PSoC5 firmware

v. 1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Firmware

This is the firmware of PSoC5 logic board.

Version

1.0

This is the firmware of PSoC5 logic board. Depending on the configuration, it can control up to two motors and read its encoders. 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_counters Struct Reference

EEPROM stored structures.

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

Data Fields

- 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**

4.2.1 Detailed Description

EEPROM stored structures.

4.2.2 Field Documentation

4.2.2.1 current_hist

```
uint32 current_hist[4]
```

Current histogram - 4 zones.

4.2.2.2 emg_counter

```
uint32 emg_counter[2]
```

Counter for EMG activation - both channels.

4.2.2.3 position_hist

```
uint32 position_hist[10]
```

Positions histogram - 10 zones.

4.2.2.4 rest_counter

```
uint32 rest_counter
```

Counter for rest position occurrences.

4.2.2.5 total_time_on

```
uint32 total_time_on
```

Total time of system power (in seconds).

4.2.2.6 total_time_rest

```
uint32 total_time_rest
```

Total time of system while rest position is maintained.

4.2.2.7 wire_disp

```
uint32 wire_disp
```

Counter for total wire displacement measurement.

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

- **globals.h**

4.3 st_data Struct Reference

Data sent/received structure.

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

Data Fields

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

4.3.1 Detailed Description

Data sent/received structure.

4.3.2 Field Documentation

4.3.2.1 buffer

```
uint8 buffer[128]
```

Data buffer [CMD | DATA | CHECKSUM].

4.3.2.2 ind

```
int16 ind
```

Data buffer index.

4.3.2.3 length

```
int16 length
```

Data buffer length.

4.3.2.4 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**

4.4 st_device Struct Reference

Data Fields

- uint8 **id**
- uint8 **hw_maint_date** [3]
- uint8 **stats_period_begin_date** [3]
- uint8 **right_left**
- uint8 **reset_counters**
- uint8 **use_2nd_motor_flag**
- uint8 **baud_rate**
- uint8 **user_id**
- uint8 **dev_type**
- uint8 **unused_bytes** [3]

4.4.1 Field Documentation

4.4.1.1 baud_rate

```
uint8 baud_rate
```

Baud Rate setted.

4.4.1.2 dev_type

```
uint8 dev_type
```

Device type identificator.

4.4.1.3 hw_maint_date

```
uint8 hw_maint_date[3]
```

Date of last hardware maintenance.

4.4.1.4 id

```
uint8 id
```

Device id.

4.4.1.5 reset_counters

```
uint8 reset_counters
```

Reset counters flag.

4.4.1.6 right_left

```
uint8 right_left
```

Right/Left hand.

4.4.1.7 stats_period_begin_date

```
uint8 stats_period_begin_date[3]
```

Date of begin of usage statistics period.

4.4.1.8 unused_bytes

```
uint8 unused_bytes[3]
```

Unused bytes to fill row.

4.4.1.9 use_2nd_motor_flag

```
uint8 use_2nd_motor_flag
```

Use 2nd motor (2 powers).

4.4.1.10 user_id

```
uint8 user_id
```

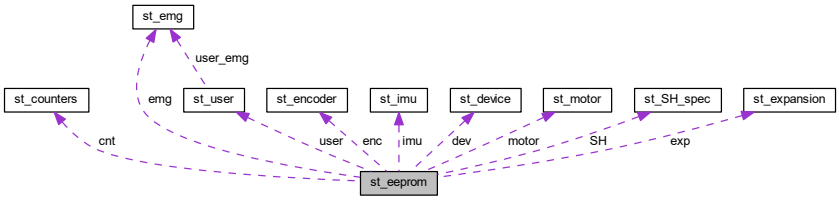
User identificator (if usual user).

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

- **globals.h**

4.5 st_eeprom Struct Reference

Collaboration diagram for st_eeprom:



Data Fields

- uint8 **flag**
- uint8 **unused_bytes** [15]
- struct **st_counters cnt**
- uint8 **unused_bytes1** [**EEPROM_BYTES_ROW** ∗4]
- struct **st_device dev**
- struct **st_motor motor** [**NUM_OF_MOTORS**]
- struct **st_encoder enc** [**N_ENCODER_LINE_MAX**]
- struct **st_emg emg**
- struct **st_imu imu**
- struct **st_expansion exp**
- struct **st_user user** [NUM_OF_USERS]
- struct **st_SH_spec SH**

4.5.1 Field Documentation

4.5.1.1 cnt

`struct` **st_counters** `cnt`

Statistics Counters.

4.5.1.2 dev

`struct` **st_device** `dev`

Device information.

4.5.1.3 emg

`struct` **st_emg** `emg`

EMG variables.

4.5.1.4 enc

`struct` **st_encoder** `enc[` **N_ENCODER_LINE_MAX**`]`

Encoder variables (1 line).

4.5.1.5 exp

`struct` **st_expansion** `exp`

SD and ADC variables.

4.5.1.6 flag

```
uint8 flag
```

If checked the device has been configured.

4.5.1.7 imu

```
struct st_imu imu
```

IMU general variables.

4.5.1.8 motor

```
struct st_motor motor[ NUM_OF_MOTORS]
```

Motor variables.

4.5.1.9 SH

```
struct st_SH_spec SH
```

SoftHand specific variables.

4.5.1.10 unused_bytes

```
uint8 unused_bytes[15]
```

Leave bytes to align structures to memory rows.

4.5.1.11 unused_bytes1

```
uint8 unused_bytes1[ EEPROM_BYTES_ROW *4]
```

Leave for rows free for further uses.

4.5.1.12 user

```
struct st_user user[NUM_OF_USERS]
```

User variables.

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

- **globals.h**

4.6 st_emg Struct Reference

Data Fields

- uint16 **emg_threshold** [**NUM_OF_INPUT_EMGS**]
- uint32 **emg_max_value** [**NUM_OF_INPUT_EMGS**]
- uint8 **emg_speed**
- uint8 **emg_calibration_flag**
- uint8 **switch_emg**
- uint8 **unused_bytes** [1]

4.6.1 Field Documentation

4.6.1.1 emg_calibration_flag

```
uint8 emg_calibration_flag
```

Enable emg calibration on startup.

4.6.1.2 emg_max_value

```
uint32 emg_max_value[ NUM_OF_INPUT_EMGS]
```

Maximum value for EMG.

4.6.1.3 emg_speed

```
uint8 emg_speed
```

Maximum closure speed when using EMG.

4.6.1.4 emg_threshold

```
uint16 emg_threshold[ NUM_OF_INPUT_EMGS]
```

Minimum value for activation of EMG control.

4.6.1.5 switch_emg

```
uint8 switch_emg
```

EMG opening/closure switch.

4.6.1.6 unused_bytes

```
uint8 unused_bytes[1]
```

Unused bytes to fill row.

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

- **globals.h**

4.7 st_emg_meas Struct Reference

Data Fields

- int32 **emg** [**NUM_OF_INPUT_EMGS**]
- int32 **add_emg** [**NUM_OF_ADDITIONAL_EMGS**]

4.7.1 Field Documentation

4.7.1.1 add_emg

```
int32 add_emg[ NUM_OF_ADDITIONAL_EMGS]
```

Additional EMG sensors values.

4.7.1.2 emg

```
int32 emg[ NUM_OF_INPUT_EMGS]
```

EMG sensors values.

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

- **globals.h**

4.8 st_encoder Struct Reference

Data Fields

- uint8 **Enc_raw_read_conf** [**N_ENCODERS_PER_LINE_MAX**]
- uint8 **res** [**NUM_OF_SENSORS**]
- int32 **m_off** [**NUM_OF_SENSORS**]
- float32 **m_mult** [**NUM_OF_SENSORS**]
- uint8 **double_encoder_on_off**
- uint8 **Enc_idx_use_for_control** [**NUM_OF_SENSORS**]
- int8 **motor_handle_ratio**
- int8 **gears_params** [3]
- uint8 **unused_bytes** [8]

4.8.1 Field Documentation

4.8.1.1 double_encoder_on_off

```
uint8 double_encoder_on_off
```

Double encoder ON/OFF.

4.8.1.2 Enc_idx_use_for_control

```
uint8 Enc_idx_use_for_control[ NUM_OF_SENSORS]
```

Indices of encoder used for motor control.

4.8.1.3 Enc_raw_read_conf

```
uint8 Enc_raw_read_conf[ N_ENCODERS_PER_LINE_MAX]
```

Encoder configuration flags for raw reading.

4.8.1.4 gears_params

```
int8 gears_params[3]
```

Number of teeth of first and second gear and related invariant.

4.8.1.5 m_mult

```
float32 m_mult[ NUM_OF_SENSORS]
```

Measurement multiplier.

4.8.1.6 m_off

```
int32 m_off[ NUM_OF_SENSORS]
```

Measurement offset.

4.8.1.7 motor_handle_ratio

```
int8 motor_handle_ratio
```

Discrete multiplier for handle device.

4.8.1.8 res

```
uint8 res[ NUM_OF_SENSORS]
```

Angle resolution.

4.8.1.9 unused_bytes

```
uint8 unused_bytes[8]
```

Unused bytes to fill row.

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

- **globals.h**

4.9 st_expansion Struct Reference

Data Fields

- uint8 **curr_time** [6]
- uint8 **read_exp_port_flag**
- uint8 **read_ADC_sensors_port_flag**
- uint8 **ADC_conf** [NUM_OF_ADC_CHANNELS_MAX]
- uint8 **unused_bytes** [12]

4.9.1 Field Documentation

4.9.1.1 ADC_conf

```
uint8 ADC_conf[NUM_OF_ADC_CHANNELS_MAX]
```

ADC configuration flags.

4.9.1.2 curr_time

```
uint8 curr_time[6]
```

Current time from RTC (DD/MM/YY HH:MM:SS).

4.9.1.3 read_ADC_sensors_port_flag

```
uint8 read_ADC_sensors_port_flag
```

Enable ADC sensors Port.

4.9.1.4 read_exp_port_flag

```
uint8 read_exp_port_flag
```

Enable Expansion Port.

4.9.1.5 unused_bytes

```
uint8 unused_bytes[12]
```

Unused bytes to fill row.

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

- **globals.h**

4.10 st_filter Struct Reference

Filter structure.

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

Data Fields

- int32 **old_value**
- int32 **gain**

4.10.1 Detailed Description

Filter structure.

4.10.2 Field Documentation

4.10.2.1 gain

```
int32 gain
```

New value filter weight.

4.10.2.2 old_value

```
int32 old_value
```

Old variable value.

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

- **globals.h**

4.11 st_imu Struct Reference

Data Fields

- uint8 **read_imu_flag**
- uint8 **SPI_read_delay**
- uint8 **IMU_conf** [N_IMU_MAX][NUM_OF_IMU_DATA]
- uint8 **unused_bytes** [5]

4.11.1 Field Documentation

4.11.1.1 IMU_conf

```
uint8 IMU_conf[N_IMU_MAX][NUM_OF_IMU_DATA]
```

IMUs configuration flags.

4.11.1.2 read_imu_flag

```
uint8 read_imu_flag
```

Enable IMU reading feature.

4.11.1.3 SPI_read_delay

```
uint8 SPI_read_delay
```

Delay on SPI reading.

4.11.1.4 unused_bytes

```
uint8 unused_bytes[5]
```

Unused bytes to fill row.

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

- **globals.h**

4.12 st_imu_data Struct Reference

Data Fields

- uint8 **flags**
- int16 **accel_value** [3]
- int16 **gyro_value** [3]
- int16 **mag_value** [3]
- float **quat_value** [4]
- int16 **temp_value**

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

- **globals.h**

4.13 st_meas Struct Reference

Measurements structure.

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

Data Fields

- int32 **pos** [**NUM_OF_SENSORS**]
- int32 **curr**
- int32 **estim_curr**
- int8 **rot** [**NUM_OF_SENSORS**]
- int32 **vel** [**NUM_OF_SENSORS**]
- int32 **acc** [**NUM_OF_SENSORS**]

4.13.1 Detailed Description

Measurements structure.

4.13.2 Field Documentation

4.13.2.1 acc

```
int32 acc[ NUM_OF_SENSORS]
```

Encoder rotational acceleration.

4.13.2.2 curr

```
int32 curr
```

Motor current.

4.13.2.3 estim_curr

```
int32 estim_curr
```

Current estimation.

4.13.2.4 pos

```
int32 pos[ NUM_OF_SENSORS]
```

Encoder sensor position.

4.13.2.5 rot

```
int8 rot[ NUM_OF_SENSORS]
```

Encoder sensor rotations.

4.13.2.6 vel

```
int32 vel[ NUM_OF_SENSORS]
```

Encoder rotational velocity.

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

- **globals.h**

4.14 st_motor Struct Reference

Data Fields

- 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 **activate_pwm_rescaling**
- uint8 **motor_driver_type**
- uint8 **pos_lim_flag**
- int32 **pos_lim_inf**
- int32 **pos_lim_sup**
- int32 **max_step_neg**
- int32 **max_step_pos**
- float **curr_lookup** [**LOOKUP_DIM**]
- int16 **current_limit**
- uint8 **input_mode**
- uint8 **control_mode**
- uint8 **encoder_line**
- uint8 **pwm_rate_limiter**
- uint8 **not_revers_motor_flag**
- uint8 **unused_bytes** [13]

4.14.1 Field Documentation

4.14.1.1 activ

```
uint8 activ
```

Startup activation.

4.14.1.2 activate_pwm_rescaling

```
uint8 activate_pwm_rescaling
```

Activation of PWM rescaling for 12V motor.

4.14.1.3 control_mode

```
uint8 control_mode
```

Motor Control mode.

4.14.1.4 curr_lookup

```
float curr_lookup[ LOOKUP_DIM]
```

Table of values to get estimated curr.

4.14.1.5 current_limit

```
int16 current_limit
```

Limit for absorbed current.

4.14.1.6 encoder_line

```
uint8 encoder_line
```

Encoder line associated to the motor control.

4.14.1.7 input_mode

```
uint8 input_mode
```

Motor Input mode.

4.14.1.8 k_d

```
int32 k_d
```

Position controller derivative constant.

4.14.1.9 k_d_c

```
int32 k_d_c
```

Current controller derivative constant.

4.14.1.10 k_d_c_dl

```
int32 k_d_c_dl
```

Double loop current controller deriv. constant.

4.14.1.11 k_d_dl

```
int32 k_d_dl
```

Double loop position controller deriv. constant.

4.14.1.12 k_i

```
int32 k_i
```

Position controller integrative constant.

4.14.1.13 k_i_c

```
int32 k_i_c
```

Current controller integrative constant.

4.14.1.14 k_i_c_dl

```
int32 k_i_c_dl
```

Double loop current controller integr. constant.

4.14.1.15 k_i_dl

```
int32 k_i_dl
```

Double loop position controller integr. constant.

4.14.1.16 k_p

```
int32 k_p
```

Position controller proportional constant.

4.14.1.17 k_p_c

```
int32 k_p_c
```

Current controller proportional constant.

4.14.1.18 k_p_c_dl

```
int32 k_p_c_dl
```

Double loop current controller prop. constant.

4.14.1.19 k_p_dl

```
int32 k_p_dl
```

Double loop position controller prop. constant.

4.14.1.20 max_step_neg

```
int32 max_step_neg
```

Maximum number of steps per cycle for negative positions.

4.14.1.21 max_step_pos

```
int32 max_step_pos
```

Maximum number of steps per cycle for positive positions.

4.14.1.22 motor_driver_type

```
uint8 motor_driver_type
```

Specify motor type.

4.14.1.23 not_revers_motor_flag

```
uint8 not_revers_motor_flag
```

Flag to know if the motor is reversible or not.

4.14.1.24 pos_lim_flag

```
uint8 pos_lim_flag
```

Position limit active/inactive.

4.14.1.25 pos_lim_inf

```
int32 pos_lim_inf
```

Inferior position limit for motor.

4.14.1.26 pos_lim_sup

```
int32 pos_lim_sup
```

Superior position limit for motor[0].

4.14.1.27 pwm_rate_limiter

```
uint8 pwm_rate_limiter
```

PWM rate limiter max asscoaited to the motor.

4.14.1.28 unused_bytes

```
uint8 unused_bytes[13]
```

Unused bytes to fill row.

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

- **globals.h**

4.15 st_ref Struct Reference

Motor Reference structure.

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

Data Fields

- int32 **pos**
- int32 **curr**
- int32 **pwm**
- uint8 **onoff**

4.15.1 Detailed Description

Motor Reference structure.

4.15.2 Field Documentation

4.15.2.1 curr

```
int32 curr
```

Motor current reference.

4.15.2.2 onoff

`uint8 onoff`

Motor drivers enable.

4.15.2.3 pos

`int32 pos`

Motor position reference.

4.15.2.4 pwm

`int32 pwm`

Motor direct pwm control.

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

- **globals.h**

4.16 st_SH_spec Struct Reference

Data Fields

- int32 **rest_pos**
- int32 **rest_delay**
- int32 **rest_vel**
- uint8 **rest_position_flag**
- uint8 **unused_bytes** [3]

4.16.1 Field Documentation

4.16.1.1 rest_delay

`int32 rest_delay`

Hand rest position delay while in EMG mode.

4.16.1.2 rest_pos

`int32 rest_pos`

Hand rest position while in EMG mode.

4.16.1.3 rest_position_flag

```
uint8 rest_position_flag
```

Enable rest position feature.

4.16.1.4 rest_vel

```
int32 rest_vel
```

Hand velocity closure for rest position reaching.

4.16.1.5 unused_bytes

```
uint8 unused_bytes[3]
```

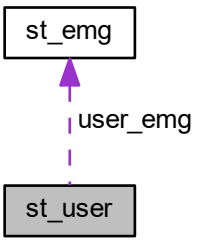
Unused bytes to fill row.

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

- **globals.h**

4.17 st_user Struct Reference

Collaboration diagram for st_user:



Data Fields

- char **user_code_string** [8]
- struct **st_emg** **user_emg**
- uint8 **unused_bytes** [8]

4.17.1 Field Documentation

4.17.1.1 unused_bytes

```
uint8 unused_bytes[8]
```

Unused bytes to fill row.

4.17.1.2 user_code_string

```
char user_code_string[8]
```

User code string.

4.17.1.3 user_emg

```
struct  st_emg user_emg
```

st_emg (p. 15) structure to store user emg values.

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

- **globals.h**

Chapter 5

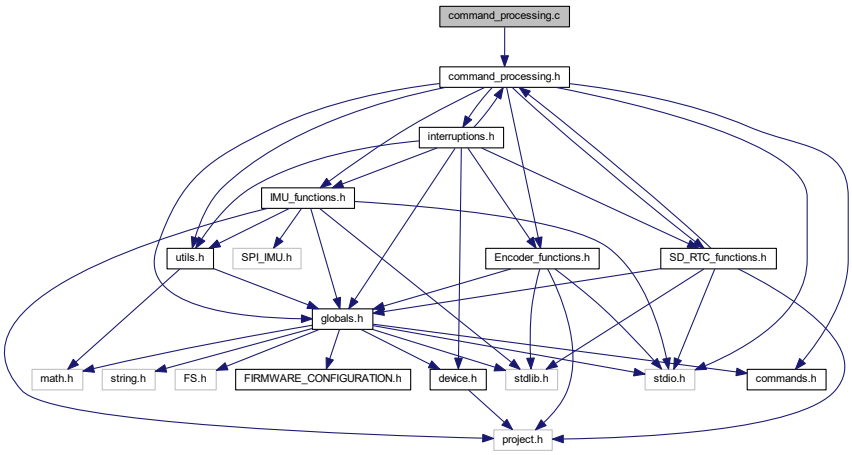
File Documentation

5.1 command_processing.c File Reference

Command processing functions.

```
#include "command_processing.h"
```

Include dependency graph for command_processing.c:



Functions

- void **commProcess** (void)
- void **infoSend** (void)
- void **infoGet** (uint16 info_type)
- void **get_param_list** (uint8 ∗VAR_P[**NUM_OF_PARAMS**], uint8 TYPES[**NUM_OF_PARAMS**], uint8 N↩ UM_ITEMS[**NUM_OF_PARAMS**], uint8 NUM_STRUCT[**NUM_OF_PARAMS**], uint8 ∗NUM_MENU, const char ∗PARAMS_STR[**NUM_OF_PARAMS**], uint8 CUSTOM_PARAM_GET[**NUM_OF_PARAMS**], const char ∗MENU_STR[**NUM_OF_PARAMS_MENU**])
- void **manage_param_list** (uint16 index)
- void **set_custom_param** (uint16 index)
- void **get_IMU_param_list** (uint16 index)

- void **setZeros** ()
- void **prepare_generic_info** (char ∗info_string)
- void **prepare_counter_info** (char ∗info_string)
- void **prepare_SD_param_info** (char ∗info_string)
- void **prepare_SD_legend** (char ∗info_string)
- void **prepare_SD_info** (char ∗info_string)
- void **IMU_reading_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 **memInit_SoftHandPro** (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_get_inputs** ()
- void **cmd_store_params** ()
- void **cmd_get_emg** ()
- void **cmd_get_imu_readings** ()
- void **cmd_get_encoder_map** ()
- void **cmd_get_encoder_raw** ()
- void **cmd_get_ADC_map** ()
- void **cmd_get_ADC_raw** ()

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

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_ADC_map()

```
void cmd_get_ADC_map ( )
```

This function gets ADC map

5.1.2.5 cmd_get_ADC_raw()

```
void cmd_get_ADC_raw ( )
```

This function gets Additional emg raw values

5.1.2.6 cmd_get_curr_and_meas()

```
void cmd_get_curr_and_meas ( )
```

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

5.1.2.7 cmd_get_currents()

```
void cmd_get_currents ( )
```

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

5.1.2.8 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. 47) function.

5.1.2.9 cmd_get_emg()

```
void cmd_get_emg ( )
```

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

5.1.2.10 cmd_get_encoder_map()

```
void cmd_get_encoder_map ( )
```

This function gets Encoder map

5.1.2.11 cmd_get_encoder_raw()

```
void cmd_get_encoder_raw ( )
```

This function gets Encoder raw values

5.1.2.12 cmd_get_imu_readings()

```
void cmd_get_imu_readings ( )
```

This function gets IMU readings

5.1.2.13 cmd_get_inputs()

```
void cmd_get_inputs ( )
```

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

5.1.2.14 cmd_get_measurements()

```
void cmd_get_measurements ( )
```

Bunch of functions used on request from UART communication

5.1.2.15 cmd_get_velocities()

```
void cmd_get_velocities ( )
```

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

5.1.2.16 cmd_ping()

```
void cmd_ping ( )
```

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

5.1.2.17 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.

5.1.2.18 cmd_set_inputs()

```
void cmd_set_inputs ( )
```

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

5.1.2.19 cmd_store_params()

```
void cmd_store_params ( )
```

This function stores the parameters to the EEPROM memory

5.1.2.20 commProcess()

```
void commProcess ( )
```

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

5.1.2.21 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.

5.1.2.22 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.

5.1.2.23 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.

5.1.2.24 get_IMU_param_list()

```
void get_IMU_param_list (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values.

5.1.2.25 get_param_list()

```
void get_param_list (
            uint8 * VAR_P[NUM_OF_PARAMS],
            uint8 TYPES[NUM_OF_PARAMS],
            uint8 NUM_ITEMS[NUM_OF_PARAMS],
            uint8 NUM_STRUCT[NUM_OF_PARAMS],
            uint8 * NUM_MENU,
            const char * PARAMS_STR[NUM_OF_PARAMS],
            uint8 CUSTOM_PARAM_SET[NUM_OF_PARAMS],
            const char * MENU_STR[NUM_OF_PARAMS_MENU] )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values.

5.1.2.26 IMU_reading_info()

```
void IMU_reading_info (
            char * info_string )
```

This function is used to prepare an information string about the IMU sensors last reading.

Parameters

info_string	An array of chars containing the requested information.

5.1.2.27 infoGet()

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

This function sends the firmware information prepared with prepare_general_info or **prepare_counter_info** (p. 51) through the serial port to the user interface. Is used when the ID is specified.

Parameters

info_type	The type of the information needed.

5.1.2.28 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.

5.1.2.29 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.

5.1.2.30 manage_param_list()

```
void manage_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.

5.1.2.31 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.

5.1.2.32 memInit_SoftHandPro()

```
void memInit_SoftHandPro ( )
```

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

5.1.2.33 memRecall()

```
void memRecall ( )
```

This function loads user's settings from the EEPROM.

5.1.2.34 memRestore()

```
uint8 memRestore ( )
```

This function loads default settings from the EEPROM.

Returns

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

5.1.2.35 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.

5.1.2.36 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 information.

5.1.2.37 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 information.

5.1.2.38 prepare_SD_info()

```
void prepare_SD_info (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.1.2.39 prepare_SD_legend()

```
void prepare_SD_legend (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.1.2.40 prepare_SD_param_info()

```
void prepare_SD_param_info (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.1.2.41 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.

5.1.2.42 set_custom_param()

```
void set_custom_param (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, sets the specific 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.

5.1.2.43 setZeros()

```
void setZeros ( )
```

This function sets the encoders zero position.

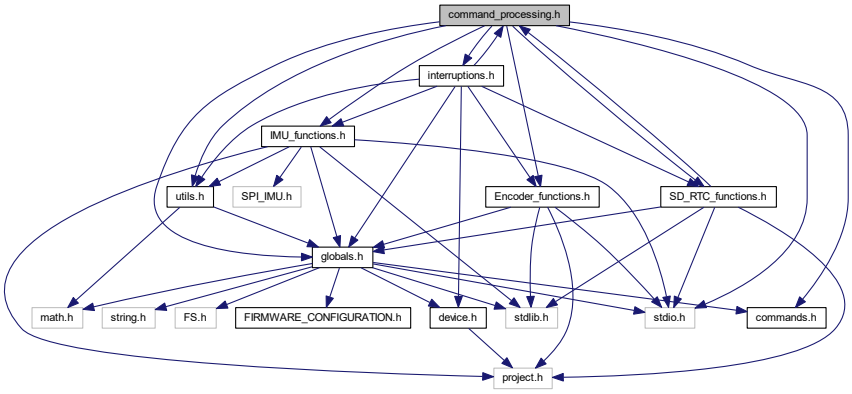
5.2 command_processing.h File Reference

Received commands processing functions.

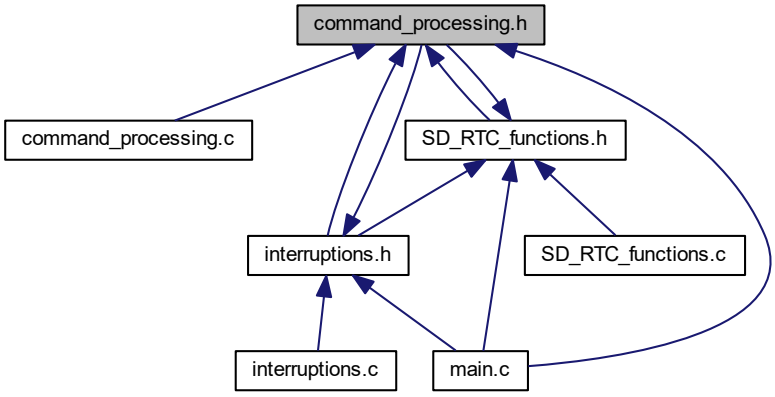
```
#include "globals.h"
#include "IMU_functions.h"
#include "Encoder_functions.h"
```

```
#include "SD_RTC_functions.h"
#include "interruptions.h"
#include "utils.h"
#include "commands.h"
#include <stdio.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 **prepare_SD_info** (char ∗info_string)
- void **prepare_SD_param_info** (char ∗info_string)
- void **prepare_SD_legend** (char ∗info_string)
- void **IMU_reading_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 **manage_param_list** (uint16 index)
- void **get_param_list** (uint8 ∗VAR_P[**NUM_OF_PARAMS**], uint8 TYPES[**NUM_OF_PARAMS**], uint8 NUM_ITEMS[**NUM_OF_PARAMS**], uint8 NUM_STRUCT[**NUM_OF_PARAMS**], uint8 ∗NUM_MENU, const char ∗PARAMS_STR[**NUM_OF_PARAMS**], uint8 CUSTOM_PARAM_SET[**NUM_OF_PARAMS**], const char ∗MENU_STR[**NUM_OF_PARAMS_MENU**])
- void **set_custom_param** (uint16 index)
- void **get_IMU_param_list** (uint16 index)
- void **setZeros** ()
- uint8 **memStore** (int displacement)
- void **memRecall** ()
- uint8 **memRestore** ()
- uint8 **memInit** ()
- void **memInit_SoftHandPro** ()

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_curr_and_meas** ()
- void **cmd_get_velocities** ()
- void **cmd_get_accelerations** ()
- void **cmd_get_currents** ()
- void **cmd_get_currents_for_cuff** ()
- void **cmd_get_emg** ()
- void **cmd_get_activate** ()
- void **cmd_set_baudrate** ()
- void **cmd_get_inputs** ()
- void **cmd_store_params** ()
- void **cmd_ping** ()
- void **cmd_get_imu_readings** ()
- void **cmd_get_encoder_map** ()
- void **cmd_get_encoder_raw** ()
- void **cmd_get_ADC_map** ()
- void **cmd_get_ADC_raw** ()
- void **cmd_get_SD_files** ()

5.2.1 Detailed Description

Received commands processing functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

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_ADC_map()

```
void cmd_get_ADC_map ( )
```

This function gets ADC map

5.2.2.5 cmd_get_ADC_raw()

```
void cmd_get_ADC_raw ( )
```

This function gets Additional emg raw values

5.2.2.6 cmd_get_curr_and_meas()

```
void cmd_get_curr_and_meas ( )
```

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

5.2.2.7 cmd_get_currents()

```
void cmd_get_currents ( )
```

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

5.2.2.8 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. 47) function.

5.2.2.9 cmd_get_emg()

```
void cmd_get_emg ( )
```

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

5.2.2.10 cmd_get_encoder_map()

```
void cmd_get_encoder_map ( )
```

This function gets Encoder map

5.2.2.11 cmd_get_encoder_raw()

```
void cmd_get_encoder_raw ( )
```

This function gets Encoder raw values

5.2.2.12 cmd_get_imu_readings()

```
void cmd_get_imu_readings ( )
```

This function gets IMU readings

5.2.2.13 cmd_get_inputs()

```
void cmd_get_inputs ( )
```

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

5.2.2.14 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

5.2.2.15 cmd_get_SD_files()

```
void cmd_get_SD_files ( )
```

This function gets both SD parameters and data files

5.2.2.16 cmd_get_velocities()

```
void cmd_get_velocities ( )
```

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

5.2.2.17 cmd_ping()

```
void cmd_ping ( )
```

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

5.2.2.18 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.

5.2.2.19 cmd_set_inputs()

```
void cmd_set_inputs ( )
```

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

5.2.2.20 cmd_store_params()

```
void cmd_store_params ( )
```

This function stores the parameters to the EEPROM memory

5.2.2.21 commProcess()

```
void commProcess ( )
```

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

5.2.2.22 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.

5.2.2.23 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.

5.2.2.24 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.

5.2.2.25 get_IMU_param_list()

```
void get_IMU_param_list (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values.

5.2.2.26 get_param_list()

```
void get_param_list (
            uint8 * VAR_P[NUM_OF_PARAMS],
            uint8 TYPES[NUM_OF_PARAMS],
            uint8 NUM_ITEMS[NUM_OF_PARAMS],
            uint8 NUM_STRUCT[NUM_OF_PARAMS],
            uint8 * NUM_MENU,
            const char * PARAMS_STR[NUM_OF_PARAMS],
            uint8 CUSTOM_PARAM_SET[NUM_OF_PARAMS],
            const char * MENU_STR[NUM_OF_PARAMS_MENU] )
```

This function, depending on the **Firmware** (p. 1) received, gets the list of parameters with their values.

5.2.2.27 IMU_reading_info()

```
void IMU_reading_info (
            char * info_string )
```

This function is used to prepare an information string about the IMU sensors last reading.

Parameters

info_string	An array of chars containing the requested information.

5.2.2.28 infoGet()

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

This function sends the firmware information prepared with prepare_general_info or **prepare_counter_info** (p. 51) through the serial port to the user interface. Is used when the ID is specified.

Parameters

info_type	The type of the information needed.

5.2.2.29 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.

5.2.2.30 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.

5.2.2.31 manage_param_list()

```
void manage_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.

5.2.2.32 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.

5.2.2.33 memInit_SoftHandPro()

```
void memInit_SoftHandPro ( )
```

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

5.2.2.34 memRecall()

```
void memRecall ( )
```

This function loads user's settings from the EEPROM.

5.2.2.35 memRestore()

```
uint8 memRestore ( )
```

This function loads default settings from the EEPROM.

Returns

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

5.2.2.36 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.

5.2.2.37 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 information.

5.2.2.38 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 information.

5.2.2.39 prepare_SD_info()

```
void prepare_SD_info (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.2.2.40 prepare_SD_legend()

```
void prepare_SD_legend (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.2.2.41 prepare_SD_param_info()

```
void prepare_SD_param_info (
            char * info_string )
```

This function is used to prepare an information string to be on a SD card

Parameters

info_string	An array of chars containing the requested information.

5.2.2.42 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.

5.2.2.43 set_custom_param()

```
void set_custom_param (
            uint16 index )
```

This function, depending on the **Firmware** (p. 1) received, sets the specific 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.

5.2.2.44 setZeros()

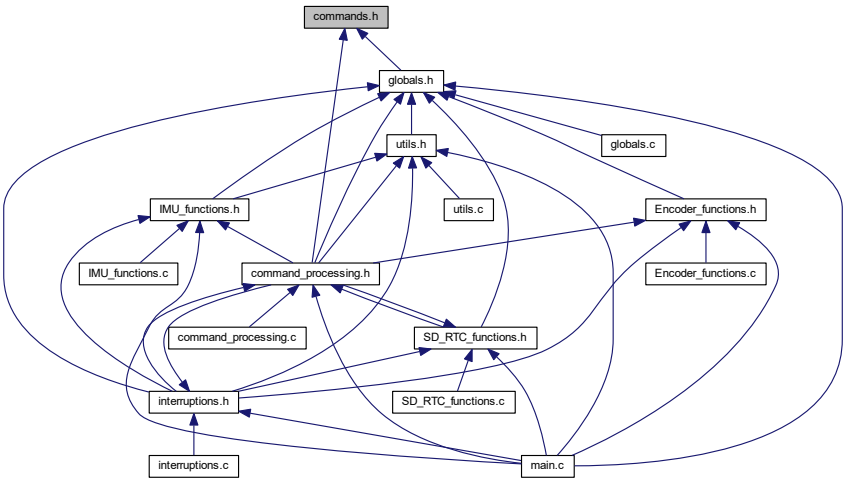
```
void setZeros ( )
```

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

SoftHand Information Strings

- #define **INFO_ALL** 0
  Generic device information.
- #define **CYCLES_INFO** 1
  Cycles counter information.
- #define **GET_SD_PARAM** 2
  Read Firmware Parameters from SD file.
- #define **GET_SD_DATA** 3
  Read Usage Data from SD file.

SoftHand Commands

- #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 **SH_command** {
  CMD_PING = 0, **CMD_SET_ZEROS** = 1, **CMD_STORE_PARAMS** = 3, **CMD_STORE_DEFAULT_PAR↩AMS** = 4,
  CMD_RESTORE_PARAMS = 5, **CMD_GET_INFO** = 6, **CMD_BOOTLOADER** = 9, **CMD_INIT_MEM** = 10, **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_G↩ET_CURRENTS** = 133,
  CMD_GET_CURR_AND_MEAS = 134, **CMD_GET_EMG** = 136, **CMD_GET_VELOCITIES** = 137, **CMD↩_GET_ACCEL** = 139,
  CMD_GET_CURR_DIFF = 140, **CMD_SET_CUFF_INPUTS** = 142, **CMD_SET_BAUDRATE** = 144, **CM↩D_GET_IMU_READINGS** = 161,
  CMD_GET_IMU_PARAM = 162, **CMD_GET_ENCODER_CONF** = 163, **CMD_GET_ENCODER_RAW** = 164, **CMD_GET_ADC_CONF** = 165,
  CMD_GET_ADC_RAW = 166 }
- enum **SH_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 **SH_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 **SH_control_mode** { **CONTROL_ANGLE** = 0, **CONTROL_PWM** = 1, **CONTROL_CURRENT** = 2, **CURR_AND_POS_CONTROL** = 3 }
- enum **motor_supply_type** { **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, **TYPE_STRING** = 9 }

5.3.1 Detailed Description

Definitions for SoftHand commands, parameters and packages.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

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 SH_command

enum **SH_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_BOOTLOADER	Sets the bootloader modality to update the firmware.
CMD_INIT_MEM	Initialize the memory with the defalut values.
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 SoftHand.
CMD_ACTIVATE	Command for activating/deactivating the device.
CMD_GET_ACTIVATE	Command for getting device activation state.
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_GET_EMG	Command for asking device's emg sensors measurements.
CMD_GET_VELOCITIES	Command for asking device's velocity measurements.
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.
CMD_SET_CUFF_INPUTS	Command used to set Cuff device inputs .
CMD_SET_BAUDRATE	Command for setting baudrate of communication.

5.3.2.2 SH_control_mode

enum **SH_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 SH_input_mode

`enum` **SH_input_mode**

Enumerator

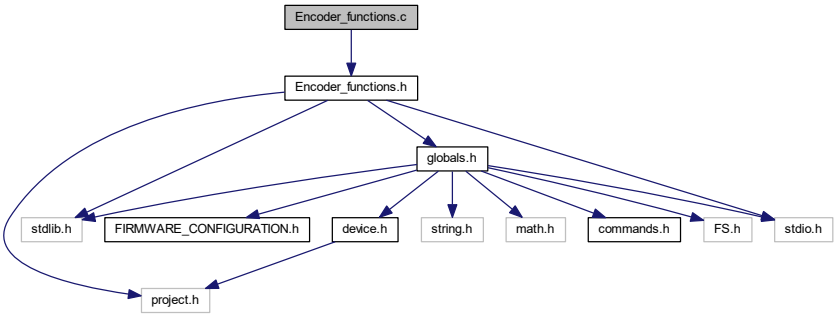
INPUT_MODE_EXTERNAL	References through external commands (default).
INPUT_MODE_ENCODER3	Encoder 3 drives all inputs.
INPUT_MODE_EMG_PROPORTIONAL	Use EMG measure to proportionally. drive the position of the motor.
INPUT_MODE_EMG_INTEGRAL	Use 2 EMG signals to drive motor position.
INPUT_MODE_EMG_FCFS	Use 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.

5.4 Encoder_functions.c File Reference

Implementation of SPI module functions.

```
#include "Encoder_functions.h"
```

Include dependency graph for Encoder_functions.c:



Functions

- void **Change_CS_EncoderLine** (int n)
- void **EncoderReset** ()
- void **InitEncoderGeneral** ()
- void **InitEncoderLine** (uint8 n)
- void **ReadEncoderLine** (int n_encoders, int n_line)

5.4.1 Detailed Description

Implementation of SPI module functions.

Date

February 13, 2019

Author

Centro "E.Piaggio"

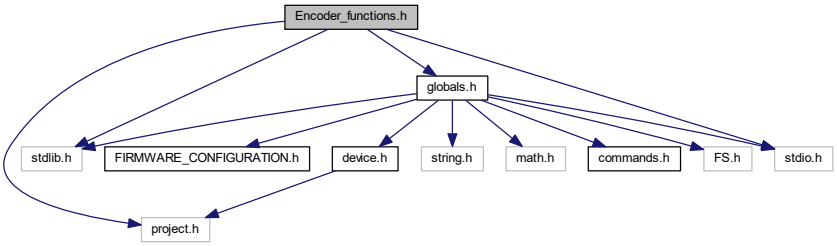
Copyright

5.5 Encoder_functions.h File Reference

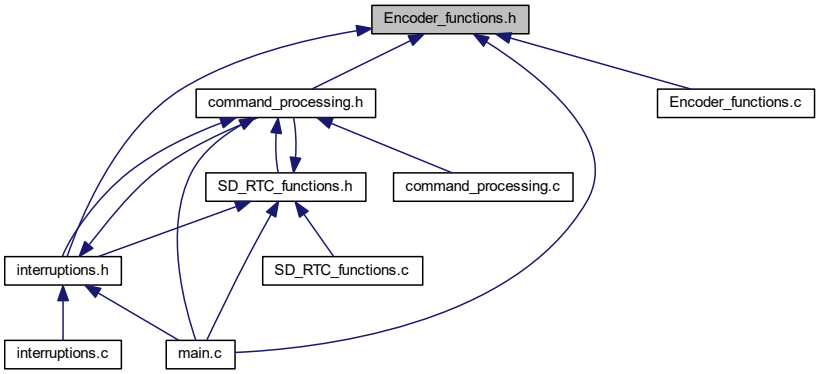
Definition of Encoder module functions.

```
#include <project.h>
#include "globals.h"
#include <stdlib.h>
#include <stdio.h>
```

Include dependency graph for Encoder_functions.h:



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



Functions

- void **EncoderReset** ()
- void **InitEncoderLine** (uint8 n)
- void **InitEncoderGeneral** ()
- void **ReadEncoderLine** (int n_encoders, int n_line)
- void **Change_CS_EncoderLine** (int n)

5.5.1 Detailed Description

Definition of Encoder module functions.

Date

February 13, 2019

Author

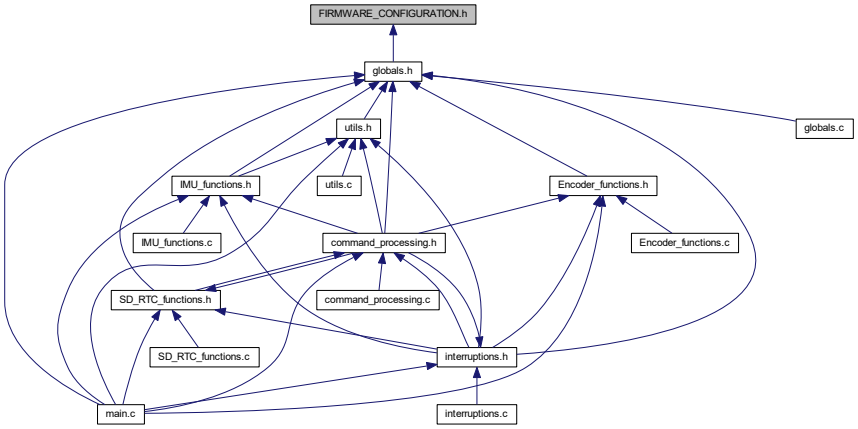
Centro "E.Piaggio"

Copyright

5.6 FIRMWARE_CONFIGURATION.h File Reference

Definitions for SoftHand and Other Devices commands, parameters and packages.

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



Macros

- #define **SOFTHAND_FW**
- #define **VERSION** "SoftHand PRO firmware v. 1.7 (PSoC5)"
- #define **NUM_OF_DEV_PARAMS** (**NUM_OF_PARAMS** - 35)
- #define **NUM_OF_DEV_PARAM_MENUS** (**NUM_OF_PARAMS_MENU** - 2)
- #define **NUM_DEV_IMU** 1

5.6.1 Detailed Description

Definitions for SoftHand and Other Devices commands, parameters and packages.

Date

January 30, 2019

Author

Centro "E.Piaggio"

Copyright

This file is included in the firmware, in its libraries and applications. It contains all definitions that are necessary to discriminate the right firmware.

5.6.2 Macro Definition Documentation

5.6.2.1 NUM_DEV_IMU

```
#define NUM_DEV_IMU 1
```

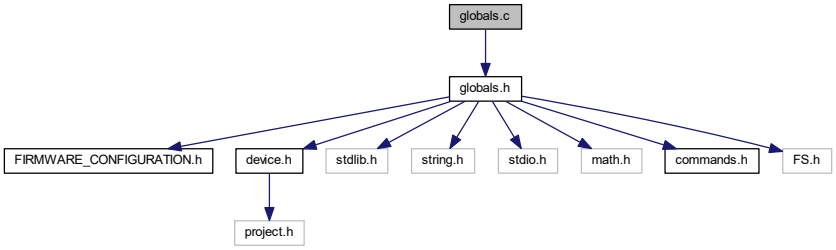
Number of device IMU for SOFTHAND FIRMWARE.

5.7 globals.c File Reference

Global variables.

```
#include "globals.h"
```

Include dependency graph for globals.c:



Variables

- struct **st_ref g_ref** [**NUM_OF_MOTORS**]
- struct **st_ref g_refNew** [**NUM_OF_MOTORS**]
- struct **st_ref g_refOld** [**NUM_OF_MOTORS**]
- struct **st_meas g_meas** [**N_ENCODER_LINE_MAX**]
- struct **st_meas g_measOld** [**N_ENCODER_LINE_MAX**]
- struct **st_emg_meas** g_emg_meas **g_emg_measOld**
- struct **st_data g_rx**
- struct **st_eeprom** g_mem **c_mem**
- struct **st_calib calib**
- struct **st_filter filt_v** [**NUM_OF_MOTORS**]
- struct **st_filter filt_curr_diff** [**NUM_OF_MOTORS**]
- struct **st_filter filt_i** [**NUM_OF_MOTORS**]
- struct **st_filter filt_vel** [**NUM_OF_SENSORS**]
- struct **st_filter filt_emg** [**NUM_OF_INPUT_EMGS**+ **NUM_OF_ADDITIONAL_EMGS**]
- uint16 **timer_value**
- uint16 **timer_value0**
- float **cycle_time**
- int32 **dev_tension** [**NUM_OF_MOTORS**]
- uint8 **dev_pwm_limit** [**NUM_OF_MOTORS**]
- uint8 **dev_pwm_sat** [**NUM_OF_MOTORS**] = {100,100}
- int32 **dev_tension_f** [**NUM_OF_MOTORS**]
- int32 **pow_tension** [**NUM_OF_MOTORS**]
- **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 **cycles_interrupt_flag** = FALSE
- uint8 **maintenance_flag** = FALSE
- CYBIT **can_write** = TRUE
- uint8 **rest_enabled**
- uint8 **forced_open**
- uint8 **battery_low_SoC** = FALSE
- uint8 **change_ext_ref_flag** = FALSE
- CYBIT **reset_PSoC_flag** = FALSE
- int16 **ADC_buf** [NUM_OF_ADC_CHANNELS_MAX]
- uint8 **NUM_OF_ANALOG_INPUTS** = 4
- int8 **pwm_sign**
- uint32 **data_encoder_raw** [**N_ENCODERS_PER_LINE_MAX**]
- uint8 **N_Encoder_Line_Connected** [**N_ENCODER_LINE_MAX**]
- uint16 **Encoder_Value** [**N_ENCODER_LINE_MAX**][**N_ENCODERS_PER_LINE_MAX**]
- uint8 **Encoder_Check** [**N_ENCODER_LINE_MAX**][**N_ENCODERS_PER_LINE_MAX**]
- int32 **rest_pos_curr_ref**
- FS_FILE ∗ **pFile**
- char **sdFile** [100] = ""
- char **sdParam** [100] = ""
- uint8 **N_IMU_Connected**
- uint8 **IMU_connected** [N_IMU_MAX]
- int **imus_data_size**
- int **single_imu_size** [N_IMU_MAX]
- struct **st_imu_data g_imu** [N_IMU_MAX]
- struct **st_imu_data g_imuNew** [N_IMU_MAX]

- uint8 **Accel** [N_IMU_MAX][6]
- uint8 **Gyro** [N_IMU_MAX][6]
- uint8 **Mag** [N_IMU_MAX][6]
- uint8 **MagCal** [N_IMU_MAX][3]
- uint8 **Temp** [N_IMU_MAX][2]
- float **Quat** [N_IMU_MAX][4]

5.7.1 Detailed Description

Global variables.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.7.2 Variable Documentation

5.7.2.1 battery_low_SoC

```
uint8 battery_low_SoC = FALSE
```

Battery low State of Charge flag (re-open terminal device when active).

5.7.2.2 c_mem

```
struct  st_eeprom g_mem c_mem
```

Memory parameters.

5.7.2.3 calib

```
struct  st_calib calib
```

Calibration variables.

5.7.2.4 can_write

```
CYBIT can_write = TRUE
```

Write to EEPROM flag.

5.7.2.5 change_ext_ref_flag

```
uint8 change_ext_ref_flag = FALSE
```

This flag is set when an external reference command is received.

5.7.2.6 cycle_time

```
float cycle_time
```

Variable used to calculate how much time a cycle takes.

5.7.2.7 cycles_interrupt_flag

```
CYBIT cycles_interrupt_flag = FALSE
```

Cycles timer interrupt flag enabler.

5.7.2.8 cycles_status

```
counter_status CYDATA cycles_status =  NONE
```

Cycles counter state machine status.

5.7.2.9 dev_pwm_limit

```
uint8 dev_pwm_limit[ NUM_OF_MOTORS]
```

Device pwm limit. It may change during execution.

5.7.2.10 dev_pwm_sat

```
uint8 dev_pwm_sat[ NUM_OF_MOTORS] = {100,100}
```

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

5.7.2.11 dev_tension

```
int32 dev_tension[ NUM_OF_MOTORS]
```

Power supply tension.

5.7.2.12 dev_tension_f

```
int32 dev_tension_f[ NUM_OF_MOTORS]
```

Filtered power supply tension.

5.7.2.13 emg_1_status

```
emg_status CYDATA emg_1_status =  RESET
```

First EMG sensor status.

5.7.2.14 emg_2_status

```
emg_status CYDATA emg_2_status =  RESET
```

Second EMG sensor status.

5.7.2.15 filt_emg

```
struct  st_filter filt_emg[ NUM_OF_INPUT_EMGS+ NUM_OF_ADDITIONAL_EMGS]
```

EMG filter variables.

5.7.2.16 filt_i

```
struct  st_filter filt_i[ NUM_OF_MOTORS]
```

Voltage and current filter variables.

5.7.2.17 filt_vel

```
struct  st_filter filt_vel[ NUM_OF_SENSORS]
```

Velocity filter variables.

5.7.2.18 forced_open

```
uint8 forced_open
```

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

5.7.2.19 g_emg_measOld

```
struct  st_emg_meas g_emg_meas g_emg_measOld
```

EMG Measurements.

5.7.2.20 g_measOld

```
struct  st_meas g_measOld[ N_ENCODER_LINE_MAX]
```

Measurements.

5.7.2.21 g_refOld

```
struct  st_ref g_refOld[ NUM_OF_MOTORS]
```

Reference variables.

5.7.2.22 g_rx

```
struct  st_data g_rx
```

Incoming/Outcoming data.

5.7.2.23 interrupt_flag

```
CYBIT interrupt_flag = FALSE
```

Interrupt flag enabler.

5.7.2.24 maintenance_flag

```
uint8 maintenance_flag = FALSE
```

Maintenance flag.

5.7.2.25 NUM_OF_ANALOG_INPUTS

```
uint8 NUM_OF_ANALOG_INPUTS = 4
```

ADC measurements buffer.

5.7.2.26 pow_tension

```
int32 pow_tension[ NUM_OF_MOTORS]
```

Computed power supply tension.

5.7.2.27 pwm_sign

```
int8 pwm_sign
```

ADC currently configured channels. Sign of pwm driven. Used to obtain current sign.

5.7.2.28 reset_last_value_flag

```
CYBIT reset_last_value_flag
```

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

5.7.2.29 reset_PSoC_flag

```
CYBIT reset_PSoC_flag = FALSE
```

This flag is set when a board fw reset is necessary.

5.7.2.30 rest_enabled

```
uint8 rest_enabled
```

Rest position flag.

5.7.2.31 rest_pos_curr_ref

```
int32 rest_pos_curr_ref
```

Rest position current reference.

5.7.2.32 tension_valid

```
CYBIT tension_valid
```

Tension validation bit.

5.7.2.33 timer_value

```
uint16 timer_value
```

End time of the firmware main loop.

5.7.2.34 timer_value0

```
uint16 timer_value0
```

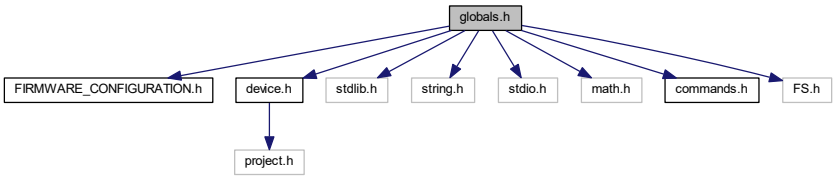
Start time of the firmware main loop.

5.8 globals.h File Reference

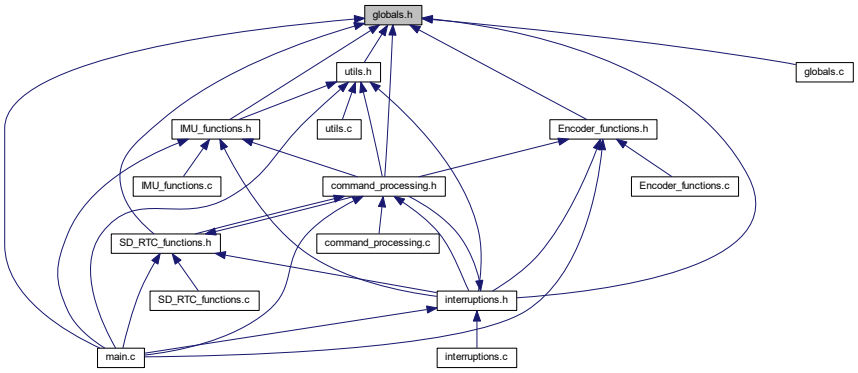
Global definitions and macros are set in this file.

```
#include "FIRMWARE_CONFIGURATION.h"
#include "device.h"
#include "stdlib.h"
#include "string.h"
#include "stdio.h"
#include "math.h"
#include "commands.h"
#include "FS.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_emg_meas**
- struct **st_data**

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

  EEPROM stored structures.

- struct **st_device**
- struct **st_motor**
- struct **st_encoder**
- struct **st_emg**
- struct **st_imu**
- struct **st_expansion**
- struct **st_user**
- struct **st_SH_spec**
- struct **st_eeprom**
- struct **st_imu_data**
- struct **st_filter**

  Filter structure.
- struct **st_calib**

  Hand calibration structure.

Macros

- #define **NUM_OF_MOTORS** 2
- #define **NUM_OF_SENSORS** 3
- #define **NUM_OF_INPUT_EMGS** 2
- #define **NUM_OF_ADDITIONAL_EMGS** 6
- #define **NUM_OF_ADC_CHANNELS_MAX** (4+ **NUM_OF_INPUT_EMGS**+ **NUM_OF_ADDITIONAL_EM↩ GS**)
- #define **NUM_OF_PARAMS** 71
- #define **NUM_OF_PARAMS_MENU** 10
- #define **N_IMU_MAX** 5
- #define **NUM_OF_IMU_DATA** 5
- #define **N_ENCODER_LINE_MAX** 2
- #define **N_ENCODERS_PER_LINE_MAX** 5
- #define **N_ENCODERS NUM_OF_SENSORS**
- #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 **SPI_DELAY_LOW** 10
- #define **SPI_DELAY_HIGH** 100
- #define **EXP_NONE** 0
- #define **EXP_SD_RTC** 1
- #define **EXP_WIFI** 2
- #define **EXP_OTHER** 3
- #define **DRIVER_MC33887** 0
- #define **DRIVER_VNH5019** 1
- #define **RIGHT_HAND** 0

- #define **LEFT_HAND** 1
- #define **NUM_OF_USERS** 3
- #define **GENERIC_USER** 0
- #define **MARIA** 1
- #define **ROZA** 2
- #define **SOFTHAND_PRO** 0
- #define **GENERIC_2_MOTORS** 1
- #define **CUFF** 2
- #define **SH_N1** 35
- #define **SH_N2** 3
- #define **SH_I1** -1
- #define **ST_DEVICE** 0
- #define **ST_MOTOR** 10
- #define **ST_ENCODER** 20
- #define **ST_EMG** 30
- #define **ST_IMU** 40
- #define **ST_EXPANSION** 50
- #define **ST_USER** 60
- #define **ST_SH_SPEC** 70
- #define **FALSE** 0
- #define **TRUE** 1
- #define **DEFAULT_EEPROM_DISPLACEMENT** 50
- #define **EEPROM_BYTES_ROW** 16
- #define **EEPROM_COUNTERS_ROWS** 5
- #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 **REST_POS_ERR_THR_GAIN** 10
- #define **POS_INTEGRAL_SAT_LIMIT** 50000000
- #define **CURR_INTEGRAL_SAT_LIMIT** 100000
- #define **PWM_RATE_LIMITER_MAX** 1
- #define **SAFE_STARTUP_MOTOR_READINGS** 8000
- #define **LOOKUP_DIM** 6
- #define **PREREVISION_CYCLES** 400000

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** [**NUM_OF_MOTORS**]
- struct **st_ref g_refNew** [**NUM_OF_MOTORS**]
- struct **st_ref g_refOld** [**NUM_OF_MOTORS**]
- struct **st_meas g_meas** [**N_ENCODER_LINE_MAX**]
- struct **st_meas g_measOld** [**N_ENCODER_LINE_MAX**]
- struct **st_emg_meas** g_emg_meas **g_emg_measOld**
- struct **st_data g_rx**
- struct **st_eeprom** g_mem **c_mem**
- struct **st_calib calib**
- struct **st_filter filt_v** [**NUM_OF_MOTORS**]
- struct **st_filter filt_curr_diff** [**NUM_OF_MOTORS**]
- struct **st_filter filt_i** [**NUM_OF_MOTORS**]
- struct **st_filter filt_vel** [**NUM_OF_SENSORS**]
- struct **st_filter filt_emg** [**NUM_OF_INPUT_EMGS**+ **NUM_OF_ADDITIONAL_EMGS**]
- uint16 **timer_value**
- uint16 **timer_value0**
- float **cycle_time**
- int32 **dev_tension** [**NUM_OF_MOTORS**]
- uint8 **dev_pwm_limit** [**NUM_OF_MOTORS**]
- uint8 **dev_pwm_sat** [**NUM_OF_MOTORS**]
- int32 **dev_tension_f** [**NUM_OF_MOTORS**]
- int32 **pow_tension** [**NUM_OF_MOTORS**]
- **counter_status** CYDATA **cycles_status**
- **emg_status** CYDATA **emg_1_status**
- **emg_status** CYDATA **emg_2_status**
- CYBIT **reset_last_value_flag**
- CYBIT **tension_valid**
- CYBIT **interrupt_flag**
- CYBIT **cycles_interrupt_flag**
- uint8 **maintenance_flag**
- CYBIT **can_write**
- uint8 **rest_enabled**
- uint8 **forced_open**
- uint8 **battery_low_SoC**
- uint8 **change_ext_ref_flag**
- CYBIT **reset_PSoC_flag**
- int16 **ADC_buf** [NUM_OF_ADC_CHANNELS_MAX]
- uint8 **NUM_OF_ANALOG_INPUTS**
- int8 **pwm_sign**
- uint32 **data_encoder_raw** [**N_ENCODERS_PER_LINE_MAX**]
- uint8 **N_Encoder_Line_Connected** [**N_ENCODER_LINE_MAX**]
- uint16 **Encoder_Value** [**N_ENCODER_LINE_MAX**][**N_ENCODERS_PER_LINE_MAX**]
- uint8 **Encoder_Check** [**N_ENCODER_LINE_MAX**][**N_ENCODERS_PER_LINE_MAX**]
- int32 **rest_pos_curr_ref**
- FS_FILE ∗ **pFile**
- char **sdFile** [100]
- char **sdParam** [100]
- uint8 **N_IMU_Connected**
- uint8 **IMU_connected** [N_IMU_MAX]
- int **imus_data_size**
- int **single_imu_size** [N_IMU_MAX]
- struct **st_imu_data g_imu** [N_IMU_MAX]
- struct **st_imu_data g_imuNew** [N_IMU_MAX]

- uint8 **Accel** [N_IMU_MAX][6]
- uint8 **Gyro** [N_IMU_MAX][6]
- uint8 **Mag** [N_IMU_MAX][6]
- uint8 **MagCal** [N_IMU_MAX][3]
- uint8 **Temp** [N_IMU_MAX][2]
- float **Quat** [N_IMU_MAX][4]

5.8.1 Detailed Description

Global definitions and macros are set in this file.

Date

February 01, 2018

Author

Centro "E.Piaggio"

Copyright

5.8.2 Macro Definition Documentation

5.8.2.1 ANTI_WINDUP

```
#define ANTI_WINDUP 1000
```

Anti windup saturation.

5.8.2.2 CALIBRATION_DIV

```
#define CALIBRATION_DIV 10
```

Frequency divisor for hand calibration (100Hz).

5.8.2.3 COUNTER_INC

```
#define COUNTER_INC 2
```

Counter cycle increment.

5.8.2.4 CURR_INTEGRAL_SAT_LIMIT

```
#define CURR_INTEGRAL_SAT_LIMIT 100000
```

Anti windup on current control.

5.8.2.5 CURRENT_HYSTERESIS

```
#define CURRENT_HYSTERESIS 10
```

milliAmperes of hysteresis for current control.

5.8.2.6 DEFAULT_CURRENT_LIMIT

```
#define DEFAULT_CURRENT_LIMIT 1500
```

Default Current limit, 0 stands for unlimited.

5.8.2.7 DEFAULT_EEPROM_DISPLACEMENT

```
#define DEFAULT_EEPROM_DISPLACEMENT 50
```

Number of pages occupied by the EEPROM.

5.8.2.8 DIV_INIT_VALUE

```
#define DIV_INIT_VALUE 1
```

Initial value for hand counter calibration.

5.8.2.9 EEPROM_BYTES_ROW

```
#define EEPROM_BYTES_ROW 16
```

EEPROM number of bytes per row.

5.8.2.10 EEPROM_COUNTERS_ROWS

```
#define EEPROM_COUNTERS_ROWS 5
```

EEPROM number of rows dedicated to store counters.

5.8.2.11 EMG_SAMPLE_TO_DISCARD

```
#define EMG_SAMPLE_TO_DISCARD 500
```

Number of sample to discard before calibration.

5.8.2.12 LOOKUP_DIM

```
#define LOOKUP_DIM 6
```

Dimension of the current lookup table.

5.8.2.13 N_ENCODER_LINE_MAX

```
#define N_ENCODER_LINE_MAX 2
```

Max number of CS lines which can contain encoders.

5.8.2.14 N_ENCODERS_PER_LINE_MAX

```
#define N_ENCODERS_PER_LINE_MAX 5
```

Max number of encoders per line.

5.8.2.15 NUM_OF_ADDITIONAL_EMGS

```
#define NUM_OF_ADDITIONAL_EMGS 6
```

Number of additional emg channels.

5.8.2.16 NUM_OF_INPUT_EMGS

```
#define NUM_OF_INPUT_EMGS 2
```

Number of emg channels.

5.8.2.17 NUM_OF_MOTORS

```
#define NUM_OF_MOTORS 2
```

Number of motors.

5.8.2.18 NUM_OF_PARAMS

```
#define NUM_OF_PARAMS 71
```

Number of parameters saved in the EEPROM.

5.8.2.19 NUM_OF_PARAMS_MENU

```
#define NUM_OF_PARAMS_MENU 10
```

Number of parameters menu.

5.8.2.20 NUM_OF_SENSORS

```
#define NUM_OF_SENSORS 3
```

Number of encoders.

5.8.2.21 POS_INTEGRAL_SAT_LIMIT

```
#define POS_INTEGRAL_SAT_LIMIT 50000000
```

Anti windup on position control.

5.8.2.22 PREREVISION_CYCLES

```
#define PREREVISION_CYCLES 400000
```

Number of SoftHand Pro cycles before maintenance.

5.8.2.23 PWM_MAX_VALUE

```
#define PWM_MAX_VALUE 100
```

Maximum value of the PWM signal.

5.8.2.24 RECEIVE

```
#define RECEIVE 3
```

Package data receiving status.

5.8.2.25 REST_POS_ERR_THR_GAIN

```
#define REST_POS_ERR_THR_GAIN 10
```

Gain related to stop condition threshold in rest position routine.

5.8.2.26 SAFE_STARTUP_MOTOR_READINGS

```
#define SAFE_STARTUP_MOTOR_READINGS 8000
```

Number of encoder readings after position reconstruction before activating motor.

5.8.2.27 SAMPLES_FOR_EMG_MEAN

```
#define SAMPLES_FOR_EMG_MEAN 1000
```

Number of samples used to mean emg values.

5.8.2.28 SAMPLES_FOR_MEAN

```
#define SAMPLES_FOR_MEAN 100
```

Number of samples used to mean current values.

5.8.2.29 SH_I1

```
#define SH_I1 -1
```

First gear invariant value in SoftHandPro device.

5.8.2.30 SH_N1

```
#define SH_N1 35
```

Number of teeth of the first encoder gear in SoftHandPro device.

5.8.2.31 SH_N2

```
#define SH_N2 3
```

Number of teeth of the second encoder gear in SoftHandPro device.

5.8.2.32 STATE_ACTIVE

```
#define STATE_ACTIVE 1
```

Closed SoftHand position / EMG Active.

5.8.2.33 STATE_INACTIVE

```
#define STATE_INACTIVE 0
```

Open SoftHand position / EMG Inactive.

5.8.2.34 UNLOAD

```
#define UNLOAD 4
```

Package data flush status.

5.8.2.35 WAIT_ID

```
#define WAIT_ID 1
```

Package ID waiting status.

5.8.2.36 WAIT_LENGTH

```
#define WAIT_LENGTH 2
```

Package lenght waiting status.

5.8.2.37 WAIT_START

```
#define WAIT_START 0
```

Package start waiting status.

5.8.3 Enumeration Type Documentation

5.8.3.1 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.

5.8.3.2 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.

5.8.4 Variable Documentation

5.8.4.1 battery_low_SoC

```
uint8 battery_low_SoC
```

Battery low State of Charge flag (re-open terminal device when active).

5.8.4.2 c_mem

```
struct  st_eeprom g_mem c_mem
```

Memory parameters.

5.8.4.3 calib

```
struct  st_calib calib
```

Calibration variables.

5.8.4.4 can_write

```
CYBIT can_write
```

Write to EEPROM flag.

5.8.4.5 change_ext_ref_flag

```
uint8 change_ext_ref_flag
```

This flag is set when an external reference command is received.

5.8.4.6 cycle_time

```
float cycle_time
```

Variable used to calculate how much time a cycle takes.

5.8.4.7 cycles_interrupt_flag

```
CYBIT cycles_interrupt_flag
```

Cycles timer interrupt flag enabler.

5.8.4.8 cycles_status

```
counter_status CYDATA cycles_status
```

Cycles counter state machine status.

5.8.4.9 dev_pwm_limit

```
uint8 dev_pwm_limit[ NUM_OF_MOTORS]
```

Device pwm limit. It may change during execution.

5.8.4.10 dev_pwm_sat

```
uint8 dev_pwm_sat[ NUM_OF_MOTORS]
```

Device pwm saturation.

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

5.8.4.11 dev_tension

```
int32 dev_tension[ NUM_OF_MOTORS]
```

Power supply tension.

5.8.4.12 dev_tension_f

```
int32 dev_tension_f[ NUM_OF_MOTORS]
```

Filtered power supply tension.

5.8.4.13 emg_1_status

```
emg_status CYDATA emg_1_status
```

First EMG sensor status.

5.8.4.14 emg_2_status

```
emg_status CYDATA emg_2_status
```

Second EMG sensor status.

5.8.4.15 filt_emg

```
struct  st_filter filt_emg[ NUM_OF_INPUT_EMGS+ NUM_OF_ADDITIONAL_EMGS]
```

EMG filter variables.

5.8.4.16 filt_i

```
struct  st_filter filt_i[ NUM_OF_MOTORS]
```

Voltage and current filter variables.

5.8.4.17 filt_vel

```
struct  st_filter filt_vel[ NUM_OF_SENSORS]
```

Velocity filter variables.

5.8.4.18 forced_open

```
uint8 forced_open
```

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

5.8.4.19 g_emg_measOld

```
struct  st_emg_meas g_emg_meas g_emg_measOld
```

EMG Measurements.

5.8.4.20 g_measOld

```
struct  st_meas g_measOld[ N_ENCODER_LINE_MAX]
```

Measurements.

5.8.4.21 g_refOld

```
struct  st_ref g_refOld[ NUM_OF_MOTORS]
```

Reference variables.

5.8.4.22 g_rx

```
struct  st_data g_rx
```

Incoming/Outcoming data.

5.8.4.23 interrupt_flag

```
CYBIT interrupt_flag
```

Interrupt flag enabler.

5.8.4.24 maintenance_flag

```
uint8 maintenance_flag
```

Maintenance flag.

5.8.4.25 NUM_OF_ANALOG_INPUTS

```
uint8 NUM_OF_ANALOG_INPUTS
```

ADC measurements buffer (sizeof buffer equal to maximum number of ADC channels).

ADC measurements buffer.

5.8.4.26 pow_tension

```
int32 pow_tension[ NUM_OF_MOTORS]
```

Computed power supply tension.

5.8.4.27 pwm_sign

```
int8 pwm_sign
```

ADC currently configured channels. Sign of pwm driven. Used to obtain current sign.

5.8.4.28 reset_last_value_flag

```
CYBIT reset_last_value_flag
```

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

5.8.4.29 reset_PSoC_flag

```
CYBIT reset_PSoC_flag
```

This flag is set when a board fw reset is necessary.

5.8.4.30 rest_enabled

```
uint8 rest_enabled
```

Rest position flag.

5.8.4.31 rest_pos_curr_ref

```
int32 rest_pos_curr_ref
```

Rest position current reference.

5.8.4.32 tension_valid

```
CYBIT tension_valid
```

Tension validation bit.

5.8.4.33 timer_value

```
uint16 timer_value
```

End time of the firmware main loop.

5.8.4.34 timer_value0

```
uint16 timer_value0
```

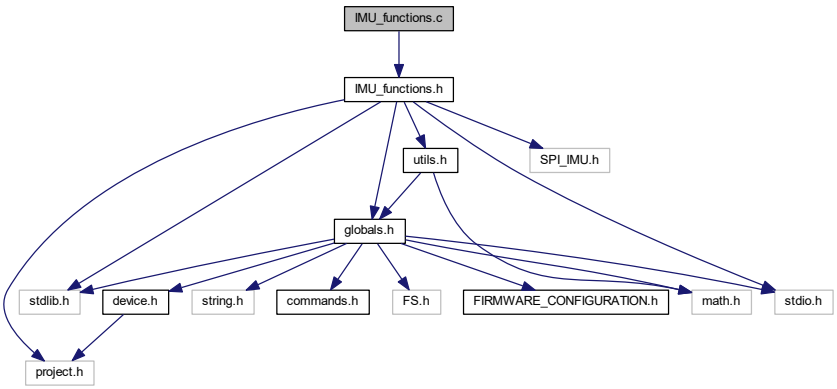
Start time of the firmware main loop.

5.9 IMU_functions.c File Reference

Implementation of IMU module functions.

```
#include "IMU_functions.h"
```

Include dependency graph for IMU_functions.c:



Functions

- void **ImusReset** ()
- void **InitIMU** ()
- void **InitIMUMagCal** ()
- void **ChipSelectorIMU** (int n)
- void **InitIMUgeneral** ()
- void **ReadIMU** (int n)
- void **ReadAcc** (int n)
- void **ReadGyro** (int n)
- void **ReadMag** (int n)
- void **ReadMagCal** (int n)
- void **ReadQuat** (int n)
- void **ReadAllIMUs** ()
- void **ReadTemp** (int n)
- void **WriteControlRegisterIMU** (uint8 address, uint8 dta)
- uint8 **ReadControlRegisterIMU** (uint8 address)
- void **SPI_delay** ()

Variables

- uint8 **Accel** [N_IMU_MAX][6]
- uint8 **Gyro** [N_IMU_MAX][6]
- uint8 **Mag** [N_IMU_MAX][6]
- uint8 **MagCal** [N_IMU_MAX][3]

5.9.1 Detailed Description

Implementation of IMU module functions.

Date

February 01, 2018

Author

Centro "E.Piaggio"

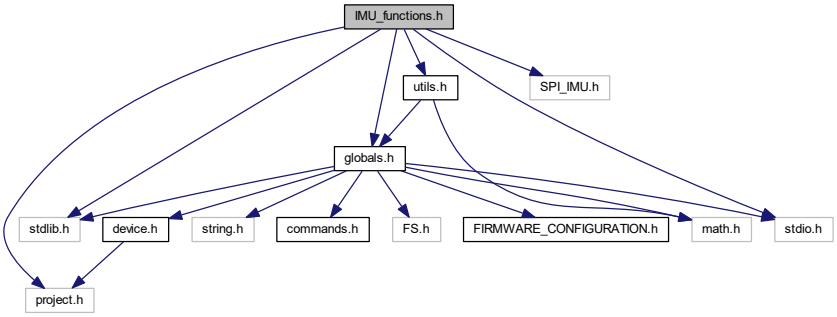
Copyright

5.10 IMU_functions.h File Reference

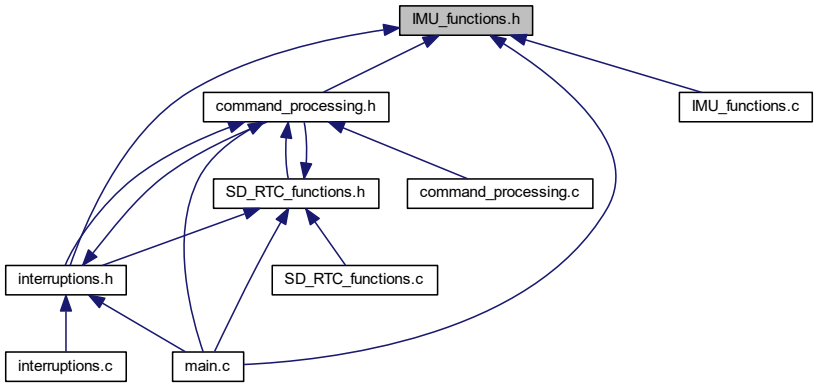
Definition of IMU module functions.

```
#include <project.h>
#include "globals.h"
#include <stdlib.h>
#include <stdio.h>
#include "utils.h"
#include <SPI_IMU.h>
```

Include dependency graph for IMU_functions.h:



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



Macros

- #define **MPU9250_RCR** 0x80
- #define **MPU9250_WCR** 0x00
- #define **MPU9250_CONFIG** 0x1A
- #define **MPU9250_GYRO_CONFIG** 0x1B
- #define **MPU9250_ACCEL_CONFIG** 0x1C
- #define **MPU9250_ACCEL_CONFIG2** 0x1D

- #define **MPU9250_ACCEL_XOUT_H** 0x3B
- #define **MPU9250_ACCEL_XOUT_L** 0x3C
- #define **MPU9250_ACCEL_YOUT_H** 0x3D
- #define **MPU9250_ACCEL_YOUT_L** 0x3E
- #define **MPU9250_ACCEL_ZOUT_H** 0x3F
- #define **MPU9250_ACCEL_ZOUT_L** 0x40
- #define **MPU9250_TEMP_OUT_H** 0x41
- #define **MPU9250_TEMP_OUT_L** 0x42
- #define **MPU9250_GYRO_XOUT_H** 0x43
- #define **MPU9250_GYRO_XOUT_L** 0x44
- #define **MPU9250_GYRO_YOUT_H** 0x45
- #define **MPU9250_GYRO_YOUT_L** 0x46
- #define **MPU9250_GYRO_ZOUT_H** 0x47
- #define **MPU9250_GYRO_ZOUT_L** 0x48
- #define **MPU9250_USER_CTRL** 0x6A
- #define **MPU9250_PWR_MGMT_1** 0x6B
- #define **MPU9250_WHO_AM_I** 0x75
- #define **MPU9250_FIFO_EN** 0x23
- #define **MPU9250_I2C_MST_CTRL** 0x24
- #define **MPU9250_I2C_SLV0_ADDR** 0x25
- #define **MPU9250_I2C_SLV0_REG** 0x26
- #define **MPU9250_I2C_SLV0_CTRL** 0x27
- #define **MPU9250_I2C_SLV1_ADDR** 0x28
- #define **MPU9250_I2C_SLV1_REG** 0x29
- #define **MPU9250_I2C_SLV1_CTRL** 0x2A
- #define **MPU9250_EXT_SENS_DATA_00** 0x49
- #define **MPU9250_EXT_SENS_DATA_01** 0x4A
- #define **MPU9250_EXT_SENS_DATA_02** 0x4B
- #define **MPU9250_EXT_SENS_DATA_03** 0x4C
- #define **MPU9250_EXT_SENS_DATA_04** 0x4D
- #define **MPU9250_EXT_SENS_DATA_05** 0x4E
- #define **MPU9250_EXT_SENS_DATA_06** 0x4F
- #define **MPU9250_EXT_SENS_DATA_07** 0x50
- #define **MPU9250_I2C_SLV0_D0** 0x63
- #define **MPU9250_I2C_SLV1_D0** 0x64
- #define **MPU9250_I2C_MST_DELAY_CTRL** 0x67
- #define **AK8936_ADDRESS** 0x0C
- #define **AK8936_WIA** 0x00
- #define **AK8936_INFO** 0x01
- #define **AK8936_ST1** 0x02
- #define **AK8936_XOUT_L** 0x03
- #define **AK8936_XOUT_H** 0x04
- #define **AK8936_YOUT_L** 0x05
- #define **AK8936_YOUT_H** 0x06
- #define **AK8936_ZOUT_L** 0x07
- #define **AK8936_ZOUT_H** 0x08
- #define **AK8936_ST2** 0x09
- #define **AK8936_CNTL** 0x0A
- #define **AK8963_CNTL2** 0x0B
- #define **AK8936_ASTC** 0x0C
- #define **AK8936_I2CDIS** 0x0F
- #define **ACC_SF_2G** 0x00
- #define **ACC_SF_4G** 0x08
- #define **ACC_SF_8G** 0x10
- #define **ACC_SF_16G** 0x18

- #define **GYRO_SF_250** 0x00
- #define **GYRO_SF_500** 0x80
- #define **GYRO_SF_2000** 0x18
- #define **G_TO_MS2** 9.79
- #define **DEG_TO_RAD** (3.14159265359 / 180.0)
- #define **LP_ACC_FREQ_460** 0x00
- #define **LP_ACC_FREQ_184** 0x01
- #define **LP_ACC_FREQ_92** 0x02
- #define **LP_ACC_FREQ_41** 0x03
- #define **LP_ACC_FREQ_20** 0x04
- #define **LP_ACC_FREQ_10** 0x05
- #define **LP_ACC_FREQ_5** 0x06
- #define **TICK2GYRO** 0.000133158
- #define **TICK2ACC** 0.000061037
- #define **BETA** 100000.0
- #define **GYRO_THR** 0.2618

Functions

- void **ImusReset** ()
- void **InitIMU** ()
- void **InitIMUMagCal** ()
- void **InitIMUgeneral** ()
- void **ReadAcc** (int n)
- void **ReadGyro** (int n)
- void **ReadMag** (int n)
- void **ReadMagCal** (int n)
- void **ReadQuat** (int n)
- void **ReadTemp** (int n)
- void **ReadIMU** (int n)
- void **ReadAllIMUs** ()
- uint8 **ReadControlRegisterIMU** (uint8 address)
- void **WriteControlRegisterIMU** (uint8 address, uint8 dta)
- void **ChipSelectorIMU** (int n)
- void **SPI_delay** ()

5.10.1 Detailed Description

Definition of IMU module functions.

Date

February 01, 2018

Author

Centro "E.Piaggio"

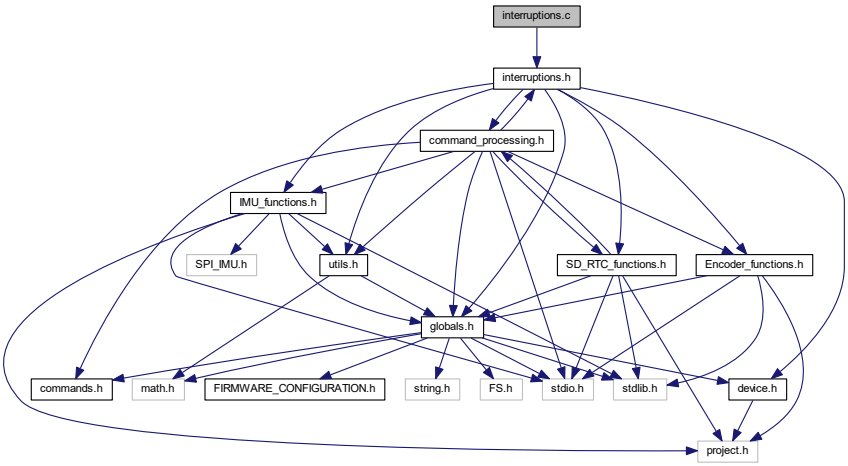
Copyright

5.11 interruptions.c File Reference

Interruption handling and firmware core functions.

```
#include "interruptions.h"
```

Include dependency graph for interruptions.c:



Functions

- **CY_ISR** (ISR_RS485_RX_ExInterrupt)
- **CY_ISR** (ISR_CYCLES_Handler)
- void **interrupt_manager** ()
- void **function_scheduler** (void)
- void **motor_control_SH** ()
- void **motor_control_generic** (uint8 idx)
- void **encoder_reading_SPI** (uint8 n_line, uint8 assoc_motor)
- void **analog_read_end** ()
- void **overcurrent_control** ()
- void **pwm_limit_search** (uint8 mot_idx)
- void **cycles_counter_update** ()
- void **save_cycles_eeprom** ()

Variables

- static const uint8 **pwm_preload_values** [29]

5.11.1 Detailed Description

Interruption handling and firmware core functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.11.2 Function Documentation

5.11.2.1 analog_read_end()

```
void analog_read_end ( )
```

This function executes and terminates the analog readings.

5.11.2.2 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.

5.11.2.3 encoder_reading_SPI()

```
void encoder_reading_SPI (
            uint8 n_line,
            uint8 assoc_motor )
```

This functions reads the value from all the connected encoders.

5.11.2.4 function_scheduler()

```
void function_scheduler (
            void  )
```

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

5.11.2.5 interrupt_manager()

```
void interrupt_manager ( )
```

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

5.11.2.6 motor_control_generic()

```
void motor_control_generic (
            uint8 index )
```

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

5.11.2.7 motor_control_SH()

```
void motor_control_SH ( )
```

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

5.11.2.8 overcurrent_control()

```
void overcurrent_control ( )
```

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

5.11.2.9 pwm_limit_search()

```
void pwm_limit_search (
            uint8 mot_idx )
```

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.

5.11.2.10 save_cycles_eeprom()

```
void save_cycles_eeprom ( )
```

This function saves cycles counters variables into EEPROM memory.

5.11.3 Variable Documentation

5.11.3.1 pwm_preload_values

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

Initial value:

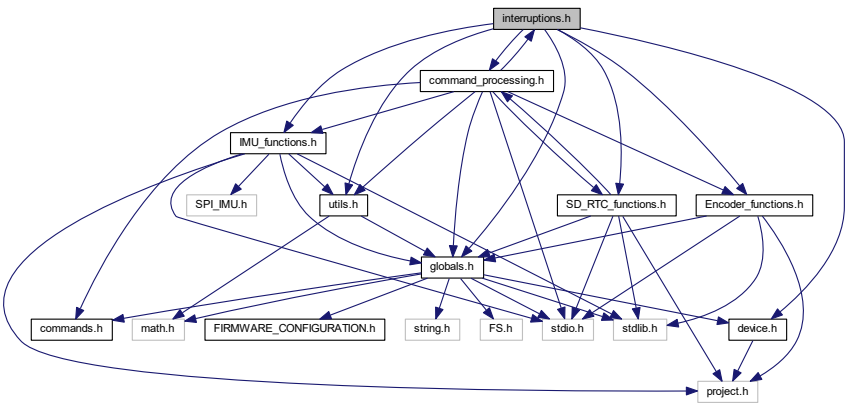
```
= {100,
                                                83,
                                                78,
                                                76,
                                                74,
                                                72,
                            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

5.12 interruptions.h File Reference

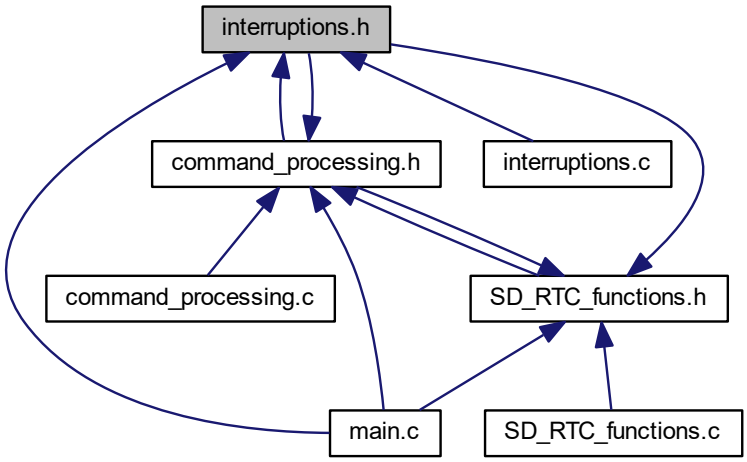
Interruptions header file.

```
#include "device.h"
#include "command_processing.h"
#include "IMU_functions.h"
#include "Encoder_functions.h"
#include "SD_RTC_functions.h"
#include "globals.h"
#include "utils.h"
```

Include dependency graph for interruptions.h:



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



Functions

- void **motor_control_generic** (uint8 index)
- void **save_cycles_eeprom** ()

Interruptions

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

General function scheduler

- void **function_scheduler** (void)

Encoder reading SPI function

- void **encoder_reading_SPI** (uint8 n_line, uint8 assoc_motor)

Motor control function

- void **motor_control_SH** ()

Analog readings

- void **analog_read_end** ()

Interrupt manager

- void **interrupt_manager** ()

Utility functions

- void **pwm_limit_search** (uint8 mot_idx)
- void **overcurrent_control** ()
- void **cycles_counter_update** ()

5.12.1 Detailed Description

Interruptions header file.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.12.2 Function Documentation

5.12.2.1 analog_read_end()

```
void analog_read_end ( )
```

This function executes and terminates the analog readings.

5.12.2.2 CY_ISR_PROTO() [1/2]

```
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.

5.12.2.3 CY_ISR_PROTO() [2/2]

```
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.

5.12.2.4 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.

5.12.2.5 encoder_reading_SPI()

```
void encoder_reading_SPI (
            uint8 n_line,
            uint8 assoc_motor )
```

This functions reads the value from all the connected encoders.

5.12.2.6 function_scheduler()

```
void function_scheduler (
            void  )
```

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

5.12.2.7 interrupt_manager()

```
void interrupt_manager ( )
```

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

5.12.2.8 motor_control_generic()

```
void motor_control_generic (
            uint8 index )
```

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

5.12.2.9 motor_control_SH()

```
void motor_control_SH ( )
```

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

5.12.2.10 overcurrent_control()

```
void overcurrent_control ( )
```

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

5.12.2.11 pwm_limit_search()

```
void pwm_limit_search (
            uint8 mot_idx )
```

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.

5.12.2.12 save_cycles_eeprom()

```
void save_cycles_eeprom ( )
```

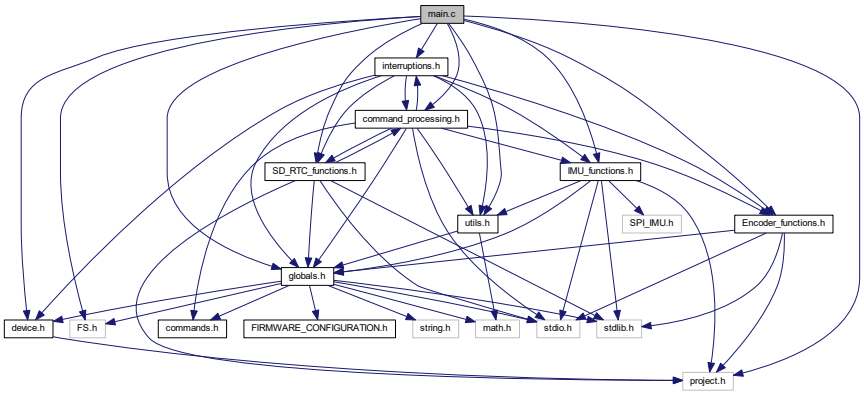
This function saves cycles counters variables into EEPROM memory.

5.13 main.c File Reference

Firmware main file.

```
#include "device.h"
#include "globals.h"
#include "interruptions.h"
#include "command_processing.h"
#include "IMU_functions.h"
#include "Encoder_functions.h"
#include "SD_RTC_functions.h"
#include "utils.h"
#include "project.h"
#include "FS.h"
```

Include dependency graph for main.c:



Functions

- int **main** ()

5.13.1 Detailed Description

Firmware main file.

Date

May 31, 2019

Author

Centro "E.Piaggio"

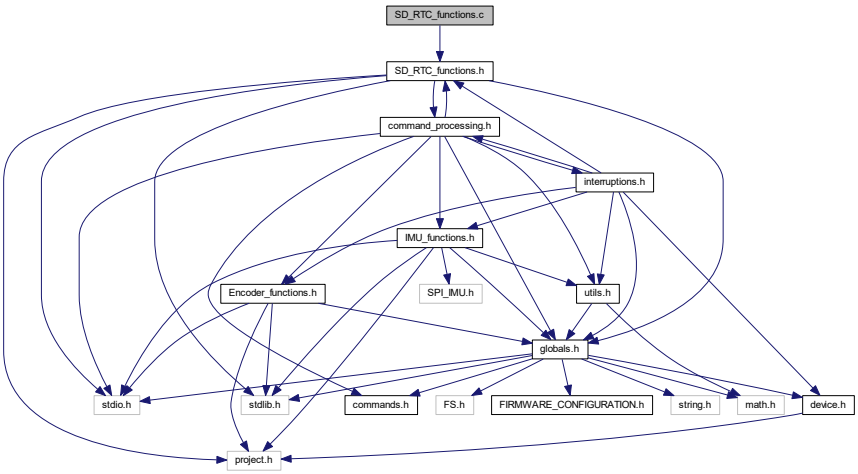
Copyright

5.14 SD_RTC_functions.c File Reference

Implementation of SD and RTC module functions.

```
#include "SD_RTC_functions.h"
```

Include dependency graph for SD_RTC_functions.c:



Functions

- void **DS1302_write** (uint8 address, uint8 data_wr)
- void **DS1302_writeByte** (uint8 data_wr)
- uint8 **DS1302_read** (uint8 address)
- uint8 **DS1302_readByte** ()
- void **shiftOut_RTC** (uint8 val)
- void **store_RTC_current_time** ()
- void **set_RTC_time** ()
- void **InitSD_FS** ()
- void **Write_SD_Param_file** ()
- void **Read_SD_Param** (char ∗info_param, int n_p)
- void **Read_SD_Data** (char ∗info_data, int n_d)

5.14.1 Detailed Description

Implementation of SD and RTC module functions.

Date

February 13, 2019

Author

Centro "E.Piaggio"

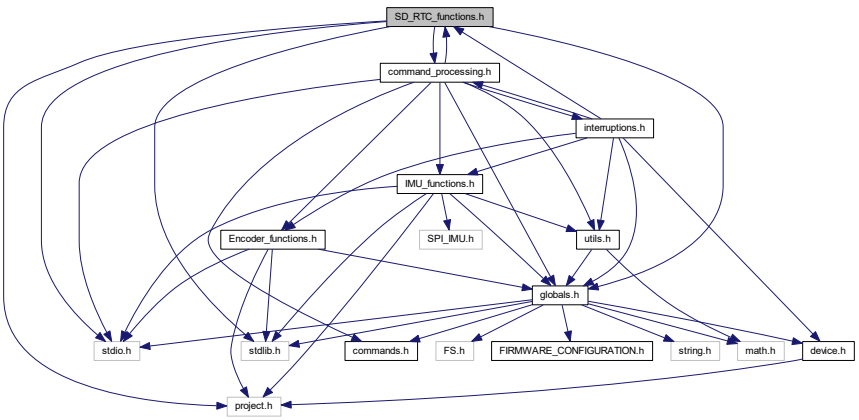
Copyright

5.15 SD_RTC_functions.h File Reference

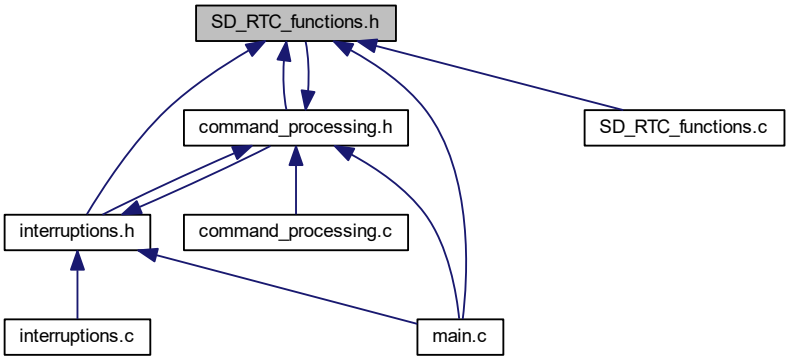
Definition of SD and RTC module functions.

```
#include <project.h>
#include "globals.h"
#include <stdlib.h>
#include <stdio.h>
#include "command_processing.h"
```

Include dependency graph for SD_RTC_functions.h:



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



Macros

- #define **DS1302_SECONDS_WR** 0x80
- #define **DS1302_SECONDS_RD** 0x81
- #define **DS1302_MINUTES_WR** 0x82
- #define **DS1302_MINUTES_RD** 0x83
- #define **DS1302_HOUR_WR** 0x84
- #define **DS1302_HOUR_RD** 0x85
- #define **DS1302_DATE_WR** 0x86
- #define **DS1302_DATE_RD** 0x87
- #define **DS1302_MONTH_WR** 0x88
- #define **DS1302_MONTH_RD** 0x89
- #define **DS1302_YEAR_WR** 0x8C
- #define **DS1302_YEAR_RD** 0x8D

Functions

- void **DS1302_write** (uint8 address, uint8 data_wr)
- void **DS1302_writeByte** (uint8 data_wr)
- uint8 **DS1302_read** (uint8 address)
- uint8 **DS1302_readByte** ()
- void **shiftOut_RTC** (uint8 val)
- void **store_RTC_current_time** ()
- void **set_RTC_time** ()
- void **InitSD_FS** ()
- void **Write_SD_Param_file** ()
- void **Read_SD_Param** (char ∗, int)
- void **Read_SD_Data** (char ∗, int)

5.15.1 Detailed Description

Definition of SD and RTC module functions.

Date

February 13, 2019

Author

Centro "E.Piaggio"

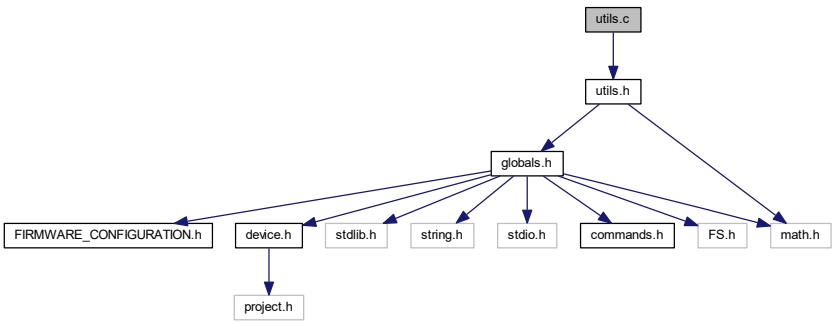
Copyright

5.16 utils.c File Reference

Definition of utility functions.

```
#include "utils.h"
```

Include dependency graph for utils.c:



Macros

- #define **M** 65536

  Number of encoder ticks per turn.

Functions

- int32 **curr_estim** (uint8 idx, int32 pos, int32 vel, int32 ref)
- int32 **filter** (int32 new_value, struct **st_filter** *f)
- uint32 **my_mod** (int32 val, int32 divisor)
- void **calibration** (void)
- int **calc_turns_fcn_SH** (const int32 pos1, const int32 pos2, const int N1, const int N2, const int I1)
- int **calc_turns_fcn** (const int32 pos1, const int32 pos2, const int N1, const int N2, const int I1)
- void **check_rest_position** (void)
- void **LED_control** (uint8 mode)
- void **battery_management** ()
- void **ADC_Set_N_Channels** ()
- void **enable_motor** (uint8 idx, uint8 val)
- void **reset_counters** ()
- float **invSqrt** (float x)
- void **v3_normalize** (float v3_in[3])
- void **v4_normalize** (float v4_in[4])

5.16.1 Detailed Description

Definition of utility functions.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.16.2 Function Documentation

5.16.2.1 calc_turns_fcn()

```
int calc_turns_fcn (
            const int32 pos1,
            const int32 pos2,
            const int N1,
            const int N2,
            const int I1 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. Generic. 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

5.16.2.2 calc_turns_fcn_SH()

```
int calc_turns_fcn_SH (
            const int32 pos1,
            const int32 pos2,
            const int N1,
            const int N2,
            const int I1 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. Only for SoftHand 3.0. 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

5.16.2.3 calibration()

```
void calibration ( )
```

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

5.16.2.4 check_rest_position()

```
void check_rest_position ( )
```

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

5.16.2.5 curr_estim()

```
int32 curr_estim (
            uint8 idx,
            int32 pos,
            int32 vel,
            int32 acc )
```

Function used to obtain current estimation through current lookup table.

Parameters

idx	Index of motor.
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.

5.16.2.6 filter()

```
int32 filter (
            int32 value,
            struct  st_filter * f )
```

Generic low pass filter. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.
f	Pointer to specific struct of type **st_filter** (p. 19).

Returns

Returns the filtered current value

5.16.2.7 LED_control()

```
void LED_control (
            uint8 mode )
```

This function switches between different LEDs condition depending on battery level of charge or needed maintenance.

5.16.2.8 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

5.16.2.9 reset_counters()

```
void reset_counters ( )
```

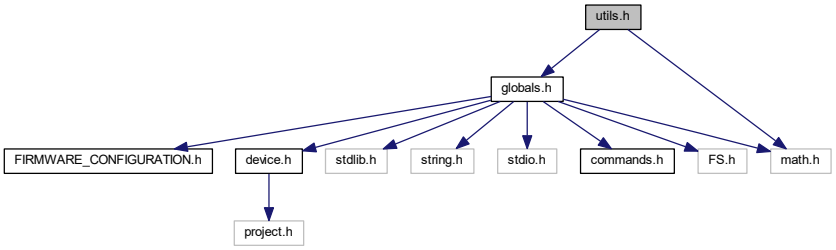
This function reset statistics counters

5.17 utils.h File Reference

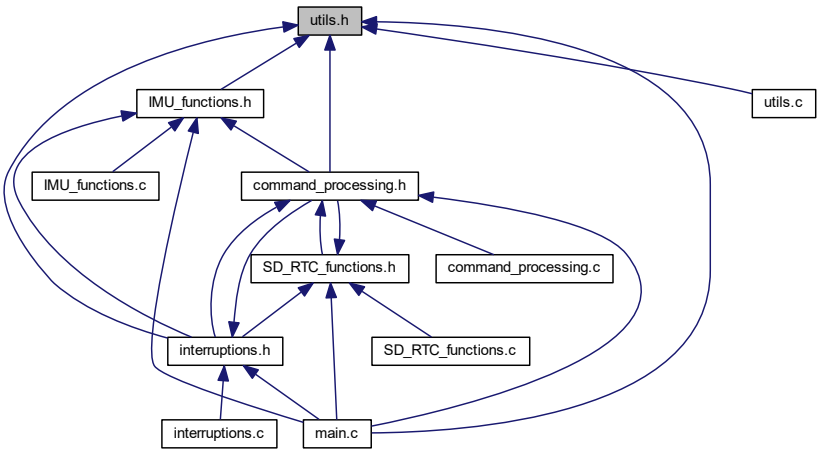
Utility functions declaration.

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

Include dependency graph for utils.h:



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



Macros

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

Functions

Filters

- int32 **filter** (int32 value, struct **st_filter** *f)

Estimating current and difference

- int32 **curr_estim** (uint8 idx, int32 pos, int32 vel, int32 acc)

Utility functions

- uint32 **my_mod** (int32 val, int32 divisor)
- int **calc_turns_fcn_SH** (const int32 pos1, const int32 pos2, const int N1, const int N2, const int I1)
- int **calc_turns_fcn** (const int32 pos1, const int32 pos2, const int N1, const int N2, const int I1)
- void **calibration** ()
- void **check_rest_position** ()
- void **LED_control** (uint8 mode)
- void **battery_management** ()
- void **ADC_Set_N_Channels** ()
- void **enable_motor** (uint8 idx, uint8 val)
- void **reset_counters** ()
- float **invSqrt** (float x)
- void **v3_normalize** (float v3_in[3])
- void **v4_normalize** (float v4_in[4])

5.17.1 Detailed Description

Utility functions declaration.

Date

October 01, 2017

Author

Centro "E.Piaggio"

Copyright

5.17.2 Macro Definition Documentation

5.17.2.1 REFSPEED

```
#define REFSPEED 20
```

Constant depending on PID values.

5.17.2.2 SIGN

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

Sign calculation function.

5.17.2.3 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.

5.17.2.4 ZMAX

```
#define ZMAX 5
```

Constant useful for current estimation procedure.

5.17.3 Function Documentation

5.17.3.1 calc_turns_fcn()

```
int calc_turns_fcn (
            const int32 pos1,
            const int32 pos2,
            const int N1,
            const int N2,
            const int I1 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. Generic. 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

5.17.3.2 calc_turns_fcn_SH()

```
int calc_turns_fcn_SH (
            const int32 pos1,
            const int32 pos2,
            const int N1,
            const int N2,
            const int I1 )
```

This function is used at startup to reconstruct the correct turn of the shaft connected to the motor. Only for SoftHand 3.0. 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

5.17.3.3 calibration()

```
void calibration ( )
```

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

5.17.3.4 check_rest_position()

```
void check_rest_position ( )
```

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

5.17.3.5 curr_estim()

```
int32 curr_estim (
            uint8 idx,
            int32 pos,
            int32 vel,
            int32 acc )
```

Function used to obtain current estimation through current lookup table.

Parameters

idx	Index of motor.
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.

5.17.3.6 filter()

```
int32 filter (
            int32 value,
            struct  st_filter * f )
```

Generic low pass filter. The weighted average between the old value and the new one is executed.

Parameters

value	New value of the filter.
f	Pointer to specific struct of type **st_filter** (p. 19).

Returns

Returns the filtered current value

5.17.3.7 LED_control()

```
void LED_control (
            uint8 mode )
```

This function switches between different LEDs condition depending on battery level of charge or needed maintenance.

5.17.3.8 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

5.17.3.9 reset_counters()

```
void reset_counters ( )
```

This function reset statistics counters

Index