STM32F4xx

1

Generated by Doxygen 1.8.7

Tue Oct 21 2014 15:13:21

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

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

Module Documentation

4.1 Template_Project

Functions

- void NMI_Handler (void)

  This function handles NMI exception.
- void HardFault_Handler (void)

  This function handles Hard Fault exception.
- void MemManage_Handler (void)

  This function handles Memory Manage exception.
- void BusFault_Handler (void)

  This function handles Bus Fault exception.
- void UsageFault_Handler (void)

  This function handles Usage Fault exception.
- __weak void SVC_Handler (void)

  This function handles SVCall exception.
- void DebugMon_Handler (void)

  This function handles Debug Monitor exception.
- __weak void PendSV_Handler (void)

  This function handles PendSVC exception.
- __weak void SysTick_Handler (void)

  This function handles SysTick Handler.
- __weak void EXTI9_5_IRQHandler (void)

  This function handles EXTI 3 interrupt request.
- __weak void EXTI15_10_IRQHandler (void)

  This function handles EXTI 15-10 interrupt request.

4.1.1 Detailed Description

4.1.2 Function Documentation

4.1.2.1 void BusFault_Handler (void)

This function handles Bus Fault exception.

Parameters

None	

Return values

None	

4.1.2.2 void DebugMon_Handler (void)

This function handles Debug Monitor exception.

Parameters

None	

Return values

None	

4.1.2.3 __weak void EXTI15_10_IRQHandler (void)

This function handles EXTI 15-10 interrupt request.

Parameters

None	

Return values

None	

4.1.2.4 __weak void EXTI9_5_IRQHandler (void)

This function handles EXTI 3 interrupt request.

Parameters

None	

Return values

None	

4.1.2.5 void HardFault_Handler (void)

This function handles Hard Fault exception.

Parameters

None	

Return values

None	

4.1.2.6 void MemManage_Handler (void)

This function handles Memory Manage exception.

Parameters

None	

Return values

None	

4.1.2.7 void NMI_Handler (void)

This function handles NMI exception.

Parameters

None	

Return values

None	

4.1.2.8 __weak void PendSV_Handler (void)

This function handles PendSVC exception.

Parameters

None	

Return values

None	

4.1.2.9 __weak void SVC_Handler (void)

This function handles SVCall exception.

Parameters

None	

Return values

None	

4.1.2.10 __weak void SysTick_Handler (void)

This function handles SysTick Handler.

Parameters

None	

Return values

None	

4.1.2.11 void UsageFault_Handler (void)

This function handles Usage Fault exception.

Parameters

None	

Return values

None	

Chapter 5

Data Structure Documentation

5.1 QueueTelegram Struct Reference

Telegram communication betwen task.

Data Fields

- QueueCommand Qcmd
- size_t size
- uint8_t data [100]

5.1.1 Detailed Description

Telegram communication betwen task.

Telegram communication between tasks in specific format

5.1.2 Field Documentation

5.1.2.1 uint8_t QueueTelegram::data[100]

data of telegram

5.1.2.2 QueueCommand QueueTelegram::Qcmd

QueueCommand what type of telegram we received

5.1.2.3 size_t QueueTelegram::size

size of data transmited

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

- main.h

5.2 Spd_Settings Struct Reference

Data Fields

- uint16_t **speed**
- uint16_t **param1**
- uint16_t **maxRPM**
- uint16_t **upRamp**
- uint16_t **downRamp**

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

- modbus_mk.h

Chapter 6

File Documentation

6.1 main.c File Reference

Main file with main functions for MOTOR TCP CLI application.

```
#include "main.h"
#include "spi.h"
#include "W5200.h"
#include "modbus_mk.h"
#include "tcpCLI.h"
```

Macros

- #define mainFLASH_TASK_PRIORITY (tskIDLE_PRIORITY + 1UL)
- #define **mainQUEUE_POLL_PRIORITY** (tskIDLE_PRIORITY + 2UL)
- #define **mainSEM_TEST_PRIORITY** (tskIDLE_PRIORITY + 1UL)
- #define **mainBLOCK_Q_PRIORITY** (tskIDLE_PRIORITY + 2UL)
- #define **mainCREATOR_TASK_PRIORITY** (tskIDLE_PRIORITY + 3UL)
- #define **mainFLOP_TASK_PRIORITY** (tskIDLE_PRIORITY)
- #define **mainCHECK_TIMER_PERIOD_MS** (3000UL / portTICK_RATE_MS)
- #define **mainERROR_CHECK_TIMER_PERIOD_MS** (200UL / portTICK_RATE_MS)
- #define **mainCREATE_SIMPLE_LED_FLASHER_DEMO_ONLY** 0

Functions

- void **vRegTest1Task** (void *pvParameters)
- void **vRegTest2Task** (void *pvParameters)
- void **vRegTestClearFlopRegistersToParameterValue** (unsigned long ulValue)
- unsigned long **ulRegTestCheckFlopRegistersContainParameterValue** (unsigned long ulValue)
- int **main** (void)
- void **vApplicationTickHook** (void)
- void **TIM3_IRQHandler** (void)
- void **TIM2_IRQHandler** (void)
- void **vApplicationMallocFailedHook** (void)
- void **vApplicationIdleHook** (void)
- void **vApplicationStackOverflowHook** (xTaskHandle pxTask, signed char *pcTaskName)
- void assert_failed (uint8_t *file, uint32_t line)

--/

Variables

- volatile unsigned long **ulRegTest1LoopCounter** = 0UL
- volatile unsigned long **ulRegTest2LoopCounter** = 0UL
- volatile unsigned long **ulFPUInterruptNesting** = 0UL
- volatile unsigned long **ulMaxFPUInterruptNesting** = 0UL
- volatile unsigned long **ulButtonPressCounts** = 0UL

6.1.1 Detailed Description

Main file with main functions for MOTOR TCP CLI application.

6.1.2 Macro Definition Documentation

6.1.2.1 #define mainFLASH_TASK_PRIORITY (tskIDLE_PRIORITY + 1UL)

- Priorities for the demo application tasks. ∗/

6.2 main.h File Reference

A main header file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "semphr.h"
#include "stm32f4xx.h"
#include "stm32f4xx_conf.h"
#include <stm32f4xx_usart.h>
#include <stm32f4xx_spi.h>
#include <stm32f4xx_dma.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
```

Data Structures

- struct QueueTelegram

  Telegram communication betwen task.

Enumerations

- enum QueueCommand { DATA, IDLE, DELETE }

  Queue telegram command.

Variables

- xSemaphoreHandle xSemaphoreDMASPI

  Semaphore handle for DMA SPI pheriphal.
- xSemaphoreHandle xSmphrUSART

  Sempahore handle for USART port.
- xTaskHandle setSpeedHandle

  Task handles for setspeed.
- motorHBHandle

  Task handles for motor heart bit tasks.
- xQueueHandle **QSpd_handle**
- QStatus_handle

  Queue handles.
- int socket_0

  Socket 0 descriptor.

6.2.1 Detailed Description

A main header file.

6.2.2 Enumeration Type Documentation

6.2.2.1 enum QueueCommand

Queue telegram command.

Queue telegram command for tasks commmunication

Enumerator

DATA enum DATA if sending data.

IDLE enum IDLE if task has to go to idle mode.

DELETE enum DELETE if task has to delete itself.

6.3 W5200.c File Reference

File for W5200.c wiznet functions Wiznet control, creating and closing sockets, sending, receiving data via sockets, etc...

.

```
#include "W5200.h"
```

Functions

- void init_W5200 (void)

  Initialize Wiznet module.
- void W5200_set_hardware_address (const macaddress_t address)

  set W5200 mac address
- void W5200_get_hardware_address (macaddress_t address)

  get W5200 mac address

- void W5200_configure_network (const ipv4address_t address, const ipv4address_t subnet, const ipv4address_t gateway)

  Configures network of wiznet.
- void W5200_get_ipaddress (ipv4address_t address)

  sets ip address to Wiznet
- uint8_t socket (uint8_t mode, uint16_t port, uint8_t ip_proto)

  Create socket return socket handle.
- int closesocket (int sck_fd)

  Close socket handle.
- int connect (uint8_t sck_fd, uint8_t *to_ip, uint16_t to_port)

  Connecto to host by ip via port.
- int send (uint8_t sck_fd, uint8_t *buf, uint16_t len, uint16_t flag)

  Send buf data via socket.
- int recv (uint8_t sck_fd, uint8_t *buf, uint16_t len, uint16_t flag)

  Receive data to buffer via socket.
- int listen (int sck_fd)

  It listens on socket file descriptor previously created by function socket.
- void locate_interrupt ()

  Function used by interrupt handler used to identify wiznet interrupt.
- void set_macTask (void *pvParameters)

  Test task TCP simple server.

6.3.1 Detailed Description

File for W5200.c wiznet functions Wiznet control, creating and closing sockets, sending, receiving data via sockets, etc...
.

6.3.2 Function Documentation

6.3.2.1 int closesocket (int *sck_fd*)

Close socket handle.

Parameters

sck_fd	- socket file descriptor to close

6.3.2.2 int connect (uint8_t *sck_fd*, uint8_t * *to_ip*, uint16_t *to_port*)

Connecto to host by ip via port.

```
\param sck_fd - socket file descriptor
```

Parameters

*to_ip	- destination ip number in hex format
to_port	- destination port number

6.3.2.3 void init_W5200 (void)

Initialize Wiznet module.

Initialize Wiznet module - setup MAC address, ip address, gateway, subnet, etc..

- Default settings:
  - mac address : dd:aa:bb:cc:11:22
  - ip address : 192.168.0.8
  - subnet : 255.255.255.0
  - gateway : 192.168.0.254
  - Ping : enable
  - timeout : 200 mili sec.
  - retry count : 3
  - TX memory size : 2kB
  - RX memory size : 2kB
  - IT masking : 0xff

Create and listen on socket '0', port 80.

6.3.2.4 int listen (int *sck_fd*)

It listens on socket file descriptor previously created by function socket.

Parameters

sck_fd	- socket file descriptor

6.3.2.5 void locate_interrupt ( )

Function used by interrupt handler used to identify wiznet interrupt.

Function used by interrupt service routine.
Reads wiznet interrupt registers and identify interupt plus on which socket interrupt occured.
It depends on interrupt what follows

6.3.2.6 int recv (uint8_t *sck_fd,* uint8_t * *buf,* uint16_t *len,* uint16_t *flag*)

Receive data to buffer via socket.

Parameters

sck_fd	- socket file descriptor
**buf*	- pointer to memory buffer data
len	- length of buffered data
flag	- socket flag NOT USED !!!!!

6.3.2.7 int send (uint8_t *sck_fd,* uint8_t * *buf,* uint16_t *len,* uint16_t *flag*)

Send buf data via socket.

Parameters

sck_fd	- socket file descriptor
∗buf	- pointer to memory buffer data
len	- length of buffered data
flag	- socket flag NOT USED !!!!!

6.3.2.8 void set_macTask (void ∗ *pvParameters*)

Test task TCP simple server.

Parameters

∗pvParameters	- possible parameters for task

Task opens socket on port 80, starts listening on port and susped suspend itself. If interrupt occures it process CLI command

6.3.2.9 uint8_t socket (uint8_t *mode,* uint16_t *port,* uint8_t *ip_proto*)

Create socket return socket handle.

Parameters

mode	- socket mode (TCP, UDP, ...)
port	- socket port
ip_proto	- ip protocol number

6.3.2.10 void W5200_configure_network (const ipv4address_t *address,* const ipv4address_t *subnet,* const ipv4address_t *gateway*)

Configures network of wiznet.

Parameters

address	- ip addres in hex
subnet	- subnet in dec
gateway	- gateway address in hex

It configures ip, subnet and gateway

6.3.2.11 W5200_get_hardware_address (macaddress_t *address*)

get W5200 mac address

Parameters

address	- mac address

Function reads wiznet register mac address and copies it to address

6.3.2.12 void W5200_get_ipaddress (ipv4address_t *address*)

sets ip address to Wiznet

Parameters

address	- ip address in hex

6.3.2.13 W5200_set_hardware_address (const macaddress_t *address*)

set W5200 mac address

```
\param address - mac address
```

Function set wiznet register to address via SPI DMA

6.4 W5200.h File Reference

Header file for W5200.c wiznet functions.

```
#include "main.h"
#include "spi.h"
```

Macros

- #define **W5200_MAX_SOCKETS** 8
- #define **W5200_MR** 0x0000
- #define **W5200_GAR** 0x0001
- #define **W5200_SUBR** 0x0005
- #define **W5200_SHAR** 0x0009
- #define **W5200_SIPR** 0x000F
- #define **W5200_IR** 0x0015
- #define **W5200_IMR** 0x0016
- #define **W5200_RTR** 0x0017
- #define **W5200_RCR** 0x0019
- #define **W5200_PATR** 0x001C
- #define **W5200_PPPALGO** 0x001E
- #define **W5200_PTIMER** 0x0028
- #define **W5200_PMAGIC** 0x0029
- #define **W5200_INTLEVEL** 0x0030
- #define **W5200_IR2** 0x0034
- #define **W5200_PSTATUS** 0x0035
- #define **W5200_IMR2** 0x0036
- #define **W5200_SOCKET_REG**(s) (0x4000 + (s << 8))
- #define **W5200_SOCKET_TX_BASE**(s) (0x8000 + (8*s << 8))
- #define **W5200_SOCKET_RX_BASE**(s) (0xc000 + (8*s << 8))
- #define **W5200_Sn_MR**(s) (W5200_SOCKET_REG(s) + 0x00)
- #define **W5200_Sn_CR**(s) (W5200_SOCKET_REG(s) + 0x01)
- #define **W5200_Sn_IR**(s) (W5200_SOCKET_REG(s) + 0x02)
- #define **W5200_Sn_SR**(s) (W5200_SOCKET_REG(s) + 0x03)
- #define **W5200_Sn_PORT**(s) (W5200_SOCKET_REG(s) + 0x04)
- #define **W5200_Sn_DHAR**(s) (W5200_SOCKET_REG(s) + 0x06)
- #define **W5200_Sn_DIPR**(s) (W5200_SOCKET_REG(s) + 0x0C)
- #define **W5200_Sn_DPORT**(s) (W5200_SOCKET_REG(s) + 0x10)
- #define **W5200_Sn_MSSR**(s) (W5200_SOCKET_REG(s) + 0x12)
- #define **W5200_Sn_PROTO**(s) (W5200_SOCKET_REG(s) + 0x14)

- #define **W5200_Sn_IP_TOS**(s) (W5200_SOCKET_REG(s) + 0x15)
- #define **W5200_Sn_IP_TTL**(s) (W5200_SOCKET_REG(s) + 0x16)
- #define **W5200_Sn_RXMEM_SIZE**(s) (W5200_SOCKET_REG(s) + 0x1E)
- #define **W5200_Sn_TXMEM_SIZE**(s) (W5200_SOCKET_REG(s) + 0x1F)
- #define **W5200_Sn_TX_FSR**(s) (W5200_SOCKET_REG(s) + 0x20)
- #define **W5200_Sn_TX_RD**(s) (W5200_SOCKET_REG(s) + 0x22)
- #define **W5200_Sn_TX_WR**(s) (W5200_SOCKET_REG(s) + 0x24)
- #define **W5200_Sn_RX_RSR**(s) (W5200_SOCKET_REG(s) + 0x26)
- #define **W5200_Sn_RX_RD**(s) (W5200_SOCKET_REG(s) + 0x28)
- #define **W5200_Sn_RX_WR**(s) (W5200_SOCKET_REG(s) + 0x2A)
- #define **W5200_Sn_IMR**(s) (W5200_SOCKET_REG(s) + 0x2C)
- #define **W5200_Sn_FRAG**(s) (W5200_SOCKET_REG(s) + 0x2D)
- #define **W5200_MR_RST** 0x80
- #define **W5200_MR_PB** 0x10
- #define **W5200_MR_PPPOE_ENABLE** 0x08
- #define **W5200_IR_CONFLICT** 0x80
- #define **W5200_IR_PPPOE_CLOSE** 0x20
- #define **W5200_IMR_CONFLICT** 0x80
- #define **W5200_IMR_PPPOE_CLOSE** 0x20
- #define **W5200_PSTATUS_LINK** 0x20
- #define **W5200_PSTATUS_POWERDOWN** 0x08
- #define **W5200_Sn_MR_MULTI** 0x80
- #define **W5200_Sn_MR_MF** 0x40
- #define **W5200_Sn_MR_ND** 0x20
- #define **W5200_Sn_MR_MC** 0x20
- #define **W5200_Sn_MR_CLOSED** 0x00
- #define **W5200_Sn_MR_TCP** 0x01
- #define **W5200_Sn_MR_UDP** 0x02
- #define **W5200_Sn_MR_IPRAW** 0x03
- #define **W5200_Sn_MR_MACRAW** 0x04
- #define **W5200_Sn_CR_IDLE** 0x00
- #define **W5200_Sn_CR_OPEN** 0x01
- #define **W5200_Sn_CR_LISTEN** 0x02
- #define **W5200_Sn_CR_CONNECT** 0x04
- #define **W5200_Sn_CR_DISCON** 0x08
- #define **W5200_Sn_CR_CLOSE** 0x10
- #define **W5200_Sn_CR_SEND** 0x20
- #define **W5200_Sn_CR_SEND_MAC** 0x21
- #define **W5200_Sn_CR_SEND_KEEP** 0x22
- #define **W5200_Sn_CR_RECV** 0x40
- #define **W5200_Sn_SR_SOCK_CLOSED** 0x0000
- #define **W5200_Sn_SR_SOCK_ARP** 0x0001
- #define **W5200_Sn_SR_SOCK_INIT** 0x0013
- #define **W5200_Sn_SR_SOCK_LISTEN** 0x0014
- #define **W5200_Sn_SR_SOCK_SYNSENT** 0x0015
- #define **W5200_Sn_SR_SOCK_SYNRECV** 0x0016
- #define **W5200_Sn_SR_SOCK_ESTABLISHED** 0x0017
- #define **W5200_Sn_SR_SOCK_FIN_WAIT** 0x0018
- #define **W5200_Sn_SR_SOCK_CLOSING** 0x001A
- #define **W5200_Sn_SR_SOCK_TIME_WAIT** 0x001B
- #define **W5200_Sn_SR_SOCK_CLOSE_WAIT** 0x001C
- #define **W5200_Sn_SR_SOCK_LAST_ACK** 0x001D
- #define **W5200_Sn_SR_SOCK_UDP** 0x0022
- #define **W5200_Sn_SR_SOCK_IPRAW** 0x0032
- #define **W5200_Sn_SR_SOCK_MACRAW** 0x0042

- #define **W5200_Sn_SR_SOCK_PPPOE** 0x005F
- #define **W5200_RAMSIZE** 16384
- #define **W5200_TX_RAM_ADDR** 0x8000
- #define **W5200_RX_RAM_ADDR** 0xC000
- #define **W5200_RAMSIZE_CONFIG_WHOLE** 0x0F
- #define **W5200_RAMSIZE_CONFIG_HALF** 0x08
- #define **W5200_RAMSIZE_CONFIG_QUARTER** 0x04
- #define **W5200_RAMSIZE_CONFIG_EIGHTH** 0x02

Typedefs

- typedef uint8_t **macaddress_t** [6]
- typedef uint8_t **ipv4address_t** [4]

Functions

- void init_W5200 (void)

  Initialize Wiznet module.
- void W5200_set_hardware_address (const macaddress_t address)

  set W5200 mac address
- void W5200_get_hardware_address (macaddress_t address)

  get W5200 mac address
- void W5200_configure_network (const ipv4address_t address, const ipv4address_t subnet, const ipv4address_t gateway)

  Configures network of wiznet.
- void W5200_get_ipaddress (ipv4address_t address)

  sets ip address to Wiznet
- uint8_t socket (uint8_t mode, uint16_t port, uint8_t ip_proto)

  Create socket return socket handle.
- int connect (uint8_t sck_fd, uint8_t *to_ip, uint16_t to_port)

  Connecto to host by ip via port.
- int send (uint8_t sck_fd, uint8_t *buf, uint16_t len, uint16_t flag)

  Send buf data via socket.
- int closesocket (int sck_fd)

  Close socket handle.
- int recv (uint8_t sck_fd, uint8_t *buf, uint16_t len, uint16_t flag)

  Receive data to buffer via socket.
- void set_macTask (void *pvParameters)

  Test task TCP simple server.
- int listen (int sck_fd)

  It listens on socket file descriptor previously created by function socket.

Variables

- xTaskHandle **set_macTaskHandle**

6.4.1 Detailed Description

Header file for W5200.c wiznet functions.

6.4.2 Function Documentation

6.4.2.1 int closesocket (int *sck_fd*)

Close socket handle.

Parameters

sck_fd	- socket file descriptor to close

6.4.2.2 int connect (uint8_t *sck_fd,* uint8_t * *to_ip,* uint16_t *to_port*)

Connecto to host by ip via port.

```
\param sck_fd - socket file descriptor
```

Parameters

**to_ip*	- destination ip number in hex format
to_port	- destination port number

6.4.2.3 void init_W5200 (void)

Initialize Wiznet module.

Initialize Wiznet module - setup MAC address, ip address, gateway, subnet, etc..

- Default settings:
  - mac address : dd:aa:bb:cc:11:22
  - ip address : 192.168.0.8
  - subnet : 255.255.255.0
  - gateway : 192.168.0.254
  - Ping : enable
  - timeout : 200 mili sec.
  - retry count : 3
  - TX memory size : 2kB
  - RX memory size : 2kB
  - IT masking : 0xff

Create and listen on socket '0', port 80.

6.4.2.4 int listen (int *sck_fd*)

It listens on socket file descriptor previously created by function socket.

Parameters

sck_fd	- socket file descriptor

6.4.2.5 int recv (uint8_t *sck_fd,* uint8_t * *buf,* uint16_t *len,* uint16_t *flag*)

Receive data to buffer via socket.

Parameters

sck_fd	- socket file descriptor
∗buf	- pointer to memory buffer data
len	- length of buffered data
flag	- socket flag NOT USED !!!!!

6.4.2.6 int send (uint8_t *sck_fd,* uint8_t ∗ *buf,* uint16_t *len,* uint16_t *flag*)

Send buf data via socket.

Parameters

sck_fd	- socket file descriptor
∗buf	- pointer to memory buffer data
len	- length of buffered data
flag	- socket flag NOT USED !!!!!

6.4.2.7 void set_macTask (void ∗ *pvParameters*)

Test task TCP simple server.

Parameters

∗pvParameters	- possible parameters for task

Task opens socket on port 80, starts listening on port and susped suspend itself. If interrupt occures it process CLI command

6.4.2.8 uint8_t socket (uint8_t *mode,* uint16_t *port,* uint8_t *ip_proto*)

Create socket return socket handle.

Parameters

mode	- socket mode (TCP, UDP, ...)
port	- socket port
ip_proto	- ip protocol number

6.4.2.9 void W5200_configure_network (const ipv4address_t *address,* const ipv4address_t *subnet,* const ipv4address_t *gateway*)

Configures network of wiznet.

Parameters

address	- ip addres in hex
subnet	- subnet in dec
gateway	- gateway address in hex

It configures ip, subnet and gateway

6.4.2.10 void W5200_get_hardware_address (macaddress_t *address*)

get W5200 mac address

Parameters

address	- mac address

Function reads wiznet register mac address and copies it to address

6.4.2.11 void W5200_get_ipaddress (ipv4address_t *address*)

sets ip address to Wiznet

Parameters

address	- ip address in hex

6.4.2.12 void W5200_set_hardware_address (const macaddress_t *address*)

set W5200 mac address

```
\param address - mac address
```

Function set wiznet register to address via SPI DMA

Index