8 **flag**
- uint8 **id**
- int32 **k_p**
- int32 **k_i**
- int32 **k_d**
- int32 **k_p_c**
- int32 **k_i_c**
- int32 **k_d_c**
- int32 **k_p_dl**
- int32 **k_i_dl**
- int32 **k_d_dl**
- int32 **k_p_c_dl**
- int32 **k_i_c_dl**
- int32 **k_d_c_dl**
- uint8 **activ**
- uint8 **input_mode**
- uint8 **control_mode**
- uint8 **res** [**NUM_OF_SENSORS**]
- int32 **m_off** [**NUM_OF_SENSORS**]
- float **m_mult** [**NUM_OF_SENSORS**]
- uint8 **pos_lim_flag**
- int32 **pos_lim_inf** [**NUM_OF_MOTORS**]
- int32 **pos_lim_sup** [**NUM_OF_MOTORS**]
- int32 **max_step_pos**
- int32 **max_step_neg**
- int16 **current_limit**
- uint16 **emg_threshold** [**NUM_OF_EMGS**]
- uint8 **emg_calibration_flag**
- uint32 **emg_max_value** [**NUM_OF_EMGS**]
- uint8 **emg_speed**
- uint8 **double_encoder_on_off**
- int8 **motor_handle_ratio**
- uint8 **activate_pwm_rescaling**
- float **curr_lookup** [**LOOKUP_DIM**]
- uint8 **baud_rate**
- uint8 **watchdog_period**
- int32 **rest_pos**
- float **rest_delay**
- float **rest_vel**
- float **rest_ratio**
- uint8 **maint_day**
- uint8 **maint_month**
- uint8 **maint_year**
- uint8 **switch_emg**
- uint8 **rest_position_flag**

- uint8 **unused_bytes** [5]
- uint32 **emg_counter** [2]
- uint32 **position_hist** [10]
- uint32 **current_hist** [4]
- uint32 **rest_counter**
- uint32 **wire_disp**
- uint32 **total_time_on**
- uint32 **total_time_rest**

4.5.1 Detailed Description

EEPROM stored structure.

4.5.2 Field Documentation

4.5.2.1 activ

`uint8 activ`

Startup activation.

4.5.2.2 activate_pwm_rescaling

`uint8 activate_pwm_rescaling`

Activation of PWM rescaling for 12V motors.

4.5.2.3 baud_rate

`uint8 baud_rate`

Baud Rate setted.

4.5.2.4 control_mode

`uint8 control_mode`

Motor Control mode.

4.5.2.5 curr_lookup

`float curr_lookup[ LOOKUP_DIM]`

Table of values to get estimated curr.

4.5.2.6 current_hist

```
uint32 current_hist[4]
```

Current histogram - 4 zones.

4.5.2.7 current_limit

```
int16 current_limit
```

Limit for absorbed current.

4.5.2.8 double_encoder_on_off

```
uint8 double_encoder_on_off
```

Double encoder ON/OFF.

4.5.2.9 emg_calibration_flag

```
uint8 emg_calibration_flag
```

Enable emg calibration on startup.

4.5.2.10 emg_counter

```
uint32 emg_counter[2]
```

Counter for EMG activation - both channels.

4.5.2.11 emg_max_value

```
uint32 emg_max_value[ NUM_OF_EMGS]
```

Maximum value for EMG.

4.5.2.12 emg_speed

```
uint8 emg_speed
```

Maximum closure speed when using EMG.

4.5.2.13 emg_threshold

```
uint16 emg_threshold[ NUM_OF_EMGS]
```

Minimum value for activation of EMG control.

4.5.2.14 flag

```
uint8 flag
```

If checked the device has been configured.

4.5.2.15 id

```
uint8 id
```

Device id.

4.5.2.16 input_mode

```
uint8 input_mode
```

Motor Input mode.

4.5.2.17 k_d

```
int32 k_d
```

Position controller derivative constant.

4.5.2.18 k_d_c

```
int32 k_d_c
```

Current controller derivative constant.

4.5.2.19 k_d_c_dl

```
int32 k_d_c_dl
```

Double loop current controller deriv. constant.

4.5.2.20 k_d_dl

```
int32 k_d_dl
```

Double loop position controller deriv. constant.

4.5.2.21 k_i

```
int32 k_i
```

Position controller integrative constant.

4.5.2.22 k_i_c

```
int32 k_i_c
```

Current controller integrative constant.

4.5.2.23 k_i_c_dl

```
int32 k_i_c_dl
```

Double loop current controller integr. constant.

4.5.2.24 k_i_dl

```
int32 k_i_dl
```

Double loop position controller integr. constant.

4.5.2.25 k_p

```
int32 k_p
```

Position controller proportional constant.

4.5.2.26 k_p_c

```
int32 k_p_c
```

Current controller proportional constant.

4.5.2.27 k_p_c_dl

```
int32 k_p_c_dl
```

Double loop current controller prop. constant.

4.5.2.28 k_p_dl

```
int32 k_p_dl
```

Double loop position controller prop. constant.

4.5.2.29 m_mult

```
float m_mult[ NUM_OF_SENSORS]
```

Measurement multiplier.

4.5.2.30 m_off

```
int32 m_off[ NUM_OF_SENSORS]
```

Measurement offset.

4.5.2.31 maint_day

```
uint8 maint_day
```

Day of maintenance.

4.5.2.32 maint_month

```
uint8 maint_month
```

Month of maintenance.

4.5.2.33 maint_year

```
uint8 maint_year
```

Year of maintenance.

4.5.2.34 max_step_neg

```
int32 max_step_neg
```

Maximum number of steps per cycle for negative positions.

4.5.2.35 max_step_pos

```
int32 max_step_pos
```

Maximum number of steps per cycle for positive positions.

4.5.2.36 motor_handle_ratio

```
int8 motor_handle_ratio
```

Discrete multiplier for handle device.

4.5.2.37 pos_lim_flag

```
uint8 pos_lim_flag
```

Position limit active/inactive.

4.5.2.38 pos_lim_inf

```
int32 pos_lim_inf[ NUM_OF_MOTORS]
```

Inferior position limit for motors.

4.5.2.39 pos_lim_sup

```
int32 pos_lim_sup[ NUM_OF_MOTORS]
```

Superior position limit for motors.

4.5.2.40 position_hist

```
uint32 position_hist[10]
```

Positions histogram - 10 zones.

4.5.2.41 res

```
uint8 res[ NUM_OF_SENSORS]
```

Angle resolution.

4.5.2.42 rest_counter

```
uint32 rest_counter
```

Counter for rest position occurrences.

4.5.2.43 rest_delay

```
float rest_delay
```

Hand rest position delay while in EMG mode.

4.5.2.44 rest_pos

```
int32 rest_pos
```

Hand rest position while in EMG mode.

4.5.2.45 rest_position_flag

```
uint8 rest_position_flag
```

Enable rest position feature.

4.5.2.46 rest_ratio

```
float rest_ratio
```

Hand rest ratio between velocity closure and rest position error

4.5.2.47 rest_vel

```
float rest_vel
```

Hand velocity closure for rest position reaching

4.5.2.48 switch_emg

```
uint8 switch_emg
```

EMG opening/closure switch.

4.5.2.49 total_time_on

```
uint32 total_time_on
```

Total time of system power (in seconds).

4.5.2.50 total_time_rest

```
uint32 total_time_rest
```

Total time of system while rest position is maintained.

4.5.2.51 watchdog_period

```
uint8 watchdog_period
```

Watchdog period setted, 255 = disable.

4.5.2.52 wire_disp

```
uint32 wire_disp
```

Counter for total wire displacement measurement.

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

- **globals.h**

4.6 st_ref Struct Reference

Motor Reference structure.

```
#include <globals.h>
```

Data Fields

- int32 **pos** [**NUM_OF_MOTORS**]
- int32 **curr** [**NUM_OF_MOTORS**]
- int32 **pwm** [**NUM_OF_MOTORS**]
- uint8 **onoff**

4.6.1 Detailed Description

Motor Reference structure.

4.6.2 Field Documentation

4.6.2.1 curr

```
int32 curr[ NUM_OF_MOTORS]
```

Motor current reference.

4.6.2.2 onoff

```
uint8 onoff
```

Motor drivers enable.

4.6.2.3 pos

```
int32 pos[ NUM_OF_MOTORS]
```

Motor position reference.

4.6.2.4 pwm

```
int32 pwm[ NUM_OF_MOTORS]
```

Motor direct pwm control.

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

- **globals.h**

Chapter 5

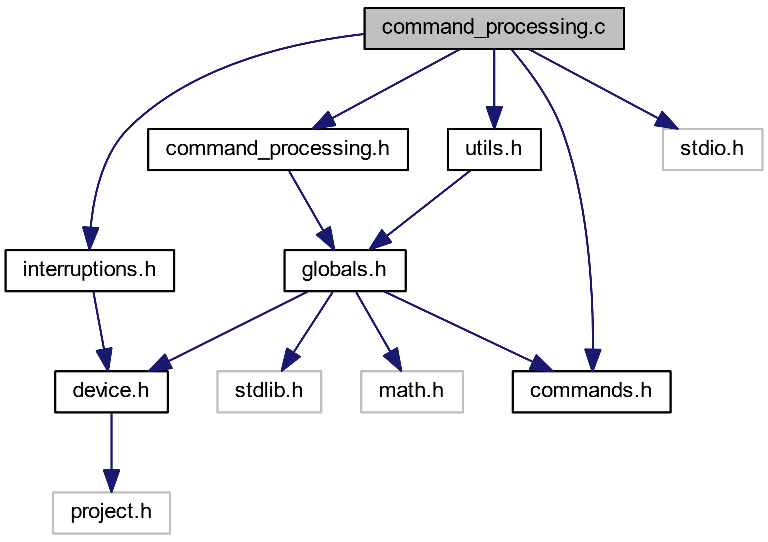
File Documentation

5.1 command_processing.c File Reference

Command processing functions.

```
#include <command_processing.h>
#include <interruptions.h>
#include <stdio.h>
#include <utils.h>
#include "commands.h"
```

Include dependency graph for command_processing.c:



Functions

- void **commProcess** (void)
- void **infoSend** (void)
- void **infoGet** (uint16 info_type)
- void **get_param_list** (uint16 index)
- void **setZeros** ()
- void **prepare_generic_info** (char ∗info_string)
- void **prepare_counter_info** (char ∗info_string)
- void **commWrite_old_id** (uint8 ∗packet_data, uint16 packet_lenght, uint8 old_id)
- void **commWrite** (uint8 ∗packet_data, uint16 packet_lenght)
- void **commWrite_to_cuff** (uint8 ∗packet_data, uint16 packet_lenght)
- uint8 **LCRChecksum** (uint8 ∗data_array, uint8 data_length)
- void **sendAcknowledgment** (uint8 value)
- uint8 **memStore** (int displacement)
- void **memRecall** (void)
- uint8 **memRestore** (void)
- uint8 **memInit** (void)
- void **cmd_get_measurements** ()
- void **cmd_get_velocities** ()
- void **cmd_get_accelerations** ()
- void **cmd_set_inputs** ()
- void **cmd_activate** ()
- void **cmd_get_activate** ()
- void **cmd_get_curr_and_meas** ()
- void **cmd_get_currents** ()
- void **cmd_get_currents_for_cuff** ()
- void **cmd_set_baudrate** ()
- void **cmd_ping** ()
- void **cmd_set_watchdog** ()
- void **cmd_get_inputs** ()
- void **cmd_store_params** ()
- void **cmd_get_emg** ()

Variables

- reg8 ∗ **EEPROM_ADDR** = (reg8 ∗) CYDEV_EE_BASE

5.1.1 Detailed Description

Command processing functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.1.2 Function Documentation

5.1.2.1 cmd_activate()

```
void cmd_activate ( )
```

This function activates the board

5.1.2.2 cmd_get_accelerations()

```
void cmd_get_accelerations ( )
```

This function gets the encoders accelerations and puts them in the package to be sent.

5.1.2.3 cmd_get_activate()

```
void cmd_get_activate ( )
```

This function gets the board activation status and puts it in the package to be sent.

5.1.2.4 cmd_get_curr_and_meas()

```
void cmd_get_curr_and_meas ( )
```

This function gets the motor current and measures and puts it in the package to be sent.

5.1.2.5 cmd_get_currents()

```
void cmd_get_currents ( )
```

This function gets the motor current and puts it in the package to be sent.

5.1.2.6 cmd_get_currents_for_cuff()

```
void cmd_get_currents_for_cuff ( )
```

This function gets the motor current and puts it in the package to be sent to the Cuff device, using the **comm↩ Write_to_cuff** (p. 33) function.

5.1.2.7 cmd_get_emg()

```
void cmd_get_emg ( )
```

This function gets the electromyographic sensors measurements and puts them in the package to be sent.

5.1.2.8 cmd_get_inputs()

```
void cmd_get_inputs ( )
```

This function gets the current motor reference inputs and puts them in the package to be sent.

5.1.2.9 cmd_get_measurements()

```
void cmd_get_measurements ( )
```

Bunch of functions used on request from UART communication

5.1.2.10 cmd_get_velocities()

```
void cmd_get_velocities ( )
```

This function gets the encoders velocities and puts them in the package to be sent.

5.1.2.11 cmd_ping()

```
void cmd_ping ( )
```

This function is used to ping the device and see if is connected.

5.1.2.12 cmd_set_baudrate()

```
void cmd_set_baudrate ( )
```

This function sets the desired communication baudrate. It is possible to select a value equal to 460800 or 2000000.

5.1.2.13 cmd_set_inputs()

```
void cmd_set_inputs ( )
```

This function gets the inputs from the received package and sets them as motor reference.

5.1.2.14 cmd_set_watchdog()

```
void cmd_set_watchdog ( )
```

This function sets the watchdog timer to the one received from the package. The board automatically deactivate when the time equivalent, to watchdog timer, has passed.

5.1.2.15 cmd_store_params()

```
void cmd_store_params ( )
```

This function stores the parameters to the EEPROM memory

5.1.2.16 commProcess()

```
void commProcess ( )
```

This function unpacks the received package, depending on the command received.

5.1.2.17 commWrite()

```
void commWrite (
             uint8 * packet_data,
             uint16 packet_lenght )
```

This function writes on the serial port the package that needs to be sent to the user.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.

5.1.2.18 commWrite_old_id()

```
void commWrite_old_id (
             uint8 * packet_data,
             uint16 packet_lenght,
             uint8 old_id )
```

This function writes on the serial port the package that needs to be sent to the user. Is used only when a new is set, to communicate back to the APIs that the new ID setting went fine or there was an error.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.
old_id	The previous id of the board, before setting a new one.

5.1.2.19 commWrite_to_cuff()

```
void commWrite_to_cuff (
            uint8 * packet_data,
            uint16 packet_lenght )
```

This function writes on the serial port the package that needs to be sent to the Cuff device. It is used only when a specific device is connected to the hand. The Hand must have ID equal to the one of the Cuff plus one.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.

5.1.2.20 get_param_list()

```
void get_param_list (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values and sends them to user or sets a parameter from all the parameters of the device.

Parameters

index	The index of the parameters to be setted. If 0 gets full parameters list.

5.1.2.21 infoGet()

```
void infoGet (
            uint16 info_type )
```

This function sends the firmware information prepared with prepare_general_info or **prepare_counter_info** (p. 36) through the serial port to the user interface. Is used when the ID is specified.

Parameters

info_type	The type of the information needed.

5.1.2.22 infoSend()

```
void infoSend ( )
```

This function sends the firmware information prepared with infoPrepare through the serial port to the user interface. Is used when no ID is specified.

5.1.2.23 LCRChecksum()

```
uint8 LCRChecksum (
             uint8 * data_array,
             uint8 data_length )
```

This function calculates a checksum of the array to see if the received data is consistent.

Parameters

data_array	The array of data that must be checked.
data_lenght	Lenght of the data array that must be checked.

Returns

The calculated checksum for the relative data_array.

5.1.2.24 memInit()

```
uint8 memInit ( )
```

This functions initializes the memory. It is used also to restore the the parameters to their default values.

Returns

A true value if the memory is correctly initialized, false otherwise.

5.1.2.25 memRecall()

```
void memRecall ( )
```

This function loads user's settings from the EEPROM.

5.1.2.26 memRestore()

```
uint8 memRestore ( )
```

This function loads default settings from the EEPROM.

Returns

A true value if the memory is correctly restored, false otherwise.

5.1.2.27 memStore()

```
uint8 memStore (
             int displacement )
```

This function stores the setted parameters to the internal EEPROM memory. It is usually called, by the user, after a parameter is set.

Parameters

displacement	The address where the parameters will be written.

Returns

A true value if the memory is correctly stored, false otherwise.

5.1.2.28 prepare_counter_info()

```
void prepare_counter_info (
            char * info_string )
```

This function is used to prepare an information string about the cycles counter of the hand

Parameters

info_string	An array of chars containing the requested informations.

5.1.2.29 prepare_generic_info()

```
void prepare_generic_info (
            char * info_string )
```

This function is used to prepare a generic information string on the device parameters and measurements

Parameters

info_string	An array of chars containing the requested informations.

5.1.2.30 sendAcknowledgment()

```
void sendAcknowledgment (
            uint8 value )
```

This functions sends an acknowledgment to see if a command has been executed properly or not.

Parameters

value	An ACK_OK(1) or ACK_ERROR(0) value.

5.1.2.31 setZeros()

```
void setZeros ( )
```

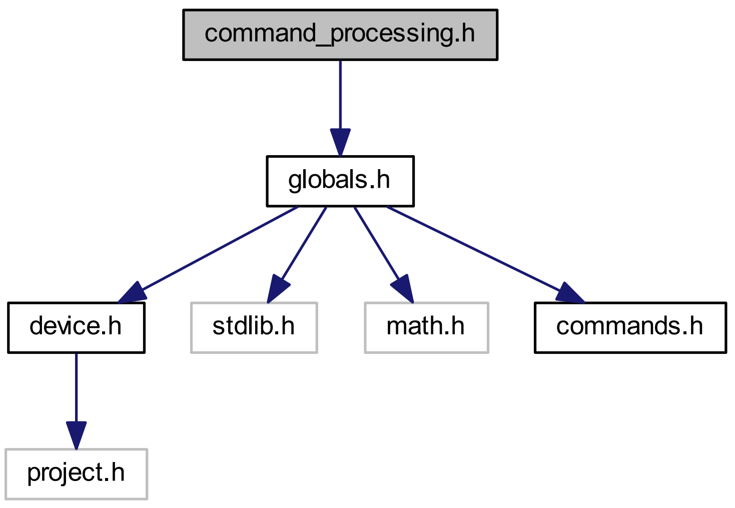
This function sets the encoders zero position.

5.2 command_processing.h File Reference

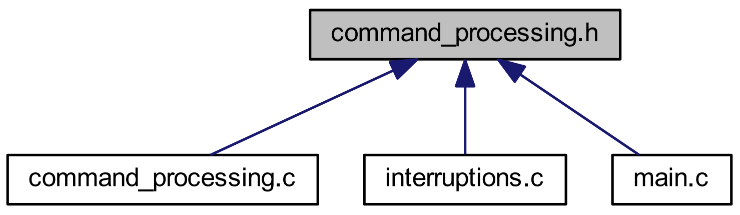
Received commands processing functions.

```
#include <globals.h>
```

Include dependency graph for command_processing.h:



This graph shows which files directly or indirectly include this file:



Functions

Firmware information functions

- void **prepare_generic_info** (char ∗info_string)
- void **prepare_counter_info** (char ∗info_string)
- void **infoSend** ()
- void **infoGet** (uint16 info_type)

Command receiving and sending functions

- void **commProcess** ()
- void **commWrite_old_id** (uint8 ∗packet_data, uint16 packet_lenght, uint8 old_id)
- void **commWrite** (uint8 ∗packet_data, uint16 packet_lenght)
- void **commWrite_to_cuff** (uint8 ∗packet_data, uint16 packet_lenght)

Memory management functions

- void **get_param_list** (uint16 index)
- void **setZeros** ()
- uint8 **memStore** (int displacement)
- void **memRecall** ()
- uint8 **memRestore** ()
- uint8 **memInit** ()

Utility functions

- uint8 **LCRChecksum** (uint8 ∗data_array, uint8 data_length)
- void **sendAcknowledgment** (uint8 value)

Command processing functions

- void **cmd_activate** ()
- void **cmd_set_inputs** ()
- void **cmd_get_measurements** ()
- void **cmd_get_velocities** ()
- void **cmd_get_accelerations** ()
- void **cmd_get_currents** ()
- void **cmd_get_curr_and_meas** ()
- void **cmd_get_currents_for_cuff** ()
- void **cmd_get_emg** ()
- void **cmd_set_watchdog** ()
- void **cmd_get_activate** ()
- void **cmd_set_baudrate** ()
- void **cmd_get_inputs** ()
- void **cmd_store_params** ()
- void **cmd_ping** ()

5.2.1 Detailed Description

Received commands processing functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

This file contains all the definitions of the functions used to process the commands sent from the user interfaces (simulink, command line, GUI)

5.2.2 Function Documentation

5.2.2.1 cmd_activate()

```
void cmd_activate ( )
```

This function activates the board

5.2.2.2 cmd_get_accelerations()

```
void cmd_get_accelerations ( )
```

This function gets the encoders accelerations and puts them in the package to be sent.

5.2.2.3 cmd_get_activate()

```
void cmd_get_activate ( )
```

This function gets the board activation status and puts it in the package to be sent.

5.2.2.4 cmd_get_curr_and_meas()

```
void cmd_get_curr_and_meas ( )
```

This function gets the motor current and measures and puts it in the package to be sent.

5.2.2.5 cmd_get_currents()

```
void cmd_get_currents ( )
```

This function gets the motor current and puts it in the package to be sent.

5.2.2.6 cmd_get_currents_for_cuff()

```
void cmd_get_currents_for_cuff ( )
```

This function gets the motor current and puts it in the package to be sent to the Cuff device, using the **comm↩ Write_to_cuff** (p. 33) function.

5.2.2.7 cmd_get_emg()

```
void cmd_get_emg ( )
```

This function gets the electromyographic sensors measurements and puts them in the package to be sent.

5.2.2.8 cmd_get_inputs()

```
void cmd_get_inputs ( )
```

This function gets the current motor reference inputs and puts them in the package to be sent.

5.2.2.9 cmd_get_measurements()

```
void cmd_get_measurements ( )
```

This function gets the encoders measurements and puts them in the package to be sent.

Bunch of functions used on request from UART communication

5.2.2.10 cmd_get_velocities()

```
void cmd_get_velocities ( )
```

This function gets the encoders velocities and puts them in the package to be sent.

5.2.2.11 cmd_ping()

```
void cmd_ping ( )
```

This function is used to ping the device and see if is connected.

5.2.2.12 cmd_set_baudrate()

```
void cmd_set_baudrate ( )
```

This function sets the desired communication baudrate. It is possible to select a value equal to 460800 or 2000000.

5.2.2.13 cmd_set_inputs()

```
void cmd_set_inputs ( )
```

This function gets the inputs from the received package and sets them as motor reference.

5.2.2.14 cmd_set_watchdog()

```
void cmd_set_watchdog ( )
```

This function sets the watchdog timer to the one received from the package. The board automatically deactivate when the time equivalent, to watchdog timer, has passed.

5.2.2.15 cmd_store_params()

```
void cmd_store_params ( )
```

This function stores the parameters to the EEPROM memory

5.2.2.16 commProcess()

```
void commProcess ( )
```

This function unpacks the received package, depending on the command received.

5.2.2.17 commWrite()

```
void commWrite (
             uint8 * packet_data,
             uint16 packet_lenght )
```

This function writes on the serial port the package that needs to be sent to the user.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.

5.2.2.18 commWrite_old_id()

```
void commWrite_old_id (
             uint8 * packet_data,
             uint16 packet_lenght,
             uint8 old_id )
```

This function writes on the serial port the package that needs to be sent to the user. Is used only when a new is set, to communicate back to the APIs that the new ID setting went fine or there was an error.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.
old_id	The previous id of the board, before setting a new one.

5.2.2.19 commWrite_to_cuff()

```
void commWrite_to_cuff (
            uint8 * packet_data,
            uint16 packet_lenght )
```

This function writes on the serial port the package that needs to be sent to the Cuff device. It is used only when a specific device is connected to the hand. The Hand must have ID equal to the one of the Cuff plus one.

Parameters

packet_data	The array of data that must be written.
packet_lenght	The lenght of the data array.

5.2.2.20 get_param_list()

```
void get_param_list (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values and sends them to user or sets a parameter from all the parameters of the device.

Parameters

index	The index of the parameters to be setted. If 0 gets full parameters list.

5.2.2.21 infoGet()

```
void infoGet (
            uint16 info_type )
```

This function sends the firmware information prepared with prepare_general_info or **prepare_counter_info** (p. 36) through the serial port to the user interface. Is used when the ID is specified.

Parameters

info_type	The type of the information needed.

5.2.2.22 infoSend()

```
void infoSend ( )
```

This function sends the firmware information prepared with infoPrepare through the serial port to the user interface. Is used when no ID is specified.

5.2.2.23 LCRChecksum()

```
uint8 LCRChecksum (
            uint8 * data_array,
            uint8 data_length )
```

This function calculates a checksum of the array to see if the received data is consistent.

Parameters

data_array	The array of data that must be checked.
data_lenght	Lenght of the data array that must be checked.

Returns

The calculated checksum for the relative data_array.

5.2.2.24 memInit()

```
uint8 memInit ( )
```

This functions initializes the memory. It is used also to restore the the parameters to their default values.

Returns

A true value if the memory is correctly initialized, false otherwise.

5.2.2.25 memRecall()

```
void memRecall ( )
```

This function loads user's settings from the EEPROM.

5.2.2.26 memRestore()

```
uint8 memRestore ( )
```

This function loads default settings from the EEPROM.

Returns

A true value if the memory is correctly restored, false otherwise.

5.2.2.27 memStore()

```
uint8 memStore (
            int displacement )
```

This function stores the setted parameters to the internal EEPROM memory. It is usually called, by the user, after a parameter is set.

Parameters

displacement	The address where the parameters will be written.

Returns

A true value if the memory is correctly stored, false otherwise.

5.2.2.28 prepare_counter_info()

```
void prepare_counter_info (
            char * info_string )
```

This function is used to prepare an information string about the cycles counter of the hand

Parameters

info_string	An array of chars containing the requested informations.

5.2.2.29 prepare_generic_info()

```
void prepare_generic_info (
            char * info_string )
```

This function is used to prepare a generic information string on the device parameters and measurements

Parameters

info_string	An array of chars containing the requested informations.

5.2.2.30 sendAcknowledgment()

```
void sendAcknowledgment (
            uint8 value )
```

This functions sends an acknowledgment to see if a command has been executed properly or not.

Parameters

value	An ACK_OK(1) or ACK_ERROR(0) value.

5.2.2.31 setZeros()

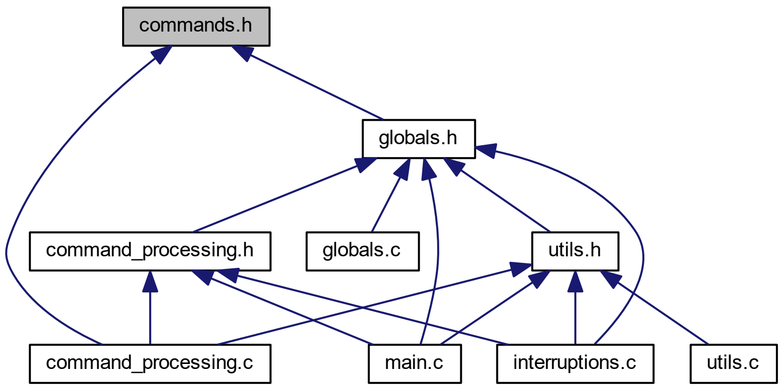
```
void setZeros ( )
```

This function sets the encoders zero position.

5.3 commands.h File Reference

Definitions for SoftHand commands, parameters and packages.

This graph shows which files directly or indirectly include this file:



Macros

QB Move Information Strings

- #define **INFO_ALL** 0
  Generic device informations.
- #define **CYCLES_INFO** 1
  Cycles counter informations.

Enumerations

qbMove and qbHand Commands

- enum **qbmove_command** {
  CMD_PING = 0, **CMD_SET_ZEROS** = 1, **CMD_STORE_PARAMS** = 3, **CMD_STORE_DEFAULT_P↩
  ARAMS** = 4,
  CMD_RESTORE_PARAMS = 5, **CMD_GET_INFO** = 6, **CMD_SET_VALUE** = 7, **CMD_GET_VALUE** =
  8,
  CMD_BOOTLOADER = 9, **CMD_INIT_MEM** = 10, **CMD_CALIBRATE** = 11, **CMD_GET_PARAM_LIST**
  = 12,
  CMD_HAND_CALIBRATE = 13, **CMD_ACTIVATE** = 128, **CMD_GET_ACTIVATE** = 129, **CMD_SET↩
  _INPUTS** = 130,
  CMD_GET_INPUTS = 131, **CMD_GET_MEASUREMENTS** = 132, **CMD_GET_CURRENTS** = 133, **C↩
  MD_GET_CURR_AND_MEAS** = 134,
  CMD_SET_POS_STIFF = 135, **CMD_GET_EMG** = 136, **CMD_GET_VELOCITIES** = 137, **CMD_GET↩
  _COUNTERS** = 138,
  CMD_GET_ACCEL = 139, **CMD_GET_CURR_DIFF** = 140, **CMD_SET_CURR_DIFF** = 141, **CMD_S↩
  ET_CUFF_INPUTS** = 142,
  CMD_SET_WATCHDOG = 143, **CMD_SET_BAUDRATE** = 144, **CMD_EXT_DRIVE** = 145, **CMD_G↩
  ET_JOYSTICK** = 146 }

qbMove and qbHand Parameters

- #define **PARAM_BYTE_SLOT** 50

  Number of bytes reserved to a param information.
- #define **PARAM_MENU_SLOT** 150

  Number of bytes reserved to a param menu.
- enum **qbmove_parameter** {
  PARAM_ID = 0, **PARAM_PID_CONTROL** = 1, **PARAM_STARTUP_ACTIVATION** = 2, **PARAM_INPU↩
  T_MODE** = 3,
  PARAM_CONTROL_MODE = 4, **PARAM_MEASUREMENT_OFFSET** = 5, **PARAM_MEASUREMENT↩
  _MULTIPLIER** = 6, **PARAM_POS_LIMIT_FLAG** = 7,
  PARAM_POS_LIMIT = 8, **PARAM_MAX_STEP_POS** = 9, **PARAM_MAX_STEP_NEG** = 10, **PARAM_↩
  POS_RESOLUTION** = 11,
  PARAM_CURRENT_LIMIT = 12, **PARAM_EMG_CALIB_FLAG** = 13, **PARAM_EMG_THRESHOLD** = 14,
  PARAM_EMG_MAX_VALUE = 15,
  PARAM_EMG_SPEED = 16, **PARAM_PID_CURR_CONTROL** = 18, **PARAM_DOUBLE_ENC_ON_OFF**
  = 19, **PARAM_MOT_HANDLE_RATIO** = 20,
  PARAM_MOTOR_SUPPLY = 21, **PARAM_CURRENT_LOOKUP** = 23, **PARAM_DL_POS_PID** = 24, **P↩
  ARAM_DL_CURR_PID** = 25 }
- enum **qbmove_resolution** {
  RESOLUTION_360 = 0, **RESOLUTION_720** = 1, **RESOLUTION_1440** = 2, **RESOLUTION_2880** = 3,
  RESOLUTION_5760 = 4, **RESOLUTION_11520** = 5, **RESOLUTION_23040** = 6, **RESOLUTION_46080** = 7,
  RESOLUTION_92160 = 8 }
- enum **qbmove_input_mode** {
  INPUT_MODE_EXTERNAL = 0, **INPUT_MODE_ENCODER3** = 1, **INPUT_MODE_EMG_PROPORTION↩
  AL** = 2, **INPUT_MODE_EMG_INTEGRAL** = 3,
  INPUT_MODE_EMG_FCFS = 4, **INPUT_MODE_EMG_FCFS_ADV** = 5 }
- enum **qbmove_control_mode** { **CONTROL_ANGLE** = 0, **CONTROL_PWM** = 1, **CONTROL_CURRENT**
  = 2, **CURR_AND_POS_CONTROL** = 3 }
- enum **motor_supply_tipe** { **MAXON_24V** = 0, **MAXON_12V** = 1 }
- enum **acknowledgment_values** { **ACK_ERROR** = 0, **ACK_OK** = 1 }
- enum **data_types** {
  TYPE_FLAG = 0, **TYPE_INT8** = 1, **TYPE_UINT8** = 2, **TYPE_INT16** = 3,
  TYPE_UINT16 = 4, **TYPE_INT32** = 5, **TYPE_UINT32** = 6, **TYPE_FLOAT** = 7,
  TYPE_DOUBLE = 8 }

5.3.1 Detailed Description

Definitions for SoftHand commands, parameters and packages.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

This file is included in the firmware, in its libraries and applications. It contains all definitions that are necessary for the contruction of communication packages.

It includes definitions for all of the device commands, parameters and also the size of answer packages.

5.3.2 Enumeration Type Documentation

5.3.2.1 qbmove_command

`enum qbmove_command`

Enumerator

CMD_PING	Asks for a ping message.
CMD_SET_ZEROS	Command for setting the encoders zero position.
CMD_STORE_PARAMS	Stores all parameters in memory and loads them.
CMD_STORE_DEFAULT_PARAMS	Store current parameters as factory parameters.
CMD_RESTORE_PARAMS	Restore default factory parameters.
CMD_GET_INFO	Asks for a string of information about.
CMD_SET_VALUE	Not Used.
CMD_GET_VALUE	Not Used.
CMD_BOOTLOADER	Sets the bootloader modality to update the firmware.
CMD_INIT_MEM	Initialize the memory with the defalut values.
CMD_CALIBRATE	Starts the stiffness calibration of the qbMove or the hand closure and opening calibration.
CMD_GET_PARAM_LIST	Command to get the parameters list or to set a defined value chosen by the use.
CMD_HAND_CALIBRATE	Starts a series of opening and closures of the hand.
CMD_ACTIVATE	Command for activating/deactivating the device.
CMD_GET_ACTIVATE	Command for getting device activation state.

Enumerator

CMD_SET_INPUTS	Command for setting reference inputs.
CMD_GET_INPUTS	Command for getting reference inputs.
CMD_GET_MEASUREMENTS	Command for asking device's position measurements.
CMD_GET_CURRENTS	Command for asking device's current measurements.
CMD_GET_CURR_AND_MEAS	Command for asking device's measurements and currents.
CMD_SET_POS_STIFF	Not used in the softhand firmware.
CMD_GET_EMG	Command for asking device's emg sensors measurements.
CMD_GET_VELOCITIES	Command for asking device's velocity measurements.
CMD_GET_COUNTERS	Command for asking device's counters (mostly used for debugging sent commands).
CMD_GET_ACCEL	Command for asking device's acceleration measurements.
CMD_GET_CURR_DIFF	Command for asking device's current difference between a measured one and an estimated one (Only for SoftHand).
CMD_SET_CURR_DIFF	Command used to set current difference modality (Only for Cuff device).
CMD_SET_CUFF_INPUTS	Command used to set Cuff device inputs (Only for Cuff device).
CMD_SET_WATCHDOG	Command for setting watchdog timer or disable it.
CMD_SET_BAUDRATE	Command for setting baudrate of communication.
CMD_EXT_DRIVE	Command to set the actual measurements as inputs to another device (Only for Armslider device).
CMD_GET_JOYSTICK	Command to get the joystick measurements (Only for devices driven by a joystick).

5.3.2.2 qbmove_control_mode

enum **qbmove_control_mode**

Enumerator

CONTROL_ANGLE	Classic position control.
CONTROL_PWM	Direct PWM value.
CONTROL_CURRENT	Current control.
CURR_AND_POS_CONTROL	Current and position control.

5.3.2.3 qbmove_input_mode

enum **qbmove_input_mode**

Enumerator

INPUT_MODE_EXTERNAL	References through external commands (default).
INPUT_MODE_ENCODER3	Encoder 3 drives all inputs.

Enumerator

INPUT_MODE_EMG_PROPORTIONAL	Use EMG measure to proportionally drive the position of the motor 1.
INPUT_MODE_EMG_INTEGRAL	Use 2 EMG signals to drive motor position.
INPUT_MODE_EMG_FCFS	Use 2 EMG. First reaching threshold wins and its value defines hand closure.
INPUT_MODE_EMG_FCFS_ADV	Use 2 EMG. First reaching threshold wins and its value defines hand closure Wait for both EMG to lower under threshold.

5.3.2.4 qbmove_parameter

`enum` **`qbmove_parameter`**

Enumerator

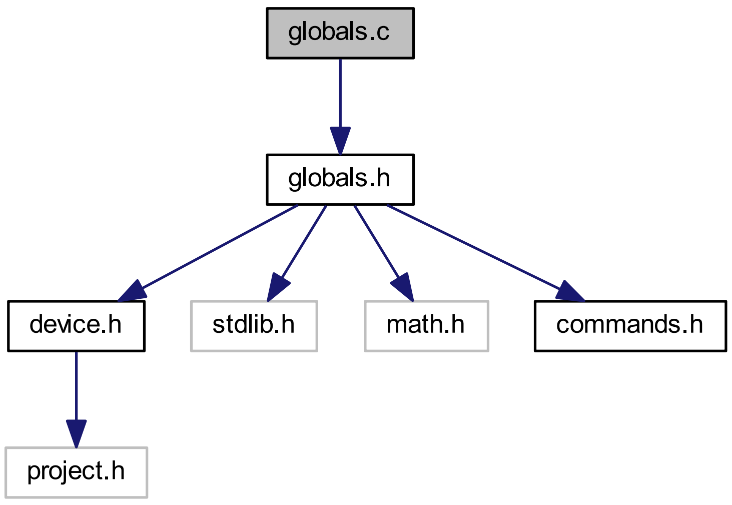
PARAM_ID	Device's ID number.
PARAM_PID_CONTROL	PID parameters.
PARAM_STARTUP_ACTIVATION	Start up activation byte.
PARAM_INPUT_MODE	Input mode.
PARAM_CONTROL_MODE	Choose the kind of control between position control, current control, direct PWM value or current+position control.
PARAM_MEASUREMENT_OFFSET	Adds a constant offset to the measurements.
PARAM_MEASUREMENT_MULTIPLIER	Adds a multiplier to the measurements.
PARAM_POS_LIMIT_FLAG	Enable/disable position limiting.
PARAM_POS_LIMIT	Position limit values | int32 | int32 | int32 | int32 | | INF_LIM_1 | SUP_LIM_1 | INF_LIM_2 | SUP_LIM_2 |
PARAM_MAX_STEP_POS	Used to slow down movements for positive values.
PARAM_MAX_STEP_NEG	Used to slow down movements for negative values.
PARAM_POS_RESOLUTION	Angle resolution for inputs and measurements. Used during communication.
PARAM_CURRENT_LIMIT	Limit for absorbed current.
PARAM_EMG_CALIB_FLAG	Enable calibration on startup.
PARAM_EMG_THRESHOLD	Minimum value to have effect.
PARAM_EMG_MAX_VALUE	Maximum value of EMG.
PARAM_EMG_SPEED	Closure speed when using EMG.
PARAM_PID_CURR_CONTROL	PID current control.
PARAM_DOUBLE_ENC_ON_OFF	Double Encoder Y/N.
PARAM_MOT_HANDLE_RATIO	Multiplier between handle and motor.
PARAM_MOTOR_SUPPLY	Motor supply voltage of the hand.
PARAM_CURRENT_LOOKUP	Table of values used to calculate an estimated current of the SoftHand.
PARAM_DL_POS_PID	Double loop position PID.
PARAM_DL_CURR_PID	Double loop current PID.

5.4 globals.c File Reference

Global variables.

`#include <globals.h>`

Include dependency graph for globals.c:



Variables

- struct **st_ref** g_ref g_refNew **g_refOld**
- struct **st_meas** g_meas **g_measOld**
- struct **st_data** **g_rx**
- struct **st_mem** g_mem **c_mem**
- struct **st_calib** **calib**
- float **tau_feedback**
- uint32 **timer_value**
- uint32 **timer_value0**
- float **cycle_time**
- int32 **dev_tension**
- uint8 **dev_pwm_limit**
- uint8 **dev_pwm_sat** = 100
- int32 **dev_tension_f**
- int32 **pow_tension**
- **counter_status** CYDATA **cycles_status** = **NONE**
- **emg_status** CYDATA **emg_1_status** = **RESET**
- **emg_status** CYDATA **emg_2_status** = **RESET**
- CYBIT **reset_last_value_flag**
- CYBIT **tension_valid**
- CYBIT **interrupt_flag** = FALSE
- CYBIT **watchdog_flag** = FALSE
- CYBIT **cycles_interrupt_flag** = FALSE

- CYBIT **can_write** = TRUE
- uint8 **rest_enabled**
- uint8 **forced_open**
- int16 **ADC_buf** [4]
- int8 **pwm_sign**
- int32 **rest_pos_curr_ref**

5.4.1 Detailed Description

Global variables.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.4.2 Variable Documentation

5.4.2.1 c_mem

```
struct st_mem g_mem c_mem
```

Memory parameters.

5.4.2.2 calib

```
struct st_calib calib
```

Calibration variables.

5.4.2.3 can_write

```
CYBIT can_write = TRUE
```

Write to EEPROM flag.

5.4.2.4 cycle_time

```
float cycle_time
```

Variable used to calculate how much time a cycle takes.

5.4.2.5 cycles_interrupt_flag

```
CYBIT cycles_interrupt_flag = FALSE
```

Cycles timer interrupt flag enabler.

5.4.2.6 cycles_status

```
counter_status CYDATA cycles_status = NONE
```

Cycles counter state machine status.

5.4.2.7 dev_pwm_limit

```
uint8 dev_pwm_limit
```

Device pwm limit. It may change during execution.

5.4.2.8 dev_pwm_sat

```
uint8 dev_pwm_sat = 100
```

Device pwm saturation. By default the saturation value must not exceed 100.

5.4.2.9 dev_tension

```
int32 dev_tension
```

Power supply tension.

5.4.2.10 dev_tension_f

```
int32 dev_tension_f
```

Filtered power supply tension.

5.4.2.11 emg_1_status

```
emg_status CYDATA emg_1_status = RESET
```

First EMG sensor status.

5.4.2.12 emg_2_status

```
emg_status CYDATA emg_2_status = RESET
```

Second EMG sensor status.

5.4.2.13 forced_open

```
uint8 forced_open
```

Forced open flag (used in position with rest position control).

5.4.2.14 g_measOld

```
struct st_meas g_meas g_measOld
```

Measurements.

5.4.2.15 g_refOld

```
struct st_ref g_ref g_refNew g_refOld
```

Reference variables.

5.4.2.16 g_rx

```
struct st_data g_rx
```

Incoming/Outcoming data.

5.4.2.17 interrupt_flag

```
CYBIT interrupt_flag = FALSE
```

Interrupt flag enabler.

5.4.2.18 pow_tension

```
int32 pow_tension
```

Computed power supply tension.

5.4.2.19 pwm_sign

```
int8 pwm_sign
```

ADC measurements buffer. Sign of pwm driven. Used to obtain current sign.

5.4.2.20 reset_last_value_flag

```
CYBIT reset_last_value_flag
```

This flag is set when the encoders last values must be resetted.

5.4.2.21 rest_enabled

```
uint8 rest_enabled
```

Rest position flag.

5.4.2.22 rest_pos_curr_ref

```
int32 rest_pos_curr_ref
```

Rest position current reference.

5.4.2.23 tau_feedback

```
float tau_feedback
```

Torque feedback.

5.4.2.24 tension_valid

```
CYBIT tension_valid
```

Tension validation bit.

5.4.2.25 timer_value

```
uint32 timer_value
```

End time of the firmware main loop.

5.4.2.26 timer_value0

```
uint32 timer_value0
```

Start time of the firmware main loop.

5.4.2.27 watchdog_flag

```
CYBIT watchdog_flag = FALSE
```

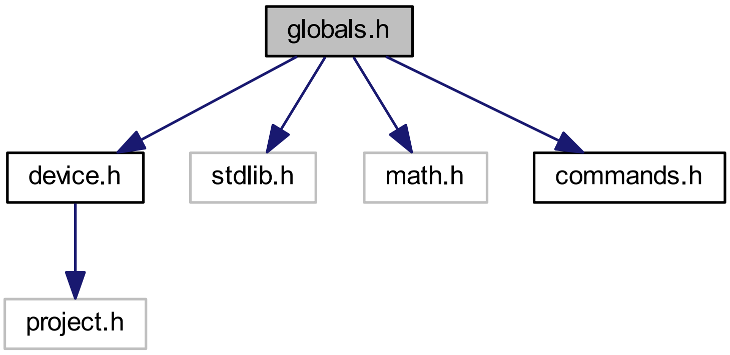
Watchdog flag enabler.

5.5 globals.h File Reference

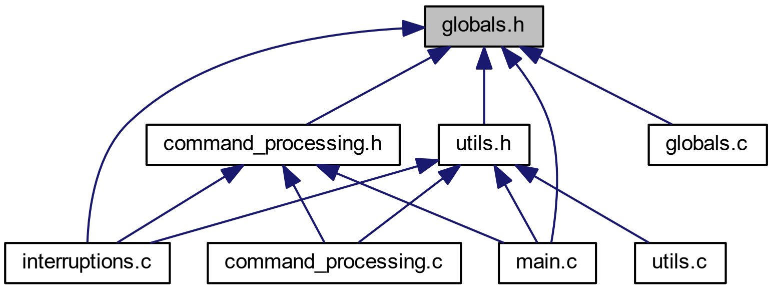
Global definitions and macros are set in this file.

```
#include <device.h>
#include "stdlib.h"
#include "math.h"
#include "commands.h"
```

Include dependency graph for globals.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **st_ref**

  Motor Reference structure.
- struct **st_meas**

  Measurements structure.
- struct **st_data**

Data sent/received structure.
- struct **st_mem**

  EEPROM stored structure.
- struct **st_dev**

  Device related structure.
- struct **st_calib**

  Hand calibration structure.

Macros

- #define **VERSION** "SH-PRO v6.1.6 Centro Piaggio"
- #define **NUM_OF_MOTORS** 2
- #define **NUM_OF_SENSORS** 3
- #define **NUM_OF_EMGS** 2
- #define **NUM_OF_ANALOG_INPUTS** 4
- #define **NUM_OF_PARAMS** 29
- #define **CALIBRATION_DIV** 10
- #define **DIV_INIT_VALUE** 1
- #define **DMA_BYTES_PER_BURST** 2
- #define **DMA_REQUEST_PER_BURST** 1
- #define **DMA_SRC_BASE** (CYDEV_PERIPH_BASE)
- #define **DMA_DST_BASE** (CYDEV_SRAM_BASE)
- #define **WAIT_START** 0
- #define **WAIT_ID** 1
- #define **WAIT_LENGTH** 2
- #define **RECEIVE** 3
- #define **UNLOAD** 4
- #define **STATE_INACTIVE** 0
- #define **STATE_ACTIVE** 1
- #define **COUNTER_INC** 2
- #define **FALSE** 0
- #define **TRUE** 1
- #define **DEFAULT_EEPROM_DISPLACEMENT** 50
- #define **EEPROM_BYTES_ROW** 16
- #define **EEPROM_COUNTERS_ROWS** 5
- #define **MAX_WATCHDOG_TIMER** 250
- #define **PWM_MAX_VALUE** 100
- #define **ANTI_WINDUP** 1000
- #define **DEFAULT_CURRENT_LIMIT** 1500
- #define **CURRENT_HYSTERESIS** 10
- #define **EMG_SAMPLE_TO_DISCARD** 500
- #define **SAMPLES_FOR_MEAN** 100
- #define **SAMPLES_FOR_EMG_MEAN** 1000
- #define **CALIB_DECIMATION** 1
- #define **NUM_OF_CLOSURES** 5
- #define **POS_INTEGRAL_SAT_LIMIT** 50000000
- #define **CURR_INTEGRAL_SAT_LIMIT** 100000
- #define **MIN_CURR_SAT_LIMIT** 30
- #define **LOOKUP_DIM** 6

Enumerations

- enum **emg_status** {
  NORMAL = 0, **RESET** = 1, **DISCARD** = 2, **SUM_AND_MEAN** = 3,
  WAIT = 4, **WAIT_EoC** = 5 }
- enum **counter_status** {
  PREPARE_DATA = 0, **WRITE_CYCLES** = 1, **WAIT_QUERY** = 2, **WRITE_END** = 3,
  NONE = 4 }

Variables

- struct **st_ref** g_ref g_refNew **g_refOld**
- struct **st_meas** g_meas **g_measOld**
- struct **st_data** **g_rx**
- struct **st_mem** g_mem **c_mem**
- struct **st_calib** **calib**
- uint32 **timer_value**
- uint32 **timer_value0**
- float **cycle_time**
- int32 **dev_tension**
- uint8 **dev_pwm_sat**
- uint8 **dev_pwm_limit**
- int32 **dev_tension_f**
- int32 **pow_tension**
- CYBIT **reset_last_value_flag**
- CYBIT **tension_valid**
- CYBIT **interrupt_flag**
- CYBIT **watchdog_flag**
- float **tau_feedback**
- CYBIT **cycles_interrupt_flag**
- CYBIT **can_write**
- uint8 **rest_enabled**
- uint8 **forced_open**
- **counter_status** CYDATA **cycles_status**
- **emg_status** CYDATA **emg_1_status**
- **emg_status** CYDATA **emg_2_status**
- int16 **ADC_buf** [4]
- int8 **pwm_sign**
- int32 **rest_pos_curr_ref**

5.5.1 Detailed Description

Global definitions and macros are set in this file.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.5.2 Macro Definition Documentation

5.5.2.1 ANTI_WINDUP

```
#define ANTI_WINDUP 1000
```

Anti windup saturation.

5.5.2.2 CALIBRATION_DIV

```
#define CALIBRATION_DIV 10
```

Frequency divisor for hand calibration (100Hz).

5.5.2.3 COUNTER_INC

```
#define COUNTER_INC 2
```

Counter cycle increment.

5.5.2.4 CURR_INTEGRAL_SAT_LIMIT

```
#define CURR_INTEGRAL_SAT_LIMIT 100000
```

Anti windup on current control.

5.5.2.5 CURRENT_HYSTERESIS

```
#define CURRENT_HYSTERESIS 10
```

milliAmperes of hysteresis for current control.

5.5.2.6 DEFAULT_CURRENT_LIMIT

```
#define DEFAULT_CURRENT_LIMIT 1500
```

Default Current limit, 0 stands for unlimited.

5.5.2.7 DEFAULT_EEPROM_DISPLACEMENT

```
#define DEFAULT_EEPROM_DISPLACEMENT 50
```

Number of pages occupied by the EEPROM.

5.5.2.8 DIV_INIT_VALUE

```
#define DIV_INIT_VALUE 1
```

Initial value for hand counter calibration.

5.5.2.9 EEPROM_BYTES_ROW

```
#define EEPROM_BYTES_ROW 16
```

EEPROM number of bytes per row.

5.5.2.10 EEPROM_COUNTERS_ROWS

```
#define EEPROM_COUNTERS_ROWS 5
```

EEPROM number of rows dedicated to store counters.

5.5.2.11 EMG_SAMPLE_TO_DISCARD

```
#define EMG_SAMPLE_TO_DISCARD 500
```

Number of sample to discard before calibration.

5.5.2.12 LOOKUP_DIM

```
#define LOOKUP_DIM 6
```

Dimension of the current lookup table.

5.5.2.13 MAX_WATCHDOG_TIMER

```
#define MAX_WATCHDOG_TIMER 250
```

num $*$ 2 [cs].

5.5.2.14 NUM_OF_ANALOG_INPUTS

```
#define NUM_OF_ANALOG_INPUTS 4
```

Total number of analogic inputs.

5.5.2.15 NUM_OF_EMGS

```
#define NUM_OF_EMGS 2
```

Number of emg channels.

5.5.2.16 NUM_OF_MOTORS

```
#define NUM_OF_MOTORS 2
```

Number of motors.

5.5.2.17 NUM_OF_PARAMS

```
#define NUM_OF_PARAMS 29
```

Number of parameters saved in the EEPROM.

5.5.2.18 NUM_OF_SENSORS

```
#define NUM_OF_SENSORS 3
```

Number of encoders.

5.5.2.19 POS_INTEGRAL_SAT_LIMIT

```
#define POS_INTEGRAL_SAT_LIMIT 50000000
```

Anti windup on position control.

5.5.2.20 PWM_MAX_VALUE

```
#define PWM_MAX_VALUE 100
```

Maximum value of the PWM signal.

5.5.2.21 RECEIVE

```
#define RECEIVE 3
```

Package data receiving status.

5.5.2.22 SAMPLES_FOR_EMG_MEAN

```
#define SAMPLES_FOR_EMG_MEAN 1000
```

Number of samples used to mean emg values.

5.5.2.23 SAMPLES_FOR_MEAN

```
#define SAMPLES_FOR_MEAN 100
```

Number of samples used to mean current values.

5.5.2.24 STATE_ACTIVE

```
#define STATE_ACTIVE 1
```

Closed SoftHand position / EMG Active.

5.5.2.25 STATE_INACTIVE

```
#define STATE_INACTIVE 0
```

Open SoftHand position / EMG Inactive.

5.5.2.26 UNLOAD

```
#define UNLOAD 4
```

Package data flush status.

5.5.2.27 WAIT_ID

```
#define WAIT_ID 1
```

Package ID waiting status.

5.5.2.28 WAIT_LENGTH

```
#define WAIT_LENGTH 2
```

Package lenght waiting status.

5.5.2.29 WAIT_START

```
#define WAIT_START 0
```

Package start waiting status.

5.5.3 Enumeration Type Documentation

5.5.3.1 counter_status

```
enum counter_status
```

Enumerator

PREPARE_DATA	Prepare data to be written on EEPROM.
WRITE_CYCLES	Cycles writing on EEPROM is enabled and control is passed to query.
WAIT_QUERY	Wait until EEPROM_Query() has finished writing on EEPROM and then disable cycles writing.
WRITE_END	End of EEPROM writing.
NONE	Cycles writing on EEPROM is disabled.

5.5.3.2 emg_status

```
enum emg_status
```

Enumerator

NORMAL	Normal execution
RESET	Reset analog measurements
DISCARD	Discard first samples to obtain a correct value
SUM_AND_MEAN	Sum and mean a definite value of samples
WAIT	The second emg waits until the first emg has a valid value
WAIT_EoC	The second emg waits for end of calibration

5.5.4 Variable Documentation

5.5.4.1 c_mem

```
struct st_mem g_mem c_mem
```

Memory parameters.

5.5.4.2 calib

```
struct st_calib calib
```

Calibration variables.

5.5.4.3 can_write

```
CYBIT can_write
```

Write to EEPROM flag.

5.5.4.4 cycle_time

```
float cycle_time
```

Variable used to calculate how much time a cycle takes.

5.5.4.5 cycles_interrupt_flag

```
CYBIT cycles_interrupt_flag
```

Cycles timer interrupt flag enabler.

5.5.4.6 cycles_status

```
counter_status CYDATA cycles_status
```

Cycles counter state machine status.

5.5.4.7 dev_pwm_limit

```
uint8 dev_pwm_limit
```

Device pwm limit

Device pwm limit. It may change during execution.

5.5.4.8 dev_pwm_sat

```
uint8 dev_pwm_sat
```

Device pwm limit saturation

Device pwm saturation. By default the saturation value must not exceed 100.

5.5.4.9 dev_tension

```
int32 dev_tension
```

Power supply tension

Power supply tension.

5.5.4.10 dev_tension_f

```
int32 dev_tension_f
```

Filtered power supply tension

Filtered power supply tension.

5.5.4.11 emg_1_status

```
emg_status CYDATA emg_1_status
```

First EMG sensor status.

5.5.4.12 emg_2_status

```
emg_status CYDATA emg_2_status
```

Second EMG sensor status.

5.5.4.13 forced_open

```
uint8 forced_open
```

Forced open flag (used in position with rest position control).

5.5.4.14 g_measOld

```
struct st_meas g_meas g_measOld
```

Measurements.

5.5.4.15 g_refOld

```
struct st_ref g_ref g_refNew g_refOld
```

Reference variables.

5.5.4.16 g_rx

```
struct st_data g_rx
```

Incoming/Outcoming data.

5.5.4.17 interrupt_flag

```
CYBIT interrupt_flag
```

Interrupt flag enabler.

5.5.4.18 pow_tension

```
int32 pow_tension
```

Computed power supply tension.

5.5.4.19 pwm_sign

```
int8 pwm_sign
```

ADC measurements buffer. Sign of pwm driven. Used to obtain current sign.

5.5.4.20 reset_last_value_flag

```
CYBIT reset_last_value_flag
```

This flag is set when the encoders last values must be resetted.

5.5.4.21 rest_enabled

```
uint8 rest_enabled
```

Rest position flag.

5.5.4.22 rest_pos_curr_ref

```
int32 rest_pos_curr_ref
```

Rest position current reference.

5.5.4.23 tau_feedback

```
float tau_feedback
```

Torque feedback.

5.5.4.24 tension_valid

```
CYBIT tension_valid
```

Tension validation bit.

5.5.4.25 timer_value

```
uint32 timer_value
```

End time of the firmware main loop.

5.5.4.26 timer_value0

```
uint32 timer_value0
```

Start time of the firmware main loop.

5.5.4.27 watchdog_flag

```
CYBIT watchdog_flag
```

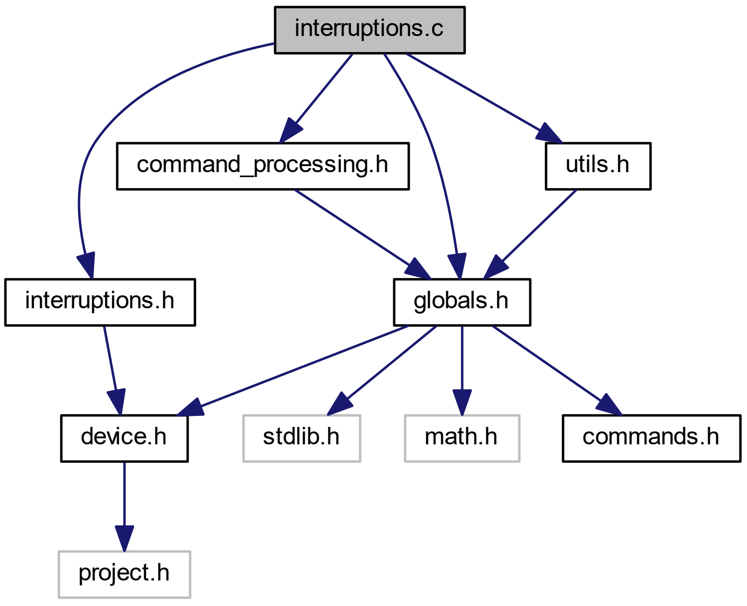
Watchdog flag enabler.

5.6 interruptions.c File Reference

Interruption handling and firmware core functions.

```
#include <interruptions.h>
#include <command_processing.h>
#include <globals.h>
#include <utils.h>
```

Include dependency graph for interruptions.c:



Functions

- **CY_ISR** (ISR_WATCHDOG_Handler)
- **CY_ISR** (ISR_RS485_RX_ExInterrupt)
- **CY_ISR** (ISR_CYCLES_Handler)
- void **interrupt_manager** ()
- void **function_scheduler** (void)
- void **motor_control** ()
- void **encoder_reading** (const uint8 idx)
- void **analog_read_end** ()
- void **overcurrent_control** ()
- void **pwm_limit_search** ()
- void **cycles_counter_update** ()
- void **save_cycles_eeprom** ()

Variables

- static const uint8 **pwm_preload_values** [29]
- CYCODE uint8 **pwm_preload_values_6v** [32]

5.6.1 Detailed Description

Interruption handling and firmware core functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.6.2 Function Documentation

5.6.2.1 analog_read_end()

```
void analog_read_end ( )
```

This function executes and terminates the analog readings.

5.6.2.2 cycles_counter_update()

```
void cycles_counter_update ( )
```

This function increases the cycles counters variables, depending on SoftHand position and the current absorbed by the motor.

5.6.2.3 encoder_reading()

```
void encoder_reading (
            const uint8 index )
```

This functions reads the value from the encoder pointed by index.

Parameters

index	The number of the encoder that must be read.

5.6.2.4 function_scheduler()

```
void function_scheduler (
            void  )
```

This function schedules the other functions in an order that optimizes the controller usage.

5.6.2.5 interrupt_manager()

```
void interrupt_manager ( )
```

This function is called in predefinited moments during firmware execution in order to unpack the received package.

5.6.2.6 motor_control()

```
void motor_control ( )
```

This function controls the motor direction and velocity, depending on the input and control modality set.

5.6.2.7 overcurrent_control()

```
void overcurrent_control ( )
```

This function increases or decreases the pwm maximum value, depending on the current absorbed by the motor.

5.6.2.8 pwm_limit_search()

```
void pwm_limit_search ( )
```

This function scales the pwm value of the motor, depending on the power supply voltage, in order to not make the motor wind too fast.

5.6.2.9 save_cycles_eeprom()

```
void save_cycles_eeprom ( )
```

This function saves cycles counters variables into EEPROM memory.

5.6.3 Variable Documentation

5.6.3.1 pwm_preload_values

```
const uint8 pwm_preload_values[29]  [static]
```

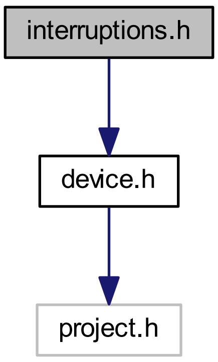
Initial value:

```
= {100,
                                                        83,
                                                        78,
                                                        76,
                                                        74,
                                                        72,
                                                        70,
                                                        68,
                                                        67,
                                                        65,
                                                        64,
                                                        63,
                                                        62,
                                                        61,
                                                        60,
                                                        59,
                                                        58,
                                                        57,
                                                        56,
                                                        56,
                                                        55,
                                                        54,
                                                        54,
                                                        53,
                                                        52,
                                                        52,
                                                        52,
                                                        51,
                                                        51}
```

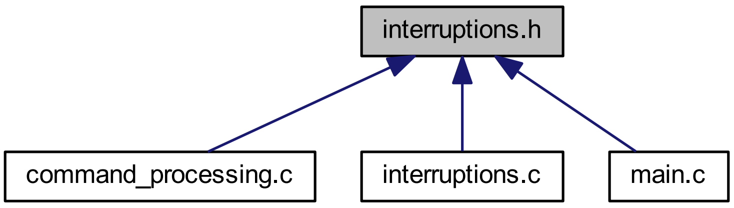
5.7 interruptions.h File Reference

Interruptions header file.

`#include <device.h>`
Include dependency graph for interruptions.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **save_cycles_eeprom** ()

Interruptions

- **CY_ISR_PROTO** (ISR_RS485_RX_ExInterrupt)
- **CY_ISR_PROTO** (ISR_WATCHDOG_Handler)
- **CY_ISR_PROTO** (ISR_CYCLES_Handler)

General function scheduler

- void **function_scheduler** (void)

Encoder reading function

- void **encoder_reading** (const uint8 index)

Motor control function

- void **motor_control** ()

Analog readings

- void **analog_read_end** ()

Interrupt manager

- void **interrupt_manager** ()

Utility functions

- void **pwm_limit_search** ()
- void **overcurrent_control** ()
- void **cycles_counter_update** ()

5.7.1 Detailed Description

Interruptions header file.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.7.2 Function Documentation

5.7.2.1 analog_read_end()

```
void analog_read_end ( )
```

This function executes and terminates the analog readings.

5.7.2.2 CY_ISR_PROTO() [1/3]

```
CY_ISR_PROTO (
             ISR_RS485_RX_ExInterrupt  )
```

This interruption sets a flag to let the firmware know that a communication interruption is pending and needs to be handled. The interruption will be handled in predefined moments during the firmware execution. When this interruption is handled, it unpacks the package received on the RS485 communication bus.

5.7.2.3 CY_ISR_PROTO() [2/3]

```
CY_ISR_PROTO (
             ISR_WATCHDOG_Handler  )
```

This interruption sets a flag to let the firmware know that a watchdog interruption is pending and needs to be handled. The interrpution will be handled in predefined moments during the firmware execution. When this interruption is handled, it deactivates the board because the watchdog timer has expired.

5.7.2.4 CY_ISR_PROTO() [3/3]

```
CY_ISR_PROTO (
            ISR_CYCLES_Handler  )
```

This interruption sets a flag to let the firmware know that a cycles timer interruption is pending and needs to be handled. The interrpution will be handled in predefined moments during the firmware execution. When this interruption is handled, it updates cycles counters.

5.7.2.5 cycles_counter_update()

```
void cycles_counter_update ( )
```

This function increases the cycles counters variables, depending on SoftHand position and the current absorbed by the motor.

5.7.2.6 encoder_reading()

```
void encoder_reading (
            const uint8 index )
```

This functions reads the value from the encoder pointed by index.

Parameters

index	The number of the encoder that must be read.

5.7.2.7 function_scheduler()

```
void function_scheduler (
            void  )
```

This function schedules the other functions in an order that optimizes the controller usage.

5.7.2.8 interrupt_manager()

```
void interrupt_manager ( )
```

This function is called in predefinited moments during firmware execution in order to unpack the received package.

5.7.2.9 motor_control()

```
void motor_control ( )
```

This function controls the motor direction and velocity, depending on the input and control modality set.

5.7.2.10 overcurrent_control()

```
void overcurrent_control ( )
```

This function increases or decreases the pwm maximum value, depending on the current absorbed by the motor.

5.7.2.11 pwm_limit_search()

```
void pwm_limit_search ( )
```

This function scales the pwm value of the motor, depending on the power supply voltage, in order to not make the motor wind too fast.

5.7.2.12 save_cycles_eeprom()

```
void save_cycles_eeprom ( )
```

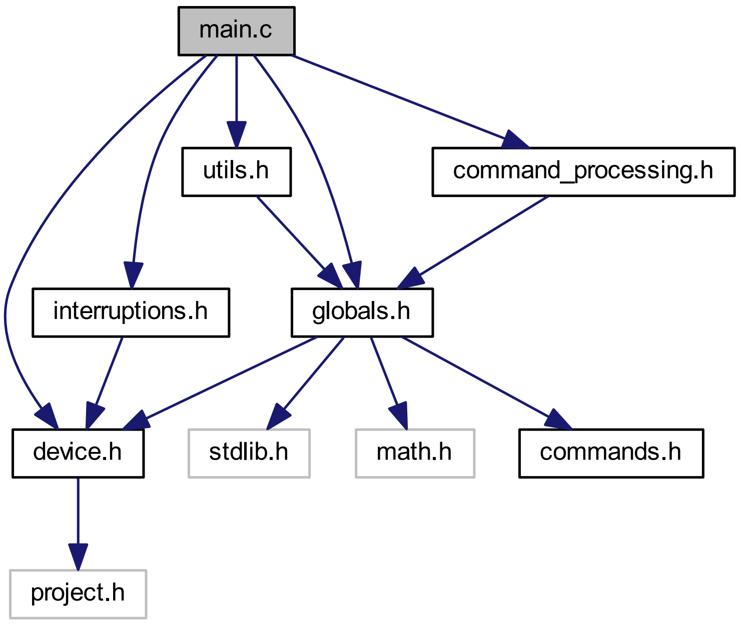
This function saves cycles counters variables into EEPROM memory.

5.8 main.c File Reference

Firmware main file.

```
#include <device.h>
#include <globals.h>
#include <interruptions.h>
#include <command_processing.h>
#include <utils.h>
```

Include dependency graph for main.c:



Functions

- int **main** ()

5.8.1 Detailed Description

Firmware main file.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

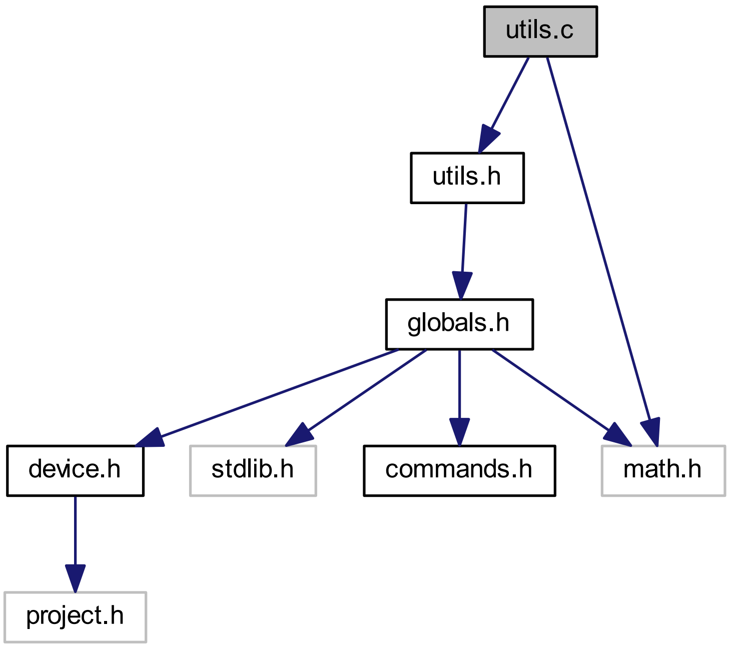
(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.9 utils.c File Reference

Definition of utility functions.

```
#include <utils.h>
#include <math.h>
```

Include dependency graph for utils.c:



Macros

- #define **N1** 15
- #define **N2** 14
- #define **I1** 1

  First wheel invariant value.
- #define **I2** (-1)

  Second wheel invariant value.
- #define **M** 65536

  Number of encoder ticks per turn.

Functions

- int32 **curr_estim** (int32 pos, int32 vel, int32 ref)
- int32 **filter_v** (int32 new_value)
- int32 **filter_i1** (int32 new_value)
- int32 **filter_ch1** (int32 new_value)
- int32 **filter_ch2** (int32 new_value)
- int32 **filter_vel_1** (int32 new_value)
- int32 **filter_vel_2** (int32 new_value)
- int32 **filter_vel_3** (int32 new_value)
- int32 **filter_acc_1** (int32 new_value)
- int32 **filter_acc_2** (int32 new_value)
- int32 **filter_acc_3** (int32 new_value)
- int32 **filter_curr_diff** (int32 new_value)
- int32 **filter_voltage** (int32 new_value)
- CYBIT **check_enc_data** (const uint32 ∗value)
- int **my_round** (const double x)
- uint32 **my_mod** (int32 val, int32 divisor)
- void **calibration** (void)
- int **calc_turns_fcn** (const int32 pos1, const int32 pos2)
- void **check_rest_position** (void)
- void **reset_counters** ()

5.9.1 Detailed Description

Definition of utility functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.9.2 Function Documentation

5.9.2.1 calc_turns_fcn()

```
int calc_turns_fcn (
             const int32 pos1,
             const int32 pos2 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. It need two encoders to work.

Parameters

pos1	First encoder position
pos2	Second encoder position

Returns

Returns the number of turns of motor pulley at startup

5.9.2.2 calibration()

```
void calibration ( )
```

This function counts a series of hand opening and closing used to execute a calibration of the device.

5.9.2.3 check_enc_data()

```
CYBIT check_enc_data (
             const uint32 * value )
```

This function controls if the read encoder data is correct or not.

Parameters

value	A pointer to the encoder data read

Returns

Returns 1 if the read data is correct, 0 otherwise

5.9.2.4 check_rest_position()

```
void check_rest_position ( )
```

This function checks for rest position and, in case, gives a position reference to qbhand.

5.9.2.5 curr_estim()

```
int32 curr_estim (
            int32 pos,
            int32 vel,
            int32 acc )
```

Function used to obtain current estimation through current lookup table.

Parameters

pos	Position of the encoder in ticks.
vel	Speed of the encoder.
accel	Acceleration of the encoder

Returns

Returns an estimation of the motor current, depending on its position, velocity and acceleration.

5.9.2.6 filter_acc_1()

```
int32 filter_acc_1 (
            int32 value )
```

Filter on first encoder rotational acceleration. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered first encoder rotational acceleration value

5.9.2.7 filter_acc_2()

```
int32 filter_acc_2 (
            int32 value )
```

Filter on second encoder rotation acceleration (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered second encoder rotational acceleration value

5.9.2.8 filter_acc_3()

```
int32 filter_acc_3 (
            int32 value )
```

Filter on third encoder rotation acceleration (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered third encoder rotational acceleration value

5.9.2.9 filter_ch1()

```
int32 filter_ch1 (
            int32 value )
```

Filter on the first EMG sensor converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered emg sensor value

5.9.2.10 filter_ch2()

```
int32 filter_ch2 (
            int32 value )
```

Filter on the second EMG sensor converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered emg sensor value

5.9.2.11 filter_curr_diff()

```
int32 filter_curr_diff (
            int32 curr_diff )
```

Low pass filter on current difference between measured and estimated current

Parameters

curr_diff	Difference between the measured current and the estimated one.

Returns

Returns the filtered current difference value

5.9.2.12 filter_i1()

```
int32 filter_i1 (
            int32 value )
```

Filter on the motor current converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered current value

5.9.2.13 filter_v()

```
int32 filter_v (
            int32 new_value )
```

Filter on the converted voltage value. The weighted average between the old value and the new one is executed.

Parameters

new_value	New value of the filter.

Returns

Returns the filtered voltage value

5.9.2.14 filter_vel_1()

```
int32 filter_vel_1 (
            int32 value )
```

Filter on first encoder rotational speed. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered first encoder rotational speed value

5.9.2.15 filter_vel_2()

```
int32 filter_vel_2 (
            int32 value )
```

Filter on second encoder rotational speed (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered second encoder rotational speed value

5.9.2.16 filter_vel_3()

```
int32 filter_vel_3 (
            int32 value )
```

Filter on third encoder rotational speed (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered third encoder rotational speed value

5.9.2.17 filter_voltage()

```
int32 filter_voltage (
            int32 value )
```

Filter on voltage readings. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered voltage value

5.9.2.18 my_mod()

```
uint32 my_mod (
            int32 val,
            int32 divisor )
```

This function computes the module function, returning positive values regardless of wheter the value passed is negative

Parameters

val	The value of which the module needs to be calculated
divisor	The divisor according to which the module is calculated

5.9.2.19 my_round()

```
int my_round (
             const double x )
```

This functions approximates the value passed to the nearest integer

Parameters

x	The floating point value that needs to be rounded

5.9.2.20 reset_counters()

```
void reset_counters ( )
```

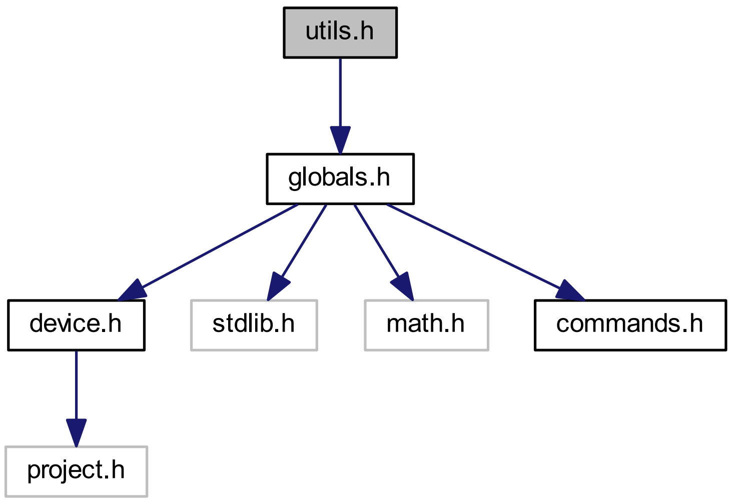
This function reset statistics counters

5.10 utils.h File Reference

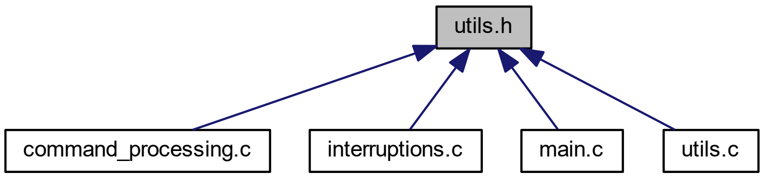
Utility functions declaration.

`#include <globals.h>`

Include dependency graph for utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **TIMER_CLOCK** 10000
- #define **ALPHA** 32

  Voltage and current filters constant.
- #define **BETA** 50

  Emg filters constant.
- #define **GAMMA** 128

  Velocity filters constant.
- #define **DELTA** 8

  Acceleration filters constant.
- #define **ETA** 16

  Current residual filter.

- #define **EPSILON** 8

  Voltage readings filter.
- #define **ZMAX** 5
- #define **ZERO_TOL** 100
- #define **REFSPEED** 20
- #define **SIGN**(A) (((A) $>$=0) ? (1) : (-1))

Functions

- int32 **filter_voltage** (int32 value)

Filters

- int32 **filter_v** (int32 new_value)
- int32 **filter_ch1** (int32 value)
- int32 **filter_ch2** (int32 value)
- int32 **filter_i1** (int32 value)
- int32 **filter_vel_1** (int32 value)
- int32 **filter_vel_2** (int32 value)
- int32 **filter_vel_3** (int32 value)
- int32 **filter_acc_1** (int32 value)
- int32 **filter_acc_2** (int32 value)
- int32 **filter_acc_3** (int32 value)

Estimating current and difference

- int32 **curr_estim** (int32 pos, int32 vel, int32 acc)
- int32 **filter_curr_diff** (int32 curr_diff)

Utility functions

- int **my_round** (const double x)
- uint32 **my_mod** (int32 val, int32 divisor)
- CYBIT **check_enc_data** (const uint32 $*$value)
- int **calc_turns_fcn** (const int32 pos1, const int32 pos2)
- void **calibration** ()
- void **check_rest_position** ()
- void **reset_counters** ()

5.10.1 Detailed Description

Utility functions declaration.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

(C) 2012-2016 qbrobotics. All rights reserved.
(C) 2017 Centro "E.Piaggio". All rights reserved.

5.10.2 Macro Definition Documentation

5.10.2.1 REFSPEED

```
#define REFSPEED 20
```

Constant depending on PID values.

5.10.2.2 SIGN

```
#define SIGN(
             A ) (((A) >=0) ?  (1) :  (-1))
```

Sign calculation function.

5.10.2.3 ZERO_TOL

```
#define ZERO_TOL 100
```

Deadband used to put to zero the virtual position due to the fact that the friction model has errors when the position is near to zero.

5.10.2.4 ZMAX

```
#define ZMAX 5
```

Constant useful for current estimation procedure.

5.10.3 Function Documentation

5.10.3.1 calc_turns_fcn()

```
int calc_turns_fcn (
             const int32 pos1,
             const int32 pos2 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. It need two encoders to work.

Parameters

pos1	First encoder position
pos2	Second encoder position

Returns

Returns the number of turns of motor pulley at startup

5.10.3.2 calibration()

```
void calibration ( )
```

This function counts a series of hand opening and closing used to execute a calibration of the device.

5.10.3.3 check_enc_data()

```
CYBIT check_enc_data (
            const uint32 * value )
```

This function controls if the read encoder data is correct or not.

Parameters

value	A pointer to the encoder data read

Returns

Returns 1 if the read data is correct, 0 otherwise

5.10.3.4 check_rest_position()

```
void check_rest_position ( )
```

This function checks for rest position and, in case, gives a position reference to qbhand.

5.10.3.5 curr_estim()

```
int32 curr_estim (
            int32 pos,
            int32 vel,
            int32 acc )
```

Function used to obtain current estimation through current lookup table.

Parameters

pos	Position of the encoder in ticks.
vel	Speed of the encoder.
accel	Acceleration of the encoder

Returns

Returns an estimation of the motor current, depending on its position, velocity and acceleration.

5.10.3.6 filter_acc_1()

```
int32 filter_acc_1 (
            int32 value )
```

Filter on first encoder rotational acceleration. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered first encoder rotational acceleration value

5.10.3.7 filter_acc_2()

```
int32 filter_acc_2 (
            int32 value )
```

Filter on second encoder rotation acceleration (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered second encoder rotational acceleration value

5.10.3.8 filter_acc_3()

```
int32 filter_acc_3 (
            int32 value )
```

Filter on third encoder rotation acceleration (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered third encoder rotational acceleration value

5.10.3.9 filter_ch1()

```
int32 filter_ch1 (
            int32 value )
```

Filter on the first EMG sensor converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered emg sensor value

5.10.3.10 filter_ch2()

```
int32 filter_ch2 (
            int32 value )
```

Filter on the second EMG sensor converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered emg sensor value

5.10.3.11 filter_curr_diff()

```
int32 filter_curr_diff (
            int32 curr_diff )
```

Low pass filter on current difference between measured and estimated current

Parameters

curr_diff	Difference between the measured current and the estimated one.

Returns

Returns the filtered current difference value

5.10.3.12 filter_i1()

```
int32 filter_i1 (
            int32 value )
```

Filter on the motor current converted value. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered current value

5.10.3.13 filter_v()

```
int32 filter_v (
            int32 new_value )
```

Filter on the converted voltage value. The weighted average between the old value and the new one is executed.

Parameters

new_value	New value of the filter.

Returns

Returns the filtered voltage value

5.10.3.14 filter_vel_1()

```
int32 filter_vel_1 (
            int32 value )
```

Filter on first encoder rotational speed. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered first encoder rotational speed value

5.10.3.15 filter_vel_2()

```
int32 filter_vel_2 (
            int32 value )
```

Filter on second encoder rotational speed (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered second encoder rotational speed value

5.10.3.16 filter_vel_3()

```
int32 filter_vel_3 (
            int32 value )
```

Filter on third encoder rotational speed (if present). The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered third encoder rotational speed value

5.10.3.17 filter_voltage()

```
int32 filter_voltage (
            int32 value )
```

Filter on voltage readings. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.

Returns

Returns the filtered voltage value

5.10.3.18 my_mod()

```
uint32 my_mod (
            int32 val,
            int32 divisor )
```

This function computes the module function, returning positive values regardless of wheter the value passed is negative

Parameters

val	The value of which the module needs to be calculated
divisor	The divisor according to which the module is calculated

5.10.3.19 my_round()

```
int my_round (
            const double x )
```

This functions approximates the value passed to the nearest integer

Parameters

x	The floating point value that needs to be rounded

5.10.3.20 reset_counters()

```
void reset_counters ( )
```

This function reset statistics counters

Index