Socket APIs

Todo List

globalScope> Member _WIZCHIP_

You should select one, **5100**, **5200**, **5300**, **5500** or etc.

ex> `#define _WIZCHIP_ 5500`

globalScope> Member _WIZCHIP_IO_BASE_

Should re-define it to fit your system when BUS IF Mode (**_WIZCHIP_IO_MODE_BUS_**, **_WIZCHIP_IO_MODE_BUS_DIR_**, **_WIZCHIP_IO_MODE_BUS_INDIR_**).

ex> `#define _WIZCHIP_IO_BASE_ 0x00008000`

globalScope> Member _WIZCHIP_IO_MODE_

Should select interface mode as chip.

- **_WIZCHIP_IO_MODE_SPI_**
  -**_WIZCHIP_IO_MODE_SPI_VDM_** : Valid only in **_WIZCHIP_** == 5500
  -**_WIZCHIP_IO_MODE_SPI_FDM_** : Valid only in **_WIZCHIP_** == 5500
- **_WIZCHIP_IO_MODE_BUS_**
  - **_WIZCHIP_IO_MODE_BUS_DIR_**
  - **_WIZCHIP_IO_MODE_BUS_INDIR_**
- Others will be defined in future.

  ex> `#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_SPI_VDM_`

globalScope> Member reg_wizchip_bus_cbfunc (iodata_t(*bus_rb)(uint32_t addr), void(*bus_wb)(uint32_t addr, iodata_t wb))

Describe wizchip_bus_readbyte and wizchip_bus_writebyte function

or register your functions.

> **Note**
>
> If you do not describe or register, null function is called.

globalScope> Member reg_wizchip_cris_cbfunc (void(*cris_en)(void), void(*cris_ex)(void))

Describe **WIZCHIP_CRITICAL_ENTER** and **WIZCHIP_CRITICAL_EXIT** marco or register your functions.

> **Note**
>
> If you do not describe or register, default functions(**wizchip_cris_enter** & **wizchip_cris_exit**) is called.

globalScope> Member reg_wizchip_cs_cbfunc (void(*cs_sel)(void), void(*cs_desel)(void))

Describe **wizchip_cs_select** and **wizchip_cs_deselect** function or register your functions.

> **Note**
>
> If you do not describe or register, null function is called.

globalScope> Member reg_wizchip_spi_cbfunc (uint8_t(*spi_rb)(void), void(*spi_wb)(uint8_t wb))

Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte** function or register your functions.

> **Note**
>
> If you do not describe or register, null function is called.

globalScope> Member reg_wizchip_spiburst_cbfunc (void(*spi_rb)(uint8_t *pBuf, uint16_t len), void(*spi_wb)(uint8_t *pBuf, uint16_t len))

Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte** function or register your functions.

> **Note**
>
> If you do not describe or register, null function is called.

Socket APIs

Modules

Here is a list of all modules:

[detail level 1 2 3]

1. WIZnet socket APIs	WIZnet socket APIs are based on Berkeley socket APIs, thus it has much similar name and interface. But there is a little bit of difference
2. WIZnet Extra Functions	These functions is optional function. It could be replaced at WIZCHIP I/O function because they were made by WIZCHIP I/O functions
DATA TYPE	
▾ **W5100**	WHIZCHIP register defines and I/O functions of **W5100**
▾ **WIZCHIP I/O functions**	This supports the basic I/O functions for **WIZCHIP register**
Basic I/O function	These are basic input/output functions to read values from register or write values to register
Common register access functions	These are functions to access **common registers**

Socket register access functions	These are functions to access **socket registers**
▾ **WIZCHIP register**	WIZCHIP register defines register group of **W5100**
Common register	Common register group It set the basic for the networking It set the configuration such as interrupt, network information, ICMP, etc
Socket register	Socket register group Socket register configures and control SOCKETn which is necessary to data communication
▾ **W5200**	WHIZCHIP register defines and I/O functions of **W5200**
▾ **WIZCHIP I/O functions**	This supports the basic I/O functions for **WIZCHIP register**
Basic I/O function	These are basic input/output functions to read values from register or write values to register
Common register access functions	These are functions to access **common registers**
Socket register access functions	These are functions to access **socket registers**
▾ **WIZCHIP register**	WIZCHIP register defines register group of **W5200**
Common register	Common register group It set the basic for the networking

	It set the configuration such as interrupt, network information, ICMP, etc
Socket register	Socket register group Socket register configures and control SOCKETn which is necessary to data communication
▾ **W5300**	WHIZCHIP register defines and I/O functions of **W5300**
▾ **WIZCHIP I/O functions**	This supports the basic I/O functions for **WIZCHIP register**
Basic I/O function	These are basic input/output functions to read values from register or write values to register
Common register access functions	These are functions to access **common registers**
Socket register access functions	These are functions to access **socket registers**
▾ **WIZCHIP register**	WHIZCHIP register defines register group of **W5300**
Common register	Common register group It set the basic for the networking It set the configuration such as interrupt, network information, ICMP, etc
Socket register	Socket register group. Socket register

	configures and control SOCKETn which is necessary to data communication
▾ **W5500**	WHIZCHIP register defines and I/O functions of **W5500**
▾ **WIZCHIP I/O functions**	This supports the basic I/O functions for **WIZCHIP register**
Basic I/O function	These are basic input/output functions to read values from register or write values to register
Common register access functions	These are functions to access **common registers**
Socket register access functions	These are functions to access **socket registers**
▾ **WIZCHIP register**	WHIZCHIP register defines register group of **W5500**
Common register	Common register group It set the basic for the networking It set the configuration such as interrupt, network information, ICMP, etc
Socket register	Socket register group. Socket register configures and control SOCKETn which is necessary to data communication

Socket APIs

1. WIZnet socket APIs

WIZnet socket APIs are based on Berkeley socket APIs, thus it has much similar name and interface. But there is a little bit of difference. More...

Functions

int8_t **socket** (uint8_t sn, uint8_t protocol, uint16_t port, uint8_t flag)
Open a socket. More...

int8_t **close** (uint8_t sn)
Close a socket. More...

int8_t **listen** (uint8_t sn)
Listen to a connection request from a client. More...

int8_t **connect** (uint8_t sn, uint8_t *addr, uint16_t port)
Try to connect a server. More...

int8_t **disconnect** (uint8_t sn)
Try to disconnect a connection socket. More...

int32_t **send** (uint8_t sn, uint8_t *buf, uint16_t len)
Send data to the connected peer in TCP socket. More...

int32_t **recv** (uint8_t sn, uint8_t *buf, uint16_t len)
Receive data from the connected peer. More...

int32_t **sendto** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t port)
Sends datagram to the peer with destination IP address and port number passed as parameter. More...

int32_t **recvfrom** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t *port)
Receive datagram of UDP or MACRAW. More...

int8_t **ctlsocket** (uint8_t sn, **ctlsock_type** cstype, void *arg)
Control socket. More...

int8_t **setsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
set socket options More...

int8_t **getsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
get socket options More...

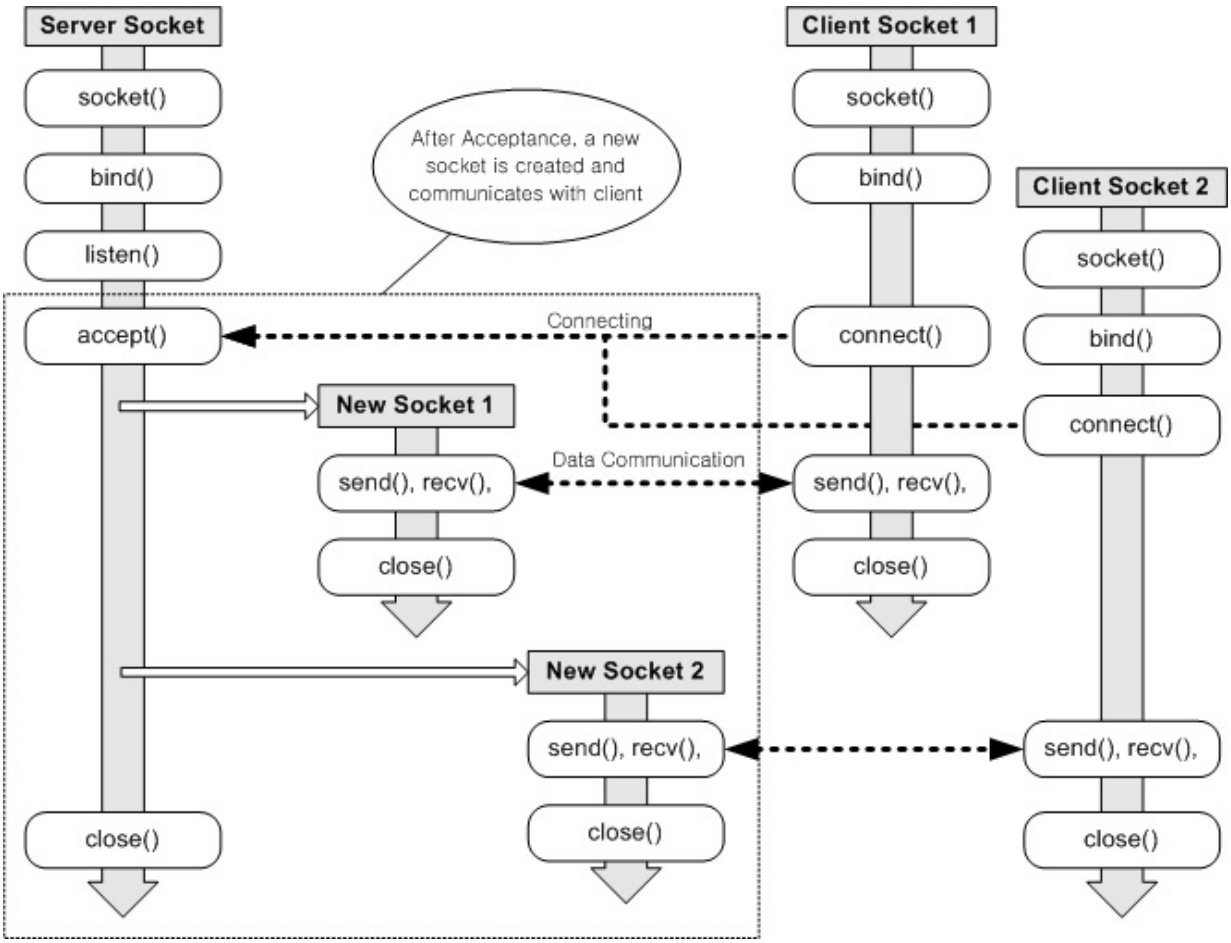
Detailed Description

WIZnet socket APIs are based on Berkeley socket APIs, thus it has much similar name and interface. But there is a little bit of difference.

Comparison between WIZnet and Berkeley SOCKET APIs

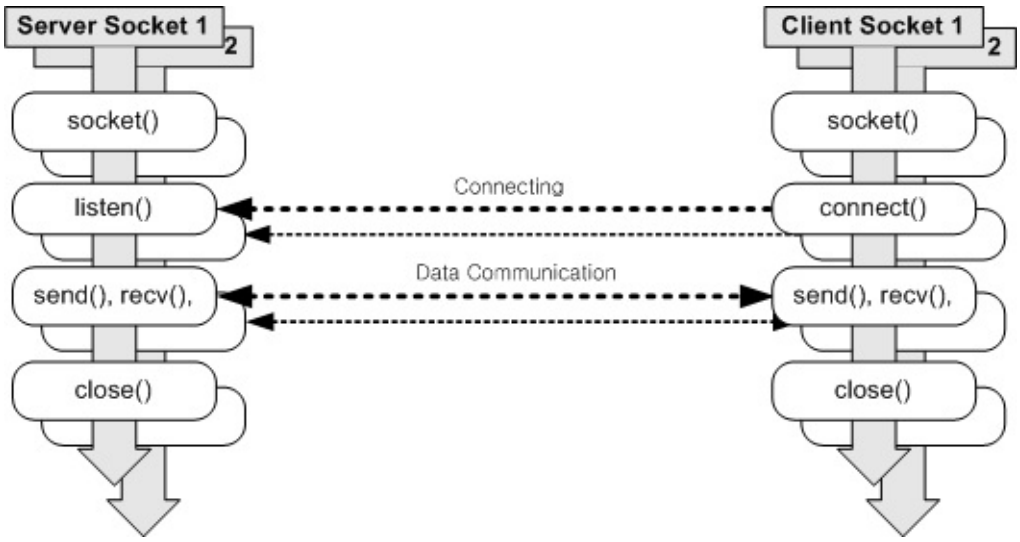
API	WIZnet	Berkeley
socket()	O	O
bind()	X	O
listen()	O	O
connect()	O	O
accept()	X	O
recv()	O	O
send()	O	O
recvfrom()	O	O
sendto()	O	O
closesocket()	O **close()** & **disconnect()**	O

There are **bind()** and **accept()** functions in **Berkeley** SOCKET API but, not in **WIZnet** SOCKET API. Because **socket()** of WIZnet is not only creating a SOCKET but also binding a local port number, and **listen()** of WIZnet is not only listening to connection request from client but also accepting the connection request.
When you program "TCP SERVER" with Berkeley SOCKET API, you can use only one listen port. When the listen SOCKET accepts a connection request from a client, it keeps listening. After accepting the connection request, a new SOCKET is created and the new SOCKET is used in communication with the client.
Following figure shows network flow diagram by Berkeley SOCKET API.



<Berkeley SOCKET API>

But, When you program "TCP SERVER" with WIZnet SOCKET API, you can use as many as 8 listen SOCKET with same port number. Because there's no accept() in WIZnet SOCKET APIs, when the listen SOCKET accepts a connection request from a client, it is changed in order to communicate with the client. And the changed SOCKET is not listening any more and is dedicated for communicating with the client. If there're many listen SOCKET with same listen port number and a client requests a connection, the SOCKET which has the smallest SOCKET number accepts the request and is changed as communication SOCKET.
Following figure shows network flow diagram by WIZnet SOCKET API.



<WIZnet SOCKET API>

Function Documentation

```
int8_t socket ( uint8_t   sn,
                uint8_t   protocol,
                uint16_t  port,
                uint8_t   flag
              )
```

Open a socket.

Initializes the socket with 'sn' passed as parameter and open.

Parameters

sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
protocol	Protocol type to operate such as TCP, UDP and MACRAW.
port	Port number to be bined.
flag	Socket flags as **SF_ETHER_OWN**, **SF_IGMP_VER2**, **SF_TCP_NODELAY**, **SF_MULTI_ENABLE**, **SF_IO_NONBLOCK** and so on. Valid flags only in W5500 : **SF_BROAD_BLOCK**, **SF_MULTI_BLOCK**, **SF_IPv6_BLOCK**, and **SF_UNI_BLOCK**.

See also

Sn_MR

Returns

Success : The socket number **'sn'** passed as parameter
Fail :
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_SOCKMODE - Not support socket mode as TCP, UDP, and so on.
SOCKERR_SOCKFLAG - Invaild socket flag.

Definition at line **105** of file **socket.c**.

References **CHECK_SOCKNUM**, **close()**, **getSIPR**, **getSn_CR**, **getSn_SR**, **PACK_COMPLETED**, **setSn_CR**, **setSn_MR**, **setSn_PORT**, **SF_IGMP_VER2**, **SF_IO_NONBLOCK**, **SF_MULTI_ENABLE**, **SF_TCP_NODELAY**, **SF_UNI_BLOCK**, **Sn_CR_OPEN**, **Sn_MR_IPRAW**, **Sn_MR_MACRAW**, **Sn_MR_PPPoE**, **Sn_MR_TCP**, **Sn_MR_UDP**, **SOCK_ANY_PORT_NUM**, **SOCK_CLOSED**, **sock_pack_info**, **SOCKERR_SOCKFLAG**, **SOCKERR_SOCKINIT**, and **SOCKERR_SOCKMODE**.

int8_t close (uint8_t sn)

Close a socket.

It closes the socket with 'sn' passed as parameter.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

Success : **SOCK_OK**
Fail : **SOCKERR_SOCKNUM** - Invalid socket number

Definition at line **197** of file **socket.c**.

References **CHECK_SOCKNUM**, **getSn_CR**, **getSn_MR**, **getSn_SR**, **getSn_TX_FSR()**, **getSn_TxMAX**, **sendto()**, **setSn_CR**, **setSn_IR**, **setSn_MR**, **setSn_PORTR**, **Sn_CR_CLOSE**, **Sn_CR_OPEN**, **Sn_MR_TCP**, **Sn_MR_UDP**, **SOCK_CLOSED**, **SOCK_OK**, **sock_pack_info**, and **SOCK_UDP**.

Referenced by **disconnect()**, **listen()**, **recv()**, **recvfrom()**, **send()**, and **socket()**.

int8_t listen (uint8_t sn)

Listen to a connection request from a client.

It is listening to a connection request from a client. If connection request is accepted successfully, the connection is established. Socket sn is used in passive(server) mode.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

Success : **SOCK_OK**
Fail :
SOCKERR_SOCKINIT - Socket is not initialized
SOCKERR_SOCKCLOSED - Socket closed unexpectedly.

Definition at line **240** of file **socket.c**.

References **CHECK_SOCKINIT**, **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **close()**, **getSn_CR**, **getSn_SR**, **setSn_CR**, **Sn_CR_LISTEN**, **Sn_MR_TCP**, **SOCK_CLOSED**, **SOCK_LISTEN**, **SOCK_OK**, and **SOCKERR_SOCKCLOSED**.

```
int8_t connect ( uint8_t    sn,
                 uint8_t *  addr,
                 uint16_t   port
               )
```

Try to connect a server.

It requests connection to the server with destination IP address and port number passed as parameter.

Note

It is valid only in TCP client mode. In block io mode, it does not return until connection is completed. In Non-block io mode, it return **SOCK_BUSY** immediately.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

addr Pointer variable of destination IP address. It should be allocated 4 bytes.

port Destination port number.

Returns

Success : **SOCK_OK**
Fail :
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_SOCKMODE - Invalid socket mode
SOCKERR_SOCKINIT - Socket is not initialized
SOCKERR_IPINVALID - Wrong server IP address
SOCKERR_PORTZERO - Server port zero
SOCKERR_TIMEOUT - Timeout occurred during request connection
SOCK_BUSY - In non-block io mode, it returned immediately

Definition at line **259** of file **socket.c**.

References **CHECK_SOCKINIT**, **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **getSn_CR**, **getSn_IR**, **getSn_SR**, **setSn_CR**, **setSn_DIPR**, **setSn_DPORT**, **setSn_IR**, **Sn_CR_CONNECT**, **Sn_IR_TIMEOUT**, **Sn_MR_TCP**, **SOCK_BUSY**, **SOCK_CLOSED**, **SOCK_ESTABLISHED**, **SOCK_OK**, **SOCKERR_IPINVALID**, **SOCKERR_PORTZERO**, **SOCKERR_SOCKCLOSED**, and **SOCKERR_TIMEOUT**.

int8_t disconnect (uint8_t sn)

Try to disconnect a connection socket.

It sends request message to disconnect the TCP socket 'sn' passed as parameter to the server or client.

Note

It is valid only in TCP server or client mode.
In block io mode, it does not return until disconnection is completed.

In Non-block io mode, it return **SOCK_BUSY** immediately.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

Success : **SOCK_OK**
Fail :
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_SOCKMODE - Invalid operation in the socket
SOCKERR_TIMEOUT - Timeout occurred
SOCK_BUSY - Socket is busy.

Definition at line **299** of file **socket.c**.

References **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **close()**, **getSn_CR**, **getSn_IR**, **getSn_SR**, **setSn_CR**, **Sn_CR_DISCON**, **Sn_IR_TIMEOUT**, **Sn_MR_TCP**, **SOCK_BUSY**, **SOCK_CLOSED**, **SOCK_OK**, and **SOCKERR_TIMEOUT**.

```
int32_t send ( uint8_t    sn,
               uint8_t *  buf,
               uint16_t   len
             )
```

Send data to the connected peer in TCP socket.

It is used to send outgoing data to the connected socket.

Note

It is valid only in TCP server or client mode. It can't send data greater than socket buffer size.
In block io mode, It doesn't return until data send is completed - socket buffer size is greater than data.
In non-block io mode, It return **SOCK_BUSY** immediately when socket buffer is not enough.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

buf Pointer buffer containing data to be sent.
len The byte length of data in buf.

Returns
Success : The sent data size
Fail :
SOCKERR_SOCKSTATUS - Invalid socket status for socket operation
SOCKERR_TIMEOUT - Timeout occurred
SOCKERR_SOCKMODE - Invalid operation in the socket
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_DATALEN - zero data length
SOCK_BUSY - Socket is busy.

Definition at line **319** of file **socket.c**.

References **CHECK_SOCKDATA**, **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **close()**, **getSn_CR**, **getSn_IR**, **getSn_SR**, **getSn_TX_FSR()**, **getSn_TX_RD**, **getSn_TxMAX**, **setSn_CR**, **setSn_IR**, **setSn_TX_WRSR**, **Sn_CR_SEND**, **Sn_IR_SENDOK**, **Sn_IR_TIMEOUT**, **Sn_MR_TCP**, **SOCK_BUSY**, **SOCK_CLOSE_WAIT**, **SOCK_ESTABLISHED**, **SOCKERR_SOCKSTATUS**, **SOCKERR_TIMEOUT**, and **wiz_send_data()**.

```
int32_t recv ( uint8_t    sn,
               uint8_t *  buf,
               uint16_t   len
             )
```

Receive data from the connected peer.

It is used to read incoming data from the connected socket.
It waits for data as much as the application wants to receive.

Note
It is valid only in TCP server or client mode. It can't receive data greater than socket buffer size.

In block io mode, it doesn't return until data reception is completed - data is filled as *len* in socket buffer.
In non-block io mode, it return **SOCK_BUSY** immediately when *len* is greater than data size in socket buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

buf Pointer buffer to read incoming data.

len The max data length of data in buf.

Returns

Success : The real received data size
Fail :
SOCKERR_SOCKSTATUS - Invalid socket status for socket operation
SOCKERR_SOCKMODE - Invalid operation in the socket
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_DATALEN - zero data length
SOCK_BUSY - Socket is busy.

Definition at line **387** of file **socket.c**.

References **CHECK_SOCKDATA**, **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **close()**, **getMR**, **getSn_CR**, **getSn_MR**, **getSn_RX_RSR()**, **getSn_RxMAX**, **getSn_SR**, **getSn_TX_FSR()**, **getSn_TxMAX**, **MR_FS**, **PACK_COMPLETED**, **PACK_FIFOBYTE**, **PACK_FIRST**, **PACK_REMAINED**, **setSn_CR**, **Sn_CR_RECV**, **Sn_MR_ALIGN**, **Sn_MR_TCP**, **SOCK_BUSY**, **SOCK_CLOSE_WAIT**, **SOCK_ESTABLISHED**, **sock_pack_info**, **SOCKERR_SOCKSTATUS**, and **wiz_recv_data()**.

```
int32_t sendto ( uint8_t    sn,
                 uint8_t *  buf,
                 uint16_t   len,
                 uint8_t *  addr,
                 uint16_t   port
               )
```

Sends datagram to the peer with destination IP address and port number passed as parameter.

It sends datagram of UDP or MACRAW to the peer with destination IP address and port number passed as parameter.
Even if the connectionless socket has been previously connected to a specific address, the address and port number parameters override the destination address for that particular datagram only.

Note

In block io mode, It doesn't return until data send is completed - socket buffer size is greater than *len*. In non-block io mode, It return **SOCK_BUSY** immediately when socket buffer is not enough.

Parameters

- **sn** Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
- **buf** Pointer buffer to send outgoing data.
- **len** The byte length of data in buf.
- **addr** Pointer variable of destination IP address. It should be allocated 4 bytes.
- **port** Destination port number.

Returns

Success : The sent data size
Fail :
SOCKERR_SOCKNUM - Invalid socket number
SOCKERR_SOCKMODE - Invalid operation in the socket
SOCKERR_SOCKSTATUS - Invalid socket status for socket operation
SOCKERR_DATALEN - zero data length
SOCKERR_IPINVALID - Wrong server IP address
SOCKERR_PORTZERO - Server port zero
SOCKERR_SOCKCLOSED - Socket unexpectedly closed
SOCKERR_TIMEOUT - Timeout occurred
SOCK_BUSY - Socket is busy.

Definition at line **492** of file **socket.c**.

References **CHECK_SOCKDATA**, **CHECK_SOCKNUM**, **getSIPR**, **getSn_CR**, **getSn_IR**, **getSn_MR**, **getSn_SR**, **getSn_TX_FSR()**, **getSn_TxMAX**, **getSUBR**, **setSn_CR**, **setSn_DIPR**, **setSn_DPORT**, **setSn_IR**, **setSn_TX_WRSR**, **setSUBR**, **Sn_CR_SEND**, **Sn_IR_SENDOK**, **Sn_IR_TIMEOUT**, **Sn_MR_MACRAW**, **Sn_MR_UDP**, **SOCK_BUSY**, **SOCK_CLOSED**, **SOCK_MACRAW**, **SOCK_UDP**, **SOCKERR_IPINVALID**, **SOCKERR_PORTZERO**, **SOCKERR_SOCKCLOSED**, **SOCKERR_SOCKMODE**, **SOCKERR_SOCKSTATUS**, **SOCKERR_TIMEOUT**, and **wiz_send_data()**.

Referenced by **close()**.

```
int32_t recvfrom ( uint8_t      sn,
                   uint8_t *    buf,
                   uint16_t     len,
                   uint8_t *    addr,
                   uint16_t *   port
                 )
```

Receive datagram of UDP or MACRAW.

This function is an application I/F function which is used to receive the data in other then TCP mode.
This function is used to receive UDP and MAC_RAW mode, and handle the header as well. This function can divide to received the packet data. On the MACRAW SOCKET, the addr and port parameters are ignored.

Note

In block io mode, it doesn't return until data reception is completed - data is filled as *len* in socket buffer In non-block io mode, it return **SOCK_BUSY** immediately when *len* is greater than data size in socket buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

buf Pointer buffer to read incoming data.

len The max data length of data in buf. When the received packet size <= len, receives data as packet sized. When others, receives data as len.

addr Pointer variable of destination IP address. It should be allocated 4 bytes. It is valid only when the first call recvfrom for receiving the packet. When it is valid, packinfo[7] should be set as '1' after call getsockopt(sn, SO_PACKINFO, &packinfo).

port Pointer variable of destination port number. It is valid only when the first call recvform for receiving the packet. When it is valid, packinfo[7] should be set as '1' after call getsockopt(sn, SO_PACKINFO, &packinfo).

Returns

Success : This function return real received data size for success.
Fail : **SOCKERR_DATALEN** - zero data length
SOCKERR_SOCKMODE - Invalid operation in the socket
SOCKERR_SOCKNUM - Invalid socket number
SOCKBUSY - Socket is busy.

Definition at line **588** of file **socket.c**.

References **CHECK_SOCKDATA**, **CHECK_SOCKNUM**, **close()**, **getMR**, **getSn_CR**, **getSn_MR**, **getSn_RX_RSR()**, **getSn_SR**, **MR_FS**, **PACK_COMPLETED**, **PACK_FIFOBYTE**, **PACK_FIRST**, **PACK_REMAINED**, **setSn_CR**, **Sn_CR_RECV**, **Sn_MR_IPRAW**, **Sn_MR_MACRAW**, **Sn_MR_PPPoE**, **Sn_MR_UDP**, **SOCK_BUSY**, **SOCK_CLOSED**, **sock_pack_info**, **SOCKERR_SOCKCLOSED**, **SOCKERR_SOCKMODE**, **SOCKFATAL_PACKLEN**, **wiz_recv_data()**, and **wiz_recv_ignore()**.

```
int8_t ctlsocket ( uint8_t          sn,
                   ctlsock_type     cstype,
                   void *           arg
                 )
```

Control socket.

Control IO mode, Interrupt & Mask of socket and get the socket buffer information. Refer to **ctlsock_type**.

Parameters

sn socket number

cstype type of control socket. refer to **ctlsock_type**.

arg Data type and value is determined according to **ctlsock_ty**

cstype	**data** type	**value**
CS_SET_IOMODE **CS_GET_IOMODE**	uint8_t	**SOCK_IO_BLO** **SOCK_IO_NONI**
CS_GET_MAXTXBUF **CS_GET_MAXRXBUF**	uint16_t	0 ~ 16K
CS_CLR_INTERRUPT **CS_GET_INTERRUPT** **CS_SET_INTMASK** **CS_GET_INTMASK**	**sockint_kind**	**SIK_CONNECTE**

Returns

Success **SOCK_OK**

fail **SOCKERR_ARG** - Invalid argument

Definition at line **764** of file **socket.c**.

References **CHECK_SOCKNUM**, **CS_CLR_INTERRUPT**, **CS_GET_INTERRUPT**, **CS_GET_INTMASK**, **CS_GET_IOMODE**, **CS_GET_MAXRXBUF**, **CS_GET_MAXTXBUF**, **CS_SET_INTMASK**, **CS_SET_IOMODE**, **getSn_IMR**, **getSn_IR**, **getSn_RxMAX**, **getSn_Tx** **setSn_IMR**, **setSn_IR**, **SIK_ALL**, **SOCK_IO_BLOCK**, **SOCK_IO_NONBLOCK**, **SOCK_OK**, and **SOCKERR_ARG**.

```
int8_t setsockopt ( uint8_t      sn,
                    sockopt_type sotype,
                    void *       arg
                  )
```

set socket options

Set socket option like as TTL, MSS, TOS, and so on. Refer to **sockopt_type**.

Parameters

sn socket number

sotype socket option type. refer to **sockopt_type**

arg Data type and value is determined according to *sotype*.

sotype	**data** type	**value**
SO_TTL	uint8_t	0 ~ 255
SO_TOS	uint8_t	0 ~ 255
SO_MSS	uint16_t	0 ~ 65535
SO_DESTIP	uint8_t[4]	
SO_DESTPORT	uint16_t	0 ~ 65535
SO_KEEPALIVESEND	null	null
SO_KEEPALIVEAUTO	uint8_t	0 ~ 255

Returns

- **Success** : **SOCK_OK**
- **Fail**
  - **SOCKERR_SOCKNUM** - Invalid Socket number
  - **SOCKERR_SOCKMODE** - Invalid socket mode
  - **SOCKERR_SOCKOPT** - Invalid socket option or its value
  - **SOCKERR_TIMEOUT** - Timeout occurred when sending keep-alive packet

Definition at line **810** of file **socket.c**.

References **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **getSn_CR**, **getSn_IR**, **getSn_KPALVTR**, **setSn_CR**, **setSn_DIPR**, **setSn_DPORT**, **setSn_IR**, **setSn_KPALVTR**, **setSn_MSSR**, **setSn_TOS**, **setSn_TTL**, **Sn_CR_SEND_KEEP**, **Sn_IR_TIMEOUT**, **Sn_MR_TCP**, **SO_DESTIP**, **SO_DESTPORT**, **SO_KEEPALIVEAUTO**, **SO_KEEPALIVESEND**, **SO_MSS**, **SO_TOS**, **SO_TTL**, **SOCK_OK**, **SOCKERR_ARG**,

SOCKERR_SOCKOPT, and **SOCKERR_TIMEOUT**.

```
int8_t getsockopt ( uint8_t      sn,
                    sockopt_type sotype,
                    void *       arg
                  )
```

get socket options

Get socket option like as FLAG, TTL, MSS, and so on. Refer to **sockopt_type**

Parameters

sn socket number

sotype socket option type. refer to **sockopt_type**

arg Data type and value is determined according to *sotype*.

sotype	**data** type	**value**
SO_FLAG	uint8_t	**SF_ETHER_OWN**, etc...
SO_TOS	uint8_t	0 ~ 255
SO_MSS	uint16_t	0 ~ 65535
SO_DESTIP	uint8_t[4]	
SO_DESTPORT	uint16_t	
SO_KEEPALIVEAUTO	uint8_t	0 ~ 255
SO_SENDBUF	uint16_t	0 ~ 65535
SO_RECVBUF	uint16_t	0 ~ 65535
SO_STATUS	uint8_t	**SOCK_ESTABLISHI** etc..
SO_REMAINSIZE	uint16_t	0~ 65535
SO_PACKINFO	uint8_t	**PACK_FIRST**, etc...

Returns

- **Success** : **SOCK_OK**

- **Fail**
  - **SOCKERR_SOCKNUM** - Invalid Socket number
  - **SOCKERR_SOCKOPT** - Invalid socket option or its value
  - **SOCKERR_SOCKMODE** - Invalid socket mode

Note

The option as PACK_REMAINED and SO_PACKINFO is valid only NON-TCP mode and after call **recvfrom()**.
When SO_PACKINFO value is PACK_FIRST and the return value **recvfrom()** is zero, This means the zero byte UDP data(UDP Head only) received.

Definition at line **863** of file **socket.c**.

References **CHECK_SOCKMODE**, **CHECK_SOCKNUM**, **getSn_DIPR**, **getSn_DPORT**, **getSn_KPALVTR**, **getSn_MR**, **getSn_MSSR**, **getSn_RX_RSR()**, **getSn_SR**, **getSn_TOS**, **getSn_TTL**, **getSn_TX_FSR()**, **Sn_MR_TCP**, **SO_DESTIP**, **SO_DESTPORT**, **SO_FLAG**, **SO_KEEPALIVEAUTO**, **SO_MSS**, **SO_PACKINFO**, **SO_RECVBUF**, **SO_REMAINSIZE**, **SO_SENDBUF**, **SO_STATUS**, **SO_TOS**, **SO_TTL**, **SOCK_OK**, **sock_pack_info**, and **SOCKERR_SOCKOPT**.

Socket APIs

2. WIZnet Extra Functions

These functions is optional function. It could be replaced at WIZCHIP I/O function because they were made by WIZCHIP I/O functions.
More...

Functions

int8_t **ctlwizchip** (**ctlwizchip_type** cwtype, void *arg)
Controls to the WIZCHIP. More...

int8_t **ctlnetwork** (**ctlnetwork_type** cntype, void *arg)
Controls to network. More...

void **wizchip_sw_reset** (void)
Reset WIZCHIP by softly. More...

int8_t **wizchip_init** (uint8_t *txsize, uint8_t *rxsize)
Initializes WIZCHIP with socket buffer size. More...

void **wizchip_clrinterrupt** (**intr_kind** intr)
Clear Interrupt of WIZCHIP. More...

intr_kind **wizchip_getinterrupt** (void)
Get Interrupt of WIZCHIP. More...

void **wizchip_setinterruptmask** (**intr_kind** intr)
Mask or Unmask Interrupt of WIZCHIP. More...

intr_kind **wizchip_getinterruptmask** (void)
Get Interrupt mask of WIZCHIP. More...

void **wizphy_setphyconf** (**wiz_PhyConf** *phyconf)
Set the phy information for WIZCHIP without power mode. More...

void **wizphy_getphyconf** (**wiz_PhyConf** *phyconf)
Get phy configuration information. More...

void **wizphy_getphystat** (**wiz_PhyConf** *phyconf)
Get phy status. More...

int8_t	**wizphy_setphypmode** (uint8_t pmode) set the power mode of phy inside WIZCHIP. Refer to **PHYCFGR** in W5500, **PHYSTATUS** in W5200 More...
void	**wizchip_setnetinfo** (**wiz_NetInfo** *pnetinfo) Set the network information for WIZCHIP. More...
void	**wizchip_getnetinfo** (**wiz_NetInfo** *pnetinfo) Get the network information for WIZCHIP. More...
int8_t	**wizchip_setnetmode** (**netmode_type** netmode) Set the network mode such WOL, PPPoE, Ping Block, and etc. More...
netmode_type	**wizchip_getnetmode** (void) Get the network mode such WOL, PPPoE, Ping Block, and etc. More...
void	**wizchip_settimeout** (**wiz_NetTimeout** *nettime) Set retry time value(*RTR*) and retry count(*RCR*). More...
void	**wizchip_gettimeout** (**wiz_NetTimeout** *nettime) Get retry time value(*RTR*) and retry count(*RCR*). More...

Detailed Description

These functions is optional function. It could be replaced at WIZCHIP I/O function because they were made by WIZCHIP I/O functions.

There are functions of configuring WIZCHIP, network, interrupt, phy, network information and timer.

Function Documentation

int8_t ctlwizchip (ctlwizchip_type cwtype,
void * arg
)

Controls to the WIZCHIP.

Resets WIZCHIP & internal PHY, Configures PHY mode, Monitor PHY(Link,Speed,Half/Full/Auto), controls interrupt & mask and so on.

Parameters

cwtype : Decides to the control type
arg : arg type is dependent on cwtype.

Returns

0 : Success
-1 : Fail because of invalid **ctlwizchip_type** or unsupported **ctlwizchip_type** in WIZCHIP

Definition at line **277** of file **wizchip_conf.c**.

References **_WIZCHIP_SOCK_NUM_**, **CW_CLR_INTERRUPT**, **CW_GET_ID**, **CW_GET_INTERRUPT**, **CW_GET_INTRMASK**, **CW_GET_INTRTIME**, **CW_GET_PHYCONF**, **CW_GET_PHYLINK**, **CW_GET_PHYPOWMODE**, **CW_GET_PHYSTATUS**, **CW_INIT_WIZCHIP**, **CW_RESET_PHY**, **CW_RESET_WIZCHIP**, **CW_SET_INTRMASK**, **CW_SET_INTRTIME**, **CW_SET_PHYCONF**, **CW_SET_PHYPOWMODE**, **getINTLEVEL**, **__WIZCHIP::id**, **setINTLEVEL**, **wizchip_clrinterrupt()**, **wizchip_getinterrupt()**, **wizchip_getinterruptmask()**, **wizchip_init()**, **wizchip_setinterruptmask()**, **wizchip_sw_reset()**, **wizphy_getphyconf()**, **wizphy_getphylink()**, **wizphy_getphypmode()**, **wizphy_reset()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

int8_t ctlnetwork (ctlnetwork_type cntype,
void * arg
)

Controls to network.

Controls to network environment, mode, timeout and so on.

Parameters

cntype : Input. Decides to the control type
arg : Inout. arg type is dependent on cntype.

Returns

-1 : Fail because of invalid **ctlnetwork_type** or unsupported **ctlnetwork_type** in WIZCHIP
0 : Success

Definition at line **359** of file **wizchip_conf.c**.

References **CN_GET_NETINFO**, **CN_GET_NETMODE**, **CN_GET_TIMEOUT**, **CN_SET_NETINFO**, **CN_SET_NETMODE**, **CN_SET_TIMEOUT**, **wizchip_getnetinfo()**, **wizchip_getnetmode()**, **wizchip_gettimeout()**, **wizchip_setnetinfo()**, **wizchip_setnetmode()**, and **wizchip_settimeout()**.

void wizchip_sw_reset (void)

Reset WIZCHIP by softly.

Definition at line **387** of file **wizchip_conf.c**.

References **getGAR**, **getMR**, **getSHAR**, **getSIPR**, **getSUBR**, **MR_IND**, **MR_RST**, **setGAR**, **setMR**, **setSHAR**, **setSIPR**, and **setSUBR**.

Referenced by **ctlwizchip()**, and **wizchip_init()**.

int8_t wizchip_init (uint8_t * txsize,
uint8_t * rxsize
)

Initializes WIZCHIP with socket buffer size.

Parameters

txsize Socket tx buffer sizes. If null, initialized the default size 2KB.

rxsize Socket rx buffer sizes. If null, initialized the default size 2KB.

Returns

0 : succcess
-1 : fail. Invalid buffer size

Definition at line **412** of file **wizchip_conf.c**.

References **_WIZCHIP_SOCK_NUM_**, **setSn_RXBUF_SIZE**, **setSn_TXBUF_SIZE**, and **wizchip_sw_reset()**.

Referenced by **ctlwizchip()**.

void wizchip_clrinterrupt (intr_kind intr)

Clear Interrupt of WIZCHIP.

Parameters

intr : **intr_kind** value operated OR. It can type-cast to uint16_t.

Definition at line **464** of file **wizchip_conf.c**.

References **setIR**, and **setSIR**.

Referenced by **ctlwizchip()**.

intr_kind wizchip_getinterrupt (void)

Get Interrupt of WIZCHIP.

Returns
intr_kind value operated OR. It can type-cast to uint16_t.

Definition at line **491** of file **wizchip_conf.c**.

References **getIR**, and **getSIR**.

Referenced by **ctlwizchip()**.

void wizchip_setinterruptmask (intr_kind intr)

Mask or Unmask Interrupt of WIZCHIP.

Parameters
intr : **intr_kind** value operated OR. It can type-cast to uint16_t.

Definition at line **522** of file **wizchip_conf.c**.

References **setIMR**, and **setSIMR**.

Referenced by **ctlwizchip()**.

intr_kind wizchip_getinterruptmask (void)

Get Interrupt mask of WIZCHIP.

Returns
: The operated OR vaule of **intr_kind**. It can type-cast to uint16_t.

Definition at line **546** of file **wizchip_conf.c**.

References **getIMR**, and **getSIMR**.

Referenced by **ctlwizchip()**.

void wizphy_setphyconf (wiz_PhyConf * phyconf)

Set the phy information for WIZCHIP without power mode.

Parameters

phyconf : **wiz_PhyConf**

Definition at line **627** of file **wizchip_conf.c**.

References **wiz_PhyConf_t::by**, **wiz_PhyConf_t::duplex**, **wiz_PhyConf_t::mode**, **PHY_CONFBY_SW**, **PHY_DUPLEX_FULL**, **PHY_MODE_AUTONEGO**, **PHY_SPEED_100**, **PHYCFGR_OPMD**, **PHYCFGR_OPMDC_100F**, **PHYCFGR_OPMDC_100H**, **PHYCFGR_OPMDC_10F**, **PHYCFGR_OPMDC_10H**, **PHYCFGR_OPMDC_ALLA**, **setPHYCFGR**, **wiz_PhyConf_t::speed**, and **wizphy_reset()**.

Referenced by **ctlwizchip()**.

void wizphy_getphyconf (wiz_PhyConf * phyconf)

Get phy configuration information.

Parameters

phyconf : **wiz_PhyConf**

Definition at line **657** of file **wizchip_conf.c**.

References **wiz_PhyConf_t::by**, **wiz_PhyConf_t::duplex**, **getPHYCFGR**, **wiz_PhyConf_t::mode**, **PHY_CONFBY_HW**, **PHY_CONFBY_SW**, **PHY_DUPLEX_FULL**, **PHY_DUPLEX_HALF**, **PHY_MODE_AUTONEGO**, **PHY_MODE_MANUAL**, **PHY_SPEED_10**, **PHY_SPEED_100**, **PHYCFGR_OPMD**,

PHYCFGR_OPMDC_100F, **PHYCFGR_OPMDC_100FA**, **PHYCFGR_OPMDC_100H**, **PHYCFGR_OPMDC_10F**, **PHYCFGR_OPMDC_ALLA**, and **wiz_PhyConf_t::speed**.

Referenced by **ctlwizchip()**.

void wizphy_getphystat (wiz_PhyConf * phyconf)

Get phy status.

Parameters

phyconf : **wiz_PhyConf**

Definition at line **696** of file **wizchip_conf.c**.

References **wiz_PhyConf_t::duplex**, **getPHYCFGR**, **PHY_DUPLEX_FULL**, **PHY_DUPLEX_HALF**, **PHY_SPEED_10**, **PHY_SPEED_100**, **PHYCFGR_DPX_FULL**, **PHYCFGR_SPD_100**, and **wiz_PhyConf_t::speed**.

int8_t wizphy_setphypmode (uint8_t pmode)

set the power mode of phy inside WIZCHIP. Refer to **PHYCFGR** in W5500, **PHYSTATUS** in W5200

Parameters

pmode Settig value of power down mode.

Definition at line **703** of file **wizchip_conf.c**.

References **getPHYCFGR**, **PHY_POWER_DOWN**, **PHYCFGR_OPMD**, **PHYCFGR_OPMDC_ALLA**, **PHYCFGR_OPMDC_PDOWN**, **setPHYCFGR**, and **wizphy_reset()**.

Referenced by **ctlwizchip()**.

void wizchip_setnetinfo (wiz_NetInfo * pnetinfo)

Set the network information for WIZCHIP.

Parameters

pnetinfo : wizNetInfo

Definition at line **729** of file **wizchip_conf.c**.

References **wiz_NetInfo_t::dhcp**, **wiz_NetInfo_t::dns**, **wiz_NetInfo_t::gw**, **wiz_NetInfo_t::ip**, **wiz_NetInfo_t::mac**, **setGAR**, **setSHAR**, **setSIPR**, **setSUBR**, and **wiz_NetInfo_t::sn**.

Referenced by **ctlnetwork()**.

void wizchip_getnetinfo (wiz_NetInfo * pnetinfo)

Get the network information for WIZCHIP.

Parameters

pnetinfo : wizNetInfo

Definition at line **742** of file **wizchip_conf.c**.

References **wiz_NetInfo_t::dhcp**, **wiz_NetInfo_t::dns**, **getGAR**, **getSHAR**, **getSIPR**, **getSUBR**, **wiz_NetInfo_t::gw**, **wiz_NetInfo_t::ip**, **wiz_NetInfo_t::mac**, and **wiz_NetInfo_t::sn**.

Referenced by **ctlnetwork()**.

int8_t wizchip_setnetmode (netmode_type netmode)

Set the network mode such WOL, PPPoE, Ping Block, and etc.

Parameters

pnetinfo Value of network mode. Refer to **netmode_type**.

Definition at line **755** of file **wizchip_conf.c**.

References **getMR**, **NM_FORCEARP**, **NM_PINGBLOCK**, **NM_PPPOE**, **NM_WAKEONLAN**, and **setMR**.

Referenced by **ctlnetwork()**.

netmode_type **wizchip_getnetmode (void)**

Get the network mode such WOL, PPPoE, Ping Block, and etc.

Returns

Value of network mode. Refer to **netmode_type**.

Definition at line **769** of file **wizchip_conf.c**.

References **getMR**.

Referenced by **ctlnetwork()**.

void wizchip_settimeout (wiz_NetTimeout * nettime)

Set retry time value(*RTR*) and retry count(*RCR*).

RTR configures the retransmission timeout period and *RCR* configures the number of time of retransmission.

Parameters

nettime *RTR* value and *RCR* value. Refer to **wiz_NetTimeout**.

Definition at line **774** of file **wizchip_conf.c**.

References **wiz_NetTimeout_t::retry_cnt**, **setRCR**, **setRTR**, and **wiz_NetTimeout_t::time_100us**.

Referenced by **ctlnetwork()**.

void wizchip_gettimeout (wiz_NetTimeout * nettime)

Get retry time value(*RTR*) and retry count(*RCR*).

RTR configures the retransmission timeout period and *RCR* configures the number of time of retransmission.

Parameters

nettime *RTR* value and *RCR* value. Refer to **wiz_NetTimeout**.

Definition at line **780** of file **wizchip_conf.c**.

References **getRCR**, **getRTR**, **wiz_NetTimeout_t::retry_cnt**, and **wiz_NetTimeout_t::time_100us**.

Referenced by **ctlnetwork()**.

Socket APIs

DATA TYPE

Classes

struct	**__WIZCHIP** The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**. More...
union	**__WIZCHIP::_IF**
struct	**__WIZCHIP::_CS**
struct	**__WIZCHIP::_CRIS**
struct	**wiz_PhyConf_t**
struct	**wiz_NetInfo_t**
struct	**wiz_NetTimeout_t**

Typedefs

typedef struct __**WIZCHIP**	**_WIZCHIP** The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**. More...
typedef struct **wiz_PhyConf_t**	**wiz_PhyConf**
typedef struct **wiz_NetInfo_t**	**wiz_NetInfo**
typedef struct **wiz_NetTimeout_t**	**wiz_NetTimeout**

Enumerations

enum	**sockint_kind** {
	SIK_CONNECTED = (1 << 0), **SIK_DISCONNECTED** = (1 << 1), **SIK_RECEIVED** = (1 << 2), **SIK_TIMEOUT** = (1 << 3),
	SIK_SENT = (1 << 4), **SIK_ALL** = 0x1F
	}
	The kind of Socket Interrupt. More...

enum	**ctlsock_type** {
	CS_SET_IOMODE, **CS_GET_IOMODE**, **CS_GET_MAXTXBUF**, **CS_GET_MAXRXBUF**,
	CS_CLR_INTERRUPT, **CS_GET_INTERRUPT**, **CS_SET_INTMASK**, **CS_GET_INTMASK**
	}
	The type of **ctlsocket()**. More...

enum	**sockopt_type** {
	SO_FLAG, **SO_TTL**, **SO_TOS**, **SO_MSS**,
	SO_DESTIP, **SO_DESTPORT**, **SO_KEEPALIVESEND**, **SO_KEEPALIVEAUTO**,
	SO_SENDBUF, **SO_RECVBUF**, **SO_STATUS**, **SO_REMAINSIZE**,
	SO_PACKINFO
	}
	The type of socket option in **setsockopt()** or **getsockopt()** More...

enum	**ctlwizchip_type** {
	CW_RESET_WIZCHIP, **CW_INIT_WIZCHIP**, **CW_GET_INTERRUPT**, **CW_CLR_INTERRUPT**,
	CW_SET_INTRMASK, **CW_GET_INTRMASK**, **CW_SET_INTRTIME**, **CW_GET_INTRTIME**,
	CW_GET_ID, **CW_RESET_PHY**, **CW_SET_PHYCONF**, **CW_GET_PHYCONF**,
	CW_GET_PHYSTATUS, **CW_SET_PHYPOWMODE**, **CW_GET_PHYPOWMODE**, **CW_GET_PHYLINK**
	}

enum	**ctlnetwork_type** {
	CN_SET_NETINFO, **CN_GET_NETINFO**,
	CN_SET_NETMODE, **CN_GET_NETMODE**,
	CN_SET_TIMEOUT, **CN_GET_TIMEOUT**
	}

enum	**intr_kind** {
	IK_WOL = (1 << 4), **IK_PPPOE_TERMINATED** = (1 << 5),
	IK_DEST_UNREACH = (1 << 6), **IK_IP_CONFLICT** = (1 << 7),
	IK_SOCK_0 = (1 << 8), **IK_SOCK_1** = (1 << 9), **IK_SOCK_2** = (1 << 10), **IK_SOCK_3** = (1 << 11),
	IK_SOCK_4 = (1 << 12), **IK_SOCK_5** = (1 << 13),
	IK_SOCK_6 = (1 << 14), **IK_SOCK_7** = (1 << 15),
	IK_SOCK_ALL = (0xFF << 8)
	}

enum	**dhcp_mode** { **NETINFO_STATIC** = 1, **NETINFO_DHCP** }
enum	**netmode_type** { **NM_FORCEARP** = (1<<1), **NM_WAKEONLAN** = (1<<5), **NM_PINGBLOCK** = (1<<4), **NM_PPPOE** = (1<<3) }

Detailed Description

Typedef Documentation

typedef struct __WIZCHIP _WIZCHIP

The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**.

typedef struct wiz_PhyConf_t wiz_PhyConf

It configures PHY configuration when CW_SET PHYCONF or CW_GET_PHYCONF in W5500, and it indicates the real PHY status configured by HW or SW in all WIZCHIP.
Valid only in W5500.

typedef struct wiz_NetInfo_t wiz_NetInfo

Network Information for WIZCHIP

typedef struct wiz_NetTimeout_t wiz_NetTimeout

Used in CN_SET_TIMEOUT or CN_GET_TIMEOUT of **ctlwizchip()** for timeout configruation.

Enumeration Type Documentation

enum sockint_kind

The kind of Socket Interrupt.

See also
Sn_IR, **Sn_IMR**, **setSn_IR()**, **getSn_IR()**, **setSn_IMR()**, **getSn_IMR()**

Enumerator	
SIK_CONNECTED	connected
SIK_DISCONNECTED	disconnected
SIK_RECEIVED	data received
SIK_TIMEOUT	timeout occurred
SIK_SENT	send ok
SIK_ALL	all interrupt

Definition at line **345** of file **socket.h**.

enum ctlsock_type

The type of **ctlsocket()**.

Enumerator	
CS_SET_IOMODE	set socket IO mode with **SOCK_IO_BLOCK** or **SOCK_IO_NONBLOCK**
CS_GET_IOMODE	get socket IO mode
CS_GET_MAXTXBUF	get the size of socket buffer allocated in TX memory

CS_GET_MAXRXBUF	get the size of socket buffer allocated in RX memory
CS_CLR_INTERRUPT	clear the interrupt of socket with **sockint_kind**
CS_GET_INTERRUPT	get the socket interrupt. refer to **sockint_kind**
CS_SET_INTMASK	set the interrupt mask of socket with **sockint_kind**, Not supported in W5100
CS_GET_INTMASK	get the masked interrupt of socket. refer to **sockint_kind**, Not supported in W5100

Definition at line **361** of file **socket.h**.

enum sockopt_type

The type of socket option in **setsockopt()** or **getsockopt()**

Enumerator	
SO_FLAG	Valid only in **getsockopt()**, For set flag of socket refer to *flag* in **socket()**.
SO_TTL	Set TTL. **Sn_TTL** (**setSn_TTL()**, **getSn_TTL()**)
SO_TOS	Set TOS. **Sn_TOS** (**setSn_TOS()**, **getSn_TOS()**)
SO_MSS	Set MSS. **Sn_MSSR** (**setSn_MSSR()**, **getSn_MSSR()**)
SO_DESTIP	Set the destination IP address. **Sn_DIPR** (**setSn_DIPR()**, **getSn_DIPR()**)
SO_DESTPORT	Set the destination Port number. **Sn_DPORT** (**setSn_DPORT()**, **getSn_DPORT()**)
SO_KEEPALIVESEND	Valid only in setsockopt. Manually send keep-alive packet in TCP mode, Not

	supported in W5100.
SO_KEEPALIVEAUTO	Set/Get keep-alive auto transmission timer in TCP mode, Not supported in W5100, W5200.
SO_SENDBUF	Valid only in getsockopt. Get the free data size of Socekt TX buffer. **Sn_TX_FSR**, **getSn_TX_FSR()**
SO_RECVBUF	Valid only in getsockopt. Get the received data size in socket RX buffer. **Sn_RX_RSR**, **getSn_RX_RSR()**
SO_STATUS	Valid only in getsockopt. Get the socket status. **Sn_SR**, **getSn_SR()**
SO_REMAINSIZE	Valid only in getsockopt. Get the remained packet size in other then TCP mode.
SO_PACKINFO	Valid only in getsockopt. Get the packet information as **PACK_FIRST**, **PACK_REMAINED**, and **PACK_COMPLETED** in other then TCP mode.

Definition at line **380** of file **socket.h**.

enum ctlwizchip_type

WIZCHIP control type enumration used in **ctlwizchip()**.

Enumerator	
CW_RESET_WIZCHIP	Resets WIZCHIP by softly.
CW_INIT_WIZCHIP	Initializes to WIZCHIP with SOCKET buffer size 2 or 1 dimension array typed uint8_t.
CW_GET_INTERRUPT	Get Interrupt status of WIZCHIP.
CW_CLR_INTERRUPT	Clears interrupt.

CW_SET_INTRMASK	Masks interrupt.
CW_GET_INTRMASK	Get interrupt mask.
CW_SET_INTRTIME	Set interval time between the current and next interrupt.
CW_GET_INTRTIME	Set interval time between the current and next interrupt.
CW_GET_ID	Gets WIZCHIP name.
CW_RESET_PHY	Resets internal PHY. Valid Only W5500.
CW_SET_PHYCONF	When PHY configured by internal register, PHY operation mode (Manual/Auto, 10/100, Half/Full). Valid Only W5000.
CW_GET_PHYCONF	Get PHY operation mode in internal register. Valid Only W5500.
CW_GET_PHYSTATUS	Get real PHY status on operating. Valid Only W5500.
CW_SET_PHYPOWMODE	Set PHY power mode as normal and down when PHYSTATUS.OPMD == 1. Valid Only W5500.
CW_GET_PHYPOWMODE	Get PHY Power mode as down or normal, Valid Only W5100, W5200.
CW_GET_PHYLINK	Get PHY Link status, Valid Only W5100, W5200.

Definition at line **262** of file **wizchip_conf.h**.

enum ctlnetwork_type

Network control type enumration used in **ctlnetwork()**.

Enumerator	
CN_SET_NETINFO	Set Network with **wiz_NetInfo**.
CN_GET_NETINFO	Get Network with **wiz_NetInfo**.

CN_SET_NETMODE	Set network mode as WOL, PPPoE, Ping Block, and Force ARP mode.
CN_GET_NETMODE	Get network mode as WOL, PPPoE, Ping Block, and Force ARP mode.
CN_SET_TIMEOUT	Set network timeout as retry count and time.
CN_GET_TIMEOUT	Get network timeout as retry count and time.

Definition at line **293** of file **wizchip_conf.h**.

enum intr_kind

Interrupt kind when CW_SET_INTRRUPT, CW_GET_INTERRUPT, CW_SET_INTRMASK and CW_GET_INTRMASK is used in **ctlnetwork()**. It can be used with OR operation.

Enumerator	
IK_WOL	Wake On Lan by receiving the magic packet. Valid in W500.
IK_PPPOE_TERMINATED	PPPoE Disconnected.
IK_DEST_UNREACH	Destination IP & Port Unreachable, No use in W5200.
IK_IP_CONFLICT	IP conflict occurred.
IK_SOCK_0	Socket 0 interrupt.
IK_SOCK_1	Socket 1 interrupt.
IK_SOCK_2	Socket 2 interrupt.
IK_SOCK_3	Socket 3 interrupt.
IK_SOCK_4	Socket 4 interrupt, No use in 5100.
IK_SOCK_5	Socket 5 interrupt, No use in 5100.
IK_SOCK_6	Socket 6 interrupt, No use in 5100.
IK_SOCK_7	Socket 7 interrupt, No use in 5100.
IK_SOCK_ALL	All Socket interrupt.

Definition at line **309** of file **wizchip_conf.h**.

enum dhcp_mode

It used in setting dhcp_mode of **wiz_NetInfo**.

Enumerator	
NETINFO_STATIC	Static IP configuration by manually.
NETINFO_DHCP	Dynamic IP configruation from a DHCP sever.

Definition at line **379** of file **wizchip_conf.h**.

enum netmode_type

Network mode

Enumerator	
NM_FORCEARP	Force to APP send whenever udp data is sent. Valid only in W5500.
NM_WAKEONLAN	Wake On Lan.
NM_PINGBLOCK	Block ping-request.
NM_PPPOE	PPPoE mode.

Definition at line **403** of file **wizchip_conf.h**.

Socket APIs

Classes | Public Attributes | List of all members

__WIZCHIP Struct Reference

DATA TYPE

The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**. More...

```
#include <wizchip_conf.h>
```

Classes

struct	**_CRIS**
struct	**_CS**
union	**_IF**

Public Attributes

uint16_t	**if_mode** host interface mode More...
uint8_t	**id** [6] **WIZCHIP** ID such as **5100**, **5200**, **5500**, and so on. More...
struct __**WIZCHIP::_CRIS**	**CRIS**
struct __**WIZCHIP::_CS**	**CS**
union __**WIZCHIP::_IF**	**IF**

Detailed Description

The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**.

Definition at line **201** of file **wizchip_conf.h**.

Member Data Documentation

uint16_t __WIZCHIP::if_mode

host interface mode

Definition at line **203** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_bus_cbfunc()**, **reg_wizchip_spi_cbfunc()**, and **reg_wizchip_spiburst_cbfunc()**.

uint8_t __WIZCHIP::id[6]

WIZCHIP ID such as **5100**, **5200**, **5500**, and so on.

Definition at line **204** of file **wizchip_conf.h**.

Referenced by **ctlwizchip()**.

struct __WIZCHIP::_CRIS __WIZCHIP::CRIS

Referenced by **reg_wizchip_cris_cbfunc()**.

struct __WIZCHIP::_CS __WIZCHIP::CS

Referenced by **reg_wizchip_cs_cbfunc()**.

union __WIZCHIP::_IF __WIZCHIP::IF

Referenced by **reg_wizchip_bus_cbfunc()**,

reg_wizchip_spi_cbfunc(), and **reg_wizchip_spiburst_cbfunc()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

__WIZCHIP::_IF Union Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

struct { **iodata_t**(* **_read_data**)(uint32_t AddrSel)	
void(* **_write_data**)(uint32_t AddrSel, **iodata_t** wb)	
}	**BUS**
struct { uint8_t(* **_read_byte**)(void)	
void(* **_write_byte**)(uint8_t wb)	
void(* **_read_burst**)(uint8_t *pBuf, uint16_t len)	
void(* **_write_burst**)(uint8_t *pBuf, uint16_t len)	
}	**SPI**

Detailed Description

The set of interface IO callback func.

Definition at line **224** of file **wizchip_conf.h**.

Member Data Documentation

iodata_t(* __WIZCHIP::_IF::_read_data) (uint32_t AddrSel)

Definition at line **237** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_bus_cbfunc()**.

void(* __WIZCHIP::_IF::_write_data) (uint32_t AddrSel, iodata_t wb)

Definition at line **238** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_bus_cbfunc()**.

struct { ... } __WIZCHIP::_IF::BUS

For BUS interface IO

Referenced by **reg_wizchip_bus_cbfunc()**.

uint8_t(* __WIZCHIP::_IF::_read_byte) (void)

Definition at line **246** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_spi_cbfunc()**.

void(* __WIZCHIP::_IF::_write_byte) (uint8_t wb)

Definition at line **247** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_spi_cbfunc()**.

void(* __WIZCHIP::_IF::_read_burst) (uint8_t *pBuf, uint16_t len)

Definition at line **248** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_spiburst_cbfunc()**.

void(* __WIZCHIP::_IF::_write_burst) (uint8_t *pBuf, uint16_t len)

Definition at line **249** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_spiburst_cbfunc()**.

struct { ... } __WIZCHIP::_IF::SPI

For SPI interface IO

Referenced by **reg_wizchip_spi_cbfunc()**, and **reg_wizchip_spiburst_cbfunc()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

__WIZCHIP::_CS Struct Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

void(* **_select**)(void)
**WIZCHIP** selected More...

void(* **_deselect**)(void)
**WIZCHIP** deselected More...

Detailed Description

The set of _**WIZCHIP**_ select control callback func.

Definition at line **216** of file **wizchip_conf.h**.

Member Data Documentation

void(* __WIZCHIP::_CS::_select) (void)

**WIZCHIP** selected

Definition at line **218** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_cs_cbfunc()**.

void(* __WIZCHIP::_CS::_deselect) (void)

**WIZCHIP** deselected

Definition at line **219** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_cs_cbfunc()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

__WIZCHIP::_CRIS Struct Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

void(* **_enter**)(void)
crtical section enter More...

void(* **_exit**)(void)
critial section exit More...

Detailed Description

The set of critical section callback func.

Definition at line **208** of file **wizchip_conf.h**.

Member Data Documentation

void(* __WIZCHIP::_CRIS::_enter) (void)

crtical section enter

Definition at line **210** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_cris_cbfunc()**.

void(* __WIZCHIP::_CRIS::_exit) (void)

critial section exit

Definition at line **211** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_cris_cbfunc()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

wiz_PhyConf_t Struct Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

uint8_t **by**
set by **PHY_CONFBY_HW** or **PHY_CONFBY_SW** More...

uint8_t **mode**
set by **PHY_MODE_MANUAL** or **PHY_MODE_AUTONEGO** More...

uint8_t **speed**
set by **PHY_SPEED_10** or **PHY_SPEED_100** More...

uint8_t **duplex**
set by **PHY_DUPLEX_HALF PHY_DUPLEX_FULL** More...

Detailed Description

It configures PHY configuration when CW_SET PHYCONF or CW_GET_PHYCONF in W5500, and it indicates the real PHY status configured by HW or SW in all WIZCHIP.
Valid only in W5500.

Definition at line **364** of file **wizchip_conf.h**.

Member Data Documentation

uint8_t wiz_PhyConf_t::by

set by **PHY_CONFBY_HW** or **PHY_CONFBY_SW**

Definition at line **366** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

uint8_t wiz_PhyConf_t::mode

set by **PHY_MODE_MANUAL** or **PHY_MODE_AUTONEGO**

Definition at line **367** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

uint8_t wiz_PhyConf_t::speed

set by **PHY_SPEED_10** or **PHY_SPEED_100**

Definition at line **368** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_getphystat()**, and **wizphy_setphyconf()**.

uint8_t wiz_PhyConf_t::duplex

set by **PHY_DUPLEX_HALF PHY_DUPLEX_FULL**

Definition at line **369** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_getphystat()**, and **wizphy_setphyconf()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

wiz_NetInfo_t Struct Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

uint8_t	**mac** [6] Source Mac Address. More...
uint8_t	**ip** [4] Source IP Address. More...
uint8_t	**sn** [4] Subnet Mask. More...
uint8_t	**gw** [4] Gateway IP Address. More...
uint8_t	**dns** [4] DNS server IP Address. More...
dhcp_mode	**dhcp** 1 - Static, 2 - DHCP More...

Detailed Description

Network Information for WIZCHIP

Definition at line **389** of file **wizchip_conf.h**.

Member Data Documentation

uint8_t wiz_NetInfo_t::mac[6]

Source Mac Address.

Definition at line **391** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

uint8_t wiz_NetInfo_t::ip[4]

Source IP Address.

Definition at line **392** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

uint8_t wiz_NetInfo_t::sn[4]

Subnet Mask.

Definition at line **393** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

uint8_t wiz_NetInfo_t::gw[4]

Gateway IP Address.

Definition at line **394** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

uint8_t wiz_NetInfo_t::dns[4]

DNS server IP Address.

Definition at line **395** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

dhcp_mode wiz_NetInfo_t::dhcp

1 - Static, 2 - DHCP

Definition at line **396** of file **wizchip_conf.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_setnetinfo()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

Public Attributes | List of all members

wiz_NetTimeout_t Struct Reference

DATA TYPE

```
#include <wizchip_conf.h>
```

Public Attributes

uint8_t	**retry_cnt** retry count More...
uint16_t	**time_100us** time unit 100us More...

Detailed Description

Used in CN_SET_TIMEOUT or CN_GET_TIMEOUT of **ctlwizchip()** for timeout configruation.

Definition at line **417** of file **wizchip_conf.h**.

Member Data Documentation

uint8_t wiz_NetTimeout_t::retry_cnt

retry count

Definition at line **419** of file **wizchip_conf.h**.

Referenced by **wizchip_gettimeout()**, and **wizchip_settimeout()**.

uint16_t wiz_NetTimeout_t::time_100us

time unit 100us

Definition at line **420** of file **wizchip_conf.h**.

Referenced by **wizchip_gettimeout()**, and **wizchip_settimeout()**.

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

- Ethernet/**wizchip_conf.h**

Socket APIs

W5100

WHIZCHIP register defines and I/O functions of **W5100**. More...

Modules

WIZCHIP I/O functions
This supports the basic I/O functions for **WIZCHIP register**.

WIZCHIP register
WIZCHIP register defines register group of **W5100** .

Detailed Description

WHIZCHIP register defines and I/O functions of **W5100**.

- **WIZCHIP register** : **Common register** and **Socket register**
- **WIZCHIP I/O functions** : **Basic I/O function**, **Common register access functions** and **Socket register**

Socket APIs

WIZCHIP I/O functions

W5100

This supports the basic I/O functions for **WIZCHIP register**. More...

Modules

Basic I/O function
These are basic input/output functions to read values from register or write values to register.

Common register access functions
These are functions to access **common registers**.

Socket register access functions
These are functions to access **socket registers**.

Detailed Description

This supports the basic I/O functions for **WIZCHIP register**.

- **Basic I/O function**
  WIZCHIP_READ(), **WIZCHIP_WRITE()**, **WIZCHIP_READ_BUF()**, **WIZCHIP_WRITE_BUF()**

- **Common register access functions**
  1. **Mode**
     getMR(), **setMR()**
  2. **Interrupt**
     getIR(), **setIR()**, **getIMR()**, **setIMR()**,
  3. **Network Information**
     getSHAR(), **setSHAR()**, **getGAR()**, **setGAR()**, **getSUBR()**, **setSUBR()**, **getSIPR()**, **setSIPR()**
  4. **Retransmission**
     getRCR(), **setRCR()**, **getRTR()**, **setRTR()**
  5. **PPPoE**
     getPTIMER(), **setPTIMER()**, **getPMAGIC()**, **getPMAGIC()**
- **Socket register access functions**
  1. **SOCKET control**
     getSn_MR(), **setSn_MR()**, **getSn_CR()**, **setSn_CR()**, **getSn_IR()**, **setSn_IR()**
  2. **SOCKET information**
     getSn_SR(), **getSn_DHAR()**, **setSn_DHAR()**, **getSn_PORT()**, **setSn_PORT()**, **getSn_DIPR()**, **setSn_DIPR()**, **getSn_DPORT()**, **setSn_DPORT()** **getSn_MSSR()**, **setSn_MSSR()**
  3. **SOCKET communication**
     getSn_RXMEM_SIZE(), **setSn_RXMEM_SIZE()**, **getSn_TXMEM_SIZE()**, **setSn_TXMEM_SIZE()** **getSn_TX_RD()**, **getSn_TX_WR()**, **setSn_TX_WR()** **getSn_RX_RD()**, **setSn_RX_RD()**, **getSn_RX_WR()** **getSn_TX_FSR()**, **getSn_RX_RSR()**
  4. **IP header field**
     getSn_FRAG(), **setSn_FRAG()**, **getSn_TOS()**, **setSn_TOS()**

getSn_TTL(), setSn_TTL()

Socket APIs

Basic I/O function

These are basic input/output functions to read values from register or write values to register. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...
void	**wiz_recv_ignore** (uint8_t sn, uint16_t len) It discard the received data in RX memory. More...

Detailed Description

These are basic input/output functions to read values from register or write values to register.

Function Documentation

uint8_t WIZCHIP_READ (uint32_t AddrSel)

It reads 1 byte value from a register.

Parameters

AddrSel Register address

Returns

The value of register

void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb)

It writes 1 byte value to a register.

Parameters

AddrSel Register address

wb Write data

Returns

void

void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t * pBuf, uint16_t len)

It reads sequence data from registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to read data
len	Data length

```
void WIZCHIP_WRITE_BUF ( uint32_t  AddrSel,
                         uint8_t *  pBuf,
                         uint16_t  len
                       )
```

It writes sequence data to registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to write data
len	Data length

```
void wiz_send_data ( uint8_t    sn,
                     uint8_t *  wizdata,
                     uint16_t  len
                   )
```

It copies data to internal TX memory.

This function reads the Tx write pointer register and after that, it copies the *wizdata(pointer buffer)* of the length of *len(variable)* bytes to internal TX memory and updates the Tx write pointer register. This function is being called by **send()** and **sendto()** function also.

Parameters

sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
wizdata	Pointer buffer to write data
len	Data length

See also

wiz_recv_data()

Referenced by **send()**, and **sendto()**.

---

void wiz_recv_data (uint8_t sn,
uint8_t * wizdata,
uint16_t len
)

It copies data to your buffer from internal RX memory.

This function read the Rx read pointer register and after that, it copies the received data from internal RX memory to *wizdata(pointer variable)* of the length of *len(variable)* bytes. This function is being called by **recv()** also.

Parameters

sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
wizdata	Pointer buffer to read data
len	Data length

See also

wiz_send_data()

Referenced by **recv()**, and **recvfrom()**.

void wiz_recv_ignore (uint8_t sn,
uint16_t len
)

It discard the received data in RX memory.

It discards the data of the length of *len(variable)* bytes in internal RX memory.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
len	Data length

Referenced by **recvfrom()**.

Socket APIs

Common register access functions

These are functions to access **common registers**. More...

Macros

#define **setMR**(mr) (*((uint8_t*)**MR**) = mr)
Set Mode Register. More...

#define **getMR**() (*(uint8_t*)**MR**)
Get **MR**. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set **GAR**. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get **GAR**. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**,subr,4)
Set **SUBR**. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get **SUBR**. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set **SHAR**. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get **SHAR**. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set **SIPR**. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get **SIPR**. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xA0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xA0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**_IMR_**, imr)
Set *IMR* register. More...

#define **getIMR**() **WIZCHIP_READ**(**_IMR_**)
Get *IMR* register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(**_RTR_**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **setRMSR**(rmsr) **WIZCHIP_WRITE**(**RMSR**)
Get **RMSR** register. More...

#define **getRMSR**() **WIZCHIP_READ**()
Get **RMSR** register. More...

#define **setTMSR**(rmsr) **WIZCHIP_WRITE**(**TMSR**)
Get **TMSR** register. More...

#define **getPATR**() (((uint16_t)**WIZCHIP_READ**(**PATR**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PATR**,1)))
Get **TMSR** register. More...

#define **getPPPALGO**() **WIZCHIP_READ**(**PPPALGO**)

Get **PPPALGO** register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)
Get **PMAGIC** register. More...

Detailed Description

These are functions to access **common registers**.

Macro Definition Documentation

#define setMR (mr) (*((uint8_t*)MR) = mr)

Set Mode Register.

Parameters

(uint8_t)mr The value to be set.

See also

getMR()

Definition at line **1108** of file **w5100.h**.

Referenced by **wizchip_setnetmode()**, and **wizchip_sw_reset()**.

#define getMR ( ) (*(uint8_t*)MR)

Get **MR**.

Returns

uint8_t. The value of Mode register.

See also

setMR()

Definition at line **1120** of file **w5100.h**.

Referenced by **recv()**, **recvfrom()**, **wizchip_getnetmode()**, **wizchip_setnetmode()**, and **wizchip_sw_reset()**.

#define setGAR (gar) WIZCHIP_WRITE_BUF(GAR,gar,4)

Set **GAR**.

Parameters

(uint8_t*)gar Pointer variable to set gateway IP address. It should be allocated 4 bytes.

See also

getGAR()

Definition at line **1129** of file **w5100.h**.

Referenced by **wizchip_setnetinfo()**, and **wizchip_sw_reset()**.

#define getGAR (gar) WIZCHIP_READ_BUF(GAR,gar,4)

Get **GAR**.

Parameters

(uint8_t*)gar Pointer variable to get gateway IP address. It should be allocated 4 bytes.

See also

setGAR()

Definition at line **1138** of file **w5100.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_sw_reset()**.

#define setSUBR (subr) WIZCHIP_WRITE_BUF(SUBR,subr,4)

Set **SUBR**.

Parameters

(uint8_t*)subr Pointer variable to set subnet mask address. It should be allocated 4 bytes.

Note

If subr is null pointer, set the backup subnet to SUBR.
If subr is 0.0.0.0, back up SUBR and clear it.
Otherwize, set subr to SUBR

See also

getSUBR()

Definition at line **1150** of file **w5100.h**.

Referenced by **sendto()**, **wizchip_setnetinfo()**, and **wizchip_sw_reset()**.

#define getSUBR (subr) WIZCHIP_READ_BUF(SUBR, subr, 4)

Get **SUBR**.

Parameters

(uint8_t*)subr Pointer variable to get subnet mask address. It should be allocated 4 bytes.

See also

setSUBR()

Definition at line **1159** of file **w5100.h**.

Referenced by **sendto()**, **wizchip_getnetinfo()**, and **wizchip_sw_reset()**.

#define setSHAR (shar) WIZCHIP_WRITE_BUF(SHAR, shar, 6)

Set **SHAR**.

Parameters

(uint8_t*)shar Pointer variable to set local MAC address. It should be allocated 6 bytes.

See also
getSHAR()

Definition at line **1168** of file **w5100.h**.

Referenced by **wizchip_setnetinfo()**, and **wizchip_sw_reset()**.

#define getSHAR (shar) WIZCHIP_READ_BUF(SHAR, shar, 6)

Get **SHAR**.

Parameters

(uint8_t*)shar Pointer variable to get local MAC address. It should be allocated 6 bytes.

See also
setSHAR()

Definition at line **1177** of file **w5100.h**.

Referenced by **wizchip_getnetinfo()**, and **wizchip_sw_reset()**.

#define setSIPR (sipr) WIZCHIP_WRITE_BUF(SIPR, sipr, 4)

Set **SIPR**.

Parameters

(uint8_t*)sipr Pointer variable to set local IP address. It should be allocated 4 bytes.

See also
getSIPR()

Definition at line **1186** of file **w5100.h**.

Referenced by **wizchip_setnetinfo()**, and **wizchip_sw_reset()**.

#define getSIPR (sipr) WIZCHIP_READ_BUF(SIPR, sipr, 4)

Get **SIPR**.

Parameters

(uint8_t*)sipr Pointer variable to get local IP address. It should be allocated 4 bytes.

See also

setSIPR()

Definition at line **1195** of file **w5100.h**.

Referenced by **sendto()**, **socket()**, **wizchip_getnetinfo()**, and **wizchip_sw_reset()**.

#define setIR (ir) WIZCHIP_WRITE(IR, (ir & 0xA0))

Set **IR** register.

Parameters

(uint8_t)ir Value to set **IR** register.

See also

getIR()

Definition at line **1204** of file **w5100.h**.

Referenced by **wizchip_clrinterrupt()**.

#define getIR ( ) (WIZCHIP_READ(IR) & 0xA0)

Get **IR** register.

Returns

uint8_t. Value of **IR** register.

See also
setIR()

Definition at line **1212** of file **w5100.h**.

Referenced by **wizchip_getinterrupt()**.

#define setIMR (imr) WIZCHIP_WRITE(_IMR_, imr)

Set *IMR* register.

Parameters
(uint8_t)imr Value to set *IMR* register.

See also
getIMR()

Definition at line **1221** of file **w5100.h**.

Referenced by **wizchip_setinterruptmask()**.

#define getIMR ( ) WIZCHIP_READ(_IMR_)

Get *IMR* register.

Returns
uint8_t. Value of *IMR* register.

See also
setIMR()

Definition at line **1230** of file **w5100.h**.

Referenced by **wizchip_getinterruptmask()**.

#define setRTR (rtr)

Value:

```
{\
                WIZCHIP_WRITE(_RTR_,   (uint8_t)
    (rtr >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),
    (uint8_t) rtr); \
    }
```

Set *RTR* register.

Parameters

(uint16_t)rtr Value to set *RTR* register.

See also

getRTR()

Definition at line **1239** of file **w5100.h**.

Referenced by **wizchip_settimeout()**.

#define getRTR ( ) (((uint16_t)WIZCHIP_READ(_RTR_) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))

Get *RTR* register.

Returns

uint16_t. Value of *RTR* register.

See also

setRTR()

Definition at line **1250** of file **w5100.h**.

Referenced by **wizchip_gettimeout()**.

#define setRCR (rcr) WIZCHIP_WRITE(_RCR_, rcr)

Set *RCR* register.

Parameters
(uint8_t)rcr Value to set *RCR* register.

See also
getRCR()

Definition at line **1259** of file **w5100.h**.

Referenced by **wizchip_settimeout()**.

#define getRCR ( ) WIZCHIP_READ(_RCR_)

Get *RCR* register.

Returns
uint8_t. Value of *RCR* register.

See also
setRCR()

Definition at line **1268** of file **w5100.h**.

Referenced by **wizchip_gettimeout()**.

#define setRMSR (rmsr) WIZCHIP_WRITE(RMSR)

Get **RMSR** register.

See also
getRMSR()

Definition at line **1276** of file **w5100.h**.

#define getRMSR ( ) WIZCHIP_READ()

Get **RMSR** register.

Returns
uint8_t. Value of **RMSR** register.

See also
setRMSR()

Definition at line **1285** of file **w5100.h**.

#define setTMSR (rmsr) WIZCHIP_WRITE(TMSR)

Get **TMSR** register.

See also
getTMSR()

Definition at line **1293** of file **w5100.h**.

#define getPATR ( ) (((uint16_t)WIZCHIP_READ(PATR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR,1)))

Get **TMSR** register.

Returns
uint8_t. Value of **TMSR** register.

See also
setTMSR()

Get **PATR** register

Returns
uint16_t. Value to set **PATR** register

Definition at line **1309** of file **w5100.h**.

#define getPPPALGO ( ) WIZCHIP_READ(PPPALGO)

Get **PPPALGO** register.

Returns
uint8_t. Value to set **PPPALGO** register

Definition at line **1317** of file **w5100.h**.

#define setPTIMER (ptimer) WIZCHIP_WRITE(PTIMER, ptimer)

Set **PTIMER** register.

Parameters
(uint8_t)ptimer Value to set **PTIMER** register.

See also
getPTIMER()

Definition at line **1327** of file **w5100.h**.

#define getPTIMER ( ) WIZCHIP_READ(PTIMER)

Get **PTIMER** register.

Returns
uint8_t. Value of **PTIMER** register.

See also
setPTIMER()

Definition at line **1336** of file **w5100.h**.

#define setPMAGIC (pmagic) WIZCHIP_WRITE(PMAGIC, pmagic)

Set **PMAGIC** register.

Parameters

(uint8_t)pmagic Value to set **PMAGIC** register.

See also

getPMAGIC()

Definition at line **1345** of file **w5100.h**.

#define getPMAGIC ( ) WIZCHIP_READ(PMAGIC)

Get **PMAGIC** register.

Returns

uint8_t. Value of **PMAGIC** register.

See also

setPMAGIC()

Definition at line **1354** of file **w5100.h**.

Socket APIs

Socket register access functions

These are functions to access **socket registers**. More...

Macros

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ir)
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) **WIZCHIP_READ**(**Sn_IR**(sn))
Get **Sn_IR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define **setSn_PORT**(sn, port)
Set **Sn_PORT** register. More...

#define **getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn)
Get **Sn_PORT** register. More...

#define **setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s
dhar, 6)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(s

	dhar, 6) Get **Sn_DHAR** register. More...
#define	**setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn), 4) Set **Sn_DIPR** register. More...
#define	**getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn), 4) Get **Sn_DIPR** register. More...
#define	**setSn_DPORT**(sn, dport) Set **Sn_DPORT** register. More...
#define	**getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT** << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1) Get **Sn_DPORT** register. More...
#define	**setSn_MSSR**(sn, mss) Set **Sn_MSSR** register. More...
#define	**getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn) Get **Sn_MSSR** register. More...
#define	**setSn_PROTO**(sn, proto) **WIZCHIP_WRITE**(**Sn_TOS**(sn), to Set **Sn_PROTO** register. More...
#define	**getSn_PROTO**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn)) Get **Sn_PROTO** register. More...
#define	**setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos) Set **Sn_TOS** register. More...
#define	**getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn)) Get **Sn_TOS** register. More...

#define **setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl)
Set **Sn_TTL** register. More...

#define **getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn))
Get **Sn_TTL** register. More...

#define **setSn_RXMEM_SIZE**(sn, rxmemsize) **WIZCHIP_WRITE**(**RM**
(**WIZCHIP_READ**(**RMSR**) & ~(0x03 << (2*sn))) | (rxmemsize <
(2*sn)))
Set **Sn_RXMEM_SIZE** register. More...

#define **getSn_RXMEM_SIZE**(sn) ((**WIZCHIP_READ**(**RMSR**) & (0x0
(2*sn))) >> (2*sn))
Get **Sn_RXMEM_SIZE** register. More...

#define **setSn_TXMEM_SIZE**(sn, txmemsize) **WIZCHIP_WRITE**(**TM**
(**WIZCHIP_READ**(**TMSR**) & ~(0x03 << (2*sn))) | (txmemsize <
(2*sn)))
Set **Sn_TXMEM_SIZE** register. More...

#define **getSn_TXMEM_SIZE**(sn) ((**WIZCHIP_READ**(**TMSR**) & (0x03
(2*sn))) >> (2*sn))
Get **Sn_TXMEM_SIZE** register. More...

#define **getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sn),1))
Get **Sn_TX_RD** register. More...

#define **setSn_TX_WR**(sn, txwr)
Set **Sn_TX_WR** register. More...

#define **getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1)
Get **Sn_TX_WR** register. More...

#define **setSn_RX_RD**(sn, rxrd)
Set **Sn_RX_RD** register. More...

#define **getSn_RX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1))
Get **Sn_RX_RD** register. More...

#define **setSn_RX_WR**(sn, rxwr)
Set **Sn_RX_WR** register. More...

#define **getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1)
Get **Sn_RX_WR** register. More...

#define **setSn_FRAG**(sn, frag)
Set **Sn_FRAG** register. More...

#define **getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn)
Get **Sn_FRAG** register. More...

#define **getSn_RxMAX**(sn) ((uint16_t)(1 << **getSn_RXMEM_SIZE**(sr
<< 10)
Get the max RX buffer size of socket sn. More...

#define **getSn_TxMAX**(sn) ((uint16_t)(1 << **getSn_TXMEM_SIZE**(sn
10)
Get the max TX buffer size of socket sn. More...

#define **getSn_RxMASK**(sn) (**getSn_RxMAX**(sn) - 1)
Get the mask of socket sn RX buffer. More...

#define **getSn_TxMASK**(sn) (**getSn_TxMAX**(sn) - 1)
Get the mask of socket sn TX buffer. More...

Functions

uint16_t **getSn_TX_FSR** (uint8_t sn)
Get **Sn_TX_FSR** register. More...

uint16_t **getSn_RX_RSR** (uint8_t sn)
Get **Sn_RX_RSR** register. More...

uint32_t **getSn_RxBASE** (uint8_t sn)
Get the base address of socket sn RX buffer. More...

uint32_t **getSn_TxBASE** (uint8_t sn)
Get the base address of socket sn TX buffer. More...

Detailed Description

These are functions to access **socket registers**.

Macro Definition Documentation

#define setSn_MR (sn,
mr
) WIZCHIP_WRITE(Sn_MR(sn),mr)

Set **Sn_MR** register.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** expect **bit 4**.

mr Value to set **Sn_MR**

See also

getSn_MR()

Definition at line **1367** of file **w5100.h**.

Referenced by **close()**, and **socket()**.

#define getSn_MR (sn) WIZCHIP_READ(Sn_MR(sn))

Get **Sn_MR** register.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** expect **bit 4**.

Returns

Value of **Sn_MR**.

See also

setSn_MR()

Definition at line **1377** of file **w5100.h**.

Referenced by **close()**, **getsockopt()**, **recv()**, **recvfrom()**, and **sendto()**.

```
#define setSn_CR (  sn,
                    cr
                 )    WIZCHIP_WRITE(Sn_CR(sn), cr)
```

Set **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)cr Value to set **Sn_CR**

See also

getSn_CR()

Definition at line **1387** of file **w5100.h**.

Referenced by **close()**, **connect()**, **disconnect()**, **listen()**, **recv()**, **recvfrom()**, **send()**, **sendto()**, **setsockopt()**, and **socket()**.

```
#define getSn_CR (  sn )    WIZCHIP_READ(Sn_CR(sn))
```

Get **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_CR**.

See also

setSn_CR()

Definition at line **1397** of file **w5100.h**.

Referenced by **close()**, **connect()**, **disconnect()**, **listen()**, **recv()**, **recvfrom()**, **send()**, **sendto()**, **setsockopt()**, and **socket()**.

#define setSn_IR (sn, ir) WIZCHIP_WRITE(Sn_IR(sn), ir)

Set **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)ir Value to set **Sn_IR**

See also

getSn_IR()

Definition at line **1407** of file **w5100.h**.

Referenced by **close()**, **connect()**, **ctlsocket()**, **send()**, **sendto()**, and **setsockopt()**.

#define getSn_IR (sn) WIZCHIP_READ(Sn_IR(sn))

Get **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_IR**.

See also

setSn_IR()

Definition at line **1417** of file **w5100.h**.

Referenced by **connect()**, **ctlsocket()**, **disconnect()**, **send()**, **sendto()**, and **setsockopt()**.

#define getSn_SR (sn) WIZCHIP_READ(Sn_SR(sn))

Get **Sn_SR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_SR**.

Definition at line **1426** of file **w5100.h**.

Referenced by **close()**, **connect()**, **disconnect()**, **getsockopt()**, **listen()**, **recv()**, **recvfrom()**, **send()**, **sendto()**, and **socket()**.

#define setSn_PORT (sn, port)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    (uint8_t) port); \
    }
```

Set **Sn_PORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint16_t)port Value to set **Sn_PORT**.

See also
getSn_PORT()

Definition at line **1436** of file **w5100.h**.

Referenced by **socket()**.

#define getSn_PORT (sn) (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PO

Get **Sn_PORT** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns
uint16_t. Value of **Sn_PORT**.

See also
setSn_PORT()

Definition at line **1448** of file **w5100.h**.

#define setSn_DHAR (sn, dhar) WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6)

Set **Sn_DHAR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~**

_WIZCHIP_SOCK_NUM_.

(uint8_t*)dhar Pointer variable to set socket n destination hardware address. It should be allocated 6 bytes.

See also

getSn_DHAR()

Definition at line **1458** of file **w5100.h**.

```
#define
getSn_DHAR        (  sn,
                     dhar
                       WIZCHIP_READ_BUF(Sn_DHAR(sn),
                   )  dhar, 6)
```

Get **Sn_DHAR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~** _WIZCHIP_SOCK_NUM_.

(uint8_t*)dhar Pointer variable to get socket n destination hardware address. It should be allocated 6 bytes.

See also

setSn_DHAR()

Definition at line **1468** of file **w5100.h**.

```
#define
setSn_DIPR        (  sn,
                     dipr
                       WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
                   )  4)
```

Set **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t*)dipr	Pointer variable to set socket n destination IP address. It should be allocated 4 bytes.

See also

getSn_DIPR()

Definition at line **1478** of file **w5100.h**.

Referenced by **connect()**, **sendto()**, and **setsockopt()**.

```
#define
getSn_DIPR          (  sn,
                       dipr
                         WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
                    )  4)
```

Get **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t*)dipr	Pointer variable to get socket n destination IP address. It should be allocated 4 bytes.

See also

SetSn_DIPR()

Definition at line **1488** of file **w5100.h**.

Referenced by **getsockopt()**.

#define setSn_DPORT (sn, dport)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    (uint8_t)  dport); \
    }
```

Set **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)dport Value to set **Sn_DPORT**

See also

getSn_DPORT()

Definition at line **1498** of file **w5100.h**.

Referenced by **connect()**, **sendto()**, and **setsockopt()**.

#define getSn_DPORT (sn) (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_I

Get **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_DPORT**.

See also
setSn_DPORT()

Definition at line **1510** of file **w5100.h**.

Referenced by **getsockopt()**.

#define setSn_MSSR (sn, mss)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    (uint8_t) mss); \
    }
```

Set **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)mss Value to set **Sn_MSSR**

See also
setSn_MSSR()

Definition at line **1520** of file **w5100.h**.

Referenced by **setsockopt()**.

#define getSn_MSSR (sn) (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_M

Get **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_MSSR**.

See also

setSn_MSSR()

Definition at line **1532** of file **w5100.h**.

Referenced by **getsockopt()**.

```
#define setSn_PROTO (  sn,
                       proto
                    )    WIZCHIP_WRITE(Sn_TOS(sn), tos)
```

Set **Sn_PROTO** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)proto Value to set **Sn_PROTO**

See also

getSn_PROTO()

Definition at line **1542** of file **w5100.h**.

```
#define getSn_PROTO (  sn )    WIZCHIP_READ(Sn_TOS(sn))
```

Get **Sn_PROTO** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_PROTO**.

See also

setSn_PROTO()

Definition at line **1552** of file **w5100.h**.

#define setSn_TOS (sn, tos) WIZCHIP_WRITE(Sn_TOS(sn), tos)

Set **Sn_TOS** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)tos Value to set **Sn_TOS**

See also

getSn_TOS()

Definition at line **1562** of file **w5100.h**.

Referenced by **setsockopt()**.

#define getSn_TOS (sn) WIZCHIP_READ(Sn_TOS(sn))

Get **Sn_TOS** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

Returns
uint8_t. Value of Sn_TOS.

See also
setSn_TOS()

Definition at line **1572** of file **w5100.h**.

Referenced by **getsockopt()**.

#define setSn_TTL (sn, ttl) WIZCHIP_WRITE(Sn_TTL(sn), ttl)

Set **Sn_TTL** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .
(uint8_t)ttl Value to set **Sn_TTL**

See also
getSn_TTL()

Definition at line **1582** of file **w5100.h**.

Referenced by **setsockopt()**.

#define getSn_TTL (sn) WIZCHIP_READ(Sn_TTL(sn))

Get **Sn_TTL** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

Returns

uint8_t. Value of **Sn_TTL**.

See also
setSn_TTL()

Definition at line **1592** of file **w5100.h**.

Referenced by **getsockopt()**.

#define setSn_RXMEM_SIZE (sn, rxmemsize) WIZCHIP_WRITE(RMSR, (WIZCHIP_READ(RMSR) & ~(0x03 << (2*sn))) | (rxmemsize << (2*sn)))

Set **Sn_RXMEM_SIZE** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .
(uint8_t)rxmemsize	Value to set **Sn_RXMEM_SIZE**

See also
getSn_RXMEM_SIZE()

Definition at line **1602** of file **w5100.h**.

#define getSn_RXMEM_SIZE (sn) ((WIZCHIP_READ(RMSR) & (0x03 << (2*sn))) >> (2*sn))

Get **Sn_RXMEM_SIZE** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns
uint8_t. Value of Sn_RXMEM.

See also
setSn_RXMEM_SIZE()

Definition at line **1613** of file **w5100.h**.

```
#define
setSn_TXMEM_SIZE (  sn,
                    txmemsize
                      WIZCHIP_WRITE(TMSR,
                    (WIZCHIP_READ(TMSR) & ~(0x03 <<
                  )  (2*sn))) | (txmemsize << (2*sn)))
```

Set **Sn_TXMEM_SIZE** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)txmemsize	Value to set **Sn_TXMEM_SIZE**

See also
getSn_TXMEM_SIZE()

Definition at line **1624** of file **w5100.h**.

```
#define                          ((WIZCHIP_READ(TMSR) & (0x03
getSn_TXMEM_SIZE (  sn ) << (2*sn))) >> (2*sn))
```

Get **Sn_TXMEM_SIZE** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_TXMEM_SIZE**.

See also
setSn_TXMEM_SIZE()

Definition at line **1635** of file **w5100.h**.

#define getSn_TX_RD (sn) (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T

Get **Sn_TX_RD** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns
uint16_t. Value of **Sn_TX_RD**.

Definition at line **1653** of file **w5100.h**.

Referenced by **send()**.

#define setSn_TX_WR (sn, txwr)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_TX_WR(sn),
    (uint8_t)(txwr>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    (uint8_t) txwr); \
        }
```

Set **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)txwr Value to set **Sn_TX_WR**

See also

GetSn_TX_WR()

Definition at line **1663** of file **w5100.h**.

#define getSn_TX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn))<<8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T

Get **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_TX_WR**.

See also

setSn_TX_WR()

Definition at line **1675** of file **w5100.h**.

#define setSn_RX_RD (sn, rxrd)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_RX_RD(sn),
    (uint8_t)(rxrd>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    (uint8_t) rxrd); \
```

```
    }
```

Set **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)rxrd Value to set **Sn_RX_RD**

See also

getSn_RX_RD()

Definition at line **1693** of file **w5100.h**.

#define getSn_RX_RD (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_R

Get **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N** uint16_t. Value of **Sn_RX_RD**.

See also

setSn_RX_RD()

Definition at line **1705** of file **w5100.h**.

#define setSn_RX_WR (sn, rxwr)

Value:

```
{ \
                    WIZCHIP_WRITE(Sn_RX_WR(sn),
    (uint8_t)(rxwr>>8)); \
```

```
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1
    (uint8_t) rxwr); \
    }
```

Set **Sn_RX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)rxwr Value to set **Sn_RX_WR**

See also

getSn_RX_WR()

Definition at line **1715** of file **w5100.h**.

#define getSn_RX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_

Get **Sn_RX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_RX_WR**.

Definition at line **1727** of file **w5100.h**.

#define setSn_FRAG (sn, frag)

Value:

```
{ \
                    WIZCHIP_WRITE(Sn_FRAG(sn),
    (uint8_t)(frag >>8)); \
```

```
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)
    (uint8_t) frag); \
    }
```

Set **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)frag Value to set **Sn_FRAG**

See also

getSn_FRAG()

Definition at line **1737** of file **w5100.h**.

#define getSn_FRAG (sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FR

Get **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_FRAG**.

See also

setSn_FRAG()

Definition at line **1749** of file **w5100.h**.

#define getSn_RxMAX (sn) ((uint16_t)(1 << getSn_RXMEM_SIZE(sn)) << 10)

Get the max RX buffer size of socket sn.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Max buffer size

Definition at line **1758** of file **w5100.h**.

Referenced by **ctlsocket()**, and **recv()**.

#define getSn_TxMAX (sn) ((uint16_t)(1 << getSn_TXMEM_SIZE(sn)) << 10)

Get the max TX buffer size of socket sn.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Max buffer size

Definition at line **1768** of file **w5100.h**.

Referenced by **close()**, **ctlsocket()**, **recv()**, **send()**, and **sendto()**.

#define getSn_RxMASK (sn) (getSn_RxMAX(sn) - 1)

Get the mask of socket sn RX buffer.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Mask value

Definition at line **1777** of file **w5100.h**.

#define getSn_TxMASK (sn) (getSn_TxMAX(sn) - 1)

Get the mask of socket sn TX buffer.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Mask value

Definition at line **1786** of file **w5100.h**.

Function Documentation

uint16_t getSn_TX_FSR (uint8_t sn)

Get **Sn_TX_FSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of **Sn_TX_FSR**.

Referenced by **close()**, **getsockopt()**, **recv()**, **send()**, and **sendto()**.

uint16_t getSn_RX_RSR (uint8_t sn)

Get **Sn_RX_RSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of **Sn_RX_RSR**.

Referenced by **getsockopt()**, **recv()**, and **recvfrom()**.

uint32_t getSn_RxBASE (uint8_t sn)

Get the base address of socket sn RX buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of Socket n RX buffer base address.

uint32_t getSn_TxBASE (uint8_t sn)

Get the base address of socket sn TX buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of Socket n TX buffer base address.

Socket APIs

WIZCHIP register

W5100

WIZCHIP register defines register group of **W5100** . More...

Modules

Common register
Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

Socket register
Socket register group
Socket register configures and control SOCKETn which is necessary to data communication.

Detailed Description

WIZCHIP register defines register group of **W5100** .

- **Common register** : Common register group W5100
- **Socket register** : SOCKET n register group W5100

Socket APIs

Common register

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc. More...

Macros

#define **MR** (**_WIZCHIP_IO_BASE_** + (0x0000))
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...

#define **GAR** (**_W5100_IO_BASE_** + (0x0001))
Gateway IP Register address(R/W) More...

#define **SUBR** (**_W5100_IO_BASE_** + (0x0005))
Subnet mask Register address(R/W) More...

#define **SHAR** (**_W5100_IO_BASE_** + (0x0009))
Source MAC Register address(R/W) More...

#define **SIPR** (**_W5100_IO_BASE_** + (0x000F))
Source IP Register address(R/W) More...

#define **IR** (**_W5100_IO_BASE_** + (0x0015))
Interrupt Register(R/W) More...

#define **_IMR_** (**_W5100_IO_BASE_** + (0x0016))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (**_W5100_IO_BASE_** + (0x0017))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5100_IO_BASE_** + (0x0019))
Retry count register(R/W) More...

#define **PATR** (**_W5100_IO_BASE_** + (0x001C))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PTIMER** (**_W5100_IO_BASE_** + (0x0028))

PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PMAGIC** (**_W5100_IO_BASE_** + (0x0029))
PPP LCP Magic number register in PPPoE mode(R) More...

Detailed Description

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

See also

MR : Mode register.

GAR, **SUBR**, **SHAR**, **SIPR**

IR, **Sn_IR**, *IMR* : Interrupt.

RTR, *RCR* : Data retransmission.

PTIMER, **PMAGIC** : PPPoE.

Macro Definition Documentation

#define MR (_WIZCHIP_IO_BASE_ + (0x0000))

Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc.

Each bit of **MR** defined as follows.

7	6	5	4	3	2	1	0
RST	Reserved	WOL	PB	PPPoE	Reserved	AI	IND

- **MR_RST** : Reset
- **MR_PB** : Ping block
- **MR_PPPOE** : PPPoE mode
- **MR_AI** : Address Auto-Increment in Indirect Bus Interface
- **MR_IND** : Indirect Bus Interface mode

Definition at line **202** of file **w5100.h**.

#define GAR (_W5100_IO_BASE_ + (0x0001))

Gateway IP Register address(R/W)

GAR configures the default gateway address.

Definition at line **212** of file **w5100.h**.

#define SUBR (_W5100_IO_BASE_ + (0x0005))

Subnet mask Register address(R/W)

SUBR configures the subnet mask address.

Definition at line **219** of file **w5100.h**.

#define SHAR (_W5100_IO_BASE_ + (0x0009))

Source MAC Register address(R/W)

SHAR configures the source hardware address.

Definition at line **226** of file **w5100.h**.

#define SIPR (_W5100_IO_BASE_ + (0x000F))

Source IP Register address(R/W)

SIPR configures the source IP address.

Definition at line **233** of file **w5100.h**.

#define IR (_W5100_IO_BASE_ + (0x0015))

Interrupt Register(R/W)

IR indicates the interrupt status. Each bit of **IR** will be still until the bit wi
to by the host. If **IR** is not equal to x00 INTn PIN is asserted to low until

Each bit of **IR** defined as follows.

7	6	5	4	3	2	1
CONFLICT	UNREACH	PPPoE	Reserved	S3_INT	S2_INT	S1_I

- **IR_CONFLICT** : IP conflict
- **IR_UNREACH** : Destination unreachable
- **IR_PPPoE** : PPPoE connection close
- **IR_SOCK(3)** : SOCKET 3 Interrupt
- **IR_SOCK(2)** : SOCKET 2 Interrupt
- **IR_SOCK(1)** : SOCKET 1 Interrupt

- **IR_SOCK(0)** : SOCKET 0 Interrupt

Definition at line **256** of file **w5100.h**.

#define _IMR_ (_W5100_IO_BASE_ + (0x0016))

Socket Interrupt Mask Register(R/W)

Each bit of *IMR* corresponds to each bit of **IR**. When a bit of *IMR* is and the corresponding bit of **IR** is set, Interrupt will be issued.

Definition at line **264** of file **w5100.h**.

#define _RTR_ (_W5100_IO_BASE_ + (0x0017))

Timeout register address(1 is 100us)(R/W)

RTR configures the retransmission timeout period. The unit of timeout period is 100us and the default of *RTR* is x07D0or 000 And so the default timeout period is 200ms(100us X 2000). During the time configured by *RTR*, W5100 waits for the peer response to the packet that is transmitted by **Sn_CR** (CONNECT, DISCON, CLOSE, SEND, SEND_MAC, SEND_KEEP command). If the peer does not respond within the *RTR* time, W5100 retransmits the packet or issues timeout.

Definition at line **274** of file **w5100.h**.

#define _RCR_ (_W5100_IO_BASE_ + (0x0019))

Retry count register(R/W)

RCR configures the number of time of retransmission. When retransmission occurs as many as ref *RCR*+1 Timeout interrupt is issued (**Sn_IR_TIMEOUT** = '1').

Definition at line **282** of file **w5100.h**.

#define PATR (_W5100_IO_BASE_ + (0x001C))

PPP LCP Request Timer register in PPPoE mode(R)

PATR notifies authentication method that has been agreed at the connection with PPPoE Server. W5100 supports two types of Authentication method - PAP and CHAP.

Definition at line **293** of file **w5100.h**.

#define PTIMER (_W5100_IO_BASE_ + (0x0028))

PPP LCP Request Timer register in PPPoE mode(R)

PTIMER configures the time for sending LCP echo request. The unit of time is 25ms.

Definition at line **301** of file **w5100.h**.

#define PMAGIC (_W5100_IO_BASE_ + (0x0029))

PPP LCP Magic number register in PPPoE mode(R)

PMAGIC configures the 4bytes magic number to be used in LCP negotiation.

Definition at line **308** of file **w5100.h**.

Socket APIs

Socket register

Socket register group
Socket register configures and control SOCKETn which is necessary to data communication. More...

Macros

#define	**Sn_MR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0000)) socket Mode register(R/W) More...
#define	**Sn_CR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0001)) Socket command register(R/W) More...
#define	**Sn_IR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0002)) Socket interrupt register(R) More...
#define	**Sn_SR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0003)) Socket status register(R) More...
#define	**Sn_PORT**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0004)) source port register(R/W) More...
#define	**Sn_DHAR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0006)) Peer MAC register address(R/W) More...
#define	**Sn_DIPR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x000C)) Peer IP register address(R/W) More...
#define	**Sn_DPORT**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0010)) Peer port register address(R/W) More...
#define	**Sn_MSSR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0012))

	Maximum Segment Size(Sn_MSSR0) register address(R/W) More...
#define	**Sn_PROTO**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0014)) IP Protocol(PROTO) Register(R/W) More...
#define	**Sn_TOS**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + 0x0015) IP Type of Service(TOS) Register(R/W) More...
#define	**Sn_TTL**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0016)) IP Time to live(TTL) Register(R/W) More...
#define	**Sn_TX_FSR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0020)) Transmit free memory size register(R) More...
#define	**Sn_TX_RD**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0022)) Transmit memory read pointer register address(R) More...
#define	**Sn_TX_WR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0024)) Transmit memory write pointer register address(R/W) More...
#define	**Sn_RX_RSR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0026)) Received data size register(R) More...
#define	**Sn_RX_RD**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0028)) Read point of Receive memory(R/W) More...
#define	**Sn_RX_WR**(sn) (**_W5100_IO_BASE_** +

WIZCHIP_SREG_BLOCK(sn) + (0x002A))
Write point of Receive memory(R) More...

Detailed Description

Socket register group
Socket register configures and control SOCKETn which is necessary to data communication.

See also

Sn_MR, **Sn_CR**, **Sn_IR** : SOCKETn Control

Sn_SR, **Sn_PORT**, **Sn_DHAR**, **Sn_DIPR**, **Sn_DPORT** : SOCKETn Information

Sn_MSSR, **Sn_TOS**, **Sn_TTL**, **Sn_FRAG** : Internet protocol.

Sn_RXMEM_SIZE, **Sn_TXMEM_SIZE**, **Sn_TX_FSR**, **Sn_TX_RD**, **Sn_TX_WR**, **Sn_RX_RSR**, **Sn_RX_RD**, **Sn_RX_WR** : Data communication

Macro Definition Documentation

#define Sn_MR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLO (0x0000))

socket Mode register(R/W)

Sn_MR configures the option or protocol type of Socket n.

Each bit of **Sn_MR** defined as the following.

7	6	5	4	3	2	1
MULTI	MF	ND/MC	Reserved	Protocol[3]	Protocol[2]	Protocol[1]

- **Sn_MR_MULTI** : Support UDP Multicasting
- **Sn_MR_MF** : Support MACRAW
- **Sn_MR_ND** : No Delayed Ack(TCP) flag
- **Sn_MR_MC** : IGMP version used **in UDP mulitcasting**
- **Protocol**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	1	0	0	MACRAW

- **In case of Socket 0**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	1	0	0	MACRAW
0	1	0	1	PPPoE

  - **Sn_MR_MACRAW** : MAC LAYER RAW SOCK

Sn_MR_UDP : UDP

- **Sn_MR_TCP** : TCP

- **Sn_MR_CLOSE** : Unused socket

  Note
  MACRAW mode should be only used in Socket 0.

Definition at line **352** of file **w5100.h**.

#define Sn_CR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001))

Socket command register(R/W)

This is used to set the command for Socket n such as OPEN, CLOSE, CONNECT, LISTEN, SEND, and RECEIVE.
After W5100 accepts the command, the **Sn_CR** register is automatically cleared to 0x00. Even though **Sn_CR** is cleared to 0x00, the command is still being processed.
To check whether the command is completed or not, please check the **Sn_IR** or **Sn_SR**.

- **Sn_CR_OPEN** : Initialize or open socket.
- **Sn_CR_LISTEN** : Wait connection request in TCP mode(**Server mode**)
- **Sn_CR_CONNECT** : Send connection request in TCP mode(**Client mode**)
- **Sn_CR_DISCON** : Send closing request in TCP mode.
- **Sn_CR_CLOSE** : Close socket.
- **Sn_CR_SEND** : Update TX buffer pointer and send data.
- **Sn_CR_SEND_MAC** : Send data with MAC address, so without ARP process.
- **Sn_CR_SEND_KEEP** : Send keep alive message.
- **Sn_CR_RECV** : Update RX buffer pointer and receive data.
- **In case of S0_MR(P3:P0) = S0_MR_PPPoE**

Value	Symbol	Description
0x23	PCON	PPPoE connection begins by transmitting PPPoE discovery packet
0x24	PDISCON	Closes PPPoE connection
0x25	PCR	In each phase, it transmits REQ message.
0x26	PCN	In each phase, it transmits NAK message.

0x27	PCJ	In each phase, it transmits REJECT message.

Definition at line **380** of file **w5100.h**.

#define Sn_IR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOC + (0x0002))

Socket interrupt register(R)

Sn_IR indicates the status of Socket Interrupt such as establishment, termination, receiving data, timeout).
When an interrupt occurs and the corresponding bit **IR_SOCK(N)** in *IM* set, **IR_SOCK(N)** in **IR** becomes '1'.
In order to clear the **Sn_IR** bit, the host should write the bit to

7	6	5	4	3	2	1	
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	

- **Sn_IR_PRECV** : **PPP Receive Interrupt**
- **Sn_IR_PFAIL** : **PPP Fail Interrupt**
- **Sn_IR_PNEXT** : **PPP Next Phase Interrupt**
- **Sn_IR_SENDOK** : **SEND_OK Interrupt**
- **Sn_IR_TIMEOUT** : **TIMEOUT Interrupt**
- **Sn_IR_RECV** : **RECV Interrupt**
- **Sn_IR_DISCON** : **DISCON Interrupt**
- **Sn_IR_CON** : **CON Interrupt**

Definition at line **401** of file **w5100.h**.

#define Sn_SR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0003))

Socket status register(R)

Sn_SR indicates the status of Socket n.
The status of Socket n is changed by **Sn_CR** or some special

control packet as SYN, FIN packet in TCP.

Normal status

- **SOCK_CLOSED** : Closed
- **SOCK_INIT** : Initiate state
- **SOCK_LISTEN** : Listen state
- **SOCK_ESTABLISHED** : Success to connect
- **SOCK_CLOSE_WAIT** : Closing state
- **SOCK_UDP** : UDP socket
- **SOCK_MACRAW** : MAC raw mode socket

Temporary status during changing the status of Socket n.

- **SOCK_SYNSENT** : This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
- **SOCK_SYNRECV** : It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
- **SOCK_FIN_WAIT** : Connection state
- **SOCK_CLOSING** : Closing state
- **SOCK_TIME_WAIT** : Closing state
- **SOCK_LAST_ACK** : Closing state

Definition at line **424** of file **w5100.h**.

```
#define Sn_PORT ( sn ) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004))
```

source port register(R/W)

Sn_PORT configures the source port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. It should be set before OPEN command is ordered.

Definition at line **432** of file **w5100.h**.

```
#define Sn_DHAR ( sn ) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006))
```

Peer MAC register address(R/W)

Sn_DHAR configures the destination hardware address of Socket n when using SEND_MAC command in UDP mode or it indicates that it is acquired in ARP-process by CONNECT/SEND command.

Definition at line **440** of file **w5100.h**.

```
#define                 (_W5100_IO_BASE_ +
Sn_DIPR     (  sn ) WIZCHIP_SREG_BLOCK(sn) + (0x000C))
```

Peer IP register address(R/W)

Sn_DIPR configures or indicates the destination IP address of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP client mode, it configures an IP address of TCP server before CONNECT command. In TCP server mode, it indicates an IP address of TCP client after successfully establishing connection. In UDP mode, it configures an IP address of peer to be received the UDP packet by SEND or SEND_MAC command.

Definition at line **450** of file **w5100.h**.

```
#define                 (_W5100_IO_BASE_ +
Sn_DPORT    (  sn ) WIZCHIP_SREG_BLOCK(sn) + (0x0010))
```

Peer port register address(R/W)

Sn_DPORT configures or indicates the destination port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP clientmode, it configures the listen port number of TCP server before CONNECT command. In TCP Servermode, it indicates the port number of TCP client after successfully establishing connection. In UDP mode, it configures the port number of peer to be transmitted the UDP packet by SEND/SEND_MAC command.

Definition at line **460** of file **w5100.h**.

#define Sn_MSSR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0012))

Maximum Segment Size(Sn_MSSR0) register address(R/W)

Sn_MSSR configures or indicates the MTU(Maximum Transfer Unit) of Socket n.

Definition at line **467** of file **w5100.h**.

#define Sn_PROTO (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0014))

IP Protocol(PROTO) Register(R/W)

Sn_PROTO that sets the protocol number field of the IP header at the IP layer. It is valid only in IPRAW mode, and ignored in other modes.

Definition at line **475** of file **w5100.h**.

#define Sn_TOS (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + 0x0015)

IP Type of Service(TOS) Register(R/W)

Sn_TOS configures the TOS(Type Of Service field in IP Header) of Socket n. It is set before OPEN command.

Definition at line **483** of file **w5100.h**.

#define Sn_TTL (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0016))

IP Time to live(TTL) Register(R/W)

Sn_TTL configures the TTL(Time To Live field in IP header) of Socket n. It is set before OPEN command.

Definition at line **491** of file **w5100.h**.

```
#define Sn_TX_FSR ( sn ) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020))
```

Transmit free memory size register(R)

Sn_TX_FSR indicates the free size of Socket n TX Buffer Block. It is initialized to the configured size by **Sn_TXMEM_SIZE**. Data bigger than **Sn_TX_FSR** should not be saved in the Socket n TX Buffer because the bigger data overwrites the previous saved data not yet sent. Therefore, check before saving the data to the Socket n TX Buffer, and if data is equal or smaller than its checked size, transmit the data with SEND/SEND_MAC command after saving the data in Socket n TX buffer. But, if data is bigger than its checked size, transmit the data after dividing into the checked size and saving in the Socket n TX buffer.

Definition at line **510** of file **w5100.h**.

```
#define Sn_TX_RD ( sn ) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022))
```

Transmit memory read pointer register address(R)

Sn_TX_RD is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001), it is re-initialized while connecting with TCP. After its initialization, it is auto-increased by SEND command. SEND command transmits the saved data from the current **Sn_TX_RD** to the **Sn_TX_WR** in the Socket n TX Buffer. After transmitting the saved data, the SEND command increases the **Sn_TX_RD** as same as the **Sn_TX_WR**. If its increment value

exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **522** of file **w5100.h**.

#define Sn_TX_WR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024))

Transmit memory write pointer register address(R/W)

Sn_TX_WR is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001), it is re-initialized while connecting with TCP.
It should be read or be updated like as follows.

1. Read the starting address for saving the transmitting data.
2. Save the transmitting data from the starting address of Socket n TX buffer.
3. After saving the transmitting data, update **Sn_TX_WR** to the increased value as many as transmitting data size. If the increment value exceeds the maximum value 0xFFFF(greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.
4. Transmit the saved data in Socket n TX Buffer by using SEND/SEND command

Definition at line **536** of file **w5100.h**.

#define Sn_RX_RSR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0026))

Received data size register(R)

Sn_RX_RSR indicates the data size received and saved in Socket n RX Buffer. **Sn_RX_RSR** does not exceed the **Sn_RXMEM_SIZE** and is calculated as the difference between Socket n RX Write

Pointer (**Sn_RX_WR**)and Socket n RX Read Pointer (**Sn_RX_RD**)

Definition at line **545** of file **w5100.h**.

#define Sn_RX_RD (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0028))

Read point of Receive memory(R/W)

Sn_RX_RD is initialized by OPEN command. Make sure to be read or updated as follows.

1. Read the starting save address of the received data.
2. Read data from the starting address of Socket n RX Buffer.
3. After reading the received data, Update **Sn_RX_RD** to the increased value as many as the reading size. If the increment value exceeds the maximum value 0xFFFF, that is, is greater than 0x10000 and the carry bit occurs, update with the lower 16bits value ignored the carry bit.
4. Order RECV command is for notifying the updated **Sn_RX_RD** to W5100.

Definition at line **558** of file **w5100.h**.

#define Sn_RX_WR (sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002A))

Write point of Receive memory(R)

Sn_RX_WR is initialized by OPEN command and it is auto-increased by the data reception. If the increased value exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **567** of file **w5100.h**.

Socket APIs

W5200

WHIZCHIP register defines and I/O functions of **W5200**. More...

Modules

WIZCHIP I/O functions
This supports the basic I/O functions for **WIZCHIP register**.

WIZCHIP register
WIZCHIP register defines register group of **W5200** .

Detailed Description

WHIZCHIP register defines and I/O functions of **W5200**.

- **WIZCHIP register** : **Common register** and **Socket register**
- **WIZCHIP I/O functions** : **Basic I/O function**, **Common register access functions** and **Socket register**

Socket APIs

WIZCHIP I/O functions

W5200

This supports the basic I/O functions for **WIZCHIP register**. More...

Modules

Basic I/O function
These are basic input/output functions to read values from register or write values to register.

Common register access functions
These are functions to access **common registers**.

Socket register access functions
These are functions to access **socket registers**.

Detailed Description

This supports the basic I/O functions for **WIZCHIP register**.

- **Basic I/O function**
  WIZCHIP_READ(), **WIZCHIP_WRITE()**, **WIZCHIP_READ_BUF()**, **WIZCHIP_WRITE_BUF()**

- **Common register access functions**
  1. **Mode**
     getMR(), **setMR()**
  2. **Interrupt**
     getIR(), **setIR()**, **getIMR()**, **setIMR()**, **getIR2()**, **setIR2()**, **getIMR2()**, **setIMR2()**, **getINTLEVEL()**, **setINTLEVEL()**
  3. **Network Information**
     getSHAR(), **setSHAR()**, **getGAR()**, **setGAR()**, **getSUBR()**, **setSUBR()**, **getSIPR()**, **setSIPR()**
  4. **Retransmission**
     getRCR(), **setRCR()**, **getRTR()**, **setRTR()**
  5. **PPPoE**
     getPTIMER(), **setPTIMER()**, **getPMAGIC()**, **getPMAGIC()**
  6. **etc**.
     getPHYSTATUS(), **getVERSIONR()**

- **Socket register access functions**
  1. **SOCKET control**
     getSn_MR(), **setSn_MR()**, **getSn_CR()**, **setSn_CR()**, **getSn_IMR()**, **setSn_IMR()**, **getSn_IR()**, **setSn_IR()**
  2. **SOCKET information**
     getSn_SR(), **getSn_DHAR()**, **setSn_DHAR()**, **getSn_PORT()**, **setSn_PORT()**, **getSn_DIPR()**, **setSn_DIPR()**, **getSn_DPORT()**, **setSn_DPORT()**
     getSn_MSSR(), **setSn_MSSR()**
  3. **SOCKET communication**
     getSn_RXMEM_SIZE(), **setSn_RXMEM_SIZE()**, **getSn_TXMEM_SIZE()**, **setSn_TXMEM_SIZE()**
     getSn_TX_RD(), **getSn_TX_WR()**, **setSn_TX_WR()**

getSn_RX_RD(), **setSn_RX_RD()**, **getSn_RX_WR()**
getSn_TX_FSR(), **getSn_RX_RSR()**

4. **IP header field**
   getSn_FRAG(), **setSn_FRAG()**, **getSn_TOS()**, **setSn_TOS()**
   getSn_TTL(), **setSn_TTL()**

Socket APIs

Basic I/O function

These are basic input/output functions to read values from register or write values to register. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...
void	**wiz_recv_ignore** (uint8_t sn, uint16_t len) It discard the received data in RX memory. More...

Detailed Description

These are basic input/output functions to read values from register or write values to register.

Function Documentation

uint8_t WIZCHIP_READ (uint32_t AddrSel)

It reads 1 byte value from a register.

Parameters

AddrSel Register address

Returns

The value of register

void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb)

It writes 1 byte value to a register.

Parameters

AddrSel Register address

wb Write data

Returns

void

void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t * pBuf, uint16_t len)

It reads sequence data from registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to read data
len	Data length

```
void WIZCHIP_WRITE_BUF ( uint32_t  AddrSel,
                         uint8_t *  pBuf,
                         uint16_t  len
                       )
```

It writes sequence data to registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to write data
len	Data length

```
void wiz_send_data ( uint8_t    sn,
                     uint8_t *  wizdata,
                     uint16_t  len
                   )
```

It copies data to internal TX memory.

This function reads the Tx write pointer register and after that, it copies the *wizdata(pointer buffer)* of the length of *len(variable)* bytes to internal TX memory and updates the Tx write pointer register. This function is being called by **send()** and **sendto()** function also.

Parameters

sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
wizdata	Pointer buffer to write data
len	Data length

See also

wiz_recv_data()

---

void wiz_recv_data (uint8_t sn,
uint8_t * wizdata,
uint16_t len
)

It copies data to your buffer from internal RX memory.

This function read the Rx read pointer register and after that, it copies the received data from internal RX memory to *wizdata(pointer variable)* of the length of *len(variable)* bytes. This function is being called by **recv()** also.

Parameters

sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
wizdata	Pointer buffer to read data
len	Data length

See also

wiz_send_data()

---

void wiz_recv_ignore (uint8_t sn,
uint16_t len
)

It discard the received data in RX memory.

It discards the data of the length of *len(variable)* bytes in internal RX memory.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

len	Data length

Socket APIs

Common register access functions

These are functions to access **common registers**. More...

Macros

#define **setMR**(mr) (*((uint8_t*)**MR**) = mr)
Set Mode Register. More...

#define **getMR**() (*(uint8_t*)**MR**)
Get **MR**. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set **GAR**. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get **GAR**. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**, subr,4)
Set **SUBR**. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get **SUBR**. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set **SHAR**. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get **SHAR**. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set **SIPR**. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get **SIPR**. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xA0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xA0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**IMR2**, imr & 0xA0)
Set **IMR2** register. More...

#define **getIMR**() (**WIZCHIP_READ**(**IMR2**) & 0xA0)
Get **IMR2** register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(**_RTR_**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **getPATR**() (((uint16_t)**WIZCHIP_READ**(**PATR**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PATR**,1)))
Get **PATR** register. More...

#define **getPPPALGO**() **WIZCHIP_READ**(**PPPALGO**)
Get **PPPALGO** register. More...

#define **getVERSIONR**() **WIZCHIP_READ**(**VERSIONR**)
Get **VERSIONR** register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)

Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)
Get **PMAGIC** register. More...

#define **setINTLEVEL**(intlevel)
Set **INTLEVEL** register. More...

#define **getINTLEVEL**() (((uint16_t)**WIZCHIP_READ**(**INTLEVEL**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**INTLEVEL**,1)))
Get **INTLEVEL** register. More...

#define **setIR2**(ir2) **WIZCHIP_WRITE**(**IR2**, ir2)
Set **IR2** register. More...

#define **getIR2**() **WIZCHIP_READ**(**IR2**)
Get **IR2** register. More...

#define **getPHYSTATUS**() **WIZCHIP_READ**(**PHYSTATUS**)
Get **PHYSTATUS** register. More...

#define **setIMR2**(imr2) **WIZCHIP_WRITE**(**_IMR_**, imr2)
Set *IMR* register. More...

#define **getIMR2**() **WIZCHIP_READ**(**_IMR_**)
Get *IMR* register. More...

Detailed Description

These are functions to access **common registers**.

Macro Definition Documentation

#define setMR (mr) (*((uint8_t*)MR) = mr)

Set Mode Register.

Parameters

(uint8_t)mr The value to be set.

See also

getMR()

Definition at line **1238** of file **w5200.h**.

#define getMR ( ) (*(uint8_t*)MR)

Get **MR**.

Returns

uint8_t. The value of Mode register.

See also

setMR()

Definition at line **1250** of file **w5200.h**.

#define setGAR (gar) WIZCHIP_WRITE_BUF(GAR,gar,4)

Set **GAR**.

Parameters

(uint8_t*)gar Pointer variable to set gateway IP address. It should be allocated 4 bytes.

See also
getGAR()

Definition at line **1259** of file **w5200.h**.

#define getGAR (gar) WIZCHIP_READ_BUF(GAR,gar,4)

Get **GAR**.

Parameters

(uint8_t*)gar Pointer variable to get gateway IP address. It should be allocated 4 bytes.

See also
setGAR()

Definition at line **1268** of file **w5200.h**.

#define setSUBR (subr) WIZCHIP_WRITE_BUF(SUBR, subr,4)

Set **SUBR**.

Parameters

(uint8_t*)subr Pointer variable to set subnet mask address. It should be allocated 4 bytes.

Note
If subr is null pointer, set the backup subnet to SUBR.
If subr is 0.0.0.0, back up SUBR and clear it.
Otherwize, set subr to SUBR

See also
getSUBR()

Definition at line **1280** of file **w5200.h**.

#define getSUBR (subr) WIZCHIP_READ_BUF(SUBR, subr, 4)

Get **SUBR**.

Parameters

(uint8_t*)subr Pointer variable to get subnet mask address. It should be allocated 4 bytes.

See also

setSUBR()

Definition at line **1289** of file **w5200.h**.

#define setSHAR (shar) WIZCHIP_WRITE_BUF(SHAR, shar, 6)

Set **SHAR**.

Parameters

(uint8_t*)shar Pointer variable to set local MAC address. It should be allocated 6 bytes.

See also

getSHAR()

Definition at line **1298** of file **w5200.h**.

#define getSHAR (shar) WIZCHIP_READ_BUF(SHAR, shar, 6)

Get **SHAR**.

Parameters

(uint8_t*)shar Pointer variable to get local MAC address. It should be allocated 6 bytes.

See also

setSHAR()

Definition at line **1307** of file **w5200.h**.

#define setSIPR (sipr) WIZCHIP_WRITE_BUF(SIPR, sipr, 4)

Set **SIPR**.

Parameters

(uint8_t*)sipr Pointer variable to set local IP address. It should be allocated 4 bytes.

See also

getSIPR()

Definition at line **1316** of file **w5200.h**.

#define getSIPR (sipr) WIZCHIP_READ_BUF(SIPR, sipr, 4)

Get **SIPR**.

Parameters

(uint8_t*)sipr Pointer variable to get local IP address. It should be allocated 4 bytes.

See also

setSIPR()

Definition at line **1325** of file **w5200.h**.

#define setIR (ir) WIZCHIP_WRITE(IR, (ir & 0xA0))

Set **IR** register.

Parameters

(uint8_t)ir Value to set **IR** register.

See also

getIR()

Definition at line **1334** of file **w5200.h**.

#define getIR ( ) (WIZCHIP_READ(IR) & 0xA0)

Get **IR** register.

Returns
uint8_t. Value of **IR** register.

See also
setIR()

Definition at line **1342** of file **w5200.h**.

#define setIMR (imr) WIZCHIP_WRITE(IMR2, imr & 0xA0)

Set **IMR2** register.

Parameters
(uint8_t)imr Value to set **IMR2** register.

See also
getIMR()

Definition at line **1356** of file **w5200.h**.

#define getIMR ( ) (WIZCHIP_READ(IMR2) & 0xA0)

Get **IMR2** register.

Returns
uint8_t. Value of **IMR2** register.

See also

setIMR()

Definition at line **1370** of file **w5200.h**.

#define setRTR (rtr)

Value:

```
{\
                WIZCHIP_WRITE(_RTR_,   (uint8_t)
    (rtr >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),
    (uint8_t) rtr); \
    }
```

Set *RTR* register.

Parameters

(uint16_t)rtr Value to set *RTR* register.

See also

getRTR()

Definition at line **1379** of file **w5200.h**.

#define getRTR ( ) (((uint16_t)WIZCHIP_READ(_RTR_) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))

Get *RTR* register.

Returns

uint16_t. Value of *RTR* register.

See also

setRTR()

Definition at line **1390** of file **w5200.h**.

#define setRCR (rcr) WIZCHIP_WRITE(_RCR_, rcr)

Set *RCR* register.

Parameters
(uint8_t)rcr Value to set *RCR* register.

See also
getRCR()

Definition at line **1399** of file **w5200.h**.

#define getRCR ( ) WIZCHIP_READ(_RCR_)

Get *RCR* register.

Returns
uint8_t. Value of *RCR* register.

See also
setRCR()

Definition at line **1408** of file **w5200.h**.

#define getPATR ( ) (((uint16_t)WIZCHIP_READ(PATR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR,1)))

Get **PATR** register.

Returns
uint16_t. Value to set **PATR** register

Definition at line **1416** of file **w5200.h**.

#define getPPPALGO ( ) WIZCHIP_READ(PPPALGO)

Get **PPPALGO** register.

Returns
uint8_t. Value to set **PPPALGO** register

Definition at line **1424** of file **w5200.h**.

#define getVERSIONR ( ) WIZCHIP_READ(VERSIONR)

Get **VERSIONR** register.

Returns
uint8_t. Value to set **VERSIONR** register

Definition at line **1433** of file **w5200.h**.

#define setPTIMER (ptimer) WIZCHIP_WRITE(PTIMER, ptimer)

Set **PTIMER** register.

Parameters
(uint8_t)ptimer Value to set **PTIMER** register.

See also
getPTIMER()

Definition at line **1442** of file **w5200.h**.

#define getPTIMER ( ) WIZCHIP_READ(PTIMER)

Get **PTIMER** register.

Returns
uint8_t. Value of **PTIMER** register.

See also
setPTIMER()

Definition at line **1451** of file **w5200.h**.

#define setPMAGIC (pmagic) WIZCHIP_WRITE(PMAGIC, pmagic)

Set **PMAGIC** register.

Parameters
(uint8_t)pmagic Value to set **PMAGIC** register.

See also
getPMAGIC()

Definition at line **1460** of file **w5200.h**.

#define getPMAGIC ( ) WIZCHIP_READ(PMAGIC)

Get **PMAGIC** register.

Returns
uint8_t. Value of **PMAGIC** register.

See also
setPMAGIC()

Definition at line **1469** of file **w5200.h**.

#define setINTLEVEL (intlevel)

Value:

```
{\
                  WIZCHIP_WRITE(INTLEVEL,
    (uint8_t)(intlevel >> 8)); \
```

```
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL,1),
    (uint8_t) intlevel); \
    }
```

Set **INTLEVEL** register.

Parameters

(uint16_t)intlevel Value to set **INTLEVEL** register.

See also

getINTLEVEL()

Definition at line **1478** of file **w5200.h**.

Referenced by **ctlwizchip()**.

#define getINTLEVEL ( ) (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVE

Get **INTLEVEL** register.

Returns

uint16_t. Value of **INTLEVEL** register.

See also

setINTLEVEL()

Definition at line **1488** of file **w5200.h**.

Referenced by **ctlwizchip()**.

#define setIR2 (ir2) WIZCHIP_WRITE(IR2, ir2)

Set **IR2** register.

Parameters

(uint8_t)ir2 Value to set **IR2** register.

See also
getIR2()

Definition at line **1497** of file **w5200.h**.

#define getIR2 ( ) WIZCHIP_READ(IR2)

Get **IR2** register.

Returns
uint8_t. Value of **IR2** register.

See also
setIR2()

Definition at line **1507** of file **w5200.h**.

#define getPHYSTATUS ( ) WIZCHIP_READ(PHYSTATUS)

Get **PHYSTATUS** register.

Returns
uint8_t. Value to set **PHYSTATUS** register.

Definition at line **1516** of file **w5200.h**.

Referenced by **wizphy_getphylink()**, and **wizphy_getphypmode()**.

#define setIMR2 (imr2) WIZCHIP_WRITE(_IMR_, imr2)

Set *IMR* register.

Parameters
(uint8_t)imr2 Value to set **IMR2** register.

See also
getIMR2()

Note
If possible, Don't use this function. Instead, Use **setSIMR()** for compatible with ioLibrary.

Definition at line **1531** of file **w5200.h**.

#define getIMR2 ( ) WIZCHIP_READ(_IMR_)

Get *IMR* register.

Returns
uint8_t. Value of **IMR2** register.

See also
setIMR2()

Definition at line **1546** of file **w5200.h**.

Socket APIs

Socket register access functions

These are functions to access **socket registers**. More...

Macros

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ir)
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) **WIZCHIP_READ**(**Sn_IR**(sn))
Get **Sn_IR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), imr)
Set **Sn_IMR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), imr)
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) **WIZCHIP_READ**(**Sn_IMR**(sn))
Get **Sn_IMR** register. More...

#define **getSn_IMR**(sn) **WIZCHIP_READ**(**Sn_IMR**(sn))
Get **Sn_IMR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define	**setSn_PORT**(sn, port) Set **Sn_PORT** register. More...
#define	**getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn) Get **Sn_PORT** register. More...
#define	**setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s dhar, 6) Set **Sn_DHAR** register. More...
#define	**getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(s dhar, 6) Get **Sn_DHAR** register. More...
#define	**setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn), 4) Set **Sn_DIPR** register. More...
#define	**getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn), Get **Sn_DIPR** register. More...
#define	**setSn_DPORT**(sn, dport) Set **Sn_DPORT** register. More...
#define	**getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT** << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1) Get **Sn_DPORT** register. More...
#define	**setSn_MSSR**(sn, mss) Set **Sn_MSSR** register. More...
#define	**getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn) Get **Sn_MSSR** register. More...

#define **setSn_PROTO**(sn, proto) **WIZCHIP_WRITE**(**Sn_PROTO**(sn) proto)
Set **Sn_PROTO** register. More...

#define **getSn_PROTO**(sn) **WIZCHIP_READ**(**Sn_PROTO**(sn))
Get **Sn_PROTO** register. More...

#define **setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos)
Set **Sn_TOS** register. More...

#define **getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn))
Get **Sn_TOS** register. More...

#define **setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl)
Set **Sn_TTL** register. More...

#define **getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn))
Get **Sn_TTL** register. More...

#define **setSn_RXMEM_SIZE**(sn, rxmemsize) **WIZCHIP_WRITE**(**Sn_RXMEM_SIZE**(sn),rxmem
Set **Sn_RXMEM_SIZE** register. More...

#define **getSn_RXMEM_SIZE**(sn) **WIZCHIP_READ**(**Sn_RXMEM_SI**
Get **Sn_RXMEM_SIZE** register. More...

#define **setSn_TXMEM_SIZE**(sn, txmemsize) **WIZCHIP_WRITE**(**Sn_TXMEM_SIZE**(sn), txmem
Set **Sn_TXMEM_SIZE** register. More...

#define **getSn_TXMEM_SIZE**(sn) **WIZCHIP_READ**(**Sn_TXMEM_SIZ**
Get **Sn_TXMEM_SIZE** register. More...

#define **getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sr
Get **Sn_TX_RD** register. More...

#define **setSn_TX_WR**(sn, txwr)
Set **Sn_TX_WR** register. More...

#define **getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1)
Get **Sn_TX_WR** register. More...

#define **setSn_RX_RD**(sn, rxrd)
Set **Sn_RX_RD** register. More...

#define **getSn_RX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1)
Get **Sn_RX_RD** register. More...

#define **setSn_RX_WR**(sn, rxwr)
Set **Sn_RX_WR** register. More...

#define **getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1)
Get **Sn_RX_WR** register. More...

#define **setSn_FRAG**(sn, frag)
Set **Sn_FRAG** register. More...

#define **getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn)
Get **Sn_FRAG** register. More...

#define **getSn_RxMAX**(sn) ((uint16_t)**getSn_RXMEM_SIZE**(sn) << 1
Get the max RX buffer size of socket sn. More...

#define **getSn_TxMAX**(sn) ((uint16_t)**getSn_TXMEM_SIZE**(sn) << 1
Get the max TX buffer size of socket sn. More...

#define **getSn_RxMASK**(sn) ((uint16_t)**getSn_RxMAX**(sn) - 1)
Get the mask of socket sn RX buffer. More...

#define **getSn_TxMASK**(sn) ((uint16_t)**getSn_TxMAX**(sn) - 1)
Get the mask of socket sn TX buffer. More...

Functions

uint16_t **getSn_TX_FSR** (uint8_t sn)
Get **Sn_TX_FSR** register. More...

uint16_t **getSn_RX_RSR** (uint8_t sn)
Get **Sn_RX_RSR** register. More...

uint16_t **getSn_RxBASE** (uint8_t sn)
Get the base address of socket sn RX buffer. More...

uint16_t **getSn_TxBASE** (uint8_t sn)
Get the base address of socket sn TX buffer. More...

Detailed Description

These are functions to access **socket registers**.

Macro Definition Documentation

#define setSn_MR (sn,
mr
) WIZCHIP_WRITE(Sn_MR(sn),mr)

Set **Sn_MR** register.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** expect **bit 4**.

mr Value to set **Sn_MR**

See also

getSn_MR()

Definition at line **1559** of file **w5200.h**.

#define getSn_MR (sn) WIZCHIP_READ(Sn_MR(sn))

Get **Sn_MR** register.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** expect **bit 4**.

Returns

Value of **Sn_MR**.

See also

setSn_MR()

Definition at line **1569** of file **w5200.h**.

```
#define setSn_CR ( sn,
                   cr
                 )   WIZCHIP_WRITE(Sn_CR(sn), cr)
```

Set **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)cr Value to set **Sn_CR**

See also

getSn_CR()

Definition at line **1579** of file **w5200.h**.

```
#define getSn_CR ( sn )   WIZCHIP_READ(Sn_CR(sn))
```

Get **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_CR**.

See also

setSn_CR()

Definition at line **1589** of file **w5200.h**.

```
#define setSn_IR ( sn,
                   ir
                 )   WIZCHIP_WRITE(Sn_IR(sn), ir)
```

Set **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)ir Value to set **Sn_IR**

See also

getSn_IR()

Definition at line **1599** of file **w5200.h**.

#define getSn_IR (sn) WIZCHIP_READ(Sn_IR(sn))

Get **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_IR**.

See also

setSn_IR()

Definition at line **1609** of file **w5200.h**.

#define setSn_IMR (sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), imr)

Set **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)imr Value to set **Sn_IMR**

See also
getSn_IMR()

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)imr Value to set **Sn_IMR**

See also
getSn_IMR()

Definition at line **1963** of file **w5200.h**.

Referenced by **ctlsocket()**.

#define setSn_IMR (sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), imr)

Set **Sn_IMR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)imr Value to set **Sn_IMR**

See also
getSn_IMR()

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)imr Value to set **Sn_IMR**

See also
getSn_IMR()

Definition at line **1963** of file **w5200.h**.

#define getSn_IMR (sn) WIZCHIP_READ(Sn_IMR(sn))

Get **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_IMR**.

See also

setSn_IMR()

Definition at line **1973** of file **w5200.h**.

Referenced by **ctlsocket()**.

#define getSn_IMR (sn) WIZCHIP_READ(Sn_IMR(sn))

Get **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_IMR**.

See also

setSn_IMR()

Definition at line **1973** of file **w5200.h**.

#define getSn_SR (sn) WIZCHIP_READ(Sn_SR(sn))

Get **Sn_SR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_SR**.

Definition at line **1638** of file **w5200.h**.

#define setSn_PORT (sn, port)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    (uint8_t) port); \
    }
```

Set **Sn_PORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)port Value to set **Sn_PORT**.

See also

getSn_PORT()

Definition at line **1648** of file **w5200.h**.

#define getSn_PORT (sn) (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PO

Get **Sn_PORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_PORT**.

See also

setSn_PORT()

Definition at line **1660** of file **w5200.h**.

#define setSn_DHAR (sn, dhar) WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6)

Set **Sn_DHAR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t*)dhar	Pointer variable to set socket n destination hardware address. It should be allocated 6 bytes.

See also

getSn_DHAR()

Definition at line **1670** of file **w5200.h**.

#define

```
getSn_DHAR ( sn,
             dhar
               WIZCHIP_READ_BUF(Sn_DHAR(sn),
           ) dhar, 6)
```

Get **Sn_DHAR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t*)dhar	Pointer variable to get socket n destination hardware address. It should be allocated 6 bytes.

See also

setSn_DHAR()

Definition at line **1680** of file **w5200.h**.

```
#define
setSn_DIPR ( sn,
             dipr
               WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
           ) 4)
```

Set **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t*)dipr	Pointer variable to set socket n destination IP address. It should be allocated 4 bytes.

See also

getSn_DIPR()

Definition at line **1690** of file **w5200.h**.

#define
getSn_DIPR (sn,
dipr
WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
) 4)

Get **Sn_DIPR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t*)dipr Pointer variable to get socket n destination IP address. It should be allocated 4 bytes.

See also

SetSn_DIPR()

Definition at line **1700** of file **w5200.h**.

#define setSn_DPORT (sn,
dport
)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    (uint8_t)  dport); \
    }
```

Set **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~**

_WIZCHIP_SOCK_NUM_.

(uint16_t)dport Value to set **Sn_DPORT**

See also

getSn_DPORT()

Definition at line **1710** of file **w5200.h**.

**#define getSn_DPORT (sn) (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn))
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_I**

Get **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_DPORT**.

See also

setSn_DPORT()

Definition at line **1722** of file **w5200.h**.

#define setSn_MSSR (sn, mss)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    (uint8_t) mss); \
    }
```

Set **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)mss Value to set **Sn_MSSR**

See also

setSn_MSSR()

Definition at line **1732** of file **w5200.h**.

#define getSn_MSSR (sn) (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_M

Get **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_MSSR**.

See also

setSn_MSSR()

Definition at line **1744** of file **w5200.h**.

```
#define setSn_PROTO ( sn, proto ) WIZCHIP_WRITE(Sn_PROTO(sn), proto)
```

Set **Sn_PROTO** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)proto Value to set **Sn_PROTO**

See also
getSn_PROTO()

Definition at line **1759** of file **w5200.h**.

#define getSn_PROTO (sn) WIZCHIP_READ(Sn_PROTO(sn))

Get **Sn_PROTO** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns
uint8_t. Value of **Sn_PROTO**.

See also
setSn_PROTO()

Definition at line **1774** of file **w5200.h**.

#define setSn_TOS (sn, tos) WIZCHIP_WRITE(Sn_TOS(sn), tos)

Set **Sn_TOS** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)tos Value to set **Sn_TOS**

See also

getSn_TOS()

Definition at line **1784** of file **w5200.h**.

```
#define getSn_TOS ( sn )  WIZCHIP_READ(Sn_TOS(sn))
```

Get **Sn_TOS** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

Returns

uint8_t. Value of Sn_TOS.

See also

setSn_TOS()

Definition at line **1794** of file **w5200.h**.

```
#define setSn_TTL ( sn,
                    ttl
                  )  WIZCHIP_WRITE(Sn_TTL(sn), ttl)
```

Set **Sn_TTL** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

(uint8_t)ttl Value to set **Sn_TTL**

See also

getSn_TTL()

Definition at line **1804** of file **w5200.h**.

#define getSn_TTL (sn) WIZCHIP_READ(Sn_TTL(sn))

Get **Sn_TTL** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

Returns

uint8_t. Value of **Sn_TTL**.

See also

setSn_TTL()

Definition at line **1814** of file **w5200.h**.

#define setSn_RXMEM_SIZE (sn, rxmemsize) WIZCHIP_WRITE(Sn_RXMEM_SIZE(sn),rx

Set **Sn_RXMEM_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

(uint8_t)rxmemsize Value to set **Sn_RXMEM_SIZE**

See also

getSn_RXMEM_SIZE()

Definition at line **1824** of file **w5200.h**.

#define getSn_RXMEM_SIZE (sn) WIZCHIP_READ(Sn_RXMEM_SIZE(sr

Get **Sn_RXMEM_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of Sn_RXMEM.

See also

setSn_RXMEM_SIZE()

Definition at line **1836** of file **w5200.h**.

```
#define
setSn_TXMEM_SIZE  (  sn,
                     txmemsize
                       WIZCHIP_WRITE(Sn_TXMEM_SIZE(sn),
                   )  txmemsize)
```

Set **Sn_TXMEM_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)txmemsize Value to set **Sn_TXMEM_SIZE**

See also

getSn_TXMEM_SIZE()

Definition at line **1848** of file **w5200.h**.

```
#define
getSn_TXMEM_SIZE (  sn )   WIZCHIP_READ(Sn_TXMEM_SIZE(sn
```

Get **Sn_TXMEM_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint8_t. Value of **Sn_TXMEM_SIZE**.

See also

setSn_TXMEM_SIZE()

Definition at line **1860** of file **w5200.h**.

#define getSn_TX_RD (sn) (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T

Get **Sn_TX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_TX_RD**.

Definition at line **1879** of file **w5200.h**.

#define setSn_TX_WR (sn, txwr)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_TX_WR(sn),
    (uint8_t)(txwr>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    (uint8_t) txwr); \
```

```
        }
```

Set **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)txwr Value to set **Sn_TX_WR**

See also

GetSn_TX_WR()

Definition at line **1889** of file **w5200.h**.

#define getSn_TX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T

Get **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_TX_WR**.

See also

setSn_TX_WR()

Definition at line **1901** of file **w5200.h**.

#define setSn_RX_RD (sn, rxrd)

Value:

```
{ \
                    WIZCHIP_WRITE(Sn_RX_RD(sn),
```

```
    (uint8_t)(rxrd>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    (uint8_t) rxrd); \
    }
```

Set **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)rxrd Value to set **Sn_RX_RD**

See also

getSn_RX_RD()

Definition at line **1919** of file **w5200.h**.

#define getSn_RX_RD (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_R

Get **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_RX_RD**.

See also

setSn_RX_RD()

Definition at line **1931** of file **w5200.h**.

#define setSn_RX_WR (sn, rxwr)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_RX_WR(sn),
    (uint8_t)(rxwr>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1
    (uint8_t) rxwr); \
    }
```

Set **Sn_RX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)rxwr Value to set **Sn_RX_WR**

See also

getSn_RX_WR()

Definition at line **1941** of file **w5200.h**.

#define getSn_RX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_

Get **Sn_RX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_RX_WR**.

Definition at line **1953** of file **w5200.h**.

#define setSn_FRAG (sn, frag

)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_FRAG(sn),
    (uint8_t)(frag >>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)
    (uint8_t) frag); \
    }
```

Set **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint16_t)frag Value to set **Sn_FRAG**

See also

getSn_FRAG()

Definition at line **1983** of file **w5200.h**.

#define getSn_FRAG (sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FR

Get **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_N**

Returns

uint16_t. Value of **Sn_FRAG**.

See also

setSn_FRAG()

Definition at line **1995** of file **w5200.h**.

#define getSn_RxMAX (sn) ((uint16_t)getSn_RXMEM_SIZE(sn) << 10)

Get the max RX buffer size of socket sn.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Max buffer size

Definition at line **2004** of file **w5200.h**.

#define getSn_TxMAX (sn) ((uint16_t)getSn_TXMEM_SIZE(sn) << 10)

Get the max TX buffer size of socket sn.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Max buffer size

Definition at line **2013** of file **w5200.h**.

#define getSn_RxMASK (sn) ((uint16_t)getSn_RxMAX(sn) - 1)

Get the mask of socket sn RX buffer.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns
uint16_t. Mask value

Definition at line **2022** of file **w5200.h**.

#define getSn_TxMASK (sn) ((uint16_t)getSn_TxMAX(sn) - 1)

Get the mask of socket sn TX buffer.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns
uint16_t. Mask value

Definition at line **2031** of file **w5200.h**.

Function Documentation

uint16_t getSn_TX_FSR (uint8_t sn)

Get **Sn_TX_FSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of **Sn_TX_FSR**.

uint16_t getSn_RX_RSR (uint8_t sn)

Get **Sn_RX_RSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of **Sn_RX_RSR**.

uint16_t getSn_RxBASE (uint8_t sn)

Get the base address of socket sn RX buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of Socket n RX buffer base address.

uint16_t getSn_TxBASE (uint8_t sn)

Get the base address of socket sn TX buffer.

Parameters

sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns

uint16_t. Value of Socket n TX buffer base address.

Socket APIs

WIZCHIP register

W5200

WIZCHIP register defines register group of **W5200** . More...

Modules

Common register
Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

Socket register
Socket register group
Socket register configures and control SOCKETn which is necessary to data communication.

Detailed Description

WIZCHIP register defines register group of **W5200** .

- **Common register** : Common register group w5200
- **Socket register** : SOCKET n register group w5200

Socket APIs

Common register

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc. More...

Macros

#define **MR** (_W5200_IO_BASE_ + (0x0000))
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...

#define **GAR** (_W5200_IO_BASE_ + (0x0001))
Gateway IP Register address(R/W) More...

#define **SUBR** (_W5200_IO_BASE_ + (0x0005))
Subnet mask Register address(R/W) More...

#define **SHAR** (_W5200_IO_BASE_ + (0x0009))
Source MAC Register address(R/W) More...

#define **SIPR** (_W5200_IO_BASE_ + (0x000F))
Source IP Register address(R/W) More...

#define **IR** (_W5200_IO_BASE_ + (0x0015))
Interrupt Register(R/W) More...

#define **_IMR_** (_W5200_IO_BASE_ + (0x0016))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (_W5200_IO_BASE_ + (0x0017))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (_W5200_IO_BASE_ + (0x0019))
Retry count register(R/W) More...

#define **PATR** (_W5200_IO_BASE_ + (0x001C))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PPPALGO** (_W5200_IO_BASE_ + (0x001E))

PPP LCP Request Timer register in PPPoE mode(R) More...

#define **VERSIONR** (_W5200_IO_BASE_ + (0x001F))
chip version register address(R) More...

#define **PTIMER** (_W5200_IO_BASE_ + (0x0028))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PMAGIC** (_W5200_IO_BASE_ + (0x0029))
PPP LCP Magic number register in PPPoE mode(R) More...

#define **INTLEVEL** (_W5200_IO_BASE_ + (0x0030))
Set Interrupt low level timer register address(R/W) More...

#define **IR2** (_W5200_IO_BASE_ + (0x0034))
Socket Interrupt Register(R/W) More...

#define **PHYSTATUS** (_W5200_IO_BASE_ + (0x0035))
PHYSTATUS(R/W) More...

#define **IMR2** (_W5200_IO_BASE_ + (0x0036))
Interrupt mask register(R/W) More...

Detailed Description

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

See also

MR : Mode register.

GAR, **SUBR**, **SHAR**, **SIPR**

INTLEVEL, **IR**, *IMR*, **IR2**, **IMR2** : Interrupt.

RTR, *RCR* : Data retransmission.

PTIMER, **PMAGIC** : PPPoE.

PHYSTATUS, **VERSIONR** : etc.

Macro Definition Documentation

#define MR (_W5200_IO_BASE_ + (0x0000))

Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc.

Each bit of **MR** defined as follows.

7	6	5	4	3	2	1	0
RST	Reserved	WOL	PB	PPPoE	Reserved	AI	IND

- **MR_RST** : Reset
- **MR_WOL** : Wake on LAN
- **MR_PB** : Ping block
- **MR_PPPOE** : PPPoE mode
- **MR_AI** : Address Auto-Increment in Indirect Bus Interface
- **MR_IND** : Indirect Bus Interface mode

Definition at line **207** of file **w5200.h**.

#define GAR (_W5200_IO_BASE_ + (0x0001))

Gateway IP Register address(R/W)

GAR configures the default gateway address.

Definition at line **215** of file **w5200.h**.

#define SUBR (_W5200_IO_BASE_ + (0x0005))

Subnet mask Register address(R/W)

SUBR configures the subnet mask address.

Definition at line **222** of file **w5200.h**.

#define SHAR (_W5200_IO_BASE_ + (0x0009))

Source MAC Register address(R/W)

SHAR configures the source hardware address.

Definition at line **229** of file **w5200.h**.

#define SIPR (_W5200_IO_BASE_ + (0x000F))

Source IP Register address(R/W)

SIPR configures the source IP address.

Definition at line **236** of file **w5200.h**.

#define IR (_W5200_IO_BASE_ + (0x0015))

Interrupt Register(R/W)

IR indicates the interrupt status. Each bit of **IR** will be still until the bit wi
the host. If **IR** is not equal to x00 INTn PIN is asserted to low until it is x

Each bit of **IR** defined as follows.

7	6	5	4	3	2	1
CONFLICT	Reserved	PPPoE	Reserved	Reserved	Reserved	Re

- **IR_CONFLICT** : IP conflict
- **IR_PPPoE** : PPPoE connection close

Definition at line **254** of file **w5200.h**.

#define _IMR_ (_W5200_IO_BASE_ + (0x0016))

Socket Interrupt Mask Register(R/W)

Each bit of *IMR* corresponds to each bit of **IR2**. When a bit of *IMR* is and the corresponding bit of **IR2** is Interrupt will be issued. In other words, if a bit of *IMR*, an interrupt will be not issued even if the corresponding bit of **IR2** is set

Note

This Register is same operated as **SMIR of W5100, W5300 and W5550.**
So, setSIMR() set a value to *IMR* for integrating with ioLibrary

Definition at line **265** of file **w5200.h**.

#define _RTR_ (_W5200_IO_BASE_ + (0x0017))

Timeout register address(1 is 100us)(R/W)

RTR configures the retransmission timeout period. The unit of timeout period is 100us and the default of *RTR* is x07D0. And so the default timeout period is 200ms(100us X 2000). During the time configured by *RTR*, W5200 waits for the peer response to the packet that is transmitted by **Sn_CR** (CONNECT, DISCON, CLOSE, SEND, SEND_MAC, SEND_KEEP command). If the peer does not respond within the *RTR* time, W5200 retransmits the packet or issues timeout.

Definition at line **275** of file **w5200.h**.

#define _RCR_ (_W5200_IO_BASE_ + (0x0019))

Retry count register(R/W)

RCR configures the number of time of retransmission. When retransmission occurs as many as ref *RCR*+1 Timeout interrupt is issued (**Sn_IR_TIMEOUT** = '1').

Definition at line **283** of file **w5200.h**.

#define PATR (_W5200_IO_BASE_ + (0x001C))

PPP LCP Request Timer register in PPPoE mode(R)

PATR notifies authentication method that has been agreed at the connection with PPPoE Server. W5200 supports two types of Authentication method - PAP and CHAP.

Definition at line **294** of file **w5200.h**.

#define PPPALGO (_W5200_IO_BASE_ + (0x001E))

PPP LCP Request Timer register in PPPoE mode(R)

PPPALGO notifies authentication algorithm in PPPoE mode. For detailed information, please refer to PPPoE application note.

Definition at line **302** of file **w5200.h**.

#define VERSIONR (_W5200_IO_BASE_ + (0x001F))

chip version register address(R)

VERSIONR always indicates the W5200 version as **0x03**.

Definition at line **309** of file **w5200.h**.

#define PTIMER (_W5200_IO_BASE_ + (0x0028))

PPP LCP Request Timer register in PPPoE mode(R)

PTIMER configures the time for sending LCP echo request. The unit of time is 25ms.

Definition at line **325** of file **w5200.h**.

#define PMAGIC (_W5200_IO_BASE_ + (0x0029))

PPP LCP Magic number register in PPPoE mode(R)

PMAGIC configures the 4bytes magic number to be used in LCP negotiation.

Definition at line **332** of file **w5200.h**.

#define INTLEVEL (_W5200_IO_BASE_ + (0x0030))

Set Interrupt low level timer register address(R/W)

INTLEVEL configures the Interrupt Assert Time.

Definition at line **346** of file **w5200.h**.

#define IR2 (_W5200_IO_BASE_ + (0x0034))

Socket Interrupt Register(R/W)

IR2 indicates the interrupt status of Socket.
Each bit of **IR2** be still until **Sn_IR** is cleared by the host.
If **Sn_IR** is not equal to x00 the n-th bit of **IR2** is and INTn PIN is asserted until **IR2** is x00

Definition at line **357** of file **w5200.h**.

#define PHYSTATUS (_W5200_IO_BASE_ + (0x0035))

PHYSTATUS(R/W)

PHYSTATUS is the Register to indicate W5200 status of PHY.

7	6	5	4	3	2
Reserved	Reserved	LINK	POWERSAVE	POWERDOWN	Reserv

- **PHYSTATUS_LINK** : Link Status Register[Read Only]
- **PHYSTATUS_POWERSAVE** : Power save mode of PHY[R/W]
- **PHYSTATUS_POWERDOWN** : Power down mode of PHY[R/W]

Definition at line **371** of file **w5200.h**.

#define IMR2 (_W5200_IO_BASE_ + (0x0036))

Interrupt mask register(R/W)

IMR2 is used to mask interrupts. Each bit of *IMR* corresponds to each b
a bit of **IMR2** is and the corresponding bit of **IR** is an interrupt will be iss
words, if a bit of **IMR2** is an interrupt will not be issued even if the corre
IR is

Each bit of **IMR2** defined as the following.

7	6	5	4	3	2	1
IM_IR7	Reserved	IM_IR5	Reserved	Reserved	Reserved	Reser

- **IM_IR7** : IP Conflict Interrupt Mask
- **IM_IR5** : PPPoE Close Interrupt Mask

  Note

  This Register is same operated as **_IMR_ of W5100, W5300 a**
  So, setIMR() set a value to IMR2 for integrating with ioLibr

Definition at line **389** of file **w5200.h**.

Socket APIs

Socket register

Socket register group
Socket register configures and control SOCKETn which is necessary to data communication. More...

Macros

#define **Sn_MR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0000))
socket Mode register(R/W) More...

#define **Sn_CR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0001))
Socket command register(R/W) More...

#define **Sn_IR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0002))
Socket interrupt register(R) More...

#define **Sn_SR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0003))
Socket status register(R) More...

#define **Sn_PORT**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0004))
source port register(R/W) More...

#define **Sn_DHAR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0006))
Peer MAC register address(R/W) More...

#define **Sn_DIPR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x000C))
Peer IP register address(R/W) More...

#define **Sn_DPORT**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0010))
Peer port register address(R/W) More...

#define **Sn_MSSR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0012))

Maximum Segment Size(Sn_MSSR0) register address(R/W) More...

#define **Sn_PROTO**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0014))
IP Protocol(PROTO) Register(R/W) More...

#define **Sn_TOS**(sn) (**WIZCHIP_SREG_BLOCK**(sn) + 0x0015)
IP Type of Service(TOS) Register(R/W) More...

#define **Sn_TTL**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0016))
IP Time to live(TTL) Register(R/W) More...

#define **Sn_RXMEM_SIZE**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x001E))
Receive memory size register(R/W) More...

#define **Sn_TXMEM_SIZE**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x001F))
Transmit memory size register(R/W) More...

#define **Sn_TX_FSR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0020))
Transmit free memory size register(R) More...

#define **Sn_TX_RD**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0022))
Transmit memory read pointer register address(R) More...

#define **Sn_TX_WR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0024))
Transmit memory write pointer register address(R/W) More...

#define **Sn_RX_RSR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0026))

	Received data size register(R) More...
#define	**Sn_RX_RD**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0028)) Read point of Receive memory(R/W) More...
#define	**Sn_RX_WR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x002A)) Write point of Receive memory(R) More...
#define	**Sn_IMR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x002C)) socket interrupt mask register(R) More...
#define	**Sn_FRAG**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x002D)) Fragment field value in IP header register(R/W) More...

Detailed Description

Socket register group
Socket register configures and control SOCKETn which is necessary to data communication.

See also

Sn_MR, **Sn_CR**, **Sn_IR**, **Sn_IMR** : SOCKETn Control

Sn_SR, **Sn_PORT**, **Sn_DHAR**, **Sn_DIPR**, **Sn_DPORT** : SOCKETn Information

Sn_MSSR, **Sn_TOS**, **Sn_TTL**, **Sn_FRAG** : Internet protocol.

Sn_RXMEM_SIZE, **Sn_TXMEM_SIZE**, **Sn_TX_FSR**, **Sn_TX_RD**, **Sn_TX_WR**, **Sn_RX_RSR**, **Sn_RX_RD**, **Sn_RX_WR** : Data communication

Macro Definition Documentation

#define Sn_MR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLC (0x0000))

socket Mode register(R/W)

Sn_MR configures the option or protocol type of Socket n.

Each bit of **Sn_MR** defined as the following.

7	6	5	4	3	2	1
MULTI	MF	ND/MC	Reserved	Protocol[3]	Protocol[2]	Protocol[1]

- **Sn_MR_MULTI** : Support UDP Multicasting
- **Sn_MR_MF** : Support MACRAW
- **Sn_MR_ND** : No Delayed Ack(TCP) flag
- **Sn_MR_MC** : IGMP version used **in UDP mulitcasting**
- **Protocol**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	1	0	0	MACRAW

- **In case of Socket 0**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	1	0	0	MACRAW
0	1	0	1	PPPoE

  - **Sn_MR_MACRAW** : MAC LAYER RAW SOCK
  - **Sn_MR_UDP** : UDP
  - **Sn_MR_TCP** : TCP
  - **Sn_MR_CLOSE** : Unused socket

    Note

> MACRAW mode should be only used in Socket 0.

Definition at line **429** of file **w5200.h**.

#define Sn_CR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001))

Socket command register(R/W)

This is used to set the command for Socket n such as OPEN, CLOSE, CONNECT, LISTEN, SEND, and RECEIVE.
After W5200 accepts the command, the **Sn_CR** register is automatically cleared to 0x00. Even though **Sn_CR** is cleared to 0x00, the command is still being processed.
To check whether the command is completed or not, please check the **Sn_IR** or **Sn_SR**.

- **Sn_CR_OPEN** : Initialize or open socket.
- **Sn_CR_LISTEN** : Wait connection request in TCP mode(**Server mode**)
- **Sn_CR_CONNECT** : Send connection request in TCP mode(**Client mode**)
- **Sn_CR_DISCON** : Send closing request in TCP mode.
- **Sn_CR_CLOSE** : Close socket.
- **Sn_CR_SEND** : Update TX buffer pointer and send data.
- **Sn_CR_SEND_MAC** : Send data with MAC address, so without ARP process.
- **Sn_CR_SEND_KEEP** : Send keep alive message.
- **Sn_CR_RECV** : Update RX buffer pointer and receive data.
- **In case of S0_MR(P3:P0) = S0_MR_PPPoE**

Value	Symbol	Description
0x23	PCON	PPPoE connection begins by transmitting PPPoE discovery packet
0x24	PDISCON	Closes PPPoE connection
0x25	PCR	In each phase, it transmits REQ message.
0x26	PCN	In each phase, it transmits NAK message.
0x27	PCJ	In each phase, it transmits REJECT

		message.

Definition at line **457** of file **w5200.h**.

#define (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOC
Sn_IR (sn) + (0x0002))

Socket interrupt register(R)

Sn_IR indicates the status of Socket Interrupt such as establishment, termination, receiving data, timeout).
When an interrupt occurs and the corresponding bit of **Sn_IMR** is the corresponding bit of **Sn_IR** becomes
In order to clear the **Sn_IR** bit, the host should write the bit to

7	6	5	4	3	2	1	(
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	(

- **Sn_IR_PRECV** : **PPP Receive Interrupt**
- **Sn_IR_PFAIL** : **PPP Fail Interrupt**
- **Sn_IR_PNEXT** : **PPP Next Phase Interrupt**
- **Sn_IR_SENDOK** : **SEND_OK Interrupt**
- **Sn_IR_TIMEOUT** : **TIMEOUT Interrupt**
- **Sn_IR_RECV** : **RECV Interrupt**
- **Sn_IR_DISCON** : **DISCON Interrupt**
- **Sn_IR_CON** : **CON Interrupt**

Definition at line **478** of file **w5200.h**.

#define (_W5200_IO_BASE_ +
Sn_SR (sn) WIZCHIP_SREG_BLOCK(sn) + (0x0003))

Socket status register(R)

Sn_SR indicates the status of Socket n.
The status of Socket n is changed by **Sn_CR** or some special control packet as SYN, FIN packet in TCP.

Normal status

- **SOCK_CLOSED** : Closed
- **SOCK_INIT** : Initiate state
- **SOCK_LISTEN** : Listen state
- **SOCK_ESTABLISHED** : Success to connect
- **SOCK_CLOSE_WAIT** : Closing state
- **SOCK_UDP** : UDP socket
- **SOCK_MACRAW** : MAC raw mode socket

Temporary status during changing the status of Socket n.

- **SOCK_SYNSENT** : This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
- **SOCK_SYNRECV** : It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
- **SOCK_FIN_WAIT** : Connection state
- **SOCK_CLOSING** : Closing state
- **SOCK_TIME_WAIT** : Closing state
- **SOCK_LAST_ACK** : Closing state

Definition at line **501** of file **w5200.h**.

#define Sn_PORT (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004))

---

source port register(R/W)

Sn_PORT configures the source port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. It should be set before OPEN command is ordered.

Definition at line **509** of file **w5200.h**.

#define Sn_DHAR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006))

Peer MAC register address(R/W)

Sn_DHAR configures the destination hardware address of Socket n when using SEND_MAC command in UDP mode or it indicates that it is acquired in ARP-process by CONNECT/SEND command.

Definition at line **517** of file **w5200.h**.

#define Sn_DIPR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x000C))

Peer IP register address(R/W)

Sn_DIPR configures or indicates the destination IP address of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP client mode, it configures an IP address of TCP server before CONNECT command. In TCP server mode, it indicates an IP address of TCP client after successfully establishing connection. In UDP mode, it configures an IP address of peer to be received the UDP packet by SEND or SEND_MAC command.

Definition at line **527** of file **w5200.h**.

#define Sn_DPORT (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0010))

Peer port register address(R/W)

Sn_DPORT configures or indicates the destination port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP clientmode, it configures the listen port number of TCP server before CONNECT command. In TCP Servermode, it indicates the port number of TCP client after successfully establishing connection. In UDP mode, it configures the port number of peer to be transmitted the UDP packet by SEND/SEND_MAC command.

Definition at line **537** of file **w5200.h**.

#define (_W5200_IO_BASE_ +

Sn_MSSR (sn) WIZCHIP_SREG_BLOCK(sn) + (0x0012))

Maximum Segment Size(Sn_MSSR0) register address(R/W)

Sn_MSSR configures or indicates the MTU(Maximum Transfer Unit) of Socket n.

Definition at line **544** of file **w5200.h**.

#define (_W5200_IO_BASE_ +
Sn_PROTO (sn) WIZCHIP_SREG_BLOCK(sn) + (0x0014))

IP Protocol(PROTO) Register(R/W)

Sn_PROTO that sets the protocol number field of the IP header at the IP layer. It is valid only in IPRAW mode, and ignored in other modes.

Definition at line **552** of file **w5200.h**.

#define Sn_TOS (sn) (WIZCHIP_SREG_BLOCK(sn) + 0x0015)

IP Type of Service(TOS) Register(R/W)

Sn_TOS configures the TOS(Type Of Service field in IP Header) of Socket n. It is set before OPEN command.

Definition at line **560** of file **w5200.h**.

#define (_W5200_IO_BASE_ +
Sn_TTL (sn) WIZCHIP_SREG_BLOCK(sn) + (0x0016))

IP Time to live(TTL) Register(R/W)

Sn_TTL configures the TTL(Time To Live field in IP header) of Socket n. It is set before OPEN command.

Definition at line **568** of file **w5200.h**.

#define Sn_RXMEM_SIZE (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001E))

Receive memory size register(R/W)

Sn_RXMEM_SIZE configures the RX buffer block size of Socket n. Socket n RX Buffer Block size can be configured with 1,2,4,8, and 16 Kbytes. If a different size is configured, the data cannot be normally received from a peer. Although Socket n RX Buffer Block size is initially configured to 2Kbytes, user can re-configure its size using **Sn_RXMEM_SIZE**. The total sum of **Sn_RXMEM_SIZE** can not be exceed 16Kbytes. When exceeded, the data reception error is occurred.

Definition at line **588** of file **w5200.h**.

#define Sn_TXMEM_SIZE (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001F))

Transmit memory size register(R/W)

Sn_TXMEM_SIZE configures the TX buffer block size of Socket n. Socket n TX Buffer Block size can be configured with 1,2,4,8, and 16 Kbytes. If a different size is configured, the data can't be normally transmitted to a peer. Although Socket n TX Buffer Block size is initially configured to 2Kbytes, user can be re-configure its size using **Sn_TXMEM_SIZE**. The total sum of **Sn_TXMEM_SIZE** can not be exceed 16Kbytes. When exceeded, the data transmission error is occurred.

Definition at line **599** of file **w5200.h**.

#define Sn_TX_FSR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020))

Transmit free memory size register(R)

Sn_TX_FSR indicates the free size of Socket n TX Buffer Block. It is initialized to the configured size by **Sn_TXMEM_SIZE**. Data bigger than **Sn_TX_FSR** should not be saved in the Socket n TX Buffer because the bigger data overwrites the previous saved data not yet sent. Therefore, check before saving the data to the Socket n TX Buffer, and if data is equal or smaller than its checked size, transmit the data with SEND/SEND_MAC command after saving the data in Socket n TX buffer. But, if data is bigger than its checked size, transmit the data after dividing into the checked size and saving in the Socket n TX buffer.

Definition at line **610** of file **w5200.h**.

#define Sn_TX_RD (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022))

Transmit memory read pointer register address(R)

Sn_TX_RD is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001), it is re-initialized while connecting with TCP. After its initialization, it is auto-increased by SEND command. SEND command transmits the saved data from the current **Sn_TX_RD** to the **Sn_TX_WR** in the Socket n TX Buffer. After transmitting the saved data, the SEND command increases the **Sn_TX_RD** as same as the **Sn_TX_WR**. If its increment value exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **622** of file **w5200.h**.

#define Sn_TX_WR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024))

Transmit memory write pointer register address(R/W)

Sn_TX_WR is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001), it is re-initialized while connecting with TCP.
It should be read or be updated like as follows.

1. Read the starting address for saving the transmitting data.
2. Save the transmitting data from the starting address of Socket n TX buffer.
3. After saving the transmitting data, update **Sn_TX_WR** to the increased value as many as transmitting data size. If the increment value exceeds the maximum value 0xFFFF(greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.
4. Transmit the saved data in Socket n TX Buffer by using SEND/SEND command

Definition at line **636** of file **w5200.h**.

#define Sn_RX_RSR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0026))

Received data size register(R)

Sn_RX_RSR indicates the data size received and saved in Socket n RX Buffer. **Sn_RX_RSR** does not exceed the **Sn_RXMEM_SIZE** and is calculated as the difference between Socket n RX Write Pointer (**Sn_RX_WR**)and Socket n RX Read Pointer (**Sn_RX_RD**)

Definition at line **645** of file **w5200.h**.

#define Sn_RX_RD (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0028))

Read point of Receive memory(R/W)

Sn_RX_RD is initialized by OPEN command. Make sure to be read or updated as follows.

1. Read the starting save address of the received data.
2. Read data from the starting address of Socket n RX Buffer.
3. After reading the received data, Update **Sn_RX_RD** to the increased value as many as the reading size. If the increment value exceeds the maximum value 0xFFFF, that is, is greater than 0x10000 and the carry bit occurs, update with the lower 16bits value ignored the carry bit.
4. Order RECV command is for notifying the updated **Sn_RX_RD** to W5200.

Definition at line **658** of file **w5200.h**.

#define Sn_RX_WR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002A))

Write point of Receive memory(R)

Sn_RX_WR is initialized by OPEN command and it is auto-increased by the data reception. If the increased value exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **667** of file **w5200.h**.

#define Sn_IMR (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002C))

socket interrupt mask register(R)

Sn_IMR masks the interrupt of Socket n. Each bit corresponds to each bit of **Sn_IR**. When a Socket n Interrupt is occurred and the corresponding bit of **Sn_IMR** is the corresponding bit of **Sn_IR** becomes When both the corresponding bit of **Sn_IMR** and **Sn_IR** are and the n-th bit of **IR** is Host is interrupted by asserted INTn PIN to low.

Definition at line **677** of file **w5200.h**.

#define Sn_FRAG (sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002D))

Fragment field value in IP header register(R/W)

Sn_FRAG configures the FRAG(Fragment field in IP header).

Definition at line **684** of file **w5200.h**.

Socket APIs

W5300

WHIZCHIP register defines and I/O functions of **W5300**. More...

Modules

WIZCHIP I/O functions
This supports the basic I/O functions for **WIZCHIP register**.

WIZCHIP register
WHIZCHIP register defines register group of **W5300**.

Detailed Description

WHIZCHIP register defines and I/O functions of **W5300**.

- **WIZCHIP register** : **Common register** and **Socket register**
- **WIZCHIP I/O functions** : **Basic I/O function**, **Common register access functions** and **Socket register access functions**

Socket APIs

WIZCHIP I/O functions

W5300

This supports the basic I/O functions for **WIZCHIP register**. More...

Modules

Basic I/O function
These are basic input/output functions to read values from register or write values to register.

Common register access functions
These are functions to access **common registers**.

Socket register access functions
These are functions to access **socket registers**.

Detailed Description

This supports the basic I/O functions for **WIZCHIP register**.

- **Basic I/O function**
  WIZCHIP_READ(), **WIZCHIP_WRITE()**

- **Common register access functions**
  1. **Mode**
     getMR(), **setMR()**
  2. **Interrupt**
     getIR(), **setIR()**, **getIMR()**, **setIMR()**, **getSIR()**, **setSIR()**, **getSIMR()**, **setSIMR()**
  3. **Network Information**
     getSHAR(), **setSHAR()**, **getGAR()**, **setGAR()**, **getSUBR()**, **setSUBR()**, **getSIPR()**, **setSIPR()**
  4. **Retransmission**
     getRCR(), **setRCR()**, **getRTR()**, **setRTR()**
  5. **PPPoE**
     getPTIMER(), **setPTIMER()**, **getPMAGIC()**, **getPMAGIC()**, **getPSID()**, **setPSID()**, **getPHAR()**, **setPHAR()**, **getPMRU()**, **setPMRU()**
  6. **ICMP packet**
     getUIPR(), **getUPORTR()**

Socket Memory
getMTYPER(), **setMTYPER()**
getTMS01R(), **getTMS23R()**, **getTMS45R()**, **getTMS67R()**,
setTMS01R(), **setTMS23R()**, **setTMS45R()**, **setTMS67R()**
getRMS01R(), **getRMS23R()**, **getRMS45R()**, **getRMS67R()**,
setRMS01R(), **setRMS23R()**, **setRMS45R()**, **setRMS67R()**

1. **etc**.
   getPn_BRDYR(), **setPn_BRDYR()**, **getPn_BDPTHR()**, **setPn_BDPTHR()**, **getIDR()**

Socket register access functions

1. **SOCKET control**
   getSn_MR(), **setSn_MR()**, **getSn_CR()**, **setSn_CR()**,
   getSn_IMR(), **setSn_IMR()**, **getSn_IR()**, **setSn_IR()**
2. **SOCKET information**
   getSn_SR(), **getSn_DHAR()**, **setSn_DHAR()**, **getSn_PORT()**,
   setSn_PORT(), **getSn_DIPR()**, **setSn_DIPR()**, **getSn_DPORT()**,
   setSn_DPORT() **getSn_MSSR()**, **setSn_MSSR()**
3. **SOCKET communication**
   getSn_RXBUF_SIZE(), **setSn_RXBUF_SIZE()**,
   getSn_TXBUF_SIZE(), **setSn_TXBUF_SIZE()**
   getSn_TX_RD(), **getSn_TX_WR()**, **setSn_TX_WR()**
   getSn_RX_RD(), **setSn_RX_RD()**, **getSn_RX_WR()**
   getSn_TX_FSR(), **getSn_RX_RSR()**, **getSn_KPALVTR()**,
   setSn_KPALVTR()
4. **IP header field**
   getSn_FRAG(), **setSn_FRAG()**, **getSn_TOS()**, **setSn_TOS()**
   getSn_TTL(), **setSn_TTL()**

Socket APIs

Basic I/O function

These are basic input/output functions to read values from register or write values to register. More...

Functions

uint16_t **WIZCHIP_READ** (uint32_t AddrSel)
It reads 1 byte value from a register. More...

void **WIZCHIP_WRITE** (uint32_t AddrSel, uint16_t wb)
It writes 1 byte value to a register. More...

void **wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint32_t len)
It copies data to internal TX memory. More...

void **wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint32_t len)
It copies data to your buffer from internal RX memory. More...

void **wiz_recv_ignore** (uint8_t sn, uint32_t len)
It discard the received data in RX memory. More...

Detailed Description

These are basic input/output functions to read values from register or write values to register.

Function Documentation

uint16_t WIZCHIP_READ (uint32_t AddrSel)

It reads 1 byte value from a register.

Parameters

AddrSel Register address

Returns

The value of register

void WIZCHIP_WRITE (uint32_t AddrSel, uint16_t wb)

It writes 1 byte value to a register.

Parameters

AddrSel Register address

wb Write data

Returns

void

void wiz_send_data (uint8_t sn, uint8_t * wizdata, uint32_t len)

It copies data to internal TX memory.

This function reads the Tx write pointer register and after that, it copies the *wizdata(pointer buffer)* of the length of *len(variable)* bytes to internal TX memory and updates the Tx write pointer register. This function is being called by **send()** and **sendto()** function also.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
wizdata	Pointer buffer to write data
len	Data length

See also

wiz_recv_data()

```
void wiz_recv_data ( uint8_t    sn,
                     uint8_t *  wizdata,
                     uint32_t   len
                   )
```

It copies data to your buffer from internal RX memory.

This function read the Rx read pointer register and after that, it copies the received data from internal RX memory to *wizdata(pointer variable)* of the length of *len(variable)* bytes. This function is being called by **recv()** also.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
wizdata	Pointer buffer to read data
len	Data length

See also

wiz_send_data()

```
void wiz_recv_ignore ( uint8_t    sn,
                       uint32_t   len
                     )
```

It discard the received data in RX memory.

It discards the data of the length of *len(variable)* bytes in internal RX memory.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
len	Data length

Socket APIs

Common register access functions

These are functions to access **common registers**. More...

Macros

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, ir & 0xF0FF)
Set Mode Register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xF0FF)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**_IMR_**, imr & 0xF0FF)
Set *IMR* register. More...

#define **getIMR**() (**WIZCHIP_READ**(**_IMR_**) & 0xF0FF)
Get *IMR* register. More...

#define **setSHAR**(shar)
Set local MAC address. More...

#define **setGAR**(gar)
Set gateway IP address. More...

#define **getGAR**(gar)
Get gateway IP address. More...

#define **setSUBR**(subr)
Set subnet mask address. More...

#define **getSUBR**(subr)
Get subnet mask address. More...

#define **setSIPR**(sipr)
Set local IP address. More...

#define **getSIPR**(sipr)
Get local IP address. More...

#define **setRTR**(rtr) **WIZCHIP_WRITE**(**_RTR_**, rtr)
Set *RTR* register. More...

#define **getRTR**() **WIZCHIP_READ**(**_RTR_**)
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, ((uint16_t)rcr)&0x00FF)
Set *RCR* register. More...

#define **getRCR**() ((uint8_t)(**WIZCHIP_READ**(**_RCR_**) & 0x00FF))
Get *RCR* register. More...

#define **setTMS01R**(tms01r) **WIZCHIP_WRITE**(**TMS01R**,tms01r)
Set **TMS01R** register. More...

#define **getTMS01R**() **WIZCHIP_READ**(**TMS01R**)
Get **TMS01R** register. More...

#define **setTMS23R**(tms23r) **WIZCHIP_WRITE**(**TMS23R**,tms23r)
Set **TMS23R** register. More...

#define **getTMS23R**() **WIZCHIP_READ**(**TMS23R**)
Get **TMS23R** register. More...

#define **setTMS45R**(tms45r) **WIZCHIP_WRITE**(**TMS45R**,tms45r)
Set **TMS45R** register. More...

#define **getTMS45R**() **WIZCHIP_READ**(**TMS45R**)
Get **TMS45R** register. More...

#define **setTMS67R**(tms67r) **WIZCHIP_WRITE**(**TMS67R**,tms67r)
Set **TMS67R** register. More...

#define **getTMS67R**() **WIZCHIP_READ**(**TMS67R**)
Get **TMS67R** register. More...

#define **setRMS01R**(rms01r) **WIZCHIP_WRITE**(**RMS01R**,rms01r)
Set **RMS01R** register. More...

#define **getRMS01R**() **WIZCHIP_READ**(**RMS01R**)
Get **RMS01R** register. More...

#define **setRMS23R**(rms23r) **WIZCHIP_WRITE**(**RMS23R**,rms23r)
Set **RMS23R** register. More...

#define **getRMS23R**() **WIZCHIP_READ**(**RMS23R**)
Get **RMS23R** register. More...

#define **setRMS45R**(rms45r) **WIZCHIP_WRITE**(**RMS45R**,rms45r)
Set **RMS45R** register. More...

#define **getRMS45R**() **WIZCHIP_READ**(**RMS45R**)
Get **RMS45R** register. More...

#define **setRMS67R**(rms67r) **WIZCHIP_WRITE**(**RMS67R**,rms67r)
Set **RMS67R** register. More...

#define **getRMS67R**() **WIZCHIP_READ**(**RMS67R**)
Get **RMS67R** register. More...

#define **setMTYPER**(mtype) **WIZCHIP_WRITE**(**MTYPER**, mtype)
Set **MTYPER** register. More...

#define **getMTYPER**() **WIZCHIP_READ**(**MTYPER**)
Get **MTYPER** register. More...

#define **getPATR**() **WIZCHIP_READ**(**PATR**)
Get RATR register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ((uint16_t)ptimer) & 0x00FF)

Set **PTIMER** register. More...

#define **getPTIMER**() ((uint8_t)(**WIZCHIP_READ**(**PTIMER**) & 0x00FF))
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, ((uint16_t)pmagic) & 0x00FF)
Set **PMAGIC** register. More...

#define **getPMAGIC**() ((uint8_t)(**WIZCHIP_READ**(**PMAGIC**) & 0x00FF))
Get **PMAGIC** register. More...

#define **getPSIDR**() **WIZCHIP_READ**(**PSIDR**)
Get **PSID** register. More...

#define **getPDHAR**(pdhar)
Get **PDHAR** register. More...

#define **getUIPR**(uipr)
Get unreachable IP address. **UIPR**. More...

#define **getUPORTR**() **WIZCHIP_READ**(**UPORTR**)
Get **UPORTR** register. More...

#define **getFMTUR**() **WIZCHIP_READ**(**FMTUR**)
Get **FMTUR** register. More...

#define **getPn_BRDYR**(p) ((uint8_t) (**WIZCHIP_READ**(**Pn_BRDYR**(p)) & 0x00FF))
Get **Pn_BRDYR** register. More...

#define **setPn_BRDYR**(p, brdyr) **WIZCHIP_WRITE**(**Pn_BRDYR**(p), brdyr & 0x00E7)
Set **Pn_BRDYR** register. More...

#define **getPn_BDPTHR**(p) **WIZCHIP_READ**(**Pn_BDPTHR**(p))
Get **Pn_BDPTHR** register. More...

#define **setPn_BDPTHR**(p, bdpthr) **WIZCHIP_WRITE**(**Pn_BDPTHR**(p),bdpthr)
Set **Pn_BDPTHR** register. More...

#define **getIDR**() **WIZCHIP_READ**(**IDR**)
Get **IDR** register. More...

Functions

void	**setTMSR** (uint8_t sn, uint8_t tmsr) Set **TMSR0** ~ **TMSR7** register. More...
uint8_t	**getTMSR** (uint8_t sn) Get **TMSR0** ~ **TMSR7** register. More...
void	**setRMSR** (uint8_t sn, uint8_t rmsr) Set **RMS01R** ~ **RMS67R** register. More...
uint8_t	**getRMSR** (uint8_t sn) Get **RMS01R** ~ **RMS67R** register. More...

Detailed Description

These are functions to access **common registers**.

Macro Definition Documentation

#define setIR (ir) WIZCHIP_WRITE(IR, ir & 0xF0FF)

Set Mode Register.

Parameters

(

Definition at line **1370** of file **w5300.h**.

#define getIR ( ) (WIZCHIP_READ(IR) & 0xF0FF)

Get **IR** register.

Returns

uint8_t. Value of **IR** register.

See also

setIR()

Definition at line **1379** of file **w5300.h**.

#define setIMR (imr) WIZCHIP_WRITE(_IMR_, imr & 0xF0FF)

Set *IMR* register.

Parameters

(uint16_t)imr Value to set *IMR* register.

See also

getIMR()

Definition at line **1389** of file **w5300.h**.

#define getIMR ( ) (WIZCHIP_READ(_IMR_) & 0xF0FF)

Get *IMR* register.

Returns
uint16_t. Value of **IR** register.

See also
setIMR()

Definition at line **1398** of file **w5300.h**.

#define setSHAR (shar)

Value:

```
{ \
      WIZCHIP_WRITE(SHAR,
    (((uint16_t)((shar)[0])) << 8) +
    (((uint16_t)((shar)[1])) & 0x00FF)); \
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SHAR,2),
    (((uint16_t)((shar)[2])) << 8) +
    (((uint16_t)((shar)[3])) & 0x00FF)); \
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SHAR,4),
    (((uint16_t)((shar)[4])) << 8) +
    (((uint16_t)((shar)[5])) & 0x00FF)); \
   }
```

Set local MAC address.

Parameters
(uint8_t*)shar Pointer variable to set local MAC address. It should be allocated 6 bytes.

See also
getSHAR()

Definition at line **1407** of file **w5300.h**.

#define setGAR (gar)

Value:

```
{ \
      WIZCHIP_WRITE(GAR,
    (((uint16_t)((gar)[0])) << 8) + (((uint16_t)
    ((gar)[1])) & 0x00FF)); \
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(GAR,2),
    (((uint16_t)((gar)[2])) << 8) + (((uint16_t)
    ((gar)[3])) & 0x00FF)); \
   }
```

Set gateway IP address.

Parameters

(uint8_t*)gar Pointer variable to set gateway IP address. It should be allocated 4 bytes.

See also

getGAR()

Definition at line **1434** of file **w5300.h**.

#define getGAR (gar)

Value:

```
{ \
        (gar)[0] = (uint8_t)(WIZCHIP_READ(GAR)
    >> 8); \
        (gar)[1] = (uint8_t)(WIZCHIP_READ(GAR));
    \
        (gar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(GAR,2)) >>
    8); \
        (gar)[3] = (uint8_t)
```

```
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(GAR,2))); \
  }
```

Get gateway IP address.

Parameters

(uint8_t*)gar Pointer variable to get gateway IP address. It should be allocated 4 bytes.

See also

setGAR()

Definition at line **1445** of file **w5300.h**.

#define setSUBR (subr)

Value:

```
{ \
      WIZCHIP_WRITE(SUBR,
    (((uint16_t)((subr)[0])) << 8) +
    (((uint16_t)((subr)[1])) & 0x00FF)); \
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SUBR,2),
    (((uint16_t)((subr)[2])) << 8) +
    (((uint16_t)((subr)[3])) & 0x00FF)); \
  }
```

Set subnet mask address.

Parameters

(uint8_t*)subr Pointer variable to set subnet mask address. It should be allocated 4 bytes.

See also

getSUBR()

Definition at line **1458** of file **w5300.h**.

#define getSUBR (subr)

Value:

```
{ \
        (subr)[0] = (uint8_t)(WIZCHIP_READ(SUBR)
    >> 8); \
        (subr)[1] = (uint8_t)
    (WIZCHIP_READ(SUBR));      \
        (subr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2)) >>
    8); \
        (subr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2)));
    \
   }
```

Get subnet mask address.

Parameters

(uint8_t*)subr Pointer variable to get subnet mask address. It should be allocated 4 bytes.

See also

setSUBR()

Definition at line **1469** of file **w5300.h**.

#define setSIPR (sipr)

Value:

```
{ \
      WIZCHIP_WRITE(SIPR,
    (((uint16_t)((sipr)[0])) << 8) +
    (((uint16_t)((sipr)[1])) & 0x00FF)); \
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SIPR,2),
    (((uint16_t)((sipr)[2])) << 8) +
    (((uint16_t)((sipr)[3])) & 0x00FF)); \
```

```
    }
```

Set local IP address.

Parameters

(uint8_t*)sipr Pointer variable to set local IP address. It should be allocated 4 bytes.

See also

getSIPR()

Definition at line **1482** of file **w5300.h**.

#define getSIPR (sipr)

Value:

```
{ \
        (sipr)[0] = (uint8_t)(WIZCHIP_READ(SIPR)
    >> 8); \
        (sipr)[1] = (uint8_t)
    (WIZCHIP_READ(SIPR));      \
       (sipr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SIPR,2)) >>
    8); \
        (sipr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SIPR,2)));
    \
   }
```

Get local IP address.

Parameters

(uint8_t*)sipr Pointer variable to get local IP address. It should be allocated 4 bytes.

See also

setSIPR()

Definition at line **1493** of file **w5300.h**.

#define setRTR (rtr) WIZCHIP_WRITE(_RTR_, rtr)

Set *RTR* register.

Parameters
(uint16_t)rtr Value to set *RTR* register.

See also
getRTR()

Definition at line **1507** of file **w5300.h**.

#define getRTR ( ) WIZCHIP_READ(_RTR_)

Get *RTR* register.

Returns
uint16_t. Value of *RTR* register.

See also
setRTR()

Definition at line **1516** of file **w5300.h**.

#define setRCR (rcr) WIZCHIP_WRITE(_RCR_, ((uint16_t)rcr)&0x00FF)

Set *RCR* register.

Parameters
(uint8_t)rcr Value to set *RCR* register.

See also
getRCR()

Definition at line **1525** of file **w5300.h**.

#define getRCR ( ) ((uint8_t)(WIZCHIP_READ(_RCR_) & 0x00FF))

Get *RCR* register.

Returns
uint8_t. Value of *RCR* register.

See also
setRCR()

Definition at line **1534** of file **w5300.h**.

#define setTMS01R (tms01r) WIZCHIP_WRITE(TMS01R,tms01r)

Set **TMS01R** register.

Parameters

(uint16_t)tms01r	Value to set **TMS01R** register. The lower socket memory size is located at MSB of tms01r.

See also
getTMS01R()

Definition at line **1543** of file **w5300.h**.

#define getTMS01R ( ) WIZCHIP_READ(TMS01R)

Get **TMS01R** register.

Returns
uint16_t. Value of **TMS01R** register.

See also
setTMS01R()

Definition at line **1552** of file **w5300.h**.

#define setTMS23R (tms23r) WIZCHIP_WRITE(TMS23R,tms23r)

Set **TMS23R** register.

Parameters

(uint16_t)tms23r Value to set **TMS23R** register. The lower socket memory size is located at MSB of tms01r.

See also
getTMS23R()

Definition at line **1561** of file **w5300.h**.

#define getTMS23R ( ) WIZCHIP_READ(TMS23R)

Get **TMS23R** register.

Returns
uint16_t. Value of **TMS23R** register.

See also
setTMS23R()

Definition at line **1570** of file **w5300.h**.

#define setTMS45R (tms45r) WIZCHIP_WRITE(TMS45R,tms45r)

Set **TMS45R** register.

Parameters

(uint16_t)tms45r Value to set **TMS45R** register. The lower socket memory size is located at MSB of tms45r.

See also

getTMS45R()

Definition at line **1579** of file **w5300.h**.

#define getTMS45R ( ) WIZCHIP_READ(TMS45R)

Get **TMS45R** register.

Returns

uint16_t. Value of **TMS45R** register.

See also

setTMS45R()

Definition at line **1588** of file **w5300.h**.

#define setTMS67R (tms67r) WIZCHIP_WRITE(TMS67R,tms67r)

Set **TMS67R** register.

Parameters

(uint16_t)tms67r Value to set **TMS67R** register. The lower socket memory size is located at MSB of tms67r.

See also

getTMS67R()

Definition at line **1597** of file **w5300.h**.

#define getTMS67R ( ) WIZCHIP_READ(TMS67R)

Get **TMS67R** register.

Returns
uint16_t. Value of **TMS67R** register.

See also
setTMS67R()

Definition at line **1606** of file **w5300.h**.

#define setRMS01R (rms01r) WIZCHIP_WRITE(RMS01R,rms01r)

Set **RMS01R** register.

Parameters

(uint16_t)rms01r	Value to set **RMS01R** register. The lower socket memory size is located at MSB of rms01r.

See also
getRMS01R()

Definition at line **1635** of file **w5300.h**.

#define getRMS01R ( ) WIZCHIP_READ(RMS01R)

Get **RMS01R** register.

Returns
uint16_t. Value of **RMS01R** register.

See also
setRMS01R()

Definition at line **1644** of file **w5300.h**.

#define setRMS23R (rms23r) WIZCHIP_WRITE(RMS23R,rms23r)

Set **RMS23R** register.

Parameters

(uint16_t)rms23r Value to set **RMS23R** register. The lower socket memory size is located at MSB of rms01r.

See also

getRMS23R()

Definition at line **1653** of file **w5300.h**.

#define getRMS23R ( ) WIZCHIP_READ(RMS23R)

Get **RMS23R** register.

Returns

uint16_t. Value of **RMS23R** register.

See also

setRMS23R()

Definition at line **1662** of file **w5300.h**.

#define setRMS45R (rms45r) WIZCHIP_WRITE(RMS45R,rms45r)

Set **RMS45R** register.

Parameters

(uint16_t)rms45r Value to set **RMS45R** register. The lower

socket memory size is located at MSB of rms45r.

See also
getRMS45R()

Definition at line **1671** of file **w5300.h**.

#define getRMS45R ( ) WIZCHIP_READ(RMS45R)

Get **RMS45R** register.

Returns
uint16_t. Value of **RMS45R** register.

See also
setRMS45R()

Definition at line **1680** of file **w5300.h**.

#define setRMS67R (rms67r) WIZCHIP_WRITE(RMS67R,rms67r)

Set **RMS67R** register.

Parameters
(uint16_t)rms67r Value to set **RMS67R** register. The lower socket memory size is located at MSB of rms67r.

See also
getRMS67R()

Definition at line **1689** of file **w5300.h**.

#define getRMS67R ( ) WIZCHIP_READ(RMS67R)

Get **RMS67R** register.

Returns
uint16_t. Value of **RMS67R** register.

See also
setRMS67R()

Definition at line **1698** of file **w5300.h**.

#define setMTYPER (mtype) WIZCHIP_WRITE(MTYPER, mtype)

Set **MTYPER** register.

Parameters
(uint16_t)mtyper Value to set **MTYPER** register.

See also
getMTYPER()

Definition at line **1727** of file **w5300.h**.

#define getMTYPER ( ) WIZCHIP_READ(MTYPER)

Get **MTYPER** register.

Returns
uint16_t. Value of **MTYPER** register.

See also
setMTYPER()

Definition at line **1736** of file **w5300.h**.

#define getPATR ( ) WIZCHIP_READ(PATR)

Get RATR register.

Returns
uint16_t. Value of **PATR** register.

Definition at line **1744** of file **w5300.h**.

#define setPTIMER (ptimer) WIZCHIP_WRITE(PTIMER, ((uint16_t)ptimer) & 0x00FF)

Set **PTIMER** register.

Parameters
(uint8_t)ptimer Value to set **PTIMER** register.

See also
getPTIMER()

Definition at line **1753** of file **w5300.h**.

#define getPTIMER ( ) ((uint8_t)(WIZCHIP_READ(PTIMER) & 0x00FF))

Get **PTIMER** register.

Returns
uint8_t. Value of **PTIMER** register.

See also
setPTIMER()

Definition at line **1762** of file **w5300.h**.

#define setPMAGIC (pmagic) WIZCHIP_WRITE(PMAGIC, ((uint16_t)pmagic) & 0x00FF)

Set **PMAGIC** register.

Parameters
(uint8_t)pmagic Value to set **PMAGIC** register.

See also
getPMAGIC()

Definition at line **1771** of file **w5300.h**.

#define getPMAGIC ( ) ((uint8_t)(WIZCHIP_READ(PMAGIC) & 0x00FF))

Get **PMAGIC** register.

Returns
uint8_t. Value of **PMAGIC** register.

See also
setPMAGIC()

Definition at line **1780** of file **w5300.h**.

#define getPSIDR ( ) WIZCHIP_READ(PSIDR)

Get **PSID** register.

Returns
uint16_t. Value of **PSID** register.

Definition at line **1788** of file **w5300.h**.

#define getPDHAR (pdhar)

Value:

```
{ \
        (pdhar)[0] = (uint8_t)
    (WIZCHIP_READ(PDHAR) >> 8); \
        (pdhar)[1] = (uint8_t)
    (WIZCHIP_READ(PDHAR));      \
        (pdhar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,2))
    >> 8); \
        (pdhar)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,2)));
    \
        (pdhar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4))
    >> 8); \
        (pdhar)[5] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4)));
    \
   }
```

Get **PDHAR** register.

Parameters

(uint8_t*)pdhar Pointer variable to PPP destination MAC register address. It should be allocated 6 bytes.

Definition at line **1796** of file **w5300.h**.

#define getUIPR (uipr)

Value:

```
{ \
        (uipr)[0] = (uint8_t)(WIZCHIP_READ(UIPR)
    >> 8); \
        (uipr)[1] = (uint8_t)
    (WIZCHIP_READ(UIPR));      \
        (uipr)[2] = (uint8_t)
```

```
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,2)) >>
    8); \
        (uipr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,2)));
    \
  }
```

Get unreachable IP address. **UIPR**.

Parameters

(uint8_t*)uipr Pointer variable to get unreachable IP address. It should be allocated 4 bytes.

Definition at line **1810** of file **w5300.h**.

#define getUPORTR ( ) WIZCHIP_READ(UPORTR)

Get **UPORTR** register.

Returns

uint16_t. Value of **UPORTR** register.

Definition at line **1822** of file **w5300.h**.

#define getFMTUR ( ) WIZCHIP_READ(FMTUR)

Get **FMTUR** register.

Returns

uint16_t. Value of **FMTUR** register.

Definition at line **1830** of file **w5300.h**.

#define getPn_BRDYR (p) ((uint8_t) (WIZCHIP_READ(Pn_BRDYR(p)) & 0x00FF))

Get **Pn_BRDYR** register.

Returns
uint8_t. Value of **Pn_BRDYR** register.

Definition at line **1839** of file **w5300.h**.

```
#define
setPn_BRDYR      (  p,
                    brdyr
                      WIZCHIP_WRITE(Pn_BRDYR(p), brdyr &
                    )  0x00E7)
```

Set **Pn_BRDYR** register.

Parameters
p Pin number (p = 0,1,2,3)
brdyr Set a value **Pn_BRDYR(p)**.

Definition at line **1848** of file **w5300.h**.

```
#define getPn_BDPTHR (  p )   WIZCHIP_READ(Pn_BDPTHR(p))
```

Get **Pn_BDPTHR** register.

Parameters
p Pin number (p = 0,1,2,3)

Returns
uint16_t. Value of **Pn_BDPTHR** register.

Definition at line **1857** of file **w5300.h**.

```
#define
setPn_BDPTHR      (  p,
```

```
                    bdpthr
                ) WIZCHIP_WRITE(Pn_BDPTHR(p),bdpthr)
```

Set **Pn_BDPTHR** register.

Parameters

p	Pin number (p = 0,1,2,3)
bdpthr	Value of **Pn_BDPTHR**

Definition at line **1866** of file **w5300.h**.

#define getIDR ( ) WIZCHIP_READ(IDR)

Get **IDR** register.

Returns

uint16_t. Always 0x5300.

Definition at line **1875** of file **w5300.h**.

Function Documentation

void setTMSR (uint8_t sn,
uint8_t tmsr
)

Set **TMSR0** ~ **TMSR7** register.

Parameters

(uint8_t)sn Socket number. It should be 0 ~ 7.
(uint8_t)tmsr Value to set **TMSR0** ~**TMSR7** register.

See also

getTMSR()

uint8_t getTMSR (uint8_t sn)

Get **TMSR0** ~ **TMSR7** register.

Parameters

(uint8_t)sn Socket number. It should be 0 ~ 7.

Returns

uint8_t. Value of **TMSR0** ~ **TMSR7**

See also

getTMSR()

void setRMSR (uint8_t sn,
uint8_t rmsr
)

Set **RMS01R** ~ **RMS67R** register.

Parameters

(uint8_t)sn Socket number. It should be 0 ~ 7.

(uint8_t)rmsr Value to set **RMSR0** ~**RMSR7** register.

See also

getTMSR()

uint8_t getRMSR (uint8_t sn)

Get **RMS01R** ~ **RMS67R** register.

Parameters

(uint8_t)sn Socket number. It shoudl be 0 ~ 7.

Returns

uint8_t. Value of **RMSR0** ~ **RMSR7** register.

See also

setRMSR()

Socket APIs

Socket register access functions

These are functions to access **socket registers**. More...

Macros

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), ((uint16_t)c
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_CR**(sn)))
Get **Sn_CR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), ((uint16_
0x00FF)
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_IMR**(sn)))
Get **Sn_IMR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ((uint16_t)ir) &
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_IR**(sn)))
Get **Sn_IR** register. More...

#define **getSn_SSR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_SR**(sn)))
Get **Sn_SR** register. More...

#define **setSn_PORTR**(sn, port) **WIZCHIP_WRITE**(**Sn_PORTR**(sn),
Set **Sn_PORTR** register. More...

#define **getSn_PORTR**(sn, port) **WIZCHIP_READ**(**Sn_PORTR**(sn))
Get **Sn_PORTR** register. More...

#define **setSn_DHAR**(sn, dhar)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar)
Get **Sn_MR** register. More...

#define **setSn_DPORTR**(sn, dport) **WIZCHIP_WRITE**(**Sn_DPORTR**(
Set **Sn_DPORT** register. More...

#define **getSn_DPORTR**(sn) **WIZCHIP_READ**(**Sn_DPORTR**(sn))
Get **Sn_DPORT** register. More...

#define **setSn_DIPR**(sn, dipr)
Set **Sn_DIPR** register. More...

#define **getSn_DIPR**(sn, dipr)
Get **Sn_DIPR** register. More...

#define **setSn_MSSR**(sn, mss) **WIZCHIP_WRITE**(**Sn_MSSR**(sn), ms
Set **Sn_MSSR** register. More...

#define **getSn_MSSR**(sn) **WIZCHIP_READ**(**Sn_MSSR**(sn))
Get **Sn_MSSR** register. More...

#define **setSn_KPALVTR**(sn, kpalvt) **WIZCHIP_WRITE**(**Sn_KPALVT**
(**WIZCHIP_READ**(**Sn_KPALVTR**(sn)) & 0x00FF) | (((uint16_t)l
Set **Sn_KPALVTR** register. More...

#define **getSn_KPALVTR**(sn) ((uint8_t)(**WIZCHIP_READ**(**Sn_KPALV**
Get **Sn_KPALVTR** register. More...

#define **setSn_PROTOR**(sn, proto) **WIZCHIP_WRITE**(**Sn_PROTOR**(
(**WIZCHIP_READ**(**Sn_PROTOR**(sn) & 0xFF00) | (((uint16_t)pr
0x00FF))
Set **Sn_PROTOR** register. More...

#define **getSn_PROTOR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_PROTO**
Get **Sn_PROTOR** register. More...

#define **setSn_TX_WRSR**(sn, txwrs)
Set **Sn_TX_WRSR** register. More...

#define **getSn_TX_WRSR**(sn) ((((uint32_t)**WIZCHIP_READ**(**Sn_TX_**
<< 16) +
(((uint32_t)**WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_**
& 0x0000FFFF))
Get **Sn_TX_WRSR** register. More...

#define **setSn_TX_FIFOR**(sn, txfifo) **WIZCHIP_WRITE**(**Sn_TX_FIFO**
Set **Sn_TX_FIFOR** register. More...

#define **getSn_RX_FIFOR**(sn) **WIZCHIP_READ**(**Sn_RX_FIFOR**(sn))
Get **Sn_RX_FIFOR** register. More...

#define **setSn_TOSR**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), ((uint1
0x00FF)
Set **Sn_TOSR** register. More...

#define **getSn_TOSR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_TOSR**(sn)))
Get **Sn_TOSR** register. More...

#define **setSn_TTLR**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTLR**(sn), ((uint1
0x00FF)
Set **Sn_TTLR** register. More...

#define **getSn_TTLR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_TTL**(sn)))
Get **Sn_TTLR** register. More...

#define **setSn_FRAGR**(sn, frag) **WIZCHIP_WRITE**(**Sn_FRAGR**(sn),
>>8))
Set **Sn_FRAGR** register. More...

#define **getSn_FRAGR**(sn) (**WIZCHIP_READ**(**Sn_FRAG**(sn)) << 8)
Get **Sn_FRAGR** register. More...

Functions

uint32_t **getSn_TX_FSR** (uint8_t sn)
Get **Sn_TX_FSR** register. More...

uint32_t **getSn_RX_RSR** (uint8_t sn)
Get **Sn_RX_RSR** register. More...

Detailed Description

These are functions to access **socket registers**.

Macro Definition Documentation

#define setSn_MR (sn,
mr
) WIZCHIP_WRITE(Sn_MR(sn),mr)

Set **Sn_MR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)mr Value to set **Sn_MR**

See also

getSn_MR()

Definition at line **1890** of file **w5300.h**.

#define getSn_MR (sn) WIZCHIP_READ(Sn_MR(sn))

Get **Sn_MR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_MR**.

See also

setSn_MR()

Definition at line **1900** of file **w5300.h**.

#define

```
setSn_CR ( sn,
           cr
             WIZCHIP_WRITE(Sn_CR(sn), ((uint16_t)cr) &
         ) 0x00FF)
```

Set **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)cr Value to set **Sn_CR**

See also

getSn_CR()

Definition at line **1910** of file **w5300.h**.

```
#define getSn_CR ( sn ) ((uint8_t)WIZCHIP_READ(Sn_CR(sn)))
```

Get **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_CR**.

See also

setSn_CR()

Definition at line **1920** of file **w5300.h**.

```
#define
setSn_IMR ( sn,
            imr
              WIZCHIP_WRITE(Sn_IMR(sn), ((uint16_t)imr)
          ) & 0x00FF)
```

Set **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint8_t)imr Value to set **Sn_IMR**

See also

getSn_IMR()

Definition at line **1930** of file **w5300.h**.

#define
getSn_IMR (sn) ((uint8_t)WIZCHIP_READ(Sn_IMR(sn)))

Get **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_IMR**.

See also

setSn_IMR()

Definition at line **1940** of file **w5300.h**.

#define
setSn_IR (sn,
ir
WIZCHIP_WRITE(Sn_IR(sn), ((uint16_t)ir) &
) 0x00FF)

Set **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint8_t)ir Value to set **Sn_IR**

See also
getSn_IR()

Definition at line **1950** of file **w5300.h**.

#define getSn_IR (sn) ((uint8_t)WIZCHIP_READ(Sn_IR(sn)))

Get **Sn_IR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint8_t. Value of **Sn_IR**.

See also
setSn_IR()

Definition at line **1960** of file **w5300.h**.

#define getSn_SSR (sn) ((uint8_t)WIZCHIP_READ(Sn_SR(sn)))

Get **Sn_SR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint8_t. Value of **Sn_SR**.

Definition at line **1969** of file **w5300.h**.

#define

```
setSn_PORTR ( sn,
              port
                WIZCHIP_WRITE(Sn_PORTR(sn),
            ) port)
```

Set **Sn_PORTR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)port Variable to set **Sn_PORTR**.

See also

getSn_PORTR()

Definition at line **1980** of file **w5300.h**.

Referenced by **close()**.

```
#define getSn_PORTR ( sn,
                      port
                    )   WIZCHIP_READ(Sn_PORTR(sn))
```

Get **Sn_PORTR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Variable of **Sn_PORTR**.

See also

setSn_PORTR()

Definition at line **1991** of file **w5300.h**.

```
#define setSn_DHAR ( sn,
```

dhar
)

Value:

```
{ \
      WIZCHIP_WRITE(Sn_DHAR(sn),
    (((uint16_t)((dhar)[0])) << 8) + (((uint16_t)
    ((dhar)[1])) & 0x00FF)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2)
    (((uint16_t)((dhar)[0])) << 8) + (((uint16_t)
    ((dhar)[1])) & 0x00FF)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4)
    (((uint16_t)((dhar)[0])) << 8) + (((uint16_t)
    ((dhar)[1])) & 0x00FF)); \
  }
```

Set **Sn_DHAR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dhar	Pointer variable to set socket n destination hardwa address. It should be allocated 6 bytes.

See also

getSn_DHAR()

Definition at line **2002** of file **w5300.h**.

#define getSn_DHAR (sn,
dhar
)

Value:

```
{ \
      (dhar)[0] = (uint8_t)
```

```
    (WIZCHIP_READ(Sn_DHAR(sn)) >> 8); \
      (dhar)[1] = (uint8_t)
    WIZCHIP_READ(Sn_DHAR(sn)); \
      (dhar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2)
    >> 8); \
      (dhar)[3] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2))
    \
      (dhar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4)
    >> 8); \
      (dhar)[5] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4))
    \
  }
```

Get **Sn_MR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dhar	Pointer variable to get socket n destination hardwa address. It should be allocated 6 bytes.

See also

setSn_DHAR()

Definition at line **2015** of file **w5300.h**.

#define
setSn_DPORTR (sn,
dport
) WIZCHIP_WRITE(Sn_DPORTR(sn),dport)

Set **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint16_t)dport Value to set **Sn_DPORT**

See also
getSn_DPORT()

Definition at line **2031** of file **w5300.h**.

#define
getSn_DPORTR (sn) WIZCHIP_READ(Sn_DPORTR(sn))

Get **Sn_DPORT** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint16_t. Value of **Sn_DPORT**.

See also
setSn_DPORT()

Note
This function is not available because W5300 have a bug to read **Sn_DPORTR**.
Don't use this function.

Definition at line **2045** of file **w5300.h**.

#define setSn_DIPR (sn,
dipr
)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_DIPR(sn),
    (((uint16_t)((dipr)[0])) << 8) + (((uint16_t)
```

```
    ((dipr)[1])) & 0x00FF)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2)
    (((uint16_t)((dipr)[2])) << 8) + (((uint16_t)
    ((dipr)[3])) & 0x00FF)); \
  }
```

Set **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dipr	Pointer variable to set socket n destination IP address. It should be allocated 4 bytes.

See also

getSn_DIPR()

Definition at line **2056** of file **w5300.h**.

#define getSn_DIPR (sn, dipr)

Value:

```
{ \
        (dipr)[0] = (uint8_t)
    (WIZCHIP_READ(Sn_DIPR(sn)) >> 8); \
        (dipr)[1] = (uint8_t)
    WIZCHIP_READ(Sn_DIPR(sn));  \
        (dipr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2)
    >> 8); \
        (dipr)[3] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2))
      \
  }
```

Get **Sn_DIPR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint8_t*)dipr Pointer variable to get socket n destination IP address. It should be allocated 4 bytes.

See also

setSn_DIPR()

Definition at line **2068** of file **w5300.h**.

```
#define setSn_MSSR (  sn,
                      mss
                   )    WIZCHIP_WRITE(Sn_MSSR(sn), mss)
```

Set **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)mss Value to set **Sn_MSSR**

See also

setSn_MSSR()

Definition at line **2082** of file **w5300.h**.

```
#define getSn_MSSR (  sn )   WIZCHIP_READ(Sn_MSSR(sn))
```

Get **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_MSSR**.

See also
setSn_MSSR()

Definition at line **2092** of file **w5300.h**.

```
#define
setSn_KPALVTR (  sn,
                 kpalvt
                   WIZCHIP_WRITE(Sn_KPALVTR(sn),
                 (WIZCHIP_READ(Sn_KPALVTR(sn)) &
               )  0x00FF) | (((uint16_t)kpalvt)<<8))
```

Set **Sn_KPALVTR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)kpalvt Value to set **Sn_KPALVTR**

See also
getSn_KPALVTR()

Definition at line **2102** of file **w5300.h**.

Referenced by **setsockopt()**.

```
                                ((uint8_t)
#define                        (WIZCHIP_READ(Sn_KPALVTR(sn)) >>
getSn_KPALVTR    (  sn ) 8))
```

Get **Sn_KPALVTR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint8_t. Value of **Sn_KPALVTR**.

See also
setSn_KPALVTR()

Definition at line **2112** of file **w5300.h**.

Referenced by **getsockopt()**, and **setsockopt()**.

#define
setSn_PROTOR (sn,
proto
WIZCHIP_WRITE(Sn_PROTOR(sn),
(WIZCHIP_READ(Sn_PROTOR(sn) & 0xFF00) |
) (((uint16_t)proto) & 0x00FF))

Set **Sn_PROTOR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.
(uint8_t)proto Value to set **Sn_PROTOR**

See also
getSn_PROTOR()

Definition at line **2122** of file **w5300.h**.

#define
getSn_PROTOR (sn) ((uint8_t)WIZCHIP_READ(Sn_PROTOR(sn

Get **Sn_PROTOR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

Returns
uint8_t. Value of **Sn_PROTOR**.

See also
setSn_PROTOR()

Definition at line **2133** of file **w5300.h**.

#define setSn_TX_WRSR (sn, txwrs)

Value:

```
{ \
      WIZCHIP_WRITE(Sn_TX_WRSR(sn),
    (uint16_t)(((uint32_t)txwrs) >> 16)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn)
    (uint16_t)txwrs); \
   }
```

Set **Sn_TX_WRSR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint32_t)txwrs	Value to set **Sn_KPALVTR** (It should be <= 0x00010000)

See also
getSn_TX_WRSR()

Definition at line **2144** of file **w5300.h**.

Referenced by **send()**, and **sendto()**.

#define getSn_TX_WRSR (sn) ((((uint32_t)WIZCHIP_READ(Sn_TX_WR (((uint32_t)WIZCHIP_READ(WIZCHIP_OFF & 0x0000FFFF))

Get **Sn_TX_WRSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint32_t. Value of Sn_TX_WRSR.

See also

setSn_TX_WRSR()

Definition at line **2156** of file **w5300.h**.

```
#define
setSn_TX_FIFOR      (  sn,
                       txfifo
                         WIZCHIP_WRITE(Sn_TX_FIFOR(sn),
                    )  txfifo);
```

Set **Sn_TX_FIFOR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)txfifo. Value to set **Sn_TX_FIFOR**.

Definition at line **2181** of file **w5300.h**.

```
#define
getSn_RX_FIFOR    (  sn )   WIZCHIP_READ(Sn_RX_FIFOR(sn));
```

Get **Sn_RX_FIFOR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_RX_FIFOR**.

Definition at line **2190** of file **w5300.h**.

```
#define
setSn_TOSR      (  sn,
                   tos
                     WIZCHIP_WRITE(Sn_TOS(sn),
                )  ((uint16_t)tos) & 0x00FF)
```

Set **Sn_TOSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_**.

(uint8_t)tos Value to set **Sn_TOSR**

See also

getSn_TOSR()

Definition at line **2200** of file **w5300.h**.

```
#define
getSn_TOSR     (  sn )    ((uint8_t)WIZCHIP_READ(Sn_TOSR(sn)))
```

Get **Sn_TOSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ _WIZCHIP_SOCK_NUM_** .

Returns

uint8_t. Value of Sn_TOSR.

See also

setSn_TOSR()

Definition at line **2211** of file **w5300.h**.

```
#define
setSn_TTLR      (  sn,
                   ttl
                     WIZCHIP_WRITE(Sn_TTLR(sn),
                )  ((uint16_t)ttl) & 0x00FF)
```

Set **Sn_TTLR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)ttl Value to set **Sn_TTLR**

See also

getSn_TTLR()

Definition at line **2222** of file **w5300.h**.

```
#define
getSn_TTLR      (  sn )   ((uint8_t)WIZCHIP_READ(Sn_TTL(sn)))
```

Get **Sn_TTLR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_TTLR**.

See also

setSn_TTLR()

Definition at line **2233** of file **w5300.h**.

```
#define
```

```
setSn_FRAGR     (  sn,
                   frag
                     WIZCHIP_WRITE(Sn_FRAGR(sn),
                ) (uint16_t)(frag >>8))
```

Set **Sn_FRAGR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)frag Value to set **Sn_FRAGR**

See also

getSn_FRAGR()

Definition at line **2244** of file **w5300.h**.

```
#define                          (WIZCHIP_READ(Sn_FRAG(sn)) <<
getSn_FRAGR          (  sn ) 8)
```

Get **Sn_FRAGR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_FRAGR**.

See also

setSn_FRAGR()

Definition at line **2255** of file **w5300.h**.

Function Documentation

uint32_t getSn_TX_FSR (uint8_t sn)

Get **Sn_TX_FSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint32_t. Value of **Sn_TX_FSR**.

uint32_t getSn_RX_RSR (uint8_t sn)

Get **Sn_RX_RSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint32_t. Value of **Sn_RX_RSR**.

Socket APIs

WIZCHIP register

W5300

WHIZCHIP register defines register group of **W5300**. More...

Modules

Common register
Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

Socket register
Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication.

Detailed Description

WHIZCHIP register defines register group of **W5300**.

- **Common register** : Common register group
- **Socket register** : SOCKET n register group

Socket APIs

Common register

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc. More...

Macros

#define **MR** (**_WIZCHIP_IO_BASE_**)
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...

#define **IR** (**_W5300_IO_BASE_** + 0x02)
Interrupt Register(R/W) More...

#define **_IMR_** (**_W5300_IO_BASE_** + 0x04)
Socket Interrupt Mask Register(R/W) More...

#define **SHAR** (**_W5300_IO_BASE_** + 0x08)
Source MAC Register address(R/W) More...

#define **GAR** (**_W5300_IO_BASE_** + 0x10)
Gateway IP Register address(R/W) More...

#define **SUBR** (**_W5300_IO_BASE_** + 0x14)
Subnet mask Register address(R/W) More...

#define **SIPR** (**_W5300_IO_BASE_** + 0x18)
Source IP Register address(R/W) More...

#define **_RTR_** (**_W5300_IO_BASE_** + 0x1C)
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5300_IO_BASE_** + 0x1E)
Retry count register(R/W) More...

#define **TMS01R** (**_W5300_IO_BASE_** + 0x20)
TX memory size of SOCKET 0 & 1. More...

#define **TMS23R** (**TMS01R** + 2)

TX memory size of SOCKET 2 & 3. More...

#define **TMS45R** (**TMS01R** + 4)
TX memory size of SOCKET 4 & 5. More...

#define **TMS67R** (**TMS01R** + 6)
TX memory size of SOCKET 6 & 7. More...

#define **TMSR0** **TMS01R**
TX memory size of SOCKET 0. More...

#define **TMSR1** (**TMSR0** + 1)
TX memory size of SOCKET 1. More...

#define **TMSR2** (**TMSR0** + 2)
TX memory size of SOCKET 2. More...

#define **TMSR3** (**TMSR0** + 3)
TX memory size of SOCKET 3. More...

#define **TMSR4** (**TMSR0** + 4)
TX memory size of SOCKET 4. More...

#define **TMSR5** (**TMSR0** + 5)
TX memory size of SOCKET 5. More...

#define **TMSR6** (**TMSR0** + 6)
TX memory size of SOCKET 6. More...

#define **TMSR7** (**TMSR0** + 7)
TX memory size of SOCKET 7. More...

#define **RMS01R** (**_W5300_IO_BASE_** + 0x28)
RX memory size of SOCKET 0 & 1. More...

#define **RMS23R** (**RMS01R** + 2)

RX memory size of SOCKET 2 & 3. More...

#define **RMS45R** (**RMS01R** + 4)
RX memory size of SOCKET 4 & 5. More...

#define **RMS67R** (**RMS01R** + 6)
RX memory size of SOCKET 6 & 7. More...

#define **RMSR0** **RMS01R**
RX memory size of SOCKET 0. More...

#define **RMSR1** (**RMSR0** + 1)
RX memory size of SOCKET 1. More...

#define **RMSR3** (**RMSR0** + 3)
RX memory size of SOCKET 3. More...

#define **RMSR4** (**RMSR0** + 4)
RX memory size of SOCKET 4. More...

#define **RMSR5** (**RMSR0** + 5)
RX memory size of SOCKET 5. More...

#define **RMSR6** (**RMSR0** + 6)
RX memory size of SOCKET 6. More...

#define **RMSR7** (**RMSR0** + 7)
RX memory size of SOCKET 7. More...

#define **MTYPER** (**_W5300_IO_BASE_** + 0x30)
Memory Type Register. More...

#define **PATR** (**_W5300_IO_BASE_** + 0x32)

PPPoE Authentication Type register. More...

#define **PTIMER** (**_W5300_IO_BASE_** + 0x36)
PPP Link Control Protocol Request Timer Register. More...

#define **PMAGICR** (**_W5300_IO_BASE_** + 0x38)
PPP LCP magic number register. More...

#define **PSIDR** (**_W5300_IO_BASE_** + 0x3C)
PPPoE session ID register. More...

#define **PDHAR** (**_W5300_IO_BASE_** + 0x40)
PPPoE destination hardware address register. More...

#define **UIPR** (**_W5300_IO_BASE_** + 0x48)
Unreachable IP address register. More...

#define **UPORTR** (**_W5300_IO_BASE_** + 0x4C)
Unreachable port number register. More...

#define **FMTUR** (**_W5300_IO_BASE_** + 0x4E)
Fragment MTU register. More...

#define **Pn_BRDYR**(n) (**_W5300_IO_BASE_** + 0x60 + n*4)
PIN 'BRDYn' configure register. More...

#define **Pn_BDPTHR**(n) (**_W5300_IO_BASE_** + 0x60 + n*4 + 2)
PIN 'BRDYn' buffer depth Register. More...

#define **IDR** (**_W5300_IO_BASE_** + 0xFE)
W5300 identification register. More...

Detailed Description

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

See also

MR : Mode register.

GAR, **SUBR**, **SHAR**, **SIPR** : Network Configuration

IR, *IMR* : Interrupt.

MTYPER, **TMS01R**,**TMS23R**, **TMS45R**, **TMS67R**,**RMS01R**,**RMS23R**, **RMS45R**, **RMS67R** : Socket TX/RX memory

RTR, *RCR* : Data retransmission.

PTIMER, **PMAGIC**, **PSID**, **PDHAR** : PPPoE.

UIPR, **UPORTR**, **FMTUR** : ICMP message.

Pn_BRDYR, **Pn_BDPTHR**, **IDR** : etc.

Macro Definition Documentation

#define MR (_WIZCHIP_IO_BASE_)

Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc.

Each bit of **MR** defined as follows.

15	14	13	12	11	10	9	8
DBW	MPF	WDF			RDF	Reserved	FS
7	6	5	4	3	2	1	0
RST	Reserved	WOL	PB	PPPoE	Reserved	FARP	Reserve

- **MR_DBW** : Data bus width (0 : 8 Bit, 1 : 16 Bit), Read Only
- **MR_MPF** : Received a Pause Frame from MAC layer (0 : Normal Frame, 1 : Pause Frame), Read Only
- **MR_WDF** : Write Data Fetch time (When CS signal is low, W5300 Fetch a written data by Host after PLL_CLK * MR_WDF)
- **MR_RDH** : Read Data Hold time (0 : No use data hold time, 1 : Us data hold time, 2 PLL_CLK)
- **MR_FS** : FIFO Swap (0 : Disable Swap, 1 : Enable Swap)
- **MR_RST** : Reset
- **MR_WOL** : Wake on LAN
- **MR_PB** : Ping block
- **MR_PPPOE** : PPPoE mode
- **MR_FARP** : Force ARP mode

Definition at line **224** of file **w5300.h**.

#define IR (_W5300_IO_BASE_ + 0x02)

Interrupt Register(R/W)

IR indicates the interrupt status. Each bit of **IR** will be still until the bit wi
to by the host. If **IR** is not equal to 0x0000 INTn PIN is asserted to low u
0x0000

Each bit of **IR** defined as follows.

15	14	13	12	11	10	9
IPCF	DPUR	PPPT	FMTU	Reserved	Reserved	Reserved
7	6	5	4	3	2	1
S7_INT	S6_INT	S5_INT	S4_INT	S3_INT	S2_INT	S1_INT

- **IR_IPCF** : IP conflict
- **IR_DPUR** : Destination Port Unreachable
- **IR_PPPT** : PPPoE Termination
- **IR_FMTU** : Fragmented MTU
- **IR_SnINT(n)** : Interrupted from SOCKETn

Note
: In W5300, IR is operated same as IR and SIR in other WIZCHIP(5100,5200,W5500)

Definition at line **246** of file **w5300.h**.

#define _IMR_ (_W5300_IO_BASE_ + 0x04)

Socket Interrupt Mask Register(R/W)

Each bit of *IMR* corresponds to each bit of **IR**. When a bit of *IMR* is and the corresponding bit of **IR** is Interrupt will be issued. In other words, if a bit of *IMR*, an interrupt will be not issued even if the corresponding bit of **IR** is set

Note
: In W5300, *IMR* is operated same as *IMR* and SIMR in other WIZCHIP(5100,5200,W5500)

Definition at line **256** of file **w5300.h**.

#define SHAR (_W5300_IO_BASE_ + 0x08)

Source MAC Register address(R/W)

SHAR configures the source hardware address.

Definition at line **267** of file **w5300.h**.

#define GAR (_W5300_IO_BASE_ + 0x10)

Gateway IP Register address(R/W)

GAR configures the default gateway address.

Definition at line **275** of file **w5300.h**.

#define SUBR (_W5300_IO_BASE_ + 0x14)

Subnet mask Register address(R/W)

SUBR configures the subnet mask address.

Definition at line **282** of file **w5300.h**.

#define SIPR (_W5300_IO_BASE_ + 0x18)

Source IP Register address(R/W)

SIPR configures the source IP address.

Definition at line **289** of file **w5300.h**.

#define _RTR_ (_W5300_IO_BASE_ + 0x1C)

Timeout register address(1 is 100us)(R/W)

RTR configures the retransmission timeout period. The unit of timeout period is 100us and the default of *RTR* is x07D0. And so the default timeout period is 200ms(100us X 2000). During the time configured by *RTR*, W5300 waits for the peer response to the packet that is transmitted by **Sn_CR** (CONNECT, DISCON, CLOSE, SEND, SEND_MAC, SEND_KEEP command). If the peer does not respond within the *RTR* time, W5300 retransmits the packet or issues timeout.

Definition at line **299** of file **w5300.h**.

#define _RCR_ (_W5300_IO_BASE_ + 0x1E)

Retry count register(R/W)

RCR configures the number of time of retransmission. When retransmission occurs as many as ref *RCR*+1 Timeout interrupt is issued (**Sn_IR_TIMEOUT** = '1').

Definition at line **307** of file **w5300.h**.

#define TMS01R (_W5300_IO_BASE_ + 0x20)

TX memory size of SOCKET 0 & 1.

TMS01R configures the TX buffer block size of SOCKET 0 & 1. The default value is configured with 8KB and can be configure from 0 to 64KB with unit 1KB. But the sum of all SOCKET TX buffer size should be multiple of 8 and the sum of all SOCKET TX and RX memory size can't exceed 128KB. When exceeded nor multiple of 8, the data transmittion is invalid.

Definition at line **316** of file **w5300.h**.

#define TMS23R (TMS01R + 2)

TX memory size of SOCKET 2 & 3.

refer to **TMS01R**

Definition at line **323** of file **w5300.h**.

#define TMS45R (TMS01R + 4)

TX memory size of SOCKET 4 & 5.

refer to **TMS01R**

Definition at line **330** of file **w5300.h**.

#define TMS67R (TMS01R + 6)

TX memory size of SOCKET 6 & 7.

refer to **TMS01R**

Definition at line **337** of file **w5300.h**.

#define TMSR0 TMS01R

TX memory size of SOCKET 0.

refer to **TMS01R**

Definition at line **344** of file **w5300.h**.

#define TMSR1 (TMSR0 + 1)

TX memory size of SOCKET 1.

refer to **TMS01R**

Definition at line **351** of file **w5300.h**.

#define TMSR2 (TMSR0 + 2)

TX memory size of SOCKET 2.

refer to **TMS01R**

Definition at line **358** of file **w5300.h**.

#define TMSR3 (TMSR0 + 3)

TX memory size of SOCKET 3.

refer to **TMS01R**

Definition at line **365** of file **w5300.h**.

#define TMSR4 (TMSR0 + 4)

TX memory size of SOCKET 4.

refer to **TMS01R**

Definition at line **372** of file **w5300.h**.

#define TMSR5 (TMSR0 + 5)

TX memory size of SOCKET 5.

refer to **TMS01R**

Definition at line **379** of file **w5300.h**.

#define TMSR6 (TMSR0 + 6)

TX memory size of SOCKET 6.

refer to **TMS01R**

Definition at line **386** of file **w5300.h**.

#define TMSR7 (TMSR0 + 7)

TX memory size of SOCKET 7.

refer to **TMS01R**

Definition at line **393** of file **w5300.h**.

#define RMS01R (_W5300_IO_BASE_ + 0x28)

RX memory size of SOCKET 0 & 1.

RMS01R configures the RX buffer block size of SOCKET 0 & 1. The default value is configured with 8KB and can be configure from 0 to 64KB with unit 1KB. But the sum of all SOCKET RX buffer size should be multiple of 8 and the sum of all SOCKET RX and TX memory size can't exceed 128KB. When exceeded nor multiple of 8, the data reception is invalid.

Definition at line **403** of file **w5300.h**.

#define RMS23R (RMS01R + 2)

RX memory size of SOCKET 2 & 3.

Refer to **RMS01R**

Definition at line **410** of file **w5300.h**.

#define RMS45R (RMS01R + 4)

RX memory size of SOCKET 4 & 5.

Refer to **RMS01R**

Definition at line **417** of file **w5300.h**.

#define RMS67R (RMS01R + 6)

RX memory size of SOCKET 6 & 7.

Refer to **RMS01R**

Definition at line **424** of file **w5300.h**.

#define RMSR0 RMS01R

RX memory size of SOCKET 0.

refer to **RMS01R**

Definition at line **431** of file **w5300.h**.

#define RMSR1 (RMSR0 + 1)

RX memory size of SOCKET 1.

refer to **RMS01R**

Definition at line **438** of file **w5300.h**.

#define RMSR3 (RMSR0 + 3)

RX memory size of SOCKET 3.

refer to **RMS01R**

Definition at line **452** of file **w5300.h**.

#define RMSR4 (RMSR0 + 4)

RX memory size of SOCKET 4.

refer to **RMS01R**

Definition at line **459** of file **w5300.h**.

#define RMSR5 (RMSR0 + 5)

RX memory size of SOCKET 5.

refer to **RMS01R**

Definition at line **466** of file **w5300.h**.

#define RMSR6 (RMSR0 + 6)

RX memory size of SOCKET 6.

refer to **RMS01R**

Definition at line **473** of file **w5300.h**.

#define RMSR7 (RMSR0 + 7)

RX memory size of SOCKET 7.

refer to **RMS01R**

Definition at line **480** of file **w5300.h**.

#define MTYPER (_W5300_IO_BASE_ + 0x30)

Memory Type Register.

W5300's 128Kbytes data memory (Internal TX/RX memory) is composed of 16 memory blocks of 8Kbytes. MTYPER configures type of each 8KB memory block in order to select RX or TX memory. The type of 8KB memory block corresponds to each bit of MTYPER. When the bit is '1', it is used as TX memory, and the bit is '0', it is used as RX memory. MTYPER is configured as TX memory type from the lower bit. The rest of the bits not configured as TX memory, should be set as '0'.

Definition at line **493** of file **w5300.h**.

#define PATR (_W5300_IO_BASE_ + 0x32)

PPPoE Authentication Type register.

It notifies authentication method negotiated with PPPoE server. W5300 supports 2 types of authentication methods.

- PAP : 0xC023
- CHAP : 0xC223

Definition at line **503** of file **w5300.h**.

#define PTIMER (_W5300_IO_BASE_ + 0x36)

PPP Link Control Protocol Request Timer Register.

It configures transmitting timer of link control protocol (LCP) echo request. Value 1 is about 25ms.

Definition at line **512** of file **w5300.h**.

#define PMAGICR (_W5300_IO_BASE_ + 0x38)

PPP LCP magic number register.

It configures byte value to be used for 4bytes “Magic Number” during LCP negotiation with PPPoE server.

Definition at line **519** of file **w5300.h**.

#define PSIDR (_W5300_IO_BASE_ + 0x3C)

PPPoE session ID register.

It notifies PPP session ID to be used for communication with PPPoE server (acquired by PPPoE-process of W5300).

Definition at line **528** of file **w5300.h**.

#define PDHAR (_W5300_IO_BASE_ + 0x40)

PPPoE destination hardware address register.

It notifies hardware address of PPPoE server (acquired by PPPoE-process of W5300).

Definition at line **535** of file **w5300.h**.

#define UIPR (_W5300_IO_BASE_ + 0x48)

Unreachable IP address register.

When trying to transmit UDP data to destination port number which is not open, W5300 can receive ICMP (Destination port unreachable) packet.
In this case, **IR_DPUR** bit of **IR** becomes '1'. And destination IP address and unreachable port number of ICMP packet can be acquired through UIPR and **UPORTR**.

Definition at line **545** of file **w5300.h**.

#define UPORTR (_W5300_IO_BASE_ + 0x4C)

Unreachable port number register.

Refer to **UIPR**.

Definition at line **552** of file **w5300.h**.

#define FMTUR (_W5300_IO_BASE_ + 0x4E)

Fragment MTU register.

When communicating with the peer having a different MTU, W5300 can receive an ICMP(Fragment MTU) packet. At this case, **IR(FMTU)** becomes ‘1’ and destination IP address and fragment MTU value of ICMP packet can be acquired through UIPR and FMTUR. In order to keep communicating with the peer having Fragment MTU, set the FMTUR first in Sn_MSSR of the SOCKETn, and try the next communication.

Definition at line **561** of file **w5300.h**.

#define Pn_BRDYR (n) (_W5300_IO_BASE_ + 0x60 + n*4)

PIN 'BRDYn' configure register.

It configures the PIN "BRDYn" which is monitoring TX/RX memory status of the specified SOCKET. If the free buffer size of TX memory is same or bigger than the buffer depth of **Pn_BDPTHR**, or received buffer size of RX memory is same or bigger than the **Pn_BDPTHR**, PIN "BRDYn" is signaled.

15	14	13	12	11	10	9	8
Reserved, Read as 0							
7	6	5	4	3	2	1	0
PEN	MT	PPL	Reserved		SN		

- **Pn_PEN** Enable PIN 'BRDYn' (0 : Disable, 1 : Enable)
- **Pn_MT** Monitoring Memory type (0 : RX memory, 1 : TX Memory)
- **Pn_PPL** PIN Polarity bit of Pn_BRDYR. (0 : Low sensitive, 1 : High sensitive)
- **Pn_SN(n)** Monitoring SOCKET number of Pn_BRDYR

Definition at line **584** of file **w5300.h**.

#define Pn_BDPTHR (_W5300_IO_BASE_ + 0x60 + n*4 + 2)

PIN 'BRDYn' buffer depth Register.

It configures buffer depth of PIN "BRDYn". When monitoring TX memory and **Sn_TX_FSR** is same or bigger than Pn_BDPTHR, the PIN "BRDYn" is signaled. When monitoring RX memory and if **Sn_RX_RSR** is same or bigger than Pn_BDPTHR, PIN "BRDYn" is signaled. The value for Pn_BDPTHR can't exceed TX/RX memory size allocated by TMSR or RMSR such like as **TMS01R** or **RMS01R**.

Definition at line **594** of file **w5300.h**.

#define IDR (_W5300_IO_BASE_ + 0xFE)

W5300 identification register.

Read Only. 0x5300.

Definition at line **601** of file **w5300.h**.

Socket APIs

Socket register

Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication. More...

Macros

#define	**Sn_MR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x00) Socket Mode register(R/W) More...
#define	**Sn_CR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x02) Socket command register(R/W) More...
#define	**Sn_IMR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x04) socket interrupt mask register(R) More...
#define	**Sn_IR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x06) Socket interrupt register(R) More...
#define	**Sn_SSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x08) Socket status register(R) More...
#define	**Sn_PORTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x0A) source port register(R/W) More...
#define	**Sn_DHAR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x0C) Peer MAC register address(R/W) More...
#define	**Sn_DPORTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x12) Peer port register address(R/W) More...
#define	**Sn_DIPR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x14)

Peer IP register address(R/W) More...

#define **Sn_MSSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x18)
Maximum Segment Size(Sn_MSSR0) register address(R/W) More...

#define **Sn_KPALVTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x1A)
Keep Alive Timer register(R/W) More...

#define **Sn_PROTOR**(n) **Sn_KPALVTR**(n)
IP Protocol(PROTO) Register(R/W) More...

#define **Sn_TOSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x1C)
IP Type of Service(TOS) Register(R/W) More...

#define **Sn_TTLR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x1E)
IP Time to live(TTL) Register(R/W) More...

#define **Sn_TX_WRSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x20)
SOCKETn TX write size register(R/W) More...

#define **Sn_TX_FSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x0024)
Transmit free memory size register(R) More...

#define **Sn_FRAGR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x002C)
Fragment field value in IP header register(R/W) More...

#define **Sn_TX_FIFOR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x2E)
SOCKET n TX FIFO regsiter. More...

#define	**Sn_RX_FIFOR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(n) + 0x30) SOCKET n RX FIFO register. More...

Detailed Description

Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication.

See also

Sn_MR, **Sn_CR**, **Sn_IR**, **Sn_IMR** : SOCKETn Control

Sn_SR, **Sn_PORT**, **Sn_DHAR**, **Sn_DIPR**, **Sn_DPORT** : SOCKETn Information

Sn_MSSR, **Sn_TOS**, **Sn_TTL**, **Sn_KPALVTR**, **Sn_FRAG** : Internet protocol.

Sn_TX_WRSR, **Sn_TX_FSR**, **Sn_TX_RD**, **Sn_TX_WR**, **Sn_RX_RSR**, **Sn_RX_RD**, **Sn_RX_WR**, **Sn_TX_FIFOR**, **Sn_RX_FIFOR** : Data communication

Macro Definition Documentation

#define Sn_MR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x00)

Socket Mode register(R/W)

Sn_MR configures the option or protocol type of Socket n.

Each bit of **Sn_MR** defined as the following.

15	14	13	12	11	10	9	8
Reserved. Read as 0							ALIGN
7	6	5	4	3	2	1	0
MULTI	MF	ND/IGMPv	Reserved	PROTOCOL[3:0]			

- **Sn_MR_ALIGN** : Alignment bit of Sn_MR, Only valid in **Sn_MR_TCP**. (C0 : Include TCP PACK_INFO, 1 : Not include TCP PACK_INFO)
- **Sn_MR_MULTI** : Support UDP Multicasting
- **Sn_MR_MF** : Enable MAC Filter (0 : Disable, 1 - Enable), When enabled, W5300 can receive only both own and broadcast packet.
- **Sn_MR_ND** : No Delayed Ack(TCP) flag
- **Sn_MR_IGMPv** : IGMP version used **in UDP mulitcasting**. (0 : Version 2, 1 : Version 2)
- **PROTOCOL[3:0]**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	0	1	1	IPCRAW
0	1	0	0	MACRAW
0	1	0	1	PPPoE

- **Sn_MR_PPPoE** : PPPoE
  - **Sn_MR_MACRAW** : MAC LAYER RAW SOCK
- **Sn_MR_IPRAW** : IP LAYER RAW SOCK
- **Sn_MR_UDP** : UDP
- **Sn_MR_TCP** : TCP
- **Sn_MR_CLOSE** : Unused socket

Note

MACRAW mode should be only used in Socket 0.

Definition at line **642** of file **w5300.h**.

#define Sn_CR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x02)

Socket command register(R/W)

This is used to set the command for Socket n such as OPEN, CLOSE, CONNECT, LISTEN, SEND, and RECEIVE.
After W5500 accepts the command, the **Sn_CR** register is automatically cleared to 0x00. Even though **Sn_CR** is cleared to 0x00, the command is still being processed.
To check whether the command is completed or not, please check the **Sn_IR** or **Sn_SR**.

- **Sn_CR_OPEN** : Initialize or open socket.
- **Sn_CR_LISTEN** : Wait connection request in TCP mode(**Server mode**)
- **Sn_CR_CONNECT** : Send connection request in TCP mode(**Client mode**)
- **Sn_CR_DISCON** : Send closing request in TCP mode.
- **Sn_CR_CLOSE** : Close socket.
- **Sn_CR_SEND** : Update TX buffer pointer and send data.
- **Sn_CR_SEND_MAC** : Send data with MAC address, so without ARP process.
- **Sn_CR_SEND_KEEP** : Send keep alive message.
- **Sn_CR_RECV** : Update RX buffer pointer and receive data.
- **Sn_CR_PCON** : PPPoE connection begins by transmitting PPPoE discovery packet.
- **Sn_CR_PDISCON** : Closes PPPoE connection.

- **Sn_CR_PCR** : In each phase, it transmits REQ message.
- **Sn_CR_PCN** : In each phase, it transmits NAK message.
- **Sn_CR_PCJ** : In each phase, it transmits REJECT message.

Definition at line **666** of file **w5300.h**.

#define Sn_IMR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x04)

socket interrupt mask register(R)

Sn_IMR masks the interrupt of Socket n. Each bit corresponds to each bit of **Sn_IR**. When a Socket n Interrupt is occurred and the corresponding bit of **Sn_IMR** is the corresponding bit of **Sn_IR** becomes When both the corresponding bit of **Sn_IMR** and **Sn_IR** are and the n-th bit of **IR** is Host is interrupted by asserted INTn PIN to low.

Definition at line **676** of file **w5300.h**.

#define Sn_IR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + 0x06)

Socket interrupt register(R)

Sn_IR indicates the status of Socket Interrupt such as establishment, termination, receiving data, timeout).
When an interrupt occurs and the corresponding bit of **Sn_IMR** is the corresponding bit of **Sn_IR** becomes
In order to clear the **Sn_IR** bit, the host should write the bit to

15	14	13	12	11	10	9	8
Reserved. Read as 0							
7	6	5	4	3	2	1	0
PRECV	PFAIL	PNEXT	SENDOK	TIMEOUT	RECV	DISCON	C

- **Sn_IR_PRECV** : PPP receive

- **Sn_IR_PFAIL** : PPP fail
- **Sn_IR_PNEXT** : PPP next phase
- **Sn_IR_SENDOK** : SENDOK
- **Sn_IR_TIMEOUT** : TIMEOUT
- **Sn_IR_RECV** : RECV
- **Sn_IR_DISCON** : DISCON
- **Sn_IR_CON** : CON

Definition at line **699** of file **w5300.h**.

#define Sn_SSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x08)

Socket status register(R)

Sn_SSR indicates the status of Socket n.
The status of Socket n is changed by **Sn_CR** or some special control packet as SYN, FIN packet in TCP.

Normal status

- **SOCK_CLOSED** : Closed
- **SOCK_INIT** : Initiate state
- **SOCK_LISTEN** : Listen state
- **SOCK_ESTABLISHED** : Success to connect
- **SOCK_CLOSE_WAIT** : Closing state
- **SOCK_UDP** : UDP socket
- **SOCK_IPRAW** : IPRAW socket
- **SOCK_MACRAW** : MAC raw mode socket
- **SOCK_PPPoE** : PPPoE mode Socket

Temporary status during changing the status of Socket n.

- **SOCK_SYNSENT** : This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
- **SOCK_SYNRECV** : It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
- **SOCK_FIN_WAIT** : Connection state
- **SOCK_CLOSING** : Closing state
- **SOCK_TIME_WAIT** : Closing state

- **SOCK_LAST_ACK** : Closing state
- **SOCK_ARP** : ARP request state

Definition at line **725** of file **w5300.h**.

#define Sn_PORTR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0A)

source port register(R/W)

Sn_PORTR configures the source port number of Socket n. It is valid when Socket n is used in TCP/UPD mode. It should be set before OPEN command is ordered.

Definition at line **734** of file **w5300.h**.

#define Sn_DHAR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0C)

Peer MAC register address(R/W)

Sn_DHAR configures the destination hardware address of Socket n when using SEND_MAC command in UDP mode or it indicates that it is acquired in ARP-process by CONNECT/SEND command.

Definition at line **743** of file **w5300.h**.

#define Sn_DPORTR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x12)

Peer port register address(R/W)

Sn_DPORTR configures or indicates the destination port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP clientmode, it configures the listen port number of TCP serverbefore CONNECT command. In TCP Servermode, it indicates

the port number of TCP client after successfully establishing connection. In UDP mode, it configures the port number of peer to be transmitted the UDP packet by SEND/SEND_MAC command.

Definition at line **753** of file **w5300.h**.

#define Sn_DIPR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x14)

Peer IP register address(R/W)

Sn_DIPR configures or indicates the destination IP address of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP client mode, it configures an IP address of TCP serverbefore CONNECT command. In TCP server mode, it indicates an IP address of TCP clientafter successfully establishing connection. In UDP mode, it configures an IP address of peer to be received the UDP packet by SEND or SEND_MAC command.

Definition at line **765** of file **w5300.h**.

#define Sn_MSSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x18)

Maximum Segment Size(Sn_MSSR0) register address(R/W)

Sn_MSSR configures or indicates the MTU(Maximum Transfer Unit) of Socket n.

Definition at line **772** of file **w5300.h**.

#define Sn_KPALVTR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1A)

Keep Alive Timer register(R/W)

Sn_KPALVTR configures the transmitting timer of KEEP ALIVE(KA)packet of SOCKETn. It is valid only in TCP mode, and ignored in other modes. The time unit is 5s. KA packet is transmittable after **Sn_SR** is changed to SOCK_ESTABLISHED and after the data is transmitted or received to/from a peer at least once. In case of '**Sn_KPALVTR** > 0', W5500 automatically transmits KA packet after time-period for checking the TCP connection (Auto-keepalive-process). In case of '**Sn_KPALVTR** = 0', Auto-keep-alive-process will not operate, and KA packet can be transmitted by SEND_KEEP command by the host (Manual-keep-alive-process). Manual-keep-alive-process is ignored in case of '**Sn_KPALVTR** > 0'.

Definition at line **785** of file **w5300.h**.

#define Sn_PROTOR (n) Sn_KPALVTR(n)

IP Protocol(PROTO) Register(R/W)

Sn_PROTO that sets the protocol number field of the IP header at the IP layer. It is valid only in IPRAW mode, and ignored in other modes.

Definition at line **793** of file **w5300.h**.

#define Sn_TOSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1C)

IP Type of Service(TOS) Register(R/W)

Sn_TOSR configures the TOS(Type Of Service field in IP Header) of Socket n. It is set before OPEN command.

Definition at line **802** of file **w5300.h**.

#define Sn_TTLR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1E)

IP Time to live(TTL) Register(R/W)

Sn_TTLR configures the TTL(Time To Live field in IP header) of Socket n. It is set before OPEN command.

Definition at line **811** of file **w5300.h**.

#define Sn_TX_WRSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x20)

SOCKETn TX write size register(R/W)

It sets the byte size of the data written in internal TX memory through **Sn_TX_FIFOR**. It is set before SEND or SEND_MAC command, and can't be bigger than internal TX memory size set by TMSR such as **TMS01R**, TMS23R and etc.

Definition at line **821** of file **w5300.h**.

#define Sn_TX_FSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0024)

Transmit free memory size register(R)

Sn_TX_FSR indicates the free size of Socket n TX Buffer Block. It is initialized to the configured size by TMSR such as TMS01SR. Data bigger than Sn_TX_FSR should not be saved in the Socket n TX Buffer because the bigger data overwrites the previous saved data not yet sent. Therefore, check before saving the data to the Socket n TX Buffer, and if data is equal or smaller than its checked size, transmit the data with SEND/SEND_MAC command after saving the data in Socket n TX buffer. But, if data is bigger than its checked size, transmit the data after dividing into the checked size and saving in the Socket n TX buffer.

Definition at line **832** of file **w5300.h**.

#define Sn_FRAGR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x002C)

Fragment field value in IP header register(R/W)

Sn_FRAGR configures the FRAG(Fragment field in IP header).

Definition at line **848** of file **w5300.h**.

#define Sn_TX_FIFOR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x2E)

SOCKET n TX FIFO regsiter.

It indirectly accesses internal TX memory of SOCKETn. The internal TX memory can't be accessed directly by the host, but can be accessed through Sn_TX_FIFOR. If **MR(MT)** = '0', only the Host-Write of internal TX memory is allowed through Sn_TX_FIFOR. But if **MR(MT)** is '1', both of Host-Read and Host-Write are allowed.

Definition at line **859** of file **w5300.h**.

#define Sn_RX_FIFOR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x30)

SOCKET n RX FIFO register.

It indirectly accesses to internal RX memory of SOCKETn. The internal RX memory can't be directly accessed by the host, but can be accessed through Sn_RX_FIFOR. If **MR(MT)** = '0', only the Host-Read of internal RX memory is allowed through Sn_RX_FIFOR. But if **MR(MT)** is '1', both of Host-Read and Host-Write are allowed.

Definition at line **869** of file **w5300.h**.

Socket APIs

W5500

WHIZCHIP register defines and I/O functions of **W5500**. More...

Modules

WIZCHIP I/O functions
This supports the basic I/O functions for **WIZCHIP register**.

WIZCHIP register
WHIZCHIP register defines register group of **W5500**.

Detailed Description

WHIZCHIP register defines and I/O functions of **W5500**.

- **WIZCHIP register** : **Common register** and **Socket register**
- **WIZCHIP I/O functions** : **Basic I/O function**, **Common register access functions** and **Socket register access functions**

Socket APIs

WIZCHIP I/O functions

W5500

This supports the basic I/O functions for **WIZCHIP register**. More...

Modules

Basic I/O function
These are basic input/output functions to read values from register or write values to register.

Common register access functions
These are functions to access **common registers**.

Socket register access functions
These are functions to access **socket registers**.

Detailed Description

This supports the basic I/O functions for **WIZCHIP register**.

- **Basic I/O function**
  WIZCHIP_READ(), **WIZCHIP_WRITE()**, **WIZCHIP_READ_BUF()**, **WIZCHIP_WRITE_BUF()**

- **Common register access functions**
  1. **Mode**
     getMR(), **setMR()**
  2. **Interrupt**
     getIR(), **setIR()**, **getIMR()**, **setIMR()**, **getSIR()**, **setSIR()**, **getSIMR()**, **setSIMR()**, **getINTLEVEL()**, **setINTLEVEL()**
  3. **Network Information**
     getSHAR(), **setSHAR()**, **getGAR()**, **setGAR()**, **getSUBR()**, **setSUBR()**, **getSIPR()**, **setSIPR()**
  4. **Retransmission**
     getRCR(), **setRCR()**, **getRTR()**, **setRTR()**
  5. **PPPoE**
     getPTIMER(), **setPTIMER()**, **getPMAGIC()**, **getPMAGIC()**, **getPSID()**, **setPSID()**, **getPHAR()**, **setPHAR()**, **getPMRU()**, **setPMRU()**
  6. **ICMP packet**
     getUIPR(), **getUPORTR()**
  7. **etc**.
     getPHYCFGR(), **setPHYCFGR()**, **getVERSIONR()**

- **Socket register access functions**
  1. **SOCKET control**
     getSn_MR(), **setSn_MR()**, **getSn_CR()**, **setSn_CR()**, **getSn_IMR()**, **setSn_IMR()**, **getSn_IR()**, **setSn_IR()**
  2. **SOCKET information**
     getSn_SR(), **getSn_DHAR()**, **setSn_DHAR()**, **getSn_PORT()**, **setSn_PORT()**, **getSn_DIPR()**, **setSn_DIPR()**, **getSn_DPORT()**, **setSn_DPORT()**
     getSn_MSSR(), **setSn_MSSR()**

3. **SOCKET communication**
   getSn_RXBUF_SIZE(), **setSn_RXBUF_SIZE()**,
   getSn_TXBUF_SIZE(), **setSn_TXBUF_SIZE()**
   getSn_TX_RD(), **getSn_TX_WR()**, **setSn_TX_WR()**
   getSn_RX_RD(), **setSn_RX_RD()**, **getSn_RX_WR()**
   getSn_TX_FSR(), **getSn_RX_RSR()**, **getSn_KPALVTR()**,
   setSn_KPALVTR()
4. **IP header field**
   getSn_FRAG(), **setSn_FRAG()**, **getSn_TOS()**, **setSn_TOS()**
   getSn_TTL(), **setSn_TTL()**

Socket APIs

Basic I/O function

These are basic input/output functions to read values from register or write values to register. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...
void	**wiz_recv_ignore** (uint8_t sn, uint16_t len) It discard the received data in RX memory. More...

Detailed Description

These are basic input/output functions to read values from register or write values to register.

Function Documentation

uint8_t WIZCHIP_READ (uint32_t AddrSel)

It reads 1 byte value from a register.

Parameters

AddrSel Register address

Returns

The value of register

void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb)

It writes 1 byte value to a register.

Parameters

AddrSel Register address

wb Write data

Returns

void

void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t * pBuf, uint16_t len)

It reads sequence data from registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to read data
len	Data length

```
void WIZCHIP_WRITE_BUF ( uint32_t  AddrSel,
                         uint8_t *  pBuf,
                         uint16_t  len
                       )
```

It writes sequence data to registers.

Parameters

AddrSel	Register address
pBuf	Pointer buffer to write data
len	Data length

```
void wiz_send_data ( uint8_t    sn,
                     uint8_t *  wizdata,
                     uint16_t  len
                   )
```

It copies data to internal TX memory.

This function reads the Tx write pointer register and after that, it copies the *wizdata(pointer buffer)* of the length of *len(variable)* bytes to internal TX memory and updates the Tx write pointer register. This function is being called by **send()** and **sendto()** function also.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
wizdata	Pointer buffer to write data
len	Data length

See also

wiz_recv_data()

```
void wiz_recv_data ( uint8_t    sn,
                     uint8_t *  wizdata,
                     uint16_t   len
                   )
```

It copies data to your buffer from internal RX memory.

This function read the Rx read pointer register and after that, it copies the received data from internal RX memory to *wizdata(pointer variable)* of the length of *len(variable)* bytes. This function is being called by **recv()** also.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
wizdata	Pointer buffer to read data
len	Data length

See also

wiz_send_data()

```
void wiz_recv_ignore ( uint8_t    sn,
                       uint16_t   len
                     )
```

It discard the received data in RX memory.

It discards the data of the length of *len(variable)* bytes in internal RX memory.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
len	Data length

Socket APIs

Common register access functions

These are functions to access **common registers**. More...

Macros

#define **getSHAR**(shar)
Get local MAC address. More...

#define **setMR**(mr) **WIZCHIP_WRITE**(**MR**,mr)
Set Mode Register. More...

#define **getMR**() **WIZCHIP_READ**(**MR**)
Get Mode Register. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set gateway IP address. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get gateway IP address. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**, subr,4)
Set subnet mask address. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get subnet mask address. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set local MAC address. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get local MAC address. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set local IP address. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get local IP address. More...

#define **setINTLEVEL**(intlevel)
Set INTLEVEL register. More...

#define **getINTLEVEL**() (((uint16_t)**WIZCHIP_READ**(**INTLEVEL**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**INTLEVEL**,1)))
Get INTLEVEL register. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xF0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xF0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(_IMR_, imr)
Set *IMR* register. More...

#define **getIMR**() **WIZCHIP_READ**(_IMR_)
Get *IMR* register. More...

#define **setSIR**(sir) **WIZCHIP_WRITE**(**SIR**, sir)
Set **SIR** register. More...

#define **getSIR**() **WIZCHIP_READ**(**SIR**)
Get **SIR** register. More...

#define **setSIMR**(simr) **WIZCHIP_WRITE**(**SIMR**, simr)
Set **SIMR** register. More...

#define **getSIMR**() **WIZCHIP_READ**(**SIMR**)
Get **SIMR** register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(_RTR_) << 8) +

WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)
Get **PMAGIC** register. More...

#define **setPHAR**(phar) **WIZCHIP_WRITE_BUF**(**PHAR**, phar, 6)
Set **PHAR** address. More...

#define **getPHAR**(phar) **WIZCHIP_READ_BUF**(**PHAR**, phar, 6)
Get **PHAR** address. More...

#define **setPSID**(psid)
Set **PSID** register. More...

#define **getPSID**() (((uint16_t)**WIZCHIP_READ**(**PSID**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PSID**,1)))
Get **PSID** register. More...

#define **setPMRU**(pmru)
Set **PMRU** register. More...

#define	**getPMRU**() (((uint16_t)**WIZCHIP_READ**(**PMRU**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PMRU**,1))) Get **PMRU** register. More...
#define	**getUIPR**(uipr) **WIZCHIP_READ_BUF**(**UIPR**,uipr,4) Get unreachable IP address. More...
#define	**getUPORTR**() (((uint16_t)**WIZCHIP_READ**(**UPORTR**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**UPORTR**,1))) Get **UPORTR** register. More...
#define	**setPHYCFGR**(phycfgr) **WIZCHIP_WRITE**(**PHYCFGR**, phycfgr) Set **PHYCFGR** register. More...
#define	**getPHYCFGR**() **WIZCHIP_READ**(**PHYCFGR**) Get **PHYCFGR** register. More...
#define	**getVERSIONR**() **WIZCHIP_READ**(**VERSIONR**) Get **VERSIONR** register. More...

Detailed Description

These are functions to access **common registers**.

Macro Definition Documentation

#define getSHAR (shar)

Value:

```
{ \
        (shar)[0] = (uint8_t)(WIZCHIP_READ(SHAR)
    >> 8); \
        (shar)[1] = (uint8_t)
    (WIZCHIP_READ(SHAR));      \
        (shar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,2)) >>
    8); \
        (shar)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,2)));
    \
        (shar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,4)) >>
    8); \
        (shar)[5] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,4)));
    \
   }
```

Get local MAC address.

Parameters

(uint8_t*)shar Pointer variable to get local MAC address. It should be allocated 6 bytes.

See also

setSHAR()

Definition at line **1419** of file **w5300.h**.

#define setMR (mr) WIZCHIP_WRITE(MR,mr)

Set Mode Register.

Parameters

(uint8_t)mr The value to be set.

See also

getMR()

Definition at line **1247** of file **w5500.h**.

#define getMR ( ) WIZCHIP_READ(MR)

Get Mode Register.

Returns

uint8_t. The value of Mode register.

See also

setMR()

Definition at line **1257** of file **w5500.h**.

#define setGAR (gar) WIZCHIP_WRITE_BUF(GAR,gar,4)

Set gateway IP address.

Parameters

(uint8_t*)gar Pointer variable to set gateway IP address. It should be allocated 4 bytes.

See also

getGAR()

Definition at line **1266** of file **w5500.h**.

#define getGAR (gar) WIZCHIP_READ_BUF(GAR,gar,4)

Get gateway IP address.

Parameters

(uint8_t*)gar Pointer variable to get gateway IP address. It should be allocated 4 bytes.

See also

setGAR()

Definition at line **1275** of file **w5500.h**.

#define setSUBR (subr) WIZCHIP_WRITE_BUF(SUBR, subr,4)

Set subnet mask address.

Parameters

(uint8_t*)subr Pointer variable to set subnet mask address. It should be allocated 4 bytes.

See also

getSUBR()

Definition at line **1284** of file **w5500.h**.

#define getSUBR (subr) WIZCHIP_READ_BUF(SUBR, subr, 4)

Get subnet mask address.

Parameters

(uint8_t*)subr Pointer variable to get subnet mask address. It should be allocated 4 bytes.

See also

setSUBR()

Definition at line **1294** of file **w5500.h**.

#define setSHAR (shar) WIZCHIP_WRITE_BUF(SHAR, shar, 6)

Set local MAC address.

Parameters

(uint8_t*)shar Pointer variable to set local MAC address. It should be allocated 6 bytes.

See also

getSHAR()

Definition at line **1303** of file **w5500.h**.

#define getSHAR (shar) WIZCHIP_READ_BUF(SHAR, shar, 6)

Get local MAC address.

Parameters

(uint8_t*)shar Pointer variable to get local MAC address. It should be allocated 6 bytes.

See also

setSHAR()

Definition at line **1312** of file **w5500.h**.

#define setSIPR (sipr) WIZCHIP_WRITE_BUF(SIPR, sipr, 4)

Set local IP address.

Parameters

(uint8_t*)sipr Pointer variable to set local IP address. It should be allocated 4 bytes.

See also
getSIPR()

Definition at line **1321** of file **w5500.h**.

#define getSIPR (sipr) WIZCHIP_READ_BUF(SIPR, sipr, 4)

Get local IP address.

Parameters
(uint8_t*)sipr Pointer variable to get local IP address. It should be allocated 4 bytes.

See also
setSIPR()

Definition at line **1330** of file **w5500.h**.

#define setINTLEVEL (intlevel)

Value:

```
{\
                WIZCHIP_WRITE(INTLEVEL,
    (uint8_t)(intlevel >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL,1),
    (uint8_t) intlevel); \
    }
```

Set INTLEVEL register.

Parameters
(uint16_t)intlevel Value to set **INTLEVEL** register.

See also
getINTLEVEL()

Definition at line **1339** of file **w5500.h**.

#define getINTLEVEL ( ) (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVE

Get INTLEVEL register.

Returns
uint16_t. Value of **INTLEVEL** register.

See also
setINTLEVEL()

Definition at line **1356** of file **w5500.h**.

#define setIR (ir) WIZCHIP_WRITE(IR, (ir & 0xF0))

Set **IR** register.

Parameters
(uint8_t)ir Value to set **IR** register.

See also
getIR()

Definition at line **1365** of file **w5500.h**.

#define getIR ( ) (WIZCHIP_READ(IR) & 0xF0)

Get **IR** register.

Returns
uint8_t. Value of **IR** register.

See also
setIR()

Definition at line **1374** of file **w5500.h**.

#define setIMR (imr) WIZCHIP_WRITE(_IMR_, imr)

Set *IMR* register.

Parameters
(uint8_t)imr Value to set *IMR* register.

See also
getIMR()

Definition at line **1382** of file **w5500.h**.

#define getIMR ( ) WIZCHIP_READ(_IMR_)

Get *IMR* register.

Returns
uint8_t. Value of *IMR* register.

See also
setIMR()

Definition at line **1391** of file **w5500.h**.

#define setSIR (sir) WIZCHIP_WRITE(SIR, sir)

Set **SIR** register.

Parameters
(uint8_t)sir Value to set **SIR** register.

See also
getSIR()

Definition at line **1400** of file **w5500.h**.

#define getSIR ( ) WIZCHIP_READ(SIR)

Get **SIR** register.

Returns
uint8_t. Value of **SIR** register.

See also
setSIR()

Definition at line **1409** of file **w5500.h**.

#define setSIMR (simr) WIZCHIP_WRITE(SIMR, simr)

Set **SIMR** register.

Parameters
(uint8_t)simr Value to set **SIMR** register.

See also
getSIMR()

Definition at line **1417** of file **w5500.h**.

#define getSIMR ( ) WIZCHIP_READ(SIMR)

Get **SIMR** register.

Returns
uint8_t. Value of **SIMR** register.

See also
setSIMR()

Definition at line **1426** of file **w5500.h**.

#define setRTR (rtr)

Value:

```
{\
                WIZCHIP_WRITE(_RTR_,   (uint8_t)
    (rtr >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),
    (uint8_t) rtr); \
    }
```

Set *RTR* register.

Parameters

(uint16_t)rtr Value to set *RTR* register.

See also

getRTR()

Definition at line **1435** of file **w5500.h**.

#define getRTR ( ) (((uint16_t)WIZCHIP_READ(_RTR_) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))

Get *RTR* register.

Returns

uint16_t. Value of *RTR* register.

See also

setRTR()

Definition at line **1451** of file **w5500.h**.

#define setRCR (rcr) WIZCHIP_WRITE(_RCR_, rcr)

Set *RCR* register.

Parameters
(uint8_t)rcr Value to set *RCR* register.

See also
getRCR()

Definition at line **1461** of file **w5500.h**.

#define getRCR ( ) WIZCHIP_READ(_RCR_)

Get *RCR* register.

Returns
uint8_t. Value of *RCR* register.

See also
setRCR()

Definition at line **1470** of file **w5500.h**.

#define setPTIMER (ptimer) WIZCHIP_WRITE(PTIMER, ptimer)

Set **PTIMER** register.

Parameters
(uint8_t)ptimer Value to set **PTIMER** register.

See also
getPTIMER()

Definition at line **1481** of file **w5500.h**.

#define getPTIMER ( ) WIZCHIP_READ(PTIMER)

Get **PTIMER** register.

Returns
uint8_t. Value of **PTIMER** register.

See also
setPTIMER()

Definition at line **1490** of file **w5500.h**.

#define setPMAGIC (pmagic) WIZCHIP_WRITE(PMAGIC, pmagic)

Set **PMAGIC** register.

Parameters
(uint8_t)pmagic Value to set **PMAGIC** register.

See also
getPMAGIC()

Definition at line **1499** of file **w5500.h**.

#define getPMAGIC ( ) WIZCHIP_READ(PMAGIC)

Get **PMAGIC** register.

Returns
uint8_t. Value of **PMAGIC** register.

See also
setPMAGIC()

Definition at line **1508** of file **w5500.h**.

#define setPHAR (phar) WIZCHIP_WRITE_BUF(PHAR, phar, 6)

Set **PHAR** address.

Parameters

(uint8_t*)phar Pointer variable to set PPP destination MAC register address. It should be allocated 6 bytes.

See also

getPHAR()

Definition at line **1517** of file **w5500.h**.

#define getPHAR (phar) WIZCHIP_READ_BUF(PHAR, phar, 6)

Get **PHAR** address.

Parameters

(uint8_t*)phar Pointer variable to PPP destination MAC register address. It should be allocated 6 bytes.

See also

setPHAR()

Definition at line **1526** of file **w5500.h**.

#define setPSID (psid)

Value:

```
{\
                WIZCHIP_WRITE(PSID,   (uint8_t)
    (psid >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PSID,1),
    (uint8_t) psid); \
```

```
    }
```

Set **PSID** register.

Parameters

(uint16_t)psid Value to set **PSID** register.

See also

getPSID()

Definition at line **1535** of file **w5500.h**.

#define getPSID ( ) (((uint16_t)WIZCHIP_READ(PSID) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1)))

Get **PSID** register.

Returns

uint16_t. Value of **PSID** register.

See also

setPSID()

Definition at line **1552** of file **w5500.h**.

#define setPMRU (pmru)

Value:

```
{ \
                WIZCHIP_WRITE(PMRU,    (uint8_t)
    (pmru>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PMRU,1),
    (uint8_t) pmru); \
    }
```

Set **PMRU** register.

Parameters

(uint16_t)pmru Value to set **PMRU** register.

See also

getPMRU()

Definition at line **1561** of file **w5500.h**.

#define getPMRU ( ) (((uint16_t)WIZCHIP_READ(PMRU) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))

Get **PMRU** register.

Returns

uint16_t. Value of **PMRU** register.

See also

setPMRU()

Definition at line **1577** of file **w5500.h**.

#define getUIPR (uipr) WIZCHIP_READ_BUF(UIPR,uipr,4)

Get unreachable IP address.

Parameters

(uint8_t*)uipr Pointer variable to get unreachable IP address. It should be allocated 4 bytes.

Definition at line **1590** of file **w5500.h**.

#define getUPORTR ( ) (((uint16_t)WIZCHIP_READ(UPORTR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1

Get **UPORTR** register.

Returns
uint16_t. Value of **UPORTR** register.

Definition at line **1603** of file **w5500.h**.

#define setPHYCFGR (phycfgr) WIZCHIP_WRITE(PHYCFGR, phycfgr)

Set **PHYCFGR** register.

Parameters
(uint8_t)phycfgr Value to set **PHYCFGR** register.

See also
getPHYCFGR()

Definition at line **1612** of file **w5500.h**.

Referenced by **wizphy_reset()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

#define getPHYCFGR ( ) WIZCHIP_READ(PHYCFGR)

Get **PHYCFGR** register.

Returns
uint8_t. Value of **PHYCFGR** register.

See also
setPHYCFGR()

Definition at line **1621** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_getphylink()**, **wizphy_getphypmode()**, **wizphy_getphystat()**, **wizphy_reset()**, and **wizphy_setphypmode()**.

#define getVERSIONR ( ) WIZCHIP_READ(VERSIONR)

Get **VERSIONR** register.

Returns

uint8_t. Value of **VERSIONR** register.

Definition at line **1629** of file **w5500.h**.

Socket APIs

Socket register access functions

These are functions to access **socket registers**. More...

Macros

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), (ir & 0x1F))
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) (**WIZCHIP_READ**(**Sn_IR**(sn)) & 0x1F)
Get **Sn_IR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), (imr & 0x
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) (**WIZCHIP_READ**(**Sn_IMR**(sn)) & 0x1F)
Get **Sn_IMR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define **setSn_PORT**(sn, port)
Set **Sn_PORT** register. More...

#define **getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn)
Get **Sn_PORT** register. More...

#define	**setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s dhar, 6) Set **Sn_DHAR** register. More...
#define	**getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(s dhar, 6) Get **Sn_MR** register. More...
#define	**setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn), 4) Set **Sn_DIPR** register. More...
#define	**getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn), 4) Get **Sn_DIPR** register. More...
#define	**setSn_DPORT**(sn, dport) Set **Sn_DPORT** register. More...
#define	**getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT** << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1) Get **Sn_DPORT** register. More...
#define	**setSn_MSSR**(sn, mss) Set **Sn_MSSR** register. More...
#define	**getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn) Get **Sn_MSSR** register. More...
#define	**setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos) Set **Sn_TOS** register. More...
#define	**getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn)) Get **Sn_TOS** register. More...

#define **setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl)
Set **Sn_TTL** register. More...

#define **getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn))
Get **Sn_TTL** register. More...

#define **setSn_RXBUF_SIZE**(sn, rxbufsize) **WIZCHIP_WRITE**(**Sn_RXBUF_SIZE**(sn),rxbufsize
Set **Sn_RXBUF_SIZE** register. More...

#define **getSn_RXBUF_SIZE**(sn) **WIZCHIP_READ**(**Sn_RXBUF_SIZ**
Get **Sn_RXBUF_SIZE** register. More...

#define **setSn_TXBUF_SIZE**(sn, txbufsize) **WIZCHIP_WRITE**(**Sn_TXBUF_SIZE**(sn), txbufsize
Set **Sn_TXBUF_SIZE** register. More...

#define **getSn_TXBUF_SIZE**(sn) **WIZCHIP_READ**(**Sn_TXBUF_SIZE**
Get **Sn_TXBUF_SIZE** register. More...

#define **getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(s
<< 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sn),1))
Get **Sn_TX_RD** register. More...

#define **setSn_TX_WR**(sn, txwr)
Set **Sn_TX_WR** register. More...

#define **getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR**
<< 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1)
Get **Sn_TX_WR** register. More...

#define **setSn_RX_RD**(sn, rxrd)
Set **Sn_RX_RD** register. More...

#define	**getSn_RX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(<< 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1)) Get **Sn_RX_RD** register. More...
#define	**getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR** << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1) Get **Sn_RX_WR** register. More...
#define	**setSn_FRAG**(sn, frag) Set **Sn_FRAG** register. More...
#define	**getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn) Get **Sn_FRAG** register. More...
#define	**setSn_KPALVTR**(sn, kpalvt) **WIZCHIP_WRITE**(**Sn_KPALVTR**(sn), kpalvt) Set **Sn_KPALVTR** register. More...
#define	**getSn_KPALVTR**(sn) **WIZCHIP_READ**(**Sn_KPALVTR**(sn)) Get **Sn_KPALVTR** register. More...

Functions

uint16_t **getSn_TX_FSR** (uint8_t sn)
Get **Sn_TX_FSR** register. More...

uint16_t **getSn_RX_RSR** (uint8_t sn)
Get **Sn_RX_RSR** register. More...

Detailed Description

These are functions to access **socket registers**.

Macro Definition Documentation

#define setSn_MR (sn,
mr
) WIZCHIP_WRITE(Sn_MR(sn),mr)

Set **Sn_MR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)mr Value to set **Sn_MR**

See also

getSn_MR()

Definition at line **1644** of file **w5500.h**.

#define getSn_MR (sn) WIZCHIP_READ(Sn_MR(sn))

Get **Sn_MR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_MR**.

See also

setSn_MR()

Definition at line **1654** of file **w5500.h**.

#define setSn_CR (sn,

```
                cr
            )    WIZCHIP_WRITE(Sn_CR(sn), cr)
```

Set **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)cr Value to set **Sn_CR**

See also

getSn_CR()

Definition at line **1664** of file **w5500.h**.

```
#define getSn_CR (  sn )   WIZCHIP_READ(Sn_CR(sn))
```

Get **Sn_CR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_CR**.

See also

setSn_CR()

Definition at line **1674** of file **w5500.h**.

```
#define setSn_IR (  sn,
                    ir
                )    WIZCHIP_WRITE(Sn_IR(sn), (ir & 0x1F))
```

Set **Sn_IR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)ir Value to set **Sn_IR**

See also
getSn_IR()

Definition at line **1684** of file **w5500.h**.

#define getSn_IR (sn) (WIZCHIP_READ(Sn_IR(sn)) & 0x1F)

Get **Sn_IR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint8_t. Value of **Sn_IR**.

See also
setSn_IR()

Definition at line **1694** of file **w5500.h**.

#define setSn_IMR (sn, imr WIZCHIP_WRITE(Sn_IMR(sn), (imr &) 0x1F))

Set **Sn_IMR** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)imr Value to set **Sn_IMR**

See also
getSn_IMR()

Definition at line **1704** of file **w5500.h**.

#define getSn_IMR (sn) (WIZCHIP_READ(Sn_IMR(sn)) & 0x1F)

Get **Sn_IMR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_IMR**.

See also

setSn_IMR()

Definition at line **1714** of file **w5500.h**.

#define getSn_SR (sn) WIZCHIP_READ(Sn_SR(sn))

Get **Sn_SR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_SR**.

Definition at line **1723** of file **w5500.h**.

#define setSn_PORT (sn, port)

Value:

```
{ \
```

```
                WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    (uint8_t) port); \
    }
```

Set **Sn_PORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)port Value to set **Sn_PORT**.

See also

getSn_PORT()

Definition at line **1733** of file **w5500.h**.

#define getSn_PORT (sn) (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PO

Get **Sn_PORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_PORT**.

See also

setSn_PORT()

Definition at line **1750** of file **w5500.h**.

#define setSn_DHAR (sn, dhar

```
                    WIZCHIP_WRITE_BUF(Sn_DHAR(sn),
                ) dhar, 6)
```

Set **Sn_DHAR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dhar	Pointer variable to set socket n destination hardware address. It should be allocated 6 bytes.

See also

getSn_DHAR()

Definition at line **1760** of file **w5500.h**.

```
#define
getSn_DHAR      ( sn,
                  dhar
                    WIZCHIP_READ_BUF(Sn_DHAR(sn),
                ) dhar, 6)
```

Get **Sn_MR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dhar	Pointer variable to get socket n destination hardware address. It should be allocated 6 bytes.

See also

setSn_DHAR()

Definition at line **1770** of file **w5500.h**.

```
#define
```

```
setSn_DIPR ( sn,
             dipr
               WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
           ) 4)
```

Set **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dipr	Pointer variable to set socket n destination IP address. It should be allocated 4 bytes.

See also

getSn_DIPR()

Definition at line **1780** of file **w5500.h**.

```
#define
getSn_DIPR ( sn,
             dipr
               WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
           ) 4)
```

Get **Sn_DIPR** register.

Parameters

(uint8_t)sn	Socket number. It should be **0 ~ 7**.
(uint8_t*)dipr	Pointer variable to get socket n destination IP address. It should be allocated 4 bytes.

See also

setSn_DIPR()

Definition at line **1790** of file **w5500.h**.

```
#define setSn_DPORT ( sn,
```

dport
)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    (uint8_t)  dport); \
    }
```

Set **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)dport Value to set **Sn_DPORT**

See also

getSn_DPORT()

Definition at line **1800** of file **w5500.h**.

#define getSn_DPORT (sn) (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn))<<8) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_D

Get **Sn_DPORT** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_DPORT**.

See also

setSn_DPORT()

Definition at line **1817** of file **w5500.h**.

#define setSn_MSSR (sn, mss)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    (uint8_t) mss); \
    }
```

Set **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)mss Value to set **Sn_MSSR**

See also

setSn_MSSR()

Definition at line **1827** of file **w5500.h**.

#define getSn_MSSR (sn) (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_M

Get **Sn_MSSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_MSSR**.

See also

setSn_MSSR()

Definition at line **1844** of file **w5500.h**.

```
#define setSn_TOS (  sn,
                     tos
                  )    WIZCHIP_WRITE(Sn_TOS(sn), tos)
```

Set **Sn_TOS** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)tos Value to set **Sn_TOS**

See also

getSn_TOS()

Definition at line **1854** of file **w5500.h**.

```
#define getSn_TOS (  sn )    WIZCHIP_READ(Sn_TOS(sn))
```

Get **Sn_TOS** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of Sn_TOS.

See also

setSn_TOS()

Definition at line **1864** of file **w5500.h**.

```
#define setSn_TTL (  sn,
                     ttl
```

) WIZCHIP_WRITE(Sn_TTL(sn), ttl)

Set **Sn_TTL** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint8_t)ttl Value to set **Sn_TTL**

See also

getSn_TTL()

Definition at line **1874** of file **w5500.h**.

#define getSn_TTL (sn) WIZCHIP_READ(Sn_TTL(sn))

Get **Sn_TTL** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_TTL**.

See also

setSn_TTL()

Definition at line **1885** of file **w5500.h**.

#define
setSn_RXBUF_SIZE (sn,
rxbufsize
) WIZCHIP_WRITE(Sn_RXBUF_SIZE(sn),rxb

Set **Sn_RXBUF_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)rxbufsize Value to set **Sn_RXBUF_SIZE**

See also
getSn_RXBUF_SIZE()

Definition at line **1896** of file **w5500.h**.

#define
getSn_RXBUF_SIZE (sn) WIZCHIP_READ(Sn_RXBUF_SIZE(sn)

Get **Sn_RXBUF_SIZE** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint8_t. Value of **Sn_RXBUF_SIZE**.

See also
setSn_RXBUF_SIZE()

Definition at line **1907** of file **w5500.h**.

#define
setSn_TXBUF_SIZE (sn,
txbufsize
WIZCHIP_WRITE(Sn_TXBUF_SIZE(sn),
) txbufsize)

Set **Sn_TXBUF_SIZE** register.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.
(uint8_t)txbufsize Value to set **Sn_TXBUF_SIZE**

See also

getSn_TXBUF_SIZE()

Definition at line **1917** of file **w5500.h**.

```
#define
getSn_TXBUF_SIZE ( sn ) WIZCHIP_READ(Sn_TXBUF_SIZE(sn))
```

Get **Sn_TXBUF_SIZE** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_TXBUF_SIZE**.

See also

setSn_TXBUF_SIZE()

Definition at line **1927** of file **w5500.h**.

```
#define (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn))
getSn_TX_RD ( sn ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T
```

Get **Sn_TX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_TX_RD**.

Definition at line **1949** of file **w5500.h**.

```
#define setSn_TX_WR ( sn,
                      txwr
                    )
```

Value:

```
{ \
                WIZCHIP_WRITE(Sn_TX_WR(sn),
    (uint8_t)(txwr>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    (uint8_t) txwr); \
        }
```

Set **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)txwr Value to set **Sn_TX_WR**

See also

GetSn_TX_WR()

Definition at line **1959** of file **w5500.h**.

#define getSn_TX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_T

Get **Sn_TX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_TX_WR**.

See also

setSn_TX_WR()

Definition at line **1976** of file **w5500.h**.

#define setSn_RX_RD (sn, rxrd)

Value:

```
{ \
                  WIZCHIP_WRITE(Sn_RX_RD(sn),
    (uint8_t)(rxrd>>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    (uint8_t) rxrd); \
    }
```

Set **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)rxrd Value to set **Sn_RX_RD**

See also

getSn_RX_RD()

Definition at line **1996** of file **w5500.h**.

#define getSn_RX_RD (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_F

Get **Sn_RX_RD** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_RX_RD**.

See also

setSn_RX_RD()

Definition at line **2013** of file **w5500.h**.

#define getSn_RX_WR (sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_

Get **Sn_RX_WR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_RX_WR**.

Definition at line **2027** of file **w5500.h**.

#define setSn_FRAG (sn, frag)

Value:

```
{ \
                WIZCHIP_WRITE(Sn_FRAG(sn),
    (uint8_t)(frag >>8)); \

    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)
    (uint8_t) frag); \
    }
```

Set **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint16_t)frag Value to set **Sn_FRAG**

See also

getSn_FRAD()

Definition at line **2037** of file **w5500.h**.

#define getSn_FRAG (sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FR

Get **Sn_FRAG** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_FRAG**.

See also

setSn_FRAG()

Definition at line **2054** of file **w5500.h**.

#define setSn_KPALVTR (sn, kpalvt) WIZCHIP_WRITE(Sn_KPALVTR(sn), kpalvt)

Set **Sn_KPALVTR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

(uint8_t)kpalvt Value to set **Sn_KPALVTR**

See also

getSn_KPALVTR()

Definition at line **2064** of file **w5500.h**.

#define

getSn_KPALVTR (sn) WIZCHIP_READ(Sn_KPALVTR(sn))

Get **Sn_KPALVTR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint8_t. Value of **Sn_KPALVTR**.

See also

setSn_KPALVTR()

Definition at line **2074** of file **w5500.h**.

Function Documentation

uint16_t getSn_TX_FSR (uint8_t sn)

Get **Sn_TX_FSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_TX_FSR**.

uint16_t getSn_RX_RSR (uint8_t sn)

Get **Sn_RX_RSR** register.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of **Sn_RX_RSR**.

Socket APIs

WIZCHIP register

W5500

WHIZCHIP register defines register group of **W5500**. More...

Modules

Common register
Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

Socket register
Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication.

Detailed Description

WHIZCHIP register defines register group of **W5500**.

- **Common register** : Common register group
- **Socket register** : SOCKET n register group

Socket APIs

Common register

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc. More...

Macros

#define	**MR** (**_W5500_IO_BASE_** + (0x0000 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Mode Register address(R/W) **MR** is used for S/W reset, ping block mode, PPPoE mode and etc. More...
#define	**GAR** (**_W5500_IO_BASE_** + (0x0001 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Gateway IP Register address(R/W) More...
#define	**SUBR** (**_W5500_IO_BASE_** + (0x0005 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Subnet mask Register address(R/W) More...
#define	**SHAR** (**_W5500_IO_BASE_** + (0x0009 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Source MAC Register address(R/W) More...
#define	**SIPR** (**_W5500_IO_BASE_** + (0x000F << 8) + (WIZCHIP_CREG_BLOCK << 3)) Source IP Register address(R/W) More...
#define	**INTLEVEL** (**_W5500_IO_BASE_** + (0x0013 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Set Interrupt low level timer register address(R/W) More...
#define	**IR** (**_W5500_IO_BASE_** + (0x0015 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Interrupt Register(R/W) More...
#define	**_IMR_** (**_W5500_IO_BASE_** + (0x0016 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Interrupt mask register(R/W) More...

#define **SIR** (**_W5500_IO_BASE_** + (0x0017 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Register(R/W) More...

#define **SIMR** (**_W5500_IO_BASE_** + (0x0018 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (**_W5500_IO_BASE_** + (0x0019 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5500_IO_BASE_** + (0x001B << 8) + (WIZCHIP_CREG_BLOCK << 3))
Retry count register(R/W) More...

#define **PTIMER** (**_W5500_IO_BASE_** + (0x001C << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP LCP Request Timer register in PPPoE mode(R/W) More...

#define **PMAGIC** (**_W5500_IO_BASE_** + (0x001D << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP LCP Magic number register in PPPoE mode(R/W) More...

#define **PHAR** (**_W5500_IO_BASE_** + (0x001E << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Destination MAC Register address(R/W) More...

#define **PSID** (**_W5500_IO_BASE_** + (0x0024 << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Session Identification Register(R/W) More...

#define **PMRU** (**_W5500_IO_BASE_** + (0x0026 << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Maximum Segment Size(MSS) register(R/W) More...

#define	**UIPR** (**_W5500_IO_BASE_** + (0x0028 << 8) + (WIZCHIP_CREG_BLOCK << 3)) Unreachable IP register address in UDP mode(R) More...
#define	**UPORTR** (**_W5500_IO_BASE_** + (0x002C << 8) + (WIZCHIP_CREG_BLOCK << 3)) Unreachable Port register address in UDP mode(R) More...
#define	**PHYCFGR** (**_W5500_IO_BASE_** + (0x002E << 8) + (WIZCHIP_CREG_BLOCK << 3)) PHY Status Register(R/W) More...
#define	**VERSIONR** (**_W5500_IO_BASE_** + (0x0039 << 8) + (WIZCHIP_CREG_BLOCK << 3)) chip version register address(R) More...

Detailed Description

Common register group
It set the basic for the networking
It set the configuration such as interrupt, network information, ICMP, etc.

See also

MR : Mode register.

GAR, **SUBR**, **SHAR**, **SIPR**

INTLEVEL, **IR**, IMR, **SIR**, **SIMR** : Interrupt.

RTR, *RCR* : Data retransmission.

PTIMER, **PMAGIC**, **PHAR**, **PSID**, **PMRU** : PPPoE.

UIPR, **UPORTR** : ICMP message.

PHYCFGR, **VERSIONR** : etc.

Macro Definition Documentation

#define MR (_W5500_IO_BASE_ + (0x0000 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and etc.

Each bit of **MR** defined as follows.

7	6	5	4	3	2	1	0
RST	Reserved	WOL	PB	PPPoE	Reserved	FARP	Reserved

- **MR_RST** : Reset
- **MR_WOL** : Wake on LAN
- **MR_PB** : Ping block
- **MR_PPPOE** : PPPoE mode
- **MR_FARP** : Force ARP mode

Definition at line **214** of file **w5500.h**.

#define GAR (_W5500_IO_BASE_ + (0x0001 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Gateway IP Register address(R/W)

GAR configures the default gateway address.

Definition at line **221** of file **w5500.h**.

#define SUBR (_W5500_IO_BASE_ + (0x0005 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Subnet mask Register address(R/W)

SUBR configures the subnet mask address.

Definition at line **228** of file **w5500.h**.

#define SHAR (_W5500_IO_BASE_ + (0x0009 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Source MAC Register address(R/W)

SHAR configures the source hardware address.

Definition at line **235** of file **w5500.h**.

#define SIPR (_W5500_IO_BASE_ + (0x000F << 8) + (WIZCHIP_CREG_BLOCK << 3))

Source IP Register address(R/W)

SIPR configures the source IP address.

Definition at line **242** of file **w5500.h**.

#define INTLEVEL (_W5500_IO_BASE_ + (0x0013 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Set Interrupt low level timer register address(R/W)

INTLEVEL configures the Interrupt Assert Time.

Definition at line **249** of file **w5500.h**.

#define IR (_W5500_IO_BASE_ + (0x0015 << 8) + (WIZCHIP_CREG

3))

Interrupt Register(R/W)

IR indicates the interrupt status. Each bit of **IR** will be still until the bit wi
by the host. If **IR** is not equal to x00 INTn PIN is asserted to low until it i

Each bit of **IR** defined as follows.

7	6	5	4	3	2	1
CONFLICT	UNREACH	PPPoE	MP	Reserved	Reserved	Reserv

- **IR_CONFLICT** : IP conflict
- **IR_UNREACH** : Destination unreachable
- **IR_PPPoE** : PPPoE connection close
- **IR_MP** : Magic packet

Definition at line **266** of file **w5500.h**.

#define _IMR_ (_W5500_IO_BASE_ + (0x0016 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Interrupt mask register(R/W)

IMR is used to mask interrupts. Each bit of *IMR* corresponds to each bi
When a bit of *IMR* is and the corresponding bit of **IR** is an interrupt will l
In other words, if a bit of *IMR* is an interrupt will not be issued even if th
corresponding bit of **IR** is

Each bit of *IMR* defined as the following.

7	6	5	4	3	2	1
IM_IR7	IM_IR6	IM_IR5	IM_IR4	Reserved	Reserved	Reserved

- **IM_IR7** : IP Conflict Interrupt Mask
- **IM_IR6** : Destination unreachable Interrupt Mask
- **IM_IR5** : PPPoE Close Interrupt Mask
- **IM_IR4** : Magic Packet Interrupt Mask

Definition at line **286** of file **w5500.h**.

#define SIR (_W5500_IO_BASE_ + (0x0017 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Socket Interrupt Register(R/W)

SIR indicates the interrupt status of Socket.
Each bit of **SIR** be still until **Sn_IR** is cleared by the host.
If **Sn_IR** is not equal to x00 the n-th bit of **SIR** is and INTn PIN is asserted until **SIR** is x00

Definition at line **294** of file **w5500.h**.

#define SIMR (_W5500_IO_BASE_ + (0x0018 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Socket Interrupt Mask Register(R/W)

Each bit of **SIMR** corresponds to each bit of **SIR**. When a bit of **SIMR** is and the corresponding bit of **SIR** is Interrupt will be issued. In other words, if a bit of **SIMR** is an interrupt will be not issued even if the corresponding bit of **SIR** is

Definition at line **303** of file **w5500.h**.

#define _RTR_ (_W5500_IO_BASE_ + (0x0019 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Timeout register address(1 is 100us)(R/W)

RTR configures the retransmission timeout period. The unit of timeout period is 100us and the default of *RTR* is x07D0. And so the default timeout period is 200ms(100us X 2000). During the time configured by *RTR*, W5500 waits for the peer response to the packet that is transmitted by **Sn_CR** (CONNECT, DISCON, CLOSE, SEND,

SEND_MAC, SEND_KEEP command). If the peer does not respond within the *RTR* time, W5500 retransmits the packet or issues timeout.

Definition at line **315** of file **w5500.h**.

#define _RCR_ (_W5500_IO_BASE_ + (0x001B << 8) + (WIZCHIP_CREG_BLOCK << 3))

Retry count register(R/W)

RCR configures the number of time of retransmission. When retransmission occurs as many as ref *RCR*+1 Timeout interrupt is issued (**Sn_IR_TIMEOUT** = '1').

Definition at line **325** of file **w5500.h**.

#define PTIMER (_W5500_IO_BASE_ + (0x001C << 8) + (WIZCHIP_CREG_BLOCK << 3))

PPP LCP Request Timer register in PPPoE mode(R/W)

PTIMER configures the time for sending LCP echo request. The unit of time is 25ms.

Definition at line **332** of file **w5500.h**.

#define PMAGIC (_W5500_IO_BASE_ + (0x001D << 8) + (WIZCHIP_CREG_BLOCK << 3))

PPP LCP Magic number register in PPPoE mode(R/W)

PMAGIC configures the 4bytes magic number to be used in LCP negotiation.

Definition at line **339** of file **w5500.h**.

#define PHAR (_W5500_IO_BASE_ + (0x001E << 8) + (WIZCHIP_CREG_BLOCK << 3))

PPP Destination MAC Register address(R/W)

PHAR configures the PPPoE server hardware address that is acquired during PPPoE connection process.

Definition at line **346** of file **w5500.h**.

#define PSID (_W5500_IO_BASE_ + (0x0024 << 8) + (WIZCHIP_CREG_BLOCK << 3))

PPP Session Identification Register(R/W)

PSID configures the PPPoE sever session ID acquired during PPPoE connection process.

Definition at line **353** of file **w5500.h**.

#define PMRU (_W5500_IO_BASE_ + (0x0026 << 8) + (WIZCHIP_CREG_BLOCK << 3))

PPP Maximum Segment Size(MSS) register(R/W)

PMRU configures the maximum receive unit of PPPoE.

Definition at line **360** of file **w5500.h**.

#define UIPR (_W5500_IO_BASE_ + (0x0028 << 8) + (WIZCHIP_CREG_BLOCK << 3))

Unreachable IP register address in UDP mode(R)

W5500 receives an ICMP packet(Destination port unreachable)

when data is sent to a port number which socket is not open and **IR_UNREACH** bit of **IR** becomes and **UIPR** & **UPORTR** indicates the destination IP address & port number respectively.

Definition at line **369** of file **w5500.h**.

#define UPORTR (_W5500_IO_BASE_ + (0x002C << 8) + (WIZCHIP_CREG_BLOCK << 3))

Unreachable Port register address in UDP mode(R)

W5500 receives an ICMP packet(Destination port unreachable) when data is sent to a port number which socket is not open and **IR_UNREACH** bit of **IR** becomes and **UIPR** & **UPORTR** indicates the destination IP address & port number respectively.

Definition at line **378** of file **w5500.h**.

#define PHYCFGR (_W5500_IO_BASE_ + (0x002E << 8) + (WIZCHIP_CREG_BLOCK << 3))

PHY Status Register(R/W)

PHYCFGR configures PHY operation mode and resets PHY. In addition, **PHYCFGR** indicates the status of PHY such as duplex, Speed, Link.

Definition at line **385** of file **w5500.h**.

#define VERSIONR (_W5500_IO_BASE_ + (0x0039 << 8) + (WIZCHIP_CREG_BLOCK << 3))

chip version register address(R)

VERSIONR always indicates the W5500 version as **0x04**.

Definition at line **403** of file **w5500.h**.

Socket APIs

Socket register

Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication. More...

Macros

#define	**Sn_MR**(N) (**_W5500_IO_BASE_** + (0x0000 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) socket Mode register(R/W) More...
#define	**Sn_CR**(N) (**_W5500_IO_BASE_** + (0x0001 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Socket command register(R/W) More...
#define	**Sn_IR**(N) (**_W5500_IO_BASE_** + (0x0002 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Socket interrupt register(R) More...
#define	**Sn_SR**(N) (**_W5500_IO_BASE_** + (0x0003 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Socket status register(R) More...
#define	**Sn_PORT**(N) (**_W5500_IO_BASE_** + (0x0004 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) source port register(R/W) More...
#define	**Sn_DHAR**(N) (**_W5500_IO_BASE_** + (0x0006 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Peer MAC register address(R/W) More...
#define	**Sn_DIPR**(N) (**_W5500_IO_BASE_** + (0x000C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Peer IP register address(R/W) More...
#define	**Sn_DPORT**(N) (**_W5500_IO_BASE_** + (0x0010 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Peer port register address(R/W) More...
#define	**Sn_MSSR**(N) (**_W5500_IO_BASE_** + (0x0012 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

	Maximum Segment Size(Sn_MSSR0) register address(R/W) More...
#define	**Sn_TOS**(N) (**_W5500_IO_BASE_** + (0x0015 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) IP Type of Service(TOS) Register(R/W) More...
#define	**Sn_TTL**(N) (**_W5500_IO_BASE_** + (0x0016 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) IP Time to live(TTL) Register(R/W) More...
#define	**Sn_RXBUF_SIZE**(N) (**_W5500_IO_BASE_** + (0x001E << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Receive memory size register(R/W) More...
#define	**Sn_TXBUF_SIZE**(N) (**_W5500_IO_BASE_** + (0x001F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory size register(R/W) More...
#define	**Sn_TX_FSR**(N) (**_W5500_IO_BASE_** + (0x0020 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit free memory size register(R) More...
#define	**Sn_TX_RD**(N) (**_W5500_IO_BASE_** + (0x0022 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory read pointer register address(R) More...
#define	**Sn_TX_WR**(N) (**_W5500_IO_BASE_** + (0x0024 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory write pointer register address(R/W) More...
#define	**Sn_RX_RSR**(N) (**_W5500_IO_BASE_** + (0x0026 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Received data size register(R) More...
#define	**Sn_RX_RD**(N) (**_W5500_IO_BASE_** + (0x0028 << 8) +

	(WIZCHIP_SREG_BLOCK(N) << 3)) Read point of Receive memory(R/W) More...
#define	**Sn_RX_WR**(N) (**_W5500_IO_BASE_** + (0x002A << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Write point of Receive memory(R) More...
#define	**Sn_IMR**(N) (**_W5500_IO_BASE_** + (0x002C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) socket interrupt mask register(R) More...
#define	**Sn_FRAG**(N) (**_W5500_IO_BASE_** + (0x002D << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Fragment field value in IP header register(R/W) More...
#define	**Sn_KPALVTR**(N) (**_W5500_IO_BASE_** + (0x002F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Keep Alive Timer register(R/W) More...

Detailed Description

Socket register group.
Socket register configures and control SOCKETn which is necessary to data communication.

See also

Sn_MR, **Sn_CR**, **Sn_IR**, **Sn_IMR** : SOCKETn Control

Sn_SR, **Sn_PORT**, **Sn_DHAR**, **Sn_DIPR**, **Sn_DPORT** : SOCKETn Information

Sn_MSSR, **Sn_TOS**, **Sn_TTL**, **Sn_KPALVTR**, **Sn_FRAG** : Internet protocol.

Sn_RXBUF_SIZE, **Sn_TXBUF_SIZE**, **Sn_TX_FSR**, **Sn_TX_RD**, **Sn_TX_WR**, **Sn_RX_RSR**, **Sn_RX_RD**, **Sn_RX_WR** : Data communication

Macro Definition Documentation

#define Sn_MR (N) (_W5500_IO_BASE_ + (0x0000 << 8) + (WIZC

socket Mode register(R/W)

Sn_MR configures the option or protocol type of Socket n.

Each bit of **Sn_MR** defined as the following.

7	6	5	4	3
MULTI/MFEN	BCASTB	ND/MC/MMB	UCASTB/MIP6B	Protocol[3]

- **Sn_MR_MULTI** : Support UDP Multicasting
- **Sn_MR_BCASTB** : Broadcast block **in UDP Multicasting**
- **Sn_MR_ND** : No Delayed Ack(TCP) flag
- **Sn_MR_MC** : IGMP version used **in UDP mulitcasting**
- **Sn_MR_MMB** : Multicast Blocking **in Sn_MR_MACRAW mode**
- **Sn_MR_UCASTB** : Unicast Block **in UDP Multicating**
- **Sn_MR_MIP6B** : IPv6 packet Blocking **in Sn_MR_MACRAW mod**
- **Protocol**

Protocol[3]	Protocol[2]	Protocol[1]	Protocol[0]	Meaning
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	1	0	0	MACRAW

  - **Sn_MR_MACRAW** : MAC LAYER RAW SOCK
  - **Sn_MR_UDP** : UDP
  - **Sn_MR_TCP** : TCP
  - **Sn_MR_CLOSE** : Unused socket

  Note
  MACRAW mode should be only used in Socket 0.

Definition at line **437** of file **w5500.h**.

#define Sn_CR (N) (_W5500_IO_BASE_ + (0x0001 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Socket command register(R/W)

This is used to set the command for Socket n such as OPEN, CLOSE, CONNECT, LISTEN, SEND, and RECEIVE.
After W5500 accepts the command, the **Sn_CR** register is automatically cleared to 0x00. Even though **Sn_CR** is cleared to 0x00, the command is still being processed.
To check whether the command is completed or not, please check the **Sn_IR** or **Sn_SR**.

- **Sn_CR_OPEN** : Initialize or open socket.
- **Sn_CR_LISTEN** : Wait connection request in TCP mode(**Server mode**)
- **Sn_CR_CONNECT** : Send connection request in TCP mode(**Client mode**)
- **Sn_CR_DISCON** : Send closing request in TCP mode.
- **Sn_CR_CLOSE** : Close socket.
- **Sn_CR_SEND** : Update TX buffer pointer and send data.
- **Sn_CR_SEND_MAC** : Send data with MAC address, so without ARP process.
- **Sn_CR_SEND_KEEP** : Send keep alive message.
- **Sn_CR_RECV** : Update RX buffer pointer and receive data.

Definition at line **456** of file **w5500.h**.

#define Sn_IR (N) (_W5500_IO_BASE_ + (0x0002 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Socket interrupt register(R)

Sn_IR indicates the status of Socket Interrupt such as establishment, te
receiving data, timeout).
When an interrupt occurs and the corresponding bit of **Sn_IMR** is the c
bit of **Sn_IR** becomes

In order to clear the **Sn_IR** bit, the host should write the bit to

7	6	5	4	3	2	1
Reserved	Reserved	Reserved	SEND_OK	TIMEOUT	RECV	DIS

- **Sn_IR_SENDOK** : **SEND_OK Interrupt**
- **Sn_IR_TIMEOUT** : **TIMEOUT Interrupt**
- **Sn_IR_RECV** : **RECV Interrupt**
- **Sn_IR_DISCON** : **DISCON Interrupt**
- **Sn_IR_CON** : **CON Interrupt**

Definition at line **474** of file **w5500.h**.

#define Sn_SR (N) (_W5500_IO_BASE_ + (0x0003 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Socket status register(R)

Sn_SR indicates the status of Socket n.
The status of Socket n is changed by **Sn_CR** or some special control packet as SYN, FIN packet in TCP.

Normal status

- **SOCK_CLOSED** : Closed
- **SOCK_INIT** : Initiate state
- **SOCK_LISTEN** : Listen state
- **SOCK_ESTABLISHED** : Success to connect
- **SOCK_CLOSE_WAIT** : Closing state
- **SOCK_UDP** : UDP socket
- **SOCK_MACRAW** : MAC raw mode socket

Temporary status during changing the status of Socket n.

- **SOCK_SYNSENT** : This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
- **SOCK_SYNRECV** : It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
- **SOCK_FIN_WAIT** : Connection state
- **SOCK_CLOSING** : Closing state

- **SOCK_TIME_WAIT** : Closing state
- **SOCK_LAST_ACK** : Closing state

Definition at line **497** of file **w5500.h**.

#define Sn_PORT (N) (_W5500_IO_BASE_ + (0x0004 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

source port register(R/W)

Sn_PORT configures the source port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. It should be set before OPEN command is ordered.

Definition at line **505** of file **w5500.h**.

#define Sn_DHAR (N) (_W5500_IO_BASE_ + (0x0006 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Peer MAC register address(R/W)

Sn_DHAR configures the destination hardware address of Socket n when using SEND_MAC command in UDP mode or it indicates that it is acquired in ARP-process by CONNECT/SEND command.

Definition at line **513** of file **w5500.h**.

#define Sn_DIPR (N) (_W5500_IO_BASE_ + (0x000C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Peer IP register address(R/W)

Sn_DIPR configures or indicates the destination IP address of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP client mode, it configures an IP address of TCP serverbefore CONNECT command. In TCP server mode, it indicates an IP

address of TCP clientafter successfully establishing connection. In UDP mode, it configures an IP address of peer to be received the UDP packet by SEND or SEND_MAC command.

Definition at line **523** of file **w5500.h**.

#define Sn_DPORT (N) (_W5500_IO_BASE_ + (0x0010 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Peer port register address(R/W)

Sn_DPORT configures or indicates the destination port number of Socket n. It is valid when Socket n is used in TCP/UDP mode. In TCP clientmode, it configures the listen port number of TCP serverbefore CONNECT command. In TCP Servermode, it indicates the port number of TCP client after successfully establishing connection. In UDP mode, it configures the port number of peer to be transmitted the UDP packet by SEND/SEND_MAC command.

Definition at line **533** of file **w5500.h**.

#define Sn_MSSR (N) (_W5500_IO_BASE_ + (0x0012 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Maximum Segment Size(Sn_MSSR0) register address(R/W)

Sn_MSSR configures or indicates the MTU(Maximum Transfer Unit) of Socket n.

Definition at line **540** of file **w5500.h**.

#define Sn_TOS (N) (_W5500_IO_BASE_ + (0x0015 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

IP Type of Service(TOS) Register(R/W)

Sn_TOS configures the TOS(Type Of Service field in IP Header) of Socket n. It is set before OPEN command.

Definition at line **550** of file **w5500.h**.

#define Sn_TTL (N) (_W5500_IO_BASE_ + (0x0016 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

IP Time to live(TTL) Register(R/W)

Sn_TTL configures the TTL(Time To Live field in IP header) of Socket n. It is set before OPEN command.

Definition at line **557** of file **w5500.h**.

#define Sn_RXBUF_SIZE (N) (_W5500_IO_BASE_ + (0x001E << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Receive memory size register(R/W)

Sn_RXBUF_SIZE configures the RX buffer block size of Socket n. Socket n RX Buffer Block size can be configured with 1,2,4,8, and 16 Kbytes. If a different size is configured, the data cannot be normally received from a peer. Although Socket n RX Buffer Block size is initially configured to 2Kbytes, user can re-configure its size using **Sn_RXBUF_SIZE**. The total sum of **Sn_RXBUF_SIZE** can not be exceed 16Kbytes. When exceeded, the data reception error is occurred.

Definition at line **576** of file **w5500.h**.

#define Sn_TXBUF_SIZE (N) (_W5500_IO_BASE_ + (0x001F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Transmit memory size register(R/W)

Sn_TXBUF_SIZE configures the TX buffer block size of Socket n. Socket n TX Buffer Block size can be configured with 1,2,4,8, and 16 Kbytes. If a different size is configured, the data can't be normally transmitted to a peer. Although Socket n TX Buffer Block size is initially configured to 2Kbytes, user can be re-configure its size using **Sn_TXBUF_SIZE**. The total sum of **Sn_TXBUF_SIZE** can not be exceed 16Kbytes. When exceeded, the data transmission error is occurred.

Definition at line **587** of file **w5500.h**.

#define Sn_TX_FSR (N) (_W5500_IO_BASE_ + (0x0020 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Transmit free memory size register(R)

Sn_TX_FSR indicates the free size of Socket n TX Buffer Block. It is initialized to the configured size by **Sn_TXBUF_SIZE**. Data bigger than **Sn_TX_FSR** should not be saved in the Socket n TX Buffer because the bigger data overwrites the previous saved data not yet sent. Therefore, check before saving the data to the Socket n TX Buffer, and if data is equal or smaller than its checked size, transmit the data with SEND/SEND_MAC command after saving the data in Socket n TX buffer. But, if data is bigger than its checked size, transmit the data after dividing into the checked size and saving in the Socket n TX buffer.

Definition at line **598** of file **w5500.h**.

#define Sn_TX_RD (N) (_W5500_IO_BASE_ + (0x0022 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Transmit memory read pointer register address(R)

Sn_TX_RD is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001, it is re-initialized while connecting with TCP. After its initialization, it is auto-increased by SEND command. SEND command transmits the saved data from the

current **Sn_TX_RD** to the **Sn_TX_WR** in the Socket n TX Buffer. After transmitting the saved data, the SEND command increases the **Sn_TX_RD** as same as the **Sn_TX_WR**. If its increment value exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **610** of file **w5500.h**.

#define Sn_TX_WR (N) (_W5500_IO_BASE_ + (0x0024 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Transmit memory write pointer register address(R/W)

Sn_TX_WR is initialized by OPEN command. However, if **Sn_MR(P[3:0])** is TCP mode(001, it is re-initialized while connecting with TCP.
It should be read or be updated like as follows.

1. Read the starting address for saving the transmitting data.
2. Save the transmitting data from the starting address of Socket n TX buffer.
3. After saving the transmitting data, update **Sn_TX_WR** to the increased value as many as transmitting data size. If the increment value exceeds the maximum value 0xFFFF(greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.
4. Transmit the saved data in Socket n TX Buffer by using SEND/SEND command

Definition at line **624** of file **w5500.h**.

#define Sn_RX_RSR (N) (_W5500_IO_BASE_ + (0x0026 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Received data size register(R)

Sn_RX_RSR indicates the data size received and saved in Socket n RX Buffer. **Sn_RX_RSR** does not exceed the **Sn_RXBUF_SIZE** and is calculated as the difference between Socket n RX Write Pointer (**Sn_RX_WR**)and Socket n RX Read Pointer (**Sn_RX_RD**)

Definition at line **633** of file **w5500.h**.

#define Sn_RX_RD (N) (_W5500_IO_BASE_ + (0x0028 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Read point of Receive memory(R/W)

Sn_RX_RD is initialized by OPEN command. Make sure to be read or updated as follows.

1. Read the starting save address of the received data.
2. Read data from the starting address of Socket n RX Buffer.
3. After reading the received data, Update **Sn_RX_RD** to the increased value as many as the reading size. If the increment value exceeds the maximum value 0xFFFF, that is, is greater than 0x10000 and the carry bit occurs, update with the lower 16bits value ignored the carry bit.
4. Order RECV command is for notifying the updated **Sn_RX_RD** to W5500.

Definition at line **646** of file **w5500.h**.

#define Sn_RX_WR (N) (_W5500_IO_BASE_ + (0x002A << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Write point of Receive memory(R)

Sn_RX_WR is initialized by OPEN command and it is auto-increased by the data reception. If the increased value exceeds the maximum value 0xFFFF, (greater than 0x10000 and the carry bit occurs), then the carry bit is ignored and will automatically update with the lower 16bits value.

Definition at line **655** of file **w5500.h**.

#define Sn_IMR (N) (_W5500_IO_BASE_ + (0x002C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

socket interrupt mask register(R)

Sn_IMR masks the interrupt of Socket n. Each bit corresponds to each bit of **Sn_IR**. When a Socket n Interrupt is occurred and the corresponding bit of **Sn_IMR** is the corresponding bit of **Sn_IR** becomes When both the corresponding bit of **Sn_IMR** and **Sn_IR** are and the n-th bit of **IR** is Host is interrupted by asserted INTn PIN to low.

Definition at line **665** of file **w5500.h**.

#define Sn_FRAG (N) (_W5500_IO_BASE_ + (0x002D << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Fragment field value in IP header register(R/W)

Sn_FRAG configures the FRAG(Fragment field in IP header).

Definition at line **672** of file **w5500.h**.

#define Sn_KPALVTR (N) (_W5500_IO_BASE_ + (0x002F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

Keep Alive Timer register(R/W)

Sn_KPALVTR configures the transmitting timer of KEEP ALIVE(KA)packet of SOCKETn. It is valid only in TCP mode, and ignored in other modes. The time unit is 5s. KA packet is transmittable after **Sn_SR** is changed to SOCK_ESTABLISHED and after the data is transmitted or received to/from a peer at least once. In case of '**Sn_KPALVTR** > 0', W5500 automatically transmits KA

packet after time-period for checking the TCP connection (Auto-keepalive-process). In case of '**Sn_KPALVTR** = 0', Auto-keep-alive-process will not operate, and KA packet can be transmitted by SEND_KEEP command by the host (Manual-keep-alive-process). Manual-keep-alive-process is ignored in case of '**Sn_KPALVTR** > 0'.

Definition at line **685** of file **w5500.h**.

Socket APIs

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[detail level 1 2]

▾ __WIZCHIP	The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**
_CRIS	
_CS	
_IF	
wiz_NetInfo_t	
wiz_NetTimeout_t	
wiz_PhyConf_t	

Socket APIs

Class Index

W | _

_
__WIZCHIP
__WIZCHIP::_CRIS
__WIZCHIP::_CS
__WIZCHIP::_IF
W
wiz_NetInfo_t
wiz_NetTimeou
wiz_PhyConf_

W | _

Socket APIs

Here is a list of all class members with links to the classes they belong to:

- _deselect : **__WIZCHIP::_CS**
- _enter : **__WIZCHIP::_CRIS**
- _exit : **__WIZCHIP::_CRIS**
- _read_burst : **__WIZCHIP::_IF**
- _read_byte : **__WIZCHIP::_IF**
- _read_data : **__WIZCHIP::_IF**
- _select : **__WIZCHIP::_CS**
- _write_burst : **__WIZCHIP::_IF**
- _write_byte : **__WIZCHIP::_IF**
- _write_data : **__WIZCHIP::_IF**
- BUS : **__WIZCHIP::_IF**
- by : **wiz_PhyConf_t**
- CRIS : **__WIZCHIP**
- CS : **__WIZCHIP**
- dhcp : **wiz_NetInfo_t**
- dns : **wiz_NetInfo_t**
- duplex : **wiz_PhyConf_t**
- gw : **wiz_NetInfo_t**
- id : **__WIZCHIP**
- IF : **__WIZCHIP**
- if_mode : **__WIZCHIP**
- ip : **wiz_NetInfo_t**
- mac : **wiz_NetInfo_t**
- mode : **wiz_PhyConf_t**
- retry_cnt : **wiz_NetTimeout_t**
- sn : **wiz_NetInfo_t**
- speed : **wiz_PhyConf_t**
- SPI : **__WIZCHIP::_IF**
- time_100us : **wiz_NetTimeout_t**

Socket APIs

- _deselect : **__WIZCHIP::_CS**
- _enter : **__WIZCHIP::_CRIS**
- _exit : **__WIZCHIP::_CRIS**
- _read_burst : **__WIZCHIP::_IF**
- _read_byte : **__WIZCHIP::_IF**
- _read_data : **__WIZCHIP::_IF**
- _select : **__WIZCHIP::_CS**
- _write_burst : **__WIZCHIP::_IF**
- _write_byte : **__WIZCHIP::_IF**
- _write_data : **__WIZCHIP::_IF**
- BUS : **__WIZCHIP::_IF**
- by : **wiz_PhyConf_t**
- CRIS : **__WIZCHIP**
- CS : **__WIZCHIP**
- dhcp : **wiz_NetInfo_t**
- dns : **wiz_NetInfo_t**
- duplex : **wiz_PhyConf_t**
- gw : **wiz_NetInfo_t**
- id : **__WIZCHIP**
- IF : **__WIZCHIP**
- if_mode : **__WIZCHIP**
- ip : **wiz_NetInfo_t**
- mac : **wiz_NetInfo_t**
- mode : **wiz_PhyConf_t**
- retry_cnt : **wiz_NetTimeout_t**
- sn : **wiz_NetInfo_t**
- speed : **wiz_PhyConf_t**
- SPI : **__WIZCHIP::_IF**
- time_100us : **wiz_NetTimeout_t**

Socket APIs

File List

Here is a list of all files with brief descriptions:

[detail level 1 2 3]

▾ Ethernet	
▾ W5100	
w5100.c	W5100 HAL Interface
w5100.h	W5100 HAL Header File
▾ W5200	
w5200.c	W5200 HAL Interface
w5200.h	W5200 HAL Header File
▾ W5300	
w5300.c	
w5300.h	W5300 HAL implement File
▾ W5500	
w5500.c	W5500 HAL Interface
w5500.h	W5500 HAL Header File
socket.c	SOCKET APIs Implements file
socket.h	SOCKET APIs Header file
wizchip_conf.c	WIZCHIP Config Header File
wizchip_conf.h	WIZCHIP Config Header File

Socket APIs

Ethernet Directory Reference

Directories

directory	**W5100**
directory	**W5200**
directory	**W5300**
directory	**W5500**

Files

file	**socket.c** [code] SOCKET APIs Implements file.
file	**socket.h** [code] SOCKET APIs Header file.
file	**wizchip_conf.c** [code] WIZCHIP Config Header File.
file	**wizchip_conf.h** [code] WIZCHIP Config Header File.

Socket APIs

W5100 Directory Reference

Files

file **w5100.c** [code]
W5100 HAL Interface.

file **w5100.h** [code]
W5100 HAL Header File.

Socket APIs

w5100.c File Reference

W5100 HAL Interface. More...

```
#include "w5100.h"
```

Go to the source code of this file.

Detailed Description

W5100 HAL Interface.

Version
1.0.0

Date
2013/10/21

Revision history
<2013/10/21> 1st Release

Author
MidnightCow

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

```
* Redistributions of source code must retain the a
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED

WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **w5100.c**.

Socket APIs

Macros | Functions

w5100.h File Reference

W5100 HAL Header File. More...

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

Go to the source code of this file.

Macros

#define **_WIZCHIP_SN_BASE_** (0x0400)

#define **_WIZCHIP_SN_SIZE_** (0x0100)

#define **_WIZCHIP_IO_TXBUF_** (0x4000) /* Internal Tx buffer addres
the iinchip */

#define **_WIZCHIP_IO_RXBUF_** (0x6000) /* Internal Rx buffer addres
the iinchip */

#define **WIZCHIP_CREG_BLOCK** 0x00
Common register block. More...

#define **WIZCHIP_SREG_BLOCK**(N) (**_WIZCHIP_SN_BASE_**+ **_WIZCHIP_SN_SIZE_***N)
Socket N register block. More...

#define **WIZCHIP_OFFSET_INC**(ADDR, N) (ADDR + N)
Increase offset address. More...

#define **_W5100_IO_BASE_** **_WIZCHIP_IO_BASE_**

#define **IINCHIP_READ**(ADDR) **WIZCHIP_READ**(ADDR)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE**(ADDR, VAL) **WIZCHIP_WRITE**(ADDR,VAL)
The defined for legacy chip driver. More...

#define **IINCHIP_READ_BUF**(ADDR, BUF, LEN) **WIZCHIP_READ_BUF**(ADDR,BUF,LEN)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE_BUF**(ADDR, BUF, LEN) **WIZCHIP_WRITE**(ADDR,BUF,LEN)

The defined for legacy chip driver. More...

#define **MR** (**_WIZCHIP_IO_BASE_** + (0x0000))
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and
More...

#define **GAR** (**_W5100_IO_BASE_** + (0x0001))
Gateway IP Register address(R/W) More...

#define **SUBR** (**_W5100_IO_BASE_** + (0x0005))
Subnet mask Register address(R/W) More...

#define **SHAR** (**_W5100_IO_BASE_** + (0x0009))
Source MAC Register address(R/W) More...

#define **SIPR** (**_W5100_IO_BASE_** + (0x000F))
Source IP Register address(R/W) More...

#define **IR** (**_W5100_IO_BASE_** + (0x0015))
Interrupt Register(R/W) More...

#define **_IMR_** (**_W5100_IO_BASE_** + (0x0016))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (**_W5100_IO_BASE_** + (0x0017))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5100_IO_BASE_** + (0x0019))
Retry count register(R/W) More...

#define **RMSR** (**_W5100_IO_BASE_** + (0x001A))

#define **TMSR** (**_W5100_IO_BASE_** + (0x001B))

#define **PATR** (**_W5100_IO_BASE_** + (0x001C))

PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PTIMER** (**_W5100_IO_BASE_** + (0x0028))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PMAGIC** (**_W5100_IO_BASE_** + (0x0029))
PPP LCP Magic number register in PPPoE mode(R) More...

#define **UIPR0** (**_W5100_IO_BASE_** + (0x002A))

#define **UPORT0** (_W5100_IO_BASE + (0x002E))

#define **Sn_MR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0000))
socket Mode register(R/W) More...

#define **Sn_CR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0001))
Socket command register(R/W) More...

#define **Sn_IR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**
+ (0x0002))
Socket interrupt register(R) More...

#define **Sn_SR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0003))
Socket status register(R) More...

#define **Sn_PORT**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0004))
source port register(R/W) More...

#define **Sn_DHAR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0006))
Peer MAC register address(R/W) More...

Sn_DIPR(sn) (**_W5100_IO_BASE_** +

#define **WIZCHIP_SREG_BLOCK**(sn) + (0x000C))

Peer IP register address(R/W) More...

#define **Sn_DPORT**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0010))
Peer port register address(R/W) More...

#define **Sn_MSSR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0012))
Maximum Segment Size(Sn_MSSR0) register address(R/W) More...

#define **Sn_PROTO**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0014))
IP Protocol(PROTO) Register(R/W) More...

#define **Sn_TOS**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + 0x0015)
IP Type of Service(TOS) Register(R/W) More...

#define **Sn_TTL**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0016))
IP Time to live(TTL) Register(R/W) More...

#define **Sn_TX_FSR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0020))
Transmit free memory size register(R) More...

#define **Sn_TX_RD**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0022))
Transmit memory read pointer register address(R) More...

#define **Sn_TX_WR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0024))
Transmit memory write pointer register address(R/W) More...

Sn_RX_RSR(sn) (**_W5100_IO_BASE_** +

#define **WIZCHIP_SREG_BLOCK**(sn) + (0x0026))

Received data size register(R) More...

#define **Sn_RX_RD**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x0028))
Read point of Receive memory(R/W) More...

#define **Sn_RX_WR**(sn) (**_W5100_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(sn) + (0x002A))
Write point of Receive memory(R) More...

#define **MR_RST** 0x80
Reset. More...

#define **MR_PB** 0x10
Ping block. More...

#define **MR_PPPOE** 0x08
Enable PPPoE. More...

#define **MR_AI** 0x02
Address Auto-Increment in Indirect Bus Interface. More...

#define **MR_IND** 0x01
Indirect Bus Interface mode. More...

#define **IR_CONFLICT** 0x80
Check IP conflict. More...

#define **IR_UNREACH** 0x40
Get the destination unreachable message in UDP sending. Mo

#define **IR_PPPoE** 0x20
Get the PPPoE close message. More...

#define **IR_SOCK**(sn) (0x01 << sn)

check socket interrupt More...

#define **Sn_MR_CLOSE** 0x00
Unused socket. More...

#define **Sn_MR_TCP** 0x01
TCP. More...

#define **Sn_MR_UDP** 0x02
UDP. More...

#define **Sn_MR_IPRAW** 0x03
IP LAYER RAW SOCK. More...

#define **Sn_MR_MACRAW** 0x04
MAC LAYER RAW SOCK. More...

#define **Sn_MR_PPPoE** 0x05
PPPoE. More...

#define **Sn_MR_ND** 0x20
No Delayed Ack(TCP), Multicast flag. More...

#define **Sn_MR_MC** **Sn_MR_ND**
Support UDP Multicasting. More...

#define **Sn_MR_MF** 0x40
MAC filter enable in **Sn_MR_MACRAW** mode. More...

#define **Sn_MR_MFEN** **Sn_MR_MF**

#define **Sn_MR_MULTI** 0x80
Support UDP Multicasting. More...

#define **Sn_CR_OPEN** 0x01
Initialize or open socket. More...

#define **Sn_CR_LISTEN** 0x02
Wait connection request in TCP mode(Server mode) More...

#define **Sn_CR_CONNECT** 0x04
Send connection request in TCP mode(Client mode) More...

#define **Sn_CR_DISCON** 0x08
Send closing request in TCP mode. More...

#define **Sn_CR_CLOSE** 0x10
Close socket. More...

#define **Sn_CR_SEND** 0x20
Update TX buffer pointer and send data. More...

#define **Sn_CR_SEND_MAC** 0x21
Send data with MAC address, so without ARP process. More..

#define **Sn_CR_SEND_KEEP** 0x22
Send keep alive message. More...

#define **Sn_CR_RECV** 0x40
Update RX buffer pointer and receive data. More...

#define **Sn_CR_PCON** 0x23
PPPoE connection. More...

#define **Sn_CR_PDISCON** 0x24
Closes PPPoE connection. More...

#define **Sn_CR_PCR** 0x25
REQ message transmission. More...

#define **Sn_CR_PCN** 0x26
NAK massage transmission. More...

#define **Sn_CR_PCJ** 0x27
REJECT message transmission. More...

#define **Sn_IR_PRECV** 0x80
PPP Receive Interrupt. More...

#define **Sn_IR_PFAIL** 0x40
PPP Fail Interrupt. More...

#define **Sn_IR_PNEXT** 0x20
PPP Next Phase Interrupt. More...

#define **Sn_IR_SENDOK** 0x10
SEND_OK Interrupt. More...

#define **Sn_IR_TIMEOUT** 0x08
TIMEOUT Interrupt. More...

#define **Sn_IR_RECV** 0x04
RECV Interrupt. More...

#define **Sn_IR_DISCON** 0x02
DISCON Interrupt. More...

#define **Sn_IR_CON** 0x01
CON Interrupt. More...

#define **SOCK_CLOSED** 0x00
Closed. More...

#define **SOCK_INIT** 0x13
Initiate state. More...

#define **SOCK_LISTEN** 0x14
Listen state. More...

#define **SOCK_SYNSENT** 0x15
Connection state. More...

#define **SOCK_SYNRECV** 0x16
Connection state. More...

#define **SOCK_ESTABLISHED** 0x17
Success to connect. More...

#define **SOCK_FIN_WAIT** 0x18
Closing state. More...

#define **SOCK_CLOSING** 0x1A
Closing state. More...

#define **SOCK_TIME_WAIT** 0x1B
Closing state. More...

#define **SOCK_CLOSE_WAIT** 0x1C
Closing state. More...

#define **SOCK_LAST_ACK** 0x1D
Closing state. More...

#define **SOCK_UDP** 0x22
UDP socket. More...

#define **SOCK_IPRAW** 0x32
IP raw mode socket. More...

#define **SOCK_MACRAW** 0x42
MAC raw mode socket. More...

#define **SOCK_PPPOE** 0x5F
PPPoE mode socket. More...

#define **IPPROTO_IP** 0
Dummy for IP. More...

#define **IPPROTO_ICMP** 1
Control message protocol. More...

#define **IPPROTO_IGMP** 2
Internet group management protocol. More...

#define **IPPROTO_GGP** 3
GW^2 (deprecated) More...

#define **IPPROTO_TCP** 6
TCP. More...

#define **IPPROTO_PUP** 12
PUP. More...

#define **IPPROTO_UDP** 17
UDP. More...

#define **IPPROTO_IDP** 22
XNS idp. More...

#define **IPPROTO_ND** 77
UNOFFICIAL net disk protocol. More...

#define **IPPROTO_RAW** 255
Raw IP packet. More...

#define **WIZCHIP_CRITICAL_ENTER**() WIZCHIP.CRIS._enter()
Enter a critical section. More...

#define **WIZCHIP_CRITICAL_EXIT**() WIZCHIP.CRIS._exit()
Exit a critical section. More...

#define **setMR**(mr) (*((uint8_t*)**MR**) = mr)
Set Mode Register. More...

#define **getMR**() (*(uint8_t*)**MR**)
Get **MR**. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set **GAR**. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get **GAR**. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**,subr,4)
Set **SUBR**. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get **SUBR**. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set **SHAR**. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get **SHAR**. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set **SIPR**. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get **SIPR**. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xA0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xA0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**_IMR_**, imr)
Set *IMR* register. More...

#define **getIMR**() **WIZCHIP_READ**(**_IMR_**)
Get *IMR* register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(**_RTR_**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **setRMSR**(rmsr) **WIZCHIP_WRITE**(**RMSR**)
Get **RMSR** register. More...

#define **getRMSR**() **WIZCHIP_READ**()
Get **RMSR** register. More...

#define **setTMSR**(rmsr) **WIZCHIP_WRITE**(**TMSR**)
Get **TMSR** register. More...

#define **getPATR**() (((uint16_t)**WIZCHIP_READ**(**PATR**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PATR**,1)))
Get **TMSR** register. More...

#define **getPPPALGO**() **WIZCHIP_READ**(**PPPALGO**)
Get **PPPALGO** register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)
Get **PMAGIC** register. More...

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ir)
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) **WIZCHIP_READ**(**Sn_IR**(sn))
Get **Sn_IR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define **setSn_PORT**(sn, port)
Set **Sn_PORT** register. More...

#define **getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn)
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn)
Get **Sn_PORT** register. More...

#define **setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s
dhar, 6)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(sı
dhar, 6)
Get **Sn_DHAR** register. More...

#define **setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn),
4)
Set **Sn_DIPR** register. More...

#define **getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn),
4)
Get **Sn_DIPR** register. More...

#define **setSn_DPORT**(sn, dport)
Set **Sn_DPORT** register. More...

#define **getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1)
Get **Sn_DPORT** register. More...

#define **setSn_MSSR**(sn, mss)
Set **Sn_MSSR** register. More...

#define **getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn)
Get **Sn_MSSR** register. More...

#define **setSn_PROTO**(sn, proto) **WIZCHIP_WRITE**(**Sn_TOS**(sn), to
Set **Sn_PROTO** register. More...

#define **getSn_PROTO**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn))
Get **Sn_PROTO** register. More...

#define **setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos)
Set **Sn_TOS** register. More...

#define **getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn))
Get **Sn_TOS** register. More...

#define **setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl)
Set **Sn_TTL** register. More...

#define **getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn))
Get **Sn_TTL** register. More...

#define **setSn_RXMEM_SIZE**(sn, rxmemsize) **WIZCHIP_WRITE**(**RM**
(**WIZCHIP_READ**(**RMSR**) & ~(0x03 << (2*sn))) | (rxmemsize <
(2*sn)))
Set **Sn_RXMEM_SIZE** register. More...

#define **setSn_RXBUF_SIZE**(sn,
rxmemsize) **setSn_RXMEM_SIZE**(sn,rxmemsize)

#define **getSn_RXMEM_SIZE**(sn) ((**WIZCHIP_READ**(**RMSR**) & (0x0
(2*sn))) >> (2*sn))
Get **Sn_RXMEM_SIZE** register. More...

#define **getSn_RXBUF_SIZE**(sn) **getSn_RXMEM_SIZE**(sn)

#define **setSn_TXMEM_SIZE**(sn, txmemsize) **WIZCHIP_WRITE**(**TM**
(**WIZCHIP_READ**(**TMSR**) & ~(0x03 << (2*sn))) | (txmemsize <
(2*sn)))
Set **Sn_TXMEM_SIZE** register. More...

#define **setSn_TXBUF_SIZE**(sn,
txmemsize) **setSn_TXMEM_SIZE**(sn,txmemsize)

#define **getSn_TXMEM_SIZE**(sn) ((**WIZCHIP_READ**(**TMSR**) & (0x03
(2*sn))) >> (2*sn))
Get **Sn_TXMEM_SIZE** register. More...

#define **getSn_TXBUF_SIZE**(sn) **getSn_TXMEM_SIZE**(sn)

#define **getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sn),1))
Get **Sn_TX_RD** register. More...

#define **setSn_TX_WR**(sn, txwr)
Set **Sn_TX_WR** register. More...

#define **getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1)
Get **Sn_TX_WR** register. More...

#define **setSn_RX_RD**(sn, rxrd)
Set **Sn_RX_RD** register. More...

#define **getSn_RX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1))
Get **Sn_RX_RD** register. More...

#define **setSn_RX_WR**(sn, rxwr)
Set **Sn_RX_WR** register. More...

#define **getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1)
Get **Sn_RX_WR** register. More...

#define **setSn_FRAG**(sn, frag)

	Set **Sn_FRAG** register. More...
#define	**getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn) Get **Sn_FRAG** register. More...
#define	**getSn_RxMAX**(sn) ((uint16_t)(1 << **getSn_RXMEM_SIZE**(sr << 10) Get the max RX buffer size of socket sn. More...
#define	**getSn_TxMAX**(sn) ((uint16_t)(1 << **getSn_TXMEM_SIZE**(sn 10) Get the max TX buffer size of socket sn. More...
#define	**getSn_RxMASK**(sn) (**getSn_RxMAX**(sn) - 1) Get the mask of socket sn RX buffer. More...
#define	**getSn_TxMASK**(sn) (**getSn_TxMAX**(sn) - 1) Get the mask of socket sn TX buffer. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
uint16_t	**getSn_TX_FSR** (uint8_t sn) Get **Sn_TX_FSR** register. More...
uint16_t	**getSn_RX_RSR** (uint8_t sn) Get **Sn_RX_RSR** register. More...
uint32_t	**getSn_RxBASE** (uint8_t sn) Get the base address of socket sn RX buffer. More...
uint32_t	**getSn_TxBASE** (uint8_t sn) Get the base address of socket sn TX buffer. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...

void **wiz_recv_ignore** (uint8_t sn, uint16_t len)
It discard the received data in RX memory. More...

Detailed Description

W5100 HAL Header File.

Version
1.0.0

Date
2013/10/21

Revision history
<2013/10/21> 1st Release

Author
MidnightCow

Copyright

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

```
* Redistributions of source code must retain the a
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **w5100.h**.

Macro Definition Documentation

#define _WIZCHIP_SN_BASE_ (0x0400)

Definition at line **50** of file **w5100.h**.

#define _WIZCHIP_SN_SIZE_ (0x0100)

Definition at line **51** of file **w5100.h**.

#define _WIZCHIP_IO_TXBUF_ (0x4000) /* Internal Tx buffer address of the iinchip */

Definition at line **52** of file **w5100.h**.

#define _WIZCHIP_IO_RXBUF_ (0x6000) /* Internal Rx buffer address of the iinchip */

Definition at line **53** of file **w5100.h**.

#define WIZCHIP_CREG_BLOCK 0x00

Common register block.

Definition at line **56** of file **w5100.h**.

#define WIZCHIP_SREG_BLOCK (N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_*N)

Socket N register block.

Definition at line **57** of file **w5100.h**.

```
#define WIZCHIP_OFFSET_INC (  ADDR,
                              N
                           )    (ADDR + N)
```

Increase offset address.

Definition at line **59** of file **w5100.h**.

```
#define _W5100_IO_BASE_   _WIZCHIP_IO_BASE_
```

Definition at line **62** of file **w5100.h**.

```
#define IINCHIP_READ (  ADDR )   WIZCHIP_READ(ADDR)
```

The defined for legacy chip driver.

Definition at line **76** of file **w5100.h**.

```
#define IINCHIP_WRITE (  ADDR,
                         VAL
                      )    WIZCHIP_WRITE(ADDR,VAL)
```

The defined for legacy chip driver.

Definition at line **77** of file **w5100.h**.

```
#define
IINCHIP_READ_BUF   (  ADDR,
                      BUF,
```

```
                                LEN
                            )   WIZCHIP_READ_BUF(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **78** of file **w5100.h**.

```
#define
IINCHIP_WRITE_BUF       (   ADDR,
                            BUF,
                            LEN
                        )   WIZCHIP_WRITE(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **79** of file **w5100.h**.

```
#define RMSR   (_W5100_IO_BASE_ + (0x001A))
```

Definition at line **283** of file **w5100.h**.

```
#define TMSR   (_W5100_IO_BASE_ + (0x001B))
```

Definition at line **284** of file **w5100.h**.

```
#define UIPR0   (_W5100_IO_BASE_ + (0x002A))
```

Definition at line **310** of file **w5100.h**.

```
#define UPORT0   (_W5100_IO_BASE + (0x002E))
```

Definition at line **311** of file **w5100.h**.

#define MR_RST 0x80

Reset.

If this bit is All internal registers will be initialized. It will be automatically cleared as after S/W reset.reset

Definition at line **577** of file **w5100.h**.

Referenced by **wizchip_sw_reset()**.

#define MR_PB 0x10

Ping block.

0 : Disable Ping block
1 : Enable Ping block
If the bit is it blocks the response to a ping request.ping block

Definition at line **586** of file **w5100.h**.

#define MR_PPPOE 0x08

Enable PPPoE.

0 : DisablePPPoE mode
1 : EnablePPPoE mode
If you use ADSL, this bit should be '1'.enable pppoe

Definition at line **594** of file **w5100.h**.

#define MR_AI 0x02

Address Auto-Increment in Indirect Bus Interface.

0 : Disable auto-increment
1 : Enable auto-incremente
At the Indirect Bus Interface mode, if this bit is set as ��1��, the address will be automatically increased by 1 whenever read and write are performed.auto-increment in indirect mode

Definition at line **603** of file **w5100.h**.

#define MR_IND 0x01

Indirect Bus Interface mode.

0 : Disable Indirect bus Interface mode
1 : Enable Indirect bus Interface mode
If this bit is set as ��1��, Indirect Bus Interface mode is set.enable indirect mode

Definition at line **611** of file **w5100.h**.

Referenced by **wizchip_sw_reset()**.

#define IR_CONFLICT 0x80

Check IP conflict.

Bit is set as when own source IP address is same with the sender IP address in the received ARP request.check ip confict

Definition at line **618** of file **w5100.h**.

#define IR_UNREACH 0x40

Get the destination unreachable message in UDP sending.

When receiving the ICMP (Destination port unreachable) packet, this bit is set as When this bit is Destination Information such as IP address and Port number may be checked with the corresponding **UIPR** & UPORTR.check destination unreachable

Definition at line **625** of file **w5100.h**.

#define IR_PPPoE 0x20

Get the PPPoE close message.

When PPPoE is disconnected during PPPoE mode, this bit is set.get the PPPoE close message

Definition at line **631** of file **w5100.h**.

#define IR_SOCK (sn) (0x01 << sn)

check socket interrupt

Definition at line **633** of file **w5100.h**.

#define Sn_MR_CLOSE 0x00

Unused socket.

This configures the protocol mode of Socket n.unused socket

Definition at line **642** of file **w5100.h**.

#define Sn_MR_TCP 0x01

TCP.

This configures the protocol mode of Socket n.TCP

Definition at line **648** of file **w5100.h**.

Referenced by **close()**, **connect()**, **disconnect()**, **getsockopt()**, **listen()**, **recv()**, **send()**, **setsockopt()**, and **socket()**.

#define Sn_MR_UDP 0x02

UDP.

This configures the protocol mode of Socket n.UDP

Definition at line **654** of file **w5100.h**.

Referenced by **close()**, **recvfrom()**, **sendto()**, and **socket()**.

#define Sn_MR_IPRAW 0x03

IP LAYER RAW SOCK.

Definition at line **655** of file **w5100.h**.

Referenced by **recvfrom()**, and **socket()**.

#define Sn_MR_MACRAW 0x04

MAC LAYER RAW SOCK.

This configures the protocol mode of Socket n.

Note
MACRAW mode should be only used in Socket 0.MAC LAYER RAW SOCK

Definition at line **662** of file **w5100.h**.

Referenced by **recvfrom()**, **sendto()**, and **socket()**.

#define Sn_MR_PPPoE 0x05

PPPoE.

This configures the protocol mode of Socket n.

Note
PPPoE mode should be only used in Socket 0.PPPoE

Definition at line **669** of file **w5100.h**.

Referenced by **recvfrom()**, and **socket()**.

#define Sn_MR_ND 0x20

No Delayed Ack(TCP), Multicast flag.

0 : Disable No Delayed ACK option
1 : Enable No Delayed ACK option
This bit is applied only during TCP mode (P[3:0] = 001).
When this bit is It sends the ACK packet without delay as soon as a Data packet is received from a peer.
When this bit is It sends the ACK packet after waiting for the timeout time configured by *RTR*.No Delayed Ack(TCP) flag

Definition at line **679** of file **w5100.h**.

#define Sn_MR_MC Sn_MR_ND

Support UDP Multicasting.

0 : using IGMP version 2
1 : using IGMP version 1
This bit is applied only during UDP mode(P[3:0] = 010 and MULTI = '1') It configures the version for IGMP messages (Join/Leave/Report).Select IGMP version 1(0) or 2(1)

Definition at line **688** of file **w5100.h**.

#define Sn_MR_MF 0x40

MAC filter enable in **Sn_MR_MACRAW** mode.

0 : disable MAC Filtering
1 : enable MAC Filtering
This bit is applied only during MACRAW mode(P[3:0] = 100.
When set as W5100 can only receive broadcasting packet or packet sent to itself. When this bit is W5100 can receive all packets on Ethernet. If user wants to implement Hybrid TCP/IP stack, it is recommended that this bit is set as for reducing host overhead to process the all received packets.Use MAC filter

Definition at line **700** of file **w5100.h**.

#define Sn_MR_MFEN Sn_MR_MF

Definition at line **701** of file **w5100.h**.

#define Sn_MR_MULTI 0x80

Support UDP Multicasting.

0 : disable Multicasting
1 : enable Multicasting
This bit is applied only during UDP mode(P[3:0] = 010).
To use multicasting, **Sn_DIPR** & **Sn_DPORT** should be respectively configured with the multicast group IP address & port number before Socket n is opened by OPEN command of Sn_CR.support multicating

Definition at line **713** of file **w5100.h**.

#define Sn_CR_OPEN 0x01

Initialize or open socket.

Socket n is initialized and opened according to the protocol selected in **Sn_MR(P3:P0)**. The table below shows the value of **Sn_SR** corresponding to **Sn_MR**.

Sn_MR (P[3:0])	**Sn_SR**
Sn_MR_CLOSE (000)	–
Sn_MR_TCP (001)	SOCK_INIT (0x13)
Sn_MR_UDP (010)	SOCK_UDP (0x22)
S0_MR_IPRAW (011)	SOCK_IPRAW (0x32)
S0_MR_MACRAW (100)	SOCK_MACRAW (0x42)
S0_MR_PPPoE (101)	SOCK_PPPoE (0x5F)

initialize or open socket

Definition at line **730** of file **w5100.h**.

Referenced by **close()**, and **socket()**.

#define Sn_CR_LISTEN 0x02

Wait connection request in TCP mode(Server mode)

This is valid only in TCP mode (**Sn_MR(P3:P0)** = **Sn_MR_TCP**).// In this mode, Socket n operates as a 'TCP server' and waits for connection-request (SYN packet) from any 'TCP client'.// The **Sn_SR** changes the state from SOCK_INIT to SOCKET_LISTEN.// When a 'TCP client' connection request is successfully established, the **Sn_SR** changes from SOCK_LISTEN to SOCK_ESTABLISHED and the **Sn_IR(0)** becomes But when a 'TCP client' connection request is failed, **Sn_IR(3)** becomes and the status of **Sn_SR** changes to SOCK_CLOSED.wait connection request in tcp mode(Server mode)

Definition at line **741** of file **w5100.h**.

Referenced by **listen()**.

#define Sn_CR_CONNECT 0x04

Send connection request in TCP mode(Client mode)

To connect, a connect-request (SYN packet) is sent to **TCP server**configured by **Sn_DIPR** & **Sn_DPORT(destination address & port)**. If the connect-request is successful, the **Sn_SR** is changed to **SOCK_ESTABLISHED** and the **Sn_IR(0)** becomes

The connect-request fails in the following three cases.

1. When a **ARPTO** occurs (**Sn_IR**[3] = '1') because destination hardware address is not acquired through the ARP-process.
2. When a **SYN/ACK** packet is not received and **TCPTO** (**Sn_IR(3)** ='1')
3. When a **RST** packet is received instead of a **SYN/ACK** packet. In these cases, **Sn_SR** is changed to **SOCK_CLOSED**.

> **Note**
>
> This is valid only in TCP mode and operates when Socket n acts as **TCP client**send connection request in tcp mode(Client mode)

Definition at line **753** of file **w5100.h**.

Referenced by **connect()**.

#define Sn_CR_DISCON 0x08

Send closing request in TCP mode.

Regardless of **TCP server**or **TCP client** the DISCON command processes the disconnect-process (**Active close**or **Passive close**.

Active close

it transmits disconnect-request(FIN packet) to the connected peer

Passive close
When FIN packet is received from peer, a FIN packet is replied back to the peer.

When the disconnect-process is successful (that is, FIN/ACK packet is received successfully), **Sn_SR** is changed to **SOCK_CLOSED**. Otherwise, TCPTO occurs (**Sn_IR(3)**='1') and then **Sn_SR** is changed to **SOCK_CLOSED**.

Note
Valid only in TCP mode.send closing reqeuset in tcp mode

Definition at line **766** of file **w5100.h**.

Referenced by **disconnect()**.

#define Sn_CR_CLOSE 0x10

Close socket.

Sn_SR is changed to **SOCK_CLOSED**.

Definition at line **772** of file **w5100.h**.

Referenced by **close()**.

#define Sn_CR_SEND 0x20

Update TX buffer pointer and send data.

SEND transmits all the data in the Socket n TX buffer.
For more details, please refer to Socket n TX Free Size Register (**Sn_TX_FSR**), Socket n, TX Write Pointer Register(**Sn_TX_WR**), and Socket n TX Read Pointer Register(**Sn_TX_RD**).

Definition at line **780** of file **w5100.h**.

Referenced by **send()**, and **sendto()**.

#define Sn_CR_SEND_MAC 0x21

Send data with MAC address, so without ARP process.

The basic operation is same as SEND.
Normally SEND transmits data after destination hardware address is acquired by the automatic ARP-process(Address Resolution Protocol).
But SEND_MAC transmits data without the automatic ARP-process. In this case, the destination hardware address is acquired from **Sn_DHAR** configured by host, instead of APR-process.

Note
Valid only in UDP mode.

Definition at line **790** of file **w5100.h**.

#define Sn_CR_SEND_KEEP 0x22

Send keep alive message.

It checks the connection status by sending 1byte keep-alive packet. If the peer can not respond to the keep-alive packet during timeout time, the connection is terminated and the timeout interrupt will occur.

Note
Valid only in TCP mode.

Definition at line **798** of file **w5100.h**.

Referenced by **setsockopt()**.

#define Sn_CR_RECV 0x40

Update RX buffer pointer and receive data.

RECV completes the processing of the received data in Socket n RX Buffer by using a RX read pointer register (**Sn_RX_RD**).
For more details, refer to Socket n RX Received Size Register (**Sn_RX_RSR**), Socket n RX Write Pointer Register (**Sn_RX_WR**), and Socket n RX Read Pointer Register (**Sn_RX_RD**).

Definition at line **806** of file **w5100.h**.

Referenced by **recv()**, and **recvfrom()**.

#define Sn_CR_PCON 0x23

PPPoE connection.

PPPoE connection begins by transmitting PPPoE discovery packet

Definition at line **812** of file **w5100.h**.

#define Sn_CR_PDISCON 0x24

Closes PPPoE connection.

Closes PPPoE connection

Definition at line **818** of file **w5100.h**.

#define Sn_CR_PCR 0x25

REQ message transmission.

In each phase, it transmits REQ message.

Definition at line **824** of file **w5100.h**.

#define Sn_CR_PCN 0x26

NAK massage transmission.

In each phase, it transmits NAK message.

Definition at line **830** of file **w5100.h**.

#define Sn_CR_PCJ 0x27

REJECT message transmission.

In each phase, it transmits REJECT message.

Definition at line **836** of file **w5100.h**.

#define Sn_IR_PRECV 0x80

PPP Receive Interrupt.

PPP Receive Interrupts when the option which is not supported is received.

Definition at line **843** of file **w5100.h**.

#define Sn_IR_PFAIL 0x40

PPP Fail Interrupt.

PPP Fail Interrupts when PAP Authentication is failed.

Definition at line **849** of file **w5100.h**.

#define Sn_IR_PNEXT 0x20

PPP Next Phase Interrupt.

PPP Next Phase Interrupts when the phase is changed during ADSL connection process.

Definition at line **855** of file **w5100.h**.

#define Sn_IR_SENDOK 0x10

SEND_OK Interrupt.

This is issued when SEND command is completed.complete sending

Definition at line **861** of file **w5100.h**.

Referenced by **send()**, and **sendto()**.

#define Sn_IR_TIMEOUT 0x08

TIMEOUT Interrupt.

This is issued when ARPTO or TCPTO occurs.assert timeout

Definition at line **867** of file **w5100.h**.

Referenced by **connect()**, **disconnect()**, **send()**, **sendto()**, and **setsockopt()**.

#define Sn_IR_RECV 0x04

RECV Interrupt.

This is issued whenever data is received from a peer.

Definition at line **873** of file **w5100.h**.

#define Sn_IR_DISCON 0x02

DISCON Interrupt.

This is issued when FIN or FIN/ACK packet is received from a peer.

Definition at line **879** of file **w5100.h**.

#define Sn_IR_CON 0x01

CON Interrupt.

This is issued one time when the connection with peer is successful and then **Sn_SR** is changed to **SOCK_ESTABLISHED**.

Definition at line **885** of file **w5100.h**.

#define SOCK_CLOSED 0x00

Closed.

This indicates that Socket n is released.
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to **SOCK_CLOSED** regardless of previous status.closed

Definition at line **893** of file **w5100.h**.

Referenced by **close()**, **connect()**, **disconnect()**, **listen()**, **recvfrom()**, **sendto()**, and **socket()**.

#define SOCK_INIT 0x13

Initiate state.

This indicates Socket n is opened with TCP mode.
It is changed to **SOCK_INIT** when **Sn_MR(P[3:0])** = 001)and OPEN command is ordered.
After **SOCK_INIT**, user can use LISTEN /CONNECT command.init state

Definition at line **901** of file **w5100.h**.

#define SOCK_LISTEN 0x14

Listen state.

This indicates Socket n is operating as **TCP server**mode and waiting for connection-request (SYN packet) from a peer (**TCP client**).
It will change to **SOCK_ESTABLISHED** when the connection-request is successfully accepted.
Otherwise it will change to **SOCK_CLOSED** after TCPTO occurred (**Sn_IR(TIMEOUT)** = '1').

Definition at line **909** of file **w5100.h**.

Referenced by **listen()**.

#define SOCK_SYNSENT 0x15

Connection state.

This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
It is temporarily shown when **Sn_SR** is changed from **SOCK_INIT** to **SOCK_ESTABLISHED** by CONNECT command.
If connect-accept(SYN/ACK packet) is received from the peer at SOCK_SYNSENT, it changes to **SOCK_ESTABLISHED**.
Otherwise, it changes to **SOCK_CLOSED** after TCPTO

(**Sn_IR**[TIMEOUT] = '1') is occurred.

Definition at line **918** of file **w5100.h**.

#define SOCK_SYNRECV 0x16

Connection state.

It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
If socket n sends the response (SYN/ACK packet) to the peer successfully, it changes to **SOCK_ESTABLISHED**.
If not, it changes to **SOCK_CLOSED** after timeout occurs (**Sn_IR**[TIMEOUT] = '1').

Definition at line **926** of file **w5100.h**.

#define SOCK_ESTABLISHED 0x17

Success to connect.

This indicates the status of the connection of Socket n.
It changes to **SOCK_ESTABLISHED** when the **TCP SERVER**processed the SYN packet from the **TCP CLIENT**during **SOCK_LISTEN**, or when the CONNECT command is successful.
During **SOCK_ESTABLISHED**, DATA packet can be transferred using SEND or RECV command.

Definition at line **935** of file **w5100.h**.

Referenced by **connect()**, **recv()**, and **send()**.

#define SOCK_FIN_WAIT 0x18

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **943** of file **w5100.h**.

#define SOCK_CLOSING 0x1A

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **951** of file **w5100.h**.

#define SOCK_TIME_WAIT 0x1B

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **959** of file **w5100.h**.

#define SOCK_CLOSE_WAIT 0x1C

Closing state.

This indicates Socket n received the disconnect-request (FIN packet) from the connected peer.
This is half-closing status, and data can be transferred.
For full-closing, DISCON command is used. But For just-closing, **Sn_CR_CLOSE** command is used.

Definition at line **967** of file **w5100.h**.

Referenced by **recv()**, and **send()**.

#define SOCK_LAST_ACK 0x1D

Closing state.

This indicates Socket n is waiting for the response (FIN/ACK packet) to the disconnect-request (FIN packet) by passive-close.
It changes to **SOCK_CLOSED** when Socket n received the response successfully, or when timeout occurs (**Sn_IR**[TIMEOUT] = '1').

Definition at line **974** of file **w5100.h**.

#define SOCK_UDP 0x22

UDP socket.

This indicates Socket n is opened in UDP mode(**Sn_MR(P[3:0])** = 010).
It changes to SOCK_UDP when **Sn_MR(P[3:0])** = 010 and **Sn_CR_OPEN** command is ordered.
Unlike TCP mode, data can be transfered without the connection-process.udp socket

Definition at line **982** of file **w5100.h**.

Referenced by **close()**, and **sendto()**.

#define SOCK_IPRAW 0x32

IP raw mode socket.

TThe socket is opened in IPRAW mode. The SOCKET status is change to SOCK_IPRAW when **Sn_MR** (P3:P0) is Sn_MR_IPRAW and **Sn_CR_OPEN** command is used.
IP Packet can be transferred without a connection similar to the UDP mode.ip raw mode socket

Definition at line **990** of file **w5100.h**.

#define SOCK_MACRAW 0x42

MAC raw mode socket.

This indicates Socket 0 is opened in MACRAW mode (**Sn_MR(P[3:0])** = '100' and n=0) and is valid only in Socket 0.
It changes to SOCK_MACRAW when **Sn_MR(P[3:0])** = '100' and **Sn_CR_OPEN** command is ordered.
Like UDP mode socket, MACRAW mode Socket 0 can transfer a MAC packet (Ethernet frame) without the connection-process.mac raw mode socket

Definition at line **998** of file **w5100.h**.

Referenced by **sendto()**.

#define SOCK_PPPOE 0x5F

PPPoE mode socket.

It is the status that SOCKET0 is open as PPPoE mode. It is changed to SOCK_PPPoE in case of S0_CR=OPEN and S0_MR (P3:P0)=S0_MR_PPPoE.
It is temporarily used at the PPPoE connection.pppoe socket

Definition at line **1007** of file **w5100.h**.

#define IPPROTO_IP 0

Dummy for IP.

Definition at line **1010** of file **w5100.h**.

#define IPPROTO_ICMP 1

Control message protocol.

Definition at line **1011** of file **w5100.h**.

#define IPPROTO_IGMP 2

Internet group management protocol.

Definition at line **1012** of file **w5100.h**.

#define IPPROTO_GGP 3

GW^2 (deprecated)

Definition at line **1013** of file **w5100.h**.

#define IPPROTO_TCP 6

TCP.

Definition at line **1014** of file **w5100.h**.

#define IPPROTO_PUP 12

PUP.

Definition at line **1015** of file **w5100.h**.

#define IPPROTO_UDP 17

UDP.

Definition at line **1016** of file **w5100.h**.

#define IPPROTO_IDP 22

XNS idp.

Definition at line **1017** of file **w5100.h**.

#define IPPROTO_ND 77

UNOFFICIAL net disk protocol.

Definition at line **1018** of file **w5100.h**.

#define IPPROTO_RAW 255

Raw IP packet.

Definition at line **1019** of file **w5100.h**.

#define WIZCHIP_CRITICAL_ENTER ( ) WIZCHIP.CRIS._enter()

Enter a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**,
WIZCHIP_READ_BUF(), **WIZCHIP_WRITE_BUF()**
WIZCHIP_CRITICAL_EXIT()

Definition at line **1032** of file **w5100.h**.

#define WIZCHIP_CRITICAL_EXIT ( ) WIZCHIP.CRIS._exit()

Exit a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**,
WIZCHIP_READ_BUF(), **WIZCHIP_WRITE_BUF()**
WIZCHIP_CRITICAL_ENTER()

Definition at line **1049** of file **w5100.h**.

#define
setSn_RXBUF_SIZE (sn,
rxmemsize

) setSn_RXMEM_SIZE(sn,rxmemsize)

Definition at line **1604** of file **w5100.h**.

Referenced by **wizchip_init()**.

#define getSn_RXBUF_SIZE (sn) getSn_RXMEM_SIZE(sn)

Definition at line **1615** of file **w5100.h**.

#define setSn_TXBUF_SIZE (sn, txmemsize) setSn_TXMEM_SIZE(sn,txmemsize)

Definition at line **1626** of file **w5100.h**.

Referenced by **wizchip_init()**.

#define getSn_TXBUF_SIZE (sn) getSn_TXMEM_SIZE(sn)

Definition at line **1637** of file **w5100.h**.

Socket APIs

W5200 Directory Reference

Files

file **w5200.c** [code]
W5200 HAL Interface.

file **w5200.h** [code]
W5200 HAL Header File.

Socket APIs

w5200.c File Reference

W5200 HAL Interface. More...

```
#include "w5200.h"
```

Go to the source code of this file.

Detailed Description

W5200 HAL Interface.

Version
1.0.0

Date
2013/10/21

Revision history
<2013/10/21> 1st Release

Author
MidnightCow

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

```
* Redistributions of source code must retain the a
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED

WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **w5200.c**.

Socket APIs

Macros | Functions

w5200.h File Reference

W5200 HAL Header File. More...

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

Go to the source code of this file.

Macros

#define **_WIZCHIP_SN_BASE_** (0x4000)

#define **_WIZCHIP_SN_SIZE_** (0x0100)

#define **_WIZCHIP_IO_TXBUF_** (0x8000) /* Internal Tx buffer addres
the iinchip */

#define **_WIZCHIP_IO_RXBUF_** (0xC000) /* Internal Rx buffer addre
the iinchip */

#define **_W5200_SPI_READ_** (0x00 << 7)
SPI interface Read operation in Control Phase. More...

#define **_W5200_SPI_WRITE_** (0x01 << 7)
SPI interface Write operation in Control Phase. More...

#define **WIZCHIP_CREG_BLOCK** 0x00
Common register block. More...

#define **WIZCHIP_SREG_BLOCK**(N) (**_WIZCHIP_SN_BASE_**+ **_WIZCHIP_SN_SIZE_***N)
Socket N register block. More...

#define **WIZCHIP_OFFSET_INC**(ADDR, N) (ADDR + N)
Increase offset address. More...

#define **IINCHIP_READ**(ADDR) **WIZCHIP_READ**(ADDR)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE**(ADDR, VAL) **WIZCHIP_WRITE**(ADDR,VAL)
The defined for legacy chip driver. More...

#define **IINCHIP_READ_BUF**(ADDR, BUF, LEN) **WIZCHIP_READ_BUF**(ADDR,BUF,LEN)

The defined for legacy chip driver. More...

#define **IINCHIP_WRITE_BUF**(ADDR, BUF, LEN) **WIZCHIP_WRITE**(ADDR,BUF,LEN)
The defined for legacy chip driver. More...

#define **MR** (_W5200_IO_BASE_ + (0x0000))
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and More...

#define **GAR** (_W5200_IO_BASE_ + (0x0001))
Gateway IP Register address(R/W) More...

#define **SUBR** (_W5200_IO_BASE_ + (0x0005))
Subnet mask Register address(R/W) More...

#define **SHAR** (_W5200_IO_BASE_ + (0x0009))
Source MAC Register address(R/W) More...

#define **SIPR** (_W5200_IO_BASE_ + (0x000F))
Source IP Register address(R/W) More...

#define **IR** (_W5200_IO_BASE_ + (0x0015))
Interrupt Register(R/W) More...

#define **_IMR_** (_W5200_IO_BASE_ + (0x0016))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (_W5200_IO_BASE_ + (0x0017))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (_W5200_IO_BASE_ + (0x0019))
Retry count register(R/W) More...

#define **PATR** (_W5200_IO_BASE_ + (0x001C))

PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PPPALGO** (_W5200_IO_BASE_ + (0x001E))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **VERSIONR** (_W5200_IO_BASE_ + (0x001F))
chip version register address(R) More...

#define **PTIMER** (_W5200_IO_BASE_ + (0x0028))
PPP LCP Request Timer register in PPPoE mode(R) More...

#define **PMAGIC** (_W5200_IO_BASE_ + (0x0029))
PPP LCP Magic number register in PPPoE mode(R) More...

#define **INTLEVEL** (_W5200_IO_BASE_ + (0x0030))
Set Interrupt low level timer register address(R/W) More...

#define **IR2** (_W5200_IO_BASE_ + (0x0034))
Socket Interrupt Register(R/W) More...

#define **PHYSTATUS** (_W5200_IO_BASE_ + (0x0035))
PHYSTATUS(R/W) More...

#define **IMR2** (_W5200_IO_BASE_ + (0x0036))
Interrupt mask register(R/W) More...

#define **Sn_MR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOC** + (0x0000))
socket Mode register(R/W) More...

#define **Sn_CR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOC** (0x0001))
Socket command register(R/W) More...

#define **Sn_IR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK** (0x0002))

Socket interrupt register(R) More...

#define **Sn_SR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCI**
(0x0003))
Socket status register(R) More...

#define **Sn_PORT**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x0004))
source port register(R/W) More...

#define **Sn_DHAR**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x0006))
Peer MAC register address(R/W) More...

#define **Sn_DIPR**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x000C))
Peer IP register address(R/W) More...

#define **Sn_DPORT**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x0010))
Peer port register address(R/W) More...

#define **Sn_MSSR**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x0012))
Maximum Segment Size(Sn_MSSR0) register address(R/W) M

#define **Sn_PROTO**(sn) (_W5200_IO_BASE_ +
WIZCHIP_SREG_BLOCK(sn) + (0x0014))
IP Protocol(PROTO) Register(R/W) More...

#define **Sn_TOS**(sn) (**WIZCHIP_SREG_BLOCK**(sn) + 0x0015)
IP Type of Service(TOS) Register(R/W) More...

#define **Sn_TTL**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOC**
+ (0x0016))
IP Time to live(TTL) Register(R/W) More...

#define **Sn_RXMEM_SIZE**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x001E))
Receive memory size register(R/W) More...

#define **Sn_TXMEM_SIZE**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x001F))
Transmit memory size register(R/W) More...

#define **Sn_TX_FSR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0020))
Transmit free memory size register(R) More...

#define **Sn_TX_RD**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0022))
Transmit memory read pointer register address(R) More...

#define **Sn_TX_WR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0024))
Transmit memory write pointer register address(R/W) More...

#define **Sn_RX_RSR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0026))
Received data size register(R) More...

#define **Sn_RX_RD**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x0028))
Read point of Receive memory(R/W) More...

#define **Sn_RX_WR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOCK**(sn) + (0x002A))
Write point of Receive memory(R) More...

#define **Sn_IMR**(sn) (_W5200_IO_BASE_ + **WIZCHIP_SREG_BLOC**
+ (0x002C))
socket interrupt mask register(R) More...

#define **Sn_FRAG**(sn) (_W5200_IO_BASE_ +

WIZCHIP_SREG_BLOCK(sn) + (0x002D))
Fragment field value in IP header register(R/W) More...

#define **MR_RST** 0x80
Reset. More...

#define **MR_WOL** 0x20
Wake on LAN. More...

#define **MR_PB** 0x10
Ping block. More...

#define **MR_PPPOE** 0x08
Enable PPPoE. More...

#define **MR_AI** 0x02
Address Auto-Increment in Indirect Bus Interface. More...

#define **MR_IND** 0x01
Indirect Bus Interface mode. More...

#define **IR_CONFLICT** 0x80
Check IP conflict. More...

#define **IR_PPPoE** 0x20
Get the PPPoE close message. More...

#define **PHYSTATUS_LINK** 0x20
Link Status [Read Only]. More...

#define **PHYSTATUS_POWERSAVE** 0x10
Power save mode of PHY. More...

#define **PHYSTATUS_POWERDOWN** 0x08
Power down mode of PHY. More...

#define **Sn_MR_CLOSE** 0x00
Unused socket. More...

#define **Sn_MR_TCP** 0x01
TCP. More...

#define **Sn_MR_UDP** 0x02
UDP. More...

#define **Sn_MR_IPRAW** 0x03
IP LAYER RAW SOCK. More...

#define **Sn_MR_MACRAW** 0x04
MAC LAYER RAW SOCK. More...

#define **Sn_MR_PPPOE** 0x05
PPPoE. More...

#define **Sn_MR_ND** 0x20
No Delayed Ack(TCP), Multicast flag. More...

#define **Sn_MR_MC** **Sn_MR_ND**
Support UDP Multicasting. More...

#define **Sn_MR_MF** 0x40
Multicast Blocking in **Sn_MR_MACRAW** mode. More...

#define **Sn_MR_MFEN** **Sn_MR_MF**

#define **Sn_MR_MULTI** 0x80
Support UDP Multicasting. More...

#define **Sn_CR_OPEN** 0x01
Initialize or open socket. More...

#define **Sn_CR_LISTEN** 0x02

Wait connection request in TCP mode(Server mode) More...

#define **Sn_CR_CONNECT** 0x04
Send connection request in TCP mode(Client mode) More...

#define **Sn_CR_DISCON** 0x08
Send closing request in TCP mode. More...

#define **Sn_CR_CLOSE** 0x10
Close socket. More...

#define **Sn_CR_SEND** 0x20
Update TX buffer pointer and send data. More...

#define **Sn_CR_SEND_MAC** 0x21
Send data with MAC address, so without ARP process. More..

#define **Sn_CR_SEND_KEEP** 0x22
Send keep alive message. More...

#define **Sn_CR_RECV** 0x40
Update RX buffer pointer and receive data. More...

#define **Sn_CR_PCON** 0x23
PPPoE connection. More...

#define **Sn_CR_PDISCON** 0x24
Closes PPPoE connection. More...

#define **Sn_CR_PCR** 0x25
REQ message transmission. More...

#define **Sn_CR_PCN** 0x26
NAK massage transmission. More...

#define **Sn_CR_PCJ** 0x27

REJECT message transmission. More...

#define **Sn_IR_PRECV** 0x80
PPP Receive Interrupt. More...

#define **Sn_IR_PFAIL** 0x40
PPP Fail Interrupt. More...

#define **Sn_IR_PNEXT** 0x20
PPP Next Phase Interrupt. More...

#define **Sn_IR_SENDOK** 0x10
SEND_OK Interrupt. More...

#define **Sn_IR_TIMEOUT** 0x08
TIMEOUT Interrupt. More...

#define **Sn_IR_RECV** 0x04
RECV Interrupt. More...

#define **Sn_IR_DISCON** 0x02
DISCON Interrupt. More...

#define **Sn_IR_CON** 0x01
CON Interrupt. More...

#define **SOCK_CLOSED** 0x00
Closed. More...

#define **SOCK_INIT** 0x13
Initiate state. More...

#define **SOCK_LISTEN** 0x14
Listen state. More...

#define **SOCK_SYNSENT** 0x15

Connection state. More...

#define **SOCK_SYNRECV** 0x16
Connection state. More...

#define **SOCK_ESTABLISHED** 0x17
Success to connect. More...

#define **SOCK_FIN_WAIT** 0x18
Closing state. More...

#define **SOCK_CLOSING** 0x1A
Closing state. More...

#define **SOCK_TIME_WAIT** 0x1B
Closing state. More...

#define **SOCK_CLOSE_WAIT** 0x1C
Closing state. More...

#define **SOCK_LAST_ACK** 0x1D
Closing state. More...

#define **SOCK_UDP** 0x22
UDP socket. More...

#define **SOCK_IPRAW** 0x32
IP raw mode socket. More...

#define **SOCK_MACRAW** 0x42
MAC raw mode socket. More...

#define **SOCK_PPPOE** 0x5F
PPPoE mode socket. More...

#define **IPPROTO_IP** 0

Dummy for IP. More...

#define **IPPROTO_ICMP** 1
Control message protocol. More...

#define **IPPROTO_IGMP** 2
Internet group management protocol. More...

#define **IPPROTO_GGP** 3
GW^2 (deprecated) More...

#define **IPPROTO_TCP** 6
TCP. More...

#define **IPPROTO_PUP** 12
PUP. More...

#define **IPPROTO_UDP** 17
UDP. More...

#define **IPPROTO_IDP** 22
XNS idp. More...

#define **IPPROTO_ND** 77
UNOFFICIAL net disk protocol. More...

#define **IPPROTO_RAW** 255
Raw IP packet. More...

#define **WIZCHIP_CRITICAL_ENTER**() WIZCHIP.CRIS._enter()
Enter a critical section. More...

#define **WIZCHIP_CRITICAL_EXIT**() WIZCHIP.CRIS._exit()
Exit a critical section. More...

#define **setMR**(mr) (*((uint8_t*)**MR**) = mr)

Set Mode Register. More...

#define **getMR**() (*(uint8_t*)**MR**)
Get **MR**. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set **GAR**. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get **GAR**. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**, subr,4)
Set **SUBR**. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get **SUBR**. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set **SHAR**. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get **SHAR**. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set **SIPR**. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get **SIPR**. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xA0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xA0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**IMR2**, imr & 0xA0)

Set **IMR2** register. More...

#define **getIMR**() (**WIZCHIP_READ**(**IMR2**) & 0xA0)
Get **IMR2** register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(**_RTR_**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **getPATR**() (((uint16_t)**WIZCHIP_READ**(**PATR**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PATR**,1)))
Get **PATR** register. More...

#define **getPPPALGO**() **WIZCHIP_READ**(**PPPALGO**)
Get **PPPALGO** register. More...

#define **getVERSIONR**() **WIZCHIP_READ**(**VERSIONR**)
Get **VERSIONR** register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)
Get **PMAGIC** register. More...

#define **setINTLEVEL**(intlevel)
Set **INTLEVEL** register. More...

#define **getINTLEVEL**() (((uint16_t)**WIZCHIP_READ**(**INTLEVEL**) << **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**INTLEVEL**,1)))
Get **INTLEVEL** register. More...

#define **setIR2**(ir2) **WIZCHIP_WRITE**(**IR2**, ir2)
Set **IR2** register. More...

#define **setSIR**(ir2) **setIR2**(ir2)

#define **getIR2**() **WIZCHIP_READ**(**IR2**)
Get **IR2** register. More...

#define **getSIR**() **getIR2**()

#define **getPHYSTATUS**() **WIZCHIP_READ**(**PHYSTATUS**)
Get **PHYSTATUS** register. More...

#define **setIMR2**(imr2) **WIZCHIP_WRITE**(**_IMR_**, imr2)
Set *IMR* register. More...

#define **setSIMR**(imr2) **setIMR2**(imr2)

#define **getIMR2**() **WIZCHIP_READ**(**_IMR_**)
Get *IMR* register. More...

#define **getSIMR**() **getIMR2**()

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ir)
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) **WIZCHIP_READ**(**Sn_IR**(sn))
Get **Sn_IR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), imr)
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) **WIZCHIP_READ**(**Sn_IMR**(sn))
Get **Sn_IMR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define **setSn_PORT**(sn, port)
Set **Sn_PORT** register. More...

#define **getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn)
Get **Sn_PORT** register. More...

#define **setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s
dhar, 6)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(s
dhar, 6)
Get **Sn_DHAR** register. More...

#define **setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn),
4)
Set **Sn_DIPR** register. More...

#define **getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn),
Get **Sn_DIPR** register. More...

#define **setSn_DPORT**(sn, dport)
Set **Sn_DPORT** register. More...

#define **getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1)
Get **Sn_DPORT** register. More...

#define **setSn_MSSR**(sn, mss)
Set **Sn_MSSR** register. More...

#define **getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn)
Get **Sn_MSSR** register. More...

#define **setSn_PROTO**(sn, proto) **WIZCHIP_WRITE**(**Sn_PROTO**(sn)
proto)
Set **Sn_PROTO** register. More...

#define **getSn_PROTO**(sn) **WIZCHIP_READ**(**Sn_PROTO**(sn))
Get **Sn_PROTO** register. More...

#define **setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos)
Set **Sn_TOS** register. More...

#define **getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn))

Get **Sn_TOS** register. More...

#define **setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl)
Set **Sn_TTL** register. More...

#define **getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn))
Get **Sn_TTL** register. More...

#define **setSn_RXMEM_SIZE**(sn, rxmemsize) **WIZCHIP_WRITE**(**Sn_RXMEM_SIZE**(sn),rxmem
Set **Sn_RXMEM_SIZE** register. More...

#define **setSn_RXBUF_SIZE**(sn, rxmemsize) **setSn_RXMEM_SIZE**(sn,rxmemsize)

#define **getSn_RXMEM_SIZE**(sn) **WIZCHIP_READ**(**Sn_RXMEM_SI**
Get **Sn_RXMEM_SIZE** register. More...

#define **getSn_RXBUF_SIZE**(sn) **getSn_RXMEM_SIZE**(sn)

#define **setSn_TXMEM_SIZE**(sn, txmemsize) **WIZCHIP_WRITE**(**Sn_TXMEM_SIZE**(sn), txmem
Set **Sn_TXMEM_SIZE** register. More...

#define **setSn_TXBUF_SIZE**(sn, txmemsize) **setSn_TXMEM_SIZE**(sn,txmemsize)

#define **getSn_TXMEM_SIZE**(sn) **WIZCHIP_READ**(**Sn_TXMEM_SIZ**
Get **Sn_TXMEM_SIZE** register. More...

#define **getSn_TXBUF_SIZE**(sn) **getSn_TXMEM_SIZE**(sn)

#define **getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sr
Get **Sn_TX_RD** register. More...

#define **setSn_TX_WR**(sn, txwr)

Set **Sn_TX_WR** register. More...

#define **getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1)
Get **Sn_TX_WR** register. More...

#define **setSn_RX_RD**(sn, rxrd)
Set **Sn_RX_RD** register. More...

#define **getSn_RX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1)
Get **Sn_RX_RD** register. More...

#define **setSn_RX_WR**(sn, rxwr)
Set **Sn_RX_WR** register. More...

#define **getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1)
Get **Sn_RX_WR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), imr)
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) **WIZCHIP_READ**(**Sn_IMR**(sn))
Get **Sn_IMR** register. More...

#define **setSn_FRAG**(sn, frag)
Set **Sn_FRAG** register. More...

#define **getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn)
Get **Sn_FRAG** register. More...

#define **getSn_RxMAX**(sn) ((uint16_t)**getSn_RXMEM_SIZE**(sn) << 1

Get the max RX buffer size of socket sn. More...

#define **getSn_TxMAX**(sn) ((uint16_t)**getSn_TXMEM_SIZE**(sn) << 1
Get the max TX buffer size of socket sn. More...

#define **getSn_RxMASK**(sn) ((uint16_t)**getSn_RxMAX**(sn) - 1)
Get the mask of socket sn RX buffer. More...

#define **getSn_TxMASK**(sn) ((uint16_t)**getSn_TxMAX**(sn) - 1)
Get the mask of socket sn TX buffer. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
uint16_t	**getSn_TX_FSR** (uint8_t sn) Get **Sn_TX_FSR** register. More...
uint16_t	**getSn_RX_RSR** (uint8_t sn) Get **Sn_RX_RSR** register. More...
uint16_t	**getSn_RxBASE** (uint8_t sn) Get the base address of socket sn RX buffer. More...
uint16_t	**getSn_TxBASE** (uint8_t sn) Get the base address of socket sn TX buffer. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...

void **wiz_recv_ignore** (uint8_t sn, uint16_t len)
It discard the received data in RX memory. More...

Detailed Description

W5200 HAL Header File.

Version
1.0.0

Date
2015/03/23

Revision history
<2013/10/21> 1st Release

Author
MidnightCow

Copyright

Definition in file **w5200.h**.

Macro Definition Documentation

#define _WIZCHIP_SN_BASE_ (0x4000)

Definition at line **50** of file **w5200.h**.

#define _WIZCHIP_SN_SIZE_ (0x0100)

Definition at line **51** of file **w5200.h**.

#define _WIZCHIP_IO_TXBUF_ (0x8000) /* Internal Tx buffer address of the iinchip */

Definition at line **52** of file **w5200.h**.

#define _WIZCHIP_IO_RXBUF_ (0xC000) /* Internal Rx buffer address of the iinchip */

Definition at line **53** of file **w5200.h**.

#define _W5200_SPI_READ_ (0x00 << 7)

SPI interface Read operation in Control Phase.

Definition at line **55** of file **w5200.h**.

#define _W5200_SPI_WRITE_ (0x01 << 7)

SPI interface Write operation in Control Phase.

Definition at line **56** of file **w5200.h**.

#define WIZCHIP_CREG_BLOCK 0x00

Common register block.

Definition at line **58** of file **w5200.h**.

#define WIZCHIP_SREG_BLOCK (N) (_WIZCHIP_SN_BASE_+ _WIZCHIP_SN_SIZE_*N)

Socket N register block.

Definition at line **59** of file **w5200.h**.

#define WIZCHIP_OFFSET_INC (ADDR, N) (ADDR + N)

Increase offset address.

Definition at line **61** of file **w5200.h**.

#define IINCHIP_READ (ADDR) WIZCHIP_READ(ADDR)

The defined for legacy chip driver.

Definition at line **75** of file **w5200.h**.

#define IINCHIP_WRITE (ADDR,

```
                                VAL
                            )     WIZCHIP_WRITE(ADDR,VAL)
```

The defined for legacy chip driver.

Definition at line **76** of file **w5200.h**.

```
#define
IINCHIP_READ_BUF    (  ADDR,
                       BUF,
                       LEN
                    )     WIZCHIP_READ_BUF(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **77** of file **w5200.h**.

```
#define
IINCHIP_WRITE_BUF        (  ADDR,
                            BUF,
                            LEN
                         )     WIZCHIP_WRITE(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **78** of file **w5200.h**.

```
#define MR_RST   0x80
```

Reset.

If this bit is All internal registers will be initialized. It will be automatically cleared as after S/W reset.reset

Definition at line **694** of file **w5200.h**.

#define MR_WOL 0x20

Wake on LAN.

0 : Disable WOL mode
1 : Enable WOL mode
If WOL mode is enabled and the received magic packet over UDP has been normally processed, the Interrupt PIN (INTn) asserts to low. When using WOL mode, the UDP Socket should be opened with any source port number. (Refer to Socket n Mode Register (**Sn_MR**) for opening Socket.)

Note

The magic packet over UDP supported by W5200 consists of 6 bytes synchronization stream (xFFFFFFFFFFFF and 16 times Target MAC address stream in UDP payload. The options such like password are ignored. You can use any UDP source port number for WOL mode.Wake on Lan

Definition at line **705** of file **w5200.h**.

#define MR_PB 0x10

Ping block.

0 : Disable Ping block
1 : Enable Ping block
If the bit is it blocks the response to a ping request.ping block

Definition at line **713** of file **w5200.h**.

#define MR_PPPOE 0x08

Enable PPPoE.

0 : DisablePPPoE mode
1 : EnablePPPoE mode
If you use ADSL, this bit should be '1'.enable pppoe

Definition at line **721** of file **w5200.h**.

#define MR_AI 0x02

Address Auto-Increment in Indirect Bus Interface.

0 : Disable auto-increment
1 : Enable auto-incremente
At the Indirect Bus Interface mode, if this bit is set as ��1��, the address will be automatically increased by 1 whenever read and write are performed.auto-increment in indirect mode

Definition at line **730** of file **w5200.h**.

#define MR_IND 0x01

Indirect Bus Interface mode.

0 : Disable Indirect bus Interface mode
1 : Enable Indirect bus Interface mode
If this bit is set as ��1��, Indirect Bus Interface mode is set.enable indirect mode

Definition at line **738** of file **w5200.h**.

#define IR_CONFLICT 0x80

Check IP conflict.

Bit is set as when own source IP address is same with the sender IP address in the received ARP request.check ip confict

Definition at line **745** of file **w5200.h**.

#define IR_PPPoE 0x20

Get the PPPoE close message.

When PPPoE is disconnected during PPPoE mode, this bit is set.get the PPPoE close message

Definition at line **751** of file **w5200.h**.

#define PHYSTATUS_LINK 0x20

Link Status [Read Only].

0: Link down
1: Link up

Definition at line **757** of file **w5200.h**.

Referenced by **wizphy_getphylink()**.

#define PHYSTATUS_POWERSAVE 0x10

Power save mode of PHY.

0: Disable Power save mode
1: Enable Power save mode

Definition at line **763** of file **w5200.h**.

#define PHYSTATUS_POWERDOWN 0x08

Power down mode of PHY.

0: Disable Power down mode
1: Enable Power down mode

Definition at line **769** of file **w5200.h**.

Referenced by **wizphy_getphypmode()**.

#define Sn_MR_CLOSE 0x00

Unused socket.

This configures the protocol mode of Socket n.unused socket

Definition at line **777** of file **w5200.h**.

#define Sn_MR_TCP 0x01

TCP.

This configures the protocol mode of Socket n.TCP

Definition at line **783** of file **w5200.h**.

#define Sn_MR_UDP 0x02

UDP.

This configures the protocol mode of Socket n.UDP

Definition at line **789** of file **w5200.h**.

#define Sn_MR_IPRAW 0x03

IP LAYER RAW SOCK.

Definition at line **790** of file **w5200.h**.

#define Sn_MR_MACRAW 0x04

MAC LAYER RAW SOCK.

This configures the protocol mode of Socket n.

Note
MACRAW mode should be only used in Socket 0.MAC LAYER RAW SOCK

Definition at line **797** of file **w5200.h**.

#define Sn_MR_PPPOE 0x05

PPPoE.

This configures the protocol mode of Socket n.

Note
PPPoE mode should be only used in Socket 0.PPPoE

Definition at line **804** of file **w5200.h**.

#define Sn_MR_ND 0x20

No Delayed Ack(TCP), Multicast flag.

0 : Disable No Delayed ACK option
1 : Enable No Delayed ACK option
This bit is applied only during TCP mode (P[3:0] = 001).
When this bit is It sends the ACK packet without delay as soon as a Data packet is received from a peer.
When this bit is It sends the ACK packet after waiting for the timeout time configured by *RTR*.No Delayed Ack(TCP) flag

Definition at line **814** of file **w5200.h**.

#define Sn_MR_MC Sn_MR_ND

Support UDP Multicasting.

0 : disable Multicasting
1 : enable Multicasting
This bit is applied only during UDP mode(P[3:0] = 010).
To use multicasting, **Sn_DIPR** & **Sn_DPORT** should be respectively configured with the multicast group IP address & port number before Socket n is opened by OPEN command of Sn_CR.Select IGMP version 1(0) or 2(1)

Definition at line **825** of file **w5200.h**.

#define Sn_MR_MF 0x40

Multicast Blocking in **Sn_MR_MACRAW** mode.

0 : using IGMP version 2
1 : using IGMP version 1
This bit is applied only during UDP mode(P[3:0] = 010 and MULTI = '1') It configures the version for IGMP messages (Join/Leave/Report).Use MAC filter

Definition at line **834** of file **w5200.h**.

#define Sn_MR_MFEN Sn_MR_MF

Definition at line **835** of file **w5200.h**.

#define Sn_MR_MULTI 0x80

Support UDP Multicasting.

0 : disable Multicasting
1 : enable Multicasting
This bit is applied only during UDP mode(P[3:0] = 010).
To use multicasting, **Sn_DIPR** & **Sn_DPORT** should be respectively configured with the multicast group IP address & port number before Socket n is opened by OPEN command of Sn_CR.support multicating

Definition at line **846** of file **w5200.h**.

#define Sn_CR_OPEN 0x01

Initialize or open socket.

Socket n is initialized and opened according to the protocol selected in **Sn_MR(P3:P0)**. The table below shows the value of **Sn_SR** corresponding to **Sn_MR**.

Sn_MR (P[3:0])	**Sn_SR**
Sn_MR_CLOSE (000)	–
Sn_MR_TCP (001)	SOCK_INIT (0x13)
Sn_MR_UDP (010)	SOCK_UDP (0x22)
S0_MR_IPRAW (011)	SOCK_IPRAW (0x32)
S0_MR_MACRAW (100)	SOCK_MACRAW (0x42)
S0_MR_PPPoE (101)	SOCK_PPPoE (0x5F)

initialize or open socket

Definition at line **863** of file **w5200.h**.

#define Sn_CR_LISTEN 0x02

Wait connection request in TCP mode(Server mode)

This is valid only in TCP mode (**Sn_MR(P3:P0)** = **Sn_MR_TCP**).// In this mode, Socket n operates as a 'TCP server' and waits for

connection-request (SYN packet) from any 'TCP client'.// The **Sn_SR** changes the state from SOCK_INIT to SOCKET_LISTEN.// When a 'TCP client' connection request is successfully established, the **Sn_SR** changes from SOCK_LISTEN to SOCK_ESTABLISHED and the **Sn_IR(0)** becomes But when a 'TCP client' connection request is failed, **Sn_IR(3)** becomes and the status of **Sn_SR** changes to SOCK_CLOSED.wait connection request in tcp mode(Server mode)

Definition at line **874** of file **w5200.h**.

#define Sn_CR_CONNECT 0x04

Send connection request in TCP mode(Client mode)

To connect, a connect-request (SYN packet) is sent to **TCP server**configured by **Sn_DIPR** & **Sn_DPORT(destination address & port)**. If the connect-request is successful, the **Sn_SR** is changed to **SOCK_ESTABLISHED** and the **Sn_IR(0)** becomes

The connect-request fails in the following three cases.

1. When a **ARPTO** occurs (**Sn_IR**[3] = '1') because destination hardware address is not acquired through the ARP-process.
2. When a **SYN/ACK** packet is not received and **TCPTO** (**Sn_IR(3)** ='1')
3. When a **RST** packet is received instead of a **SYN/ACK** packet. In these cases, **Sn_SR** is changed to **SOCK_CLOSED**.

> **Note**
>
> This is valid only in TCP mode and operates when Socket n acts as **TCP client**send connection request in tcp mode(Client mode)

Definition at line **886** of file **w5200.h**.

#define Sn_CR_DISCON 0x08

Send closing request in TCP mode.

Regardless of **TCP server**or **TCP client** the DISCON command processes the disconnect-process (**Active close**or **Passive close**.

Active close
: it transmits disconnect-request(FIN packet) to the connected peer

Passive close
: When FIN packet is received from peer, a FIN packet is replied back to the peer.

When the disconnect-process is successful (that is, FIN/ACK packet is received successfully), **Sn_SR** is changed to **SOCK_CLOSED**. Otherwise, TCPTO occurs (**Sn_IR(3)**='1') and then **Sn_SR** is changed to **SOCK_CLOSED**.

Note
: Valid only in TCP mode.send closing reqeuset in tcp mode

Definition at line **899** of file **w5200.h**.

#define Sn_CR_CLOSE 0x10

Close socket.

Sn_SR is changed to **SOCK_CLOSED**.

Definition at line **905** of file **w5200.h**.

#define Sn_CR_SEND 0x20

Update TX buffer pointer and send data.

SEND transmits all the data in the Socket n TX buffer.
For more details, please refer to Socket n TX Free Size Register (**Sn_TX_FSR**), Socket n, TX Write Pointer Register(**Sn_TX_WR**), and Socket n TX Read Pointer Register(**Sn_TX_RD**).

Definition at line **913** of file **w5200.h**.

#define Sn_CR_SEND_MAC 0x21

Send data with MAC address, so without ARP process.

The basic operation is same as SEND.
Normally SEND transmits data after destination hardware address is acquired by the automatic ARP-process(Address Resolution Protocol).
But SEND_MAC transmits data without the automatic ARP-process.
In this case, the destination hardware address is acquired from **Sn_DHAR** configured by host, instead of APR-process.

Note
Valid only in UDP mode.

Definition at line **923** of file **w5200.h**.

#define Sn_CR_SEND_KEEP 0x22

Send keep alive message.

It checks the connection status by sending 1byte keep-alive packet.
If the peer can not respond to the keep-alive packet during timeout time, the connection is terminated and the timeout interrupt will occur.

Note
Valid only in TCP mode.

Definition at line **931** of file **w5200.h**.

#define Sn_CR_RECV 0x40

Update RX buffer pointer and receive data.

RECV completes the processing of the received data in Socket n RX Buffer by using a RX read pointer register (**Sn_RX_RD**).
For more details, refer to Socket n RX Received Size Register (**Sn_RX_RSR**), Socket n RX Write Pointer Register (**Sn_RX_WR**), and Socket n RX Read Pointer Register (**Sn_RX_RD**).

Definition at line **939** of file **w5200.h**.

#define Sn_CR_PCON 0x23

PPPoE connection.

PPPoE connection begins by transmitting PPPoE discovery packet

Definition at line **945** of file **w5200.h**.

#define Sn_CR_PDISCON 0x24

Closes PPPoE connection.

Closes PPPoE connection

Definition at line **951** of file **w5200.h**.

#define Sn_CR_PCR 0x25

REQ message transmission.

In each phase, it transmits REQ message.

Definition at line **957** of file **w5200.h**.

#define Sn_CR_PCN 0x26

NAK massage transmission.

In each phase, it transmits NAK message.

Definition at line **963** of file **w5200.h**.

#define Sn_CR_PCJ 0x27

REJECT message transmission.

In each phase, it transmits REJECT message.

Definition at line **969** of file **w5200.h**.

#define Sn_IR_PRECV 0x80

PPP Receive Interrupt.

PPP Receive Interrupts when the option which is not supported is received.

Definition at line **976** of file **w5200.h**.

#define Sn_IR_PFAIL 0x40

PPP Fail Interrupt.

PPP Fail Interrupts when PAP Authentication is failed.

Definition at line **982** of file **w5200.h**.

#define Sn_IR_PNEXT 0x20

PPP Next Phase Interrupt.

PPP Next Phase Interrupts when the phase is changed during ADSL connection process.

Definition at line **988** of file **w5200.h**.

#define Sn_IR_SENDOK 0x10

SEND_OK Interrupt.

This is issued when SEND command is completed.complete sending

Definition at line **994** of file **w5200.h**.

#define Sn_IR_TIMEOUT 0x08

TIMEOUT Interrupt.

This is issued when ARPTO or TCPTO occurs.assert timeout

Definition at line **1000** of file **w5200.h**.

#define Sn_IR_RECV 0x04

RECV Interrupt.

This is issued whenever data is received from a peer.

Definition at line **1006** of file **w5200.h**.

#define Sn_IR_DISCON 0x02

DISCON Interrupt.

This is issued when FIN or FIN/ACK packet is received from a peer.

Definition at line **1012** of file **w5200.h**.

#define Sn_IR_CON 0x01

CON Interrupt.

This is issued one time when the connection with peer is successful and then **Sn_SR** is changed to **SOCK_ESTABLISHED**.

Definition at line **1018** of file **w5200.h**.

#define SOCK_CLOSED 0x00

Closed.

This indicates that Socket n is released.
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to **SOCK_CLOSED** regardless of previous status.closed

Definition at line **1026** of file **w5200.h**.

#define SOCK_INIT 0x13

Initiate state.

This indicates Socket n is opened with TCP mode.
It is changed to **SOCK_INIT** when **Sn_MR(P[3:0])** = 001)and OPEN command is ordered.
After **SOCK_INIT**, user can use LISTEN /CONNECT command.init state

Definition at line **1034** of file **w5200.h**.

#define SOCK_LISTEN 0x14

Listen state.

This indicates Socket n is operating as **TCP server**mode and waiting for connection-request (SYN packet) from a peer (**TCP client**).
It will change to **SOCK_ESTABLISHED** when the connection-request is successfully accepted.
Otherwise it will change to **SOCK_CLOSED** after TCPTO occurred (**Sn_IR(TIMEOUT)** = '1').

Definition at line **1042** of file **w5200.h**.

#define SOCK_SYNSENT 0x15

Connection state.

This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
It is temporarily shown when **Sn_SR** is changed from **SOCK_INIT** to **SOCK_ESTABLISHED** by CONNECT command.
If connect-accept(SYN/ACK packet) is received from the peer at SOCK_SYNSENT, it changes to **SOCK_ESTABLISHED**.
Otherwise, it changes to **SOCK_CLOSED** after TCPTO (**Sn_IR**[TIMEOUT] = '1') is occurred.

Definition at line **1051** of file **w5200.h**.

#define SOCK_SYNRECV 0x16

Connection state.

It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
If socket n sends the response (SYN/ACK packet) to the peer successfully, it changes to **SOCK_ESTABLISHED**.
If not, it changes to **SOCK_CLOSED** after timeout occurs (**Sn_IR**[TIMEOUT] = '1').

Definition at line **1059** of file **w5200.h**.

#define SOCK_ESTABLISHED 0x17

Success to connect.

This indicates the status of the connection of Socket n.
It changes to **SOCK_ESTABLISHED** when the **TCP SERVER**processed the SYN packet from the **TCP CLIENT**during **SOCK_LISTEN**, or when the CONNECT command is successful.
During **SOCK_ESTABLISHED**, DATA packet can be transferred using SEND or RECV command.

Definition at line **1068** of file **w5200.h**.

#define SOCK_FIN_WAIT 0x18

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1076** of file **w5200.h**.

#define SOCK_CLOSING 0x1A

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1084** of file **w5200.h**.

#define SOCK_TIME_WAIT 0x1B

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1092** of file **w5200.h**.

#define SOCK_CLOSE_WAIT 0x1C

Closing state.

This indicates Socket n received the disconnect-request (FIN packet) from the connected peer.
This is half-closing status, and data can be transferred.
For full-closing, DISCON command is used. But For just-closing, CLOSE command is used.

Definition at line **1100** of file **w5200.h**.

#define SOCK_LAST_ACK 0x1D

Closing state.

This indicates Socket n is waiting for the response (FIN/ACK packet) to the disconnect-request (FIN packet) by passive-close.
It changes to **SOCK_CLOSED** when Socket n received the response successfully, or when timeout occurs (**Sn_IR**[TIMEOUT] = '1').

Definition at line **1107** of file **w5200.h**.

#define SOCK_UDP 0x22

UDP socket.

This indicates Socket n is opened in UDP mode(**Sn_MR(P[3:0])** = 010).
It changes to SOCK_UDP when **Sn_MR(P[3:0])** = 010 and OPEN command is ordered.
Unlike TCP mode, data can be transfered without the connection-process.udp socket

Definition at line **1115** of file **w5200.h**.

#define SOCK_IPRAW 0x32

IP raw mode socket.

TThe socket is opened in IPRAW mode. The SOCKET status is change to SOCK_IPRAW when Sn_MR (P3:P0) is Sn_MR_IPRAW and OPEN command is used.
IP Packet can be transferred without a connection similar to the UDP mode.ip raw mode socket

Definition at line **1123** of file **w5200.h**.

#define SOCK_MACRAW 0x42

MAC raw mode socket.

This indicates Socket 0 is opened in MACRAW mode (S0_MR(P[3:0]) = 100and is valid only in Socket 0.
It changes to SOCK_MACRAW when S0_MR(P[3:0] = 100)and OPEN command is ordered.
Like UDP mode socket, MACRAW mode Socket 0 can transfer a MAC packet (Ethernet frame) without the connection-process.mac raw mode socket

Definition at line **1131** of file **w5200.h**.

#define SOCK_PPPOE 0x5F

PPPoE mode socket.

It is the status that SOCKET0 is open as PPPoE mode. It is changed to SOCK_PPPoE in case of S0_CR=OPEN and S0_MR (P3:P0)=S0_MR_PPPoE.
It is temporarily used at the PPPoE connection.pppoe socket

Definition at line **1140** of file **w5200.h**.

#define IPPROTO_IP 0

Dummy for IP.

Definition at line **1143** of file **w5200.h**.

#define IPPROTO_ICMP 1

Control message protocol.

Definition at line **1144** of file **w5200.h**.

#define IPPROTO_IGMP 2

Internet group management protocol.

Definition at line **1145** of file **w5200.h**.

#define IPPROTO_GGP 3

GW^2 (deprecated)

Definition at line **1146** of file **w5200.h**.

#define IPPROTO_TCP 6

TCP.

Definition at line **1147** of file **w5200.h**.

#define IPPROTO_PUP 12

PUP.

Definition at line **1148** of file **w5200.h**.

#define IPPROTO_UDP 17

UDP.

Definition at line **1149** of file **w5200.h**.

#define IPPROTO_IDP 22

XNS idp.

Definition at line **1150** of file **w5200.h**.

#define IPPROTO_ND 77

UNOFFICIAL net disk protocol.

Definition at line **1151** of file **w5200.h**.

#define IPPROTO_RAW 255

Raw IP packet.

Definition at line **1152** of file **w5200.h**.

#define WIZCHIP_CRITICAL_ENTER ( ) WIZCHIP.CRIS._enter()

Enter a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**,
WIZCHIP_READ_BUF(), **WIZCHIP_WRITE_BUF()**

WIZCHIP_CRITICAL_EXIT()

Definition at line **1165** of file **w5200.h**.

#define WIZCHIP_CRITICAL_EXIT ( ) WIZCHIP.CRIS._exit()

Exit a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.

In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**, **WIZCHIP_READ_BUF()**, **WIZCHIP_WRITE_BUF()**

WIZCHIP_CRITICAL_ENTER()

Definition at line **1182** of file **w5200.h**.

#define setSIR (ir2) setIR2(ir2)

Definition at line **1499** of file **w5200.h**.

Referenced by **wizchip_clrinterrupt()**.

#define getSIR ( ) getIR2()

Definition at line **1509** of file **w5200.h**.

Referenced by **wizchip_getinterrupt()**.

#define setSIMR (imr2) setIMR2(imr2)

Definition at line **1533** of file **w5200.h**.

Referenced by **wizchip_setinterruptmask()**.

#define getSIMR ( ) getIMR2()

Definition at line **1548** of file **w5200.h**.

Referenced by **wizchip_getinterruptmask()**.

```
#define
setSn_RXBUF_SIZE        (  sn,
                           rxmemsize
                        )     setSn_RXMEM_SIZE(sn,rxmemsize)
```

Definition at line **1827** of file **w5200.h**.

```
#define getSn_RXBUF_SIZE (  sn )    getSn_RXMEM_SIZE(sn)
```

Definition at line **1839** of file **w5200.h**.

```
#define
setSn_TXBUF_SIZE        (  sn,
                           txmemsize
                        )     setSn_TXMEM_SIZE(sn,txmemsize)
```

Definition at line **1851** of file **w5200.h**.

```
#define getSn_TXBUF_SIZE (  sn )    getSn_TXMEM_SIZE(sn)
```

Definition at line **1863** of file **w5200.h**.

Socket APIs

W5300 Directory Reference

Files

file **w5300.c** [code]

file **w5300.h** [code]
W5300 HAL implement File.

Socket APIs

w5300.c File Reference

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

Go to the source code of this file.

Socket APIs

Macros | Functions

w5300.h File Reference

W5300 HAL implement File. More...

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

Go to the source code of this file.

Macros

#define **_WIZCHIP_SN_BASE_** (0x0200)

#define **_WIZCHIP_SN_SIZE_** (0x0040)

#define **WIZCHIP_CREG_BLOCK** 0x00
Common register block. More...

#define **WIZCHIP_SREG_BLOCK**(N) (**_WIZCHIP_SN_BASE_** + **_WIZCHIP_SN_SIZE_***N)
Socket N register block. More...

#define **WIZCHIP_OFFSET_INC**(ADDR, N) (ADDR + N)
Increase offset address. More...

#define **_W5300_IO_BASE_** **_WIZCHIP_IO_BASE_**

#define **IINCHIP_READ**(ADDR) **WIZCHIP_READ**(ADDR)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE**(ADDR, VAL) **WIZCHIP_WRITE**(ADDR,VAL)
The defined for legacy chip driver. More...

#define **MR** (**_WIZCHIP_IO_BASE_**)
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and

#define **IR** (**_W5300_IO_BASE_** + 0x02)
Interrupt Register(R/W) More...

#define **_IMR_** (**_W5300_IO_BASE_** + 0x04)
Socket Interrupt Mask Register(R/W) More...

#define **SHAR** (**_W5300_IO_BASE_** + 0x08)
Source MAC Register address(R/W) More...

#define **GAR** (**_W5300_IO_BASE_** + 0x10)
Gateway IP Register address(R/W) More...

#define **SUBR** (**_W5300_IO_BASE_** + 0x14)
Subnet mask Register address(R/W) More...

#define **SIPR** (**_W5300_IO_BASE_** + 0x18)
Source IP Register address(R/W) More...

#define **_RTR_** (**_W5300_IO_BASE_** + 0x1C)
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5300_IO_BASE_** + 0x1E)
Retry count register(R/W) More...

#define **TMS01R** (**_W5300_IO_BASE_** + 0x20)
TX memory size of SOCKET 0 & 1. More...

#define **TMS23R** (**TMS01R** + 2)
TX memory size of SOCKET 2 & 3. More...

#define **TMS45R** (**TMS01R** + 4)
TX memory size of SOCKET 4 & 5. More...

#define **TMS67R** (**TMS01R** + 6)
TX memory size of SOCKET 6 & 7. More...

#define **TMSR0** **TMS01R**
TX memory size of SOCKET 0. More...

#define **TMSR1** (**TMSR0** + 1)
TX memory size of SOCKET 1. More...

#define **TMSR2** (**TMSR0** + 2)

TX memory size of SOCKET 2. More...

#define **TMSR3** (**TMSR0** + 3)
TX memory size of SOCKET 3. More...

#define **TMSR4** (**TMSR0** + 4)
TX memory size of SOCKET 4. More...

#define **TMSR5** (**TMSR0** + 5)
TX memory size of SOCKET 5. More...

#define **TMSR6** (**TMSR0** + 6)
TX memory size of SOCKET 6. More...

#define **TMSR7** (**TMSR0** + 7)
TX memory size of SOCKET 7. More...

#define **RMS01R** (**_W5300_IO_BASE_** + 0x28)
RX memory size of SOCKET 0 & 1. More...

#define **RMS23R** (**RMS01R** + 2)
RX memory size of SOCKET 2 & 3. More...

#define **RMS45R** (**RMS01R** + 4)
RX memory size of SOCKET 4 & 5. More...

#define **RMS67R** (**RMS01R** + 6)
RX memory size of SOCKET 6 & 7. More...

#define **RMSR0** **RMS01R**
RX memory size of SOCKET 0. More...

#define **RMSR1** (**RMSR0** + 1)
RX memory size of SOCKET 1. More...

#define **RMSR2** (**RMSR0** + 2)
RX memory size of SOCKET 2. More...

#define **RMSR3** (**RMSR0** + 3)
RX memory size of SOCKET 3. More...

#define **RMSR4** (**RMSR0** + 4)
RX memory size of SOCKET 4. More...

#define **RMSR5** (**RMSR0** + 5)
RX memory size of SOCKET 5. More...

#define **RMSR6** (**RMSR0** + 6)
RX memory size of SOCKET 6. More...

#define **RMSR7** (**RMSR0** + 7)
RX memory size of SOCKET 7. More...

#define **MTYPER** (**_W5300_IO_BASE_** + 0x30)
Memory Type Register. More...

#define **PATR** (**_W5300_IO_BASE_** + 0x32)
PPPoE Authentication Type register. More...

#define **PTIMER** (**_W5300_IO_BASE_** + 0x36)
PPP Link Control Protocol Request Timer Register. More...

#define **PMAGICR** (**_W5300_IO_BASE_** + 0x38)
PPP LCP magic number register. More...

#define **PSIDR** (**_W5300_IO_BASE_** + 0x3C)
PPPoE session ID register. More...

#define **PDHAR** (**_W5300_IO_BASE_** + 0x40)
PPPoE destination hardware address register. More...

#define **UIPR** (**_W5300_IO_BASE_** + 0x48)
Unreachable IP address register. More...

#define **UPORTR** (**_W5300_IO_BASE_** + 0x4C)
Unreachable port number register. More...

#define **FMTUR** (**_W5300_IO_BASE_** + 0x4E)
Fragment MTU register. More...

#define **Pn_BRDYR**(n) (**_W5300_IO_BASE_** + 0x60 + n*4)
PIN 'BRDYn' configure register. More...

#define **Pn_BDPTHR**(n) (**_W5300_IO_BASE_** + 0x60 + n*4 + 2)
PIN 'BRDYn' buffer depth Register. More...

#define **IDR** (**_W5300_IO_BASE_** + 0xFE)
W5300 identification register. More...

#define **VERSIONR** **IDR**

#define **Sn_MR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**
Socket Mode register(R/W) More...

#define **Sn_CR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**
Socket command register(R/W) More...

#define **Sn_IMR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOC**
socket interrupt mask register(R) More...

#define **Sn_IR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOCK**(
Socket interrupt register(R) More...

#define **Sn_SSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLOC**
Socket status register(R) More...

#define **Sn_SR**(n) **Sn_SSR**(n)
For Compatible ioLibrary. Refer to **Sn_SSR(n)** More...

#define **Sn_PORTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BL**
0x0A)
source port register(R/W) More...

#define **Sn_PORT**(n) **Sn_PORTR**(n)
For compatible ioLibrary. Refer to **Sn_PORTR(n)**. More...

#define **Sn_DHAR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLO**
Peer MAC register address(R/W) More...

#define **Sn_DPORTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_B**
0x12)
Peer port register address(R/W) More...

#define **Sn_DPORT**(n) **Sn_DPORTR**(n)
For compatible ioLibrary. Refer to **Sn_DPORTR**. More...

#define **Sn_DIPR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLO**
Peer IP register address(R/W) More...

#define **Sn_MSSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLO**
Maximum Segment Size(Sn_MSSR0) register address(R/W) M

#define **Sn_KPALVTR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_**
0x1A)
Keep Alive Timer register(R/W) More...

#define **Sn_PROTOR**(n) **Sn_KPALVTR**(n)
IP Protocol(PROTO) Register(R/W) More...

#define **Sn_TOSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLO**
IP Type of Service(TOS) Register(R/W) More...

#define **Sn_TOS**(n) **Sn_TOSR**(n)
For compatible ioLibrary. Refer to Sn_TOSR. More...

#define **Sn_TTLR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BLO**
IP Time to live(TTL) Register(R/W) More...

#define **Sn_TTL**(n) **Sn_TTLR**(n)
For compatible ioLibrary. Refer to Sn_TTLR. More...

#define **Sn_TX_WRSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_** 0x20)
SOCKETn TX write size register(R/W) More...

#define **Sn_TX_FSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BI** 0x0024)
Transmit free memory size register(R) More...

#define **Sn_RX_RSR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_B** 0x0028)
Received data size register(R) More...

#define **Sn_FRAGR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_BL** 0x002C)
Fragment field value in IP header register(R/W) More...

#define **Sn_FRAG**(n) **Sn_FRAGR**(n)

#define **Sn_TX_FIFOR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_** 0x2E)
SOCKET n TX FIFO regsiter. More...

#define **Sn_RX_FIFOR**(n) (**_W5300_IO_BASE_** + **WIZCHIP_SREG_** 0x30)
SOCKET n RX FIFO register. More...

#define **MR_DBW** (1 << 15)

```
#define MR_MPF   (1 << 14)
#define MR_WDF(X)   ((X & 0x07) << 11)
#define MR_RDH   (1 << 10)
#define MR_FS   (1 << 8)
#define MR_RST   (1 << 7)
#define MR_MT   (1 << 5)
#define MR_PB   (1 << 4)
#define MR_PPPoE   (1 << 3)
#define MR_DBS   (1 << 2)
#define MR_IND   (1 << 0)
#define IR_IPCF   (1 << 15)
#define IR_DPUR   (1 << 14)
#define IR_PPPT   (1 << 13)
#define IR_FMTU   (1 << 12)
#define IR_SnINT(n)   (0x01 << n)
#define Pn_PEN   (1 << 7)
#define Pn_MT   (1 << 6)
#define Pn_PPL   (1 << 5)
#define Pn_SN(n)   ((n & 0x07) << 0)
```

#define **Sn_MR_ALIGN** (1 << 8)
Alignment bit of **Sn_MR**. More...

#define **Sn_MR_MULTI** (1 << 7)
Multicasting bit of **Sn_MR**. More...

#define **Sn_MR_MF** (1 << 6)
MAC filter bit of **Sn_MR**. More...

#define **Sn_MR_IGMPv** (1 << 5)
IGMP version bit of **Sn_MR** details It is valid in case of **Sn_MR**
and UDP(**Sn_MR_UDP**). It configures IGMP version to send IG
such as **Join/Leave/Report** to multicast-group.
0 : IGMPv2, 1 : IGMPv1. More...

#define **Sn_MR_MC** **Sn_MR_IGMPv**
For compatible ioLibrary. More...

#define **Sn_MR_ND** (1 << 5)
No delayed ack bit of **Sn_MR**. More...

#define **Sn_MR_CLOSE** 0x00
No mode. More...

#define **Sn_MR_TCP** 0x01
TCP mode. More...

#define **Sn_MR_UDP** 0x02
UDP mode. More...

#define **Sn_MR_IPRAW** 0x03
IP LAYER RAW mode. More...

#define **Sn_MR_MACRAW** 0x04
MAC LAYER RAW mode. More...

#define **Sn_MR_PPPoE** 0x05
PPPoE mode. More...

#define **SOCK_STREAM** **Sn_MR_TCP**

#define **SOCK_DGRAM** **Sn_MR_UDP**

#define **Sn_CR_OPEN** 0x01
Initialize or open a socket. More...

#define **Sn_CR_LISTEN** 0x02
Wait connection request in TCP mode(Server mode) More...

#define **Sn_CR_CONNECT** 0x04
Send connection request in TCP mode(Client mode) More...

#define **Sn_CR_DISCON** 0x08
Send closing request in TCP mode. More...

#define **Sn_CR_CLOSE** 0x10
Close socket. More...

#define **Sn_CR_SEND** 0x20
Update TX buffer pointer and send data. More...

#define **Sn_CR_SEND_MAC** 0x21
Send data with MAC address, so without ARP process. More..

#define **Sn_CR_SEND_KEEP** 0x22
Send keep alive message. More...

#define **Sn_CR_RECV** 0x40
Update RX buffer pointer and receive data. More...

#define **Sn_CR_PCON** 0x23

#define **Sn_CR_PDISCON** 0x24

#define **Sn_CR_PCR** 0x25

#define **Sn_CR_PCN** 0x26

#define **Sn_CR_PCJ** 0x27

#define **Sn_IR_PRECV** 0x80

#define **Sn_IR_PFAIL** 0x40

#define **Sn_IR_PNEXT** 0x20

#define **Sn_IR_SENDOK** 0x10

#define **Sn_IR_TIMEOUT** 0x08

#define **Sn_IR_RECV** 0x04

#define **Sn_IR_DISCON** 0x02

#define **Sn_IR_CON** 0x01

#define **SOCK_CLOSED** 0x00
The state of SOCKET intialized or closed. More...

#define **SOCK_ARP** 0x01
The state of ARP process. More...

#define **SOCK_INIT** 0x13
Initiate state in TCP. More...

#define **SOCK_LISTEN** 0x14
Listen state. More...

#define **SOCK_SYNSENT** 0x15
Connection state. More...

#define **SOCK_SYNRECV** 0x16
Connection state. More...

#define **SOCK_ESTABLISHED** 0x17
Success to connect. More...

#define **SOCK_FIN_WAIT** 0x18
Closing state. More...

#define **SOCK_CLOSING** 0x1A
Closing state. More...

#define **SOCK_TIME_WAIT** 0x1B
Closing state. More...

#define **SOCK_CLOSE_WAIT** 0x1C
Closing state. More...

#define **SOCK_LAST_ACK** 0x1D
Closing state. More...

#define **SOCK_UDP** 0x22
UDP socket. More...

#define **SOCK_IPRAW** 0x32
IP raw mode socket. More...

#define **SOCK_MACRAW** 0x42
MAC raw mode socket. More...

#define **SOCK_PPPoE** 0x5F
PPPoE mode socket. More...

#define **IPPROTO_IP** 0

#define **IPPROTO_ICMP** 1

#define **IPPROTO_IGMP** 2

#define **IPPROTO_GGP** 3

#define **IPPROTO_TCP** 6

#define **IPPROTO_PUP** 12

#define **IPPROTO_UDP** 17

#define **IPPROTO_IDP** 22

#define **IPPROTO_ND** 77

#define **IPPROTO_RAW** 255

#define **WIZCHIP_CRITICAL_ENTER**() WIZCHIP.CRIS._enter()
Enter a critical section. More...

#define **WIZCHIP_CRITICAL_EXIT**() WIZCHIP.CRIS._exit()
Exit a critical section. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, ir & 0xF0FF)
Set Mode Register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xF0FF)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**_IMR_**, imr & 0xF0FF)
Set *IMR* register. More...

#define **getIMR**() (**WIZCHIP_READ**(**_IMR_**) & 0xF0FF)
Get *IMR* register. More...

#define **setSHAR**(shar)
Set local MAC address. More...

#define **getSHAR**(shar)
Get local MAC address. More...

#define **setGAR**(gar)
Set gateway IP address. More...

#define **getGAR**(gar)
Get gateway IP address. More...

#define **setSUBR**(subr)
Set subnet mask address. More...

#define **getSUBR**(subr)
Get subnet mask address. More...

#define **setSIPR**(sipr)
Set local IP address. More...

#define **getSIPR**(sipr)
Get local IP address. More...

#define **setRTR**(rtr) **WIZCHIP_WRITE**(**_RTR_**, rtr)
Set *RTR* register. More...

#define **getRTR**() **WIZCHIP_READ**(**_RTR_**)
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, ((uint16_t)rcr)&0x00F
Set *RCR* register. More...

#define **getRCR**() ((uint8_t)(**WIZCHIP_READ**(**_RCR_**) & 0x00FF))
Get *RCR* register. More...

#define **setTMS01R**(tms01r) **WIZCHIP_WRITE**(**TMS01R**,tms01r)
Set **TMS01R** register. More...

#define **getTMS01R**() **WIZCHIP_READ**(**TMS01R**)
Get **TMS01R** register. More...

#define **setTMS23R**(tms23r) **WIZCHIP_WRITE**(**TMS23R**,tms23r)
Set **TMS23R** register. More...

#define **getTMS23R**() **WIZCHIP_READ**(**TMS23R**)
Get **TMS23R** register. More...

#define **setTMS45R**(tms45r) **WIZCHIP_WRITE**(**TMS45R**,tms45r)
Set **TMS45R** register. More...

#define **getTMS45R**() **WIZCHIP_READ**(**TMS45R**)
Get **TMS45R** register. More...

#define **setTMS67R**(tms67r) **WIZCHIP_WRITE**(**TMS67R**,tms67r)
Set **TMS67R** register. More...

#define **getTMS67R**() **WIZCHIP_READ**(**TMS67R**)
Get **TMS67R** register. More...

#define **setSn_TXBUF_SIZE**(sn, tmsr) **setTMSR**(sn, tmsr)
For compatible ioLibrary. More...

#define **getSn_TXBUF_SIZE**(sn) **getTMSR**(sn)
For compatible ioLibrary. More...

#define **setRMS01R**(rms01r) **WIZCHIP_WRITE**(**RMS01R**,rms01r)
Set **RMS01R** register. More...

#define **getRMS01R**() **WIZCHIP_READ**(**RMS01R**)
Get **RMS01R** register. More...

#define **setRMS23R**(rms23r) **WIZCHIP_WRITE**(**RMS23R**,rms23r)
Set **RMS23R** register. More...

#define **getRMS23R**() **WIZCHIP_READ**(**RMS23R**)
Get **RMS23R** register. More...

#define **setRMS45R**(rms45r) **WIZCHIP_WRITE**(**RMS45R**,rms45r)
Set **RMS45R** register. More...

#define **getRMS45R**() **WIZCHIP_READ**(**RMS45R**)
Get **RMS45R** register. More...

#define **setRMS67R**(rms67r) **WIZCHIP_WRITE**(**RMS67R**,rms67r)
Set **RMS67R** register. More...

#define **getRMS67R**() **WIZCHIP_READ**(**RMS67R**)
Get **RMS67R** register. More...

#define **setSn_RXBUF_SIZE**(sn, rmsr) **setRMSR**(sn, rmsr)
For compatible ioLibrary. More...

#define **getSn_RXBUF_SIZE**(sn) **getRMSR**(sn)
For compatible ioLibrary. More...

#define **setMTYPER**(mtype) **WIZCHIP_WRITE**(**MTYPER**, mtype)
Set **MTYPER** register. More...

#define **getMTYPER**() **WIZCHIP_READ**(**MTYPER**)
Get **MTYPER** register. More...

#define **getPATR**() **WIZCHIP_READ**(**PATR**)
Get RATR register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ((uint16_t)pt
0x00FF)

Set **PTIMER** register. More...

#define **getPTIMER**() ((uint8_t)(**WIZCHIP_READ**(**PTIMER**) & 0x00FF
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, ((uint16_t)
0x00FF)
Set **PMAGIC** register. More...

#define **getPMAGIC**() ((uint8_t)(**WIZCHIP_READ**(**PMAGIC**) & 0x00F
Get **PMAGIC** register. More...

#define **getPSIDR**() **WIZCHIP_READ**(**PSIDR**)
Get **PSID** register. More...

#define **getPDHAR**(pdhar)
Get **PDHAR** register. More...

#define **getUIPR**(uipr)
Get unreachable IP address. **UIPR**. More...

#define **getUPORTR**() **WIZCHIP_READ**(**UPORTR**)
Get **UPORTR** register. More...

#define **getFMTUR**() **WIZCHIP_READ**(**FMTUR**)
Get **FMTUR** register. More...

#define **getPn_BRDYR**(p) ((uint8_t)(**WIZCHIP_READ**(**Pn_BRDYR**(p
Get **Pn_BRDYR** register. More...

#define **setPn_BRDYR**(p, brdyr) **WIZCHIP_WRITE**(**Pn_BRDYR**(p), b
Set **Pn_BRDYR** register. More...

#define **getPn_BDPTHR**(p) **WIZCHIP_READ**(**Pn_BDPTHR**(p))
Get **Pn_BDPTHR** register. More...

#define **setPn_BDPTHR**(p, bdpthr) **WIZCHIP_WRITE**(**Pn_BDPTHR**(
Set **Pn_BDPTHR** register. More...

#define **getIDR**() **WIZCHIP_READ**(**IDR**)
Get **IDR** register. More...

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), ((uint16_t)c
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_CR**(sn)))
Get **Sn_CR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), ((uint16_
0x00FF)
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_IMR**(sn)))
Get **Sn_IMR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), ((uint16_t)ir) *&*
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_IR**(sn)))
Get **Sn_IR** register. More...

#define **getSn_SSR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_SR**(sn)))
Get **Sn_SR** register. More...

#define **getSn_SR**(sn) **getSn_SSR**(sn)
For compatible ioLibrary. Refer to **getSn_SSR()**. More...

#define **setSn_PORTR**(sn, port) **WIZCHIP_WRITE**(**Sn_PORTR**(sn),
Set **Sn_PORTR** register. More...

#define **setSn_PORT**(sn, port) **setSn_PORTR**(sn, port)
For compatible ioLibrary. More...

#define **getSn_PORTR**(sn, port) **WIZCHIP_READ**(**Sn_PORTR**(sn))
Get **Sn_PORTR** register. More...

#define **getSn_PORT**(sn) **getSn_PORTR**(sn)
For compatible ioLibrary. More...

#define **setSn_DHAR**(sn, dhar)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar)
Get **Sn_MR** register. More...

#define **setSn_DPORTR**(sn, dport) **WIZCHIP_WRITE**(**Sn_DPORTR**(
Set **Sn_DPORT** register. More...

#define **setSn_DPORT**(sn, dport) **setSn_DPORTR**(sn,dport)
For compatible ioLibrary. Refer to **Sn_DPORTR**. More...

#define **getSn_DPORTR**(sn) **WIZCHIP_READ**(**Sn_DPORTR**(sn))
Get **Sn_DPORT** register. More...

#define **getSn_DPORT**(sn) **getSn_DPORTR**(sn)
For compatible ioLibrary. Refer to **Sn_DPORTR**. More...

#define **setSn_DIPR**(sn, dipr)
Set **Sn_DIPR** register. More...

#define **getSn_DIPR**(sn, dipr)
Get **Sn_DIPR** register. More...

#define **setSn_MSSR**(sn, mss) **WIZCHIP_WRITE**(**Sn_MSSR**(sn), ms
Set **Sn_MSSR** register. More...

#define **getSn_MSSR**(sn) **WIZCHIP_READ**(**Sn_MSSR**(sn))
Get **Sn_MSSR** register. More...

#define **setSn_KPALVTR**(sn, kpalvt) **WIZCHIP_WRITE**(**Sn_KPALVT**
(**WIZCHIP_READ**(**Sn_KPALVTR**(sn)) & 0x00FF) | (((uint16_t)l
Set **Sn_KPALVTR** register. More...

#define **getSn_KPALVTR**(sn) ((uint8_t)(**WIZCHIP_READ**(**Sn_KPALV**
Get **Sn_KPALVTR** register. More...

#define **setSn_PROTOR**(sn, proto) **WIZCHIP_WRITE**(**Sn_PROTOR**(
(**WIZCHIP_READ**(**Sn_PROTOR**(sn) & 0xFF00) | (((uint16_t)pr
0x00FF))
Set **Sn_PROTOR** register. More...

#define **setSn_PROTO**(sn, proto) **setSn_PROTOR**(sn,proto)
For compatible ioLibrary. More...

#define **getSn_PROTOR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_PROTO**
Get **Sn_PROTOR** register. More...

#define **getSn_PROTO**(sn) **getSn_PROTOR**(sn)
For compatible ioLibrary. More...

#define **setSn_TX_WRSR**(sn, txwrs)
Set **Sn_TX_WRSR** register. More...

#define **getSn_TX_WRSR**(sn) ((((uint32_t)**WIZCHIP_READ**(**Sn_TX_**
<< 16) +
(((uint32_t)**WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_**
& 0x0000FFFF))
Get **Sn_TX_WRSR** register. More...

#define **setSn_TX_FIFOR**(sn, txfifo) **WIZCHIP_WRITE**(**Sn_TX_FIFO**
Set **Sn_TX_FIFOR** register. More...

#define **getSn_RX_FIFOR**(sn) **WIZCHIP_READ**(**Sn_RX_FIFOR**(sn))
Get **Sn_RX_FIFOR** register. More...

#define **setSn_TOSR**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), ((uint1
0x00FF)
Set **Sn_TOSR** register. More...

#define **setSn_TOS**(sn, tos) **setSn_TOSR**(sn,tos)
For compatible ioLibrar. More...

#define **getSn_TOSR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_TOSR**(sn)))
Get **Sn_TOSR** register. More...

#define **getSn_TOS**(sn) **getSn_TOSR**(sn)
For compatible ioLibrar. More...

#define **setSn_TTLR**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTLR**(sn), ((uint1
0x00FF)
Set **Sn_TTLR** register. More...

#define **setSn_TTL**(sn, ttl) **setSn_TTLR**(sn,ttl)
For compatible ioLibrary. More...

#define **getSn_TTLR**(sn) ((uint8_t)**WIZCHIP_READ**(**Sn_TTL**(sn)))
Get **Sn_TTLR** register. More...

#define **getSn_TTL**(sn) **getSn_TTLR**(sn)
For compatible ioLibrary. More...

#define **setSn_FRAGR**(sn, frag) **WIZCHIP_WRITE**(**Sn_FRAGR**(sn),
>>8))
Set **Sn_FRAGR** register. More...

#define **setSn_FRAG**(sn, frag) **setSn_FRAGR**(sn,flag)

#define **getSn_FRAGR**(sn) (**WIZCHIP_READ**(**Sn_FRAG**(sn)) << 8)
Get **Sn_FRAGR** register. More...

#define **getSn_FRAG**(sn) **getSn_FRAGR**(sn)

#define **getSn_RxMAX**(sn) (((uint32_t)**getSn_RXBUF_SIZE**(sn)) <<
Socket_register_access_function_W5300. More...

#define **getSn_TxMAX**(sn) (((uint32_t)**getSn_TXBUF_SIZE**(sn)) <<
Socket_register_access_function_W5300. More...

Functions

uint16_t **WIZCHIP_READ** (uint32_t AddrSel)
It reads 1 byte value from a register. More...

void **WIZCHIP_WRITE** (uint32_t AddrSel, uint16_t wb)
It writes 1 byte value to a register. More...

void **setTMSR** (uint8_t sn, uint8_t tmsr)
Set **TMSR0** ~ **TMSR7** register. More...

uint8_t **getTMSR** (uint8_t sn)
Get **TMSR0** ~ **TMSR7** register. More...

void **setRMSR** (uint8_t sn, uint8_t rmsr)
Set **RMS01R** ~ **RMS67R** register. More...

uint8_t **getRMSR** (uint8_t sn)
Get **RMS01R** ~ **RMS67R** register. More...

uint32_t **getSn_TX_FSR** (uint8_t sn)
Get **Sn_TX_FSR** register. More...

uint32_t **getSn_RX_RSR** (uint8_t sn)
Get **Sn_RX_RSR** register. More...

void **wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint32_t len)
It copies data to internal TX memory. More...

void **wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint32_t len)
It copies data to your buffer from internal RX memory. More...

void **wiz_recv_ignore** (uint8_t sn, uint32_t len)
It discard the received data in RX memory. More...

Detailed Description

W5300 HAL implement File.

W5300 HAL Header File.

Version
1.0.0

Date
2015/05/01

Revision history
<2015/05/01> 1st Released for integrating with ioLibrary
Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >>
https://github.com/Wiznet/ioLibrary_Driver

Author
MidnightCow

Copyright

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

```
* Redistributions of source code must retain the a
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the

distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **w5300.h**.

Macro Definition Documentation

#define _WIZCHIP_SN_BASE_ (0x0200)

Definition at line **53** of file **w5300.h**.

#define _WIZCHIP_SN_SIZE_ (0x0040)

Definition at line **54** of file **w5300.h**.

#define WIZCHIP_CREG_BLOCK 0x00

Common register block.

Definition at line **57** of file **w5300.h**.

#define WIZCHIP_SREG_BLOCK (N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_*N)

Socket N register block.

Definition at line **58** of file **w5300.h**.

#define WIZCHIP_OFFSET_INC (ADDR, N) (ADDR + N)

Increase offset address.

Definition at line **60** of file **w5300.h**.

#define _W5300_IO_BASE_ _WIZCHIP_IO_BASE_

Definition at line **63** of file **w5300.h**.

#define IINCHIP_READ (ADDR) WIZCHIP_READ(ADDR)

The defined for legacy chip driver.

Definition at line **75** of file **w5300.h**.

**#define IINCHIP_WRITE (ADDR,
VAL
) WIZCHIP_WRITE(ADDR,VAL)**

The defined for legacy chip driver.

Definition at line **76** of file **w5300.h**.

#define RMSR2 (RMSR0 + 2)

RX memory size of SOCKET 2.

refer to **RMS01R**

Definition at line **445** of file **w5300.h**.

#define VERSIONR IDR

Definition at line **602** of file **w5300.h**.

#define Sn_SR (n) Sn_SSR(n)

For Compatible ioLibrary. Refer to **Sn_SSR(n)**

Definition at line **726** of file **w5300.h**.

#define Sn_PORT (n) Sn_PORTR(n)

For compatible ioLibrary. Refer to **Sn_PORTR(n)**.

Definition at line **735** of file **w5300.h**.

#define Sn_DPORT (n) Sn_DPORTR(n)

For compatible ioLibrary. Refer to **Sn_DPORTR**.

Definition at line **754** of file **w5300.h**.

#define Sn_TOS (n) Sn_TOSR(n)

For compatible ioLibrary. Refer to Sn_TOSR.

Definition at line **803** of file **w5300.h**.

#define Sn_TTL (n) Sn_TTLR(n)

For compatible ioLibrary. Refer to Sn_TTLR.

Definition at line **812** of file **w5300.h**.

#define Sn_RX_RSR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0028)

Received data size register(R)

Sn_RX_RSR indicates the data size received and saved in Socket n RX Buffer. **Sn_RX_RSR** does not exceed the RMSR such as RMS01SR and is calculated as the difference between ?Socket n RX Write Pointer (**Sn_RX_WR**)and Socket n RX Read Pointer (**Sn_RX_RD**)

Definition at line **841** of file **w5300.h**.

#define Sn_FRAG (n) Sn_FRAGR(n)

Definition at line **849** of file **w5300.h**.

#define MR_DBW (1 << 15)

Data bus width bit of **MR**. Read Only. (0 : 8Bit, 1 : 16Bit)

Definition at line **889** of file **w5300.h**.

#define MR_MPF (1 << 14)

Mac layer pause frame bit of **MR**. (0 : Disable, 1 : Enable)

Definition at line **890** of file **w5300.h**.

#define MR_WDF (X) ((X & 0x07) << 11)

Write data fetch time bit of **MR**. Fetch Data from DATA bus after PLL_CLK * MR_WDF[2:0]

Definition at line **891** of file **w5300.h**.

#define MR_RDH (1 << 10)

Read data hold time bit of **MR**. Hold Data on DATA bus during 2 * PLL_CLK after CS high

Definition at line **892** of file **w5300.h**.

#define MR_FS (1 << 8)

FIFO swap bit of **MR**. Swap MSB & LSB of **Sn_TX_FIFOR** & Sn_RX_FIFOR (0 : No swap, 1 : Swap)

Definition at line **893** of file **w5300.h**.

Referenced by **recv()**, and **recvfrom()**.

#define MR_RST (1 << 7)

S/W reset bit of **MR**. (0 : Normal Operation, 1 : Reset (automatically clear after reset))

Definition at line **894** of file **w5300.h**.

#define MR_MT (1 << 5)

Memory test bit of **MR**. (0 : Normal, 1 : Internal Socket memory write & read Test)

Definition at line **895** of file **w5300.h**.

#define MR_PB (1 << 4)

Ping block bit of **MR**. (0 : Unblock, 1 : Block)

Definition at line **896** of file **w5300.h**.

#define MR_PPPoE (1 << 3)

PPPoE bit of **MR**. (0 : No use PPPoE, 1: Use PPPoE)

Definition at line **897** of file **w5300.h**.

#define MR_DBS (1 << 2)

Data bus swap of **MR**. Valid only 16bit mode (0 : No swap, 1 : Swap)

Definition at line **898** of file **w5300.h**.

#define MR_IND (1 << 0)

Indirect mode bit of **MR**. (0 : Direct mode, 1 : Indirect mode)

Definition at line **899** of file **w5300.h**.

#define IR_IPCF (1 << 15)

IP conflict bit of **IR**. To clear, Write the bit to '1'.

Definition at line **905** of file **w5300.h**.

#define IR_DPUR (1 << 14)

Destination port unreachable bit of **IR**. To clear, Write the bit to '1'.

Definition at line **906** of file **w5300.h**.

#define IR_PPPT (1 << 13)

PPPoE terminate bit of **IR**. To clear, Write the bit to '1'.

Definition at line **907** of file **w5300.h**.

#define IR_FMTU (1 << 12)

Fragment MTU bit of IR. To clear, Write the bit to '1'.

Definition at line **908** of file **w5300.h**.

#define IR_SnINT (n) (0x01 << n)

SOCKETn interrupt occurrence bit of **IR**. To clear, Clear **Sn_IR**

Definition at line **909** of file **w5300.h**.

#define Pn_PEN (1 << 7)

PIN 'BRDYn' enable bit of Pn_BRDYR.

Definition at line **914** of file **w5300.h**.

#define Pn_MT (1 << 6)

PIN memory type bit of Pn_BRDYR.

Definition at line **915** of file **w5300.h**.

#define Pn_PPL (1 << 5)

PIN Polarity bit of Pn_BRDYR.

Definition at line **916** of file **w5300.h**.

#define Pn_SN (n) ((n & 0x07) << 0)

What socket to monitor.

Definition at line **917** of file **w5300.h**.

#define Sn_MR_ALIGN (1 << 8)

Alignment bit of **Sn_MR**.

It is valid only in the TCP (**Sn_MR_TCP**) with TCP communication, when every the received DATA packet size is of even number and set as '1', data receiving performance can be improved by removing PACKET-INFO(data size) that is attached to every the received DATA packet.

Definition at line **929** of file **w5300.h**.

Referenced by **recv()**.

#define Sn_MR_MULTI (1 << 7)

Multicasting bit of **Sn_MR**.

It is valid only in UDP (**Sn_MR_UDP**). In order to implement multicasting, set the IP address and port number in **Sn_DIPR** and **Sn_DPORTR** respectively before "OPEN" command(**Sn_CR_OPEN**).
0 : Disable, 1 : Enable

Definition at line **937** of file **w5300.h**.

#define Sn_MR_MF (1 << 6)

MAC filter bit of **Sn_MR**.

It is valid in MACRAW(**Sn_MR_MACRAW**). When this bit is set as ‘1’, W5300 can receive packet that is belong in itself or broadcasting. When this bit is set as ‘0’, W5300 can receive all packets on Ethernet. When using the hybrid TCP/IP stack, it is recommended to be set as ‘1’ for reducing the receiving overhead of host.
0 : Disable, 1 : Enable

Definition at line **947** of file **w5300.h**.

#define Sn_MR_IGMPv (1 << 5)

IGMP version bit of **Sn_MR** details It is valid in case of **Sn_MR_MULTI**='1' and UDP(**Sn_MR_UDP**). It configures IGMP version to send IGMP message such as **Join/Leave/Report** to multicast-group.
0 : IGMPv2, 1 : IGMPv1.

Definition at line **955** of file **w5300.h**.

#define Sn_MR_MC Sn_MR_IGMPv

For compatible ioLibrary.

Definition at line **956** of file **w5300.h**.

#define Sn_MR_ND (1 << 5)

No delayed ack bit of **Sn_MR**.

It is valid in TCP(**Sn_MR_TCP**). In case that it is set as '1', ACK packet is transmitted right after receiving DATA packet from the peer. It is recommended to be set as '1' for TCP performance improvement. In case that it is set as '0', ACK packet is transmitted after the time set in *RTR* regardless of DATA packet receipt.

0 : No use, 1 : Use

Definition at line **966** of file **w5300.h**.

#define Sn_MR_CLOSE 0x00

No mode.

This configures the protocol mode of Socket n.

See also
Sn_MR

Definition at line **973** of file **w5300.h**.

#define Sn_MR_TCP 0x01

TCP mode.

This configures the protocol mode of Socket n.

See also
Sn_MR

Definition at line **980** of file **w5300.h**.

#define Sn_MR_UDP 0x02

UDP mode.

This configures the protocol mode of Socket n.

See also
Sn_MRProtocol bits of **Sn_MR**.

Definition at line **987** of file **w5300.h**.

#define Sn_MR_IPRAW 0x03

IP LAYER RAW mode.

This configures the protocol mode of Socket n.

See also
Sn_MR Protocol bits of **Sn_MR**.

Definition at line **994** of file **w5300.h**.

#define Sn_MR_MACRAW 0x04

MAC LAYER RAW mode.

This configures the protocol mode of Socket 0.

See also
Sn_MR

Note
MACRAW mode should be only used in Socket 0.

Definition at line **1002** of file **w5300.h**.

#define Sn_MR_PPPoE 0x05

PPPoE mode.

This configures the protocol mode of Socket 0.

See also
Sn_MR

Note
PPPoE mode should be only used in Socket 0.Protocol bits of **Sn_MR**.

Definition at line **1010** of file **w5300.h**.

#define SOCK_STREAM Sn_MR_TCP

For Berkeley Socket API, Refer to **Sn_MR_TCP**

Definition at line **1012** of file **w5300.h**.

#define SOCK_DGRAM Sn_MR_UDP

For Berkeley Socket API, Refer to **Sn_MR_UDP**

Definition at line **1013** of file **w5300.h**.

#define Sn_CR_OPEN 0x01

Initialize or open a socket.

Socket n is initialized and opened according to the protocol selected in **Sn_MR(P3:P0)**. The table below shows the value of **Sn_SR** corresponding to **Sn_MR**.

Sn_MR (P[3:0])	**Sn_SR**
Sn_MR_CLOSE (000)	
Sn_MR_TCP (001)	SOCK_INIT (0x13)
Sn_MR_UDP (010)	SOCK_UDP (0x22)
Sn_MR_IPRAW (010)	SOCK_IPRAW (0x32)
Sn_MR_MACRAW (100)	SOCK_MACRAW (0x42)
Sn_MR_PPPoE (101)	SOCK_PPPoE (0x5F)

Definition at line **1034** of file **w5300.h**.

#define Sn_CR_LISTEN 0x02

Wait connection request in TCP mode(Server mode)

This is valid only in TCP mode (**Sn_MR(P3:P0)** = **Sn_MR_TCP**). In this mode, Socket n operates as a TCP serverand waits for connection-request (SYN packet) from any TCP client The **Sn_SR** changes the state from **SOCK_INIT** to SOCKET_LISTEN. When a TCP clientconnection request is successfully established, the **Sn_SR** changes from SOCK_LISTEN to SOCK_ESTABLISHED and the **Sn_IR(0)** becomes But when a TCP clientconnection request is failed, **Sn_IR(3)** becomes and the status of **Sn_SR** changes to SOCK_CLOSED.

Definition at line **1045** of file **w5300.h**.

#define Sn_CR_CONNECT 0x04

Send connection request in TCP mode(Client mode)

To connect, a connect-request (SYN packet) is sent to **TCP server**configured by **Sn_DIPR** & **Sn_DPORT(destination address & port)**. If the connect-request is successful, the **Sn_SR** is changed to **SOCK_ESTABLISHED** and the **Sn_IR(0)** becomes

The connect-request fails in the following three cases.

1. When a **ARPTO** occurs (**Sn_IR**[3] = '1') because destination hardware address is not acquired through the ARP-process.
2. When a **SYN/ACK** packet is not received and **TCPTO** (**Sn_IR(3)** =)
3. When a **RST** packet is received instead of a **SYN/ACK** packet. In these cases, **Sn_SR** is changed to **SOCK_CLOSED**.

   Note

   This is valid only in TCP mode and operates when Socket n acts as **TCP client**

Definition at line **1057** of file **w5300.h**.

#define Sn_CR_DISCON 0x08

Send closing request in TCP mode.

Regardless of **TCP server**or **TCP client** the DISCON command processes the disconnect-process (b>Active closeor **Passive close**.

Active close
: it transmits disconnect-request(FIN packet) to the connected peer

Passive close
: When FIN packet is received from peer, a FIN packet is replied back to the peer.

When the disconnect-process is successful (that is, FIN/ACK packet is received successfully), **Sn_SR** is changed to **SOCK_CLOSED**. Otherwise, **TCPTO** occurs (**Sn_IR**[3]='1') and then **Sn_SR** is changed to **SOCK_CLOSED**.

Note
: Valid only in TCP mode.

Definition at line **1070** of file **w5300.h**.

#define Sn_CR_CLOSE 0x10

Close socket.

Sn_SR is changed to **SOCK_CLOSED**.

Definition at line **1076** of file **w5300.h**.

#define Sn_CR_SEND 0x20

Update TX buffer pointer and send data.

SEND command transmits all the data in the Socket n TX buffer thru **Sn_TX_FIFOR**.
For more details, please refer to Socket n TX Free Size Register

(**Sn_TX_FSR**) and Socket TX Write Size register (**Sn_TX_WRSR**).

Definition at line **1083** of file **w5300.h**.

#define Sn_CR_SEND_MAC 0x21

Send data with MAC address, so without ARP process.

The basic operation is same as SEND.
Normally SEND command transmits data after destination hardware address is acquired by the automatic ARP-process(Address Resolution Protocol).
But SEND_MAC command transmits data without the automatic ARP-process.
In this case, the destination hardware address is acquired from **Sn_DHAR** configured by host, instead of APR-process.

Note
Valid only in UDP mode.

Definition at line **1093** of file **w5300.h**.

#define Sn_CR_SEND_KEEP 0x22

Send keep alive message.

It checks the connection status by sending 1byte keep-alive packet.
If the peer can not respond to the keep-alive packet during timeout time, the connection is terminated and the timeout interrupt will occur.

Note
Valid only in TCP mode.

Definition at line **1101** of file **w5300.h**.

#define Sn_CR_RECV 0x40

Update RX buffer pointer and receive data.

RECV completes the processing of the received data in Socket n RX Buffer thru **Sn_RX_FIFOR**).
For more details, refer to Socket n RX Received Size Register (**Sn_RX_RSR**) & Sn_RX_FIFOR.RECV command value of **Sn_CR**

Definition at line **1108** of file **w5300.h**.

#define Sn_CR_PCON 0x23

PPPoE connection begins by transmitting PPPoE discovery packet. Refer to **Sn_CR**

Definition at line **1110** of file **w5300.h**.

#define Sn_CR_PDISCON 0x24

Closes PPPoE connection. Refer to **Sn_CR**

Definition at line **1111** of file **w5300.h**.

#define Sn_CR_PCR 0x25

In each phase, it transmits REQ message. Refer to **Sn_CR**

Definition at line **1112** of file **w5300.h**.

#define Sn_CR_PCN 0x26

In each phase, it transmits NAK message. Refer to **Sn_CR**

Definition at line **1113** of file **w5300.h**.

#define Sn_CR_PCJ 0x27

In each phase, it transmits REJECT message. Refer to **Sn_CR**

Definition at line **1114** of file **w5300.h**.

#define Sn_IR_PRECV 0x80

It is set in the case that option data which is not supported is received. Refer to **Sn_IR**

Definition at line **1120** of file **w5300.h**.

#define Sn_IR_PFAIL 0x40

It is set in the case that PAP authentication is failed. Refer to **Sn_IR**

Definition at line **1121** of file **w5300.h**.

#define Sn_IR_PNEXT 0x20

It is set in the case that the phase is changed during PPPoE connection process. **Sn_IR**

Definition at line **1122** of file **w5300.h**.

#define Sn_IR_SENDOK 0x10

It is set when SEND command is completed. Refer to **Sn_IR**

Definition at line **1123** of file **w5300.h**.

#define Sn_IR_TIMEOUT 0x08

It is set when ARPTO or TCPTO is occured. Refer to **Sn_IR**

Definition at line **1124** of file **w5300.h**.

#define Sn_IR_RECV 0x04

It is set whenever data is received from a peer. Refer to **Sn_IR**

Definition at line **1125** of file **w5300.h**.

#define Sn_IR_DISCON 0x02

It is set when FIN or FIN/ACK packet is received from a peer. Refer to **Sn_IR**

Definition at line **1126** of file **w5300.h**.

#define Sn_IR_CON 0x01

It is set one time when the connection is successful and then **Sn_SR** is changed to **SOCK_ESTABLISHED**.

Definition at line **1127** of file **w5300.h**.

#define SOCK_CLOSED 0x00

The state of SOCKET intialized or closed.

This indicates that Socket n is released.
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to **SOCK_CLOSED** regardless of previous status.

Definition at line **1137** of file **w5300.h**.

#define SOCK_ARP 0x01

The state of ARP process.

It is temporary state for getting a peer MAC address when TCP connect or UDP Data Send
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to **SOCK_CLOSED** regardless of previous status.ARP-request is transmitted in order to acquire destination hardware address.

Definition at line **1144** of file **w5300.h**.

#define SOCK_INIT 0x13

Initiate state in TCP.

This indicates Socket n is opened with TCP mode.
It is changed to **SOCK_INIT** when **Sn_MR(P[3:0])** = '001' and OPEN command(**Sn_CR_OPEN**) is ordered.
After SOCK_INIT, user can use LISTEN(**Sn_CR_LISTEN**)/CONNECT(Sn_CR_CONNET) command.

Definition at line **1152** of file **w5300.h**.

#define SOCK_LISTEN 0x14

Listen state.

This indicates Socket n is operating as **TCP server**mode and waiting for connection-request (SYN packet) from a peer **TCP client**.
It will change to SOCK_ESTALBLISHED when the connection-request is successfully accepted.
Otherwise it will change to **SOCK_CLOSED** after TCPTO

(**Sn_IR_TIMEOUT** = '1') is occurred.

Definition at line **1160** of file **w5300.h**.

#define SOCK_SYNSENT 0x15

Connection state.

This indicates Socket n sent the connect-request packet (SYN packet) to a peer.
It is temporarily shown when **Sn_SR** is changed from **SOCK_INIT** to **SOCK_ESTABLISHED** by **Sn_CR_CONNECT** command.
If connect-accept(SYN/ACK packet) is received from the peer at SOCK_SYNSENT, it changes to **SOCK_ESTABLISHED**.
Otherwise, it changes to **SOCK_CLOSED** after TCPTO (**Sn_IR_TIMEOUT** = '1') is occurred.

Definition at line **1169** of file **w5300.h**.

#define SOCK_SYNRECV 0x16

Connection state.

It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
If socket n sends the response (SYN/ACK packet) to the peer successfully, it changes to **SOCK_ESTABLISHED**.
If not, it changes to **SOCK_CLOSED** after timeout (**Sn_IR_TIMEOUT** = '1') is occurred.

Definition at line **1177** of file **w5300.h**.

#define SOCK_ESTABLISHED 0x17

Success to connect.

This indicates the status of the connection of Socket n.
It changes to **SOCK_ESTABLISHED** when the **TCP SERVER**processed the SYN packet from the **TCP CLIENT**during **SOCK_LISTEN**, or when the **Sn_CR_CONNECT** command is successful.
During **SOCK_ESTABLISHED**, DATA packet can be transferred using **Sn_CR_SEND** or **Sn_CR_RECV** command.

Definition at line **1186** of file **w5300.h**.

#define SOCK_FIN_WAIT 0x18

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout(Sn_CR_TIMTEOUT = '1') is occurred, these change to **SOCK_CLOSED**.

Definition at line **1194** of file **w5300.h**.

#define SOCK_CLOSING 0x1A

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1202** of file **w5300.h**.

#define SOCK_TIME_WAIT 0x1B

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1210** of file **w5300.h**.

#define SOCK_CLOSE_WAIT 0x1C

Closing state.

This indicates Socket n received the disconnect-request (FIN packet) from the connected peer.
This is half-closing status, and data can be transferred.
For full-closing, **Sn_CR_DISCON** command is used. But For just-closing, **Sn_CR_CLOSE** command is used.

Definition at line **1218** of file **w5300.h**.

#define SOCK_LAST_ACK 0x1D

Closing state.

This indicates Socket n is waiting for the response (FIN/ACK packet) to the disconnect-request (FIN packet) by passive-close.
It changes to **SOCK_CLOSED** when Socket n received the response successfully, or when timeout (**Sn_IR_TIMEOUT** = '1') is occurred.

Definition at line **1225** of file **w5300.h**.

#define SOCK_UDP 0x22

UDP socket.

This indicates Socket n is opened in UDP mode(**Sn_MR(P[3:0])** = '010').
It changes to SOCK_UDP when **Sn_MR(P[3:0])** = '010' and **Sn_CR_OPEN** command is ordered.
Unlike TCP mode, data can be transfered without the connection-process.

Definition at line **1233** of file **w5300.h**.

#define SOCK_IPRAW 0x32

IP raw mode socket.

TThe socket is opened in IPRAW mode. The SOCKET status is change to SOCK_IPRAW when **Sn_MR** (P3:P0) is Sn_MR_IPRAW and **Sn_CR_OPEN** command is used.
IP Packet can be transferred without a connection similar to the UDP mode.

Definition at line **1241** of file **w5300.h**.

#define SOCK_MACRAW 0x42

MAC raw mode socket.

This indicates Socket 0 is opened in MACRAW mode (**Sn_MR(P[3:0])** = '100' and n = 0) and is valid only in Socket 0.
It changes to SOCK_MACRAW when **Sn_MR**(P[3:0] = 100)and @ ref Sn_CR_OPEN command is ordered.
Like UDP mode socket, MACRAW mode Socket 0 can transfer a MAC packet (Ethernet frame) without the connection-process.SOCKET0 is open as MACRAW mode.

Definition at line **1249** of file **w5300.h**.

#define SOCK_PPPoE 0x5F

PPPoE mode socket.

It is the status that SOCKET0 is opened as PPPoE mode. It is changed to SOCK_PPPoE in case of **Sn_CR_OPEN** command is ordered and **Sn_MR(P3:P0)**= **Sn_MR_PPPoE**
It is temporarily used at the PPPoE connection.SOCKET0 is open as PPPoE mode.

Definition at line **1257** of file **w5300.h**.

#define IPPROTO_IP 0

Definition at line **1260** of file **w5300.h**.

#define IPPROTO_ICMP 1

Definition at line **1261** of file **w5300.h**.

#define IPPROTO_IGMP 2

Definition at line **1262** of file **w5300.h**.

#define IPPROTO_GGP 3

Definition at line **1263** of file **w5300.h**.

#define IPPROTO_TCP 6

Definition at line **1264** of file **w5300.h**.

#define IPPROTO_PUP 12

Definition at line **1265** of file **w5300.h**.

#define IPPROTO_UDP 17

Definition at line **1266** of file **w5300.h**.

#define IPPROTO_IDP 22

Definition at line **1267** of file **w5300.h**.

#define IPPROTO_ND 77

Definition at line **1268** of file **w5300.h**.

#define IPPROTO_RAW 255

Definition at line **1269** of file **w5300.h**.

#define WIZCHIP_CRITICAL_ENTER ( ) WIZCHIP.CRIS._enter()

Enter a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**

WIZCHIP_CRITICAL_EXIT()

Definition at line **1283** of file **w5300.h**.

#define WIZCHIP_CRITICAL_EXIT ( ) WIZCHIP.CRIS._exit()

Exit a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also

WIZCHIP_READ(), **WIZCHIP_WRITE()**

WIZCHIP_CRITICAL_ENTER()

Definition at line **1300** of file **w5300.h**.

#define setSn_TXBUF_SIZE (sn, tmsr) setTMSR(sn, tmsr)

For compatible ioLibrary.

Definition at line **1617** of file **w5300.h**.

#define getSn_TXBUF_SIZE (sn) getTMSR(sn)

For compatible ioLibrary.

Definition at line **1627** of file **w5300.h**.

```
#define setSn_RXBUF_SIZE (  sn,
                            rmsr
                         )    setRMSR(sn, rmsr)
```

For compatible ioLibrary.

Definition at line **1709** of file **w5300.h**.

```
#define getSn_RXBUF_SIZE (  sn )   getRMSR(sn)
```

For compatible ioLibrary.

Definition at line **1719** of file **w5300.h**.

```
#define getSn_SR (  sn )   getSn_SSR(sn)
```

For compatible ioLibrary. Refer to **getSn_SSR()**.

Definition at line **1971** of file **w5300.h**.

```
#define setSn_PORT (  sn,
                      port
                   )    setSn_PORTR(sn, port)
```

For compatible ioLibrary.

Definition at line **1982** of file **w5300.h**.

```
#define getSn_PORT (  sn )   getSn_PORTR(sn)
```

For compatible ioLibrary.

Definition at line **1993** of file **w5300.h**.

```
#define setSn_DPORT (  sn,
                       dport
                    )    setSn_DPORTR(sn,dport)
```

For compatible ioLibrary. Refer to **Sn_DPORTR**.

Definition at line **2033** of file **w5300.h**.

```
#define getSn_DPORT (  sn )    getSn_DPORTR(sn)
```

For compatible ioLibrary. Refer to **Sn_DPORTR**.

Definition at line **2047** of file **w5300.h**.

```
#define setSn_PROTO (  sn,
                       proto
                    )    setSn_PROTOR(sn,proto)
```

For compatible ioLibrary.

Definition at line **2124** of file **w5300.h**.

```
#define getSn_PROTO (  sn )    getSn_PROTOR(sn)
```

For compatible ioLibrary.

Definition at line **2135** of file **w5300.h**.

```
#define setSn_TOS (  sn,
                     tos
                  )    setSn_TOSR(sn,tos)
```

For compatible ioLibrar.

Definition at line **2202** of file **w5300.h**.

```
#define getSn_TOS (  sn )   getSn_TOSR(sn)
```

For compatible ioLibrar.

Definition at line **2213** of file **w5300.h**.

```
#define setSn_TTL (  sn,
                     ttl
                  )    setSn_TTLR(sn,ttl)
```

For compatible ioLibrary.

Definition at line **2224** of file **w5300.h**.

```
#define getSn_TTL (  sn )   getSn_TTLR(sn)
```

For compatible ioLibrary.

Definition at line **2235** of file **w5300.h**.

```
#define setSn_FRAG (  sn,
                      frag
                   )    setSn_FRAGR(sn,flag)
```

Definition at line **2246** of file **w5300.h**.

#define getSn_FRAG (sn) getSn_FRAGR(sn)

Definition at line **2257** of file **w5300.h**.

#define getSn_RxMAX (((uint32_t)getSn_RXBUF_SIZE(sn)) (sn) << 10)

Socket_register_access_function_W5300.

Gets the max buffer size of socket sn passed as parameter.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint32_t. Value of Socket n RX max buffer size.

Definition at line **2270** of file **w5300.h**.

#define getSn_TxMAX (((uint32_t)getSn_TXBUF_SIZE(sn)) (sn) << 10)

Socket_register_access_function_W5300.

Gets the max buffer size of socket sn passed as parameters.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint32_t. Value of Socket n TX max buffer size.

Definition at line **2279** of file **w5300.h**.

Socket APIs

W5500 Directory Reference

Files

file **w5500.c** [code]
W5500 HAL Interface.

file **w5500.h** [code]
W5500 HAL Header File.

Socket APIs

Macros

w5500.c File Reference

W5500 HAL Interface. More...

```
#include "w5500.h"
```

Go to the source code of this file.

Macros

#define **_W5500_SPI_VDM_OP_** 0x00

#define **_W5500_SPI_FDM_OP_LEN1_** 0x01

#define **_W5500_SPI_FDM_OP_LEN2_** 0x02

#define **_W5500_SPI_FDM_OP_LEN4_** 0x03

Detailed Description

W5500 HAL Interface.

Version
1.0.2

Date
2013/10/21

Revision history
<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2014/05/01> V1.0.2

1. Implicit type casting -> Explicit type casting. Refer to M20140501 Fixed the problem on porting into under 32bit MCU Issued by Mathias ClauBen, wizwiki forum ID Think01 and bobh Thank for your interesting and serious advices. <2013/12/20> V1.0.1
1. Remove warning
2. WIZCHIP_READ_BUF WIZCHIP_WRITE_BUF in case *WIZCHIP_IO_MODE_SPI_FDM* for loop optimized(removed). refer to M20131220 <2013/10/21> 1st Release

Author
MidnightCow

Copyright

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

```
* Redistributions of source code must retain the a
```

```
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **w5500.c**.

Macro Definition Documentation

#define _W5500_SPI_VDM_OP_ 0x00

Definition at line **57** of file **w5500.c**.

#define _W5500_SPI_FDM_OP_LEN1_ 0x01

Definition at line **58** of file **w5500.c**.

#define _W5500_SPI_FDM_OP_LEN2_ 0x02

Definition at line **59** of file **w5500.c**.

#define _W5500_SPI_FDM_OP_LEN4_ 0x03

Definition at line **60** of file **w5500.c**.

Socket APIs

Macros | Functions

w5500.h File Reference

W5500 HAL Header File. More...

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

Go to the source code of this file.

Macros

#define **_W5500_IO_BASE_** 0x00000000

#define **_W5500_SPI_READ_** (0x00 << 2)

#define **_W5500_SPI_WRITE_** (0x01 << 2)

#define **WIZCHIP_CREG_BLOCK** 0x00

#define **WIZCHIP_SREG_BLOCK**(N) (1+4*N)

#define **WIZCHIP_TXBUF_BLOCK**(N) (2+4*N)

#define **WIZCHIP_RXBUF_BLOCK**(N) (3+4*N)

#define **WIZCHIP_OFFSET_INC**(ADDR, N) (ADDR + (N<<8))

#define **IINCHIP_READ**(ADDR) **WIZCHIP_READ**(ADDR)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE**(ADDR, VAL) **WIZCHIP_WRITE**(ADDR,VAL)
The defined for legacy chip driver. More...

#define **IINCHIP_READ_BUF**(ADDR, BUF, LEN) **WIZCHIP_READ_BUF**(ADDR,BUF,LEN)
The defined for legacy chip driver. More...

#define **IINCHIP_WRITE_BUF**(ADDR, BUF, LEN) **WIZCHIP_WRITE**(ADDR,BUF,LEN)
The defined for legacy chip driver. More...

#define **MR** (**_W5500_IO_BASE_** + (0x0000 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Mode Register address(R/W)
MR is used for S/W reset, ping block mode, PPPoE mode and

More...

#define **GAR** (**_W5500_IO_BASE_** + (0x0001 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Gateway IP Register address(R/W) More...

#define **SUBR** (**_W5500_IO_BASE_** + (0x0005 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Subnet mask Register address(R/W) More...

#define **SHAR** (**_W5500_IO_BASE_** + (0x0009 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Source MAC Register address(R/W) More...

#define **SIPR** (**_W5500_IO_BASE_** + (0x000F << 8) + (WIZCHIP_CREG_BLOCK << 3))
Source IP Register address(R/W) More...

#define **INTLEVEL** (**_W5500_IO_BASE_** + (0x0013 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Set Interrupt low level timer register address(R/W) More...

#define **IR** (**_W5500_IO_BASE_** + (0x0015 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Interrupt Register(R/W) More...

#define **_IMR_** (**_W5500_IO_BASE_** + (0x0016 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Interrupt mask register(R/W) More...

#define **SIR** (**_W5500_IO_BASE_** + (0x0017 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Register(R/W) More...

#define **SIMR** (**_W5500_IO_BASE_** + (0x0018 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Mask Register(R/W) More...

#define **_RTR_** (**_W5500_IO_BASE_** + (0x0019 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Timeout register address(1 is 100us)(R/W) More...

#define **_RCR_** (**_W5500_IO_BASE_** + (0x001B << 8) + (WIZCHIP_CREG_BLOCK << 3))
Retry count register(R/W) More...

#define **PTIMER** (**_W5500_IO_BASE_** + (0x001C << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP LCP Request Timer register in PPPoE mode(R/W) More...

#define **PMAGIC** (**_W5500_IO_BASE_** + (0x001D << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP LCP Magic number register in PPPoE mode(R/W) More...

#define **PHAR** (**_W5500_IO_BASE_** + (0x001E << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Destination MAC Register address(R/W) More...

#define **PSID** (**_W5500_IO_BASE_** + (0x0024 << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Session Identification Register(R/W) More...

#define **PMRU** (**_W5500_IO_BASE_** + (0x0026 << 8) + (WIZCHIP_CREG_BLOCK << 3))
PPP Maximum Segment Size(MSS) register(R/W) More...

#define **UIPR** (**_W5500_IO_BASE_** + (0x0028 << 8) + (WIZCHIP_CREG_BLOCK << 3))
Unreachable IP register address in UDP mode(R) More...

#define **UPORTR** (**_W5500_IO_BASE_** + (0x002C << 8) + (WIZCHIP_CREG_BLOCK << 3))
Unreachable Port register address in UDP mode(R) More...

#define **PHYCFGR** (**_W5500_IO_BASE_** + (0x002E << 8) + (WIZCHIP_CREG_BLOCK << 3))
PHY Status Register(R/W) More...

#define **VERSIONR** (**_W5500_IO_BASE_** + (0x0039 << 8) + (WIZCHIP_CREG_BLOCK << 3))
chip version register address(R) More...

#define **Sn_MR**(N) (**_W5500_IO_BASE_** + (0x0000 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
socket Mode register(R/W) More...

#define **Sn_CR**(N) (**_W5500_IO_BASE_** + (0x0001 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Socket command register(R/W) More...

#define **Sn_IR**(N) (**_W5500_IO_BASE_** + (0x0002 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Socket interrupt register(R) More...

#define **Sn_SR**(N) (**_W5500_IO_BASE_** + (0x0003 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Socket status register(R) More...

#define **Sn_PORT**(N) (**_W5500_IO_BASE_** + (0x0004 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
source port register(R/W) More...

#define **Sn_DHAR**(N) (**_W5500_IO_BASE_** + (0x0006 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Peer MAC register address(R/W) More...

#define **Sn_DIPR**(N) (**_W5500_IO_BASE_** + (0x000C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Peer IP register address(R/W) More...

#define **Sn_DPORT**(N) (**_W5500_IO_BASE_** + (0x0010 << 8) +

	(WIZCHIP_SREG_BLOCK(N) << 3)) Peer port register address(R/W) More...
#define	**Sn_MSSR**(N) (**_W5500_IO_BASE_** + (0x0012 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Maximum Segment Size(Sn_MSSR0) register address(R/W) More...
#define	**Sn_TOS**(N) (**_W5500_IO_BASE_** + (0x0015 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) IP Type of Service(TOS) Register(R/W) More...
#define	**Sn_TTL**(N) (**_W5500_IO_BASE_** + (0x0016 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) IP Time to live(TTL) Register(R/W) More...
#define	**Sn_RXBUF_SIZE**(N) (**_W5500_IO_BASE_** + (0x001E << 8) (WIZCHIP_SREG_BLOCK(N) << 3)) Receive memory size register(R/W) More...
#define	**Sn_TXBUF_SIZE**(N) (**_W5500_IO_BASE_** + (0x001F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory size register(R/W) More...
#define	**Sn_TX_FSR**(N) (**_W5500_IO_BASE_** + (0x0020 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit free memory size register(R) More...
#define	**Sn_TX_RD**(N) (**_W5500_IO_BASE_** + (0x0022 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory read pointer register address(R) More...
#define	**Sn_TX_WR**(N) (**_W5500_IO_BASE_** + (0x0024 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3)) Transmit memory write pointer register address(R/W) More...
#define	**Sn_RX_RSR**(N) (**_W5500_IO_BASE_** + (0x0026 << 8) +

(WIZCHIP_SREG_BLOCK(N) << 3))
Received data size register(R) More...

#define **Sn_RX_RD**(N) (**_W5500_IO_BASE_** + (0x0028 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Read point of Receive memory(R/W) More...

#define **Sn_RX_WR**(N) (**_W5500_IO_BASE_** + (0x002A << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Write point of Receive memory(R) More...

#define **Sn_IMR**(N) (**_W5500_IO_BASE_** + (0x002C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
socket interrupt mask register(R) More...

#define **Sn_FRAG**(N) (**_W5500_IO_BASE_** + (0x002D << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Fragment field value in IP header register(R/W) More...

#define **Sn_KPALVTR**(N) (**_W5500_IO_BASE_** + (0x002F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
Keep Alive Timer register(R/W) More...

#define **MR_RST** 0x80
Reset. More...

#define **MR_WOL** 0x20
Wake on LAN. More...

#define **MR_PB** 0x10
Ping block. More...

#define **MR_PPPOE** 0x08
Enable PPPoE. More...

#define **MR_FARP** 0x02
Enable UDP_FORCE_ARP CHECHK. More...

#define **IR_CONFLICT** 0x80
Check IP conflict. More...

#define **IR_UNREACH** 0x40
Get the destination unreachable message in UDP sending. Mo

#define **IR_PPPoE** 0x20
Get the PPPoE close message. More...

#define **IR_MP** 0x10
Get the magic packet interrupt. More...

#define **PHYCFGR_RST** ~(1<<7)

#define **PHYCFGR_OPMD** (1<<6)

#define **PHYCFGR_OPMDC_ALLA** (7<<3)

#define **PHYCFGR_OPMDC_PDOWN** (6<<3)

#define **PHYCFGR_OPMDC_NA** (5<<3)

#define **PHYCFGR_OPMDC_100FA** (4<<3)

#define **PHYCFGR_OPMDC_100F** (3<<3)

#define **PHYCFGR_OPMDC_100H** (2<<3)

#define **PHYCFGR_OPMDC_10F** (1<<3)

#define **PHYCFGR_OPMDC_10H** (0<<3)

#define **PHYCFGR_DPX_FULL** (1<<2)

#define **PHYCFGR_DPX_HALF** (0<<2)

#define **PHYCFGR_SPD_100** (1<<1)

#define **PHYCFGR_SPD_10** (0<<1)

#define **PHYCFGR_LNK_ON** (1<<0)

#define **PHYCFGR_LNK_OFF** (0<<0)

#define **IM_IR7** 0x80
IP Conflict Interrupt Mask. More...

#define **IM_IR6** 0x40
Destination unreachable Interrupt Mask. More...

#define **IM_IR5** 0x20
PPPoE Close Interrupt Mask. More...

#define **IM_IR4** 0x10
Magic Packet Interrupt Mask. More...

#define **Sn_MR_MULTI** 0x80
Support UDP Multicasting. More...

#define **Sn_MR_BCASTB** 0x40
Broadcast block in UDP Multicasting. More...

#define **Sn_MR_ND** 0x20
No Delayed Ack(TCP), Multicast flag. More...

#define **Sn_MR_UCASTB** 0x10
Unicast Block in UDP Multicasting. More...

#define **Sn_MR_MACRAW** 0x04
MAC LAYER RAW SOCK. More...

#define **Sn_MR_UDP** 0x02

UDP. More...

#define **Sn_MR_TCP** 0x01
TCP. More...

#define **Sn_MR_CLOSE** 0x00
Unused socket. More...

#define **Sn_MR_MFEN** **Sn_MR_MULTI**
MAC filter enable in **Sn_MR_MACRAW** mode. More...

#define **Sn_MR_MMB** **Sn_MR_ND**
Multicast Blocking in **Sn_MR_MACRAW** mode. More...

#define **Sn_MR_MIP6B** **Sn_MR_UCASTB**
IPv6 packet Blocking in **Sn_MR_MACRAW** mode. More...

#define **Sn_MR_MC** **Sn_MR_ND**
IGMP version used in UDP mulitcasting. More...

#define **SOCK_STREAM** **Sn_MR_TCP**
For Berkeley Socket API. More...

#define **SOCK_DGRAM** **Sn_MR_UDP**
For Berkeley Socket API. More...

#define **Sn_CR_OPEN** 0x01
Initialize or open socket. More...

#define **Sn_CR_LISTEN** 0x02
Wait connection request in TCP mode(Server mode) More...

#define **Sn_CR_CONNECT** 0x04
Send connection request in TCP mode(Client mode) More...

#define **Sn_CR_DISCON** 0x08

Send closing request in TCP mode. More...

#define **Sn_CR_CLOSE** 0x10
Close socket. More...

#define **Sn_CR_SEND** 0x20
Update TX buffer pointer and send data. More...

#define **Sn_CR_SEND_MAC** 0x21
Send data with MAC address, so without ARP process. More..

#define **Sn_CR_SEND_KEEP** 0x22
Send keep alive message. More...

#define **Sn_CR_RECV** 0x40
Update RX buffer pointer and receive data. More...

#define **Sn_IR_SENDOK** 0x10
SEND_OK Interrupt. More...

#define **Sn_IR_TIMEOUT** 0x08
TIMEOUT Interrupt. More...

#define **Sn_IR_RECV** 0x04
RECV Interrupt. More...

#define **Sn_IR_DISCON** 0x02
DISCON Interrupt. More...

#define **Sn_IR_CON** 0x01
CON Interrupt. More...

#define **SOCK_CLOSED** 0x00
Closed. More...

#define **SOCK_INIT** 0x13

Initiate state. More...

#define **SOCK_LISTEN** 0x14
Listen state. More...

#define **SOCK_SYNSENT** 0x15
Connection state. More...

#define **SOCK_SYNRECV** 0x16
Connection state. More...

#define **SOCK_ESTABLISHED** 0x17
Success to connect. More...

#define **SOCK_FIN_WAIT** 0x18
Closing state. More...

#define **SOCK_CLOSING** 0x1A
Closing state. More...

#define **SOCK_TIME_WAIT** 0x1B
Closing state. More...

#define **SOCK_CLOSE_WAIT** 0x1C
Closing state. More...

#define **SOCK_LAST_ACK** 0x1D
Closing state. More...

#define **SOCK_UDP** 0x22
UDP socket. More...

#define **SOCK_MACRAW** 0x42
MAC raw mode socket. More...

#define **IPPROTO_IP** 0

#define **IPPROTO_ICMP** 1

#define **IPPROTO_IGMP** 2

#define **IPPROTO_GGP** 3

#define **IPPROTO_TCP** 6

#define **IPPROTO_PUP** 12

#define **IPPROTO_UDP** 17

#define **IPPROTO_IDP** 22

#define **IPPROTO_ND** 77

#define **IPPROTO_RAW** 255

#define **WIZCHIP_CRITICAL_ENTER**() WIZCHIP.CRIS._enter()
Enter a critical section. More...

#define **WIZCHIP_CRITICAL_EXIT**() WIZCHIP.CRIS._exit()
Exit a critical section. More...

#define **setMR**(mr) **WIZCHIP_WRITE**(**MR**,mr)
Set Mode Register. More...

#define **getMR**() **WIZCHIP_READ**(**MR**)
Get Mode Register. More...

#define **setGAR**(gar) **WIZCHIP_WRITE_BUF**(**GAR**,gar,4)
Set gateway IP address. More...

#define **getGAR**(gar) **WIZCHIP_READ_BUF**(**GAR**,gar,4)
Get gateway IP address. More...

#define **setSUBR**(subr) **WIZCHIP_WRITE_BUF**(**SUBR**, subr,4)
Set subnet mask address. More...

#define **getSUBR**(subr) **WIZCHIP_READ_BUF**(**SUBR**, subr, 4)
Get subnet mask address. More...

#define **setSHAR**(shar) **WIZCHIP_WRITE_BUF**(**SHAR**, shar, 6)
Set local MAC address. More...

#define **getSHAR**(shar) **WIZCHIP_READ_BUF**(**SHAR**, shar, 6)
Get local MAC address. More...

#define **setSIPR**(sipr) **WIZCHIP_WRITE_BUF**(**SIPR**, sipr, 4)
Set local IP address. More...

#define **getSIPR**(sipr) **WIZCHIP_READ_BUF**(**SIPR**, sipr, 4)
Get local IP address. More...

#define **setINTLEVEL**(intlevel)
Set INTLEVEL register. More...

#define **getINTLEVEL**() (((uint16_t)**WIZCHIP_READ**(**INTLEVEL**) << **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**INTLEVEL**,1)))
Get INTLEVEL register. More...

#define **setIR**(ir) **WIZCHIP_WRITE**(**IR**, (ir & 0xF0))
Set **IR** register. More...

#define **getIR**() (**WIZCHIP_READ**(**IR**) & 0xF0)
Get **IR** register. More...

#define **setIMR**(imr) **WIZCHIP_WRITE**(**_IMR_**, imr)
Set *IMR* register. More...

#define **getIMR**() **WIZCHIP_READ**(**_IMR_**)
Get *IMR* register. More...

#define **setSIR**(sir) **WIZCHIP_WRITE**(**SIR**, sir)
Set **SIR** register. More...

#define **getSIR**() **WIZCHIP_READ**(**SIR**)
Get **SIR** register. More...

#define **setSIMR**(simr) **WIZCHIP_WRITE**(**SIMR**, simr)
Set **SIMR** register. More...

#define **getSIMR**() **WIZCHIP_READ**(**SIMR**)
Get **SIMR** register. More...

#define **setRTR**(rtr)
Set *RTR* register. More...

#define **getRTR**() (((uint16_t)**WIZCHIP_READ**(**_RTR_**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**_RTR_**,1)))
Get *RTR* register. More...

#define **setRCR**(rcr) **WIZCHIP_WRITE**(**_RCR_**, rcr)
Set *RCR* register. More...

#define **getRCR**() **WIZCHIP_READ**(**_RCR_**)
Get *RCR* register. More...

#define **setPTIMER**(ptimer) **WIZCHIP_WRITE**(**PTIMER**, ptimer)
Set **PTIMER** register. More...

#define **getPTIMER**() **WIZCHIP_READ**(**PTIMER**)
Get **PTIMER** register. More...

#define **setPMAGIC**(pmagic) **WIZCHIP_WRITE**(**PMAGIC**, pmagic)
Set **PMAGIC** register. More...

#define **getPMAGIC**() **WIZCHIP_READ**(**PMAGIC**)

Get **PMAGIC** register. More...

#define **setPHAR**(phar) **WIZCHIP_WRITE_BUF**(**PHAR**, phar, 6)
Set **PHAR** address. More...

#define **getPHAR**(phar) **WIZCHIP_READ_BUF**(**PHAR**, phar, 6)
Get **PHAR** address. More...

#define **setPSID**(psid)
Set **PSID** register. More...

#define **getPSID**() (((uint16_t)**WIZCHIP_READ**(**PSID**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PSID**,1)))
Get **PSID** register. More...

#define **setPMRU**(pmru)
Set **PMRU** register. More...

#define **getPMRU**() (((uint16_t)**WIZCHIP_READ**(**PMRU**) << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**PMRU**,1)))
Get **PMRU** register. More...

#define **getUIPR**(uipr) **WIZCHIP_READ_BUF**(**UIPR**,uipr,4)
Get unreachable IP address. More...

#define **getUPORTR**() (((uint16_t)**WIZCHIP_READ**(**UPORTR**) << 8) **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**UPORTR**,1)))
Get **UPORTR** register. More...

#define **setPHYCFGR**(phycfgr) **WIZCHIP_WRITE**(**PHYCFGR**, phycfg
Set **PHYCFGR** register. More...

#define **getPHYCFGR**() **WIZCHIP_READ**(**PHYCFGR**)
Get **PHYCFGR** register. More...

#define **getVERSIONR**() **WIZCHIP_READ**(**VERSIONR**)

Get **VERSIONR** register. More...

#define **setSn_MR**(sn, mr) **WIZCHIP_WRITE**(**Sn_MR**(sn),mr)
Set **Sn_MR** register. More...

#define **getSn_MR**(sn) **WIZCHIP_READ**(**Sn_MR**(sn))
Get **Sn_MR** register. More...

#define **setSn_CR**(sn, cr) **WIZCHIP_WRITE**(**Sn_CR**(sn), cr)
Set **Sn_CR** register. More...

#define **getSn_CR**(sn) **WIZCHIP_READ**(**Sn_CR**(sn))
Get **Sn_CR** register. More...

#define **setSn_IR**(sn, ir) **WIZCHIP_WRITE**(**Sn_IR**(sn), (ir & 0x1F))
Set **Sn_IR** register. More...

#define **getSn_IR**(sn) (**WIZCHIP_READ**(**Sn_IR**(sn)) & 0x1F)
Get **Sn_IR** register. More...

#define **setSn_IMR**(sn, imr) **WIZCHIP_WRITE**(**Sn_IMR**(sn), (imr & 0x
Set **Sn_IMR** register. More...

#define **getSn_IMR**(sn) (**WIZCHIP_READ**(**Sn_IMR**(sn)) & 0x1F)
Get **Sn_IMR** register. More...

#define **getSn_SR**(sn) **WIZCHIP_READ**(**Sn_SR**(sn))
Get **Sn_SR** register. More...

#define **setSn_PORT**(sn, port)
Set **Sn_PORT** register. More...

#define **getSn_PORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_PORT**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_PORT**(sn)
Get **Sn_PORT** register. More...

#define **setSn_DHAR**(sn, dhar) **WIZCHIP_WRITE_BUF**(**Sn_DHAR**(s
dhar, 6)
Set **Sn_DHAR** register. More...

#define **getSn_DHAR**(sn, dhar) **WIZCHIP_READ_BUF**(**Sn_DHAR**(s
dhar, 6)
Get **Sn_MR** register. More...

#define **setSn_DIPR**(sn, dipr) **WIZCHIP_WRITE_BUF**(**Sn_DIPR**(sn),
4)
Set **Sn_DIPR** register. More...

#define **getSn_DIPR**(sn, dipr) **WIZCHIP_READ_BUF**(**Sn_DIPR**(sn),
4)
Get **Sn_DIPR** register. More...

#define **setSn_DPORT**(sn, dport)
Set **Sn_DPORT** register. More...

#define **getSn_DPORT**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_DPORT**
<< 8) +
WIZCHIP_READ(**WIZCHIP_OFFSET_INC**(**Sn_DPORT**(sn),1)
Get **Sn_DPORT** register. More...

#define **setSn_MSSR**(sn, mss)
Set **Sn_MSSR** register. More...

#define **getSn_MSSR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_MSSR**(sr
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_MSSR**(sn)
Get **Sn_MSSR** register. More...

#define **setSn_TOS**(sn, tos) **WIZCHIP_WRITE**(**Sn_TOS**(sn), tos)
Set **Sn_TOS** register. More...

#define **getSn_TOS**(sn) **WIZCHIP_READ**(**Sn_TOS**(sn))
Get **Sn_TOS** register. More...

#define	**setSn_TTL**(sn, ttl) **WIZCHIP_WRITE**(**Sn_TTL**(sn), ttl) Set **Sn_TTL** register. More...
#define	**getSn_TTL**(sn) **WIZCHIP_READ**(**Sn_TTL**(sn)) Get **Sn_TTL** register. More...
#define	**setSn_RXBUF_SIZE**(sn, rxbufsize) **WIZCHIP_WRITE**(**Sn_RXBUF_SIZE**(sn),rxbufsize Set **Sn_RXBUF_SIZE** register. More...
#define	**getSn_RXBUF_SIZE**(sn) **WIZCHIP_READ**(**Sn_RXBUF_SIZ** Get **Sn_RXBUF_SIZE** register. More...
#define	**setSn_TXBUF_SIZE**(sn, txbufsize) **WIZCHIP_WRITE**(**Sn_TXBUF_SIZE**(sn), txbufsize Set **Sn_TXBUF_SIZE** register. More...
#define	**getSn_TXBUF_SIZE**(sn) **WIZCHIP_READ**(**Sn_TXBUF_SIZE** Get **Sn_TXBUF_SIZE** register. More...
#define	**getSn_TX_RD**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_RD**(<< 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_RD**(sn),1)) Get **Sn_TX_RD** register. More...
#define	**setSn_TX_WR**(sn, txwr) Set **Sn_TX_WR** register. More...
#define	**getSn_TX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_TX_WR** << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_TX_WR**(sn),1) Get **Sn_TX_WR** register. More...
#define	**setSn_RX_RD**(sn, rxrd) Set **Sn_RX_RD** register. More...
	getSn_RX_RD(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_RD**(

#define << 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_RX_RD**(sn),1)
Get **Sn_RX_RD** register. More...

#define **getSn_RX_WR**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_RX_WR**
<< 8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_RX_WR**(sn),1)
Get **Sn_RX_WR** register. More...

#define **setSn_FRAG**(sn, frag)
Set **Sn_FRAG** register. More...

#define **getSn_FRAG**(sn) (((uint16_t)**WIZCHIP_READ**(**Sn_FRAG**(sn
8) + **WIZCHIP_READ**(**WIZCHIP_OFFSET_INC**(**Sn_FRAG**(sn)
Get **Sn_FRAG** register. More...

#define **setSn_KPALVTR**(sn, kpalvt) **WIZCHIP_WRITE**(**Sn_KPALVTR**(sn), kpalvt)
Set **Sn_KPALVTR** register. More...

#define **getSn_KPALVTR**(sn) **WIZCHIP_READ**(**Sn_KPALVTR**(sn))
Get **Sn_KPALVTR** register. More...

#define **getSn_RxMAX**(sn) (((uint16_t)**getSn_RXBUF_SIZE**(sn)) <<
Socket_register_access_function. More...

#define **getSn_TxMAX**(sn) (((uint16_t)**getSn_TXBUF_SIZE**(sn)) <<
Socket_register_access_function. More...

Functions

uint8_t	**WIZCHIP_READ** (uint32_t AddrSel) It reads 1 byte value from a register. More...
void	**WIZCHIP_WRITE** (uint32_t AddrSel, uint8_t wb) It writes 1 byte value to a register. More...
void	**WIZCHIP_READ_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It reads sequence data from registers. More...
void	**WIZCHIP_WRITE_BUF** (uint32_t AddrSel, uint8_t *pBuf, uint16_t len) It writes sequence data to registers. More...
uint16_t	**getSn_TX_FSR** (uint8_t sn) Get **Sn_TX_FSR** register. More...
uint16_t	**getSn_RX_RSR** (uint8_t sn) Get **Sn_RX_RSR** register. More...
void	**wiz_send_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to internal TX memory. More...
void	**wiz_recv_data** (uint8_t sn, uint8_t *wizdata, uint16_t len) It copies data to your buffer from internal RX memory. More...
void	**wiz_recv_ignore** (uint8_t sn, uint16_t len) It discard the received data in RX memory. More...

Detailed Description

W5500 HAL Header File.

Version
1.0.0

Date
2013/10/21

Revision history
<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2013/10/21> 1st Release

Author
MidnightCow

Copyright

- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **w5500.h**.

Macro Definition Documentation

#define _W5500_IO_BASE_ 0x00000000

Definition at line **58** of file **w5500.h**.

#define _W5500_SPI_READ_ (0x00 << 2)

Definition at line **60** of file **w5500.h**.

#define _W5500_SPI_WRITE_ (0x01 << 2)

Definition at line **61** of file **w5500.h**.

#define WIZCHIP_CREG_BLOCK 0x00

Definition at line **63** of file **w5500.h**.

#define WIZCHIP_SREG_BLOCK (N) (1+4*N)

Definition at line **64** of file **w5500.h**.

#define WIZCHIP_TXBUF_BLOCK (N) (2+4*N)

Definition at line **65** of file **w5500.h**.

#define WIZCHIP_RXBUF_BLOCK (N) (3+4*N)

Definition at line **66** of file **w5500.h**.

```
#define WIZCHIP_OFFSET_INC (  ADDR,
                              N
                           )    (ADDR + (N<<8))
```

Definition at line **68** of file **w5500.h**.

```
#define IINCHIP_READ (  ADDR )   WIZCHIP_READ(ADDR)
```

The defined for legacy chip driver.

Definition at line **74** of file **w5500.h**.

```
#define IINCHIP_WRITE (  ADDR,
                         VAL
                      )    WIZCHIP_WRITE(ADDR,VAL)
```

The defined for legacy chip driver.

Definition at line **75** of file **w5500.h**.

```
#define
IINCHIP_READ_BUF   (  ADDR,
                      BUF,
                      LEN
                   )    WIZCHIP_READ_BUF(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **76** of file **w5500.h**.

```
#define
IINCHIP_WRITE_BUF       (  ADDR,
                           BUF,
                           LEN
                        )    WIZCHIP_WRITE(ADDR,BUF,LEN)
```

The defined for legacy chip driver.

Definition at line **77** of file **w5500.h**.

#define MR_RST 0x80

Reset.

If this bit is All internal registers will be initialized. It will be automatically cleared as after S/W reset.

Definition at line **697** of file **w5500.h**.

#define MR_WOL 0x20

Wake on LAN.

0 : Disable WOL mode
1 : Enable WOL mode
If WOL mode is enabled and the received magic packet over UDP has been normally processed, the Interrupt PIN (INTn) asserts to low. When using WOL mode, the UDP Socket should be opened with any source port number. (Refer to Socket n Mode Register (**Sn_MR**) for opening Socket.)

Note

The magic packet over UDP supported by W5500 consists of 6 bytes synchronization stream (xFFFFFFFFFFFF and 16 times Target MAC address stream in UDP payload. The options such like password are ignored. You can use any UDP source port number for WOL mode.

Definition at line **708** of file **w5500.h**.

#define MR_PB 0x10

Ping block.

0 : Disable Ping block
1 : Enable Ping block
If the bit is it blocks the response to a ping request.

Definition at line **716** of file **w5500.h**.

#define MR_PPPOE 0x08

Enable PPPoE.

0 : DisablePPPoE mode
1 : EnablePPPoE mode
If you use ADSL, this bit should be

Definition at line **724** of file **w5500.h**.

#define MR_FARP 0x02

Enable UDP_FORCE_ARP CHECHK.

0 : Disable Force ARP mode
1 : Enable Force ARP mode
In Force ARP mode, It forces on sending ARP Request whenever data is sent.

Definition at line **732** of file **w5500.h**.

#define IR_CONFLICT 0x80

Check IP conflict.

Bit is set as when own source IP address is same with the sender IP address in the received ARP request.

Definition at line **739** of file **w5500.h**.

#define IR_UNREACH 0x40

Get the destination unreachable message in UDP sending.

When receiving the ICMP (Destination port unreachable) packet, this bit is set as When this bit is Destination Information such as IP address and Port number may be checked with the corresponding **UIPR** & **UPORTR**.

Definition at line **746** of file **w5500.h**.

#define IR_PPPoE 0x20

Get the PPPoE close message.

When PPPoE is disconnected during PPPoE mode, this bit is set.

Definition at line **752** of file **w5500.h**.

#define IR_MP 0x10

Get the magic packet interrupt.

When WOL mode is enabled and receives the magic packet over UDP, this bit is set.

Definition at line **758** of file **w5500.h**.

#define PHYCFGR_RST ~(1<<7)

Definition at line **762** of file **w5500.h**.

Referenced by **wizphy_reset()**.

#define PHYCFGR_OPMD (1<<6)

Definition at line **763** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

#define PHYCFGR_OPMDC_ALLA (7<<3)

Definition at line **764** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

#define PHYCFGR_OPMDC_PDOWN (6<<3)

Definition at line **765** of file **w5500.h**.

Referenced by **wizphy_getphypmode()**, and **wizphy_setphypmode()**.

#define PHYCFGR_OPMDC_NA (5<<3)

Definition at line **766** of file **w5500.h**.

#define PHYCFGR_OPMDC_100FA (4<<3)

Definition at line **767** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**.

#define PHYCFGR_OPMDC_100F (3<<3)

Definition at line **768** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

#define PHYCFGR_OPMDC_100H (2<<3)

Definition at line **769** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

#define PHYCFGR_OPMDC_10F (1<<3)

Definition at line **770** of file **w5500.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

#define PHYCFGR_OPMDC_10H (0<<3)

Definition at line **771** of file **w5500.h**.

Referenced by **wizphy_setphyconf()**.

#define PHYCFGR_DPX_FULL (1<<2)

Definition at line **772** of file **w5500.h**.

Referenced by **wizphy_getphystat()**.

#define PHYCFGR_DPX_HALF (0<<2)

Definition at line **773** of file **w5500.h**.

#define PHYCFGR_SPD_100 (1<<1)

Definition at line **774** of file **w5500.h**.

Referenced by **wizphy_getphystat()**.

#define PHYCFGR_SPD_10 (0<<1)

Definition at line **775** of file **w5500.h**.

#define PHYCFGR_LNK_ON (1<<0)

Definition at line **776** of file **w5500.h**.

Referenced by **wizphy_getphylink()**.

#define PHYCFGR_LNK_OFF (0<<0)

Definition at line **777** of file **w5500.h**.

#define IM_IR7 0x80

IP Conflict Interrupt Mask.

0: Disable IP Conflict Interrupt
1: Enable IP Conflict Interrupt

Definition at line **785** of file **w5500.h**.

#define IM_IR6 0x40

Destination unreachable Interrupt Mask.

0: Disable Destination unreachable Interrupt
1: Enable Destination unreachable Interrupt

Definition at line **792** of file **w5500.h**.

#define IM_IR5 0x20

PPPoE Close Interrupt Mask.

0: Disable PPPoE Close Interrupt
1: Enable PPPoE Close Interrupt

Definition at line **799** of file **w5500.h**.

#define IM_IR4 0x10

Magic Packet Interrupt Mask.

0: Disable Magic Packet Interrupt
1: Enable Magic Packet Interrupt

Definition at line **806** of file **w5500.h**.

#define Sn_MR_MULTI 0x80

Support UDP Multicasting.

0 : disable Multicasting
1 : enable Multicasting

This bit is applied only during UDP mode(P[3:0] = 010.
To use multicasting, **Sn_DIPR** & **Sn_DPORT** should be respectively configured with the multicast group IP address & port number before Socket n is opened by OPEN command of **Sn_CR**.

Definition at line **817** of file **w5500.h**.

#define Sn_MR_BCASTB 0x40

Broadcast block in UDP Multicasting.

0 : disable Broadcast Blocking
1 : enable Broadcast Blocking
This bit blocks to receive broadcasting packet during UDP mode(P[3:0] = 010. In addition, This bit does when MACRAW mode(P[3:0] = 100

Definition at line **826** of file **w5500.h**.

#define Sn_MR_ND 0x20

No Delayed Ack(TCP), Multicast flag.

0 : Disable No Delayed ACK option
1 : Enable No Delayed ACK option
This bit is applied only during TCP mode (P[3:0] = 001.
When this bit is It sends the ACK packet without delay as soon as a Data packet is received from a peer.
When this bit is It sends the ACK packet after waiting for the timeout time configured by *RTR*.

Definition at line **836** of file **w5500.h**.

#define Sn_MR_UCASTB 0x10

Unicast Block in UDP Multicasting.

0 : disable Unicast Blocking
1 : enable Unicast Blocking
This bit blocks receiving the unicast packet during UDP mode(P[3:0] = 010 and MULTI =

Definition at line **844** of file **w5500.h**.

#define Sn_MR_MACRAW 0x04

MAC LAYER RAW SOCK.

This configures the protocol mode of Socket n.

Note
MACRAW mode should be only used in Socket 0.

Definition at line **851** of file **w5500.h**.

#define Sn_MR_UDP 0x02

UDP.

This configures the protocol mode of Socket n.

Definition at line **859** of file **w5500.h**.

#define Sn_MR_TCP 0x01

TCP.

This configures the protocol mode of Socket n.

Definition at line **865** of file **w5500.h**.

#define Sn_MR_CLOSE 0x00

Unused socket.

This configures the protocol mode of Socket n.

Definition at line **871** of file **w5500.h**.

#define Sn_MR_MFEN Sn_MR_MULTI

MAC filter enable in **Sn_MR_MACRAW** mode.

0 : disable MAC Filtering
1 : enable MAC Filtering
This bit is applied only during MACRAW mode(P[3:0] = 100.
When set as W5500 can only receive broadcasting packet or packet sent to itself. When this bit is W5500 can receive all packets on Ethernet. If user wants to implement Hybrid TCP/IP stack, it is recommended that this bit is set as for reducing host overhead to process the all received packets.

Definition at line **884** of file **w5500.h**.

#define Sn_MR_MMB Sn_MR_ND

Multicast Blocking in **Sn_MR_MACRAW** mode.

0 : using IGMP version 2
1 : using IGMP version 1
This bit is applied only during UDP mode(P[3:0] = 010 and MULTI =
It configures the version for IGMP messages (Join/Leave/Report).

Definition at line **893** of file **w5500.h**.

#define Sn_MR_MIP6B Sn_MR_UCASTB

IPv6 packet Blocking in **Sn_MR_MACRAW** mode.

0 : disable IPv6 Blocking
1 : enable IPv6 Blocking
This bit is applied only during MACRAW mode (P[3:0] = 100. It blocks to receiving the IPv6 packet.

Definition at line **901** of file **w5500.h**.

#define Sn_MR_MC Sn_MR_ND

IGMP version used in UDP mulitcasting.

0 : disable Multicast Blocking
1 : enable Multicast Blocking
This bit is applied only when MACRAW mode(P[3:0] = 100. It blocks to receive the packet with multicast MAC address.

Definition at line **910** of file **w5500.h**.

#define SOCK_STREAM Sn_MR_TCP

For Berkeley Socket API.

Definition at line **916** of file **w5500.h**.

#define SOCK_DGRAM Sn_MR_UDP

For Berkeley Socket API.

Definition at line **921** of file **w5500.h**.

#define Sn_CR_OPEN 0x01

Initialize or open socket.

Socket n is initialized and opened according to the protocol selected

in **Sn_MR(P3:P0)**. The table below shows the value of **Sn_SR** corresponding to **Sn_MR**.

Sn_MR (P[3:0])	**Sn_SR**
Sn_MR_CLOSE (000)	
Sn_MR_TCP (001)	SOCK_INIT (0x13)
Sn_MR_UDP (010)	SOCK_UDP (0x22)
S0_MR_MACRAW (100)	SOCK_MACRAW (0x02)

Definition at line **937** of file **w5500.h**.

#define Sn_CR_LISTEN 0x02

Wait connection request in TCP mode(Server mode)

This is valid only in TCP mode (**Sn_MR(P3:P0)** = **Sn_MR_TCP**). In this mode, Socket n operates as a TCP serverand waits for connection-request (SYN packet) from any TCP client The **Sn_SR** changes the state from **SOCK_INIT** to SOCKET_LISTEN. When a TCP clientconnection request is successfully established, the **Sn_SR** changes from SOCK_LISTEN to SOCK_ESTABLISHED and the **Sn_IR(0)** becomes But when a TCP clientconnection request is failed, **Sn_IR(3)** becomes and the status of **Sn_SR** changes to SOCK_CLOSED.

Definition at line **948** of file **w5500.h**.

#define Sn_CR_CONNECT 0x04

Send connection request in TCP mode(Client mode)

To connect, a connect-request (SYN packet) is sent to **TCP server**configured by **Sn_DIPR** & **Sn_DPORT(destination address & port)**. If the connect-request is successful, the **Sn_SR** is changed to **SOCK_ESTABLISHED** and the **Sn_IR(0)** becomes

The connect-request fails in the following three cases.

1. When a **ARPTO** occurs (**Sn_IR**[3] =) because destination hardware address is not acquired through the ARP-process.
2. When a **SYN/ACK** packet is not received and **TCPTO** (**Sn_IR(3)** =)
3. When a **RST** packet is received instead of a **SYN/ACK** packet. In these cases, **Sn_SR** is changed to **SOCK_CLOSED**.

Note

This is valid only in TCP mode and operates when Socket n acts as **TCP client**

Definition at line **960** of file **w5500.h**.

#define Sn_CR_DISCON 0x08

Send closing request in TCP mode.

Regardless of **TCP server**or **TCP client** the DISCON command processes the disconnect-process (b>Active closeor **Passive close**.

Active close

it transmits disconnect-request(FIN packet) to the connected peer

Passive close

When FIN packet is received from peer, a FIN packet is replied back to the peer.

When the disconnect-process is successful (that is, FIN/ACK packet is received successfully), **Sn_SR** is changed to **SOCK_CLOSED**. Otherwise, TCPTO occurs (**Sn_IR(3)**='1') and then **Sn_SR** is changed to **SOCK_CLOSED**.

Note

Valid only in TCP mode.

Definition at line **973** of file **w5500.h**.

#define Sn_CR_CLOSE 0x10

Close socket.

Sn_SR is changed to **SOCK_CLOSED**.

Definition at line **979** of file **w5500.h**.

#define Sn_CR_SEND 0x20

Update TX buffer pointer and send data.

SEND transmits all the data in the Socket n TX buffer.
For more details, please refer to Socket n TX Free Size Register (**Sn_TX_FSR**), Socket n, TX Write Pointer Register(**Sn_TX_WR**), and Socket n TX Read Pointer Register(**Sn_TX_RD**).

Definition at line **987** of file **w5500.h**.

#define Sn_CR_SEND_MAC 0x21

Send data with MAC address, so without ARP process.

The basic operation is same as SEND.
Normally SEND transmits data after destination hardware address is acquired by the automatic ARP-process(Address Resolution Protocol).
But SEND_MAC transmits data without the automatic ARP-process.
In this case, the destination hardware address is acquired from **Sn_DHAR** configured by host, instead of APR-process.

Note
Valid only in UDP mode.

Definition at line **997** of file **w5500.h**.

#define Sn_CR_SEND_KEEP 0x22

Send keep alive message.

It checks the connection status by sending 1byte keep-alive packet. If the peer can not respond to the keep-alive packet during timeout time, the connection is terminated and the timeout interrupt will occur.

Note
Valid only in TCP mode.

Definition at line **1005** of file **w5500.h**.

#define Sn_CR_RECV 0x40

Update RX buffer pointer and receive data.

RECV completes the processing of the received data in Socket n RX Buffer by using a RX read pointer register (**Sn_RX_RD**).
For more details, refer to Socket n RX Received Size Register (**Sn_RX_RSR**), Socket n RX Write Pointer Register (**Sn_RX_WR**), and Socket n RX Read Pointer Register (**Sn_RX_RD**).

Definition at line **1013** of file **w5500.h**.

#define Sn_IR_SENDOK 0x10

SEND_OK Interrupt.

This is issued when SEND command is completed.

Definition at line **1020** of file **w5500.h**.

#define Sn_IR_TIMEOUT 0x08

TIMEOUT Interrupt.

This is issued when ARPTO or TCPTO occurs.

Definition at line **1026** of file **w5500.h**.

#define Sn_IR_RECV 0x04

RECV Interrupt.

This is issued whenever data is received from a peer.

Definition at line **1032** of file **w5500.h**.

#define Sn_IR_DISCON 0x02

DISCON Interrupt.

This is issued when FIN or FIN/ACK packet is received from a peer.

Definition at line **1038** of file **w5500.h**.

#define Sn_IR_CON 0x01

CON Interrupt.

This is issued one time when the connection with peer is successful and then **Sn_SR** is changed to **SOCK_ESTABLISHED**.

Definition at line **1044** of file **w5500.h**.

#define SOCK_CLOSED 0x00

Closed.

This indicates that Socket n is released.
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to **SOCK_CLOSED** regardless of previous status.

Definition at line **1052** of file **w5500.h**.

#define SOCK_INIT 0x13

Initiate state.

This indicates Socket n is opened with TCP mode.
It is changed to **SOCK_INIT** when **Sn_MR(P[3:0])** = 001 and OPEN command is ordered.
After **SOCK_INIT**, user can use LISTEN /CONNECT command.

Definition at line **1060** of file **w5500.h**.

#define SOCK_LISTEN 0x14

Listen state.

This indicates Socket n is operating as **TCP server**mode and waiting for connection-request (SYN packet) from a peer **TCP client**.
It will change to SOCK_ESTALBLISHED when the connection-request is successfully accepted.
Otherwise it will change to **SOCK_CLOSED** after TCPTO **Sn_IR(TIMEOUT)** = '1') is occurred.

Definition at line **1068** of file **w5500.h**.

#define SOCK_SYNSENT 0x15

Connection state.

This indicates Socket n sent the connect-request packet (SYN

packet) to a peer.
It is temporarily shown when **Sn_SR** is changed from **SOCK_INIT** to **SOCK_ESTABLISHED** by CONNECT command.
If connect-accept(SYN/ACK packet) is received from the peer at SOCK_SYNSENT, it changes to **SOCK_ESTABLISHED**.
Otherwise, it changes to **SOCK_CLOSED** after TCPTO (**Sn_IR**[TIMEOUT] = '1') is occurred.

Definition at line **1077** of file **w5500.h**.

#define SOCK_SYNRECV 0x16

Connection state.

It indicates Socket n successfully received the connect-request packet (SYN packet) from a peer.
If socket n sends the response (SYN/ACK packet) to the peer successfully, it changes to **SOCK_ESTABLISHED**.
If not, it changes to **SOCK_CLOSED** after timeout (**Sn_IR**[TIMEOUT] = '1') is occurred.

Definition at line **1085** of file **w5500.h**.

#define SOCK_ESTABLISHED 0x17

Success to connect.

This indicates the status of the connection of Socket n.
It changes to **SOCK_ESTABLISHED** when the **TCP SERVER**processed the SYN packet from the **TCP CLIENT**during **SOCK_LISTEN**, or when the CONNECT command is successful.
During **SOCK_ESTABLISHED**, DATA packet can be transferred using SEND or RECV command.

Definition at line **1094** of file **w5500.h**.

#define SOCK_FIN_WAIT 0x18

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1102** of file **w5500.h**.

#define SOCK_CLOSING 0x1A

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1110** of file **w5500.h**.

#define SOCK_TIME_WAIT 0x1B

Closing state.

These indicate Socket n is closing.
These are shown in disconnect-process such as active-close and passive-close.
When Disconnect-process is successfully completed, or when timeout occurs, these change to **SOCK_CLOSED**.

Definition at line **1118** of file **w5500.h**.

#define SOCK_CLOSE_WAIT 0x1C

Closing state.

This indicates Socket n received the disconnect-request (FIN packet) from the connected peer.
This is half-closing status, and data can be transferred.
For full-closing, DISCON command is used. But For just-closing, CLOSE command is used.

Definition at line **1126** of file **w5500.h**.

#define SOCK_LAST_ACK 0x1D

Closing state.

This indicates Socket n is waiting for the response (FIN/ACK packet) to the disconnect-request (FIN packet) by passive-close.
It changes to **SOCK_CLOSED** when Socket n received the response successfully, or when timeout(**Sn_IR**[TIMEOUT] = '1') is occurred.

Definition at line **1133** of file **w5500.h**.

#define SOCK_UDP 0x22

UDP socket.

This indicates Socket n is opened in UDP mode(**Sn_MR(P[3:0])** = '010').
It changes to SOCK_UDP when **Sn_MR(P[3:0])** = '010' and **Sn_CR_OPEN** command is ordered.
Unlike TCP mode, data can be transfered without the connection-process.

Definition at line **1141** of file **w5500.h**.

#define SOCK_MACRAW 0x42

MAC raw mode socket.

This indicates Socket 0 is opened in MACRAW mode (S0_MR(P[3:0]) = 100and is valid only in Socket 0.
It changes to SOCK_MACRAW when S0_MR(P[3:0] = 100and OPEN command is ordered.
Like UDP mode socket, MACRAW mode Socket 0 can transfer a MAC packet (Ethernet frame) without the connection-process.

Definition at line **1151** of file **w5500.h**.

#define IPPROTO_IP 0

Definition at line **1156** of file **w5500.h**.

#define IPPROTO_ICMP 1

Definition at line **1157** of file **w5500.h**.

#define IPPROTO_IGMP 2

Definition at line **1158** of file **w5500.h**.

#define IPPROTO_GGP 3

Definition at line **1159** of file **w5500.h**.

#define IPPROTO_TCP 6

Definition at line **1160** of file **w5500.h**.

#define IPPROTO_PUP 12

Definition at line **1161** of file **w5500.h**.

#define IPPROTO_UDP 17

Definition at line **1162** of file **w5500.h**.

#define IPPROTO_IDP 22

Definition at line **1163** of file **w5500.h**.

#define IPPROTO_ND 77

Definition at line **1164** of file **w5500.h**.

#define IPPROTO_RAW 255

Definition at line **1165** of file **w5500.h**.

#define WIZCHIP_CRITICAL_ENTER ( ) WIZCHIP.CRIS._enter()

Enter a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also
WIZCHIP_READ(), **WIZCHIP_WRITE()**,

WIZCHIP_READ_BUF(), **WIZCHIP_WRITE_BUF()**
WIZCHIP_CRITICAL_EXIT()

Definition at line **1179** of file **w5500.h**.

#define WIZCHIP_CRITICAL_EXIT ( ) WIZCHIP.CRIS._exit()

Exit a critical section.

It is provided to protect your shared code which are executed without distribution.

In non-OS environment, It can be just implemented by disabling whole interrupt.
In OS environment, You can replace it to critical section api supported by OS.

See also
WIZCHIP_READ(), **WIZCHIP_WRITE()**,
WIZCHIP_READ_BUF(), **WIZCHIP_WRITE_BUF()**
WIZCHIP_CRITICAL_ENTER()

Definition at line **1196** of file **w5500.h**.

#define getSn_RxMAX (sn) (((uint16_t)getSn_RXBUF_SIZE(sn)) << 10)

Socket_register_access_function.

Gets the max buffer size of socket sn passed as parameter.

Parameters
(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns
uint16_t. Value of Socket n RX max buffer size.

Definition at line **2093** of file **w5500.h**.

#define getSn_TxMAX (sn) (((uint16_t)getSn_TXBUF_SIZE(sn)) << 10)

Socket_register_access_function.

Gets the max buffer size of socket sn passed as parameters.

Parameters

(uint8_t)sn Socket number. It should be **0 ~ 7**.

Returns

uint16_t. Value of Socket n TX max buffer size.

Definition at line **2107** of file **w5500.h**.

Socket APIs

Macros | Functions | Variables

socket.c File Reference

SOCKET APIs Implements file. More...

```
#include "socket.h"
```

Go to the source code of this file.

Macros

#define **SOCK_ANY_PORT_NUM** 0xC000

#define **CHECK_SOCKNUM**()

#define **CHECK_SOCKMODE**(mode)

#define **CHECK_SOCKINIT**()

#define **CHECK_SOCKDATA**()

Functions

int8_t **socket** (uint8_t sn, uint8_t protocol, uint16_t port, uint8_t flag)
Open a socket. More...

int8_t **close** (uint8_t sn)
Close a socket. More...

int8_t **listen** (uint8_t sn)
Listen to a connection request from a client. More...

int8_t **connect** (uint8_t sn, uint8_t *addr, uint16_t port)
Try to connect a server. More...

int8_t **disconnect** (uint8_t sn)
Try to disconnect a connection socket. More...

int32_t **send** (uint8_t sn, uint8_t *buf, uint16_t len)
Send data to the connected peer in TCP socket. More...

int32_t **recv** (uint8_t sn, uint8_t *buf, uint16_t len)
Receive data from the connected peer. More...

int32_t **sendto** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t port)
Sends datagram to the peer with destination IP address and port number passed as parameter. More...

int32_t **recvfrom** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t *port)
Receive datagram of UDP or MACRAW. More...

int8_t **ctlsocket** (uint8_t sn, **ctlsock_type** cstype, void *arg)
Control socket. More...

int8_t **setsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
set socket options More...

int8_t **getsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
get socket options More...

Variables

uint8_t **sock_pack_info** [**_WIZCHIP_SOCK_NUM_**] = {0,}

Detailed Description

SOCKET APIs Implements file.

SOCKET APIs like as Berkeley Socket APIs.

Version
1.0.3

Date
2013/10/21

Revision history
<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2014/05/01> V1.0.3. Refer to M20140501

1. Implicit type casting -> Explicit type casting.
2. replace 0x01 with PACK_REMAINED in **recvfrom()**
3. Validation a destination ip in **connect()** & **sendto()**: It occurs a fatal error on converting unint32 address if uint8* addr parameter is not aligned by 4byte address. Copy 4 byte addr value into temporary uint32 variable and then compares it. <2013/12/20> V1.0.2 Refer to M20131220 Remove Warning. <2013/11/04> V1.0.1 2nd Release. Refer to "20131104". In **sendto()**, Add to clear timeout interrupt status (Sn_IR_TIMEOUT) <2013/10/21> 1st Release

Author
MidnightCow

Copyright

met:

```
* Redistributions of source code must retain the a
bove copyright
```

notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file **socket.c**.

Macro Definition Documentation

#define SOCK_ANY_PORT_NUM 0xC000

Definition at line **60** of file **socket.c**.

Referenced by **socket()**.

#define CHECK_SOCKNUM ( )

Value:

```
do{                                     \
      if(sn > _WIZCHIP_SOCK_NUM_) return
    SOCKERR_SOCKNUM;    \
   }while(0);                \
```

Definition at line **83** of file **socket.c**.

Referenced by **close()**, **connect()**, **ctlsocket()**, **disconnect()**, **getsockopt()**, **listen()**, **recv()**, **recvfrom()**, **send()**, **sendto()**, **setsockopt()**, and **socket()**.

#define CHECK_SOCKMODE (mode)

Value:

```
do{                                     \
      if((getSn_MR(sn) & 0x0F) != mode) return
    SOCKERR_SOCKMODE;   \
   }while(0);                \
```

Definition at line **88** of file **socket.c**.

Referenced by **connect()**, **disconnect()**, **getsockopt()**, **listen()**,

recv(), **send()**, and **setsockopt()**.

#define CHECK_SOCKINIT ( )

Value:

```
do{                          \
      if((getSn_SR(sn) != SOCK_INIT)) return
    SOCKERR_SOCKINIT; \
   }while(0);                  \
```

Definition at line **93** of file **socket.c**.

Referenced by **connect()**, and **listen()**.

#define CHECK_SOCKDATA ( )

Value:

```
do{                          \
      if(len == 0) return SOCKERR_DATALEN;   \
   }while(0);                  \
```

Definition at line **98** of file **socket.c**.

Referenced by **recv()**, **recvfrom()**, **send()**, and **sendto()**.

Variable Documentation

uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_] = {0,}

Definition at line **70** of file **socket.c**.

Referenced by **close()**, **getsockopt()**, **recv()**, **recvfrom()**, and **socket()**.

Socket APIs

Macros | Enumerations | Functions

socket.h File Reference

SOCKET APIs Header file. More...

```
#include "wizchip_conf.h"
```

Go to the source code of this file.

Macros

#define **SOCKET** uint8_t
SOCKET type define for legacy driver. More...

#define **SOCK_OK** 1
Result is OK about socket process. More...

#define **SOCK_BUSY** 0
Socket is busy on processing the operation. Valid only Non-block IO Mode. More...

#define **SOCK_FATAL** -1000
Result is fatal error about socket process. More...

#define **SOCK_ERROR** 0

#define **SOCKERR_SOCKNUM** (**SOCK_ERROR** - 1)
Invalid socket number. More...

#define **SOCKERR_SOCKOPT** (**SOCK_ERROR** - 2)
Invalid socket option. More...

#define **SOCKERR_SOCKINIT** (**SOCK_ERROR** - 3)
Socket is not initialized or SIPR is Zero IP address when Sn_MR_TCP. More...

#define **SOCKERR_SOCKCLOSED** (**SOCK_ERROR** - 4)
Socket unexpectedly closed. More...

#define **SOCKERR_SOCKMODE** (**SOCK_ERROR** - 5)
Invalid socket mode for socket operation. More...

#define **SOCKERR_SOCKFLAG** (**SOCK_ERROR** - 6)
Invalid socket flag. More...

#define **SOCKERR_SOCKSTATUS** (**SOCK_ERROR** - 7)
Invalid socket status for socket operation. More...

#define **SOCKERR_ARG** (**SOCK_ERROR** - 10)
Invalid argument. More...

#define **SOCKERR_PORTZERO** (**SOCK_ERROR** - 11)
Port number is zero. More...

#define **SOCKERR_IPINVALID** (**SOCK_ERROR** - 12)
Invalid IP address. More...

#define **SOCKERR_TIMEOUT** (**SOCK_ERROR** - 13)
Timeout occurred. More...

#define **SOCKERR_DATALEN** (**SOCK_ERROR** - 14)
Data length is zero or greater than buffer max size. More...

#define **SOCKERR_BUFFER** (**SOCK_ERROR** - 15)
Socket buffer is not enough for data communication. More...

#define **SOCKFATAL_PACKLEN** (**SOCK_FATAL** - 1)
Invalid packet length. Fatal Error. More...

#define **SF_ETHER_OWN** (**Sn_MR_MFEN**)
In **Sn_MR_MACRAW**, Receive only the packet as broadcast, multicast and own packet. More...

#define **SF_IGMP_VER2** (**Sn_MR_MC**)
In **Sn_MR_UDP** with **SF_MULTI_ENABLE**, Select IGMP version 2. More...

#define **SF_TCP_NODELAY** (**Sn_MR_ND**)
In **Sn_MR_TCP**, Use to nodelayed ack. More...

#define **SF_MULTI_ENABLE** (**Sn_MR_MULTI**)
In **Sn_MR_UDP**, Enable multicast mode. More...

#define **SF_BROAD_BLOCK** (**Sn_MR_BCASTB**)
In **Sn_MR_UDP** or **Sn_MR_MACRAW**, Block broadcast packet. Valid only in W5500. More...

#define **SF_MULTI_BLOCK** (**Sn_MR_MMB**)
In **Sn_MR_MACRAW**, Block multicast packet. Valid only in W5500. More...

#define **SF_IPv6_BLOCK** (**Sn_MR_MIP6B**)
In **Sn_MR_MACRAW**, Block IPv6 packet. Valid only in W5500. More...

#define **SF_UNI_BLOCK** (**Sn_MR_UCASTB**)
In **Sn_MR_UDP** with **SF_MULTI_ENABLE**. Valid only in W5500. More...

#define **SF_IO_NONBLOCK** 0x01
Socket nonblock io mode. It used parameter in **socket()**. More...

#define **PACK_FIRST** 0x80
In Non-TCP packet, It indicates to start receiving a packet. (When W5300, This flag can be applied) More...

#define **PACK_REMAINED** 0x01
In Non-TCP packet, It indicates to remain a packet to be received. (When W5300, This flag can be applied) More...

#define **PACK_COMPLETED** 0x00
In Non-TCP packet, It indicates to complete to receive a packet. (When W5300, This flag can be applied) More...

#define **PACK_FIFOBYTE** 0x02
Valid only W5300, It indicate to have read already the

Sn_RX_FIFOR. More...

#define **SOCK_IO_BLOCK** 0
Socket Block IO Mode in **setsockopt()**. More...

#define **SOCK_IO_NONBLOCK** 1
Socket Non-block IO Mode in **setsockopt()**. More...

Enumerations

enum	**sockint_kind** {
	  SIK_CONNECTED = (1 << 0), **SIK_DISCONNECTED** = (1 << 1), **SIK_RECEIVED** = (1 << 2), **SIK_TIMEOUT** = (1 << 3),
	  SIK_SENT = (1 << 4), **SIK_ALL** = 0x1F
	}
	The kind of Socket Interrupt. More...

enum	**ctlsock_type** {
	  CS_SET_IOMODE, **CS_GET_IOMODE**, **CS_GET_MAXTXBUF**, **CS_GET_MAXRXBUF**,
	  CS_CLR_INTERRUPT, **CS_GET_INTERRUPT**, **CS_SET_INTMASK**, **CS_GET_INTMASK**
	}
	The type of **ctlsocket()**. More...

enum	**sockopt_type** {
	  SO_FLAG, **SO_TTL**, **SO_TOS**, **SO_MSS**,
	  SO_DESTIP, **SO_DESTPORT**, **SO_KEEPALIVESEND**, **SO_KEEPALIVEAUTO**,
	  SO_SENDBUF, **SO_RECVBUF**, **SO_STATUS**, **SO_REMAINSIZE**,
	  SO_PACKINFO
	}
	The type of socket option in **setsockopt()** or **getsockopt()** More...

Functions

int8_t **socket** (uint8_t sn, uint8_t protocol, uint16_t port, uint8_t flag)
Open a socket. More...

int8_t **close** (uint8_t sn)
Close a socket. More...

int8_t **listen** (uint8_t sn)
Listen to a connection request from a client. More...

int8_t **connect** (uint8_t sn, uint8_t *addr, uint16_t port)
Try to connect a server. More...

int8_t **disconnect** (uint8_t sn)
Try to disconnect a connection socket. More...

int32_t **send** (uint8_t sn, uint8_t *buf, uint16_t len)
Send data to the connected peer in TCP socket. More...

int32_t **recv** (uint8_t sn, uint8_t *buf, uint16_t len)
Receive data from the connected peer. More...

int32_t **sendto** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t port)
Sends datagram to the peer with destination IP address and port number passed as parameter. More...

int32_t **recvfrom** (uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t *port)
Receive datagram of UDP or MACRAW. More...

int8_t **ctlsocket** (uint8_t sn, **ctlsock_type** cstype, void *arg)
Control socket. More...

int8_t **setsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
set socket options More...

int8_t **getsockopt** (uint8_t sn, **sockopt_type** sotype, void *arg)
get socket options More...

Detailed Description

SOCKET APIs Header file.

SOCKET APIs like as berkeley socket api.

Version

1.0.2

Date

2013/10/21

Revision history

<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2014/05/01> V1.0.2. Refer to M20140501

1. Modify the comment : SO_REMAINED -> PACK_REMAINED
2. Add the comment as zero byte udp data reception in **getsockopt()**. <2013/10/21> 1st Release

Author

MidnightCow

Copyright

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **socket.h**.

Macro Definition Documentation

#define SOCKET uint8_t

SOCKET type define for legacy driver.

Definition at line **90** of file **socket.h**.

#define SOCK_OK 1

Result is OK about socket process.

Definition at line **92** of file **socket.h**.

Referenced by **close()**, **connect()**, **ctlsocket()**, **disconnect()**, **getsockopt()**, **listen()**, and **setsockopt()**.

#define SOCK_BUSY 0

Socket is busy on processing the operation. Valid only Non-block IO Mode.

Definition at line **93** of file **socket.h**.

Referenced by **connect()**, **disconnect()**, **recv()**, **recvfrom()**, **send()**, and **sendto()**.

#define SOCK_FATAL -1000

Result is fatal error about socket process.

Definition at line **94** of file **socket.h**.

#define SOCK_ERROR 0

Definition at line **96** of file **socket.h**.

#define SOCKERR_SOCKNUM (SOCK_ERROR - 1)

Invalid socket number.

Definition at line **97** of file **socket.h**.

#define SOCKERR_SOCKOPT (SOCK_ERROR - 2)

Invalid socket option.

Definition at line **98** of file **socket.h**.

Referenced by **getsockopt()**, and **setsockopt()**.

#define SOCKERR_SOCKINIT (SOCK_ERROR - 3)

Socket is not initialized or SIPR is Zero IP address when Sn_MR_TCP.

Definition at line **99** of file **socket.h**.

Referenced by **socket()**.

#define SOCKERR_SOCKCLOSED (SOCK_ERROR - 4)

Socket unexpectedly closed.

Definition at line **100** of file **socket.h**.

Referenced by **connect()**, **listen()**, **recvfrom()**, and **sendto()**.

#define SOCKERR_SOCKMODE (SOCK_ERROR - 5)

Invalid socket mode for socket operation.

Definition at line **101** of file **socket.h**.

Referenced by **recvfrom()**, **sendto()**, and **socket()**.

#define SOCKERR_SOCKFLAG (SOCK_ERROR - 6)

Invalid socket flag.

Definition at line **102** of file **socket.h**.

Referenced by **socket()**.

#define SOCKERR_SOCKSTATUS (SOCK_ERROR - 7)

Invalid socket status for socket operation.

Definition at line **103** of file **socket.h**.

Referenced by **recv()**, **send()**, and **sendto()**.

#define SOCKERR_ARG (SOCK_ERROR - 10)

Invalid argument.

Definition at line **104** of file **socket.h**.

Referenced by **ctlsocket()**, and **setsockopt()**.

#define SOCKERR_PORTZERO (SOCK_ERROR - 11)

Port number is zero.

Definition at line **105** of file **socket.h**.

Referenced by **connect()**, and **sendto()**.

#define SOCKERR_IPINVALID (SOCK_ERROR - 12)

Invalid IP address.

Definition at line **106** of file **socket.h**.

Referenced by **connect()**, and **sendto()**.

#define SOCKERR_TIMEOUT (SOCK_ERROR - 13)

Timeout occurred.

Definition at line **107** of file **socket.h**.

Referenced by **connect()**, **disconnect()**, **send()**, **sendto()**, and **setsockopt()**.

#define SOCKERR_DATALEN (SOCK_ERROR - 14)

Data length is zero or greater than buffer max size.

Definition at line **108** of file **socket.h**.

#define SOCKERR_BUFFER (SOCK_ERROR - 15)

Socket buffer is not enough for data communication.

Definition at line **109** of file **socket.h**.

#define SOCKFATAL_PACKLEN (SOCK_FATAL - 1)

Invalid packet length. Fatal Error.

Definition at line **111** of file **socket.h**.

Referenced by **recvfrom()**.

#define SF_ETHER_OWN (Sn_MR_MFEN)

In **Sn_MR_MACRAW**, Receive only the packet as broadcast, multicast and own packet.

Definition at line **116** of file **socket.h**.

#define SF_IGMP_VER2 (Sn_MR_MC)

In **Sn_MR_UDP** with **SF_MULTI_ENABLE**, Select IGMP version 2.

Definition at line **117** of file **socket.h**.

Referenced by **socket()**.

#define SF_TCP_NODELAY (Sn_MR_ND)

In **Sn_MR_TCP**, Use to nodelayed ack.

Definition at line **118** of file **socket.h**.

Referenced by **socket()**.

#define SF_MULTI_ENABLE (Sn_MR_MULTI)

In **Sn_MR_UDP**, Enable multicast mode.

Definition at line **119** of file **socket.h**.

Referenced by **socket()**.

#define SF_BROAD_BLOCK (Sn_MR_BCASTB)

In **Sn_MR_UDP** or **Sn_MR_MACRAW**, Block broadcast packet. Valid only in W5500.

Definition at line **122** of file **socket.h**.

#define SF_MULTI_BLOCK (Sn_MR_MMB)

In **Sn_MR_MACRAW**, Block multicast packet. Valid only in W5500.

Definition at line **123** of file **socket.h**.

#define SF_IPv6_BLOCK (Sn_MR_MIP6B)

In **Sn_MR_MACRAW**, Block IPv6 packet. Valid only in W5500.

Definition at line **124** of file **socket.h**.

#define SF_UNI_BLOCK (Sn_MR_UCASTB)

In **Sn_MR_UDP** with **SF_MULTI_ENABLE**. Valid only in W5500.

Definition at line **125** of file **socket.h**.

Referenced by **socket()**.

#define SF_IO_NONBLOCK 0x01

Socket nonblock io mode. It used parameter in **socket()**.

Definition at line **133** of file **socket.h**.

Referenced by **socket()**.

#define PACK_FIRST 0x80

In Non-TCP packet, It indicates to start receiving a packet. (When W5300, This flag can be applied)

Definition at line **138** of file **socket.h**.

Referenced by **recv()**, and **recvfrom()**.

#define PACK_REMAINED 0x01

In Non-TCP packet, It indicates to remain a packet to be received. (When W5300, This flag can be applied)

Definition at line **139** of file **socket.h**.

Referenced by **recv()**, and **recvfrom()**.

#define PACK_COMPLETED 0x00

In Non-TCP packet, It indicates to complete to receive a packet. (When W5300, This flag can be applied)

Definition at line **140** of file **socket.h**.

Referenced by **recv()**, **recvfrom()**, and **socket()**.

#define PACK_FIFOBYTE 0x02

Valid only W5300, It indicate to have read already the Sn_RX_FIFOR.

Definition at line **142** of file **socket.h**.

Referenced by **recv()**, and **recvfrom()**.

#define SOCK_IO_BLOCK 0

Socket Block IO Mode in **setsockopt()**.

Definition at line **333** of file **socket.h**.

Referenced by **ctlsocket()**.

#define SOCK_IO_NONBLOCK 1

Socket Non-block IO Mode in **setsockopt()**.

Definition at line **334** of file **socket.h**.

Referenced by **ctlsocket()**.

Socket APIs

Functions | Variables

wizchip_conf.c File Reference

WIZCHIP Config Header File. More...

```
#include <stddef.h> #include "wizchip_conf.h"
```

Go to the source code of this file.

Functions

void	**wizchip_cris_enter** (void) Default function to enable interrupt. More...
void	**wizchip_cris_exit** (void) Default function to disable interrupt. More...
void	**wizchip_cs_select** (void) Default function to select chip. More...
void	**wizchip_cs_deselect** (void) Default function to deselect chip. More...
iodata_t	**wizchip_bus_readdata** (uint32_t AddrSel) Default function to read in direct or indirect interface. More...
void	**wizchip_bus_writedata** (uint32_t AddrSel, **iodata_t** wb) Default function to write in direct or indirect interface. More...
uint8_t	**wizchip_spi_readbyte** (void) Default function to read in SPI interface. More...
void	**wizchip_spi_writebyte** (uint8_t wb) Default function to write in SPI interface. More...
void	**wizchip_spi_readburst** (uint8_t *pBuf, uint16_t len) Default function to burst read in SPI interface. More...
void	**wizchip_spi_writeburst** (uint8_t *pBuf, uint16_t len) Default function to burst write in SPI interface. More...

void	**reg_wizchip_cris_cbfunc** (void(*cris_en)(void), void(*cris_ex)(void)) Registers call back function for critical section of I/O functions such as **WIZCHIP_READ**, **WIZCHIP_WRITE**, **WIZCHIP_READ_BUF** and **WIZCHIP_WRITE_BUF**. More...
void	**reg_wizchip_cs_cbfunc** (void(*cs_sel)(void), void(*cs_desel)(void)) Registers call back function for WIZCHIP select & deselect. More...
void	**reg_wizchip_bus_cbfunc** (**iodata_t**(*bus_rb)(uint32_t addr), void(*bus_wb)(uint32_t addr, **iodata_t** wb)) Registers call back function for bus interface. More...
void	**reg_wizchip_spi_cbfunc** (uint8_t(*spi_rb)(void), void(*spi_wb)(uint8_t wb)) Registers call back function for SPI interface. More...
void	**reg_wizchip_spiburst_cbfunc** (void(*spi_rb)(uint8_t *pBuf, uint16_t len), void(*spi_wb)(uint8_t *pBuf, uint16_t len)) Registers call back function for SPI interface. More...
int8_t	**ctlwizchip** (**ctlwizchip_type** cwtype, void *arg) Controls to the WIZCHIP. More...
int8_t	**ctlnetwork** (**ctlnetwork_type** cntype, void *arg) Controls to network. More...
void	**wizchip_sw_reset** (void)

Reset WIZCHIP by softly. More...

int8_t **wizchip_init** (uint8_t *txsize, uint8_t *rxsize)
Initializes WIZCHIP with socket buffer size. More...

void **wizchip_clrinterrupt** (**intr_kind** intr)
Clear Interrupt of WIZCHIP. More...

intr_kind **wizchip_getinterrupt** (void)
Get Interrupt of WIZCHIP. More...

void **wizchip_setinterruptmask** (**intr_kind** intr)
Mask or Unmask Interrupt of WIZCHIP. More...

intr_kind **wizchip_getinterruptmask** (void)
Get Interrupt mask of WIZCHIP. More...

int8_t **wizphy_getphylink** (void)
get the link status of phy in WIZCHIP. No use in W5100 More...

int8_t **wizphy_getphypmode** (void)
get the power mode of PHY in WIZCHIP. No use in W5100 More...

void **wizphy_reset** (void)
Reset phy. Vailid only in W5500. More...

void **wizphy_setphyconf** (**wiz_PhyConf** *phyconf)
Set the phy information for WIZCHIP without power mode. More...

void **wizphy_getphyconf** (**wiz_PhyConf** *phyconf)
Get phy configuration information. More...

void **wizphy_getphystat** (**wiz_PhyConf** *phyconf)

	Get phy status. More...
int8_t	**wizphy_setphypmode** (uint8_t pmode) set the power mode of phy inside WIZCHIP. Refer to **PHYCFGR** in W5500, **PHYSTATUS** in W5200 More...
void	**wizchip_setnetinfo** (**wiz_NetInfo** *pnetinfo) Set the network information for WIZCHIP. More...
void	**wizchip_getnetinfo** (**wiz_NetInfo** *pnetinfo) Get the network information for WIZCHIP. More...
int8_t	**wizchip_setnetmode** (**netmode_type** netmode) Set the network mode such WOL, PPPoE, Ping Block, and etc. More...
netmode_type	**wizchip_getnetmode** (void) Get the network mode such WOL, PPPoE, Ping Block, and etc. More...
void	**wizchip_settimeout** (**wiz_NetTimeout** *nettime) Set retry time value(*RTR*) and retry count(*RCR*). More...
void	**wizchip_gettimeout** (**wiz_NetTimeout** *nettime) Get retry time value(*RTR*) and retry count(*RCR*). More...

Variables

_WIZCHIP WIZCHIP

Detailed Description

WIZCHIP Config Header File.

Version

1.0.1

Date

2013/10/21

Revision history

<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2014/05/01> V1.0.1 Refer to M20140501

1. Explicit type casting in **wizchip_bus_readdata()** & **wizchip_bus_writedata()**

Definition in file **wizchip_conf.c**.

Function Documentation

void wizchip_cris_enter (void)

Default function to enable interrupt.

uint32_t type converts into ptrdiff_t first. And then recoverting it into uint
the warning when pointer type size is not 32bit. If ptrdiff_t doesn't suppo
You should must replace ptrdiff_t into your suitable pointer type. <2013/

Author

MidnightCow

Copyright

ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR C
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT O
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BU
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LI
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDIN
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Note

This function help not to access wrong address. If you do not desc
register any functions, null function is called.

Definition at line **67** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_cris_cbfunc()**.

void wizchip_cris_exit (void)

Default function to disable interrupt.

Note

This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **75** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_cris_cbfunc()**.

void wizchip_cs_select (void)

Default function to select chip.

Note

This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **83** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_cs_cbfunc()**.

void wizchip_cs_deselect (void)

Default function to deselect chip.

Note
This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **91** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_cs_cbfunc()**.

iodata_t wizchip_bus_readdata (uint32_t AddrSel)

Default function to read in direct or indirect interface.

Note
This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **100** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_bus_cbfunc()**.

void wizchip_bus_writedata (uint32_t AddrSel, iodata_t wb)

Default function to write in direct or indirect interface.

Note

This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **109** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_bus_cbfunc()**.

uint8_t wizchip_spi_readbyte (void)

Default function to read in SPI interface.

Note

This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **117** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_spi_cbfunc()**.

void wizchip_spi_writebyte (uint8_t wb)

Default function to write in SPI interface.

Note

This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **125** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_spi_cbfunc()**.

void wizchip_spi_readburst (uint8_t * pBuf, uint16_t len

)

Default function to burst read in SPI interface.

Note
This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **133** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_spiburst_cbfunc()**.

void wizchip_spi_writeburst (uint8_t * pBuf,
uint16_t len
)

Default function to burst write in SPI interface.

Note
This function help not to access wrong address. If you do not describe this function or register any functions, null function is called.

Definition at line **141** of file **wizchip_conf.c**.

Referenced by **reg_wizchip_spiburst_cbfunc()**.

void reg_wizchip_cris_cbfunc (void(*)(void) cris_en,
void(*)(void) cris_ex
)

Registers call back function for critical section of I/O functions such as **WIZCHIP_READ**, **WIZCHIP_WRITE**, **WIZCHIP_READ_BUF** and **WIZCHIP_WRITE_BUF**.

Parameters

cris_en : callback function for critical section enter.
cris_ex : callback function for critical section exit.

Todo:
Describe **WIZCHIP_CRITICAL_ENTER** and **WIZCHIP_CRITICAL_EXIT** marco or register your functions.

Note
If you do not describe or register, default functions(**wizchip_cris_enter** & **wizchip_cris_exit**) is called.

Definition at line **186** of file **wizchip_conf.c**.

References **__WIZCHIP::_CRIS::_enter**, **__WIZCHIP::_CRIS::_exit**, **__WIZCHIP::CRIS**, **wizchip_cris_enter()**, and **wizchip_cris_exit()**.

---

void reg_wizchip_cs_cbfunc (void(*)(void) cs_sel,
void(*)(void) cs_desel
)

Registers call back function for WIZCHIP select & deselect.

Parameters
cs_sel : callback function for WIZCHIP select
cs_desel : callback fucntion for WIZCHIP deselect

Todo:
Describe **wizchip_cs_select** and **wizchip_cs_deselect** function or register your functions.

Note
If you do not describe or register, null function is called.

Definition at line **200** of file **wizchip_conf.c**.

References **__WIZCHIP::_CS::_deselect**, **__WIZCHIP::_CS::_select**, **__WIZCHIP::CS**, **wizchip_cs_deselect()**, and **wizchip_cs_select()**.

void
reg_wizchip_bus_cbfunc (iodata_t(*)(uint32_t addr) bus_
void(*)(uint32_t addr, iodata_t wb) bus_
)

Registers call back function for bus interface.

Parameters

bus_rb : callback function to read byte data using system bus
bus_wb : callback function to write byte data using system bus

Todo:
Describe wizchip_bus_readbyte and wizchip_bus_writebyte functic
or register your functions.

Note
If you do not describe or register, null function is called.

Definition at line **216** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_data**, **_WIZCHIP_IO_MODE_BU**
__WIZCHIP::_IF::_write_data, **__WIZCHIP::_IF::BUS**, **__WIZCHIP::I**
__WIZCHIP::if_mode, **wizchip_bus_readdata()**, and
wizchip_bus_writedata().

void reg_wizchip_spi_cbfunc (uint8_t(*)(void) spi_rb,
void(*)(uint8_t wb) spi_wb
)

Registers call back function for SPI interface.

Parameters

spi_rb : callback function to read byte using SPI
spi_wb : callback function to write byte using SPI

Todo:
Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte**

function or register your functions.

Note

If you do not describe or register, null function is called.

Definition at line **244** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_byte**, **_WIZCHIP_IO_MODE_SPI_**, **__WIZCHIP::_IF::_write_byte**, **__WIZCHIP::IF**, **__WIZCHIP::if_mode**, **__WIZCHIP::_IF::SPI**, **wizchip_spi_readbyte()**, and **wizchip_spi_writebyte()**.

void
reg_wizchip_spiburst_cbfunc (void(*)(uint8_t *pBuf, uint16_t len)
void(*)(uint8_t *pBuf, uint16_t len)
)

Registers call back function for SPI interface.

Parameters

spi_rb : callback function to burst read using SPI

spi_wb : callback function to burst write using SPI

Todo:

Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte** func
register your functions.

Note

If you do not describe or register, null function is called.

Definition at line **261** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_burst**, **_WIZCHIP_IO_MODE_SF**
__WIZCHIP::_IF::_write_burst, **__WIZCHIP::IF**, **__WIZCHIP::if_mod**
__WIZCHIP::_IF::SPI, **wizchip_spi_readburst()**, and
wizchip_spi_writeburst().

int8_t wizphy_getphylink (void)

get the link status of phy in WIZCHIP. No use in W5100

Definition at line **575** of file **wizchip_conf.c**.

References **getPHYCFGR**, **getPHYSTATUS**, **PHY_LINK_OFF**, **PHY_LINK_ON**, **PHYCFGR_LNK_ON**, and **PHYSTATUS_LINK**.

Referenced by **ctlwizchip()**.

int8_t wizphy_getphypmode (void)

get the power mode of PHY in WIZCHIP. No use in W5100

Definition at line **596** of file **wizchip_conf.c**.

References **getPHYCFGR**, **getPHYSTATUS**, **PHY_POWER_DOWN**, **PHY_POWER_NORM**, **PHYCFGR_OPMDC_PDOWN**, and **PHYSTATUS_POWERDOWN**.

Referenced by **ctlwizchip()**.

void wizphy_reset (void)

Reset phy. Vailid only in W5500.

Definition at line **617** of file **wizchip_conf.c**.

References **getPHYCFGR**, **PHYCFGR_RST**, and **setPHYCFGR**.

Referenced by **ctlwizchip()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

Variable Documentation

_WIZCHIP WIZCHIP

Initial value:

```
=
      {
 _WIZCHIP_IO_MODE_,
 _WIZCHIP_ID_ ,
 wizchip_cris_enter,
 wizchip_cris_exit,
 wizchip_cs_select,
 wizchip_cs_deselect,



 wizchip_bus_readdata,
 wizchip_bus_writedata,

      }
```

\ref _WIZCHIP instance

Definition at line **165** of file **wizchip_conf.c**.

Socket APIs

Classes | Macros | Typedefs | Enumerations | Functions | Variables

wizchip_conf.h File Reference

WIZCHIP Config Header File. More...

```
#include <stdint.h> #include "W5500/w5500.h"
```

Go to the source code of this file.

Classes

struct	**__WIZCHIP** The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**. More...
struct	**__WIZCHIP::_CRIS**
struct	**__WIZCHIP::_CS**
union	**__WIZCHIP::_IF**
struct	**wiz_PhyConf_t**
struct	**wiz_NetInfo_t**
struct	**wiz_NetTimeout_t**

Macros

#define **_WIZCHIP_** 5500
Select WIZCHIP. More...

#define **_WIZCHIP_IO_MODE_NONE_** 0x0000

#define **_WIZCHIP_IO_MODE_BUS_** 0x0100

#define **_WIZCHIP_IO_MODE_SPI_** 0x0200

#define **_WIZCHIP_IO_MODE_BUS_DIR_** (**_WIZCHIP_IO_MODE_E**
1)

#define **_WIZCHIP_IO_MODE_BUS_INDIR_** (**_WIZCHIP_IO_MODE**
+ 2)

#define **_WIZCHIP_IO_MODE_SPI_VDM_** (**_WIZCHIP_IO_MODE_S**
1)

#define **_WIZCHIP_IO_MODE_SPI_FDM_** (**_WIZCHIP_IO_MODE_S**
2)

#define **_WIZCHIP_ID_** "W5500\0"

#define **_WIZCHIP_IO_MODE_** **_WIZCHIP_IO_MODE_SPI_VDM_**
Define interface mode.
. More...

#define **_WIZCHIP_IO_BASE_** 0x00000000
Define I/O base address when BUS IF mode. More...

#define **_WIZCHIP_SOCK_NUM_** 8
The count of independant socket of **WIZCHIP**. More...

#define **PHY_CONFBY_HW** 0

Configured PHY operation mode by HW pin. More...

#define **PHY_CONFBY_SW** 1
Configured PHY operation mode by SW register. More...

#define **PHY_MODE_MANUAL** 0
Configured PHY operation mode with user setting. More...

#define **PHY_MODE_AUTONEGO** 1
Configured PHY operation mode with auto-negotiation. More...

#define **PHY_SPEED_10** 0
Link Speed 10. More...

#define **PHY_SPEED_100** 1
Link Speed 100. More...

#define **PHY_DUPLEX_HALF** 0
Link Half-Duplex. More...

#define **PHY_DUPLEX_FULL** 1
Link Full-Duplex. More...

#define **PHY_LINK_OFF** 0
Link Off. More...

#define **PHY_LINK_ON** 1
Link On. More...

#define **PHY_POWER_NORM** 0
PHY power normal mode. More...

#define **PHY_POWER_DOWN** 1
PHY power down mode. More...

Typedefs

typedef uint8_t	**iodata_t**
typedef struct __**WIZCHIP**	**_WIZCHIP** The set of callback functions for W5500:**WIZCHIP I/O functions** W5200:**WIZCHIP I/O functions**. More...
typedef struct **wiz_PhyConf_t**	**wiz_PhyConf**
typedef struct **wiz_NetInfo_t**	**wiz_NetInfo**
typedef struct **wiz_NetTimeout_t**	**wiz_NetTimeout**

Enumerations

enum	**ctlwizchip_type** {
	CW_RESET_WIZCHIP, **CW_INIT_WIZCHIP**,
	CW_GET_INTERRUPT, **CW_CLR_INTERRUPT**,
	CW_SET_INTRMASK, **CW_GET_INTRMASK**,
	CW_SET_INTRTIME, **CW_GET_INTRTIME**,
	CW_GET_ID, **CW_RESET_PHY**, **CW_SET_PHYCONF**,
	CW_GET_PHYCONF,
	CW_GET_PHYSTATUS, **CW_SET_PHYPOWMODE**,
	CW_GET_PHYPOWMODE, **CW_GET_PHYLINK**
	}

enum	**ctlnetwork_type** {
	CN_SET_NETINFO, **CN_GET_NETINFO**,
	CN_SET_NETMODE, **CN_GET_NETMODE**,
	CN_SET_TIMEOUT, **CN_GET_TIMEOUT**
	}

enum	**intr_kind** {
	IK_WOL = (1 << 4), **IK_PPPOE_TERMINATED** = (1 << 5),
	IK_DEST_UNREACH = (1 << 6), **IK_IP_CONFLICT** = (1 << 7),
	IK_SOCK_0 = (1 << 8), **IK_SOCK_1** = (1 << 9), **IK_SOCK_2** = (1 << 10), **IK_SOCK_3** = (1 << 11),
	IK_SOCK_4 = (1 << 12), **IK_SOCK_5** = (1 << 13),
	IK_SOCK_6 = (1 << 14), **IK_SOCK_7** = (1 << 15),
	IK_SOCK_ALL = (0xFF << 8)
	}

enum	**dhcp_mode** { **NETINFO_STATIC** = 1, **NETINFO_DHCP** }
enum	**netmode_type** { **NM_FORCEARP** = (1<<1), **NM_WAKEONLAN** = (1<<5), **NM_PINGBLOCK** = (1<<4), **NM_PPPOE** = (1<<3) }

Functions

void	**reg_wizchip_cris_cbfunc** (void(*cris_en)(void), void(*cris_ex)(void)) Registers call back function for critical section of I/O functions such as **WIZCHIP_READ**, **WIZCHIP_WRITE**, **WIZCHIP_READ_BUF** and **WIZCHIP_WRITE_BUF**. More...
void	**reg_wizchip_cs_cbfunc** (void(*cs_sel)(void), void(*cs_desel)(void)) Registers call back function for WIZCHIP select & deselect. More...
void	**reg_wizchip_bus_cbfunc** (**iodata_t**(*bus_rb)(uint32_t addr), void(*bus_wb)(uint32_t addr, **iodata_t** wb)) Registers call back function for bus interface. More...
void	**reg_wizchip_spi_cbfunc** (uint8_t(*spi_rb)(void), void(*spi_wb)(uint8_t wb)) Registers call back function for SPI interface. More...
void	**reg_wizchip_spiburst_cbfunc** (void(*spi_rb)(uint8_t *pBuf, uint16_t len), void(*spi_wb)(uint8_t *pBuf, uint16_t len)) Registers call back function for SPI interface. More...
int8_t	**ctlwizchip** (**ctlwizchip_type** cwtype, void *arg) Controls to the WIZCHIP. More...
int8_t	**ctlnetwork** (**ctlnetwork_type** cntype, void *arg) Controls to network. More...

void	**wizchip_sw_reset** (void) Reset WIZCHIP by softly. More...
int8_t	**wizchip_init** (uint8_t *txsize, uint8_t *rxsize) Initializes WIZCHIP with socket buffer size. More...
void	**wizchip_clrinterrupt** (**intr_kind** intr) Clear Interrupt of WIZCHIP. More...
intr_kind	**wizchip_getinterrupt** (void) Get Interrupt of WIZCHIP. More...
void	**wizchip_setinterruptmask** (**intr_kind** intr) Mask or Unmask Interrupt of WIZCHIP. More...
intr_kind	**wizchip_getinterruptmask** (void) Get Interrupt mask of WIZCHIP. More...
int8_t	**wizphy_getphylink** (void) get the link status of phy in WIZCHIP. No use in W5100 More...
int8_t	**wizphy_getphypmode** (void) get the power mode of PHY in WIZCHIP. No use in W5100 More...
void	**wizphy_reset** (void) Reset phy. Vailid only in W5500. More...
void	**wizphy_setphyconf** (**wiz_PhyConf** *phyconf) Set the phy information for WIZCHIP without power mode. More...
void	**wizphy_getphyconf** (**wiz_PhyConf** *phyconf) Get phy configuration information. More...

void	**wizphy_getphystat** (**wiz_PhyConf** *phyconf) Get phy status. More...
int8_t	**wizphy_setphypmode** (uint8_t pmode) set the power mode of phy inside WIZCHIP. Refer to **PHYCFGR** in W5500, **PHYSTATUS** in W5200 More...
void	**wizchip_setnetinfo** (**wiz_NetInfo** *pnetinfo) Set the network information for WIZCHIP. More...
void	**wizchip_getnetinfo** (**wiz_NetInfo** *pnetinfo) Get the network information for WIZCHIP. More...
int8_t	**wizchip_setnetmode** (**netmode_type** netmode) Set the network mode such WOL, PPPoE, Ping Block, and etc. More...
netmode_type	**wizchip_getnetmode** (void) Get the network mode such WOL, PPPoE, Ping Block, and etc. More...
void	**wizchip_settimeout** (**wiz_NetTimeout** *nettime) Set retry time value(*RTR*) and retry count(*RCR*). More...
void	**wizchip_gettimeout** (**wiz_NetTimeout** *nettime) Get retry time value(*RTR*) and retry count(*RCR*). More...

Variables

_WIZCHIP WIZCHIP

Detailed Description

WIZCHIP Config Header File.

Version
1.0.0

Date
2013/10/21

Revision history
<2015/02/05> Notice The version history is not updated after this point. Download the latest version directly from GitHub. Please visit the our GitHub repository for ioLibrary. >> https://github.com/Wiznet/ioLibrary_Driver <2013/10/21> 1st Release

Author
MidnightCow

Copyright

- Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Definition in file **wizchip_conf.h**.

Macro Definition Documentation

#define _WIZCHIP_ 5500

Select WIZCHIP.

Todo:
You should select one, **5100**, **5200**, **5300**, **5500** or etc.

ex> `#define _WIZCHIP_ 5500`

Definition at line **64** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_NONE_ 0x0000

Definition at line **67** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_BUS_ 0x0100

Bus interface mode

Definition at line **68** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_bus_cbfunc()**.

#define _WIZCHIP_IO_MODE_SPI_ 0x0200

SPI interface mode

Definition at line **69** of file **wizchip_conf.h**.

Referenced by **reg_wizchip_spi_cbfunc()**, and

reg_wizchip_spiburst_cbfunc().

#define _WIZCHIP_IO_MODE_BUS_DIR_ (_WIZCHIP_IO_MODE_BUS_ + 1)

BUS interface mode for direct

Definition at line **75** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_BUS_INDIR_ (_WIZCHIP_IO_MODE_BUS_ + 2)

BUS interface mode for indirect

Definition at line **76** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_SPI_VDM_ (_WIZCHIP_IO_MODE_SPI_ + 1)

SPI interface mode for variable length data

Definition at line **78** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_SPI_FDM_ (_WIZCHIP_IO_MODE_SPI_ + 2)

SPI interface mode for fixed length data mode

Definition at line **79** of file **wizchip_conf.h**.

#define _WIZCHIP_ID_ "W5500\0"

Definition at line **111** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_SPI_VDM_

Define interface mode.
.

Todo:
Should select interface mode as chip.

- **_WIZCHIP_IO_MODE_SPI_**
  -**_WIZCHIP_IO_MODE_SPI_VDM_** : Valid only in **_WIZCHIP_** == 5500
  -**_WIZCHIP_IO_MODE_SPI_FDM_** : Valid only in **_WIZCHIP_** == 5500
- **_WIZCHIP_IO_MODE_BUS_**
  - **_WIZCHIP_IO_MODE_BUS_DIR_**
  - **_WIZCHIP_IO_MODE_BUS_INDIR_**
- Others will be defined in future.

ex> `#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_SPI_VDM_`

Definition at line **128** of file **wizchip_conf.h**.

#define _WIZCHIP_IO_BASE_ 0x00000000

Define I/O base address when BUS IF mode.

Todo:
Should re-define it to fit your system when BUS IF Mode (**_WIZCHIP_IO_MODE_BUS_**, **_WIZCHIP_IO_MODE_BUS_DIR_**, **_WIZCHIP_IO_MODE_BUS_INDIR_**).

ex> `#define _WIZCHIP_IO_BASE_ 0x00008000`

Definition at line **176** of file **wizchip_conf.h**.

#define _WIZCHIP_SOCK_NUM_ 8

The count of independant socket of **WIZCHIP**.

Definition at line **188** of file **wizchip_conf.h**.

Referenced by **ctlwizchip()**, and **wizchip_init()**.

#define PHY_CONFBY_HW 0

Configured PHY operation mode by HW pin.

Definition at line **343** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**.

#define PHY_CONFBY_SW 1

Configured PHY operation mode by SW register.

Definition at line **344** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

#define PHY_MODE_MANUAL 0

Configured PHY operation mode with user setting.

Definition at line **345** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**.

#define PHY_MODE_AUTONEGO 1

Configured PHY operation mode with auto-negotiation.

Definition at line **346** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_setphyconf()**.

#define PHY_SPEED_10 0

Link Speed 10.

Definition at line **347** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_getphystat()**.

#define PHY_SPEED_100 1

Link Speed 100.

Definition at line **348** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_getphystat()**, and **wizphy_setphyconf()**.

#define PHY_DUPLEX_HALF 0

Link Half-Duplex.

Definition at line **349** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, and **wizphy_getphystat()**.

#define PHY_DUPLEX_FULL 1

Link Full-Duplex.

Definition at line **350** of file **wizchip_conf.h**.

Referenced by **wizphy_getphyconf()**, **wizphy_getphystat()**, and **wizphy_setphyconf()**.

#define PHY_LINK_OFF 0

Link Off.

Definition at line **351** of file **wizchip_conf.h**.

Referenced by **wizphy_getphylink()**.

#define PHY_LINK_ON 1

Link On.

Definition at line **352** of file **wizchip_conf.h**.

Referenced by **wizphy_getphylink()**.

#define PHY_POWER_NORM 0

PHY power normal mode.

Definition at line **353** of file **wizchip_conf.h**.

Referenced by **wizphy_getphypmode()**.

#define PHY_POWER_DOWN 1

PHY power down mode.

Definition at line **354** of file **wizchip_conf.h**.

Referenced by **wizphy_getphypmode()**, and **wizphy_setphypmode()**.

Typedef Documentation

typedef uint8_t iodata_t

Definition at line **131** of file **wizchip_conf.h**.

Function Documentation

void reg_wizchip_cris_cbfunc (void(*)(void) cris_en,
void(*)(void) cris_ex
)

Registers call back function for critical section of I/O functions such as **WIZCHIP_READ**, **WIZCHIP_WRITE**, **WIZCHIP_READ_BUF** and **WIZCHIP_WRITE_BUF**.

Parameters

cris_en : callback function for critical section enter.

cris_ex : callback function for critical section exit.

Todo:

Describe **WIZCHIP_CRITICAL_ENTER** and **WIZCHIP_CRITICAL_EXIT** marco or register your functions.

Note

If you do not describe or register, default functions(**wizchip_cris_enter** & **wizchip_cris_exit**) is called.

Definition at line **186** of file **wizchip_conf.c**.

References **__WIZCHIP::_CRIS::_enter**, **__WIZCHIP::_CRIS::_exit**, **__WIZCHIP::CRIS**, **wizchip_cris_enter()**, and **wizchip_cris_exit()**.

void reg_wizchip_cs_cbfunc (void(*)(void) cs_sel,
void(*)(void) cs_desel
)

Registers call back function for WIZCHIP select & deselect.

Parameters

cs_sel : callback function for WIZCHIP select
cs_desel : callback fucntion for WIZCHIP deselect

Todo:
Describe **wizchip_cs_select** and **wizchip_cs_deselect** function or register your functions.

Note
If you do not describe or register, null function is called.

Definition at line **200** of file **wizchip_conf.c**.

References **__WIZCHIP::_CS::_deselect**, **__WIZCHIP::_CS::_select**, **__WIZCHIP::CS**, **wizchip_cs_deselect()**, and **wizchip_cs_select()**.

---

void
reg_wizchip_bus_cbfunc (iodata_t(*)(uint32_t addr) bus_
void(*)(uint32_t addr, iodata_t wb) bus_
)

Registers call back function for bus interface.

Parameters

bus_rb : callback function to read byte data using system bus
bus_wb : callback function to write byte data using system bus

Todo:
Describe wizchip_bus_readbyte and wizchip_bus_writebyte functic or register your functions.

Note
If you do not describe or register, null function is called.

Definition at line **216** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_data**, **_WIZCHIP_IO_MODE_BU** **__WIZCHIP::_IF::_write_data**, **__WIZCHIP::_IF::BUS**, **__WIZCHIP::I** **__WIZCHIP::if_mode**, **wizchip_bus_readdata()**, and

wizchip_bus_writedata().

---

void reg_wizchip_spi_cbfunc (uint8_t(*)(void) spi_rb,
void(*)(uint8_t wb) spi_wb
)

Registers call back function for SPI interface.

Parameters

spi_rb : callback function to read byte using SPI
spi_wb : callback function to write byte using SPI

Todo:
Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte** function or register your functions.

Note
If you do not describe or register, null function is called.

Definition at line **244** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_byte**, **_WIZCHIP_IO_MODE_SPI_**, **__WIZCHIP::_IF::_write_byte**, **__WIZCHIP::IF**, **__WIZCHIP::if_mode**, **__WIZCHIP::_IF::SPI**, **wizchip_spi_readbyte()**, and **wizchip_spi_writebyte()**.

void
reg_wizchip_spiburst_cbfunc (void(*)(uint8_t *pBuf, uint16_t len)
void(*)(uint8_t *pBuf, uint16_t len)
)

Registers call back function for SPI interface.

Parameters

spi_rb : callback function to burst read using SPI
spi_wb : callback function to burst write using SPI

Todo:

Describe **wizchip_spi_readbyte** and **wizchip_spi_writebyte** func
register your functions.

Note

If you do not describe or register, null function is called.

Definition at line **261** of file **wizchip_conf.c**.

References **__WIZCHIP::_IF::_read_burst**, **_WIZCHIP_IO_MODE_SP**
__WIZCHIP::_IF::_write_burst, **__WIZCHIP::IF**, **__WIZCHIP::if_mod**
__WIZCHIP::_IF::SPI, **wizchip_spi_readburst()**, and
wizchip_spi_writeburst().

int8_t wizphy_getphylink (void)

get the link status of phy in WIZCHIP. No use in W5100

Definition at line **575** of file **wizchip_conf.c**.

References **getPHYCFGR**, **getPHYSTATUS**, **PHY_LINK_OFF**,
PHY_LINK_ON, **PHYCFGR_LNK_ON**, and **PHYSTATUS_LINK**.

Referenced by **ctlwizchip()**.

int8_t wizphy_getphypmode (void)

get the power mode of PHY in WIZCHIP. No use in W5100

Definition at line **596** of file **wizchip_conf.c**.

References **getPHYCFGR**, **getPHYSTATUS**,
PHY_POWER_DOWN, **PHY_POWER_NORM**,
PHYCFGR_OPMDC_PDOWN, and **PHYSTATUS_POWERDOWN**.

Referenced by **ctlwizchip()**.

void wizphy_reset (void)

Reset phy. Vailid only in W5500.

Definition at line **617** of file **wizchip_conf.c**.

References **getPHYCFGR**, **PHYCFGR_RST**, and **setPHYCFGR**.

Referenced by **ctlwizchip()**, **wizphy_setphyconf()**, and **wizphy_setphypmode()**.

Variable Documentation

_WIZCHIP WIZCHIP

\ref _WIZCHIP instance

Definition at line **165** of file **wizchip_conf.c**.

Socket APIs

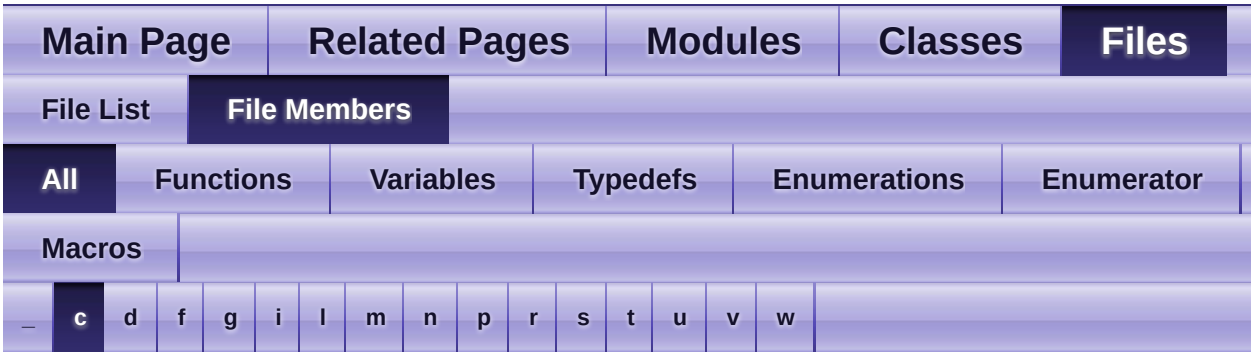
Here is a list of all file members with links to the files they belong to:

- _ -

- _IMR_ : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- _RCR_ : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- _RTR_ : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- _W5100_IO_BASE_ : **w5100.h**
- _W5200_SPI_READ_ : **w5200.h**
- _W5200_SPI_WRITE_ : **w5200.h**
- _W5300_IO_BASE_ : **w5300.h**
- _W5500_IO_BASE_ : **w5500.h**
- _W5500_SPI_FDM_OP_LEN1_ : **w5500.c**
- _W5500_SPI_FDM_OP_LEN2_ : **w5500.c**
- _W5500_SPI_FDM_OP_LEN4_ : **w5500.c**
- _W5500_SPI_READ_ : **w5500.h**
- _W5500_SPI_VDM_OP_ : **w5500.c**
- _W5500_SPI_WRITE_ : **w5500.h**
- _WIZCHIP : **wizchip_conf.h**
- _WIZCHIP_ : **wizchip_conf.h**
- _WIZCHIP_ID_ : **wizchip_conf.h**
- _WIZCHIP_IO_BASE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_DIR_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_INDIR_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_NONE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_SPI_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_SPI_FDM_ : **wizchip_conf.h**

- _WIZCHIP_IO_MODE_SPI_VDM_ : **wizchip_conf.h**
- _WIZCHIP_IO_RXBUF_ : **w5100.h** , **w5200.h**
- _WIZCHIP_IO_TXBUF_ : **w5100.h** , **w5200.h**
- _WIZCHIP_SN_BASE_ : **w5300.h** , **w5100.h** , **w5200.h**
- _WIZCHIP_SN_SIZE_ : **w5100.h** , **w5300.h** , **w5200.h**
- _WIZCHIP_SOCK_NUM_ : **wizchip_conf.h**

Socket APIs



Here is a list of all file members with links to the files they belong to:

- c -

- CHECK_SOCKDATA : **socket.c**
- CHECK_SOCKINIT : **socket.c**
- CHECK_SOCKMODE : **socket.c**
- CHECK_SOCKNUM : **socket.c**
- close() : **socket.c** , **socket.h**
- CN_GET_NETINFO : **wizchip_conf.h**
- CN_GET_NETMODE : **wizchip_conf.h**
- CN_GET_TIMEOUT : **wizchip_conf.h**
- CN_SET_NETINFO : **wizchip_conf.h**
- CN_SET_NETMODE : **wizchip_conf.h**
- CN_SET_TIMEOUT : **wizchip_conf.h**
- connect() : **socket.c** , **socket.h**
- CS_CLR_INTERRUPT : **socket.h**
- CS_GET_INTERRUPT : **socket.h**
- CS_GET_INTMASK : **socket.h**
- CS_GET_IOMODE : **socket.h**
- CS_GET_MAXRXBUF : **socket.h**
- CS_GET_MAXTXBUF : **socket.h**
- CS_SET_INTMASK : **socket.h**
- CS_SET_IOMODE : **socket.h**
- ctlnetwork() : **wizchip_conf.c** , **wizchip_conf.h**
- ctlnetwork_type : **wizchip_conf.h**
- ctlsock_type : **socket.h**
- ctlsocket() : **socket.c** , **socket.h**
- ctlwizchip() : **wizchip_conf.c** , **wizchip_conf.h**

- ctlwizchip_type : **wizchip_conf.h**
- CW_CLR_INTERRUPT : **wizchip_conf.h**
- CW_GET_ID : **wizchip_conf.h**
- CW_GET_INTERRUPT : **wizchip_conf.h**
- CW_GET_INTRMASK : **wizchip_conf.h**
- CW_GET_INTRTIME : **wizchip_conf.h**
- CW_GET_PHYCONF : **wizchip_conf.h**
- CW_GET_PHYLINK : **wizchip_conf.h**
- CW_GET_PHYPOWMODE : **wizchip_conf.h**
- CW_GET_PHYSTATUS : **wizchip_conf.h**
- CW_INIT_WIZCHIP : **wizchip_conf.h**
- CW_RESET_PHY : **wizchip_conf.h**
- CW_RESET_WIZCHIP : **wizchip_conf.h**
- CW_SET_INTRMASK : **wizchip_conf.h**
- CW_SET_INTRTIME : **wizchip_conf.h**
- CW_SET_PHYCONF : **wizchip_conf.h**
- CW_SET_PHYPOWMODE : **wizchip_conf.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- d -

- dhcp_mode : **wizchip_conf.h**
- disconnect() : **socket.h** , **socket.c**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- f -

- FMTUR : **w5300.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- g -

- GAR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- getFMTUR : **w5300.h**
- getGAR : **w5300.h** , **w5500.h** , **w5100.h** , **w5200.h**
- getIDR : **w5300.h**
- getIMR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getIMR2 : **w5200.h**
- getINTLEVEL : **w5200.h** , **w5500.h**
- getIR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getIR2 : **w5200.h**
- getMR : **w5100.h** , **w5200.h** , **w5500.h**
- getMTYPER : **w5300.h**
- getPATR : **w5100.h** , **w5200.h** , **w5300.h**
- getPDHAR : **w5300.h**
- getPHAR : **w5500.h**
- getPHYCFGR : **w5500.h**
- getPHYSTATUS : **w5200.h**
- getPMAGIC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getPMRU : **w5500.h**
- getPn_BDPTHR : **w5300.h**
- getPn_BRDYR : **w5300.h**
- getPPPALGO : **w5100.h** , **w5200.h**
- getPSID : **w5500.h**
- getPSIDR : **w5300.h**
- getPTIMER : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getRCR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**

- getRMS01R : **w5300.h**
- getRMS23R : **w5300.h**
- getRMS45R : **w5300.h**
- getRMS67R : **w5300.h**
- getRMSR : **w5100.h** , **w5300.h**
- getRTR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSHAR : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- getSIMR : **w5200.h** , **w5500.h**
- getSIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSIR : **w5200.h** , **w5500.h**
- getSn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DPORTR : **w5300.h**
- getSn_FRAG : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_FRAGR : **w5300.h**
- getSn_IMR : **w5200.h** , **w5300.h** , **w5500.h**
- getSn_IR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_KPALVTR : **w5300.h** , **w5500.h**
- getSn_MR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_MSSR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_PORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_PORTR : **w5300.h**
- getSn_PROTO : **w5100.h** , **w5200.h** , **w5300.h**
- getSn_PROTOR : **w5300.h**
- getSn_RX_FIFOR : **w5300.h**
- getSn_RX_RD : **w5500.h** , **w5100.h** , **w5200.h**
- getSn_RX_RSR() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_RX_WR : **w5100.h** , **w5200.h** , **w5500.h**
- getSn_RxBASE() : **w5200.h** , **w5100.h**
- getSn_RXBUF_SIZE : **w5300.h** , **w5100.h** , **w5200.h** , **w5500.h**
- getSn_RxMASK : **w5200.h** , **w5100.h**
- getSn_RxMAX : **w5500.h** , **w5200.h** , **w5300.h** , **w5100.h**
- getSn_RXMEM_SIZE : **w5200.h** , **w5100.h**
- getSn_SR : **w5500.h** , **w5300.h** , **w5200.h** , **w5100.h**
- getSn_SSR : **w5300.h**
- getSn_TOS : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- getSn_TOSR : **w5300.h**
- getSn_TTL : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**

- getSn_TTLR : **w5300.h**
- getSn_TX_FSR() : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- getSn_TX_RD : **w5200.h** , **w5100.h** , **w5500.h**
- getSn_TX_WR : **w5500.h** , **w5100.h** , **w5200.h**
- getSn_TX_WRSR : **w5300.h**
- getSn_TxBASE() : **w5100.h** , **w5200.h**
- getSn_TXBUF_SIZE : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- getSn_TxMASK : **w5100.h** , **w5200.h**
- getSn_TxMAX : **w5200.h** , **w5100.h** , **w5500.h** , **w5300.h**
- getSn_TXMEM_SIZE : **w5100.h** , **w5200.h**
- getsockopt() : **socket.h** , **socket.c**
- getSUBR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getTMS01R : **w5300.h**
- getTMS23R : **w5300.h**
- getTMS45R : **w5300.h**
- getTMS67R : **w5300.h**
- getTMSR() : **w5300.h**
- getUIPR : **w5300.h** , **w5500.h**
- getUPORTR : **w5300.h** , **w5500.h**
- getVERSIONR : **w5200.h** , **w5500.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- i -

- IDR : **w5300.h**
- IINCHIP_READ : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- IINCHIP_READ_BUF : **w5500.h** , **w5100.h** , **w5200.h**
- IINCHIP_WRITE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IINCHIP_WRITE_BUF : **w5100.h** , **w5200.h** , **w5500.h**
- IK_DEST_UNREACH : **wizchip_conf.h**
- IK_IP_CONFLICT : **wizchip_conf.h**
- IK_PPPOE_TERMINATED : **wizchip_conf.h**
- IK_SOCK_0 : **wizchip_conf.h**
- IK_SOCK_1 : **wizchip_conf.h**
- IK_SOCK_2 : **wizchip_conf.h**
- IK_SOCK_3 : **wizchip_conf.h**
- IK_SOCK_4 : **wizchip_conf.h**
- IK_SOCK_5 : **wizchip_conf.h**
- IK_SOCK_6 : **wizchip_conf.h**
- IK_SOCK_7 : **wizchip_conf.h**
- IK_SOCK_ALL : **wizchip_conf.h**
- IK_WOL : **wizchip_conf.h**
- IM_IR4 : **w5500.h**
- IM_IR5 : **w5500.h**
- IM_IR6 : **w5500.h**
- IM_IR7 : **w5500.h**
- IMR2 : **w5200.h**
- INTLEVEL : **w5200.h** , **w5500.h**
- intr_kind : **wizchip_conf.h**

- iodata_t : **wizchip_conf.h**
- IPPROTO_GGP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_ICMP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_IDP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_IGMP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_IP : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- IPPROTO_ND : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- IPPROTO_PUP : **w5300.h** , **w5500.h** , **w5200.h** , **w5100.h**
- IPPROTO_RAW : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_TCP : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- IPPROTO_UDP : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- IR : **w5200.h** , **w5500.h** , **w5300.h** , **w5100.h**
- IR2 : **w5200.h**
- IR_CONFLICT : **w5100.h** , **w5500.h** , **w5200.h**
- IR_DPUR : **w5300.h**
- IR_FMTU : **w5300.h**
- IR_IPCF : **w5300.h**
- IR_MP : **w5500.h**
- IR_PPPoE : **w5100.h** , **w5200.h** , **w5500.h**
- IR_PPPT : **w5300.h**
- IR_SnINT : **w5300.h**
- IR_SOCK : **w5100.h**
- IR_UNREACH : **w5100.h** , **w5500.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- l -

- listen() : **socket.c** , **socket.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- m -

- MR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- MR_AI : **w5100.h** , **w5200.h**
- MR_DBS : **w5300.h**
- MR_DBW : **w5300.h**
- MR_FARP : **w5500.h**
- MR_FS : **w5300.h**
- MR_IND : **w5100.h** , **w5200.h** , **w5300.h**
- MR_MPF : **w5300.h**
- MR_MT : **w5300.h**
- MR_PB : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- MR_PPPOE : **w5500.h**
- MR_PPPoE : **w5300.h**
- MR_PPPOE : **w5100.h** , **w5200.h**
- MR_RDH : **w5300.h**
- MR_RST : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- MR_WDF : **w5300.h**
- MR_WOL : **w5200.h** , **w5500.h**
- MTYPER : **w5300.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- n -

- NETINFO_DHCP : **wizchip_conf.h**
- NETINFO_STATIC : **wizchip_conf.h**
- netmode_type : **wizchip_conf.h**
- NM_FORCEARP : **wizchip_conf.h**
- NM_PINGBLOCK : **wizchip_conf.h**
- NM_PPPOE : **wizchip_conf.h**
- NM_WAKEONLAN : **wizchip_conf.h**

Socket APIs

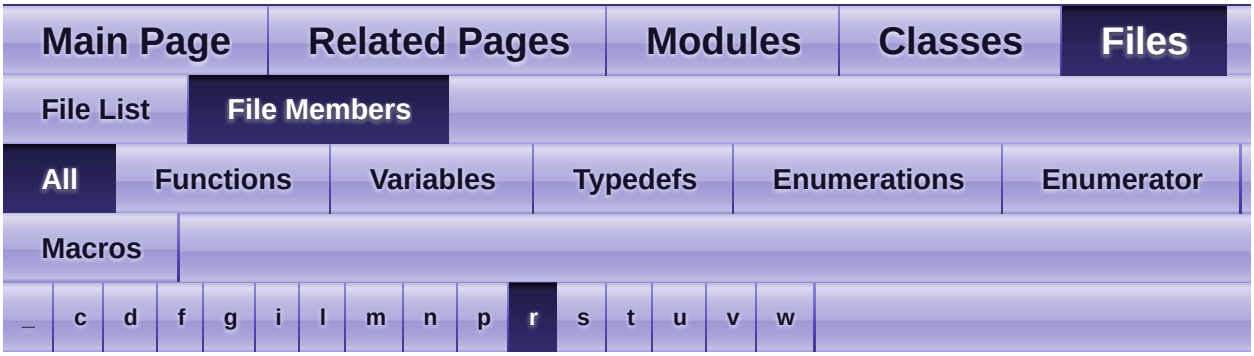
Here is a list of all file members with links to the files they belong to:

- p -

- PACK_COMPLETED : **socket.h**
- PACK_FIFOBYTE : **socket.h**
- PACK_FIRST : **socket.h**
- PACK_REMAINED : **socket.h**
- PATR : **w5100.h** , **w5200.h** , **w5300.h**
- PDHAR : **w5300.h**
- PHAR : **w5500.h**
- PHY_CONFBY_HW : **wizchip_conf.h**
- PHY_CONFBY_SW : **wizchip_conf.h**
- PHY_DUPLEX_FULL : **wizchip_conf.h**
- PHY_DUPLEX_HALF : **wizchip_conf.h**
- PHY_LINK_OFF : **wizchip_conf.h**
- PHY_LINK_ON : **wizchip_conf.h**
- PHY_MODE_AUTONEGO : **wizchip_conf.h**
- PHY_MODE_MANUAL : **wizchip_conf.h**
- PHY_POWER_DOWN : **wizchip_conf.h**
- PHY_POWER_NORM : **wizchip_conf.h**
- PHY_SPEED_10 : **wizchip_conf.h**
- PHY_SPEED_100 : **wizchip_conf.h**
- PHYCFGR : **w5500.h**
- PHYCFGR_DPX_FULL : **w5500.h**
- PHYCFGR_DPX_HALF : **w5500.h**
- PHYCFGR_LNK_OFF : **w5500.h**
- PHYCFGR_LNK_ON : **w5500.h**
- PHYCFGR_OPMD : **w5500.h**

- PHYCFGR_OPMDC_100F : **w5500.h**
- PHYCFGR_OPMDC_100FA : **w5500.h**
- PHYCFGR_OPMDC_100H : **w5500.h**
- PHYCFGR_OPMDC_10F : **w5500.h**
- PHYCFGR_OPMDC_10H : **w5500.h**
- PHYCFGR_OPMDC_ALLA : **w5500.h**
- PHYCFGR_OPMDC_NA : **w5500.h**
- PHYCFGR_OPMDC_PDOWN : **w5500.h**
- PHYCFGR_RST : **w5500.h**
- PHYCFGR_SPD_10 : **w5500.h**
- PHYCFGR_SPD_100 : **w5500.h**
- PHYSTATUS : **w5200.h**
- PHYSTATUS_LINK : **w5200.h**
- PHYSTATUS_POWERDOWN : **w5200.h**
- PHYSTATUS_POWERSAVE : **w5200.h**
- PMAGIC : **w5200.h** , **w5100.h** , **w5500.h**
- PMAGICR : **w5300.h**
- PMRU : **w5500.h**
- Pn_BDPTHR : **w5300.h**
- Pn_BRDYR : **w5300.h**
- Pn_MT : **w5300.h**
- Pn_PEN : **w5300.h**
- Pn_PPL : **w5300.h**
- Pn_SN : **w5300.h**
- PPPALGO : **w5200.h**
- PSID : **w5500.h**
- PSIDR : **w5300.h**
- PTIMER : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**

Socket APIs



Here is a list of all file members with links to the files they belong to:

- r -

- recv() : **socket.c** , **socket.h**
- recvfrom() : **socket.h** , **socket.c**
- reg_wizchip_bus_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_cris_cbfunc() : **wizchip_conf.h** , **wizchip_conf.c**
- reg_wizchip_cs_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_spi_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_spiburst_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- RMS01R : **w5300.h**
- RMS23R : **w5300.h**
- RMS45R : **w5300.h**
- RMS67R : **w5300.h**
- RMSR : **w5100.h**
- RMSR0 : **w5300.h**
- RMSR1 : **w5300.h**
- RMSR2 : **w5300.h**
- RMSR3 : **w5300.h**
- RMSR4 : **w5300.h**
- RMSR5 : **w5300.h**
- RMSR6 : **w5300.h**
- RMSR7 : **w5300.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- s -

- send() : **socket.c** , **socket.h**
- sendto() : **socket.h** , **socket.c**
- setGAR : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- setIMR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setIMR2 : **w5200.h**
- setINTLEVEL : **w5200.h** , **w5500.h**
- setIR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setIR2 : **w5200.h**
- setMR : **w5100.h** , **w5200.h** , **w5500.h**
- setMTYPER : **w5300.h**
- setPHAR : **w5500.h**
- setPHYCFGR : **w5500.h**
- setPMAGIC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setPMRU : **w5500.h**
- setPn_BDPTHR : **w5300.h**
- setPn_BRDYR : **w5300.h**
- setPSID : **w5500.h**
- setPTIMER : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setRCR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setRMS01R : **w5300.h**
- setRMS23R : **w5300.h**
- setRMS45R : **w5300.h**
- setRMS67R : **w5300.h**
- setRMSR : **w5100.h** , **w5300.h**
- setRTR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**

- setSHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSIMR : **w5200.h** , **w5500.h**
- setSIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSIR : **w5500.h** , **w5200.h**
- setSn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DPORTR : **w5300.h**
- setSn_FRAG : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_FRAGR : **w5300.h**
- setSn_IMR : **w5200.h** , **w5300.h** , **w5500.h**
- setSn_IR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_KPALVTR : **w5300.h** , **w5500.h**
- setSn_MR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_MSSR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_PORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_PORTR : **w5300.h**
- setSn_PROTO : **w5100.h** , **w5200.h** , **w5300.h**
- setSn_PROTOR : **w5300.h**
- setSn_RX_RD : **w5100.h** , **w5200.h** , **w5500.h**
- setSn_RX_WR : **w5100.h** , **w5200.h**
- setSn_RXBUF_SIZE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_RXMEM_SIZE : **w5100.h** , **w5200.h**
- setSn_TOS : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- setSn_TOSR : **w5300.h**
- setSn_TTL : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_TTLR : **w5300.h**
- setSn_TX_FIFOR : **w5300.h**
- setSn_TX_WR : **w5100.h** , **w5200.h** , **w5500.h**
- setSn_TX_WRSR : **w5300.h**
- setSn_TXBUF_SIZE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_TXMEM_SIZE : **w5100.h** , **w5200.h**
- setsockopt() : **socket.c** , **socket.h**
- setSUBR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setTMS01R : **w5300.h**
- setTMS23R : **w5300.h**
- setTMS45R : **w5300.h**
- setTMS67R : **w5300.h**
- setTMSR : **w5100.h** , **w5300.h**

- SF_BROAD_BLOCK : **socket.h**
- SF_ETHER_OWN : **socket.h**
- SF_IGMP_VER2 : **socket.h**
- SF_IO_NONBLOCK : **socket.h**
- SF_IPv6_BLOCK : **socket.h**
- SF_MULTI_BLOCK : **socket.h**
- SF_MULTI_ENABLE : **socket.h**
- SF_TCP_NODELAY : **socket.h**
- SF_UNI_BLOCK : **socket.h**
- SHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SIK_ALL : **socket.h**
- SIK_CONNECTED : **socket.h**
- SIK_DISCONNECTED : **socket.h**
- SIK_RECEIVED : **socket.h**
- SIK_SENT : **socket.h**
- SIK_TIMEOUT : **socket.h**
- SIMR : **w5500.h**
- SIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SIR : **w5500.h**
- Sn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_CLOSE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_CONNECT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_DISCON : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_LISTEN : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_OPEN : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_PCJ : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCN : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCON : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCR : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PDISCON : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_RECV : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND_KEEP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND_MAC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DPORTR : **w5300.h**
- Sn_FRAG : **w5200.h** , **w5300.h** , **w5500.h**
- Sn_FRAGR : **w5300.h**

- Sn_IMR : **w5300.h** , **w5500.h** , **w5200.h**
- Sn_IR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_IR_CON : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- Sn_IR_DISCON : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- Sn_IR_PFAIL : **w5100.h** , **w5300.h** , **w5200.h**
- Sn_IR_PNEXT : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_PRECV : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_RECV : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_IR_SENDOK : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_IR_TIMEOUT : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_KPALVTR : **w5300.h** , **w5500.h**
- Sn_MR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_MR_ALIGN : **w5300.h**
- Sn_MR_BCASTB : **w5500.h**
- Sn_MR_CLOSE : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_MR_IGMPv : **w5300.h**
- Sn_MR_IPRAW : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_MR_MACRAW : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_MR_MC : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_MR_MF : **w5200.h** , **w5300.h** , **w5100.h**
- Sn_MR_MFEN : **w5100.h** , **w5500.h** , **w5200.h**
- Sn_MR_MIP6B : **w5500.h**
- Sn_MR_MMB : **w5500.h**
- Sn_MR_MULTI : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_MR_ND : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_MR_PPPoE : **w5300.h**
- Sn_MR_PPPOE : **w5200.h**
- Sn_MR_PPPoE : **w5100.h**
- Sn_MR_TCP : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- Sn_MR_UCASTB : **w5500.h**
- Sn_MR_UDP : **w5300.h** , **w5500.h** , **w5100.h** , **w5200.h**
- Sn_MSSR : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- Sn_PORT : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- Sn_PORTR : **w5300.h**
- Sn_PROTO : **w5100.h** , **w5200.h**
- Sn_PROTOR : **w5300.h**
- Sn_RX_FIFOR : **w5300.h**
- Sn_RX_RD : **w5200.h** , **w5500.h** , **w5100.h**
- Sn_RX_RSR : **w5300.h** , **w5500.h** , **w5200.h** , **w5100.h**
- Sn_RX_WR : **w5200.h** , **w5100.h** , **w5500.h**

- Sn_RXBUF_SIZE : **w5500.h**
- Sn_RXMEM_SIZE : **w5200.h**
- Sn_SR : **w5300.h** , **w5100.h** , **w5200.h** , **w5500.h**
- Sn_SSR : **w5300.h**
- Sn_TOS : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_TOSR : **w5300.h**
- Sn_TTL : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- Sn_TTLR : **w5300.h**
- Sn_TX_FIFOR : **w5300.h**
- Sn_TX_FSR : **w5300.h** , **w5100.h** , **w5200.h** , **w5500.h**
- Sn_TX_RD : **w5200.h** , **w5500.h** , **w5100.h**
- Sn_TX_WR : **w5200.h** , **w5500.h** , **w5100.h**
- Sn_TX_WRSR : **w5300.h**
- Sn_TXBUF_SIZE : **w5500.h**
- Sn_TXMEM_SIZE : **w5200.h**
- SO_DESTIP : **socket.h**
- SO_DESTPORT : **socket.h**
- SO_FLAG : **socket.h**
- SO_KEEPALIVEAUTO : **socket.h**
- SO_KEEPALIVESEND : **socket.h**
- SO_MSS : **socket.h**
- SO_PACKINFO : **socket.h**
- SO_RECVBUF : **socket.h**
- SO_REMAINSIZE : **socket.h**
- SO_SENDBUF : **socket.h**
- SO_STATUS : **socket.h**
- SO_TOS : **socket.h**
- SO_TTL : **socket.h**
- SOCK_ANY_PORT_NUM : **socket.c**
- SOCK_ARP : **w5300.h**
- SOCK_BUSY : **socket.h**
- SOCK_CLOSE_WAIT : **w5500.h** , **w5200.h** , **w5100.h** , **w5300.h**
- SOCK_CLOSED : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- SOCK_CLOSING : **w5200.h** , **w5100.h** , **w5500.h** , **w5300.h**
- SOCK_DGRAM : **w5300.h** , **w5500.h**
- SOCK_ERROR : **socket.h**
- SOCK_ESTABLISHED : **w5200.h** , **w5500.h** , **w5100.h** , **w5300.h**
- SOCK_FATAL : **socket.h**
- SOCK_FIN_WAIT : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- SOCK_INIT : **w5100.h** , **w5500.h** , **w5300.h** , **w5200.h**

- SOCK_IO_BLOCK : **socket.h**
- SOCK_IO_NONBLOCK : **socket.h**
- SOCK_IPRAW : **w5100.h** , **w5200.h** , **w5300.h**
- SOCK_LAST_ACK : **w5200.h** , **w5300.h** , **w5100.h** , **w5500.h**
- SOCK_LISTEN : **w5500.h** , **w5200.h** , **w5300.h** , **w5100.h**
- SOCK_MACRAW : **w5500.h** , **w5100.h** , **w5300.h** , **w5200.h**
- SOCK_OK : **socket.h**
- sock_pack_info : **socket.c**
- SOCK_PPPOE : **w5100.h**
- SOCK_PPPoE : **w5300.h**
- SOCK_PPPOE : **w5200.h**
- SOCK_STREAM : **w5500.h** , **w5300.h**
- SOCK_SYNRECV : **w5200.h** , **w5500.h** , **w5100.h** , **w5300.h**
- SOCK_SYNSENT : **w5300.h** , **w5500.h** , **w5200.h** , **w5100.h**
- SOCK_TIME_WAIT : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- SOCK_UDP : **w5500.h** , **w5200.h** , **w5100.h** , **w5300.h**
- SOCKERR_ARG : **socket.h**
- SOCKERR_BUFFER : **socket.h**
- SOCKERR_DATALEN : **socket.h**
- SOCKERR_IPINVALID : **socket.h**
- SOCKERR_PORTZERO : **socket.h**
- SOCKERR_SOCKCLOSED : **socket.h**
- SOCKERR_SOCKFLAG : **socket.h**
- SOCKERR_SOCKINIT : **socket.h**
- SOCKERR_SOCKMODE : **socket.h**
- SOCKERR_SOCKNUM : **socket.h**
- SOCKERR_SOCKOPT : **socket.h**
- SOCKERR_SOCKSTATUS : **socket.h**
- SOCKERR_TIMEOUT : **socket.h**
- socket() : **socket.h** , **socket.c**
- SOCKET : **socket.h**
- SOCKFATAL_PACKLEN : **socket.h**
- sockint_kind : **socket.h**
- sockopt_type : **socket.h**
- SUBR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- t -

- TMS01R : **w5300.h**
- TMS23R : **w5300.h**
- TMS45R : **w5300.h**
- TMS67R : **w5300.h**
- TMSR : **w5100.h**
- TMSR0 : **w5300.h**
- TMSR1 : **w5300.h**
- TMSR2 : **w5300.h**
- TMSR3 : **w5300.h**
- TMSR4 : **w5300.h**
- TMSR5 : **w5300.h**
- TMSR6 : **w5300.h**
- TMSR7 : **w5300.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- u -

- UIPR : **w5300.h** , **w5500.h**
- UIPR0 : **w5100.h**
- UPORT0 : **w5100.h**
- UPORTR : **w5300.h** , **w5500.h**

Socket APIs

Here is a list of all file members with links to the files they belong to:

- v -

- VERSIONR : **w5200.h** , **w5500.h** , **w5300.h**

Socket APIs

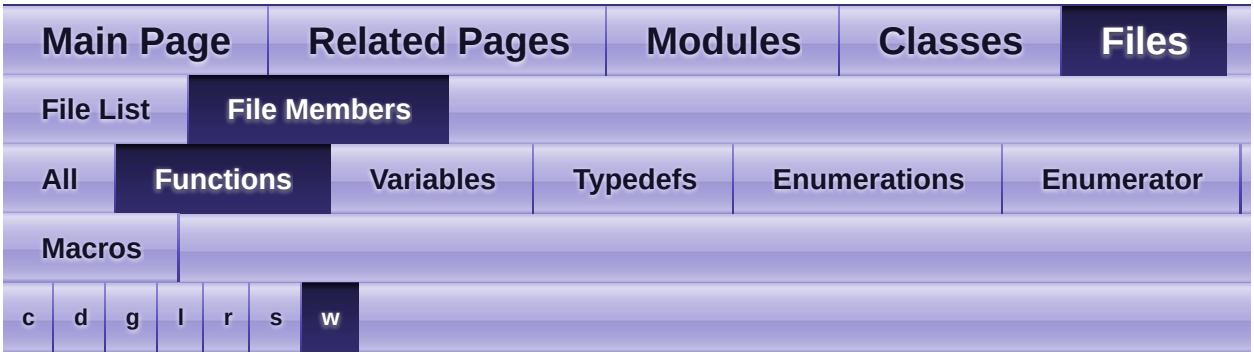
Here is a list of all file members with links to the files they belong to:

- w -

- wiz_NetInfo : **wizchip_conf.h**
- wiz_NetTimeout : **wizchip_conf.h**
- wiz_PhyConf : **wizchip_conf.h**
- wiz_recv_data() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- wiz_recv_ignore() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- wiz_send_data() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- WIZCHIP : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_bus_readdata() : **wizchip_conf.c**
- wizchip_bus_writedata() : **wizchip_conf.c**
- wizchip_clrinterrupt() : **wizchip_conf.c** , **wizchip_conf.h**
- WIZCHIP_CREG_BLOCK : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- wizchip_cris_enter() : **wizchip_conf.c**
- wizchip_cris_exit() : **wizchip_conf.c**
- WIZCHIP_CRITICAL_ENTER : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- WIZCHIP_CRITICAL_EXIT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- wizchip_cs_deselect() : **wizchip_conf.c**
- wizchip_cs_select() : **wizchip_conf.c**
- wizchip_getinterrupt() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getinterruptmask() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getnetinfo() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getnetmode() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_gettimeout() : **wizchip_conf.c** , **wizchip_conf.h**

- wizchip_init() : **wizchip_conf.c** , **wizchip_conf.h**
- WIZCHIP_OFFSET_INC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- WIZCHIP_READ() : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- WIZCHIP_READ_BUF() : **w5200.h** , **w5100.h** , **w5500.h**
- WIZCHIP_RXBUF_BLOCK : **w5500.h**
- wizchip_setinterruptmask() : **wizchip_conf.h** , **wizchip_conf.c**
- wizchip_setnetinfo() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_setnetmode() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_settimeout() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_spi_readburst() : **wizchip_conf.c**
- wizchip_spi_readbyte() : **wizchip_conf.c**
- wizchip_spi_writeburst() : **wizchip_conf.c**
- wizchip_spi_writebyte() : **wizchip_conf.c**
- WIZCHIP_SREG_BLOCK : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- wizchip_sw_reset() : **wizchip_conf.h** , **wizchip_conf.c**
- WIZCHIP_TXBUF_BLOCK : **w5500.h**
- WIZCHIP_WRITE() : **w5200.h** , **w5100.h** , **w5500.h** , **w5300.h**
- WIZCHIP_WRITE_BUF() : **w5500.h** , **w5100.h** , **w5200.h**
- wizphy_getphyconf() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_getphylink() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_getphypmode() : **wizchip_conf.c** , **wizchip_conf.h**
- wizphy_getphystat() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_reset() : **wizchip_conf.c** , **wizchip_conf.h**
- wizphy_setphyconf() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_setphypmode() : **wizchip_conf.h** , **wizchip_conf.c**

Socket APIs



- c -

- close() : **socket.c** , **socket.h**
- connect() : **socket.h** , **socket.c**
- ctlnetwork() : **wizchip_conf.c** , **wizchip_conf.h**
- ctlsocket() : **socket.c** , **socket.h**
- ctlwizchip() : **wizchip_conf.h** , **wizchip_conf.c**

- d -

- disconnect() : **socket.c** , **socket.h**

- g -

- getRMSR() : **w5300.h**
- getSn_RX_RSR() : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- getSn_RxBASE() : **w5100.h** , **w5200.h**
- getSn_TX_FSR() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_TxBASE() : **w5200.h** , **w5100.h**
- getsockopt() : **socket.h** , **socket.c**
- getTMSR() : **w5300.h**

- l -

- listen() : **socket.c** , **socket.h**

- r -

- recv() : **socket.c** , **socket.h**
- recvfrom() : **socket.h** , **socket.c**
- reg_wizchip_bus_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_cris_cbfunc() : **wizchip_conf.h** , **wizchip_conf.c**
- reg_wizchip_cs_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_spi_cbfunc() : **wizchip_conf.c** , **wizchip_conf.h**
- reg_wizchip_spiburst_cbfunc() : **wizchip_conf.h** , **wizchip_conf.c**

- s -

- send() : **socket.c** , **socket.h**
- sendto() : **socket.h** , **socket.c**
- setRMSR() : **w5300.h**
- setsockopt() : **socket.h** , **socket.c**
- setTMSR() : **w5300.h**
- socket() : **socket.c** , **socket.h**

- w -

- wiz_recv_data() : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- wiz_recv_ignore() : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- wiz_send_data() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- wizchip_bus_readdata() : **wizchip_conf.c**
- wizchip_bus_writedata() : **wizchip_conf.c**
- wizchip_clrinterrupt() : **wizchip_conf.h** , **wizchip_conf.c**
- wizchip_cris_enter() : **wizchip_conf.c**
- wizchip_cris_exit() : **wizchip_conf.c**
- wizchip_cs_deselect() : **wizchip_conf.c**
- wizchip_cs_select() : **wizchip_conf.c**
- wizchip_getinterrupt() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getinterruptmask() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getnetinfo() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_getnetmode() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_gettimeout() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_init() : **wizchip_conf.h** , **wizchip_conf.c**
- WIZCHIP_READ() : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- WIZCHIP_READ_BUF() : **w5100.h** , **w5200.h** , **w5500.h**
- wizchip_setinterruptmask() : **wizchip_conf.c** , **wizchip_conf.h**

- wizchip_setnetinfo() : **wizchip_conf.h** , **wizchip_conf.c**
- wizchip_setnetmode() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_settimeout() : **wizchip_conf.c** , **wizchip_conf.h**
- wizchip_spi_readburst() : **wizchip_conf.c**
- wizchip_spi_readbyte() : **wizchip_conf.c**
- wizchip_spi_writeburst() : **wizchip_conf.c**
- wizchip_spi_writebyte() : **wizchip_conf.c**
- wizchip_sw_reset() : **wizchip_conf.h** , **wizchip_conf.c**
- WIZCHIP_WRITE() : **w5300.h** , **w5100.h** , **w5500.h** , **w5200.h**
- WIZCHIP_WRITE_BUF() : **w5200.h** , **w5100.h** , **w5500.h**
- wizphy_getphyconf() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_getphylink() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_getphypmode() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_getphystat() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_reset() : **wizchip_conf.h** , **wizchip_conf.c**
- wizphy_setphyconf() : **wizchip_conf.c** , **wizchip_conf.h**
- wizphy_setphypmode() : **wizchip_conf.c** , **wizchip_conf.h**

Socket APIs

- sock_pack_info : **socket.c**
- WIZCHIP : **wizchip_conf.c** , **wizchip_conf.h**

Socket APIs

- _WIZCHIP : **wizchip_conf.h**
- iodata_t : **wizchip_conf.h**
- wiz_NetInfo : **wizchip_conf.h**
- wiz_NetTimeout : **wizchip_conf.h**
- wiz_PhyConf : **wizchip_conf.h**

Socket APIs

- ctlnetwork_type : **wizchip_conf.h**
- ctlsock_type : **socket.h**
- ctlwizchip_type : **wizchip_conf.h**
- dhcp_mode : **wizchip_conf.h**
- intr_kind : **wizchip_conf.h**
- netmode_type : **wizchip_conf.h**
- sockint_kind : **socket.h**
- sockopt_type : **socket.h**

Socket APIs

- c -

- CN_GET_NETINFO : **wizchip_conf.h**
- CN_GET_NETMODE : **wizchip_conf.h**
- CN_GET_TIMEOUT : **wizchip_conf.h**
- CN_SET_NETINFO : **wizchip_conf.h**
- CN_SET_NETMODE : **wizchip_conf.h**
- CN_SET_TIMEOUT : **wizchip_conf.h**
- CS_CLR_INTERRUPT : **socket.h**
- CS_GET_INTERRUPT : **socket.h**
- CS_GET_INTMASK : **socket.h**
- CS_GET_IOMODE : **socket.h**
- CS_GET_MAXRXBUF : **socket.h**
- CS_GET_MAXTXBUF : **socket.h**
- CS_SET_INTMASK : **socket.h**
- CS_SET_IOMODE : **socket.h**
- CW_CLR_INTERRUPT : **wizchip_conf.h**
- CW_GET_ID : **wizchip_conf.h**
- CW_GET_INTERRUPT : **wizchip_conf.h**
- CW_GET_INTRMASK : **wizchip_conf.h**
- CW_GET_INTRTIME : **wizchip_conf.h**
- CW_GET_PHYCONF : **wizchip_conf.h**
- CW_GET_PHYLINK : **wizchip_conf.h**
- CW_GET_PHYPOWMODE : **wizchip_conf.h**
- CW_GET_PHYSTATUS : **wizchip_conf.h**
- CW_INIT_WIZCHIP : **wizchip_conf.h**
- CW_RESET_PHY : **wizchip_conf.h**

- CW_RESET_WIZCHIP : **wizchip_conf.h**
- CW_SET_INTRMASK : **wizchip_conf.h**
- CW_SET_INTRTIME : **wizchip_conf.h**
- CW_SET_PHYCONF : **wizchip_conf.h**
- CW_SET_PHYPOWMODE : **wizchip_conf.h**

- i -

- IK_DEST_UNREACH : **wizchip_conf.h**
- IK_IP_CONFLICT : **wizchip_conf.h**
- IK_PPPOE_TERMINATED : **wizchip_conf.h**
- IK_SOCK_0 : **wizchip_conf.h**
- IK_SOCK_1 : **wizchip_conf.h**
- IK_SOCK_2 : **wizchip_conf.h**
- IK_SOCK_3 : **wizchip_conf.h**
- IK_SOCK_4 : **wizchip_conf.h**
- IK_SOCK_5 : **wizchip_conf.h**
- IK_SOCK_6 : **wizchip_conf.h**
- IK_SOCK_7 : **wizchip_conf.h**
- IK_SOCK_ALL : **wizchip_conf.h**
- IK_WOL : **wizchip_conf.h**

- n -

- NETINFO_DHCP : **wizchip_conf.h**
- NETINFO_STATIC : **wizchip_conf.h**
- NM_FORCEARP : **wizchip_conf.h**
- NM_PINGBLOCK : **wizchip_conf.h**
- NM_PPPOE : **wizchip_conf.h**
- NM_WAKEONLAN : **wizchip_conf.h**

- s -

- SIK_ALL : **socket.h**
- SIK_CONNECTED : **socket.h**
- SIK_DISCONNECTED : **socket.h**
- SIK_RECEIVED : **socket.h**
- SIK_SENT : **socket.h**
- SIK_TIMEOUT : **socket.h**
- SO_DESTIP : **socket.h**

- SO_DESTPORT : **socket.h**
- SO_FLAG : **socket.h**
- SO_KEEPALIVEAUTO : **socket.h**
- SO_KEEPALIVESEND : **socket.h**
- SO_MSS : **socket.h**
- SO_PACKINFO : **socket.h**
- SO_RECVBUF : **socket.h**
- SO_REMAINSIZE : **socket.h**
- SO_SENDBUF : **socket.h**
- SO_STATUS : **socket.h**
- SO_TOS : **socket.h**
- SO_TTL : **socket.h**

Socket APIs

- _ -

- _IMR_ : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- _RCR_ : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- _RTR_ : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- _W5100_IO_BASE_ : **w5100.h**
- _W5200_SPI_READ_ : **w5200.h**
- _W5200_SPI_WRITE_ : **w5200.h**
- _W5300_IO_BASE_ : **w5300.h**
- _W5500_IO_BASE_ : **w5500.h**
- _W5500_SPI_FDM_OP_LEN1_ : **w5500.c**
- _W5500_SPI_FDM_OP_LEN2_ : **w5500.c**
- _W5500_SPI_FDM_OP_LEN4_ : **w5500.c**
- _W5500_SPI_READ_ : **w5500.h**
- _W5500_SPI_VDM_OP_ : **w5500.c**
- _W5500_SPI_WRITE_ : **w5500.h**
- _WIZCHIP_ : **wizchip_conf.h**
- _WIZCHIP_ID_ : **wizchip_conf.h**
- _WIZCHIP_IO_BASE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_DIR_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_BUS_INDIR_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_NONE_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_SPI_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_SPI_FDM_ : **wizchip_conf.h**
- _WIZCHIP_IO_MODE_SPI_VDM_ : **wizchip_conf.h**

- _WIZCHIP_IO_RXBUF_ : **w5200.h** , **w5100.h**
- _WIZCHIP_IO_TXBUF_ : **w5100.h** , **w5200.h**
- _WIZCHIP_SN_BASE_ : **w5200.h** , **w5300.h** , **w5100.h**
- _WIZCHIP_SN_SIZE_ : **w5300.h** , **w5100.h** , **w5200.h**
- _WIZCHIP_SOCK_NUM_ : **wizchip_conf.h**

Socket APIs

- c -

- CHECK_SOCKDATA : **socket.c**
- CHECK_SOCKINIT : **socket.c**
- CHECK_SOCKMODE : **socket.c**
- CHECK_SOCKNUM : **socket.c**

Socket APIs

- f -

- FMTUR : **w5300.h**

Socket APIs

- g -

- GAR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- getFMTUR : **w5300.h**
- getGAR : **w5300.h** , **w5500.h** , **w5100.h** , **w5200.h**
- getIDR : **w5300.h**
- getIMR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getIMR2 : **w5200.h**
- getINTLEVEL : **w5200.h** , **w5500.h**
- getIR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getIR2 : **w5200.h**
- getMR : **w5100.h** , **w5200.h** , **w5500.h**
- getMTYPER : **w5300.h**
- getPATR : **w5100.h** , **w5200.h** , **w5300.h**
- getPDHAR : **w5300.h**
- getPHAR : **w5500.h**
- getPHYCFGR : **w5500.h**
- getPHYSTATUS : **w5200.h**
- getPMAGIC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getPMRU : **w5500.h**
- getPn_BDPTHR : **w5300.h**
- getPn_BRDYR : **w5300.h**
- getPPPALGO : **w5100.h** , **w5200.h**
- getPSID : **w5500.h**
- getPSIDR : **w5300.h**
- getPTIMER : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getRCR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**

- getRMS01R : **w5300.h**
- getRMS23R : **w5300.h**
- getRMS45R : **w5300.h**
- getRMS67R : **w5300.h**
- getRMSR : **w5100.h**
- getRTR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSHAR : **w5300.h** , **w5500.h** , **w5100.h** , **w5200.h**
- getSIMR : **w5200.h** , **w5500.h**
- getSIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSIR : **w5200.h** , **w5500.h**
- getSn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_DPORTR : **w5300.h**
- getSn_FRAG : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_FRAGR : **w5300.h**
- getSn_IMR : **w5200.h** , **w5300.h** , **w5500.h**
- getSn_IR : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- getSn_KPALVTR : **w5300.h** , **w5500.h**
- getSn_MR : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- getSn_MSSR : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- getSn_PORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- getSn_PORTR : **w5300.h**
- getSn_PROTO : **w5100.h** , **w5200.h** , **w5300.h**
- getSn_PROTOR : **w5300.h**
- getSn_RX_FIFOR : **w5300.h**
- getSn_RX_RD : **w5100.h** , **w5200.h** , **w5500.h**
- getSn_RX_WR : **w5200.h** , **w5500.h** , **w5100.h**
- getSn_RXBUF_SIZE : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- getSn_RxMASK : **w5200.h** , **w5100.h**
- getSn_RxMAX : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- getSn_RXMEM_SIZE : **w5100.h** , **w5200.h**
- getSn_SR : **w5300.h** , **w5500.h** , **w5200.h** , **w5100.h**
- getSn_SSR : **w5300.h**
- getSn_TOS : **w5200.h** , **w5500.h** , **w5300.h** , **w5100.h**
- getSn_TOSR : **w5300.h**
- getSn_TTL : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- getSn_TTLR : **w5300.h**
- getSn_TX_RD : **w5200.h** , **w5100.h** , **w5500.h**

- getSn_TX_WR : **w5200.h** , **w5100.h** , **w5500.h**
- getSn_TX_WRSR : **w5300.h**
- getSn_TXBUF_SIZE : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- getSn_TxMASK : **w5200.h** , **w5100.h**
- getSn_TxMAX : **w5500.h** , **w5300.h** , **w5100.h** , **w5200.h**
- getSn_TXMEM_SIZE : **w5100.h** , **w5200.h**
- getSUBR : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- getTMS01R : **w5300.h**
- getTMS23R : **w5300.h**
- getTMS45R : **w5300.h**
- getTMS67R : **w5300.h**
- getUIPR : **w5500.h** , **w5300.h**
- getUPORTR : **w5500.h** , **w5300.h**
- getVERSIONR : **w5500.h** , **w5200.h**

Socket APIs

- i -

- IDR : **w5300.h**
- IINCHIP_READ : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- IINCHIP_READ_BUF : **w5500.h** , **w5100.h** , **w5200.h**
- IINCHIP_WRITE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IINCHIP_WRITE_BUF : **w5100.h** , **w5200.h** , **w5500.h**
- IM_IR4 : **w5500.h**
- IM_IR5 : **w5500.h**
- IM_IR6 : **w5500.h**
- IM_IR7 : **w5500.h**
- IMR2 : **w5200.h**
- INTLEVEL : **w5200.h** , **w5500.h**
- IPPROTO_GGP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_ICMP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_IDP : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- IPPROTO_IGMP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_IP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_ND : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_PUP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_RAW : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- IPPROTO_TCP : **w5300.h** , **w5200.h** , **w5500.h** , **w5100.h**
- IPPROTO_UDP : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- IR : **w5500.h** , **w5200.h** , **w5100.h** , **w5300.h**
- IR2 : **w5200.h**
- IR_CONFLICT : **w5100.h** , **w5200.h** , **w5500.h**
- IR_DPUR : **w5300.h**

- IR_FMTU : **w5300.h**
- IR_IPCF : **w5300.h**
- IR_MP : **w5500.h**
- IR_PPPoE : **w5100.h** , **w5200.h** , **w5500.h**
- IR_PPPT : **w5300.h**
- IR_SnINT : **w5300.h**
- IR_SOCK : **w5100.h**
- IR_UNREACH : **w5100.h** , **w5500.h**

Socket APIs

- m -

- MR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- MR_AI : **w5100.h** , **w5200.h**
- MR_DBS : **w5300.h**
- MR_DBW : **w5300.h**
- MR_FARP : **w5500.h**
- MR_FS : **w5300.h**
- MR_IND : **w5100.h** , **w5200.h** , **w5300.h**
- MR_MPF : **w5300.h**
- MR_MT : **w5300.h**
- MR_PB : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- MR_PPPOE : **w5500.h**
- MR_PPPoE : **w5300.h**
- MR_PPPOE : **w5100.h** , **w5200.h**
- MR_RDH : **w5300.h**
- MR_RST : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- MR_WDF : **w5300.h**
- MR_WOL : **w5200.h** , **w5500.h**
- MTYPER : **w5300.h**

Socket APIs

- p -

- PACK_COMPLETED : **socket.h**
- PACK_FIFOBYTE : **socket.h**
- PACK_FIRST : **socket.h**
- PACK_REMAINED : **socket.h**
- PATR : **w5100.h** , **w5200.h** , **w5300.h**
- PDHAR : **w5300.h**
- PHAR : **w5500.h**
- PHY_CONFBY_HW : **wizchip_conf.h**
- PHY_CONFBY_SW : **wizchip_conf.h**
- PHY_DUPLEX_FULL : **wizchip_conf.h**
- PHY_DUPLEX_HALF : **wizchip_conf.h**
- PHY_LINK_OFF : **wizchip_conf.h**
- PHY_LINK_ON : **wizchip_conf.h**
- PHY_MODE_AUTONEGO : **wizchip_conf.h**
- PHY_MODE_MANUAL : **wizchip_conf.h**
- PHY_POWER_DOWN : **wizchip_conf.h**
- PHY_POWER_NORM : **wizchip_conf.h**
- PHY_SPEED_10 : **wizchip_conf.h**
- PHY_SPEED_100 : **wizchip_conf.h**
- PHYCFGR : **w5500.h**
- PHYCFGR_DPX_FULL : **w5500.h**
- PHYCFGR_DPX_HALF : **w5500.h**
- PHYCFGR_LNK_OFF : **w5500.h**
- PHYCFGR_LNK_ON : **w5500.h**
- PHYCFGR_OPMD : **w5500.h**

- PHYCFGR_OPMDC_100F : **w5500.h**
- PHYCFGR_OPMDC_100FA : **w5500.h**
- PHYCFGR_OPMDC_100H : **w5500.h**
- PHYCFGR_OPMDC_10F : **w5500.h**
- PHYCFGR_OPMDC_10H : **w5500.h**
- PHYCFGR_OPMDC_ALLA : **w5500.h**
- PHYCFGR_OPMDC_NA : **w5500.h**
- PHYCFGR_OPMDC_PDOWN : **w5500.h**
- PHYCFGR_RST : **w5500.h**
- PHYCFGR_SPD_10 : **w5500.h**
- PHYCFGR_SPD_100 : **w5500.h**
- PHYSTATUS : **w5200.h**
- PHYSTATUS_LINK : **w5200.h**
- PHYSTATUS_POWERDOWN : **w5200.h**
- PHYSTATUS_POWERSAVE : **w5200.h**
- PMAGIC : **w5200.h** , **w5100.h** , **w5500.h**
- PMAGICR : **w5300.h**
- PMRU : **w5500.h**
- Pn_BDPTHR : **w5300.h**
- Pn_BRDYR : **w5300.h**
- Pn_MT : **w5300.h**
- Pn_PEN : **w5300.h**
- Pn_PPL : **w5300.h**
- Pn_SN : **w5300.h**
- PPPALGO : **w5200.h**
- PSID : **w5500.h**
- PSIDR : **w5300.h**
- PTIMER : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**

Socket APIs

- r -

- RMS01R : **w5300.h**
- RMS23R : **w5300.h**
- RMS45R : **w5300.h**
- RMS67R : **w5300.h**
- RMSR : **w5100.h**
- RMSR0 : **w5300.h**
- RMSR1 : **w5300.h**
- RMSR2 : **w5300.h**
- RMSR3 : **w5300.h**
- RMSR4 : **w5300.h**
- RMSR5 : **w5300.h**
- RMSR6 : **w5300.h**
- RMSR7 : **w5300.h**

Socket APIs

- s -

- setGAR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- setIMR : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- setIMR2 : **w5200.h**
- setINTLEVEL : **w5200.h** , **w5500.h**
- setIR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setIR2 : **w5200.h**
- setMR : **w5100.h** , **w5200.h** , **w5500.h**
- setMTYPER : **w5300.h**
- setPHAR : **w5500.h**
- setPHYCFGR : **w5500.h**
- setPMAGIC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setPMRU : **w5500.h**
- setPn_BDPTHR : **w5300.h**
- setPn_BRDYR : **w5300.h**
- setPSID : **w5500.h**
- setPTIMER : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- setRCR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setRMS01R : **w5300.h**
- setRMS23R : **w5300.h**
- setRMS45R : **w5300.h**
- setRMS67R : **w5300.h**
- setRMSR : **w5100.h**
- setRTR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSIMR : **w5200.h** , **w5500.h**

- setSIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSIR : **w5200.h** , **w5500.h**
- setSn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_DPORTR : **w5300.h**
- setSn_FRAG : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_FRAGR : **w5300.h**
- setSn_IMR : **w5200.h** , **w5300.h** , **w5500.h**
- setSn_IR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_KPALVTR : **w5300.h** , **w5500.h**
- setSn_MR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_MSSR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_PORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_PORTR : **w5300.h**
- setSn_PROTO : **w5100.h** , **w5200.h** , **w5300.h**
- setSn_PROTOR : **w5300.h**
- setSn_RX_RD : **w5100.h** , **w5200.h** , **w5500.h**
- setSn_RX_WR : **w5100.h** , **w5200.h**
- setSn_RXBUF_SIZE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_RXMEM_SIZE : **w5100.h** , **w5200.h**
- setSn_TOS : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_TOSR : **w5300.h**
- setSn_TTL : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- setSn_TTLR : **w5300.h**
- setSn_TX_FIFOR : **w5300.h**
- setSn_TX_WR : **w5100.h** , **w5200.h** , **w5500.h**
- setSn_TX_WRSR : **w5300.h**
- setSn_TXBUF_SIZE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setSn_TXMEM_SIZE : **w5100.h** , **w5200.h**
- setSUBR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- setTMS01R : **w5300.h**
- setTMS23R : **w5300.h**
- setTMS45R : **w5300.h**
- setTMS67R : **w5300.h**
- setTMSR : **w5100.h**
- SF_BROAD_BLOCK : **socket.h**
- SF_ETHER_OWN : **socket.h**
- SF_IGMP_VER2 : **socket.h**

- SF_IO_NONBLOCK : **socket.h**
- SF_IPv6_BLOCK : **socket.h**
- SF_MULTI_BLOCK : **socket.h**
- SF_MULTI_ENABLE : **socket.h**
- SF_TCP_NODELAY : **socket.h**
- SF_UNI_BLOCK : **socket.h**
- SHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SIMR : **w5500.h**
- SIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SIR : **w5500.h**
- Sn_CR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_CLOSE : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_CONNECT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_DISCON : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_LISTEN : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_OPEN : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_PCJ : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCN : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCON : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PCR : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_PDISCON : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_CR_RECV : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND_KEEP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_CR_SEND_MAC : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DHAR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DIPR : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DPORT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_DPORTR : **w5300.h**
- Sn_FRAG : **w5200.h** , **w5300.h** , **w5500.h**
- Sn_FRAGR : **w5300.h**
- Sn_IMR : **w5200.h** , **w5300.h** , **w5500.h**
- Sn_IR : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_CON : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_DISCON : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_PFAIL : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_PNEXT : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_IR_PRECV : **w5100.h** , **w5300.h** , **w5200.h**
- Sn_IR_RECV : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_IR_SENDOK : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**

- Sn_IR_TIMEOUT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_KPALVTR : **w5300.h** , **w5500.h**
- Sn_MR : **w5200.h** , **w5300.h** , **w5500.h** , **w5100.h**
- Sn_MR_ALIGN : **w5300.h**
- Sn_MR_BCASTB : **w5500.h**
- Sn_MR_CLOSE : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_MR_IGMPv : **w5300.h**
- Sn_MR_IPRAW : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_MR_MACRAW : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_MR_MC : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_MR_MF : **w5100.h** , **w5200.h** , **w5300.h**
- Sn_MR_MFEN : **w5100.h** , **w5200.h** , **w5500.h**
- Sn_MR_MIP6B : **w5500.h**
- Sn_MR_MMB : **w5500.h**
- Sn_MR_MULTI : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_MR_ND : **w5300.h** , **w5100.h** , **w5200.h** , **w5500.h**
- Sn_MR_PPPOE : **w5200.h**
- Sn_MR_PPPoE : **w5300.h** , **w5100.h**
- Sn_MR_TCP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_MR_UCASTB : **w5500.h**
- Sn_MR_UDP : **w5200.h** , **w5100.h** , **w5500.h** , **w5300.h**
- Sn_MSSR : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- Sn_PORT : **w5200.h** , **w5500.h** , **w5300.h** , **w5100.h**
- Sn_PORTR : **w5300.h**
- Sn_PROTO : **w5100.h** , **w5200.h**
- Sn_PROTOR : **w5300.h**
- Sn_RX_FIFOR : **w5300.h**
- Sn_RX_RD : **w5100.h** , **w5500.h** , **w5200.h**
- Sn_RX_RSR : **w5200.h** , **w5500.h** , **w5100.h** , **w5300.h**
- Sn_RX_WR : **w5200.h** , **w5500.h** , **w5100.h**
- Sn_RXBUF_SIZE : **w5500.h**
- Sn_RXMEM_SIZE : **w5200.h**
- Sn_SR : **w5100.h** , **w5300.h** , **w5200.h** , **w5500.h**
- Sn_SSR : **w5300.h**
- Sn_TOS : **w5200.h** , **w5500.h** , **w5300.h** , **w5100.h**
- Sn_TOSR : **w5300.h**
- Sn_TTL : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- Sn_TTLR : **w5300.h**
- Sn_TX_FIFOR : **w5300.h**
- Sn_TX_FSR : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**

- Sn_TX_RD : **w5500.h** , **w5100.h** , **w5200.h**
- Sn_TX_WR : **w5500.h** , **w5200.h** , **w5100.h**
- Sn_TX_WRSR : **w5300.h**
- Sn_TXBUF_SIZE : **w5500.h**
- Sn_TXMEM_SIZE : **w5200.h**
- SOCK_ANY_PORT_NUM : **socket.c**
- SOCK_ARP : **w5300.h**
- SOCK_BUSY : **socket.h**
- SOCK_CLOSE_WAIT : **w5100.h** , **w5300.h** , **w5500.h** , **w5200.h**
- SOCK_CLOSED : **w5500.h** , **w5300.h** , **w5200.h** , **w5100.h**
- SOCK_CLOSING : **w5200.h** , **w5300.h** , **w5100.h** , **w5500.h**
- SOCK_DGRAM : **w5500.h** , **w5300.h**
- SOCK_ERROR : **socket.h**
- SOCK_ESTABLISHED : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SOCK_FATAL : **socket.h**
- SOCK_FIN_WAIT : **w5200.h** , **w5300.h** , **w5100.h** , **w5500.h**
- SOCK_INIT : **w5100.h** , **w5500.h** , **w5200.h** , **w5300.h**
- SOCK_IO_BLOCK : **socket.h**
- SOCK_IO_NONBLOCK : **socket.h**
- SOCK_IPRAW : **w5300.h** , **w5100.h** , **w5200.h**
- SOCK_LAST_ACK : **w5200.h** , **w5100.h** , **w5300.h** , **w5500.h**
- SOCK_LISTEN : **w5300.h** , **w5500.h** , **w5100.h** , **w5200.h**
- SOCK_MACRAW : **w5500.h** , **w5300.h** , **w5100.h** , **w5200.h**
- SOCK_OK : **socket.h**
- SOCK_PPPoE : **w5300.h**
- SOCK_PPPOE : **w5200.h** , **w5100.h**
- SOCK_STREAM : **w5300.h** , **w5500.h**
- SOCK_SYNRECV : **w5300.h** , **w5200.h** , **w5500.h** , **w5100.h**
- SOCK_SYNSENT : **w5300.h** , **w5500.h** , **w5200.h** , **w5100.h**
- SOCK_TIME_WAIT : **w5500.h** , **w5300.h** , **w5200.h** , **w5100.h**
- SOCK_UDP : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- SOCKERR_ARG : **socket.h**
- SOCKERR_BUFFER : **socket.h**
- SOCKERR_DATALEN : **socket.h**
- SOCKERR_IPINVALID : **socket.h**
- SOCKERR_PORTZERO : **socket.h**
- SOCKERR_SOCKCLOSED : **socket.h**
- SOCKERR_SOCKFLAG : **socket.h**
- SOCKERR_SOCKINIT : **socket.h**
- SOCKERR_SOCKMODE : **socket.h**

- SOCKERR_SOCKNUM : **socket.h**
- SOCKERR_SOCKOPT : **socket.h**
- SOCKERR_SOCKSTATUS : **socket.h**
- SOCKERR_TIMEOUT : **socket.h**
- SOCKET : **socket.h**
- SOCKFATAL_PACKLEN : **socket.h**
- SUBR : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**

Socket APIs

- t -

- TMS01R : **w5300.h**
- TMS23R : **w5300.h**
- TMS45R : **w5300.h**
- TMS67R : **w5300.h**
- TMSR : **w5100.h**
- TMSR0 : **w5300.h**
- TMSR1 : **w5300.h**
- TMSR2 : **w5300.h**
- TMSR3 : **w5300.h**
- TMSR4 : **w5300.h**
- TMSR5 : **w5300.h**
- TMSR6 : **w5300.h**
- TMSR7 : **w5300.h**

Socket APIs

- u -

- UIPR : **w5300.h** , **w5500.h**
- UIPR0 : **w5100.h**
- UPORT0 : **w5100.h**
- UPORTR : **w5300.h** , **w5500.h**

Socket APIs

- v -

- VERSIONR : **w5200.h** , **w5500.h** , **w5300.h**

Socket APIs

- w -

- WIZCHIP_CREG_BLOCK : **w5100.h** , **w5200.h** , **w5500.h** , **w5300.h**
- WIZCHIP_CRITICAL_ENTER : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- WIZCHIP_CRITICAL_EXIT : **w5100.h** , **w5200.h** , **w5300.h** , **w5500.h**
- WIZCHIP_OFFSET_INC : **w5500.h** , **w5300.h** , **w5100.h** , **w5200.h**
- WIZCHIP_RXBUF_BLOCK : **w5500.h**
- WIZCHIP_SREG_BLOCK : **w5500.h** , **w5100.h** , **w5200.h** , **w5300.h**
- WIZCHIP_TXBUF_BLOCK : **w5500.h**

Socket APIs

Related Pages

Here is a list of all related documentation pages:

Todo List

Socket APIs

socket.c

Go to the documentation of this file.

```
  1 //*****************************************************************************
  2 //
 54 //
 55 //*****************************************************************************
 56 #include "socket.h"
 57
 58 //M20150401 : Typing Error
 59 //#define SOCK_ANY_PORT_NUM  0xC000;
 60 #define SOCK_ANY_PORT_NUM  0xC000
 61
 62 static uint16_t sock_any_port = SOCK_ANY_PORT_NUM;
 63 static uint16_t sock_io_mode = 0;
 64 static uint16_t sock_is_sending = 0;
 65
 66 static uint16_t sock_remained_size[_WIZCHIP_SOCK_NUM_] = {0,0,};
 67
 68 //M20150601 : For extern decleation
 69 //static uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_] = {0,};
 70 uint8_t  sock_pack_info[_WIZCHIP_SOCK_NUM_] = {0,};
```

```
71 //
72
73 #if _WIZCHIP_ == 5200
74    static uint16_t
   sock_next_rd[_WIZCHIP_SOCK_NUM_] ={0,};
75 #endif
76
77 //A20150601 : For integrating with W5300
78 #if _WIZCHIP_ == 5300
79    uint8_t
   sock_remained_byte[_WIZCHIP_SOCK_NUM_] = {0,};
   // set by wiz_recv_data()
80 #endif
81
82
83 #define CHECK_SOCKNUM()   \
84    do{                    \
85       if(sn > _WIZCHIP_SOCK_NUM_) return
   SOCKERR_SOCKNUM;   \
86    }while(0);             \
87
88 #define CHECK_SOCKMODE(mode)  \
89    do{                     \
90       if((getSn_MR(sn) & 0x0F) != mode)
   return SOCKERR_SOCKMODE;  \
91    }while(0);              \
92
93 #define CHECK_SOCKINIT()   \
94    do{                     \
95       if((getSn_SR(sn) != SOCK_INIT)) return
   SOCKERR_SOCKINIT; \
96    }while(0);              \
97
98 #define CHECK_SOCKDATA()   \
99    do{                     \
100      if(len == 0) return SOCKERR_DATALEN;
   \
```

```
101     }while(0);                  \
102
103
104
105 int8_t socket(uint8_t sn, uint8_t protocol,
    uint16_t port, uint8_t flag)
106 {
107     CHECK_SOCKNUM();
108     switch(protocol)
109     {
110       case Sn_MR_TCP :
111          {
112             //M20150601 : Fixed the warning
    - taddr will never be NULL
113             /*
114             uint8_t taddr[4];
115             getSIPR(taddr);
116             */
117             uint32_t taddr;
118             getSIPR((uint8_t*)&taddr);
119             if(taddr == 0) return
    SOCKERR_SOCKINIT;
120          }
121       case Sn_MR_UDP :
122       case Sn_MR_MACRAW :
123          break;
124    #if ( _WIZCHIP_ < 5200 )
125       case Sn_MR_IPRAW :
126       case Sn_MR_PPPoE :
127          break;
128    #endif
129       default :
130          return SOCKERR_SOCKMODE;
131     }
132     //M20150601 : For SF_TCP_ALIGN & W5300
133     //if((flag & 0x06) != 0) return
    SOCKERR_SOCKFLAG;
```

```
134      if((flag & 0x04) != 0) return SOCKERR_SOCKFLAG;
135  #if _WIZCHIP_ == 5200
136     if(flag & 0x10) return SOCKERR_SOCKFLAG;
137  #endif
138
139      if(flag != 0)
140      {
141      switch(protocol)
142      {
143         case Sn_MR_TCP:
144            //M20150601 :  For SF_TCP_ALIGN & W5300
145            #if _WIZCHIP_ == 5300
146               if((flag & (SF_TCP_NODELAY|SF_IO_NONBLOCK|SF_TCP_ALIGN))==0) return SOCKERR_SOCKFLAG;
147            #else
148               if((flag & (SF_TCP_NODELAY|SF_IO_NONBLOCK))==0) return SOCKERR_SOCKFLAG;
149            #endif
150
151            break;
152         case Sn_MR_UDP:
153            if(flag & SF_IGMP_VER2)
154            {
155               if((flag & SF_MULTI_ENABLE)==0) return SOCKERR_SOCKFLAG;
156            }
157            #if _WIZCHIP_ == 5500
158               if(flag & SF_UNI_BLOCK)
159               {
160                  if((flag & SF_MULTI_ENABLE) == 0) return SOCKERR_SOCKFLAG;
161               }
162            #endif
```

```
163             break;
164          default:
165             break;
166       }
167    }
168       close(sn);
169       //M20150601
170       #if _WIZCHIP_ == 5300
171          setSn_MR(sn, ((uint16_t)(protocol |
   (flag & 0xF0))) | (((uint16_t)(flag & 0x02))
   << 7) );
172       #else
173          setSn_MR(sn, (protocol | (flag &
   0xF0)));
174       #endif
175       if(!port)
176       {
177          port = sock_any_port++;
178          if(sock_any_port == 0xFFF0)
   sock_any_port = SOCK_ANY_PORT_NUM;
179       }
180    setSn_PORT(sn,port);
181    setSn_CR(sn,Sn_CR_OPEN);
182    while(getSn_CR(sn));
183    //A20150401 : For release the previous
   sock_io_mode
184    sock_io_mode &= ~(1 <<sn);
185    //
186       sock_io_mode |= ((flag & SF_IO_NONBLOCK)
   << sn);
187    sock_is_sending &= ~(1<<sn);
188    sock_remained_size[sn] = 0;
189    //M20150601 : repalce 0 with
   PACK_COMPLETED
190    //sock_pack_info[sn] = 0;
191    sock_pack_info[sn] = PACK_COMPLETED;
192    //
```

```
193     while(getSn_SR(sn) == SOCK_CLOSED);
194     return (int8_t)sn;
195 }
196
197 int8_t close(uint8_t sn)
198 {
199     CHECK_SOCKNUM();
200 //A20160426 : Applied the erratum 1 of W5300
201 #if   (_WIZCHIP_ == 5300)
202     //M20160503 : Wrong socket parameter. s -
    > sn
203     //if( ((getSn_MR(s)& 0x0F) == Sn_MR_TCP)
    && (getSn_TX_FSR(s) != getSn_TxMAX(s)) )
204     if( ((getSn_MR(sn)& 0x0F) == Sn_MR_TCP)
    && (getSn_TX_FSR(sn) != getSn_TxMAX(sn)) )
205     {
206         uint8 destip[4] = {0, 0, 0, 1};
207         // TODO
208         // You can wait for completing to
    sending data;
209         // wait about 1 second;
210         // if you have completed to send data,
    skip the code of erratum 1
211         // ex> wait_1s();
212         //     if (getSn_TX_FSR(s) ==
    getSn_TxMAX(s)) continue;
213         //
214         //M20160503 : The socket() of close()
    calls close() itself again. It occures a
    infinite loop - close()->socket()->close()-
    >socket()-> ~
215         //socket(s,Sn_MR_UDP,0x3000,0);
216         //sendto(s,destip,1,destip,0x3000); //
    send the dummy data to an unknown
    destination(0.0.0.1).
217         setSn_MR(sn,Sn_MR_UDP);
218         setSn_PORTR(sn, 0x3000);
```

```
219         setSn_CR(sn,Sn_CR_OPEN);
220         while(getSn_CR(sn) != 0);
221         while(getSn_SR(sn) != SOCK_UDP);
222         sendto(sn,destip,1,destip,0x3000); //
    send the dummy data to an unknown
    destination(0.0.0.1).
223      };
224  #endif
225       setSn_CR(sn,Sn_CR_CLOSE);
226      /* wait to process the command... */
227       while( getSn_CR(sn) );
228       /* clear all interrupt of the socket. */
229       setSn_IR(sn, 0xFF);
230       //A20150401 : Release the sock_io_mode
    of socket n.
231       sock_io_mode &= ~(1<<sn);
232       //
233       sock_is_sending &= ~(1<<sn);
234       sock_remained_size[sn] = 0;
235       sock_pack_info[sn] = 0;
236       while(getSn_SR(sn) != SOCK_CLOSED);
237       return SOCK_OK;
238  }
239
240  int8_t listen(uint8_t sn)
241  {
242       CHECK_SOCKNUM();
243      CHECK_SOCKMODE(Sn_MR_TCP);
244       CHECK_SOCKINIT();
245       setSn_CR(sn,Sn_CR_LISTEN);
246       while(getSn_CR(sn));
247      while(getSn_SR(sn) != SOCK_LISTEN)
248      {
249         if(getSn_CR(sn) == SOCK_CLOSED)
250         {
251            close(sn);
252            return SOCKERR_SOCKCLOSED;
```

```
253         }
254     }
255     return SOCK_OK;
256 }
257
258
259 int8_t connect(uint8_t sn, uint8_t * addr,
    uint16_t port)
260 {
261     CHECK_SOCKNUM();
262     CHECK_SOCKMODE(Sn_MR_TCP);
263     CHECK_SOCKINIT();
264     //M20140501 : For avoiding fatal error on
    memory align mismatched
265     //if( *((uint32_t*)addr) == 0xFFFFFFFF ||
    *((uint32_t*)addr) == 0) return
    SOCKERR_IPINVALID;
266     {
267         uint32_t taddr;
268         taddr = ((uint32_t)addr[0] &
    0x000000FF);
269         taddr = (taddr << 8) +
    ((uint32_t)addr[1] & 0x000000FF);
270         taddr = (taddr << 8) +
    ((uint32_t)addr[2] & 0x000000FF);
271         taddr = (taddr << 8) +
    ((uint32_t)addr[3] & 0x000000FF);
272         if( taddr == 0xFFFFFFFF || taddr == 0)
    return SOCKERR_IPINVALID;
273     }
274     //
275
276      if(port == 0) return SOCKERR_PORTZERO;
277      setSn_DIPR(sn,addr);
278      setSn_DPORT(sn,port);
279      setSn_CR(sn,Sn_CR_CONNECT);
280     while(getSn_CR(sn));
```

```
281     if(sock_io_mode & (1<<sn)) return
    SOCK_BUSY;
282     while(getSn_SR(sn) != SOCK_ESTABLISHED)
283     {
284         if (getSn_IR(sn) & Sn_IR_TIMEOUT)
285         {
286             setSn_IR(sn, Sn_IR_TIMEOUT);
287             return SOCKERR_TIMEOUT;
288         }
289
290         if (getSn_SR(sn) == SOCK_CLOSED)
291         {
292             return SOCKERR_SOCKCLOSED;
293         }
294      }
295
296     return SOCK_OK;
297 }
298
299 int8_t disconnect(uint8_t sn)
300 {
301     CHECK_SOCKNUM();
302     CHECK_SOCKMODE(Sn_MR_TCP);
303      setSn_CR(sn,Sn_CR_DISCON);
304      /* wait to process the command... */
305      while(getSn_CR(sn));
306      sock_is_sending &= ~(1<<sn);
307     if(sock_io_mode & (1<<sn)) return
    SOCK_BUSY;
308      while(getSn_SR(sn) != SOCK_CLOSED)
309      {
310         if(getSn_IR(sn) & Sn_IR_TIMEOUT)
311         {
312            close(sn);
313            return SOCKERR_TIMEOUT;
314         }
315      }
```

```
316       return SOCK_OK;
317    }
318
319    int32_t send(uint8_t sn, uint8_t * buf,
       uint16_t len)
320    {
321       uint8_t tmp=0;
322       uint16_t freesize=0;
323
324       CHECK_SOCKNUM();
325       CHECK_SOCKMODE(Sn_MR_TCP);
326       CHECK_SOCKDATA();
327       tmp = getSn_SR(sn);
328       if(tmp != SOCK_ESTABLISHED && tmp !=
       SOCK_CLOSE_WAIT) return SOCKERR_SOCKSTATUS;
329       if( sock_is_sending & (1<<sn) )
330       {
331          tmp = getSn_IR(sn);
332          if(tmp & Sn_IR_SENDOK)
333          {
334             setSn_IR(sn, Sn_IR_SENDOK);
335             //M20150401 : Typing Error
336             //#if _WZICHIP_ == 5200
337             #if _WIZCHIP_ == 5200
338                if(getSn_TX_RD(sn) !=
       sock_next_rd[sn])
339                {
340                   setSn_CR(sn,Sn_CR_SEND);
341                   while(getSn_CR(sn));
342                   return SOCK_BUSY;
343                }
344             #endif
345             sock_is_sending &= ~(1<<sn);
346          }
347          else if(tmp & Sn_IR_TIMEOUT)
348          {
349             close(sn);
```

```
350             return SOCKERR_TIMEOUT;
351          }
352          else return SOCK_BUSY;
353       }
354       freesize = getSn_TxMAX(sn);
355       if (len > freesize) len = freesize; //
    check size not to exceed MAX size.
356       while(1)
357       {
358          freesize = getSn_TX_FSR(sn);
359          tmp = getSn_SR(sn);
360          if ((tmp != SOCK_ESTABLISHED) && (tmp
    != SOCK_CLOSE_WAIT))
361          {
362             close(sn);
363             return SOCKERR_SOCKSTATUS;
364          }
365          if( (sock_io_mode & (1<<sn)) && (len >
    freesize) ) return SOCK_BUSY;
366          if(len <= freesize) break;
367       }
368       wiz_send_data(sn, buf, len);
369       #if _WIZCHIP_ == 5200
370          sock_next_rd[sn] = getSn_TX_RD(sn) +
    len;
371       #endif
372
373       #if _WIZCHIP_ == 5300
374          setSn_TX_WRSR(sn,len);
375       #endif
376
377       setSn_CR(sn,Sn_CR_SEND);
378       /* wait to process the command... */
379       while(getSn_CR(sn));
380       sock_is_sending |= (1 << sn);
381       //M20150409 : Explicit Type Casting
382       //return len;
```

```
383     return (int32_t)len;
384  }
385
386
387  int32_t recv(uint8_t sn, uint8_t * buf,
     uint16_t len)
388  {
389     uint8_t  tmp = 0;
390     uint16_t recvsize = 0;
391  //A20150601 : For integarating with W5300
392  #if   _WIZCHIP_ == 5300
393     uint8_t head[2];
394     uint16_t mr;
395  #endif
396  //
397     CHECK_SOCKNUM();
398     CHECK_SOCKMODE(Sn_MR_TCP);
399     CHECK_SOCKDATA();
400
401     recvsize = getSn_RxMAX(sn);
402     if(recvsize < len) len = recvsize;
403
404  //A20150601 : For Integrating with W5300
405  #if _WIZCHIP_ == 5300
406     //sock_pack_info[sn] = PACK_COMPLETED;
     // for clear
407     if(sock_remained_size[sn] == 0)
408     {
409  #endif
410  //
411        while(1)
412        {
413           recvsize = getSn_RX_RSR(sn);
414           tmp = getSn_SR(sn);
415           if (tmp != SOCK_ESTABLISHED)
416           {
417              if(tmp == SOCK_CLOSE_WAIT)
```

```
418                {
419                   if(recvsize != 0) break;
420                   else if(getSn_TX_FSR(sn) ==
    getSn_TxMAX(sn))
421                   {
422                      close(sn);
423                      return SOCKERR_SOCKSTATUS;
424                   }
425                }
426                else
427                {
428                   close(sn);
429                   return SOCKERR_SOCKSTATUS;
430                }
431             }
432             if((sock_io_mode & (1<<sn)) &&
    (recvsize == 0)) return SOCK_BUSY;
433             if(recvsize != 0) break;
434          };
435 #if _WIZCHIP_ == 5300
436       }
437 #endif
438
439 //A20150601 : For integrating with W5300
440 #if _WIZCHIP_ == 5300
441       if((sock_remained_size[sn] == 0) ||
    (getSn_MR(sn) & Sn_MR_ALIGN))
442       {
443          mr = getMR();
444          if((getSn_MR(sn) & Sn_MR_ALIGN)==0)
445          {
446             wiz_recv_data(sn,head,2);
447             if(mr & MR_FS)
448                recvsize = (((uint16_t)head[1])
    << 8) | ((uint16_t)head[0]);
449             else
450                recvsize = (((uint16_t)head[0])
```

```
    << 8) | ((uint16_t)head[1]);
451             sock_pack_info[sn] = PACK_FIRST;
452          }
453          sock_remained_size[sn] = recvsize;
454       }
455       if(len > sock_remained_size[sn]) len =
    sock_remained_size[sn];
456       recvsize = len;
457       if(sock_pack_info[sn] & PACK_FIFOBYTE)
458       {
459          *buf = sock_remained_byte[sn];
460          buf++;
461          sock_pack_info[sn] &= ~
    (PACK_FIFOBYTE);
462          recvsize -= 1;
463          sock_remained_size[sn] -= 1;
464       }
465       if(recvsize != 0)
466       {
467          wiz_recv_data(sn, buf, recvsize);
468          setSn_CR(sn,Sn_CR_RECV);
469          while(getSn_CR(sn));
470       }
471       sock_remained_size[sn] -= recvsize;
472       if(sock_remained_size[sn] != 0)
473       {
474          sock_pack_info[sn] |= PACK_REMAINED;
475          if(recvsize & 0x1) sock_pack_info[sn]
    |= PACK_FIFOBYTE;
476       }
477       else sock_pack_info[sn] = PACK_COMPLETED;
478       if(getSn_MR(sn) & Sn_MR_ALIGN)
    sock_remained_size[sn] = 0;
479       //len = recvsize;
480    #else
481       if(recvsize < len) len = recvsize;
482       wiz_recv_data(sn, buf, len);
```

```
483     setSn_CR(sn,Sn_CR_RECV);
484     while(getSn_CR(sn));
485  #endif
486
487     //M20150409 : Explicit Type Casting
488     //return len;
489     return (int32_t)len;
490  }
491
492  int32_t sendto(uint8_t sn, uint8_t * buf,
    uint16_t len, uint8_t * addr, uint16_t port)
493  {
494     uint8_t tmp = 0;
495     uint16_t freesize = 0;
496     uint32_t taddr;
497
498     CHECK_SOCKNUM();
499     switch(getSn_MR(sn) & 0x0F)
500     {
501        case Sn_MR_UDP:
502        case Sn_MR_MACRAW:
503           break;
504        default:
505           return SOCKERR_SOCKMODE;
506     }
507     CHECK_SOCKDATA();
508     //M20140501 : For avoiding fatal error on
    memory align mismatched
509     //if(*((uint32_t*)addr) == 0) return
    SOCKERR_IPINVALID;
510     //{
511        //uint32_t taddr;
512        taddr = ((uint32_t)addr[0]) &
    0x000000FF;
513        taddr = (taddr << 8) +
    ((uint32_t)addr[1] & 0x000000FF);
514        taddr = (taddr << 8) +
```

```
    ((uint32_t)addr[2] & 0x000000FF);
515         taddr = (taddr << 8) +
    ((uint32_t)addr[3] & 0x000000FF);
516     //}
517     //
518     //if(*((uint32_t*)addr) == 0) return
    SOCKERR_IPINVALID;
519     if(taddr == 0)                return
    SOCKERR_IPINVALID;
520     if(port == 0)               return
    SOCKERR_PORTZERO;
521     tmp = getSn_SR(sn);
522     if(tmp != SOCK_MACRAW && tmp != SOCK_UDP)
    return SOCKERR_SOCKSTATUS;
523
524     setSn_DIPR(sn,addr);
525     setSn_DPORT(sn,port);
526     freesize = getSn_TxMAX(sn);
527     if (len > freesize) len = freesize; //
    check size not to exceed MAX size.
528     while(1)
529     {
530        freesize = getSn_TX_FSR(sn);
531        if(getSn_SR(sn) == SOCK_CLOSED) return
    SOCKERR_SOCKCLOSED;
532        if( (sock_io_mode & (1<<sn)) && (len >
    freesize) ) return SOCK_BUSY;
533        if(len <= freesize) break;
534     };
535      wiz_send_data(sn, buf, len);
536
537     #if _WIZCHIP_ < 5500   //M20150401 : for
    WIZCHIP Errata #4, #5 (ARP errata)
538        getSIPR((uint8_t*)&taddr);
539        if(taddr == 0)
540        {
541           getSUBR((uint8_t*)&taddr);
```

```
542 
    setSUBR((uint8_t*)"\x00\x00\x00\x00");
543         }
544         else taddr = 0;
545      #endif
546 
547  //A20150601 : For W5300
548  #if _WIZCHIP_ == 5300
549      setSn_TX_WRSR(sn, len);
550  #endif
551  //
552       setSn_CR(sn,Sn_CR_SEND);
553       /* wait to process the command... */
554       while(getSn_CR(sn));
555      while(1)
556      {
557         tmp = getSn_IR(sn);
558         if(tmp & Sn_IR_SENDOK)
559         {
560            setSn_IR(sn, Sn_IR_SENDOK);
561            break;
562         }
563         //M:20131104
564         //else if(tmp & Sn_IR_TIMEOUT) return
    SOCKERR_TIMEOUT;
565         else if(tmp & Sn_IR_TIMEOUT)
566         {
567            setSn_IR(sn, Sn_IR_TIMEOUT);
568            //M20150409 : Fixed the lost of
    sign bits by type casting.
569            //len = (uint16_t)SOCKERR_TIMEOUT;
570            //break;
571            #if _WIZCHIP_ < 5500   //M20150401
    : for WIZCHIP Errata #4, #5 (ARP errata)
572               if(taddr)
    setSUBR((uint8_t*)&taddr);
573            #endif
```

```
574             return SOCKERR_TIMEOUT;
575          }
577       }
578       #if _WIZCHIP_ < 5500   //M20150401 : for WIZCHIP Errata #4, #5 (ARP errata)
579          if(taddr) setSUBR((uint8_t*)&taddr);
580       #endif
581       //M20150409 : Explicit Type Casting
582       //return len;
583       return (int32_t)len;
584    }
585
586
587
588    int32_t recvfrom(uint8_t sn, uint8_t * buf, uint16_t len, uint8_t * addr, uint16_t *port)
589    {
590    //M20150601 : For W5300
591    #if _WIZCHIP_ == 5300
592       uint16_t mr;
593       uint16_t mr1;
594    #else
595       uint8_t  mr;
596    #endif
597    //
598       uint8_t  head[8];
599        uint16_t pack_len=0;
600
601       CHECK_SOCKNUM();
602       //CHECK_SOCKMODE(Sn_MR_UDP);
603    //A20150601
604    #if _WIZCHIP_ == 5300
605       mr1 = getMR();
606    #endif
607
608       switch((mr=getSn_MR(sn)) & 0x0F)
609       {
```

```
610         case Sn_MR_UDP:
611         case Sn_MR_MACRAW:
612            break;
613      #if ( _WIZCHIP_ < 5200 )
614         case Sn_MR_IPRAW:
615         case Sn_MR_PPPoE:
616            break;
617      #endif
618         default:
619            return SOCKERR_SOCKMODE;
620      }
621      CHECK_SOCKDATA();
622      if(sock_remained_size[sn] == 0)
623      {
624         while(1)
625         {
626            pack_len = getSn_RX_RSR(sn);
627            if(getSn_SR(sn) == SOCK_CLOSED)
   return SOCKERR_SOCKCLOSED;
628            if( (sock_io_mode & (1<<sn)) &&
   (pack_len == 0) ) return SOCK_BUSY;
629            if(pack_len != 0) break;
630         };
631      }
632  //D20150601 : Move it to bottom
633  // sock_pack_info[sn] = PACK_COMPLETED;
634       switch (mr & 0x07)
635       {
636          case Sn_MR_UDP :
637             if(sock_remained_size[sn] == 0)
638             {
639               wiz_recv_data(sn, head, 8);
640               setSn_CR(sn,Sn_CR_RECV);
641               while(getSn_CR(sn));
642               // read peer's IP address, port
   number & packet length
643         //A20150601 : For W5300
```

```
644           #if _WIZCHIP_ == 5300
645              if(mr1 & MR_FS)
646              {
647                 addr[0] = head[1];
648                 addr[1] = head[0];
649                 addr[2] = head[3];
650                 addr[3] = head[2];
651                 *port = head[5];
652                 *port = (*port << 8) +
    head[4];
653                   sock_remained_size[sn] =
    head[7];
654                   sock_remained_size[sn] =
    (sock_remained_size[sn] << 8) + head[6];
655              }
656               else
657               {
658            #endif
659                  addr[0] = head[0];
660                   addr[1] = head[1];
661                   addr[2] = head[2];
662                   addr[3] = head[3];
663                   *port = head[4];
664                   *port = (*port << 8) +
    head[5];
665                   sock_remained_size[sn] =
    head[6];
666                   sock_remained_size[sn] =
    (sock_remained_size[sn] << 8) + head[7];
667            #if _WIZCHIP_ == 5300
668               }
669            #endif
670               sock_pack_info[sn] = PACK_FIRST;
671         }
672               if(len < sock_remained_size[sn])
    pack_len = len;
673               else pack_len =
```

```
    sock_remained_size[sn];
674             //A20150601 : For W5300
675             len = pack_len;
676             #if _WIZCHIP_ == 5300
677                if(sock_pack_info[sn] &
    PACK_FIFOBYTE)
678                {
679                   *buf++ =
    sock_remained_byte[sn];
680                   pack_len -= 1;
681                   sock_remained_size[sn] -=
    1;
682                   sock_pack_info[sn] &=
    ~PACK_FIFOBYTE;
683                }
684             #endif
685             //
686             // Need to packet length check
    (default 1472)
687             //
688          wiz_recv_data(sn, buf, pack_len); //
    data copy.
689             break;
690        case Sn_MR_MACRAW :
691           if(sock_remained_size[sn] == 0)
692           {
693             wiz_recv_data(sn, head, 2);
694             setSn_CR(sn,Sn_CR_RECV);
695             while(getSn_CR(sn));
696             // read peer's IP address, port
    number & packet length
697                sock_remained_size[sn] =
    head[0];
698             sock_remained_size[sn] =
    (sock_remained_size[sn] <<8) + head[1];
699             if(sock_remained_size[sn] >
    1514)
```

```
700                {
701                   close(sn);
702                   return SOCKFATAL_PACKLEN;
703                }
704                sock_pack_info[sn] = PACK_FIRST;
705          }
706                if(len < sock_remained_size[sn]) pack_len = len;
707                else pack_len = sock_remained_size[sn];
708                wiz_recv_data(sn,buf,pack_len);
709               break;
710     #if ( _WIZCHIP_ < 5200 )
711           case Sn_MR_IPRAW:
712              if(sock_remained_size[sn] == 0)
713              {
714               wiz_recv_data(sn, head, 6);
715               setSn_CR(sn,Sn_CR_RECV);
716               while(getSn_CR(sn));
717               addr[0] = head[0];
718               addr[1] = head[1];
719               addr[2] = head[2];
720               addr[3] = head[3];
721               sock_remained_size[sn] = head[4];
722               //M20150401 : For Typing Error
723               //sock_remaiend_size[sn] = (sock_remained_size[sn] << 8) + head[5];
724               sock_remained_size[sn] = (sock_remained_size[sn] << 8) + head[5];
725               sock_pack_info[sn] = PACK_FIRST;
726            }
727               //
728               // Need to packet length check
729               //
730               if(len < sock_remained_size[sn]) pack_len = len;
```

```
731             else pack_len = sock_remained_size[sn];
732         wiz_recv_data(sn, buf, pack_len); // data copy.
733             break;
734    #endif
735       default:
736          wiz_recv_ignore(sn, pack_len); // data copy.
737          sock_remained_size[sn] = pack_len;
738          break;
739    }
740     setSn_CR(sn,Sn_CR_RECV);
741     /* wait to process the command... */
742     while(getSn_CR(sn)) ;
743     sock_remained_size[sn] -= pack_len;
744     //M20150601 :
745     //if(sock_remained_size[sn] != 0) sock_pack_info[sn] |= 0x01;
746     if(sock_remained_size[sn] != 0)
747     {
748        sock_pack_info[sn] |= PACK_REMAINED;
749    #if _WIZCHIP_ == 5300
750        if(pack_len & 0x01) sock_pack_info[sn] |= PACK_FIFOBYTE;
751    #endif
752     }
753     else sock_pack_info[sn] = PACK_COMPLETED;
754 #if _WIZCHIP_ == 5300
755    pack_len = len;
756 #endif
757    //
758    //M20150409 : Explicit Type Casting
759    //return pack_len;
760    return (int32_t)pack_len;
761 }
```

```
762
763
764 int8_t  ctlsocket(uint8_t sn, ctlsock_type cstype, void* arg)
765 {
766    uint8_t tmp = 0;
767    CHECK_SOCKNUM();
768    switch(cstype)
769    {
770       case CS_SET_IOMODE:
771          tmp = *((uint8_t*)arg);
772          if(tmp == SOCK_IO_NONBLOCK) sock_io_mode |= (1<<sn);
773          else if(tmp == SOCK_IO_BLOCK) sock_io_mode &= ~(1<<sn);
774          else return SOCKERR_ARG;
775          break;
776       case CS_GET_IOMODE:
777          //M20140501 : implict type casting -> explict type casting
778          //*((uint8_t*)arg) = (sock_io_mode >> sn) & 0x0001;
779          *((uint8_t*)arg) = (uint8_t)((sock_io_mode >> sn) & 0x0001);
780          //
781          break;
782       case CS_GET_MAXTXBUF:
783          *((uint16_t*)arg) = getSn_TxMAX(sn);
784          break;
785       case CS_GET_MAXRXBUF:
786          *((uint16_t*)arg) = getSn_RxMAX(sn);
787          break;
788       case CS_CLR_INTERRUPT:
789          if( (*(uint8_t*)arg) > SIK_ALL) return SOCKERR_ARG;
```

```
790             setSn_IR(sn,*(uint8_t*)arg);
791             break;
792          case CS_GET_INTERRUPT:
793             *((uint8_t*)arg) = getSn_IR(sn);
794             break;
795       #if _WIZCHIP_ != 5100
796          case CS_SET_INTMASK:
797             if( (*(uint8_t*)arg) > SIK_ALL)
    return SOCKERR_ARG;
798             setSn_IMR(sn,*(uint8_t*)arg);
799             break;
800          case CS_GET_INTMASK:
801             *((uint8_t*)arg) = getSn_IMR(sn);
802             break;
803       #endif
804          default:
805             return SOCKERR_ARG;
806       }
807       return SOCK_OK;
808 }
809
810 int8_t  setsockopt(uint8_t sn, sockopt_type
    sotype, void* arg)
811 {
812  // M20131220 : Remove warning
813  //uint8_t tmp;
814    CHECK_SOCKNUM();
815    switch(sotype)
816    {
817       case SO_TTL:
818          setSn_TTL(sn,*(uint8_t*)arg);
819          break;
820       case SO_TOS:
821          setSn_TOS(sn,*(uint8_t*)arg);
822          break;
823       case SO_MSS:
824          setSn_MSSR(sn,*(uint16_t*)arg);
```

```
825             break;
826          case SO_DESTIP:
827             setSn_DIPR(sn, (uint8_t*)arg);
828             break;
829          case SO_DESTPORT:
830             setSn_DPORT(sn, *(uint16_t*)arg);
831             break;
832  #if _WIZCHIP_ != 5100
833          case SO_KEEPALIVESEND:
834             CHECK_SOCKMODE(Sn_MR_TCP);
835             #if _WIZCHIP_ > 5200
836                if(getSn_KPALVTR(sn) != 0)
    return SOCKERR_SOCKOPT;
837             #endif
838                setSn_CR(sn,Sn_CR_SEND_KEEP);
839                while(getSn_CR(sn) != 0)
840                {
841                   // M20131220
842                    //if ((tmp = getSn_IR(sn)) &
    Sn_IR_TIMEOUT)
843                   if (getSn_IR(sn) &
    Sn_IR_TIMEOUT)
844                    {
845                        setSn_IR(sn,
    Sn_IR_TIMEOUT);
846                      return SOCKERR_TIMEOUT;
847                    }
848                }
849             break;
850     #if _WIZCHIP_ > 5200
851          case SO_KEEPALIVEAUTO:
852             CHECK_SOCKMODE(Sn_MR_TCP);
853             setSn_KPALVTR(sn,*(uint8_t*)arg);
854             break;
855     #endif
856  #endif
857          default:
```

```
858             return SOCKERR_ARG;
859         }
860         return SOCK_OK;
861     }
862
863     int8_t  getsockopt(uint8_t sn, sockopt_type
    sotype, void* arg)
864     {
865         CHECK_SOCKNUM();
866         switch(sotype)
867         {
868             case SO_FLAG:
869                 *(uint8_t*)arg = getSn_MR(sn) &
    0xF0;
870                 break;
871             case SO_TTL:
872                 *(uint8_t*) arg = getSn_TTL(sn);
873                 break;
874             case SO_TOS:
875                 *(uint8_t*) arg = getSn_TOS(sn);
876                 break;
877             case SO_MSS:
878                 *(uint8_t*) arg = getSn_MSSR(sn);
879                 break;
880             case SO_DESTIP:
881                 getSn_DIPR(sn, (uint8_t*)arg);
882                 break;
883             case SO_DESTPORT:
884                 *(uint16_t*) arg = getSn_DPORT(sn);
885                 break;
886         #if _WIZCHIP_ > 5200
887             case SO_KEEPALIVEAUTO:
888                 CHECK_SOCKMODE(Sn_MR_TCP);
889                 *(uint16_t*) arg =
    getSn_KPALVTR(sn);
890                 break;
891         #endif
```

```
892         case SO_SENDBUF:
893            *(uint16_t*) arg =
    getSn_TX_FSR(sn);
894            break;
895         case SO_RECVBUF:
896            *(uint16_t*) arg =
    getSn_RX_RSR(sn);
897            break;
898         case SO_STATUS:
899            *(uint8_t*) arg = getSn_SR(sn);
900            break;
901         case SO_REMAINSIZE:
902            if(getSn_MR(sn) == Sn_MR_TCP)
903               *(uint16_t*)arg =
    getSn_RX_RSR(sn);
904            else
905               *(uint16_t*)arg =
    sock_remained_size[sn];
906            break;
907         case SO_PACKINFO:
908            CHECK_SOCKMODE(Sn_MR_TCP);
909            *(uint8_t*)arg =
    sock_pack_info[sn];
910            break;
911         default:
912            return SOCKERR_SOCKOPT;
913      }
914      return SOCK_OK;
915 }
```

Socket APIs

w5100.h

Go to the documentation of this file.

```
    1 //*
      ****************************************************
      ****************************
   38 //
   39 //*************************************************
      ***********************************
   40
   41 #ifndef _W5100_H_
   42 #define _W5100_H_
   43 #include <stdint.h>
   44 #include "wizchip_conf.h"
   45
   47 #if   (_WIZCHIP_ == 5100)
   48
   50 #define _WIZCHIP_SN_BASE_  (0x0400)
   51 #define _WIZCHIP_SN_SIZE_  (0x0100)
   52 #define _WIZCHIP_IO_TXBUF_ (0x4000) /*
      Internal Tx buffer address of the iinchip */
   53 #define _WIZCHIP_IO_RXBUF_ (0x6000) /*
      Internal Rx buffer address of the iinchip */
   54
   55
   56 #define WIZCHIP_CREG_BLOCK
      0x00
   57 #define WIZCHIP_SREG_BLOCK(N)
      (_WIZCHIP_SN_BASE_+ _WIZCHIP_SN_SIZE_*N)
```

```
58
59 #define WIZCHIP_OFFSET_INC(ADDR, N)    (ADDR
    + N)
60
61 #if (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_)
62    #define _W5100_IO_BASE_
    _WIZCHIP_IO_BASE_
63 #elif (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_)
64     #define IDM_OR
    ((_WIZCHIP_IO_BASE  + 0x0000))
65     #define IDM_AR0
    ((_WIZCHIP_IO_BASE_ + 0x0001))
66     #define IDM_AR1
    ((_WIZCHIP_IO_BASE_ + 0x0002))
67     #define IDM_DR
    ((_WIZCHIP_IO_BASE_ + 0x0003))
68     #define _W5100_IO_BASE_    0x0000
69 #elif (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_)
70    #define _W5100_IO_BASE_    0x0000
71 #endif
72
74 // Definition For Legacy Chip Driver //
76 #define IINCHIP_READ(ADDR)
    WIZCHIP_READ(ADDR)
77 #define IINCHIP_WRITE(ADDR,VAL)
    WIZCHIP_WRITE(ADDR,VAL)
78 #define IINCHIP_READ_BUF(ADDR,BUF,LEN)
    WIZCHIP_READ_BUF(ADDR,BUF,LEN)
79 #define IINCHIP_WRITE_BUF(ADDR,BUF,LEN)
    WIZCHIP_WRITE(ADDR,BUF,LEN)
80
81
82 //-----------    defgroup ----------------
    -------------
```

```
 83
183  //-----------------------------------------
     ----------------------------------------
184
185 //------------------------------ W5100 Common
    Registers IOMAP ------------------------------
201 #if _WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_
202    #define MR
    (_WIZCHIP_IO_BASE_ + (0x0000))  // Mode
203 #else
204    #define MR
    (_W5100_IO_BASE_ + (0x0000))  // Mode
205 #endif
206
212 #define GAR                 (_W5100_IO_BASE_
    + (0x0001))  // GW Address
213
219 #define SUBR                (_W5100_IO_BASE_
    + (0x0005)) // SN Mask Address
220
226 #define SHAR                (_W5100_IO_BASE_
    + (0x0009)) // Source Hardware Address
227
233 #define SIPR                (_W5100_IO_BASE_
    + (0x000F)) // Source IP Address
234
235 // Reserved                 (_W5100_IO_BASE_
    + (0x0013))
236 // Reserved                 (_W5100_IO_BASE_
    + (0x0014))
237
256 #define IR                  (_W5100_IO_BASE_
    + (0x0015)) // Interrupt
257
264 #define _IMR_               (_W5100_IO_BASE_
    + (0x0016)) // Socket Interrupt Mask
```

```
265
274 #define _RTR_                (_W5100_IO_BASE_
    + (0x0017)) // Retry Time
275
282 #define _RCR_            (_W5100_IO_BASE_ +
    (0x0019)) // Retry Count
283 #define RMSR                 (_W5100_IO_BASE_
    + (0x001A)) // Receicve Memory Size
284 #define TMSR                 (_W5100_IO_BASE_
    + (0x001B)) // Trnasmit Memory Size
285
286
293 #define PATR             (_W5100_IO_BASE_ +
    (0x001C))
294
295
301 #define PTIMER               (_W5100_IO_BASE_
    + (0x0028)) // PPP LCP RequestTimer
302
308 #define PMAGIC               (_W5100_IO_BASE_
    + (0x0029)) // PPP LCP Magic number
309
310 #define UIPR0                (_W5100_IO_BASE_
    + (0x002A))
311 #define UPORT0           (_W5100_IO_BASE  +
    (0x002E))
312
313
314
315 //------------------------------ W5100 Socket
    Registers ------------------------------
316
317 //---------------------------- For Backward
    Compatibility ----------------------------
318
352 #define Sn_MR(sn)            (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0000)) // socket
```

```
    Mode register
  353
  380 #define Sn_CR(sn)           (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0001)) //
    channel Sn_CR register
  381
  401 #define Sn_IR(sn)           (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0002)) //
    channel interrupt register
  402
  424 #define Sn_SR(sn)           (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0003)) //
    channel status register
  425
  432 #define Sn_PORT(sn)     (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0004)) // source
    port register
  433
  440 #define Sn_DHAR(sn)     (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0006)) // Peer MAC
    register address
  441
  450 #define Sn_DIPR(sn)     (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x000C)) // Peer IP
    register address
  451
  460 #define Sn_DPORT(sn)    (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0010)) // Peer
    port register address
  461
  467 #define Sn_MSSR(sn)    (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0012)) // Maximum
    Segment Size(Sn_MSSR0) register address
  468
  475 #define Sn_PROTO(sn)       (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0014)) //
    Protocol of IP Header field register in IP raw
```

```
    mode
  476
  483 #define Sn_TOS(sn)          (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + 0x0015) // IP Type
    of Service(TOS) Register
  484
  491 #define Sn_TTL(sn)      (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0016)) // IP Time
    to live(TTL) Register
  492
  493 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0017))
  494 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0018))
  495 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0019))
  496 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001A))
  497 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001B))
  498 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001C))
  499 // Reserved                 (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001D))
  500
  510 #define Sn_TX_FSR(sn)   (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0020)) // Transmit
    free memory size register
  511
  522 #define Sn_TX_RD(sn)        (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0022)) //
    Transmit memory read pointer register address
  523
  536 #define Sn_TX_WR(sn)        (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0024)) //
    Transmit memory write pointer register address
  537
```

```
545 #define Sn_RX_RSR(sn)   (_W5100_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0026)) // Received
    data size register
546
558 #define Sn_RX_RD(sn)        (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0028)) // Read
    point of Receive memory
559
567 #define Sn_RX_WR(sn)        (_W5100_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x002A)) // Write
    point of Receive memory
568
569
570 //------------------------------ W5100
    Register values  ------------------------------
571
572 /* MODE register values */
577 #define MR_RST              0x80
578
579
586 #define MR_PB               0x10
587
588
594 #define MR_PPPOE            0x08
595
596
603 #define MR_AI               0x02
604
605
611 #define MR_IND              0x01
612
613 /* IR register values */
618 #define IR_CONFLICT         0x80
619
620
625 #define IR_UNREACH          0x40
626
```

```
627
631 #define IR_PPPoE            0x20
632
633 #define IR_SOCK(sn)     (0x01 << sn)
634
635
636 // Sn_MR values
637 /* Sn_MR Default values */
642 #define Sn_MR_CLOSE         0x00
643
644
648 #define Sn_MR_TCP           0x01
649
650
654 #define Sn_MR_UDP       0x02
655 #define Sn_MR_IPRAW     0x03
656
657
662 #define Sn_MR_MACRAW        0x04
663
664
669 #define Sn_MR_PPPoE         0x05
670
671
679 #define Sn_MR_ND            0x20
680
681
688 #define Sn_MR_MC            Sn_MR_ND
689
690
700 #define Sn_MR_MF            0x40
701 #define Sn_MR_MFEN      Sn_MR_MF
702
703
704 /* Sn_MR Default values */
713 #define Sn_MR_MULTI         0x80
714
```

```
715 /* Sn_CR values */
730 #define Sn_CR_OPEN          0x01
731
732
741 #define Sn_CR_LISTEN        0x02
742
743
753 #define Sn_CR_CONNECT       0x04
754
755
766 #define Sn_CR_DISCON        0x08
767
768
772 #define Sn_CR_CLOSE         0x10
773
780 #define Sn_CR_SEND          0x20
781
790 #define Sn_CR_SEND_MAC      0x21
791
798 #define Sn_CR_SEND_KEEP     0x22
799
806 #define Sn_CR_RECV          0x40
807
812 #define Sn_CR_PCON          0x23
813
818 #define Sn_CR_PDISCON       0x24
819
824 #define Sn_CR_PCR           0x25
825
830 #define Sn_CR_PCN           0x26
831
836 #define Sn_CR_PCJ           0x27
837
838 /* Sn_IR values */
843 #define Sn_IR_PRECV         0x80
844
849 #define Sn_IR_PFAIL         0x40
```

```
850
855 #define Sn_IR_PNEXT         0x20
856
861 #define Sn_IR_SENDOK        0x10
862
863
867 #define Sn_IR_TIMEOUT       0x08
868
869
873 #define Sn_IR_RECV          0x04
874
879 #define Sn_IR_DISCON        0x02
880
885 #define Sn_IR_CON           0x01
886
887 /* Sn_SR values */
893 #define SOCK_CLOSED         0x00
894
895
901 #define SOCK_INIT           0x13
902
903
909 #define SOCK_LISTEN         0x14
910
918 #define SOCK_SYNSENT        0x15
919
926 #define SOCK_SYNRECV        0x16
927
935 #define SOCK_ESTABLISHED    0x17
936
943 #define SOCK_FIN_WAIT       0x18
944
951 #define SOCK_CLOSING        0x1A
952
959 #define SOCK_TIME_WAIT      0x1B
960
967 #define SOCK_CLOSE_WAIT     0x1C
```

```
 968
 974 #define SOCK_LAST_ACK       0x1D
 975
 982 #define SOCK_UDP            0x22
 983
 984
 990 #define SOCK_IPRAW          0x32
 991
 992
 998 #define SOCK_MACRAW         0x42
 999
1000
1007 #define SOCK_PPPOE          0x5F
1008
1009 // IP PROTOCOL
1010 #define IPPROTO_IP          0
1011 #define IPPROTO_ICMP           1
1012 #define IPPROTO_IGMP           2
1013 #define IPPROTO_GGP         3
1014 #define IPPROTO_TCP         6
1015 #define IPPROTO_PUP         12
1016 #define IPPROTO_UDP         17
1017 #define IPPROTO_IDP         22
1018 #define IPPROTO_ND          77
1019 #define IPPROTO_RAW         255
1020
1021
1032 #define WIZCHIP_CRITICAL_ENTER()
    WIZCHIP.CRIS._enter()
1033
1034 #ifdef _exit
1035 #undef _exit
1036 #endif
1037
1049 #define WIZCHIP_CRITICAL_EXIT()
    WIZCHIP.CRIS._exit()
1050
```

```
1051
1052
1054 // Basic I/O Function //
1056 //
1057 //M20150601 :  uint16_t AddrSel --> uint32_t
     AddrSel
1058 //
1065 uint8_t  WIZCHIP_READ (uint32_t AddrSel);
1066
1074 void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t
     wb );
1075
1083 void WIZCHIP_READ_BUF (uint32_t AddrSel,
     uint8_t* pBuf, uint16_t len);
1084
1092 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
     uint8_t* pBuf, uint16_t len);
1093
1094
1096 // Common Register IO function //
1098
1105 #if (_WIZCHIP_IO_MODE_ &
     _WIZCHIP_IO_MODE_SPI_)
1106    #define setMR(mr)    WIZCHIP_WRITE(MR,mr)
1107 #else
1108    #define setMR(mr)    (*((uint8_t*)MR) =
     mr)
1109 #endif
1110
1117 #if (_WIZCHIP_IO_MODE_ &
     _WIZCHIP_IO_MODE_SPI_)
1118    #define getMR()      WIZCHIP_READ(MR)
1119 #else
1120    #define getMR()      (*(uint8_t*)MR)
1121 #endif
1122
1129 #define setGAR(gar) \
```

```
1130          WIZCHIP_WRITE_BUF(GAR,gar,4)
1131
1138  #define getGAR(gar) \
1139          WIZCHIP_READ_BUF(GAR,gar,4)
1140
1150  #define setSUBR(subr) \
1151        WIZCHIP_WRITE_BUF(SUBR,subr,4)
1152
1159  #define getSUBR(subr) \
1160          WIZCHIP_READ_BUF(SUBR, subr, 4)
1161
1168  #define setSHAR(shar) \
1169          WIZCHIP_WRITE_BUF(SHAR, shar, 6)
1170
1177  #define getSHAR(shar) \
1178          WIZCHIP_READ_BUF(SHAR, shar, 6)
1179
1186  #define setSIPR(sipr) \
1187          WIZCHIP_WRITE_BUF(SIPR, sipr, 4)
1188
1195  #define getSIPR(sipr) \
1196          WIZCHIP_READ_BUF(SIPR, sipr, 4)
1197
1204  #define setIR(ir) \
1205          WIZCHIP_WRITE(IR, (ir & 0xA0))
1206
1212  #define getIR() \
1213          (WIZCHIP_READ(IR) & 0xA0)
1214
1221  #define setIMR(imr) \
1222          WIZCHIP_WRITE(_IMR_, imr)
1223
1230  #define getIMR() \
1231          WIZCHIP_READ(_IMR_)
1232
1239  #define setRTR(rtr)   {\
1240          WIZCHIP_WRITE(_RTR_,   (uint8_t)(rtr
```

```
    >> 8)); \
1241|
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),
    (uint8_t) rtr); \
1242|     }
1243|
1250| #define getRTR() \
1251|         (((uint16_t)WIZCHIP_READ(_RTR_) <<
    8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))
1252|
1259| #define setRCR(rcr) \
1260|         WIZCHIP_WRITE(_RCR_, rcr)
1261|
1268| #define getRCR() \
1269|         WIZCHIP_READ(_RCR_)
1270|
1276| #define setRMSR(rmsr)   \
1277|       WIZCHIP_WRITE(RMSR) // Receicve Memory
    Size
1278|
1285|  #define getRMSR()   \
1286|       WIZCHIP_READ() // Receicve Memory Size
1287|
1293| #define setTMSR(rmsr)   \
1294|       WIZCHIP_WRITE(TMSR) // Receicve Memory
    Size
1295|
1309| #define getPATR() \
1310|         (((uint16_t)WIZCHIP_READ(PATR) << 8)
    + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR,1)))
1311|
1317| #define getPPPALGO() \
1318|         WIZCHIP_READ(PPPALGO)
1319|
1320|
1327| #define setPTIMER(ptimer) \
```

```
1328            WIZCHIP_WRITE(PTIMER, ptimer)
1329
1336    #define getPTIMER() \
1337            WIZCHIP_READ(PTIMER)
1338
1345    #define setPMAGIC(pmagic) \
1346            WIZCHIP_WRITE(PMAGIC, pmagic)
1347
1354    #define getPMAGIC() \
1355            WIZCHIP_READ(PMAGIC)
1356
1358    // Socket N register I/O function //
1360
1367    #define setSn_MR(sn, mr) \
1368            WIZCHIP_WRITE(Sn_MR(sn),mr)
1369
1377    #define getSn_MR(sn) \
1378        WIZCHIP_READ(Sn_MR(sn))
1379
1387    #define setSn_CR(sn, cr) \
1388            WIZCHIP_WRITE(Sn_CR(sn), cr)
1389
1397    #define getSn_CR(sn) \
1398            WIZCHIP_READ(Sn_CR(sn))
1399
1407    #define setSn_IR(sn, ir) \
1408            WIZCHIP_WRITE(Sn_IR(sn), ir)
1409
1417    #define getSn_IR(sn) \
1418            WIZCHIP_READ(Sn_IR(sn))
1419
1426    #define getSn_SR(sn) \
1427            WIZCHIP_READ(Sn_SR(sn))
1428
1436    #define setSn_PORT(sn, port)  { \
1437            WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \
```

```
1438 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1
    ), (uint8_t) port); \
1439     }
1440 
1448 #define getSn_PORT(sn) \
1449 
    (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    ))
1450 
1458 #define setSn_DHAR(sn, dhar) \
1459         WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar,
    6)
1460 
1468 #define getSn_DHAR(sn, dhar) \
1469         WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar,
    6)
1470 
1478 #define setSn_DIPR(sn, dipr) \
1479         WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
    4)
1480 
1488 #define getSn_DIPR(sn, dipr) \
1489         WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
    4)
1490 
1498 #define setSn_DPORT(sn, dport) { \
1499         WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \
1500 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),
    1), (uint8_t)  dport); \
1501     }
1502 
1510 #define getSn_DPORT(sn) \
1511 
```

```
    (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    )))
1512|
1520| #define setSn_MSSR(sn, mss) { \
1521|         WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \
1522|
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1
    ), (uint8_t) mss); \
1523|     }
1524|
1532| #define getSn_MSSR(sn) \
1533|
    (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    ))
1534|
1542| #define setSn_PROTO(sn, proto) \
1543|         WIZCHIP_WRITE(Sn_TOS(sn), tos)
1544|
1552| #define getSn_PROTO(sn) \
1553|         WIZCHIP_READ(Sn_TOS(sn))
1554|
1562| #define setSn_TOS(sn, tos) \
1563|         WIZCHIP_WRITE(Sn_TOS(sn), tos)
1564|
1572| #define getSn_TOS(sn) \
1573|         WIZCHIP_READ(Sn_TOS(sn))
1574|
1582| #define setSn_TTL(sn, ttl) \
1583|         WIZCHIP_WRITE(Sn_TTL(sn), ttl)
1584|
1592| #define getSn_TTL(sn) \
1593|         WIZCHIP_READ(Sn_TTL(sn))
1594|
1602| #define  setSn_RXMEM_SIZE(sn, rxmemsize) \
```

```
1603        WIZCHIP_WRITE(RMSR,
    (WIZCHIP_READ(RMSR) & ~(0x03 << (2*sn))) |
    (rxmemsize << (2*sn)))
1604  #define  setSn_RXBUF_SIZE(sn,rxmemsize)
    setSn_RXMEM_SIZE(sn,rxmemsize)
1605
1613  #define  getSn_RXMEM_SIZE(sn) \
1614        ((WIZCHIP_READ(RMSR) & (0x03 <<
    (2*sn))) >> (2*sn))
1615  #define  getSn_RXBUF_SIZE(sn)
    getSn_RXMEM_SIZE(sn)
1616
1624  #define setSn_TXMEM_SIZE(sn, txmemsize) \
1625        WIZCHIP_WRITE(TMSR,
    (WIZCHIP_READ(TMSR) & ~(0x03 << (2*sn))) |
    (txmemsize << (2*sn)))
1626  #define  setSn_TXBUF_SIZE(sn, txmemsize)
    setSn_TXMEM_SIZE(sn,txmemsize)
1627
1635  #define  getSn_TXMEM_SIZE(sn) \
1636        ((WIZCHIP_READ(TMSR) & (0x03 <<
    (2*sn))) >> (2*sn))
1637  #define  getSn_TXBUF_SIZE(sn)
    getSn_TXMEM_SIZE(sn)
1638
1645  uint16_t getSn_TX_FSR(uint8_t sn);
1646
1653  #define getSn_TX_RD(sn) \
1654
    (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1
    )))
1655
1663  #define setSn_TX_WR(sn, txwr) { \
1664          WIZCHIP_WRITE(Sn_TX_WR(sn),
    (uint8_t)(txwr>>8)); \
1665
```

```
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),
    1), (uint8_t) txwr); \
1666            }
1667
1675 #define getSn_TX_WR(sn) \
1676
    (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    )))
1677
1684 uint16_t getSn_RX_RSR(uint8_t sn);
1685
1693 #define setSn_RX_RD(sn, rxrd) { \
1694            WIZCHIP_WRITE(Sn_RX_RD(sn),
    (uint8_t)(rxrd>>8)); \
1695
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),
    1), (uint8_t) rxrd); \
1696        }
1697
1705 #define getSn_RX_RD(sn) \
1706
    (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    )))
1707
1715 #define setSn_RX_WR(sn, rxwr) { \
1716            WIZCHIP_WRITE(Sn_RX_WR(sn),
    (uint8_t)(rxwr>>8)); \
1717
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),
    1), (uint8_t) rxwr); \
1718        }
1719
1720
1727 #define getSn_RX_WR(sn) \
1728
```

```
    (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1
    )))
1729|
1737| #define setSn_FRAG(sn, frag) { \
1738|         WIZCHIP_WRITE(Sn_FRAG(sn),
    (uint8_t)(frag >>8)); \
1739|
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1
    ), (uint8_t) frag); \
1740|     }
1741|
1749| #define getSn_FRAG(sn) \
1750|
    (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)
    ))
1751|
1758| #define getSn_RxMAX(sn) \
1759|         ((uint16_t)(1 <<
    getSn_RXMEM_SIZE(sn)) << 10)
1760|
1761|
1768| #define getSn_TxMAX(sn) \
1769|         ((uint16_t)(1 <<
    getSn_TXMEM_SIZE(sn)) << 10)
1770|
1777| #define getSn_RxMASK(sn) \
1778|         (getSn_RxMAX(sn) - 1)
1779|
1786| #define getSn_TxMASK(sn) \
1787|         (getSn_TxMAX(sn) - 1)
1788|
1795| uint32_t getSn_RxBASE(uint8_t sn);
1796|
1803| uint32_t getSn_TxBASE(uint8_t sn);
1804|
```

```
1806 // Sn_TXBUF & Sn_RXBUF IO function //
1808
1822 void wiz_send_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len);
1823
1838 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len);
1839
1847 void wiz_recv_ignore(uint8_t sn, uint16_t
    len);
1848
1850 #endif
1851
1853 #endif //_W5100_H_
1854
1855
1856
```

Socket APIs

socket.h

Go to the documentation of this file.

```
  1 //*****************************************************************************
  2 //
 47 //
 48 //*****************************************************************************
 85 #ifndef _SOCKET_H_
 86 #define _SOCKET_H_
 87
 88 #include "wizchip_conf.h"
 89
 90 #define SOCKET                  uint8_t
 91
 92 #define SOCK_OK                 1
 93 #define SOCK_BUSY               0
 94 #define SOCK_FATAL              -1000
 95
 96 #define SOCK_ERROR              0
 97 #define SOCKERR_SOCKNUM         (SOCK_ERROR - 1)
 98 #define SOCKERR_SOCKOPT         (SOCK_ERROR - 2)
 99 #define SOCKERR_SOCKINIT        (SOCK_ERROR - 3)
100 #define SOCKERR_SOCKCLOSED      (SOCK_ERROR - 4)
```

```
101 #define SOCKERR_SOCKMODE       (SOCK_ERROR -
    5)
102 #define SOCKERR_SOCKFLAG       (SOCK_ERROR -
    6)
103 #define SOCKERR_SOCKSTATUS     (SOCK_ERROR -
    7)
104 #define SOCKERR_ARG            (SOCK_ERROR -
    10)
105 #define SOCKERR_PORTZERO       (SOCK_ERROR -
    11)
106 #define SOCKERR_IPINVALID      (SOCK_ERROR -
    12)
107 #define SOCKERR_TIMEOUT        (SOCK_ERROR -
    13)
108 #define SOCKERR_DATALEN        (SOCK_ERROR -
    14)
109 #define SOCKERR_BUFFER         (SOCK_ERROR -
    15)
110
111 #define SOCKFATAL_PACKLEN      (SOCK_FATAL -
    1)
112
113 /*
114  * SOCKET FLAG
115  */
116 #define SF_ETHER_OWN            (Sn_MR_MFEN)
117 #define SF_IGMP_VER2            (Sn_MR_MC)
118 #define SF_TCP_NODELAY          (Sn_MR_ND)
119 #define SF_MULTI_ENABLE         (Sn_MR_MULTI)
120
121 #if _WIZCHIP_ == 5500
122    #define SF_BROAD_BLOCK
    (Sn_MR_BCASTB)
123    #define SF_MULTI_BLOCK
    (Sn_MR_MMB)
124    #define SF_IPv6_BLOCK
    (Sn_MR_MIP6B)
```

```
125     #define SF_UNI_BLOCK
    (Sn_MR_UCASTB)
126 #endif
127
128 //A201505 : For W5300
129 #if _WIZCHIP_ == 5300
130     #define SF_TCP_ALIGN          0x02
131 #endif
132
133 #define SF_IO_NONBLOCK            0x01
134
135 /*
136  * UDP & MACRAW Packet Infomation
137  */
138 #define PACK_FIRST                0x80
139 #define PACK_REMAINED             0x01
140 #define PACK_COMPLETED            0x00
141 //A20150601 : For Integrating with W5300
142 #define PACK_FIFOBYTE             0x02
143 //
144
162 int8_t  socket(uint8_t sn, uint8_t protocol,
    uint16_t port, uint8_t flag);
163
174 int8_t  close(uint8_t sn);
175
187 int8_t  listen(uint8_t sn);
188
210 int8_t  connect(uint8_t sn, uint8_t * addr,
    uint16_t port);
211
227 int8_t  disconnect(uint8_t sn);
228
247 int32_t send(uint8_t sn, uint8_t * buf,
    uint16_t len);
248
269 int32_t recv(uint8_t sn, uint8_t * buf,
```

```
    uint16_t len);
270
297 int32_t sendto(uint8_t sn, uint8_t * buf,
    uint16_t len, uint8_t * addr, uint16_t port);
298
327 int32_t recvfrom(uint8_t sn, uint8_t * buf,
    uint16_t len, uint8_t * addr, uint16_t *port);
328
329
331 // SOCKET CONTROL & OPTION //
333 #define SOCK_IO_BLOCK         0
334 #define SOCK_IO_NONBLOCK      1
335
336
345 typedef enum
346 {
347    SIK_CONNECTED     = (1 << 0),
348    SIK_DISCONNECTED  = (1 << 1),
349    SIK_RECEIVED      = (1 << 2),
350    SIK_TIMEOUT       = (1 << 3),
351    SIK_SENT          = (1 << 4),
352    //M20150410 : Remove the comma of last
    member
353    //SIK_ALL           = 0x1F,        ///<
    all interrupt
354    SIK_ALL           = 0x1F
355 }sockint_kind;
356
361 typedef enum
362 {
363    CS_SET_IOMODE,
364    CS_GET_IOMODE,
365    CS_GET_MAXTXBUF,
366    CS_GET_MAXRXBUF,
367    CS_CLR_INTERRUPT,
368    CS_GET_INTERRUPT,
369 #if _WIZCHIP_ > 5100
```

```
370     CS_SET_INTMASK,
371     CS_GET_INTMASK
372  #endif
373  }ctlsock_type;
374
375
380  typedef enum
381  {
382     SO_FLAG,
383     SO_TTL,
384     SO_TOS,
385     SO_MSS,
386     SO_DESTIP,
387     SO_DESTPORT,
388  #if _WIZCHIP_ != 5100
389     SO_KEEPALIVESEND,
390     #if _WIZCHIP_ > 5200
391        SO_KEEPALIVEAUTO,
392     #endif
393  #endif
394     SO_SENDBUF,
395     SO_RECVBUF,
396     SO_STATUS,
397     SO_REMAINSIZE,
398     SO_PACKINFO
399  }sockopt_type;
400
418  int8_t  ctlsocket(uint8_t sn, ctlsock_type
     cstype, void* arg);
419
446  int8_t  setsockopt(uint8_t sn, sockopt_type
     sotype, void* arg);
447
480  int8_t  getsockopt(uint8_t sn, sockopt_type
     sotype, void* arg);
481
482  #endif   // _SOCKET_H_
```

Socket APIs

w5300.h

Go to the documentation of this file.

```
  1 #ifndef _W5300_H_
  2 #define _W5300_H_
  3 //*****************************************************************************
  4 //
 43 //
 44 //*****************************************************************************
 45
 46 #include <stdint.h>
 47 #include "wizchip_conf.h"
 48
 50 #if   (_WIZCHIP_ == 5300)
 51
 53 #define _WIZCHIP_SN_BASE_  (0x0200)
 54 #define _WIZCHIP_SN_SIZE_  (0x0040)
 55
 56
 57 #define WIZCHIP_CREG_BLOCK 0x00
 58 #define WIZCHIP_SREG_BLOCK(N) (_WIZCHIP_SN_BASE_+ _WIZCHIP_SN_SIZE_*N)
 59
 60 #define WIZCHIP_OFFSET_INC(ADDR, N)    (ADDR + N)
 61
```

```
62 #if (_WIZCHIP_IO_MODE_ ==
   _WIZCHIP_IO_MODE_BUS_DIR_)
63    #define _W5300_IO_BASE_
   _WIZCHIP_IO_BASE_
64 #elif (_WIZCHIP_IO_MODE_ ==
   _WIZCHIP_IO_MODE_BUS_INDIR_)
65     #define IDM_AR
   ((_WIZCHIP_IO_BASE_ + 0x0002))
66     #define IDM_DR
   ((_WIZCHIP_IO_BASE_ + 0x0004))
67     #define _W5300_IO_BASE_    0x0000
68 #elif (_WIZCHIP_IO_MODE_ &
   _WIZCHIP_IO_MODE_SPI_)
69    #error "Unkonw _WIZCHIP_IO_MODE_"
70 #endif
71
73 // Definition For Legacy Chip Driver //
75 #define IINCHIP_READ(ADDR)
   WIZCHIP_READ(ADDR)
76 #define IINCHIP_WRITE(ADDR,VAL)
   WIZCHIP_WRITE(ADDR,VAL)
77 //#define IINCHIP_READ_BUF(ADDR,BUF,LEN)
   WIZCHIP_READ_BUF(ADDR,BUF,LEN)
78 //#define IINCHIP_WRITE_BUF(ADDR,BUF,LEN)
   WIZCHIP_WRITE(ADDR,BUF,LEN)
79
80 //-------------------------  defgroup -----
   ---------------------------
199 //------------------------------ defgroup
   end ----------------------------------------
   --
200
201 //----------------------------- W5300 Common
   Registers ----------------------------
224 #define MR              (_WIZCHIP_IO_BASE_)
225
246 #define IR              (_W5300_IO_BASE_ +
```

```
    0x02)
247
256 #define _IMR_              (_W5300_IO_BASE_ +
    0x04)
257
258
259 //#define ICFGR            (_W5300_IO_BASE_ +
    0x06)
260 //#define INTLEVEL         ICFGR
261
267 #define SHAR              (_W5300_IO_BASE_ +
    0x08)
268
269
275  #define GAR               (_W5300_IO_BASE_ +
    0x10)
276
282 #define SUBR              (_W5300_IO_BASE_ +
    0x14)
283
289 #define SIPR              (_W5300_IO_BASE_ +
    0x18)
290
299  #define _RTR_            (_W5300_IO_BASE_ +
    0x1C)
300
307 #define _RCR_             (_W5300_IO_BASE_ +
    0x1E)
308
316 #define TMS01R            (_W5300_IO_BASE_ +
    0x20)
317
323 #define TMS23R            (TMS01R + 2)
324
330 #define TMS45R            (TMS01R + 4)
331
337 #define TMS67R            (TMS01R + 6)
```

```
338
344 #define TMSR0           TMS01R
345
351 #define TMSR1           (TMSR0 + 1)
352
358 #define TMSR2           (TMSR0 + 2)
359
365 #define TMSR3           (TMSR0 + 3)
366
372 #define TMSR4           (TMSR0 + 4)
373
379 #define TMSR5           (TMSR0 + 5)
380
386 #define TMSR6           (TMSR0 + 6)
387
393 #define TMSR7           (TMSR0 + 7)
394
395
403 #define RMS01R          (_W5300_IO_BASE_ +
    0x28)
404
410 #define RMS23R          (RMS01R + 2)
411
417 #define RMS45R          (RMS01R + 4)
418
424 #define RMS67R          (RMS01R + 6)
425
431 #define RMSR0           RMS01R
432
438 #define RMSR1           (RMSR0 + 1)
439
445 #define RMSR2           (RMSR0 + 2)
446
452 #define RMSR3           (RMSR0 + 3)
453
459 #define RMSR4           (RMSR0 + 4)
460
```

```
466 #define RMSR5           (RMSR0 + 5)
467
473 #define RMSR6           (RMSR0 + 6)
474
480 #define RMSR7           (RMSR0 + 7)
481
482
483
493 #define MTYPER          (_W5300_IO_BASE_ +
    0x30)
494
503 #define PATR            (_W5300_IO_BASE_ +
    0x32)
504
505 //#define PPPALGOR      (_W5300_IO_BASE_ +
    0x34)
506
512 #define PTIMER          (_W5300_IO_BASE_ +
    0x36)
513
519 #define PMAGICR         (_W5300_IO_BASE_ +
    0x38)
520
521 //#define PSTATER       (_W5300_IO_BASE_ +
    0x3A)
522
528 #define PSIDR           (_W5300_IO_BASE_ +
    0x3C)
529
535 #define PDHAR           (_W5300_IO_BASE_ +
    0x40)
536
545 #define UIPR            (_W5300_IO_BASE_ +
    0x48)
546
552 #define UPORTR          (_W5300_IO_BASE_ +
    0x4C)
```

```
553
561 #define FMTUR           (_W5300_IO_BASE_ +
    0x4E)
562
563 //#define Sn_RTCR(n)      (_W5300_IO_BASE_ +
    0x50 + n*2)
564
584 #define Pn_BRDYR(n)     (_W5300_IO_BASE_ +
    0x60 + n*4)
585
594 #define Pn_BDPTHR(n)    (_W5300_IO_BASE_ +
    0x60 + n*4 + 2)
595
601 #define IDR             (_W5300_IO_BASE_ +
    0xFE)
602 #define VERSIONR        IDR
603
604
605 //------------------------------ W5300 SOCKET
    Registers ------------------------------
606
642 #define Sn_MR(n)        (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x00)
643
666 #define Sn_CR(n)        (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x02)
667
676 #define Sn_IMR(n)       (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x04)
677
699 #define Sn_IR(n)        (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x06)
700
725 #define Sn_SSR(n)       (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x08)
726 #define Sn_SR(n)        Sn_SSR(n)
727
```

```
728
734 #define Sn_PORTR(n)     (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x0A)
735 #define Sn_PORT(n)      Sn_PORTR(n)
736
737
743 #define Sn_DHAR(n)      (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x0C)
744
753 #define Sn_DPORTR(n)    (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x12)
754 #define Sn_DPORT(n)     Sn_DPORTR(n)
755
756
765  #define Sn_DIPR(n)      (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x14)
766
772 #define Sn_MSSR(n)      (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x18)
773
785 #define Sn_KPALVTR(n)   (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x1A)
786
793 #define Sn_PROTOR(n)    Sn_KPALVTR(n)
794
795
802 #define Sn_TOSR(n)      (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x1C)
803 #define Sn_TOS(n)       Sn_TOSR(n)
804
805
811 #define Sn_TTLR(n)      (_W5300_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(n) + 0x1E)
812 #define Sn_TTL(n)       Sn_TTLR(n)
813
814
821 #define Sn_TX_WRSR(n)       (_W5300_IO_BASE_
```

```
    + WIZCHIP_SREG_BLOCK(n) + 0x20)
 822
 832 #define Sn_TX_FSR(n)        (_W5300_IO_BASE_
    + WIZCHIP_SREG_BLOCK(n) + 0x0024)
 833
 841 #define Sn_RX_RSR(n)        (_W5300_IO_BASE_
    + WIZCHIP_SREG_BLOCK(n) + 0x0028)
 842
 848 #define Sn_FRAGR(n)         (_W5300_IO_BASE_
    + WIZCHIP_SREG_BLOCK(n) + 0x002C)
 849 #define Sn_FRAG(n)          Sn_FRAGR(n)
 850
 859 #define Sn_TX_FIFOR(n)      (_W5300_IO_BASE_
    + WIZCHIP_SREG_BLOCK(n) + 0x2E)
 860
 869 #define Sn_RX_FIFOR(n)      (_W5300_IO_BASE_
    + WIZCHIP_SREG_BLOCK(n) + 0x30)
 870
 871 //#define Sn_TX_SADR(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x32)
 872
 873 //#define Sn_RX_SADR(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x34)
 874
 875 //#define Sn_TX_RD(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x36)
 876
 877 //#define Sn_TX_WR(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x38)
 878
 879 //#define Sn_TX_ACK(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x3A)
```

```
880
881 //#define Sn_RX_RD(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x3C)
882
883 //#define Sn_RX_WR(n)
    (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
    0x3E)
884
885
886 /**************************************/
887 /* The bit of MR regsiter defintion */
888 /**************************************/
889 #define MR_DBW              (1 << 15)
890 #define MR_MPF              (1 << 14)
891 #define MR_WDF(X)           ((X & 0x07) <<
    11)
892 #define MR_RDH              (1 << 10)
893 #define MR_FS               (1 << 8)
894 #define MR_RST              (1 << 7)
895 #define MR_MT               (1 << 5)
896 #define MR_PB               (1 << 4)
897 #define MR_PPPoE            (1 << 3)
898 #define MR_DBS              (1 << 2)
899 #define MR_IND              (1 << 0)
902 /**************************************/
903 /* The bit of IR regsiter defintion */
904 /**************************************/
905 #define IR_IPCF             (1 << 15)
906 #define IR_DPUR             (1 << 14)
907 #define IR_PPPT             (1 << 13)
908 #define IR_FMTU             (1 << 12)
909 #define IR_SnINT(n)         (0x01 << n)
911 /*****************************************/
912 /* The bit of Pn_BRDYR regsiter defintion*/
913 /*****************************************/
914 #define Pn_PEN              (1 << 7)
```

```
915 #define Pn_MT                 (1 << 6)
916 #define Pn_PPL                (1 << 5)
917 #define Pn_SN(n)              ((n & 0x07) << 0)
920 /**************************************/
921 /* The bit of Sn_MR regsiter defintion */
922 /**************************************/
929 #define Sn_MR_ALIGN           (1 << 8)
930
937 #define Sn_MR_MULTI           (1 << 7)
938
947 #define Sn_MR_MF              (1 << 6)
948
955 #define Sn_MR_IGMPv           (1 << 5)
956 #define Sn_MR_MC              Sn_MR_IGMPv
957
958
966 #define Sn_MR_ND              (1 << 5)
967
973 #define Sn_MR_CLOSE           0x00
974
980 #define Sn_MR_TCP             0x01
981
987 #define Sn_MR_UDP             0x02
994 #define Sn_MR_IPRAW           0x03
1002 #define Sn_MR_MACRAW          0x04
1003
1010 #define Sn_MR_PPPoE           0x05
1012 #define SOCK_STREAM           Sn_MR_TCP
1013 #define SOCK_DGRAM            Sn_MR_UDP
1017 /*****************************/
1018 /* The values of CR defintion */
1019 /*****************************/
1034 #define Sn_CR_OPEN            0x01
1035
1045 #define Sn_CR_LISTEN          0x02
1046
1057 #define Sn_CR_CONNECT         0x04
```

```
1058
1070 #define Sn_CR_DISCON       0x08
1071
1076 #define Sn_CR_CLOSE        0x10
1077
1083 #define Sn_CR_SEND         0x20
1084
1093 #define Sn_CR_SEND_MAC     0x21
1094
1101 #define Sn_CR_SEND_KEEP    0x22
1102
1108 #define Sn_CR_RECV         0x40
1110 #define Sn_CR_PCON         0x23
1111 #define Sn_CR_PDISCON      0x24
1112 #define Sn_CR_PCR          0x25
1113 #define Sn_CR_PCN          0x26
1114 #define Sn_CR_PCJ          0x27
1117 /*********************************/
1118 /* The values of Sn_IR defintion */
1119 /*********************************/
1120 #define Sn_IR_PRECV        0x80
1121 #define Sn_IR_PFAIL        0x40
1122 #define Sn_IR_PNEXT        0x20
1123 #define Sn_IR_SENDOK       0x10
1124 #define Sn_IR_TIMEOUT      0x08
1125 #define Sn_IR_RECV         0x04
1126 #define Sn_IR_DISCON       0x02
1127 #define Sn_IR_CON          0x01
1129 /*********************************/
1130 /* The values of Sn_SSR defintion */
1131 /*********************************/
1137 #define SOCK_CLOSED        0x00
1138
1144 #define SOCK_ARP           0x01
1152 #define SOCK_INIT          0x13
1153
1160 #define SOCK_LISTEN        0x14
```

```
1161
1169 #define SOCK_SYNSENT       0x15
1170
1177 #define SOCK_SYNRECV       0x16
1178
1186 #define SOCK_ESTABLISHED   0x17
1187
1194 #define SOCK_FIN_WAIT      0x18
1195
1202 #define SOCK_CLOSING       0x1A
1203
1210 #define SOCK_TIME_WAIT     0x1B
1211
1218 #define SOCK_CLOSE_WAIT    0x1C
1219
1225 #define SOCK_LAST_ACK      0x1D
1226
1233 #define SOCK_UDP           0x22
1234
1241 #define SOCK_IPRAW         0x32
1242
1249 #define SOCK_MACRAW        0x42
1257 #define SOCK_PPPoE         0x5F
1259 /* IP PROTOCOL */
1260 #define IPPROTO_IP                  0
    //< Dummy for IP
1261 #define IPPROTO_ICMP                1
    //< Control message protocol
1262 #define IPPROTO_IGMP                2
    //< Internet group management protocol
1263 #define IPPROTO_GGP                 3
    //< Gateway^2 (deprecated)
1264 #define IPPROTO_TCP                 6
    //< TCP
1265 #define IPPROTO_PUP                 12
    //< PUP
1266 #define IPPROTO_UDP                 17
```

```
    //< UDP
1267| #define IPPROTO_IDP                 22
    //< XNS idp
1268| #define IPPROTO_ND                  77
    //< UNOFFICIAL net disk protocol
1269| #define IPPROTO_RAW                 255
    //< Raw IP packet
1270|
1271|
1283| #define WIZCHIP_CRITICAL_ENTER()
    WIZCHIP.CRIS._enter()
1284|
1285| #ifdef _exit
1286| #undef _exit
1287| #endif
1288|
1300| #define WIZCHIP_CRITICAL_EXIT()
    WIZCHIP.CRIS._exit()
1301|
1303| // Basic I/O Function //
1305|
1312| uint16_t  WIZCHIP_READ (uint32_t AddrSel);
1313|
1321| void WIZCHIP_WRITE(uint32_t AddrSel,
    uint16_t wb );
1322|
1323| /**********************************
1324|  * COMMON Register Access Function *
1325|  **********************************/
1326|
1333| #if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
1334|    #if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
1335|       #define setMR(mr) \
1336|          (*((uint8_t*)MR) = (uint8_t)((mr)
    >> 8)); (*((uint8_t*)WIZCHIP_OFFSET_INC(MR,1))
    = (uint8_t)((mr) & 0xFF))
```

```
1337    #elif (_WIZCHIP_IO_BUS_WIDTH_ == 16)
1338       #define setMR(mr)    (*
    ((uint16_t*)MR)) = (uint16_t)((mr) & 0xFFFF))
1339    #else
1340       #error "Unknown
    _WIZCHIP_IO_BUS_WIDTH_. You should be define
    _WIZCHIP_IO_BUS_WIDTH as 8 or 16."
1341    #endif
1342 #else
1343    #error "Unknown _WIZCHIP_IO_MODE_"
1344 #endif
1345
1352 #if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
1353    #if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
1354       #define getMR()    (((uint16_t)(*
    ((uint8_t*)MR)) << 8) + (((uint16_t)(*
    ((uint8_t*)WIZCHIP_OFFSET_INC(MR,1)))) &
    0x00FF))
1355    #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
1356       #define getMR()    (*((uint16_t*)MR))
1357    #else
1358       #error "Unknown
    _WIZCHIP_IO_BUS_WIDTH_. You should be define
    _WIZCHIP_IO_BUS_WIDTH as 8 or 16."
1359    #endif
1360 #else
1361    #error "Unknown _WIZCHIP_IO_MODE_"
1362 #endif
1363
1370 #define  setIR(ir) \
1371    WIZCHIP_WRITE(IR, ir & 0xF0FF)
1372
1379 #define  getIR()  \
1380    (WIZCHIP_READ(IR) & 0xF0FF)
1381
1382
```

```
1389 #define  setIMR(imr) \
1390    WIZCHIP_WRITE(_IMR_, imr & 0xF0FF)
1391
1398 #define  getIMR() \
1399    (WIZCHIP_READ(_IMR_) & 0xF0FF)
1400
1407 #define  setSHAR(shar) { \
1408       WIZCHIP_WRITE(SHAR,
    (((uint16_t)((shar)[0])) << 8) + (((uint16_t)
    ((shar)[1])) & 0x00FF)); \
1409
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SHAR,2),
    (((uint16_t)((shar)[2])) << 8) + (((uint16_t)
    ((shar)[3])) & 0x00FF)); \
1410
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SHAR,4),
    (((uint16_t)((shar)[4])) << 8) + (((uint16_t)
    ((shar)[5])) & 0x00FF)); \
1411    }
1412
1419 #define  getSHAR(shar) { \
1420         (shar)[0] = (uint8_t)
    (WIZCHIP_READ(SHAR) >> 8); \
1421         (shar)[1] = (uint8_t)
    (WIZCHIP_READ(SHAR));      \
1422         (shar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,2)) >>
    8); \
1423         (shar)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,2))); \
1424         (shar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,4)) >>
    8); \
1425         (shar)[5] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SHAR,4))); \
1426    }
1427
```

```
1434 #define  setGAR(gar) { \
1435       WIZCHIP_WRITE(GAR,
    (((uint16_t)((gar)[0])) << 8) + (((uint16_t)
    ((gar)[1])) & 0x00FF)); \
1436
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(GAR,2),
    (((uint16_t)((gar)[2])) << 8) + (((uint16_t)
    ((gar)[3])) & 0x00FF)); \
1437    }
1438
1445 #define  getGAR(gar) { \
1446         (gar)[0] = (uint8_t)
    (WIZCHIP_READ(GAR) >> 8); \
1447         (gar)[1] = (uint8_t)
    (WIZCHIP_READ(GAR));      \
1448         (gar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(GAR,2)) >>
    8); \
1449         (gar)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(GAR,2))); \
1450    }
1451
1458 #define  setSUBR(subr) { \
1459       WIZCHIP_WRITE(SUBR,
    (((uint16_t)((subr)[0])) << 8) + (((uint16_t)
    ((subr)[1])) & 0x00FF)); \
1460
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SUBR,2),
    (((uint16_t)((subr)[2])) << 8) + (((uint16_t)
    ((subr)[3])) & 0x00FF)); \
1461    }
1462
1469 #define  getSUBR(subr) { \
1470         (subr)[0] = (uint8_t)
    (WIZCHIP_READ(SUBR) >> 8); \
1471         (subr)[1] = (uint8_t)
    (WIZCHIP_READ(SUBR));      \
```

```
1472          (subr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2)) >>
    8); \
1473          (subr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2))); \
1474      }
1475
1482  #define  setSIPR(sipr) { \
1483        WIZCHIP_WRITE(SIPR,
    (((uint16_t)((sipr)[0])) << 8) + (((uint16_t)
    ((sipr)[1])) & 0x00FF)); \
1484
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SIPR,2),
    (((uint16_t)((sipr)[2])) << 8) + (((uint16_t)
    ((sipr)[3])) & 0x00FF)); \
1485      }
1486
1493  #define  getSIPR(sipr) { \
1494          (sipr)[0] = (uint8_t)
    (WIZCHIP_READ(SIPR) >> 8); \
1495          (sipr)[1] = (uint8_t)
    (WIZCHIP_READ(SIPR));      \
1496         (sipr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SIPR,2)) >>
    8); \
1497          (sipr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SIPR,2))); \
1498      }
1499
1500
1507  #define  setRTR(rtr)   \
1508          WIZCHIP_WRITE(_RTR_, rtr)
1509
1516  #define  getRTR() \
1517          WIZCHIP_READ(_RTR_)
1518
1525  #define  setRCR(rcr) \
```

```
1526          WIZCHIP_WRITE(_RCR_,
    ((uint16_t)rcr)&0x00FF)
1527
1534 #define  getRCR() \
1535          ((uint8_t)(WIZCHIP_READ(_RCR_) &
    0x00FF))
1536
1543 #define  setTMS01R(tms01r) \
1544    WIZCHIP_WRITE(TMS01R,tms01r)
1545
1552 #define  getTMS01R()  \
1553    WIZCHIP_READ(TMS01R)
1554
1561 #define  setTMS23R(tms23r) \
1562    WIZCHIP_WRITE(TMS23R,tms23r)
1563
1570 #define  getTMS23R()  \
1571    WIZCHIP_READ(TMS23R)
1572
1579 #define  setTMS45R(tms45r) \
1580    WIZCHIP_WRITE(TMS45R,tms45r)
1581
1588 #define  getTMS45R()  \
1589    WIZCHIP_READ(TMS45R)
1590
1597 #define  setTMS67R(tms67r) \
1598    WIZCHIP_WRITE(TMS67R,tms67r)
1599
1606 #define  getTMS67R()  \
1607    WIZCHIP_READ(TMS67R)
1608
1616 void setTMSR(uint8_t sn,uint8_t tmsr);
1617 #define setSn_TXBUF_SIZE(sn, tmsr)
    setTMSR(sn, tmsr)
1618
1619
1626 uint8_t getTMSR(uint8_t sn);
```

```
1627 #define getSn_TXBUF_SIZE(sn)  getTMSR(sn)
1628
1629
1635 #define setRMS01R(rms01r) \
1636    WIZCHIP_WRITE(RMS01R,rms01r)
1637
1644 #define getRMS01R()  \
1645    WIZCHIP_READ(RMS01R)
1646
1653 #define setRMS23R(rms23r) \
1654    WIZCHIP_WRITE(RMS23R,rms23r)
1655
1662 #define getRMS23R()  \
1663    WIZCHIP_READ(RMS23R)
1664
1671 #define setRMS45R(rms45r) \
1672    WIZCHIP_WRITE(RMS45R,rms45r)
1673
1680 #define getRMS45R()  \
1681    WIZCHIP_READ(RMS45R)
1682
1689 #define setRMS67R(rms67r) \
1690    WIZCHIP_WRITE(RMS67R,rms67r)
1691
1698 #define getRMS67R()  \
1699    WIZCHIP_READ(RMS67R)
1700
1708 void setRMSR(uint8_t sn,uint8_t rmsr);
1709 #define setSn_RXBUF_SIZE(sn,rmsr)
    setRMSR(sn, rmsr)
1710
1711
1718 uint8_t getRMSR(uint8_t sn);
1719 #define getSn_RXBUF_SIZE(sn)  getRMSR(sn)
1720
1721
1727 #define setMTYPER(mtype) \
```

```
1728    WIZCHIP_WRITE(MTYPER, mtype)
1729
1736 #define getMTYPER() \
1737    WIZCHIP_READ(MTYPER)
1738
1744 #define getPATR() \
1745    WIZCHIP_READ(PATR)
1746
1753 #define setPTIMER(ptimer) \
1754         WIZCHIP_WRITE(PTIMER,
    ((uint16_t)ptimer) & 0x00FF)
1755
1762 #define getPTIMER() \
1763         ((uint8_t)(WIZCHIP_READ(PTIMER) &
    0x00FF))
1764
1771 #define setPMAGIC(pmagic) \
1772         WIZCHIP_WRITE(PMAGIC,
    ((uint16_t)pmagic) & 0x00FF)
1773
1780 #define getPMAGIC() \
1781         ((uint8_t)(WIZCHIP_READ(PMAGIC) &
    0x00FF))
1782
1788 #define getPSIDR() \
1789         WIZCHIP_READ(PSIDR)
1790
1796 #define getPDHAR(pdhar) { \
1797         (pdhar)[0] = (uint8_t)
    (WIZCHIP_READ(PDHAR) >> 8); \
1798         (pdhar)[1] = (uint8_t)
    (WIZCHIP_READ(PDHAR));      \
1799         (pdhar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,2)) >>
    8); \
1800         (pdhar)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,2))); \
```

```
1801          (pdhar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4)) >>
    8); \
1802          (pdhar)[5] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4))); \
1803      }
1804
1810  #define getUIPR(uipr) { \
1811          (uipr)[0] = (uint8_t)
    (WIZCHIP_READ(UIPR) >> 8); \
1812          (uipr)[1] = (uint8_t)
    (WIZCHIP_READ(UIPR));      \
1813          (uipr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,2)) >>
    8); \
1814          (uipr)[3] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,2))); \
1815      }
1816
1822  #define getUPORTR() \
1823       WIZCHIP_READ(UPORTR)
1824
1830  #define getFMTUR() \
1831      WIZCHIP_READ(FMTUR)
1832
1833
1839  #define getPn_BRDYR(p) \
1840      ((uint8_t)(WIZCHIP_READ(Pn_BRDYR(p)) &
    0x00FF))
1841
1848  #define setPn_BRDYR(p, brdyr) \
1849      WIZCHIP_WRITE(Pn_BRDYR(p), brdyr &
    0x00E7)
1850
1857  #define getPn_BDPTHR(p) \
1858      WIZCHIP_READ(Pn_BDPTHR(p))
1859
```

```
1866 #define setPn_BDPTHR(p, bdpthr) \
1867    WIZCHIP_WRITE(Pn_BDPTHR(p),bdpthr)
1868
1869
1875 #define getIDR() \
1876    WIZCHIP_READ(IDR)
1877
1878
1879 /**********************************
1880  * SOCKET Register Access Function *
1881  **********************************/
1882
1890 #define setSn_MR(sn, mr) \
1891    WIZCHIP_WRITE(Sn_MR(sn),mr)
1892
1900 #define getSn_MR(sn) \
1901     WIZCHIP_READ(Sn_MR(sn))
1902
1910 #define setSn_CR(sn, cr) \
1911    WIZCHIP_WRITE(Sn_CR(sn), ((uint16_t)cr) &
    0x00FF)
1912
1920 #define getSn_CR(sn) \
1921    ((uint8_t)WIZCHIP_READ(Sn_CR(sn)))
1922
1930 #define setSn_IMR(sn, imr) \
1931    WIZCHIP_WRITE(Sn_IMR(sn), ((uint16_t)imr)
    & 0x00FF)
1932
1940 #define getSn_IMR(sn) \
1941    ((uint8_t)WIZCHIP_READ(Sn_IMR(sn)))
1942
1950 #define setSn_IR(sn, ir) \
1951    WIZCHIP_WRITE(Sn_IR(sn), ((uint16_t)ir) &
    0x00FF)
1952
1960 #define getSn_IR(sn) \
```

```
1961     ((uint8_t)WIZCHIP_READ(Sn_IR(sn)))
1962
1969 #define getSn_SSR(sn) \
1970     ((uint8_t)WIZCHIP_READ(Sn_SR(sn)))
1971 #define getSn_SR(sn) getSn_SSR(sn)
1972
1973
1980 #define setSn_PORTR(sn, port) \
1981     WIZCHIP_WRITE(Sn_PORTR(sn), port)
1982 #define setSn_PORT(sn, port)
    setSn_PORTR(sn, port)
1983
1984
1991 #define getSn_PORTR(sn, port) \
1992     WIZCHIP_READ(Sn_PORTR(sn))
1993 #define getSn_PORT(sn)   getSn_PORTR(sn)
1994
1995
2002 #define setSn_DHAR(sn, dhar) { \
2003         WIZCHIP_WRITE(Sn_DHAR(sn),
    (((uint16_t)((dhar)[0])) << 8) + (((uint16_t)
    ((dhar)[1])) & 0x00FF)); \
2004
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2
    ), (((uint16_t)((dhar)[0])) << 8) +
    (((uint16_t)((dhar)[1])) & 0x00FF)); \
2005
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4
    ), (((uint16_t)((dhar)[0])) << 8) +
    (((uint16_t)((dhar)[1])) & 0x00FF)); \
2006     }
2007
2015 #define getSn_DHAR(sn, dhar) { \
2016         (dhar)[0] = (uint8_t)
    (WIZCHIP_READ(Sn_DHAR(sn)) >> 8); \
2017         (dhar)[1] = (uint8_t)
    WIZCHIP_READ(Sn_DHAR(sn)); \
```

```
2018        (dhar)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2
    )) >> 8); \
2019        (dhar)[3] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),2)
    ); \
2020        (dhar)[4] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4
    )) >> 8); \
2021        (dhar)[5] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DHAR(sn),4)
    ); \
2022    }
2023
2031 #define setSn_DPORTR(sn, dport) \
2032    WIZCHIP_WRITE(Sn_DPORTR(sn),dport)
2033 #define setSn_DPORT(sn, dport)
    setSn_DPORTR(sn,dport)
2034
2035
2045 #define getSn_DPORTR(sn) \
2046    WIZCHIP_READ(Sn_DPORTR(sn))
2047 #define getSn_DPORT(sn) getSn_DPORTR(sn)
2048
2049
2056 #define setSn_DIPR(sn, dipr) { \
2057         WIZCHIP_WRITE(Sn_DIPR(sn),
    (((uint16_t)((dipr)[0])) << 8) + (((uint16_t)
    ((dipr)[1])) & 0x00FF)); \
2058
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2
    ), (((uint16_t)((dipr)[2])) << 8) +
    (((uint16_t)((dipr)[3])) & 0x00FF)); \
2059    }
2060
2068 #define getSn_DIPR(sn, dipr) { \
2069         (dipr)[0] = (uint8_t)
```

```
    (WIZCHIP_READ(Sn_DIPR(sn)) >> 8); \
2070            (dipr)[1] = (uint8_t)
    WIZCHIP_READ(Sn_DIPR(sn));  \
2071            (dipr)[2] = (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2
    )) >> 8); \
2072            (dipr)[3] = (uint8_t)
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2)
    );  \
2073        }
2074
2082    #define setSn_MSSR(sn, mss) \
2083        WIZCHIP_WRITE(Sn_MSSR(sn), mss)
2084
2092    #define getSn_MSSR(sn) \
2093        WIZCHIP_READ(Sn_MSSR(sn))
2094
2102    #define setSn_KPALVTR(sn, kpalvt) \
2103        WIZCHIP_WRITE(Sn_KPALVTR(sn),
    (WIZCHIP_READ(Sn_KPALVTR(sn)) & 0x00FF) |
    (((uint16_t)kpalvt)<<8))
2104
2112    #define getSn_KPALVTR(sn) \
2113            ((uint8_t)
    (WIZCHIP_READ(Sn_KPALVTR(sn)) >> 8))
2114
2122    #define setSn_PROTOR(sn, proto) \
2123        WIZCHIP_WRITE(Sn_PROTOR(sn),
    (WIZCHIP_READ(Sn_PROTOR(sn) & 0xFF00) |
    (((uint16_t)proto) & 0x00FF))
2124    #define setSn_PROTO(sn,proto)
    setSn_PROTOR(sn,proto)
2125
2126
2133    #define getSn_PROTOR(sn) \
2134        ((uint8_t)WIZCHIP_READ(Sn_PROTOR(sn)))
2135    #define getSn_PROTO(sn)    getSn_PROTOR(sn)
```

```
2136
2137
2144 #define  setSn_TX_WRSR(sn, txwrs) { \
2145       WIZCHIP_WRITE(Sn_TX_WRSR(sn),
   (uint16_t)(((uint32_t)txwrs) >> 16)); \
2146
   WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn
   ),2), (uint16_t)txwrs); \
2147    }
2148
2156 #define  getSn_TX_WRSR(sn) \
2157    (
   (((uint32_t)WIZCHIP_READ(Sn_TX_WRSR(sn))) <<
   16) +
   (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn
   _TX_WRSR(sn),1))) & 0x0000FFFF) )
2158
2165 uint32_t getSn_TX_FSR(uint8_t sn);
2166
2173 uint32_t getSn_RX_RSR(uint8_t sn);
2174
2181 #define  setSn_TX_FIFOR(sn, txfifo) \
2182    WIZCHIP_WRITE(Sn_TX_FIFOR(sn), txfifo);
2183
2190 #define getSn_RX_FIFOR(sn) \
2191    WIZCHIP_READ(Sn_RX_FIFOR(sn));
2192
2200 #define setSn_TOSR(sn, tos) \
2201    WIZCHIP_WRITE(Sn_TOS(sn), ((uint16_t)tos)
   & 0x00FF)
2202 #define setSn_TOS(sn,tos)
   setSn_TOSR(sn,tos)
2203
2204
2211 #define getSn_TOSR(sn) \
2212    ((uint8_t)WIZCHIP_READ(Sn_TOSR(sn)))
2213 #define getSn_TOS(sn)   getSn_TOSR(sn)
```

```
2214
2215
2222 #define setSn_TTLR(sn, ttl) \
2223    WIZCHIP_WRITE(Sn_TTLR(sn),
   ((uint16_t)ttl) & 0x00FF)
2224 #define setSn_TTL(sn,ttl)
   setSn_TTLR(sn,ttl)
2225
2226
2233 #define getSn_TTLR(sn) \
2234    ((uint8_t)WIZCHIP_READ(Sn_TTL(sn)))
2235 #define getSn_TTL(sn)     getSn_TTLR(sn)
2236
2237
2244 #define setSn_FRAGR(sn, frag) \
2245    WIZCHIP_WRITE(Sn_FRAGR(sn),  (uint16_t)
   (frag >>8))
2246 #define setSn_FRAG(sn,frag)
   setSn_FRAGR(sn,flag)
2247
2255 #define getSn_FRAGR(sn) \
2256    (WIZCHIP_READ(Sn_FRAG(sn)) << 8)
2257 #define getSn_FRAG(sn)    getSn_FRAGR(sn)
2258
2259
2261 // Sn_TXBUF & Sn_RXBUF IO function //
2263
2270 #define getSn_RxMAX(sn) \
2271         (((uint32_t)getSn_RXBUF_SIZE(sn)) <<
   10)
2272
2279 #define getSn_TxMAX(sn) \
2280         (((uint32_t)getSn_TXBUF_SIZE(sn)) <<
   10)
2281
2296 void wiz_send_data(uint8_t sn, uint8_t
   *wizdata, uint32_t len);
```

```
2297
2312 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint32_t len);
2313
2321 void wiz_recv_ignore(uint8_t sn, uint32_t
    len);
2322
2324 #endif
2325
2327 #endif   // _W5300_H_
```

Socket APIs

w5200.h

Go to the documentation of this file.

```
    1 //*
      ****************************************************
      ****************************
   38 //
   39 //*************************************************
      ***********************************
   40
   41 #ifndef _W5200_H
   42 #define _W5200_H
   43 #include <stdint.h>
   44 #include "wizchip_conf.h"
   45
   47 #if   (_WIZCHIP_ == 5200)
   48
   50 #define _WIZCHIP_SN_BASE_  (0x4000)
   51 #define _WIZCHIP_SN_SIZE_  (0x0100)
   52 #define _WIZCHIP_IO_TXBUF_ (0x8000) /*
      Internal Tx buffer address of the iinchip */
   53 #define _WIZCHIP_IO_RXBUF_ (0xC000) /*
      Internal Rx buffer address of the iinchip */
   54
   55 #define _W5200_SPI_READ_   (0x00 << 7)
   56 #define _W5200_SPI_WRITE_  (0x01 << 7)
   57
   58 #define WIZCHIP_CREG_BLOCK              0x00
   59 #define WIZCHIP_SREG_BLOCK(N)
```

```
    (_WIZCHIP_SN_BASE_+ _WIZCHIP_SN_SIZE_*N)
  60
  61 #define WIZCHIP_OFFSET_INC(ADDR, N)    (ADDR
    + N)
  62
  63 #if  (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
  64     #define IDM_AR0
    ((_WIZCHIP_IO_BASE_ + 0x0001))
  65     #define IDM_AR1
    ((_WIZCHIP_IO_BASE_ + 0x0002))
  66     #define IDM_DR
    ((_WIZCHIP_IO_BASE_ + 0x0003))
  67     #define _W5200_IO_BASE_    0x0000
  68 #elif (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_)
  69    #define _W5200_IO_BASE_    0x0000
  70 #endif
  71
  73 // Definition For Legacy Chip Driver //
  75 #define IINCHIP_READ(ADDR)
    WIZCHIP_READ(ADDR)
  76 #define IINCHIP_WRITE(ADDR,VAL)
    WIZCHIP_WRITE(ADDR,VAL)
  77 #define IINCHIP_READ_BUF(ADDR,BUF,LEN)
    WIZCHIP_READ_BUF(ADDR,BUF,LEN)
  78 #define IINCHIP_WRITE_BUF(ADDR,BUF,LEN)
    WIZCHIP_WRITE(ADDR,BUF,LEN)
  79
  80
  81 //-----------    defgroup ----------------
    --------------
  82
 185  //-------------------------------------
    ----------------------------------------
 186
 187 //----------------------------- W5200 Common
```

```
    Registers IOMAP ----------------------------
204 | #if  (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
205 |    #define MR
    (_WIZCHIP_IO_BASE_ + (0x0000))  // Mode
206 | #else
207 |    #define MR
    (_W5200_IO_BASE_ + (0x0000))  // Mode
208 | #endif
209 |
215 | #define GAR                 (_W5200_IO_BASE_
    + (0x0001))  // GW Address
216 |
222 | #define SUBR                (_W5200_IO_BASE_
    + (0x0005)) // SN Mask Address
223 |
229 | #define SHAR                (_W5200_IO_BASE_
    + (0x0009)) // Source Hardware Address
230 |
236 | #define SIPR                (_W5200_IO_BASE_
    + (0x000F)) // Source IP Address
237 |
238 | // Reserved                 (_W5200_IO_BASE_
    + (0x0013))
239 | // Reserved                 (_W5200_IO_BASE_
    + (0x0014))
240 |
254 | #define IR                  (_W5200_IO_BASE_
    + (0x0015)) // Interrupt
255 |
265 | #define _IMR_               (_W5200_IO_BASE_
    + (0x0016)) // Socket Interrupt Mask
266 |
275 | #define _RTR_               (_W5200_IO_BASE_
    + (0x0017)) // Retry Time
276 |
283 | #define _RCR_               (_W5200_IO_BASE_
```

```
    + (0x0019)) // Retry Count
284
285 // Reserved                    (_W5200_IO_BASE_
    + (0x001A))
286 // Reserved                    (_W5200_IO_BASE_
    + (0x001B))
287
294 #define PATR              (_W5200_IO_BASE_ +
    (0x001C))
295
302 #define PPPALGO               (_W5200_IO_BASE_
    + (0x001E)) // Authentication Algorithm in
    PPPoE
303
309 #define VERSIONR              (_W5200_IO_BASE_
    + (0x001F)) // Chip version
310
311 // Reserved                    (_W5200_IO_BASE_
    + (0x0020))
312 // Reserved                    (_W5200_IO_BASE_
    + (0x0021))
313 // Reserved                    (_W5200_IO_BASE_
    + (0x0022))
314 // Reserved                    (_W5200_IO_BASE_
    + (0x0023))
315 // Reserved                    (_W5200_IO_BASE_
    + (0x0024))
316 // Reserved                    (_W5200_IO_BASE_
    + (0x0025))
317 // Reserved                    (_W5200_IO_BASE_
    + (0x0026))
318 // Reserved                    (_W5200_IO_BASE_
    + (0x0027))
319
325 #define PTIMER                (_W5200_IO_BASE_
    + (0x0028)) // PPP LCP RequestTimer
326
```

```
332 #define PMAGIC                  (_W5200_IO_BASE_
    + (0x0029)) // PPP LCP Magic number
333
334 // Reserved                     (_W5200_IO_BASE_
    + (0x002A))
335 // Reserved                     (_W5200_IO_BASE_
    + (0x002B))
336 // Reserved                     (_W5200_IO_BASE_
    + (0x002C))
337 // Reserved                     (_W5200_IO_BASE_
    + (0x002D))
338 // Reserved                     (_W5200_IO_BASE_
    + (0x002E))
339 // Reserved                     (_W5200_IO_BASE_
    + (0x002F))
340
346 #define INTLEVEL            (_W5200_IO_BASE_ +
    (0x0030)) // Interrupt Low Level Timer
347
348 // Reserved                     (_W5200_IO_BASE_
    + (0x0032))
349 // Reserved                     (_W5200_IO_BASE_
    + (0x0033))
350
357 #define IR2                 (_W5200_IO_BASE_ +
    (0x0034)) // Socket Interrupt
358
371 #define PHYSTATUS           (_W5200_IO_BASE_ +
    (0x0035)) // PHY Status
372
389 #define IMR2                    (_W5200_IO_BASE_
    + (0x0036)) // Interrupt Mask
390
391
392 //----------------------------- W5200 Socket
    Registers -----------------------------
393
```

```
394 //--------------------------- For Backward
    Compatibility ---------------------------
395
429 #define Sn_MR(sn)           (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0000)) // socket
    Mode register
430
457 #define Sn_CR(sn)           (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0001)) //
    channel Sn_CR register
458
478 #define Sn_IR(sn)           (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0002)) //
    channel interrupt register
479
501 #define Sn_SR(sn)           (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0003)) //
    channel status register
502
509 #define Sn_PORT(sn)     (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0004)) // source
    port register
510
517 #define Sn_DHAR(sn)     (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0006)) // Peer MAC
    register address
518
527 #define Sn_DIPR(sn)     (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x000C)) // Peer IP
    register address
528
537 #define Sn_DPORT(sn)    (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0010)) // Peer
    port register address
538
544 #define Sn_MSSR(sn)    (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0012)) // Maximum
```

```
    Segment Size(Sn_MSSR0) register address
  545
  552 #define Sn_PROTO(sn)       (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0014)) //
    Protocol of IP Header field register in IP raw
    mode
  553
  560 #define Sn_TOS(sn)
    (WIZCHIP_SREG_BLOCK(sn) + 0x0015) // IP Type
    of Service(TOS) Register
  561
  568 #define Sn_TTL(sn)      (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0016)) // IP Time
    to live(TTL) Register
  569
  570 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0017))
  571 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0018))
  572 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0019))
  573 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001A))
  574 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001B))
  575 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001C))
  576 // Reserved                  (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001D))
  577
  588 #define Sn_RXMEM_SIZE(sn)   (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001E)) //
    Receive memory size reigster
  589
  599 #define Sn_TXMEM_SIZE(sn)   (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x001F)) //
    Transmit memory size reigster
```

```
600
610 #define Sn_TX_FSR(sn)   (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0020)) // Transmit
    free memory size register
611
622 #define Sn_TX_RD(sn)        (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0022)) //
    Transmit memory read pointer register address
623
636 #define Sn_TX_WR(sn)        (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0024)) //
    Transmit memory write pointer register address
637
645 #define Sn_RX_RSR(sn)   (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x0026)) // Received
    data size register
646
658 #define Sn_RX_RD(sn)        (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x0028)) // Read
    point of Receive memory
659
667 #define Sn_RX_WR(sn)        (_W5200_IO_BASE_
    + WIZCHIP_SREG_BLOCK(sn) + (0x002A)) // Write
    point of Receive memory
668
677 #define Sn_IMR(sn)      (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x002C)) // socket
    interrupt mask register
678
684 #define Sn_FRAG(sn)     (_W5200_IO_BASE_ +
    WIZCHIP_SREG_BLOCK(sn) + (0x002D)) // frag
    field value in IP header register
685
686
687 //------------------------------ W5200
    Register values  ------------------------------
688
```

```
689 /* MODE register values */
694 #define MR_RST              0x80
695
696
705 #define MR_WOL              0x20
706
707
713 #define MR_PB               0x10
714
715
721 #define MR_PPPOE            0x08
722
723
730 #define MR_AI               0x02
731
732
738 #define MR_IND              0x01
739
740 /* IR register values */
745 #define IR_CONFLICT         0x80
746
747
751 #define IR_PPPoE            0x20
752
753
757 #define PHYSTATUS_LINK              0x20
758
763 #define PHYSTATUS_POWERSAVE             0x10
764
769 #define PHYSTATUS_POWERDOWN             0x08
770
771 // Sn_MR values
772 /* Sn_MR Default values */
777 #define Sn_MR_CLOSE         0x00
778
779
783 #define Sn_MR_TCP           0x01
```

```
784
785
789 #define Sn_MR_UDP       0x02
790 #define Sn_MR_IPRAW     0x03
791
792
797 #define Sn_MR_MACRAW        0x04
798
799
804 #define Sn_MR_PPPOE         0x05
805
806
814 #define Sn_MR_ND            0x20
815
816 /* Sn_MR Default values */
825 #define Sn_MR_MC            Sn_MR_ND
826
827
834 #define Sn_MR_MF            0x40
835 #define Sn_MR_MFEN      Sn_MR_MF
836
837 /* Sn_MR Default values */
846 #define Sn_MR_MULTI         0x80
847
848 /* Sn_CR values */
863 #define Sn_CR_OPEN          0x01
864
865
874 #define Sn_CR_LISTEN        0x02
875
876
886 #define Sn_CR_CONNECT       0x04
887
888
899 #define Sn_CR_DISCON        0x08
900
901
```

```
905  #define Sn_CR_CLOSE          0x10
906
913  #define Sn_CR_SEND           0x20
914
923  #define Sn_CR_SEND_MAC       0x21
924
931  #define Sn_CR_SEND_KEEP      0x22
932
939  #define Sn_CR_RECV           0x40
940
945  #define Sn_CR_PCON           0x23
946
951  #define Sn_CR_PDISCON        0x24
952
957  #define Sn_CR_PCR            0x25
958
963  #define Sn_CR_PCN            0x26
964
969  #define Sn_CR_PCJ            0x27
970
971  /* Sn_IR values */
976  #define Sn_IR_PRECV          0x80
977
982  #define Sn_IR_PFAIL          0x40
983
988  #define Sn_IR_PNEXT          0x20
989
994  #define Sn_IR_SENDOK         0x10
995
996
1000 #define Sn_IR_TIMEOUT        0x08
1001
1002
1006 #define Sn_IR_RECV           0x04
1007
1012 #define Sn_IR_DISCON         0x02
1013
```

```
1018 #define Sn_IR_CON           0x01
1019
1020 /* Sn_SR values */
1026 #define SOCK_CLOSED         0x00
1027
1028
1034 #define SOCK_INIT           0x13
1035
1036
1042 #define SOCK_LISTEN         0x14
1043
1051 #define SOCK_SYNSENT        0x15
1052
1059 #define SOCK_SYNRECV        0x16
1060
1068 #define SOCK_ESTABLISHED    0x17
1069
1076 #define SOCK_FIN_WAIT       0x18
1077
1084 #define SOCK_CLOSING        0x1A
1085
1092 #define SOCK_TIME_WAIT      0x1B
1093
1100 #define SOCK_CLOSE_WAIT     0x1C
1101
1107 #define SOCK_LAST_ACK       0x1D
1108
1115 #define SOCK_UDP            0x22
1116
1117
1123 #define SOCK_IPRAW          0x32
1124
1125
1131 #define SOCK_MACRAW         0x42
1132
1133
1140 #define SOCK_PPPOE          0x5F
```

```
1141
1142 // IP PROTOCOL
1143 #define IPPROTO_IP          0
1144 #define IPPROTO_ICMP           1
1145 #define IPPROTO_IGMP           2
1146 #define IPPROTO_GGP         3
1147 #define IPPROTO_TCP         6
1148 #define IPPROTO_PUP         12
1149 #define IPPROTO_UDP         17
1150 #define IPPROTO_IDP         22
1151 #define IPPROTO_ND          77
1152 #define IPPROTO_RAW         255
1153
1154
1165 #define WIZCHIP_CRITICAL_ENTER()
     WIZCHIP.CRIS._enter()
1166
1167 #ifdef _exit
1168 #undef _exit
1169 #endif
1170
1182 #define WIZCHIP_CRITICAL_EXIT()
     WIZCHIP.CRIS._exit()
1183
1184
1185
1187 // Basic I/O Function //
1189
1195 uint8_t  WIZCHIP_READ (uint32_t AddrSel);
1196
1204 void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t
     wb );
1205
1213 void WIZCHIP_READ_BUF (uint32_t AddrSel,
     uint8_t* pBuf, uint16_t len);
1214
1222 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
```

```
    uint8_t* pBuf, uint16_t len);
1223
1224
1226 // Common Register IO function //
1228
1235 #if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_)
1236    #define setMR(mr)    WIZCHIP_WRITE(MR,mr)
1237 #else
1238    #define setMR(mr)    (*((uint8_t*)MR) =
    mr)
1239 #endif
1240
1247 #if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_)
1248    #define getMR()      WIZCHIP_READ(MR)
1249 #else
1250    #define getMR()      (*(uint8_t*)MR)
1251 #endif
1252
1259 #define setGAR(gar) \
1260         WIZCHIP_WRITE_BUF(GAR,gar,4)
1261
1268 #define getGAR(gar) \
1269         WIZCHIP_READ_BUF(GAR,gar,4)
1270
1280 #define setSUBR(subr) \
1281    WIZCHIP_WRITE_BUF(SUBR, subr,4)
1282
1289 #define getSUBR(subr) \
1290         WIZCHIP_READ_BUF(SUBR, subr, 4)
1291
1298 #define setSHAR(shar) \
1299         WIZCHIP_WRITE_BUF(SHAR, shar, 6)
1300
1307 #define getSHAR(shar) \
1308         WIZCHIP_READ_BUF(SHAR, shar, 6)
```

```
1309
1316 #define setSIPR(sipr) \
1317         WIZCHIP_WRITE_BUF(SIPR, sipr, 4)
1318
1325 #define getSIPR(sipr) \
1326         WIZCHIP_READ_BUF(SIPR, sipr, 4)
1327
1334 #define setIR(ir) \
1335         WIZCHIP_WRITE(IR, (ir & 0xA0))
1336
1342 #define getIR() \
1343         (WIZCHIP_READ(IR) & 0xA0)
1344
1351 //M20150410 : Replace _IMR_ with IMR2 for
    integrating with ioLibrary
1352 /*
1353 #define setIMR(imr) \
1354         WIZCHIP_WRITE(_IMR_, imr)
1355 */
1356 #define setIMR(imr) \
1357         WIZCHIP_WRITE(IMR2, imr & 0xA0)
1358
1365 //M20150410 : Replace _IMR_ with IMR2 for
    integrating with ioLibrary
1366 /*
1367 #define getIMR() \
1368         WIZCHIP_READ(_IMR_)
1369 */
1370 #define getIMR() \
1371         (WIZCHIP_READ(IMR2) & 0xA0)
1372
1379 #define setRTR(rtr)   {\
1380         WIZCHIP_WRITE(_RTR_,   (uint8_t)(rtr
    >> 8)); \
1381
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),
    (uint8_t) rtr); \
```

```
1382      }
1383
1390  #define getRTR() \
1391          (((uint16_t)WIZCHIP_READ(_RTR_) <<
    8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))
1392
1399  #define setRCR(rcr) \
1400          WIZCHIP_WRITE(_RCR_, rcr)
1401
1408  #define getRCR() \
1409          WIZCHIP_READ(_RCR_)
1410
1416  #define getPATR() \
1417          (((uint16_t)WIZCHIP_READ(PATR) << 8)
    + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR,1)))
1418
1424  #define getPPPALGO() \
1425          WIZCHIP_READ(PPPALGO)
1426
1427
1433  #define getVERSIONR() \
1434          WIZCHIP_READ(VERSIONR)
1435
1442  #define setPTIMER(ptimer) \
1443          WIZCHIP_WRITE(PTIMER, ptimer)
1444
1451  #define getPTIMER() \
1452          WIZCHIP_READ(PTIMER)
1453
1460  #define setPMAGIC(pmagic) \
1461          WIZCHIP_WRITE(PMAGIC, pmagic)
1462
1469  #define getPMAGIC() \
1470          WIZCHIP_READ(PMAGIC)
1471
1478  #define setINTLEVEL(intlevel)  {\
```

```
1479          WIZCHIP_WRITE(INTLEVEL,   (uint8_t) (intlevel >> 8)); \
1480          WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL,1), (uint8_t) intlevel); \
1481      }
1482
1488  #define getINTLEVEL() \
1489          (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL,1)))
1490
1497  #define setIR2(ir2) \
1498          WIZCHIP_WRITE(IR2, ir2)
1499  #define  setSIR(ir2)    setIR2(ir2)
1500
1507  #define getIR2() \
1508      WIZCHIP_READ(IR2)
1509  #define  getSIR()       getIR2()
1510
1516  #define getPHYSTATUS() \
1517          WIZCHIP_READ(PHYSTATUS)
1518
1526   //M20150410 : Replace IMR2 with _IMR_ for integrating with ioLibrary
1527  /*
1528  #define setIMR2(imr2) \
1529          WIZCHIP_WRITE(IMR2, (imr2 & 0xA0))
1530  */
1531  #define setIMR2(imr2) \
1532          WIZCHIP_WRITE(_IMR_, imr2)
1533  #define  setSIMR(imr2)  setIMR2(imr2)
1534
1541   //M20150410 : Replace IMR2 with _IMR_ for integrating with ioLibrary
1542  /*
1543  #define getIMR2() \
```

```
1544            (WIZCHIP_READ(IMR2) & 0xA0)
1545    */
1546    #define getIMR2() \
1547            WIZCHIP_READ(_IMR_)
1548    #define getSIMR()      getIMR2()
1549    // Socket N register I/O function //
1552
1559    #define setSn_MR(sn, mr) \
1560            WIZCHIP_WRITE(Sn_MR(sn),mr)
1561
1569    #define getSn_MR(sn) \
1570        WIZCHIP_READ(Sn_MR(sn))
1571
1579    #define setSn_CR(sn, cr) \
1580            WIZCHIP_WRITE(Sn_CR(sn), cr)
1581
1589    #define getSn_CR(sn) \
1590            WIZCHIP_READ(Sn_CR(sn))
1591
1599    #define setSn_IR(sn, ir) \
1600            WIZCHIP_WRITE(Sn_IR(sn), ir)
1601
1609    #define getSn_IR(sn) \
1610            WIZCHIP_READ(Sn_IR(sn))
1611
1619    #define setSn_IMR(sn, imr) \
1620            WIZCHIP_WRITE(Sn_IMR(sn), imr)
1621
1629    #define getSn_IMR(sn) \
1630            WIZCHIP_READ(Sn_IMR(sn))
1631
1638    #define getSn_SR(sn) \
1639            WIZCHIP_READ(Sn_SR(sn))
1640
1648    #define setSn_PORT(sn, port)  { \
1649            WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \
```

```
1650 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1
    ), (uint8_t) port); \
1651     }
1652 
1660 #define getSn_PORT(sn) \
1661 
    (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    ))
1662 
1670 #define setSn_DHAR(sn, dhar) \
1671         WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar,
    6)
1672 
1680 #define getSn_DHAR(sn, dhar) \
1681         WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar,
    6)
1682 
1690 #define setSn_DIPR(sn, dipr) \
1691         WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
    4)
1692 
1700 #define getSn_DIPR(sn, dipr) \
1701         WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
    4)
1702 
1710 #define setSn_DPORT(sn, dport) { \
1711         WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \
1712 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),
    1), (uint8_t)  dport); \
1713     }
1714 
1722 #define getSn_DPORT(sn) \
1723 
```

```
    (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    )))
1724
1732 #define setSn_MSSR(sn, mss) { \
1733         WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \
1734
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1
    ), (uint8_t) mss); \
1735     }
1736
1744 #define getSn_MSSR(sn) \
1745
    (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    ))
1746
1754 //M20150601 : Fixed Wrong Register address
1755 /*
1756 #define setSn_PROTO(sn, proto) \
1757         WIZCHIP_WRITE(Sn_TOS(sn), tos)
1758 */
1759 #define setSn_PROTO(sn, proto) \
1760         WIZCHIP_WRITE(Sn_PROTO(sn), proto)
1761
1769 //M20150601 : Fixed Wrong Register address
1770 /*
1771 #define getSn_PROTO(sn) \
1772         WIZCHIP_READ(Sn_TOS(sn))
1773 */
1774 #define getSn_PROTO(sn) \
1775         WIZCHIP_READ(Sn_PROTO(sn))
1776
1784 #define setSn_TOS(sn, tos) \
1785         WIZCHIP_WRITE(Sn_TOS(sn), tos)
1786
```

```
1794 #define getSn_TOS(sn) \
1795         WIZCHIP_READ(Sn_TOS(sn))
1796
1804 #define setSn_TTL(sn, ttl) \
1805         WIZCHIP_WRITE(Sn_TTL(sn), ttl)
1806
1814 #define getSn_TTL(sn) \
1815         WIZCHIP_READ(Sn_TTL(sn))
1816
1824 #define setSn_RXMEM_SIZE(sn, rxmemsize) \
1825
    WIZCHIP_WRITE(Sn_RXMEM_SIZE(sn),rxmemsize)
1826
1827 #define  setSn_RXBUF_SIZE(sn,rxmemsize)
    setSn_RXMEM_SIZE(sn,rxmemsize)
1828
1836 #define getSn_RXMEM_SIZE(sn) \
1837         WIZCHIP_READ(Sn_RXMEM_SIZE(sn))
1838
1839 #define  getSn_RXBUF_SIZE(sn)
    getSn_RXMEM_SIZE(sn)
1840
1848 #define setSn_TXMEM_SIZE(sn, txmemsize) \
1849         WIZCHIP_WRITE(Sn_TXMEM_SIZE(sn),
    txmemsize)
1850
1851 #define  setSn_TXBUF_SIZE(sn, txmemsize)
    setSn_TXMEM_SIZE(sn,txmemsize)
1852
1860 #define getSn_TXMEM_SIZE(sn) \
1861         WIZCHIP_READ(Sn_TXMEM_SIZE(sn))
1862
1863 #define  getSn_TXBUF_SIZE(sn)
    getSn_TXMEM_SIZE(sn)
1864
1871 uint16_t getSn_TX_FSR(uint8_t sn);
1872
```

```
1879 #define getSn_TX_RD(sn) \
1880 (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1)))
1881
1889 #define setSn_TX_WR(sn, txwr) { \
1890         WIZCHIP_WRITE(Sn_TX_WR(sn), (uint8_t)(txwr>>8)); \
1891 WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1), (uint8_t) txwr); \
1892         }
1893
1901 #define getSn_TX_WR(sn) \
1902 (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1)))
1903
1910 uint16_t getSn_RX_RSR(uint8_t sn);
1911
1919 #define setSn_RX_RD(sn, rxrd) { \
1920         WIZCHIP_WRITE(Sn_RX_RD(sn), (uint8_t)(rxrd>>8)); \
1921 WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1), (uint8_t) rxrd); \
1922     }
1923
1931 #define getSn_RX_RD(sn) \
1932 (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1)))
1933
1941 #define setSn_RX_WR(sn, rxwr) { \
```

```
1942            WIZCHIP_WRITE(Sn_RX_WR(sn), (uint8_t)(rxwr>>8)); \
1943            WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn), 1), (uint8_t) rxwr); \
1944        }
1945
1946
1953    #define getSn_RX_WR(sn) \
1954            (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1)))
1955
1963    #define setSn_IMR(sn ,imr) \
1964            WIZCHIP_WRITE(Sn_IMR(sn), imr)
1965
1973    #define getSn_IMR(sn) \
1974            WIZCHIP_READ(Sn_IMR(sn))
1975
1983    #define setSn_FRAG(sn, frag) { \
1984            WIZCHIP_WRITE(Sn_FRAG(sn), (uint8_t)(frag >>8)); \
1985            WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1), (uint8_t) frag); \
1986        }
1987
1995    #define getSn_FRAG(sn) \
1996            (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)))
1997
2004    #define getSn_RxMAX(sn) \
2005            ((uint16_t)getSn_RXMEM_SIZE(sn) << 10)
```

```
2006
2013 #define getSn_TxMAX(sn) \
2014         ((uint16_t)getSn_TXMEM_SIZE(sn) <<
     10)
2015
2022 #define getSn_RxMASK(sn) \
2023         ((uint16_t)getSn_RxMAX(sn) - 1)
2024
2031 #define getSn_TxMASK(sn) \
2032         ((uint16_t)getSn_TxMAX(sn) - 1)
2033
2040 uint16_t getSn_RxBASE(uint8_t sn);
2041
2048 uint16_t getSn_TxBASE(uint8_t sn);
2049
2051 // Sn_TXBUF & Sn_RXBUF IO function //
2053
2067 void wiz_send_data(uint8_t sn, uint8_t
     *wizdata, uint16_t len);
2068
2083 void wiz_recv_data(uint8_t sn, uint8_t
     *wizdata, uint16_t len);
2084
2092 void wiz_recv_ignore(uint8_t sn, uint16_t
     len);
2093
2095 #endif
2096
2098 #endif //_W5200_H_
2099
2100
2101
```

Socket APIs

wizchip_conf.c

Go to the documentation of this file.

```
  1 //*****************************************
    ********************************/
 14 //            Issued by Mathias ClauBen.
 49 //
 50 //*****************************************
    ********************************/
 51 //A20140501 : for use the type - ptrdiff_t
 52 #include <stddef.h>
 53 //
 54
 55 #include "wizchip_conf.h"
 56
 58 //M20150401 : Remove ; in the default
    callback function such as
    wizchip_cris_enter(), wizchip_cs_select() and
    etc.
 60
 66 //void    wizchip_cris_enter(void)
    {};
 67 void wizchip_cris_enter(void)           {}
 68
 74 //void    wizchip_cris_exit(void)
    {};
 75 void wizchip_cris_exit(void)          {}
 76
 82 //void  wizchip_cs_select(void)
```

```
    {};
 83 void wizchip_cs_select(void)           {}
 84
 90 //void  wizchip_cs_deselect(void)
    {};
 91 void wizchip_cs_deselect(void)          {}
 92
 98  //M20150601 : Rename the function for
    integrating with W5300
 99 //uint8_t wizchip_bus_readbyte(uint32_t
    AddrSel) { return * ((volatile uint8_t *)
    ((ptrdiff_t) AddrSel)); }
100 iodata_t wizchip_bus_readdata(uint32_t
    AddrSel) { return * ((volatile iodata_t *)
    ((ptrdiff_t) AddrSel)); }
101
107 //M20150601 : Rename the function for
    integrating with W5300
108 //void  wizchip_bus_writebyte(uint32_t
    AddrSel, uint8_t wb)  { *((volatile uint8_t*)
    ((ptrdiff_t)AddrSel)) = wb; }
109 void wizchip_bus_writedata(uint32_t AddrSel,
    iodata_t wb)  { *((volatile iodata_t*)
    ((ptrdiff_t)AddrSel)) = wb; }
110
116 //uint8_t wizchip_spi_readbyte(void)
    {return 0;};
117 uint8_t wizchip_spi_readbyte(void)
    {return 0;}
118
124 //void  wizchip_spi_writebyte(uint8_t wb)
    {};
125 void wizchip_spi_writebyte(uint8_t wb) {}
126
132 //void  wizchip_spi_readburst(uint8_t* pBuf,
    uint16_t len)  {};
133 void wizchip_spi_readburst(uint8_t* pBuf,
```

```
    uint16_t len)  {}
134
140 //void  wizchip_spi_writeburst(uint8_t*
    pBuf, uint16_t len) {};
141 void wizchip_spi_writeburst(uint8_t* pBuf,
    uint16_t len) {}
142
146 //
147 //M20150401 : For a compiler didnot support
    a member of structure
148 //            Replace the assignment of
    struct members with the assingment of array
149 //
150 /*
151 _WIZCHIP  WIZCHIP =
152       {
153       .id                 = _WIZCHIP_ID_,
154       .if_mode            =
    _WIZCHIP_IO_MODE_,
155       .CRIS._enter        =
    wizchip_cris_enter,
156       .CRIS._exit         =
    wizchip_cris_exit,
157       .CS._select         =
    wizchip_cs_select,
158       .CS._deselect       =
    wizchip_cs_deselect,
159       .IF.BUS._read_byte  =
    wizchip_bus_readbyte,
160       .IF.BUS._write_byte =
    wizchip_bus_writebyte
161 //    .IF.SPI._read_byte  =
    wizchip_spi_readbyte,
162 //    .IF.SPI._write_byte =
    wizchip_spi_writebyte
163       };
164 */
```

```
165 _WIZCHIP WIZCHIP =
166       {
167       _WIZCHIP_IO_MODE_,
168       _WIZCHIP_ID_ ,
169       wizchip_cris_enter,
170       wizchip_cris_exit,
171       wizchip_cs_select,
172       wizchip_cs_deselect,
173       //M20150601 : Rename the function
174       //wizchip_bus_readbyte,
175       //wizchip_bus_writebyte
176       wizchip_bus_readdata,
177       wizchip_bus_writedata,
178 //    wizchip_spi_readbyte,
179 //    wizchip_spi_writebyte
180       };
181
182
183 static uint8_t    _DNS_[4];      // DNS
    server ip address
184 static dhcp_mode  _DHCP_;        // DHCP
    mode
185
186 void reg_wizchip_cris_cbfunc(void(*cris_en)
    (void), void(*cris_ex)(void))
187 {
188    if(!cris_en || !cris_ex)
189    {
190       WIZCHIP.CRIS._enter =
    wizchip_cris_enter;
191       WIZCHIP.CRIS._exit  =
    wizchip_cris_exit;
192    }
193    else
194    {
195       WIZCHIP.CRIS._enter = cris_en;
196       WIZCHIP.CRIS._exit  = cris_ex;
```

```
197     }
198 }
199
200 void reg_wizchip_cs_cbfunc(void(*cs_sel)
    (void), void(*cs_desel)(void))
201 {
202     if(!cs_sel || !cs_desel)
203     {
204         WIZCHIP.CS._select   =
    wizchip_cs_select;
205         WIZCHIP.CS._deselect =
    wizchip_cs_deselect;
206     }
207     else
208     {
209         WIZCHIP.CS._select   = cs_sel;
210         WIZCHIP.CS._deselect = cs_desel;
211     }
212 }
213
214 //M20150515 : For integrating with W5300
215 //void
    reg_wizchip_bus_cbfunc(uint8_t(*bus_rb)
    (uint32_t addr), void (*bus_wb)(uint32_t addr,
    uint8_t wb))
216 void
    reg_wizchip_bus_cbfunc(iodata_t(*bus_rb)
    (uint32_t addr), void (*bus_wb)(uint32_t addr,
    iodata_t wb))
217 {
218     while(!(WIZCHIP.if_mode &
    _WIZCHIP_IO_MODE_BUS_));
219     //M20150601 : Rename call back function
    for integrating with W5300
220     /*
221     if(!bus_rb || !bus_wb)
222     {
```

```
223         WIZCHIP.IF.BUS._read_byte   = wizchip_bus_readbyte;
224         WIZCHIP.IF.BUS._write_byte  = wizchip_bus_writebyte;
225     }
226     else
227     {
228         WIZCHIP.IF.BUS._read_byte   = bus_rb;
229         WIZCHIP.IF.BUS._write_byte  = bus_wb;
230     }
231     */
232     if(!bus_rb || !bus_wb)
233     {
234         WIZCHIP.IF.BUS._read_data   = wizchip_bus_readdata;
235         WIZCHIP.IF.BUS._write_data  = wizchip_bus_writedata;
236     }
237     else
238     {
239         WIZCHIP.IF.BUS._read_data   = bus_rb;
240         WIZCHIP.IF.BUS._write_data  = bus_wb;
241     }
242 }
243
244 void reg_wizchip_spi_cbfunc(uint8_t (*spi_rb)(void), void (*spi_wb)(uint8_t wb))
245 {
246     while(!(WIZCHIP.if_mode & _WIZCHIP_IO_MODE_SPI_));
247
248     if(!spi_rb || !spi_wb)
249     {
250         WIZCHIP.IF.SPI._read_byte   = wizchip_spi_readbyte;
251         WIZCHIP.IF.SPI._write_byte  = wizchip_spi_writebyte;
```

```
252     }
253     else
254     {
255        WIZCHIP.IF.SPI._read_byte   = spi_rb;
256        WIZCHIP.IF.SPI._write_byte  = spi_wb;
257     }
258  }
259
260  // 20140626 Eric Added for SPI burst
     operations
261  void reg_wizchip_spiburst_cbfunc(void
     (*spi_rb)(uint8_t* pBuf, uint16_t len), void
     (*spi_wb)(uint8_t* pBuf, uint16_t len))
262  {
263     while(!(WIZCHIP.if_mode &
     _WIZCHIP_IO_MODE_SPI_));
264
265     if(!spi_rb || !spi_wb)
266     {
267        WIZCHIP.IF.SPI._read_burst   =
     wizchip_spi_readburst;
268        WIZCHIP.IF.SPI._write_burst  =
     wizchip_spi_writeburst;
269     }
270     else
271     {
272        WIZCHIP.IF.SPI._read_burst   = spi_rb;
273        WIZCHIP.IF.SPI._write_burst  = spi_wb;
274     }
275  }
276
277  int8_t ctlwizchip(ctlwizchip_type cwtype,
     void* arg)
278  {
279  #if _WIZCHIP_ == 5200 || _WIZCHIP_ == 5500
280     uint8_t tmp = 0;
281  #endif
```

```
282     uint8_t* ptmp[2] = {0,0};
283     switch(cwtype)
284     {
285        case CW_RESET_WIZCHIP:
286           wizchip_sw_reset();
287           break;
288        case CW_INIT_WIZCHIP:
289           if(arg != 0)
290           {
291              ptmp[0] = (uint8_t*)arg;
292              ptmp[1] = ptmp[0] +
    _WIZCHIP_SOCK_NUM_;
293           }
294           return wizchip_init(ptmp[0],
    ptmp[1]);
295        case CW_CLR_INTERRUPT:
296           wizchip_clrinterrupt(*
    ((intr_kind*)arg));
297           break;
298        case CW_GET_INTERRUPT:
299          *((intr_kind*)arg) =
    wizchip_getinterrupt();
300           break;
301        case CW_SET_INTRMASK:
302           wizchip_setinterruptmask(*
    ((intr_kind*)arg));
303           break;
304        case CW_GET_INTRMASK:
305           *((intr_kind*)arg) =
    wizchip_getinterruptmask();
306           break;
307     //M20150601 : This can be supported by
    W5200, W5500
308     //#if _WIZCHIP_ > 5100
309     #if (_WIZCHIP_ == 5200 || _WIZCHIP_ ==
    5500)
310        case CW_SET_INTRTIME:
```

```
311             setINTLEVEL(*(uint16_t*)arg);
312             break;
313          case CW_GET_INTRTIME:
314             *(uint16_t*)arg = getINTLEVEL();
315             break;
316       #endif
317          case CW_GET_ID:
318             ((uint8_t*)arg)[0] = WIZCHIP.id[0];
319             ((uint8_t*)arg)[1] = WIZCHIP.id[1];
320             ((uint8_t*)arg)[2] = WIZCHIP.id[2];
321             ((uint8_t*)arg)[3] = WIZCHIP.id[3];
322             ((uint8_t*)arg)[4] = WIZCHIP.id[4];
323             ((uint8_t*)arg)[5] = 0;
324             break;
325       #if _WIZCHIP_ ==  5500
326          case CW_RESET_PHY:
327             wizphy_reset();
328             break;
329          case CW_SET_PHYCONF:
330
    wizphy_setphyconf((wiz_PhyConf*)arg);
331             break;
332          case CW_GET_PHYCONF:
333
    wizphy_getphyconf((wiz_PhyConf*)arg);
334             break;
335          case CW_GET_PHYSTATUS:
336             break;
337          case CW_SET_PHYPOWMODE:
338             return wizphy_setphypmode(*
    (uint8_t*)arg);
339       #endif
340       #if _WIZCHIP_ == 5200 || _WIZCHIP_ ==
    5500
341          case CW_GET_PHYPOWMODE:
342             tmp = wizphy_getphypmode();
343             if((int8_t)tmp == -1) return -1;
```

```
344             *(uint8_t*)arg = tmp;
345             break;
346          case CW_GET_PHYLINK:
347             tmp = wizphy_getphylink();
348             if((int8_t)tmp == -1) return -1;
349             *(uint8_t*)arg = tmp;
350             break;
351       #endif
352          default:
353             return -1;
354       }
355       return 0;
356 }
357
358
359 int8_t ctlnetwork(ctlnetwork_type cntype,
    void* arg)
360 {
361
362       switch(cntype)
363       {
364          case CN_SET_NETINFO:
365
    wizchip_setnetinfo((wiz_NetInfo*)arg);
366             break;
367          case CN_GET_NETINFO:
368
    wizchip_getnetinfo((wiz_NetInfo*)arg);
369             break;
370          case CN_SET_NETMODE:
371             return wizchip_setnetmode(*
    (netmode_type*)arg);
372          case CN_GET_NETMODE:
373             *(netmode_type*)arg =
    wizchip_getnetmode();
374             break;
375          case CN_SET_TIMEOUT:
```

```
376
    wizchip_settimeout((wiz_NetTimeout*)arg);
377             break;
378         case CN_GET_TIMEOUT:
379
    wizchip_gettimeout((wiz_NetTimeout*)arg);
380             break;
381         default:
382             return -1;
383     }
384     return 0;
385 }
386
387 void wizchip_sw_reset(void)
388 {
389     uint8_t gw[4], sn[4], sip[4];
390     uint8_t mac[6];
391 //A20150601
392 #if _WIZCHIP_IO_MODE_  ==
    _WIZCHIP_IO_MODE_BUS_INDIR_
393     uint16_t mr = (uint16_t)getMR();
394     setMR(mr | MR_IND);
395 #endif
396 //
397     getSHAR(mac);
398     getGAR(gw);  getSUBR(sn);  getSIPR(sip);
399     setMR(MR_RST);
400     getMR(); // for delay
401 //A2015051 : For indirect bus mode
402 #if _WIZCHIP_IO_MODE_  ==
    _WIZCHIP_IO_MODE_BUS_INDIR_
403     setMR(mr | MR_IND);
404 #endif
405 //
406     setSHAR(mac);
407     setGAR(gw);
408     setSUBR(sn);
```

```
409     setSIPR(sip);
410 }
411
412 int8_t wizchip_init(uint8_t* txsize,
    uint8_t* rxsize)
413 {
414     int8_t i;
415     int8_t tmp = 0;
416     wizchip_sw_reset();
417     if(txsize)
418     {
419         tmp = 0;
420     //M20150601 : For integrating with W5300
421     #if _WIZCHIP_ == 5300
422         for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
    i++)
423         {
424             if(txsize[i] >= 64) return -1;
    //No use 64KB even if W5300 support max 64KB
    memory allocation
425             tmp += txsize[i];
426             if(tmp > 128) return -1;
427         }
428         if(tmp % 8) return -1;
429     #else
430         for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
    i++)
431         {
432             tmp += txsize[i];
433             if(tmp > 16) return -1;
434         }
435     #endif
436         for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
    i++)
437             setSn_TXBUF_SIZE(i, txsize[i]);
438     }
439     if(rxsize)
```

```
440     {
441        tmp = 0;
442     #if _WIZCHIP_ == 5300
443        for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
   i++)
444        {
445           if(rxsize[i] >= 64) return -1;
   //No use 64KB even if W5300 support max 64KB
   memory allocation
446           tmp += rxsize[i];
447           if(tmp > 128) return -1;
448        }
449        if(tmp % 8) return -1;
450     #else
451        for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
   i++)
452        {
453           tmp += rxsize[i];
454           if(tmp > 16) return -1;
455        }
456     #endif
457
458        for(i = 0 ; i < _WIZCHIP_SOCK_NUM_;
   i++)
459           setSn_RXBUF_SIZE(i, rxsize[i]);
460     }
461     return 0;
462  }
463
464  void wizchip_clrinterrupt(intr_kind intr)
465  {
466     uint8_t ir  = (uint8_t)intr;
467     uint8_t sir = (uint8_t)((uint16_t)intr >>
   8);
468  #if _WIZCHIP_ < 5500
469     ir |= (1<<4); // IK_WOL
470  #endif
```

```
471 #if _WIZCHIP_ == 5200
472    ir |= (1 << 6);
473 #endif
474
475 #if _WIZCHIP_ < 5200
476    sir &= 0x0F;
477 #endif
478
479 #if _WIZCHIP_ == 5100
480    ir |= sir;
481    setIR(ir);
482 //A20150601 : For integrating with W5300
483 #elif _WIZCHIP_ == 5300
484    setIR( ((((uint16_t)ir) << 8) |
     (((uint16_t)sir) & 0x00FF)) );
485 #else
486    setIR(ir);
487    setSIR(sir);
488 #endif
489 }
490
491 intr_kind wizchip_getinterrupt(void)
492 {
493    uint8_t ir  = 0;
494    uint8_t sir = 0;
495    uint16_t ret = 0;
496 #if _WIZCHIP_ == 5100
497    ir = getIR();
498    sir = ir & 0x0F;
499 //A20150601 : For integrating with W5300
500 #elif _WIZCHIP_  == 5300
501    ret = getIR();
502    ir = (uint8_t)(ret >> 8);
503    sir = (uint8_t)ret;
504 #else
505    ir  = getIR();
506    sir = getSIR();
```

```
507 #endif
508
509 //M20150601 : For Integrating with W5300
510 //#if _WIZCHIP_ < 5500
511 #if _WIZCHIP_ < 5200
512    ir &= ~(1<<4); // IK_WOL
513 #endif
514 #if _WIZCHIP_ == 5200
515    ir &= ~(1 << 6);
516 #endif
517   ret = sir;
518   ret = (ret << 8) + ir;
519   return (intr_kind)ret;
520 }
521
522 void wizchip_setinterruptmask(intr_kind
    intr)
523 {
524    uint8_t imr  = (uint8_t)intr;
525    uint8_t simr = (uint8_t)((uint16_t)intr
    >> 8);
526 #if _WIZCHIP_ < 5500
527    imr &= ~(1<<4); // IK_WOL
528 #endif
529 #if _WIZCHIP_ == 5200
530    imr &= ~(1 << 6);
531 #endif
532
533 #if _WIZCHIP_ == 5100
534    simr &= 0x0F;
535    imr |= simr;
536    setIMR(imr);
537 //A20150601 : For integrating with W5300
538 #elif _WIZCHIP_ == 5300
539    setIMR( ((((uint16_t)imr) << 8) |
    (((uint16_t)simr) & 0x00FF)) );
540 #else
```

```
541     setIMR(imr);
542     setSIMR(simr);
543 #endif
544 }
545
546 intr_kind wizchip_getinterruptmask(void)
547 {
548     uint8_t imr  = 0;
549     uint8_t simr = 0;
550     uint16_t ret = 0;
551 #if _WIZCHIP_ == 5100
552     imr  = getIMR();
553     simr = imr & 0x0F;
554 //A20150601 : For integrating with W5300
555 #elif _WIZCHIP_ == 5300
556     ret = getIMR();
557     imr = (uint8_t)(ret >> 8);
558     simr = (uint8_t)ret;
559 #else
560     imr  = getIMR();
561     simr = getSIMR();
562 #endif
563
564 #if _WIZCHIP_ < 5500
565     imr &= ~(1<<4); // IK_WOL
566 #endif
567 #if _WIZCHIP_ == 5200
568     imr &= ~(1 << 6);  // IK_DEST_UNREACH
569 #endif
570   ret = simr;
571   ret = (ret << 8) + imr;
572   return (intr_kind)ret;
573 }
574
575 int8_t wizphy_getphylink(void)
576 {
577     int8_t tmp;
```

```
578 #if   _WIZCHIP_ == 5200
579    if(getPHYSTATUS() & PHYSTATUS_LINK)
580       tmp = PHY_LINK_ON;
581    else
582       tmp = PHY_LINK_OFF;
583 #elif _WIZCHIP_ == 5500
584    if(getPHYCFGR() & PHYCFGR_LNK_ON)
585       tmp = PHY_LINK_ON;
586    else
587       tmp = PHY_LINK_OFF;
588 #else
589    tmp = -1;
590 #endif
591    return tmp;
592 }
593
594 #if _WIZCHIP_ > 5100
595
596 int8_t wizphy_getphypmode(void)
597 {
598    int8_t tmp = 0;
599    #if   _WIZCHIP_ == 5200
600       if(getPHYSTATUS() &
   PHYSTATUS_POWERDOWN)
601          tmp = PHY_POWER_DOWN;
602       else
603          tmp = PHY_POWER_NORM;
604    #elif _WIZCHIP_ == 5500
605       if(getPHYCFGR() & PHYCFGR_OPMDC_PDOWN)
606          tmp = PHY_POWER_DOWN;
607       else
608          tmp = PHY_POWER_NORM;
609    #else
610       tmp = -1;
611    #endif
612    return tmp;
613 }
```

```
614 #endif
615
616 #if _WIZCHIP_ == 5500
617 void wizphy_reset(void)
618 {
619    uint8_t tmp = getPHYCFGR();
620    tmp &= PHYCFGR_RST;
621    setPHYCFGR(tmp);
622    tmp = getPHYCFGR();
623    tmp |= ~PHYCFGR_RST;
624    setPHYCFGR(tmp);
625 }
626
627 void wizphy_setphyconf(wiz_PhyConf* phyconf)
628 {
629    uint8_t tmp = 0;
630    if(phyconf->by == PHY_CONFBY_SW)
631       tmp |= PHYCFGR_OPMD;
632    else
633       tmp &= ~PHYCFGR_OPMD;
634    if(phyconf->mode == PHY_MODE_AUTONEGO)
635       tmp |= PHYCFGR_OPMDC_ALLA;
636    else
637    {
638       if(phyconf->duplex == PHY_DUPLEX_FULL)
639       {
640          if(phyconf->speed == PHY_SPEED_100)
641             tmp |= PHYCFGR_OPMDC_100F;
642          else
643             tmp |= PHYCFGR_OPMDC_10F;
644       }
645       else
646       {
647          if(phyconf->speed == PHY_SPEED_100)
648             tmp |= PHYCFGR_OPMDC_100H;
649          else
650             tmp |= PHYCFGR_OPMDC_10H;
```

```
651         }
652      }
653      setPHYCFGR(tmp);
654      wizphy_reset();
655   }
656
657   void wizphy_getphyconf(wiz_PhyConf* phyconf)
658   {
659      uint8_t tmp = 0;
660      tmp = getPHYCFGR();
661      phyconf->by   = (tmp & PHYCFGR_OPMD) ?
     PHY_CONFBY_SW : PHY_CONFBY_HW;
662      switch(tmp & PHYCFGR_OPMDC_ALLA)
663      {
664         case PHYCFGR_OPMDC_ALLA:
665         case PHYCFGR_OPMDC_100FA:
666            phyconf->mode = PHY_MODE_AUTONEGO;
667            break;
668         default:
669            phyconf->mode = PHY_MODE_MANUAL;
670            break;
671      }
672      switch(tmp & PHYCFGR_OPMDC_ALLA)
673      {
674         case PHYCFGR_OPMDC_100FA:
675         case PHYCFGR_OPMDC_100F:
676         case PHYCFGR_OPMDC_100H:
677            phyconf->speed = PHY_SPEED_100;
678            break;
679         default:
680            phyconf->speed = PHY_SPEED_10;
681            break;
682      }
683      switch(tmp & PHYCFGR_OPMDC_ALLA)
684      {
685         case PHYCFGR_OPMDC_100FA:
686         case PHYCFGR_OPMDC_100F:
```

```
687         case PHYCFGR_OPMDC_10F:
688            phyconf->duplex = PHY_DUPLEX_FULL;
689            break;
690         default:
691            phyconf->duplex = PHY_DUPLEX_HALF;
692            break;
693      }
694   }
695
696   void wizphy_getphystat(wiz_PhyConf* phyconf)
697   {
698      uint8_t tmp = getPHYCFGR();
699      phyconf->duplex = (tmp &
      PHYCFGR_DPX_FULL) ? PHY_DUPLEX_FULL :
      PHY_DUPLEX_HALF;
700      phyconf->speed  = (tmp & PHYCFGR_SPD_100)
      ? PHY_SPEED_100 : PHY_SPEED_10;
701   }
702
703   int8_t wizphy_setphypmode(uint8_t pmode)
704   {
705      uint8_t tmp = 0;
706      tmp = getPHYCFGR();
707      if((tmp & PHYCFGR_OPMD)== 0) return -1;
708      tmp &= ~PHYCFGR_OPMDC_ALLA;
709      if( pmode == PHY_POWER_DOWN)
710         tmp |= PHYCFGR_OPMDC_PDOWN;
711      else
712         tmp |= PHYCFGR_OPMDC_ALLA;
713      setPHYCFGR(tmp);
714      wizphy_reset();
715      tmp = getPHYCFGR();
716      if( pmode == PHY_POWER_DOWN)
717      {
718         if(tmp & PHYCFGR_OPMDC_PDOWN) return
      0;
719      }
```

```
720     else
721     {
722        if(tmp & PHYCFGR_OPMDC_ALLA) return 0;
723     }
724     return -1;
725  }
726  #endif
727
728
729  void wizchip_setnetinfo(wiz_NetInfo*
     pnetinfo)
730  {
731     setSHAR(pnetinfo->mac);
732     setGAR(pnetinfo->gw);
733     setSUBR(pnetinfo->sn);
734     setSIPR(pnetinfo->ip);
735     _DNS_[0] = pnetinfo->dns[0];
736     _DNS_[1] = pnetinfo->dns[1];
737     _DNS_[2] = pnetinfo->dns[2];
738     _DNS_[3] = pnetinfo->dns[3];
739     _DHCP_   = pnetinfo->dhcp;
740  }
741
742  void wizchip_getnetinfo(wiz_NetInfo*
     pnetinfo)
743  {
744     getSHAR(pnetinfo->mac);
745     getGAR(pnetinfo->gw);
746     getSUBR(pnetinfo->sn);
747     getSIPR(pnetinfo->ip);
748     pnetinfo->dns[0]= _DNS_[0];
749     pnetinfo->dns[1]= _DNS_[1];
750     pnetinfo->dns[2]= _DNS_[2];
751     pnetinfo->dns[3]= _DNS_[3];
752     pnetinfo->dhcp  = _DHCP_;
753  }
754
```

```
755 int8_t wizchip_setnetmode(netmode_type
    netmode)
756 {
757     uint8_t tmp = 0;
758 #if _WIZCHIP_ != 5500
759     if(netmode & ~(NM_WAKEONLAN | NM_PPPOE |
    NM_PINGBLOCK)) return -1;
760 #else
761     if(netmode & ~(NM_WAKEONLAN | NM_PPPOE |
    NM_PINGBLOCK | NM_FORCEARP)) return -1;
762 #endif
763     tmp = getMR();
764     tmp |= (uint8_t)netmode;
765     setMR(tmp);
766     return 0;
767 }
768
769 netmode_type wizchip_getnetmode(void)
770 {
771     return (netmode_type) getMR();
772 }
773
774 void wizchip_settimeout(wiz_NetTimeout*
    nettime)
775 {
776     setRCR(nettime->retry_cnt);
777     setRTR(nettime->time_100us);
778 }
779
780 void wizchip_gettimeout(wiz_NetTimeout*
    nettime)
781 {
782     nettime->retry_cnt = getRCR();
783     nettime->time_100us = getRTR();
784 }
```

Socket APIs

wizchip_conf.h

Go to the documentation of this file.

```
  1 //*****************************************************************************
  2 //
 43 //
 44 //*****************************************************************************
 45
 54 #ifndef  _WIZCHIP_CONF_H_
 55 #define  _WIZCHIP_CONF_H_
 56
 57 #include <stdint.h>
 63 #ifndef _WIZCHIP_
 64 #define _WIZCHIP_                         5500
// 5100, 5200, 5300, 5500
 65 #endif
 66
 67 #define _WIZCHIP_IO_MODE_NONE_
0x0000
 68 #define _WIZCHIP_IO_MODE_BUS_
0x0100
 69 #define _WIZCHIP_IO_MODE_SPI_
0x0200
 70 //#define _WIZCHIP_IO_MODE_IIC_
0x0400
 71 //#define _WIZCHIP_IO_MODE_SDIO_
0x0800
```

```
 72| // Add to
 73| //
 74|
 75| #define _WIZCHIP_IO_MODE_BUS_DIR_
     (_WIZCHIP_IO_MODE_BUS_ + 1)
 76| #define _WIZCHIP_IO_MODE_BUS_INDIR_
     (_WIZCHIP_IO_MODE_BUS_ + 2)
 78| #define _WIZCHIP_IO_MODE_SPI_VDM_
     (_WIZCHIP_IO_MODE_SPI_ + 1)
 79| #define _WIZCHIP_IO_MODE_SPI_FDM_
     (_WIZCHIP_IO_MODE_SPI_ + 2)
 82| #if   (_WIZCHIP_ == 5100)
 83|    #define _WIZCHIP_ID_
     "W5100\0"
 84|
 88| // #define _WIZCHIP_IO_MODE_
     _WIZCHIP_IO_MODE_BUS_DIR_
 89| // #define _WIZCHIP_IO_MODE_
     _WIZCHIP_IO_MODE_BUS_INDIR_
 90|    #define _WIZCHIP_IO_MODE_
     _WIZCHIP_IO_MODE_SPI_
 91|
 92| //A20150601 : Define the unit of IO DATA.
 93|    typedef   uint8_t   iodata_t;
 94| //A20150401 : Indclude W5100.h file
 95|    #include "W5100/w5100.h"
 96|
 97| #elif (_WIZCHIP_ == 5200)
 98|    #define _WIZCHIP_ID_
     "W5200\0"
 99|
103| #ifndef _WIZCHIP_IO_MODE_
104| // #define _WIZCHIP_IO_MODE_
     _WIZCHIP_IO_MODE_BUS_INDIR_
105|    #define _WIZCHIP_IO_MODE_
     _WIZCHIP_IO_MODE_SPI_
106| #endif
```

```
107 //A20150601 : Define the unit of IO DATA.
108     typedef   uint8_t   iodata_t;
109     #include "W5200/w5200.h"
110 #elif (_WIZCHIP_ == 5500)
111   #define _WIZCHIP_ID_
    "W5500\0"
112
126 #ifndef _WIZCHIP_IO_MODE_
127     //#define _WIZCHIP_IO_MODE_
    _WIZCHIP_IO_MODE_SPI_FDM_
128     #define _WIZCHIP_IO_MODE_
    _WIZCHIP_IO_MODE_SPI_VDM_
129 #endif
130 //A20150601 : Define the unit of IO DATA.
131     typedef   uint8_t   iodata_t;
132     #include "W5500/w5500.h"
133 #elif ( _WIZCHIP_ == 5300)
134     #define _WIZCHIP_ID_
    "W5300\0"
135
139 #ifndef _WIZCHIP_IO_MODE_
140 //   #define _WIZCHIP_IO_MODE_
    _WIZCHIP_IO_MODE_BUS_DIR_
141  #define _WIZCHIP_IO_MODE_
    _WIZCHIP_IO_MODE_BUS_INDIR_
142 #endif
143
144 //A20150601 : Define the unit and bus width
    of IO DATA.
149     #ifndef _WIZCHIP_IO_BUS_WIDTH_
150     #define _WIZCHIP_IO_BUS_WIDTH_      8
    // 16
151     #endif
152     #if _WIZCHIP_IO_BUS_WIDTH_ == 8
153        typedef   uint8_t   iodata_t;
154     #elif _WIZCHIP_IO_BUS_WIDTH_ == 16
155        typedef   uint16_t   iodata_t;
```

```
156     #else
157        #error "Unknown _WIZCHIP_IO_BUS_WIDTH_. It should be 8 or 16."
158     #endif
159  //
160     #include "W5300/w5300.h"
161  #else
162     #error "Unknown defined _WIZCHIP_. You should define one of 5100, 5200, and 5500 !!!"
163  #endif
164
165  #ifndef _WIZCHIP_IO_MODE_
166     #error "Undefined _WIZCHIP_IO_MODE_. You should define it !!!"
167  #endif
168
175  #ifndef _WIZCHIP_IO_BASE_
176  #define _WIZCHIP_IO_BASE_ 0x00000000  // 0x8000
177  #endif
178
179  //M20150401 : Typing Error
180  //#if _WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_BUS
181  #if _WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_BUS_
182     #ifndef _WIZCHIP_IO_BASE_
183        #error "You should be define _WIZCHIP_IO_BASE to fit your system memory map."
184     #endif
185  #endif
186
187  #if _WIZCHIP_ > 5100
188     #define _WIZCHIP_SOCK_NUM_   8
189  #else
190     #define _WIZCHIP_SOCK_NUM_   4
```

```
191 #endif
192
193
194 /**********************************************
    *************
195 * WIZCHIP BASIC IF functions for SPI, SDIO,
    I2C , ETC.
196 **********************************************
    *************/
201 typedef struct __WIZCHIP
202 {
203    uint16_t  if_mode;
204    uint8_t   id[6];
205
208    struct _CRIS
209    {
210       void (*_enter)  (void);
211       void (*_exit) (void);
212    }CRIS;
216    struct _CS
217    {
218       void (*_select)  (void);
219       void (*_deselect)(void);
220    }CS;
224    union _IF
225    {
229       //M20156501 : Modify the function name
    for integrating with W5300
230       //struct
231       //{
232       //   uint8_t  (*_read_byte)  (uint32_t
    AddrSel);
233       //   void     (*_write_byte) (uint32_t
    AddrSel, uint8_t wb);
234       //}BUS;
235       struct
236       {
```

```
237           iodata_t  (*_read_data)   (uint32_t
    AddrSel);
238           void      (*_write_data)  (uint32_t
    AddrSel, iodata_t wb);
239       }BUS;
240
244       struct
245       {
246           uint8_t (*_read_byte)   (void);
247           void    (*_write_byte)  (uint8_t
    wb);
248           void    (*_read_burst)  (uint8_t*
    pBuf, uint16_t len);
249           void    (*_write_burst) (uint8_t*
    pBuf, uint16_t len);
250       }SPI;
251       // To be added
252       //
253   }IF;
254 }_WIZCHIP;
255
256 extern _WIZCHIP WIZCHIP;
257
262 typedef enum
263 {
264    CW_RESET_WIZCHIP,
265    CW_INIT_WIZCHIP,
266    CW_GET_INTERRUPT,
267    CW_CLR_INTERRUPT,
268    CW_SET_INTRMASK,
269    CW_GET_INTRMASK,
270    CW_SET_INTRTIME,
271    CW_GET_INTRTIME,
272    CW_GET_ID,
273
274 //D20150601 : For no modification your
    application code
```

```
275 //#if _WIZCHIP_ ==  5500
276    CW_RESET_PHY,
277    CW_SET_PHYCONF,
278    CW_GET_PHYCONF,
279    CW_GET_PHYSTATUS,
280    CW_SET_PHYPOWMODE,
281 //#endif
282 //D20150601 : For no modification your
      application code
283 //#if _WIZCHIP_ == 5200 || _WIZCHIP_ == 5500
284    CW_GET_PHYPOWMODE,
285    CW_GET_PHYLINK
286 //#endif
287 }ctlwizchip_type;
288
293 typedef enum
294 {
295    CN_SET_NETINFO,
296    CN_GET_NETINFO,
297    CN_SET_NETMODE,
298    CN_GET_NETMODE,
299    CN_SET_TIMEOUT,
300    CN_GET_TIMEOUT,
301 }ctlnetwork_type;
302
309 typedef enum
310 {
311 #if   _WIZCHIP_ == 5500
312    IK_WOL                = (1 << 4),
313 #elif _WIZCHIP_ == 5300
314    IK_FMTU               = (1 << 4),
315 #endif
316
317    IK_PPPOE_TERMINATED   = (1 << 5),
318
319 #if _WIZCHIP_ != 5200
320    IK_DEST_UNREACH       = (1 << 6),
```

```
321 #endif
322
323     IK_IP_CONFLICT        = (1 << 7),
324
325     IK_SOCK_0             = (1 << 8),
326     IK_SOCK_1             = (1 << 9),
327     IK_SOCK_2             = (1 << 10),
328     IK_SOCK_3             = (1 << 11),
329 #if _WIZCHIP_ > 5100
330     IK_SOCK_4             = (1 << 12),
331     IK_SOCK_5             = (1 << 13),
332     IK_SOCK_6             = (1 << 14),
333     IK_SOCK_7             = (1 << 15),
334 #endif
335
336 #if _WIZCHIP_ > 5100
337     IK_SOCK_ALL           = (0xFF << 8)
338 #else
339     IK_SOCK_ALL           = (0x0F << 8)
340 #endif
341 }intr_kind;
342
343 #define PHY_CONFBY_HW             0
344 #define PHY_CONFBY_SW             1
345 #define PHY_MODE_MANUAL           0
346 #define PHY_MODE_AUTONEGO         1
347 #define PHY_SPEED_10              0
348 #define PHY_SPEED_100             1
349 #define PHY_DUPLEX_HALF           0
350 #define PHY_DUPLEX_FULL           1
351 #define PHY_LINK_OFF              0
352 #define PHY_LINK_ON               1
353 #define PHY_POWER_NORM            0
354 #define PHY_POWER_DOWN            1
355
356
357 #if _WIZCHIP_ == 5500
```

```
358
364 typedef struct wiz_PhyConf_t
365 {
366       uint8_t by;
367       uint8_t mode;
368       uint8_t speed;
369       uint8_t duplex;
370       //uint8_t power;  ///< set by @ref
    PHY_POWER_NORM or @ref PHY_POWER_DOWN
371       //uint8_t link;   ///< Valid only in
    CW_GET_PHYSTATUS. set by @ref PHY_LINK_ON or
    PHY_DUPLEX_OFF
372    }wiz_PhyConf;
373 #endif
374
379 typedef enum
380 {
381    NETINFO_STATIC = 1,
382    NETINFO_DHCP
383 }dhcp_mode;
384
389 typedef struct wiz_NetInfo_t
390 {
391    uint8_t mac[6];
392    uint8_t ip[4];
393    uint8_t sn[4];
394    uint8_t gw[4];
395    uint8_t dns[4];
396    dhcp_mode dhcp;
397 }wiz_NetInfo;
398
403 typedef enum
404 {
405 #if _WIZCHIP_ == 5500
406    NM_FORCEARP    = (1<<1),
407 #endif
408    NM_WAKEONLAN   = (1<<5),
```

```
409     NM_PINGBLOCK   = (1<<4),
410     NM_PPPOE       = (1<<3),
411  }netmode_type;
412
417  typedef struct wiz_NetTimeout_t
418  {
419     uint8_t  retry_cnt;
420     uint16_t time_100us;
421  }wiz_NetTimeout;
422
431  void reg_wizchip_cris_cbfunc(void(*cris_en)
     (void), void(*cris_ex)(void));
432
433
441  void reg_wizchip_cs_cbfunc(void(*cs_sel)
     (void), void(*cs_desel)(void));
442
451  //M20150601 : For integrating with W5300
452  //void reg_wizchip_bus_cbfunc(uint8_t
     (*bus_rb)(uint32_t addr), void (*bus_wb)
     (uint32_t addr, uint8_t wb));
453  void reg_wizchip_bus_cbfunc(iodata_t
     (*bus_rb)(uint32_t addr), void (*bus_wb)
     (uint32_t addr, iodata_t wb));
454
463  void reg_wizchip_spi_cbfunc(uint8_t
     (*spi_rb)(void), void (*spi_wb)(uint8_t wb));
464
473  void reg_wizchip_spiburst_cbfunc(void
     (*spi_rb)(uint8_t* pBuf, uint16_t len), void
     (*spi_wb)(uint8_t* pBuf, uint16_t len));
474
485  int8_t ctlwizchip(ctlwizchip_type cwtype,
     void* arg);
486
496  int8_t ctlnetwork(ctlnetwork_type cntype,
     void* arg);
```

```
497
498
499  /*
500   * The following functions are implemented for internal use.
501   * but You can call these functions for code size reduction instead of ctlwizchip() and ctlnetwork().
502   */
503
508  void wizchip_sw_reset(void);
509
518  int8_t wizchip_init(uint8_t* txsize, uint8_t* rxsize);
519
525  void wizchip_clrinterrupt(intr_kind intr);
526
532  intr_kind wizchip_getinterrupt(void);
533
539  void wizchip_setinterruptmask(intr_kind intr);
540
546  intr_kind wizchip_getinterruptmask(void);
547
548  #if _WIZCHIP_ > 5100
549     int8_t wizphy_getphylink(void);
550     int8_t wizphy_getphypmode(void);
551  #endif
552
553  #if _WIZCHIP_ == 5500
554     void wizphy_reset(void);
555
560     void wizphy_setphyconf(wiz_PhyConf* phyconf);
566     void wizphy_getphyconf(wiz_PhyConf* phyconf);
572     void wizphy_getphystat(wiz_PhyConf*
```

```
    phyconf);
578     int8_t wizphy_setphypmode(uint8_t pmode);
579 #endif
580
586 void wizchip_setnetinfo(wiz_NetInfo*
    pnetinfo);
587
593 void wizchip_getnetinfo(wiz_NetInfo*
    pnetinfo);
594
600 int8_t wizchip_setnetmode(netmode_type
    netmode);
601
607 netmode_type wizchip_getnetmode(void);
608
615 void wizchip_settimeout(wiz_NetTimeout*
    nettime);
616
623 void wizchip_gettimeout(wiz_NetTimeout*
    nettime);
624
625 #endif   // _WIZCHIP_CONF_H_
```

Socket APIs

w5500.h

Go to the documentation of this file.

```
  1 //*****************************************************************************
  2 //
 43 //
 44 //*****************************************************************************
 45
 46 //
 47
 48 #ifndef  _W5500_H_
 49 #define  _W5500_H_
 50
 51 #include <stdint.h>
 52 #include "wizchip_conf.h"
 53
 55 #if   (_WIZCHIP_ == 5500)
 56
 58 #define _W5500_IO_BASE_ 0x00000000
 59
 60 #define _W5500_SPI_READ_ (0x00 << 2) //< SPI interface Read operation in Control Phase
 61 #define _W5500_SPI_WRITE_ (0x01 << 2) //< SPI interface Write operation in Control Phase
```

```
 62|
 63| #define WIZCHIP_CREG_BLOCK          0x00
    //< Common register block
 64| #define WIZCHIP_SREG_BLOCK(N)       (1+4*N)
    //< Socket N register block
 65| #define WIZCHIP_TXBUF_BLOCK(N)      (2+4*N)
    //< Socket N Tx buffer address block
 66| #define WIZCHIP_RXBUF_BLOCK(N)      (3+4*N)
    //< Socket N Rx buffer address block
 67|
 68| #define WIZCHIP_OFFSET_INC(ADDR, N)    (ADDR
    + (N<<8)) //< Increase offset address
 69|
 70|
 72| // Definition For Legacy Chip Driver //
 74| #define IINCHIP_READ(ADDR)
    WIZCHIP_READ(ADDR)
 75| #define IINCHIP_WRITE(ADDR,VAL)
    WIZCHIP_WRITE(ADDR,VAL)
 76| #define IINCHIP_READ_BUF(ADDR,BUF,LEN)
    WIZCHIP_READ_BUF(ADDR,BUF,LEN)
 77| #define IINCHIP_WRITE_BUF(ADDR,BUF,LEN)
    WIZCHIP_WRITE(ADDR,BUF,LEN)
 78|
 79| //------------------------  defgroup -----
    ---------------------------
197| //------------------------------ defgroup
    end ----------------------------------------
    --
198| //------------------------------ W5500 Common
    Registers IOMAP ------------------------------
214| #define MR                  (_W5500_IO_BASE_
    + (0x0000 << 8) + (WIZCHIP_CREG_BLOCK << 3))
215|
221| #define GAR                 (_W5500_IO_BASE_
    + (0x0001 << 8) + (WIZCHIP_CREG_BLOCK << 3))
222|
```

```
228 #define SUBR                (_W5500_IO_BASE_
    + (0x0005 << 8) + (WIZCHIP_CREG_BLOCK << 3))
229
235 #define SHAR                (_W5500_IO_BASE_
    + (0x0009 << 8) + (WIZCHIP_CREG_BLOCK << 3))
236
242 #define SIPR                (_W5500_IO_BASE_
    + (0x000F << 8) + (WIZCHIP_CREG_BLOCK << 3))
243
249 #define INTLEVEL            (_W5500_IO_BASE_
    + (0x0013 << 8) + (WIZCHIP_CREG_BLOCK << 3))
250
266 #define IR                  (_W5500_IO_BASE_
    + (0x0015 << 8) + (WIZCHIP_CREG_BLOCK << 3))
267
284 //M20150401 : Rename SYMBOE ( Re-define
    error in a compile)
285 //#define IMR
    (_W5500_IO_BASE_ + (0x0016 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
286 #define _IMR_
    (_W5500_IO_BASE_ + (0x0016 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
287
294 #define SIR                 (_W5500_IO_BASE_
    + (0x0017 << 8) + (WIZCHIP_CREG_BLOCK << 3))
295
303 #define SIMR                (_W5500_IO_BASE_
    + (0x0018 << 8) + (WIZCHIP_CREG_BLOCK << 3))
304
313 //M20150401 : Rename SYMBOE ( Re-define
    error in a compile)
314 //#define RTR
    (_W5500_IO_BASE_ + (0x0019 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
315 #define _RTR_
    (_W5500_IO_BASE_ + (0x0019 << 8) +
```

```
    (WIZCHIP_CREG_BLOCK << 3))
316
323 //M20150401 : Rename SYMBOE ( Re-define
    error in a compile)
324 //#define RCR
    (_W5500_IO_BASE_ + (0x001B << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
325 #define _RCR_
    (_W5500_IO_BASE_ + (0x001B << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
326
332 #define PTIMER              (_W5500_IO_BASE_
    + (0x001C << 8) + (WIZCHIP_CREG_BLOCK << 3))
333
339 #define PMAGIC              (_W5500_IO_BASE_
    + (0x001D << 8) + (WIZCHIP_CREG_BLOCK << 3))
340
346 #define PHAR                 (_W5500_IO_BASE_
    + (0x001E << 8) + (WIZCHIP_CREG_BLOCK << 3))
347
353 #define PSID                (_W5500_IO_BASE_
    + (0x0024 << 8) + (WIZCHIP_CREG_BLOCK << 3))
354
360 #define PMRU                (_W5500_IO_BASE_
    + (0x0026 << 8) + (WIZCHIP_CREG_BLOCK << 3))
361
369 #define UIPR                (_W5500_IO_BASE_
    + (0x0028 << 8) + (WIZCHIP_CREG_BLOCK << 3))
370
378 #define UPORTR               (_W5500_IO_BASE_
    + (0x002C << 8) + (WIZCHIP_CREG_BLOCK << 3))
379
385 #define PHYCFGR             (_W5500_IO_BASE_
    + (0x002E << 8) + (WIZCHIP_CREG_BLOCK << 3))
386
387 // Reserved
    (_W5500_IO_BASE_ + (0x002F << 8) +
```

```
    (WIZCHIP_CREG_BLOCK << 3))
388 // Reserved
    (_W5500_IO_BASE_ + (0x0030 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
389 // Reserved
    (_W5500_IO_BASE_ + (0x0031 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
390 // Reserved
    (_W5500_IO_BASE_ + (0x0032 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
391 // Reserved
    (_W5500_IO_BASE_ + (0x0033 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
392 // Reserved
    (_W5500_IO_BASE_ + (0x0034 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
393 // Reserved
    (_W5500_IO_BASE_ + (0x0035 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
394 // Reserved
    (_W5500_IO_BASE_ + (0x0036 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
395 // Reserved
    (_W5500_IO_BASE_ + (0x0037 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
396 // Reserved
    (_W5500_IO_BASE_ + (0x0038 << 8) +
    (WIZCHIP_CREG_BLOCK << 3))
397
403 #define VERSIONR           (_W5500_IO_BASE_
    + (0x0039 << 8) + (WIZCHIP_CREG_BLOCK << 3))
404
405
406 //------------------------------ W5500 Socket
    Registers IOMAP ------------------------------
437 #define Sn_MR(N)           (_W5500_IO_BASE_
    + (0x0000 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
```

```
    3))
438
456 #define Sn_CR(N)            (_W5500_IO_BASE_
    + (0x0001 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
457
474 #define Sn_IR(N)            (_W5500_IO_BASE_
    + (0x0002 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
475
497 #define Sn_SR(N)            (_W5500_IO_BASE_
    + (0x0003 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
498
505 #define Sn_PORT(N)          (_W5500_IO_BASE_
    + (0x0004 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
506
513 #define Sn_DHAR(N)          (_W5500_IO_BASE_
    + (0x0006 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
514
523 #define Sn_DIPR(N)          (_W5500_IO_BASE_
    + (0x000C << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
524
533 #define Sn_DPORT(N)         (_W5500_IO_BASE_
    + (0x0010 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
534
540 #define Sn_MSSR(N)          (_W5500_IO_BASE_
    + (0x0012 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
541
542 // Reserved
    (_W5500_IO_BASE_ + (0x0014 << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
```

```
543
550 #define Sn_TOS(N)          (_W5500_IO_BASE_
    + (0x0015 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
551
557 #define Sn_TTL(N)          (_W5500_IO_BASE_
    + (0x0016 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
558 // Reserved
    (_W5500_IO_BASE_ + (0x0017 << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
559 // Reserved
    (_W5500_IO_BASE_ + (0x0018 << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
560 // Reserved
    (_W5500_IO_BASE_ + (0x0019 << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
561 // Reserved
    (_W5500_IO_BASE_ + (0x001A << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
562 // Reserved
    (_W5500_IO_BASE_ + (0x001B << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
563 // Reserved
    (_W5500_IO_BASE_ + (0x001C << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
564 // Reserved
    (_W5500_IO_BASE_ + (0x001D << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
565
576 #define Sn_RXBUF_SIZE(N)   (_W5500_IO_BASE_
    + (0x001E << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
577
587 #define Sn_TXBUF_SIZE(N)   (_W5500_IO_BASE_
    + (0x001F << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
```

```
588
598 #define Sn_TX_FSR(N)       (_W5500_IO_BASE_
    + (0x0020 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
599
610 #define Sn_TX_RD(N)        (_W5500_IO_BASE_
    + (0x0022 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
611
624 #define Sn_TX_WR(N)        (_W5500_IO_BASE_
    + (0x0024 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
625
633 #define Sn_RX_RSR(N)       (_W5500_IO_BASE_
    + (0x0026 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
634
646 #define Sn_RX_RD(N)        (_W5500_IO_BASE_
    + (0x0028 << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
647
655 #define Sn_RX_WR(N)        (_W5500_IO_BASE_
    + (0x002A << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
656
665 #define Sn_IMR(N)          (_W5500_IO_BASE_
    + (0x002C << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
666
672 #define Sn_FRAG(N)         (_W5500_IO_BASE_
    + (0x002D << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
673
685 #define Sn_KPALVTR(N)      (_W5500_IO_BASE_
    + (0x002F << 8) + (WIZCHIP_SREG_BLOCK(N) <<
    3))
686
```

```
687 //#define Sn_TSR(N)
    (_W5500_IO_BASE_ + (0x0030 << 8) +
    (WIZCHIP_SREG_BLOCK(N) << 3))
688
689
690 //------------------------------ W5500
    Register values  ------------------------------
691
692 /* MODE register values */
697 #define MR_RST                          0x80
698
708 #define MR_WOL                          0x20
709
716 #define MR_PB                           0x10
717
724 #define MR_PPPOE                        0x08
725
732 #define MR_FARP                         0x02
733
734 /* IR register values */
739 #define IR_CONFLICT                     0x80
740
746 #define IR_UNREACH                      0x40
747
752 #define IR_PPPoE                        0x20
753
758 #define IR_MP                           0x10
759
760
761 /* PHYCFGR register value */
762 #define PHYCFGR_RST                     ~(1<<7)
    //< For PHY reset, must operate AND mask.
763 #define PHYCFGR_OPMD                    (1<<6)
    // Configre PHY with OPMDC value
764 #define PHYCFGR_OPMDC_ALLA              (7<<3)
765 #define PHYCFGR_OPMDC_PDOWN             (6<<3)
766 #define PHYCFGR_OPMDC_NA                (5<<3)
```

```
767 #define PHYCFGR_OPMDC_100FA             (4<<3)
768 #define PHYCFGR_OPMDC_100F              (3<<3)
769 #define PHYCFGR_OPMDC_100H              (2<<3)
770 #define PHYCFGR_OPMDC_10F               (1<<3)
771 #define PHYCFGR_OPMDC_10H               (0<<3)
772 #define PHYCFGR_DPX_FULL                (1<<2)
773 #define PHYCFGR_DPX_HALF                (0<<2)
774 #define PHYCFGR_SPD_100                 (1<<1)
775 #define PHYCFGR_SPD_10                  (0<<1)
776 #define PHYCFGR_LNK_ON                  (1<<0)
777 #define PHYCFGR_LNK_OFF                 (0<<0)
778
779 /* IMR register values */
785 #define IM_IR7                          0x80
786
792 #define IM_IR6                          0x40
793
799 #define IM_IR5                          0x20
800
806 #define IM_IR4                          0x10
807
808 /* Sn_MR Default values */
817 #define Sn_MR_MULTI                     0x80
818
826 #define Sn_MR_BCASTB                    0x40
827
836 #define Sn_MR_ND                        0x20
837
844 #define Sn_MR_UCASTB                    0x10
845
851 #define Sn_MR_MACRAW                    0x04
852
853 //#define Sn_MR_IPRAW                     0x03
    /**< IP LAYER RAW SOCK */
854
859 #define Sn_MR_UDP                       0x02
860
```

```
865 #define Sn_MR_TCP                       0x01
866
871 #define Sn_MR_CLOSE                     0x00
872
873 /* Sn_MR values used with Sn_MR_MACRAW */
884 #define Sn_MR_MFEN
    Sn_MR_MULTI
885
893 #define Sn_MR_MMB
    Sn_MR_ND
894
901 #define Sn_MR_MIP6B
    Sn_MR_UCASTB
902
903 /* Sn_MR value used with Sn_MR_UDP &
    Sn_MR_MULTI */
910 #define Sn_MR_MC
    Sn_MR_ND
911
912 /* Sn_MR alternate values */
916 #define SOCK_STREAM
    Sn_MR_TCP
917
921 #define SOCK_DGRAM
    Sn_MR_UDP
922
923
924 /* Sn_CR values */
937 #define Sn_CR_OPEN                      0x01
938
948 #define Sn_CR_LISTEN                    0x02
949
960 #define Sn_CR_CONNECT                   0x04
961
973 #define Sn_CR_DISCON                    0x08
974
979 #define Sn_CR_CLOSE                     0x10
```

```
 980
 987 #define Sn_CR_SEND                  0x20
 988
 997 #define Sn_CR_SEND_MAC              0x21
 998
1005 #define Sn_CR_SEND_KEEP             0x22
1006
1013 #define Sn_CR_RECV                  0x40
1014
1015 /* Sn_IR values */
1020 #define Sn_IR_SENDOK                0x10
1021
1026 #define Sn_IR_TIMEOUT               0x08
1027
1032 #define Sn_IR_RECV                  0x04
1033
1038 #define Sn_IR_DISCON                0x02
1039
1044 #define Sn_IR_CON                   0x01
1045
1046 /* Sn_SR values */
1052 #define SOCK_CLOSED                 0x00
1053
1060 #define SOCK_INIT                   0x13
1061
1068 #define SOCK_LISTEN                 0x14
1069
1077 #define SOCK_SYNSENT                0x15
1078
1085 #define SOCK_SYNRECV                0x16
1086
1094 #define SOCK_ESTABLISHED            0x17
1095
1102 #define SOCK_FIN_WAIT               0x18
1103
1110 #define SOCK_CLOSING                0x1A
1111
```

```
1118 #define SOCK_TIME_WAIT                  0x1B
1119
1126 #define SOCK_CLOSE_WAIT                 0x1C
1127
1133 #define SOCK_LAST_ACK                   0x1D
1134
1141 #define SOCK_UDP                        0x22
1142
1143 //#define SOCK_IPRAW                      0x32
    /**< IP raw mode socket */
1144
1151 #define SOCK_MACRAW                     0x42
1152
1153 //#define SOCK_PPPOE                      0x5F
1154
1155 /* IP PROTOCOL */
1156 #define IPPROTO_IP                      0
    //< Dummy for IP
1157 #define IPPROTO_ICMP                    1
    //< Control message protocol
1158 #define IPPROTO_IGMP                    2
    //< Internet group management protocol
1159 #define IPPROTO_GGP                     3
    //< Gateway^2 (deprecated)
1160 #define IPPROTO_TCP                     6
    //< TCP
1161 #define IPPROTO_PUP                     12
    //< PUP
1162 #define IPPROTO_UDP                     17
    //< UDP
1163 #define IPPROTO_IDP                     22
    //< XNS idp
1164 #define IPPROTO_ND                      77
    //< UNOFFICIAL net disk protocol
1165 #define IPPROTO_RAW                     255
    //< Raw IP packet
1166
```

```
1167
1179 #define WIZCHIP_CRITICAL_ENTER()
     WIZCHIP.CRIS._enter()
1180
1181 #ifdef _exit
1182 #undef _exit
1183 #endif
1184
1196 #define WIZCHIP_CRITICAL_EXIT()
     WIZCHIP.CRIS._exit()
1197
1198
1200 // Basic I/O Function //
1202
1209 uint8_t  WIZCHIP_READ (uint32_t AddrSel);
1210
1218 void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t
     wb );
1219
1227 void WIZCHIP_READ_BUF (uint32_t AddrSel,
     uint8_t* pBuf, uint16_t len);
1228
1236 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
     uint8_t* pBuf, uint16_t len);
1237
1239 // Common Register I/O function //
1241
1247 #define setMR(mr) \
1248     WIZCHIP_WRITE(MR,mr)
1249
1250
1257 #define getMR() \
1258         WIZCHIP_READ(MR)
1259
1266 #define setGAR(gar) \
1267         WIZCHIP_WRITE_BUF(GAR,gar,4)
1268
```

```
1275 #define getGAR(gar) \
1276         WIZCHIP_READ_BUF(GAR,gar,4)
1277
1284 #define setSUBR(subr) \
1285         WIZCHIP_WRITE_BUF(SUBR, subr,4)
1286
1287
1294 #define getSUBR(subr) \
1295         WIZCHIP_READ_BUF(SUBR, subr, 4)
1296
1303 #define setSHAR(shar) \
1304         WIZCHIP_WRITE_BUF(SHAR, shar, 6)
1305
1312 #define getSHAR(shar) \
1313         WIZCHIP_READ_BUF(SHAR, shar, 6)
1314
1321 #define setSIPR(sipr) \
1322         WIZCHIP_WRITE_BUF(SIPR, sipr, 4)
1323
1330 #define getSIPR(sipr) \
1331         WIZCHIP_READ_BUF(SIPR, sipr, 4)
1332
1339 #define setINTLEVEL(intlevel)  {\
1340         WIZCHIP_WRITE(INTLEVEL,   (uint8_t)
    (intlevel >> 8)); \
1341
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL,1),
    (uint8_t) intlevel); \
1342     }
1343
1344
1351 //M20150401 : Type explict declaration
1352 /*
1353 #define getINTLEVEL() \
1354         ((WIZCHIP_READ(INTLEVEL) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL,1)))
1355 */
```

```
1356 #define getINTLEVEL() \
1357         (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL,1)))
1358
1365 #define setIR(ir) \
1366         WIZCHIP_WRITE(IR, (ir & 0xF0))
1367
1374 #define getIR() \
1375         (WIZCHIP_READ(IR) & 0xF0)
1376
1382 #define setIMR(imr) \
1383         WIZCHIP_WRITE(_IMR_, imr)
1384
1391 #define getIMR() \
1392         WIZCHIP_READ(_IMR_)
1393
1400 #define setSIR(sir) \
1401         WIZCHIP_WRITE(SIR, sir)
1402
1409 #define getSIR() \
1410         WIZCHIP_READ(SIR)
1411
1417 #define setSIMR(simr) \
1418         WIZCHIP_WRITE(SIMR, simr)
1419
1426 #define getSIMR() \
1427         WIZCHIP_READ(SIMR)
1428
1435 #define setRTR(rtr)   {\
1436         WIZCHIP_WRITE(_RTR_,   (uint8_t)(rtr >> 8)); \
1437 WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1), (uint8_t) rtr); \
1438     }
1439
```

```
1446 //M20150401 : Type explict declaration
1447 /*
1448 #define getRTR() \
1449         ((WIZCHIP_READ(_RTR_) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))
1450 */
1451 #define getRTR() \
1452         (((uint16_t)WIZCHIP_READ(_RTR_) <<
    8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))
1453
1454
1461 #define setRCR(rcr) \
1462         WIZCHIP_WRITE(_RCR_, rcr)
1463
1470 #define getRCR() \
1471         WIZCHIP_READ(_RCR_)
1472
1473 //==========================================
    ======== test done
    ==========================================
    ============
1474
1481 #define setPTIMER(ptimer) \
1482         WIZCHIP_WRITE(PTIMER, ptimer)
1483
1490 #define getPTIMER() \
1491         WIZCHIP_READ(PTIMER)
1492
1499 #define setPMAGIC(pmagic) \
1500         WIZCHIP_WRITE(PMAGIC, pmagic)
1501
1508 #define getPMAGIC() \
1509         WIZCHIP_READ(PMAGIC)
1510
1517 #define setPHAR(phar) \
1518         WIZCHIP_WRITE_BUF(PHAR, phar, 6)
```

```
1519
1526 #define getPHAR(phar) \
1527         WIZCHIP_READ_BUF(PHAR, phar, 6)
1528
1535 #define setPSID(psid)  {\
1536         WIZCHIP_WRITE(PSID,   (uint8_t)(psid
    >> 8)); \
1537
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PSID,1),
    (uint8_t) psid); \
1538     }
1539
1546 //uint16_t getPSID(void);
1547 //M20150401 : Type explict declaration
1548 /*
1549 #define getPSID() \
1550         ((WIZCHIP_READ(PSID) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1)))
1551 */
1552 #define getPSID() \
1553         (((uint16_t)WIZCHIP_READ(PSID) << 8)
    + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1)))
1554
1561 #define setPMRU(pmru) { \
1562         WIZCHIP_WRITE(PMRU,   (uint8_t)
    (pmru>>8)); \
1563
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PMRU,1),
    (uint8_t) pmru); \
1564     }
1565
1572 //M20150401 : Type explict declaration
1573 /*
1574 #define getPMRU() \
1575         ((WIZCHIP_READ(PMRU) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))
1576 */
```

```
1577 #define getPMRU() \
1578         (((uint16_t)WIZCHIP_READ(PMRU) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))
1579
1585 //M20150401 : Size Error of UIPR (6 -> 4)
1586 /*
1587 #define getUIPR(uipr) \
1588         WIZCHIP_READ_BUF(UIPR,uipr,6)
1589 */
1590 #define getUIPR(uipr) \
1591         WIZCHIP_READ_BUF(UIPR,uipr,4)
1592
1598 //M20150401 : Type explict declaration
1599 /*
1600 #define getUPORTR() \
1601     ((WIZCHIP_READ(UPORTR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1)))
1602 */
1603 #define getUPORTR() \
1604     (((uint16_t)WIZCHIP_READ(UPORTR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1)))
1605
1612 #define setPHYCFGR(phycfgr) \
1613         WIZCHIP_WRITE(PHYCFGR, phycfgr)
1614
1621 #define getPHYCFGR() \
1622         WIZCHIP_READ(PHYCFGR)
1623
1629 #define getVERSIONR() \
1630         WIZCHIP_READ(VERSIONR)
1631
1633
1635 // Socket N register I/O function //
1637
1644 #define setSn_MR(sn, mr) \
1645         WIZCHIP_WRITE(Sn_MR(sn),mr)
1646
```

```
1654 #define getSn_MR(sn) \
1655     WIZCHIP_READ(Sn_MR(sn))
1656
1664 #define setSn_CR(sn, cr) \
1665         WIZCHIP_WRITE(Sn_CR(sn), cr)
1666
1674 #define getSn_CR(sn) \
1675         WIZCHIP_READ(Sn_CR(sn))
1676
1684 #define setSn_IR(sn, ir) \
1685         WIZCHIP_WRITE(Sn_IR(sn), (ir &
    0x1F))
1686
1694 #define getSn_IR(sn) \
1695         (WIZCHIP_READ(Sn_IR(sn)) & 0x1F)
1696
1704 #define setSn_IMR(sn, imr) \
1705         WIZCHIP_WRITE(Sn_IMR(sn), (imr &
    0x1F))
1706
1714 #define getSn_IMR(sn) \
1715         (WIZCHIP_READ(Sn_IMR(sn)) & 0x1F)
1716
1723 #define getSn_SR(sn) \
1724         WIZCHIP_READ(Sn_SR(sn))
1725
1733 #define setSn_PORT(sn, port)  { \
1734         WIZCHIP_WRITE(Sn_PORT(sn),
    (uint8_t)(port >> 8)); \
1735
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1
    ), (uint8_t) port); \
1736     }
1737
1745 //M20150401 : Type explict declaration
1746 /*
1747 #define getSn_PORT(sn) \
```

```
1748            ((WIZCHIP_READ(Sn_PORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    ))
1749  */
1750  #define getSn_PORT(sn) \
1751
    (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)
    ))
1752
1760  #define setSn_DHAR(sn, dhar) \
1761            WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar,
    6)
1762
1770  #define getSn_DHAR(sn, dhar) \
1771            WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar,
    6)
1772
1780  #define setSn_DIPR(sn, dipr) \
1781            WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr,
    4)
1782
1790  #define getSn_DIPR(sn, dipr) \
1791            WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr,
    4)
1792
1800  #define setSn_DPORT(sn, dport) { \
1801            WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \
1802
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),
    1), (uint8_t)  dport); \
1803        }
1804
1812  //M20150401 : Type explict declaration
1813  /*
1814  #define getSn_DPORT(sn) \
```

```
1815            ((WIZCHIP_READ(Sn_DPORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    )))
1816    */
1817    #define getSn_DPORT(sn) \
1818
    (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    )))
1819
1827    #define setSn_MSSR(sn, mss) { \
1828            WIZCHIP_WRITE(Sn_MSSR(sn),
    (uint8_t)(mss>>8)); \
1829
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1
    ), (uint8_t) mss); \
1830        }
1831
1839    //M20150401 : Type explict declaration
1840    /*
1841    #define getSn_MSSR(sn) \
1842            ((WIZCHIP_READ(Sn_MSSR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    ))
1843    */
1844    #define getSn_MSSR(sn) \
1845
    (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
    ))
1846
1854    #define setSn_TOS(sn, tos) \
1855            WIZCHIP_WRITE(Sn_TOS(sn), tos)
1856
1864    #define getSn_TOS(sn) \
1865            WIZCHIP_READ(Sn_TOS(sn))
1866
```

```
1874 #define setSn_TTL(sn, ttl) \
1875         WIZCHIP_WRITE(Sn_TTL(sn), ttl)
1876
1877
1885 #define getSn_TTL(sn) \
1886         WIZCHIP_READ(Sn_TTL(sn))
1887
1888
1896 #define setSn_RXBUF_SIZE(sn, rxbufsize) \
1897
    WIZCHIP_WRITE(Sn_RXBUF_SIZE(sn),rxbufsize)
1898
1899
1907 #define getSn_RXBUF_SIZE(sn) \
1908         WIZCHIP_READ(Sn_RXBUF_SIZE(sn))
1909
1917 #define setSn_TXBUF_SIZE(sn, txbufsize) \
1918         WIZCHIP_WRITE(Sn_TXBUF_SIZE(sn),
    txbufsize)
1919
1927 #define getSn_TXBUF_SIZE(sn) \
1928         WIZCHIP_READ(Sn_TXBUF_SIZE(sn))
1929
1936 uint16_t getSn_TX_FSR(uint8_t sn);
1937
1944 //M20150401 : Type explict declaration
1945 /*
1946 #define getSn_TX_RD(sn) \
1947         ((WIZCHIP_READ(Sn_TX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1
    )))
1948 */
1949 #define getSn_TX_RD(sn) \
1950
    (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1
    )))
```

```
1951
1959 #define setSn_TX_WR(sn, txwr) { \
1960         WIZCHIP_WRITE(Sn_TX_WR(sn),
    (uint8_t)(txwr>>8)); \
1961
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),
    1), (uint8_t) txwr); \
1962         }
1963
1971 //M20150401 : Type explict declaration
1972 /*
1973 #define getSn_TX_WR(sn) \
1974         ((WIZCHIP_READ(Sn_TX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    )))
1975 */
1976 #define getSn_TX_WR(sn) \
1977
    (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1
    )))
1978
1979
1986 uint16_t getSn_RX_RSR(uint8_t sn);
1987
1988
1996 #define setSn_RX_RD(sn, rxrd) { \
1997         WIZCHIP_WRITE(Sn_RX_RD(sn),
    (uint8_t)(rxrd>>8)); \
1998
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),
    1), (uint8_t) rxrd); \
1999     }
2000
2008 //M20150401 : Type explict declaration
2009 /*
2010 #define getSn_RX_RD(sn) \
```

```
2011          ((WIZCHIP_READ(Sn_RX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    )))
2012  */
2013  #define getSn_RX_RD(sn) \
2014
    (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1
    )))
2015
2022  //M20150401 : Type explict declaration
2023  /*
2024  #define getSn_RX_WR(sn) \
2025          ((WIZCHIP_READ(Sn_RX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1
    )))
2026  */
2027  #define getSn_RX_WR(sn) \
2028
    (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1
    )))
2029
2037  #define setSn_FRAG(sn, frag) { \
2038          WIZCHIP_WRITE(Sn_FRAG(sn),
    (uint8_t)(frag >>8)); \
2039
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1
    ), (uint8_t) frag); \
2040      }
2041
2049  //M20150401 : Type explict declaration
2050  /*
2051  #define getSn_FRAG(sn) \
2052          ((WIZCHIP_READ(Sn_FRAG(sn)) << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)
    ))
```

```
2053 */
2054 #define getSn_FRAG(sn) \
2055       (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)))
2056
2064 #define setSn_KPALVTR(sn, kpalvt) \
2065         WIZCHIP_WRITE(Sn_KPALVTR(sn), kpalvt)
2066
2074 #define getSn_KPALVTR(sn) \
2075         WIZCHIP_READ(Sn_KPALVTR(sn))
2076
2078
2080 // Sn_TXBUF & Sn_RXBUF IO function //
2082
2088 //M20150401 : Type explict declaration
2089 /*
2090 #define getSn_RxMAX(sn) \
2091         (getSn_RXBUF_SIZE(sn) << 10)
2092 */
2093 #define getSn_RxMAX(sn) \
2094         (((uint16_t)getSn_RXBUF_SIZE(sn)) << 10)
2095
2102 //M20150401 : Type explict declaration
2103 /*
2104 #define getSn_TxMAX(sn) \
2105         (getSn_TXBUF_SIZE(sn) << 10)
2106 */
2107 #define getSn_TxMAX(sn) \
2108         (((uint16_t)getSn_TXBUF_SIZE(sn)) << 10)
2109
2124 void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len);
```

```
2125
2140 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len);
2141
2149 void wiz_recv_ignore(uint8_t sn, uint16_t
    len);
2150
2152 #endif
2153
2155 #endif   // _W5500_H_
```

Socket APIs

__WIZCHIP Member List

This is the complete list of members for __**WIZCHIP**, including all inherited members.

CRIS	**__WIZCHIP**
CS	**__WIZCHIP**
id	**__WIZCHIP**
IF	**__WIZCHIP**
if_mode	**__WIZCHIP**

Socket APIs

__WIZCHIP::_IF Member List

This is the complete list of members for **__WIZCHIP::_IF**, including all inherited members.

_read_burst	**__WIZCHIP::_IF**
_read_byte	**__WIZCHIP::_IF**
_read_data	**__WIZCHIP::_IF**
_write_burst	**__WIZCHIP::_IF**
_write_byte	**__WIZCHIP::_IF**
_write_data	**__WIZCHIP::_IF**
BUS	**__WIZCHIP::_IF**
SPI	**__WIZCHIP::_IF**

Socket APIs

__WIZCHIP::_CS Member List

This is the complete list of members for **__WIZCHIP::_CS**, including all inherited members.

_deselect	**__WIZCHIP::_CS**
_select	**__WIZCHIP::_CS**

Socket APIs

__WIZCHIP::_CRIS Member List

This is the complete list of members for **__WIZCHIP::_CRIS**, including all inherited members.

_enter	**__WIZCHIP::_CRIS**
_exit	**__WIZCHIP::_CRIS**

Socket APIs

wiz_PhyConf_t Member List

This is the complete list of members for **wiz_PhyConf_t**, including all inherited members.

by	**wiz_PhyConf_t**
duplex	**wiz_PhyConf_t**
mode	**wiz_PhyConf_t**
speed	**wiz_PhyConf_t**

Socket APIs

wiz_NetInfo_t Member List

This is the complete list of members for **wiz_NetInfo_t**, including all inherited members.

dhcp	**wiz_NetInfo_t**
dns	**wiz_NetInfo_t**
gw	**wiz_NetInfo_t**
ip	**wiz_NetInfo_t**
mac	**wiz_NetInfo_t**
sn	**wiz_NetInfo_t**

Socket APIs

wiz_NetTimeout_t Member List

This is the complete list of members for **wiz_NetTimeout_t**, including all inherited members.

retry_cnt	**wiz_NetTimeout_t**
time_100us	**wiz_NetTimeout_t**

Socket APIs

w5100.c

Go to the documentation of this file.

```
  1 //*****************************************************************************
  2 //
 38 //
 39 //*****************************************************************************
 40
 41 #include "w5100.h"
 42
 43 #if   (_WIZCHIP_ == 5100)
 44
 47 void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb )
 48 {
 49    WIZCHIP_CRITICAL_ENTER();
 50    WIZCHIP.CS._select();
 51
 52 #if( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_))
 53    WIZCHIP.IF.SPI._write_byte(0xF0);
 54    WIZCHIP.IF.SPI._write_byte((AddrSel & 0xFF00) >>  8);
 55    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x00FF) >>  0);
 56    WIZCHIP.IF.SPI._write_byte(wb);    // Data write (write 1byte data)
```

```
57 #elif ( (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_) )
58     //M20150601 : Rename the function for integrating with ioLibrary
59     //WIZCHIP.IF.BUS._write_byte(AddrSel,wb);
60     WIZCHIP.IF.BUS._write_data(AddrSel,wb);
61 #elif ( (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_) )
62
63     //add indirect bus
64     //M20150601 : Rename the function for integrating with ioLibrary
65     //WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0xFF00) >>  8);
66     //WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x00FF));
67     //WIZCHIP.IF.BUS._write_byte(IDM_DR,wb);
68     WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0xFF00) >>  8);
69     WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x00FF));
70     WIZCHIP.IF.BUS._write_data(IDM_DR,wb);
71 #else
72     #error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!"
73 #endif
74
75     WIZCHIP.CS._deselect();
76     WIZCHIP_CRITICAL_EXIT();
77 }
81 uint8_t  WIZCHIP_READ(uint32_t AddrSel)
82 {
83     uint8_t ret;
84
85     WIZCHIP_CRITICAL_ENTER();
86     WIZCHIP.CS._select();
87
```

```
 88 #if( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
 89     WIZCHIP.IF.SPI._write_byte(0x0F);
 90     WIZCHIP.IF.SPI._write_byte((AddrSel &
    0xFF00) >>  8);
 91     WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x00FF) >>  0);
 92     ret = WIZCHIP.IF.SPI._read_byte();
 93 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_) )
 94     //M20150601 : Rename the function for
    integrating with ioLibrary
 95     //ret =
    WIZCHIP.IF.BUS._read_byte(AddrSel);
 96     ret = WIZCHIP.IF.BUS._read_data(AddrSel);
 97 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_) )
 98
 99     //add indirect bus
100     //M20150601 : Rename the function for
    integrating with ioLibrary
101     //WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
102     //WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x00FF));
103     //ret =
    WIZCHIP.IF.BUS._read_byte(IDM_DR);
104     WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
105     WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x00FF));
106     ret = WIZCHIP.IF.BUS._read_data(IDM_DR);
107
108 #else
109     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5100. !!!"
110 #endif
```

```
111
112     WIZCHIP.CS._deselect();
113     WIZCHIP_CRITICAL_EXIT();
114     return ret;
115 }
116
117
121 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
    uint8_t* pBuf, uint16_t len)
122 {
123     uint16_t i = 0;
124
125     WIZCHIP_CRITICAL_ENTER();
126     WIZCHIP.CS._select();   //M20150601 :
    Moved here.
127
128 #if( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
129     for(i = 0; i < len; i++)
130     {
131       //M20160715 : Depricated "M20150601 :
    Remove _select() to top-side"
132       //            CS should be controlled
    every SPI frames
133       WIZCHIP.CS._select();
134       WIZCHIP.IF.SPI._write_byte(0xF0);
135       WIZCHIP.IF.SPI._write_byte((((uint16_t)
    (AddrSel+i)) & 0xFF00) >>  8);
136       WIZCHIP.IF.SPI._write_byte((((uint16_t)
    (AddrSel+i)) & 0x00FF) >>  0);
137       WIZCHIP.IF.SPI._write_byte(pBuf[i]);
    // Data write (write 1byte data)
138       //M20160715 : Depricated "M20150601 :
    Remove _select() to top-side"
139        WIZCHIP.CS._deselect();
140     }
141 #elif ( (_WIZCHIP_IO_MODE_ ==
```

```
    _WIZCHIP_IO_MODE_BUS_DIR_) )
142     for(i = 0; i < len; i++)
143     //M20150601 : Rename the function for
    integrating with ioLibrary
144     //
    WIZCHIP.IF.BUS._write_byte(AddrSel+i,pBuf[i]);

145
    WIZCHIP.IF.BUS._write_data(AddrSel+i,pBuf[i]);

146 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_) )
147     //M20150601 : Rename the function for
    integrating with ioLibrary
148     /*
149     WIZCHIP_WRITE(MR,WIZCHIP_READ(MR) |
    MR_AI);
150     WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
151     WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x00FF));
152     for(i = 0 ; i < len; i++)
153
    WIZCHIP.IF.BUS._write_byte(IDM_DR,pBuf[i]);
154     WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) &
    ~MR_AI);
155     */
156     setMR(getMR()|MR_AI);
157     WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
158     WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x00FF));
159     for(i = 0 ; i < len; i++)
160
    WIZCHIP.IF.BUS._write_data(IDM_DR,pBuf[i]);
161     setMR(getMR() & ~MR_AI);
162
```

```
163 #else
164     #error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!!"
165 #endif
166
167     WIZCHIP.CS._deselect();  //M20150601 : Moved here.
168     WIZCHIP_CRITICAL_EXIT();
169 }
170
175 void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len)
176 {
177     uint16_t i = 0;
178     WIZCHIP_CRITICAL_ENTER();
179     WIZCHIP.CS._select();   //M20150601 : Moved here.
180
181     #if( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_))
182     for(i = 0; i < len; i++)
183     {
184       //M20160715 : Depricated "M20150601 : Remove _select() to top-side"
185       //            CS should be controlled every SPI frames
186       WIZCHIP.CS._select();
187       WIZCHIP.IF.SPI._write_byte(0x0F);
188       WIZCHIP.IF.SPI._write_byte((uint16_t)((AddrSel+i) & 0xFF00) >>  8);
189       WIZCHIP.IF.SPI._write_byte((uint16_t)((AddrSel+i) & 0x00FF) >>  0);
190       pBuf[i] = WIZCHIP.IF.SPI._read_byte();
191       //M20160715 : Depricated "M20150601 : Remove _select() to top-side"
192       WIZCHIP.CS._deselect();
193     }
```

```
194 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_) )
195     for(i = 0 ; i < len; i++)
196     //M20150601 : Rename the function for
    integrating with ioLibrary
197     // pBuf[i]   =
    WIZCHIP.IF.BUS._read_byte(AddrSel+i);
198         pBuf[i]   =
    WIZCHIP.IF.BUS._read_data(AddrSel+i);
199 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_) )
200     //M20150601 : Rename the function for
    integrating with ioLibrary
201     /*
202     WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) |
    MR_AI);
203     WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
204     WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x00FF));
205     for(i = 0 ; i < len; i++)
206         pBuf[i]   =
    WIZCHIP.IF.BUS._read_byte(IDM_DR);
207     WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) &
    ~MR_AI);
208     */
209     setMR(getMR() | MR_AI);
210     WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0xFF00) >>  8);
211     WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x00FF));
212     for(i = 0 ; i < len; i++)
213         pBuf[i]   =
    WIZCHIP.IF.BUS._read_data(IDM_DR);
214     setMR(getMR() & ~MR_AI);
215
216 #else
```

```
217     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5100. !!!!"
218 #endif
219
220     WIZCHIP.CS._deselect();    //M20150601 :
    Moved Here.
221     WIZCHIP_CRITICAL_EXIT();
222 }
223
225 // Socket N regsiter IO function //
227
228 uint16_t getSn_TX_FSR(uint8_t sn)
229 {
230     uint16_t val=0,val1=0;
231     do
232     {
233        val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
234        val1 = (val1 << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
235        if (val1 != 0)
236        {
237          val = WIZCHIP_READ(Sn_TX_FSR(sn));
238          val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
239        }
240     }while (val != val1);
241     return val;
242 }
243
244
245 uint16_t getSn_RX_RSR(uint8_t sn)
246 {
247     uint16_t val=0,val1=0;
248     do
249     {
```

```
250         val1 = WIZCHIP_READ(Sn_RX_RSR(sn));
251         val1 = (val1 << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
    1));
252         if (val1 != 0)
253         {
254           val = WIZCHIP_READ(Sn_RX_RSR(sn));
255           val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
    1));
256         }
257      }while (val != val1);
258      return val;
259   }
260
262   // Sn_TXBUF & Sn_RXBUF IO function //
264   uint32_t getSn_RxBASE(uint8_t sn)
265   {
266      int8_t  i;
267   #if ( _WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_)
268      uint32_t rxbase = _W5100_IO_BASE_ +
    _WIZCHIP_IO_RXBUF_;
269   #else
270      uint32_t rxbase = _WIZCHIP_IO_RXBUF_;
271   #endif
272      for(i = 0; i < sn; i++)
273         rxbase += getSn_RxMAX(i);
274
275      return rxbase;
276   }
277
278   uint32_t getSn_TxBASE(uint8_t sn)
279   {
280      int8_t  i;
281   #if ( _WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_)
```

```
282     uint32_t txbase = _W5100_IO_BASE_ +
    _WIZCHIP_IO_TXBUF_;
283  #else
284     uint32_t txbase = _WIZCHIP_IO_TXBUF_;
285  #endif
286     for(i = 0; i < sn; i++)
287        txbase += getSn_TxMAX(i);
288     return txbase;
289  }
290
302  void wiz_send_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len)
303  {
304    uint16_t ptr;
305    uint16_t size;
306    uint16_t dst_mask;
307    uint16_t dst_ptr;
308
309    ptr = getSn_TX_WR(sn);
310
311    dst_mask = ptr & getSn_TxMASK(sn);
312    dst_ptr = getSn_TxBASE(sn) + dst_mask;
313
314    if (dst_mask + len > getSn_TxMAX(sn))
315    {
316      size = getSn_TxMAX(sn) - dst_mask;
317      WIZCHIP_WRITE_BUF(dst_ptr, wizdata,
    size);
318      wizdata += size;
319      size = len - size;
320      dst_ptr = getSn_TxBASE(sn);
321      WIZCHIP_WRITE_BUF(dst_ptr, wizdata,
    size);
322    }
323    else
324    {
325      WIZCHIP_WRITE_BUF(dst_ptr, wizdata,
```

```
    len);
  326   }
  327
  328   ptr += len;
  329
  330   setSn_TX_WR(sn, ptr);
  331 }
  332
  333
  344 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len)
  345 {
  346   uint16_t ptr;
  347   uint16_t size;
  348   uint16_t src_mask;
  349   uint16_t src_ptr;
  350
  351   ptr = getSn_RX_RD(sn);
  352
  353   src_mask = (uint32_t)ptr &
    getSn_RxMASK(sn);
  354   src_ptr = (getSn_RxBASE(sn) + src_mask);
  355
  356
  357   if( (src_mask + len) > getSn_RxMAX(sn) )
  358   {
  359     size = getSn_RxMAX(sn) - src_mask;
  360     WIZCHIP_READ_BUF((uint32_t)src_ptr,
    (uint8_t*)wizdata, size);
  361     wizdata += size;
  362     size = len - size;
  363     src_ptr = getSn_RxBASE(sn);
  364     WIZCHIP_READ_BUF(src_ptr,
    (uint8_t*)wizdata, size);
  365   }
  366   else
  367   {
```

```
368       WIZCHIP_READ_BUF(src_ptr,
    (uint8_t*)wizdata, len);
369     }
370
371     ptr += len;
372
373     setSn_RX_RD(sn, ptr);
374   }
375
376   void wiz_recv_ignore(uint8_t sn, uint16_t
    len)
377   {
378     uint16_t ptr;
379
380     ptr = getSn_RX_RD(sn);
381
382     ptr += len;
383     setSn_RX_RD(sn,ptr);
384   }
385
386   #endif
```

Socket APIs

w5200.c

Go to the documentation of this file.

```
  1 //*****************************************************
    ********************************
  2 //
 38 //
 39 //*****************************************************
    ********************************
 40
 41 #include "w5200.h"
 42
 43 #if   (_WIZCHIP_ == 5200)
 44
 47 void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t
    wb )
 48 {
 49     WIZCHIP_CRITICAL_ENTER();
 50    WIZCHIP.CS._select();
 51
 52 #if ( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
 53    WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x0000FF00) >>  8);
 54    WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x000000FF) >>  0);
 55
    WIZCHIP.IF.SPI._write_byte(_W5200_SPI_WRITE_);
      // Data write command and Write data length
```

```
    upper
56      WIZCHIP.IF.SPI._write_byte(0x01);  //
    Write data length lower
57      WIZCHIP.IF.SPI._write_byte(wb);    //
    Data write (write 1byte data)
58
59  #elif ( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_) )
60
61      //add indirect bus
62      //M20150601 : Rename the function for
    integrating with W5300
63      //WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
64      //WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x000000FF));
65      //WIZCHIP.IF.BUS._write_byte(IDM_DR,wb);
66      WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
67      WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x000000FF));
68      WIZCHIP.IF.BUS._write_data(IDM_DR,wb);
69
70  #else
71      #error "Unknown _WIZCHIP_IO_MODE_ in
    W5200. !!!"
72  #endif
73
74      WIZCHIP.CS._deselect();
75      WIZCHIP_CRITICAL_EXIT();
76  }
80  uint8_t  WIZCHIP_READ(uint32_t AddrSel)
81  {
82      uint8_t ret;
83
84      WIZCHIP_CRITICAL_ENTER();
85      WIZCHIP.CS._select();
```

```
 86
 87 #if( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
 88     WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x0000FF00) >>  8);
 89     WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x000000FF) >>  0);
 90
    WIZCHIP.IF.SPI._write_byte(_W5200_SPI_READ_);
    // Read data length upper
 91     WIZCHIP.IF.SPI._write_byte(0x01);
    // Data length lower
 92     ret = WIZCHIP.IF.SPI._read_byte();
 93
 94 #elif ( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_) )
 95
 96     //add indirect bus
 97     //M20150601 : Rename the function for
    integrating with W5300
 98     //WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
 99     //WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x000000FF));
100     //ret =
    WIZCHIP.IF.BUS._read_byte(IDM_DR);
101     WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
102     WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x000000FF));
103     ret = WIZCHIP.IF.BUS._read_data(IDM_DR);
104
105 #else
106     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5200. !!!"
107 #endif
108
```

```
109     WIZCHIP.CS._deselect();
110     WIZCHIP_CRITICAL_EXIT();
111     return ret;
112 }
113
114
118 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
    uint8_t* pBuf, uint16_t len)
119 {
120     uint16_t i = 0;
121     WIZCHIP_CRITICAL_ENTER();
122     WIZCHIP.CS._select();
123
124 #if( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
125   WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x0000FF00) >>  8);
126   WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x000000FF) >>  0);
127
    WIZCHIP.IF.SPI._write_byte(_W5200_SPI_WRITE_ |
    ((len & 0x7F00) >> 8));         // Write data
    op code and length upper
128   WIZCHIP.IF.SPI._write_byte((len & 0x00FF)
    >> 0);          // length lower
129   for(i = 0; i < len; i++)
130     WIZCHIP.IF.SPI._write_byte(pBuf[i]);
131
132 #elif ( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_) )
133     //M20150601 : Rename the function for
    integrating with W5300
134     /*
135     WIZCHIP_WRITE(MR,WIZCHIP_READ(MR) |
    MR_AI);
136     WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
```

```
137     WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x000000FF));
138     for(i = 0 ; i < len; i++)
139
    WIZCHIP.IF.BUS._write_byte(IDM_DR,pBuf[i]);
140     WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) &
    ~MR_AI);
141     */
142     setMR(getMR() | MR_AI);
143     WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
144     WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x000000FF));
145     for(i = 0 ; i < len; i++)
146
    WIZCHIP.IF.BUS._write_data(IDM_DR,pBuf[i]);
147     WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) &
    ~MR_AI);
148 #else
149     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5200. !!!!"
150 #endif
151
152     WIZCHIP.CS._deselect();
153     WIZCHIP_CRITICAL_EXIT();
154 }
155
159 void WIZCHIP_READ_BUF (uint32_t AddrSel,
    uint8_t* pBuf, uint16_t len)
160 {
161     uint16_t i = 0;
162     WIZCHIP_CRITICAL_ENTER();
163     WIZCHIP.CS._select();
164
165 #if( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_SPI_))
166   WIZCHIP.IF.SPI._write_byte((AddrSel &
```

```
    0x0000FF00) >>  8);
  167   WIZCHIP.IF.SPI._write_byte((AddrSel &
    0x000000FF) >>  0);
  168   WIZCHIP.IF.SPI._write_byte(
    _W5200_SPI_READ_ | ((len & 0x7F00) >> 8));
    // Write data op code and length upper
  169   WIZCHIP.IF.SPI._write_byte((len & 0x00FF)
    >> 0);          // length lower
  170   for(i = 0; i < len; i++)
  171     pBuf[i] = WIZCHIP.IF.SPI._read_byte();
  172
  173 #elif ( (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_) )
  174   //M20150601 : Rename the function for
    integrating with W5300
  175   /*
  176   WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) |
    MR_AI);
  177   WIZCHIP.IF.BUS._write_byte(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
  178   WIZCHIP.IF.BUS._write_byte(IDM_AR1,
    (AddrSel & 0x000000FF));
  179   for(i = 0 ; i < len; i++)
  180     pBuf[i]   =
    WIZCHIP.IF.BUS._read_byte(IDM_DR);
  181   WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) &
    ~MR_AI);
  182   */
  183   setMR(getMR() | MR_AI);
  184   WIZCHIP.IF.BUS._write_data(IDM_AR0,
    (AddrSel & 0x0000FF00) >>  8);
  185   WIZCHIP.IF.BUS._write_data(IDM_AR1,
    (AddrSel & 0x000000FF));
  186   for(i = 0 ; i < len; i++)
  187     pBuf[i]   =
    WIZCHIP.IF.BUS._read_data(IDM_DR);
  188   setMR(getMR() & ~MR_AI);
```

```
189 #else
190    #error "Unknown _WIZCHIP_IO_MODE_ in
    W5200. !!!!"
191 #endif
192
193    WIZCHIP.CS._deselect();
194    WIZCHIP_CRITICAL_EXIT();
195 }
196
198 // Socket N regsiter IO function //
200
201 uint16_t getSn_TX_FSR(uint8_t sn)
202 {
203    uint16_t val=0,val1=0;
204    do
205    {
206       val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
207       val1 = (val1 << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
208       if (val1 != 0)
209       {
210         val = WIZCHIP_READ(Sn_TX_FSR(sn));
211         val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
212       }
213    }while (val != val1);
214    return val;
215 }
216
217
218 uint16_t getSn_RX_RSR(uint8_t sn)
219 {
220    uint16_t val=0,val1=0;
221    do
222    {
```

```
223         val1 = WIZCHIP_READ(Sn_RX_RSR(sn));
224         val1 = (val1 << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
    1));
225         if (val1 != 0)
226         {
227           val = WIZCHIP_READ(Sn_RX_RSR(sn));
228           val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
    1));
229         }
230      }while (val != val1);
231      return val;
232  }
233
235  // Sn_TXBUF & Sn_RXBUF IO function //
237
238  uint16_t getSn_RxBASE(uint8_t sn)
239  {
240      int8_t  i;
241      uint16_t rxbase = _WIZCHIP_IO_RXBUF_;
242      for(i = 0; i < sn; i++)
243         rxbase += getSn_RxMAX(i);
244      return rxbase;
245  }
246
247  uint16_t getSn_TxBASE(uint8_t sn)
248  {
249      int8_t  i;
250      uint16_t txbase = _WIZCHIP_IO_TXBUF_;
251      for(i = 0; i < sn; i++)
252         txbase += getSn_TxMAX(i);
253      return txbase;
254  }
255
268  void wiz_send_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len)
```

```
269 {
270
271   uint16_t ptr;
272   uint16_t size;
273   uint16_t dst_mask;
274   uint8_t * dst_ptr;
275
276   ptr = getSn_TX_WR(sn);
277
278
279   dst_mask = (uint32_t)ptr &
    getSn_TxMASK(sn);
280   dst_ptr = (uint8_t*)
    ((uint32_t)getSn_TxBASE(sn) + dst_mask);
281
282   if (dst_mask + len > getSn_TxMAX(sn))
283   {
284     size = getSn_TxMAX(sn) - dst_mask;
285     WIZCHIP_WRITE_BUF((uint32_t)dst_ptr,
    wizdata, size);
286     wizdata += size;
287     size = len - size;
288     dst_ptr = (uint8_t*)
    ((uint32_t)getSn_TxBASE(sn));
289     WIZCHIP_WRITE_BUF((uint32_t)dst_ptr,
    wizdata, size);
290   }
291   else
292   {
293     WIZCHIP_WRITE_BUF((uint32_t)dst_ptr,
    wizdata, len);
294   }
295
296   ptr += len;
297
298   setSn_TX_WR(sn, ptr);
299 }
```

```
300
301
312 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len)
313 {
314   uint16_t ptr;
315   uint16_t size;
316   uint16_t src_mask;
317   uint8_t * src_ptr;
318
319   ptr = getSn_RX_RD(sn);
320
321   src_mask = (uint32_t)ptr &
    getSn_RxMASK(sn);
322   src_ptr = (uint8_t *)
    ((uint32_t)getSn_RxBASE(sn) + src_mask);
323
324   if( (src_mask + len) > getSn_RxMAX(sn) )
325   {
326     size = getSn_RxMAX(sn) - src_mask;
327     WIZCHIP_READ_BUF((uint32_t)src_ptr,
    (uint8_t*)wizdata, size);
328     wizdata += size;
329     size = len - size;
330     src_ptr = (uint8_t*)
    ((uint32_t)getSn_RxBASE(sn));
331     WIZCHIP_READ_BUF((uint32_t)src_ptr,
    (uint8_t*)wizdata, size);
332   }
333   else
334   {
335     WIZCHIP_READ_BUF((uint32_t)src_ptr,
    (uint8_t*)wizdata, len);
336   }
337
338   ptr += len;
339
```

```
340     setSn_RX_RD(sn, ptr);
341 }
342
343 void wiz_recv_ignore(uint8_t sn, uint16_t
    len)
344 {
345     uint16_t ptr;
346
347     ptr = getSn_RX_RD(sn);
348
349     ptr += len;
350     setSn_RX_RD(sn,ptr);
351 }
352
353 #endif
```

Socket APIs

w5300.c

Go to the documentation of this file.

```
1  //*****************************************************************************
2  //
41 //
42 //*****************************************************************************
43
44 #include <stdint.h>
45 #include "wizchip_conf.h"
46
47 #if _WIZCHIP_ == 5300
48
49    extern uint8_t sock_remained_byte[_WIZCHIP_SOCK_NUM_];
50    extern uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_];
51
52
53 /**********************
54  * Basic I/O  Function *
55  **********************/
56
57 void WIZCHIP_WRITE(uint32_t AddrSel, uint16_t wb )
58 {
59    WIZCHIP_CRITICAL_ENTER();
```

```
60      WIZCHIP.CS._select();
61
62  #if ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_) )
63      #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
64          WIZCHIP.IF.BUS._write_data(AddrSel,
    (uint8_t)(wb>>8));
65
    WIZCHIP.IF.BUS._write_data(WIZCHIP_OFFSET_INC(
    AddrSel,1),(uint8_t)wb);
66      #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
67          WIZCHIP.IF.BUS._write_data(AddrSel,
    wb);
68      #else
69          #error "Abnoraml
    _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
70      #endif
71  #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_) )
72      #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
73          WIZCHIP.IF.BUS._write_data(IDM_AR,
    (uint8_t)(AddrSel >> 8));
74
    WIZCHIP.IF.BUS._write_data(WIZCHIP_OFFSET_INC(
    IDM_AR,1),(uint8_t)AddrSel);
75          WIZCHIP.IF.BUS._write_data(IDM_DR,
    (uint8_t)(wb>>8));
76
    WIZCHIP.IF.BUS._write_data(WIZCHIP_OFFSET_INC(
    IDM_DR,1),(uint8_t)wb);
77      #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
78          WIZCHIP.IF.BUS._write_data(IDM_AR,
    (uint16_t)AddrSel);
79          WIZCHIP.IF.BUS._write_data(IDM_DR,
    wb);
80      #else
81          #error "Abnoraml
```

```
    _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
 82     #endif
 83 #else
 84     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5300. !!!"
 85 #endif
 86
 87     WIZCHIP.CS._deselect();
 88     WIZCHIP_CRITICAL_EXIT();
 89 }
 90
 91 uint16_t WIZCHIP_READ(uint32_t AddrSel)
 92 {
 93     uint16_t ret;
 94
 95     WIZCHIP_CRITICAL_ENTER();
 96     WIZCHIP.CS._select();
 97
 98 #if ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_DIR_) )
 99     #if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
100         ret =
    (((uint16_t)WIZCHIP.IF.BUS._read_data(AddrSel)
    ) << 8) |
101
    (((uint16_t)WIZCHIP.IF.BUS._read_data(WIZCHIP_
    OFFSET_INC(AddrSel,1))) & 0x00FF) ;
102     #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
103         ret =
    WIZCHIP.IF.BUS._read_data(AddrSel);
104     #else
105         #error "Abnoraml
    _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
106     #endif
107 #elif ( (_WIZCHIP_IO_MODE_ ==
    _WIZCHIP_IO_MODE_BUS_INDIR_) )
108     #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
```

```
109         WIZCHIP.IF.BUS._write_data(IDM_AR,
    (uint8_t)(AddrSel >> 8));
110
    WIZCHIP.IF.BUS._write_data(WIZCHIP_OFFSET_INC(
    IDM_AR,1),(uint8_t)AddrSel);
111         ret =
    (((uint16_t)WIZCHIP.IF.BUS._read_data(IDM_DR))
    << 8) |
112
    (((uint16_t)WIZCHIP.IF.BUS._read_data(WIZCHIP_
    OFFSET_INC(IDM_DR,1))) & 0x00FF);
113     #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
114         WIZCHIP.IF.BUS._write_data(IDM_AR,
    (uint16_t)AddrSel);
115         ret =
    WIZCHIP.IF.BUS._read_data(IDM_DR);
116     #else
117         #error "Abnoraml
    _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
118     #endif
119 #else
120     #error "Unknown _WIZCHIP_IO_MODE_ in
    W5300. !!!"
121 #endif
122
123     WIZCHIP.CS._deselect();
124     WIZCHIP_CRITICAL_EXIT();
125     return ret;
126 }
127
128
129 void setTMSR(uint8_t sn,uint8_t tmsr)
130 {
131     uint16_t tmem;
132     tmem =
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn &
    0xFE)));
```

```
133     if(sn & 0x01)  tmem = (tmem & 0xFF00) |
    (((uint16_t)tmsr ) & 0x00FF) ;
134     else tmem =  (tmem & 0x00FF) |
    (((uint16_t)tmsr) << 8) ;
135     WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(TMS01R,
    (sn & 0xFE)),tmem);
136  }
137
138  uint8_t getTMSR(uint8_t sn)
139  {
140     if(sn & 0x01)
141        return (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn &
    0xFE))) & 0x00FF);
142     return (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn &
    0xFE))) >> 8);
143  }
144
145  void setRMSR(uint8_t sn,uint8_t rmsr)
146  {
147     uint16_t rmem;
148     rmem =
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(RMS01R, (sn &
    0xFE)));
149     if(sn & 0x01)  rmem = (rmem & 0xFF00) |
    (((uint16_t)rmsr ) & 0x00FF) ;
150     else rmem =  (rmem & 0x00FF) |
    (((uint16_t)rmsr) << 8) ;
151     WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(RMS01R,
    (sn & 0xFE)),rmem);
152  }
153
154  uint8_t getRMSR(uint8_t sn)
155  {
156     if(sn & 0x01)
157        return (uint8_t)
```

```
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(RMS01R, (sn &
    0xFE))) & 0x00FF);
158     return (uint8_t)
    (WIZCHIP_READ(WIZCHIP_OFFSET_INC(RMS01R, (sn &
    0xFE))) >> 8);
159 }
160
161 uint32_t getSn_TX_FSR(uint8_t sn)
162 {
163     uint32_t free_tx_size=0;
164     uint32_t free_tx_size1=1;
165     while(1)
166     {
167         free_tx_size =
    (((uint32_t)WIZCHIP_READ(Sn_TX_FSR(sn))) <<
    16) |
168
    (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn
    _TX_FSR(sn),2))) & 0x0000FFFF);
    // read
169         if(free_tx_size == free_tx_size1)
    break;  // if first == sencond, Sn_TX_FSR
    value is valid.
170         free_tx_size1 = free_tx_size;
    // save second value into first
171     }
172     return free_tx_size;
173 }
174
175 uint32_t getSn_RX_RSR(uint8_t sn)
176 {
177     uint32_t received_rx_size=0;
178     uint32_t received_rx_size1=1;
179     while(1)
180     {
181         received_rx_size =
    (((uint32_t)WIZCHIP_READ(Sn_RX_RSR(sn))) <<
```

```
    16) |
182
    (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn
    _RX_RSR(sn),2))) & 0x0000FFFF);
183         if(received_rx_size ==
    received_rx_size1) break;
184         received_rx_size1 = received_rx_size;
    // if first == sencond, Sn_RX_RSR value is
    valid.
185     }
    // save second value into first
186     return received_rx_size + (uint32_t)
    ((sock_pack_info[sn] & 0x02) ? 1 : 0);
187 }
188
189
190 void wiz_send_data(uint8_t sn, uint8_t
    *wizdata, uint32_t len)
191 {
192     uint32_t i = 0;
193     if(len == 0)  return;
194
195     for(i = 0; i < len ; i += 2)
196         setSn_TX_FIFOR(sn,
    (((uint16_t)wizdata[i]) << 8) |
    (((uint16_t)wizdata[i+1]) & 0x00FF))
197 }
198
199 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint32_t len)
200 {
201     uint16_t rd = 0;
202     uint32_t i = 0;
203
204     if(len == 0) return;
205
206     for(i = 0; i < len; i++)
```

```
207     {
208        if((i & 0x01)==0)
209        {
210           rd = getSn_RX_FIFOR(sn);
211           wizdata[i]   = (uint8_t)(rd >> 8);
212        }
213        else  wizdata[i] = (uint8_t)rd;  //
   For checking the memory access violation
214     }
215     sock_remained_byte[sn] = (uint8_t)rd; //
   back up the remaind fifo byte.
216  }
217
218  void wiz_recv_ignore(uint8_t sn, uint32_t
   len)
219  {
220     uint32_t i = 0;
221     for(i = 0; i < len ; i += 2)
   getSn_RX_FIFOR(sn);
222  }
223
224
225  #endif
```

Socket APIs

w5500.c

Go to the documentation of this file.

```
    1 //*****************************************************************************
    2 //
   52 //
   53 //*****************************************************************************
   54 //#include <stdio.h>
   55 #include "w5500.h"
   56
   57 #define _W5500_SPI_VDM_OP_           0x00
   58 #define _W5500_SPI_FDM_OP_LEN1_      0x01
   59 #define _W5500_SPI_FDM_OP_LEN2_      0x02
   60 #define _W5500_SPI_FDM_OP_LEN4_      0x03
   61
   62 #if   (_WIZCHIP_ == 5500)
   63
   65 uint8_t  WIZCHIP_READ(uint32_t AddrSel)
   66 {
   67    uint8_t ret;
   68    uint8_t spi_data[3];
   69
   70    WIZCHIP_CRITICAL_ENTER();
   71    WIZCHIP.CS._select();
   72
   73    AddrSel |= (_W5500_SPI_READ_ | _W5500_SPI_VDM_OP_);
```

```
74
75      if(!WIZCHIP.IF.SPI._read_burst ||
        !WIZCHIP.IF.SPI._write_burst)  // byte
        operation
76      {
77          WIZCHIP.IF.SPI._write_byte((AddrSel &
        0x00FF0000) >> 16);
78          WIZCHIP.IF.SPI._write_byte((AddrSel
        & 0x0000FF00) >>  8);
79          WIZCHIP.IF.SPI._write_byte((AddrSel
        & 0x000000FF) >>  0);
80      }
81      else // burst operation
82      {
83          spi_data[0] = (AddrSel & 0x00FF0000)
        >> 16;
84          spi_data[1] = (AddrSel & 0x0000FF00)
        >> 8;
85          spi_data[2] = (AddrSel & 0x000000FF)
        >> 0;
86
        WIZCHIP.IF.SPI._write_burst(spi_data, 3);
87      }
88      ret = WIZCHIP.IF.SPI._read_byte();
89
90      WIZCHIP.CS._deselect();
91      WIZCHIP_CRITICAL_EXIT();
92      return ret;
93  }
94
95  void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t
    wb )
96  {
97      uint8_t spi_data[4];
98
99      WIZCHIP_CRITICAL_ENTER();
100     WIZCHIP.CS._select();
```

```
101
102     AddrSel |= (_W5500_SPI_WRITE_ | _W5500_SPI_VDM_OP_);
103
104     //if(!WIZCHIP.IF.SPI._read_burst || !WIZCHIP.IF.SPI._write_burst)    // byte operation
105     if(!WIZCHIP.IF.SPI._write_burst)     // byte operation
106     {
107         WIZCHIP.IF.SPI._write_byte((AddrSel & 0x00FF0000) >> 16);
108         WIZCHIP.IF.SPI._write_byte((AddrSel & 0x0000FF00) >>  8);
109         WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF) >>  0);
110         WIZCHIP.IF.SPI._write_byte(wb);
111     }
112     else // burst operation
113     {
114         spi_data[0] = (AddrSel & 0x00FF0000) >> 16;
115         spi_data[1] = (AddrSel & 0x0000FF00) >> 8;
116         spi_data[2] = (AddrSel & 0x000000FF) >> 0;
117         spi_data[3] = wb;
118
    WIZCHIP.IF.SPI._write_burst(spi_data, 4);
119     }
120
121     WIZCHIP.CS._deselect();
122     WIZCHIP_CRITICAL_EXIT();
123 }
124
125 void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len)
```

```
126 {
127     uint8_t spi_data[3];
128     uint16_t i;
129
130     WIZCHIP_CRITICAL_ENTER();
131     WIZCHIP.CS._select();
132
133     AddrSel |= (_W5500_SPI_READ_ |
    _W5500_SPI_VDM_OP_);
134
135     if(!WIZCHIP.IF.SPI._read_burst ||
    !WIZCHIP.IF.SPI._write_burst)  // byte
    operation
136     {
137          WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x00FF0000) >> 16);
138          WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x0000FF00) >>  8);
139          WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x000000FF) >>  0);
140          for(i = 0; i < len; i++)
141             pBuf[i] =
    WIZCHIP.IF.SPI._read_byte();
142     }
143     else // burst operation
144     {
145          spi_data[0] = (AddrSel & 0x00FF0000)
    >> 16;
146          spi_data[1] = (AddrSel & 0x0000FF00)
    >> 8;
147          spi_data[2] = (AddrSel & 0x000000FF)
    >> 0;
148
    WIZCHIP.IF.SPI._write_burst(spi_data, 3);
149          WIZCHIP.IF.SPI._read_burst(pBuf,
    len);
150     }
```

```
151
152     WIZCHIP.CS._deselect();
153     WIZCHIP_CRITICAL_EXIT();
154 }
155
156 void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
    uint8_t* pBuf, uint16_t len)
157 {
158     uint8_t spi_data[3];
159     uint16_t i;
160
161     WIZCHIP_CRITICAL_ENTER();
162     WIZCHIP.CS._select();
163
164     AddrSel |= (_W5500_SPI_WRITE_ |
    _W5500_SPI_VDM_OP_);
165
166     if(!WIZCHIP.IF.SPI._write_burst)     //
    byte operation
167     {
168         WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x00FF0000) >> 16);
169         WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x0000FF00) >>  8);
170         WIZCHIP.IF.SPI._write_byte((AddrSel
    & 0x000000FF) >>  0);
171         for(i = 0; i < len; i++)
172
    WIZCHIP.IF.SPI._write_byte(pBuf[i]);
173     }
174     else // burst operation
175     {
176         spi_data[0] = (AddrSel & 0x00FF0000)
    >> 16;
177         spi_data[1] = (AddrSel & 0x0000FF00)
    >> 8;
178         spi_data[2] = (AddrSel & 0x000000FF)
```

```
    >> 0;
179
    WIZCHIP.IF.SPI._write_burst(spi_data, 3);
180         WIZCHIP.IF.SPI._write_burst(pBuf,
    len);
181     }
182
183     WIZCHIP.CS._deselect();
184     WIZCHIP_CRITICAL_EXIT();
185 }
186
187
188 uint16_t getSn_TX_FSR(uint8_t sn)
189 {
190     uint16_t val=0,val1=0;
191
192     do
193     {
194         val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
195         val1 = (val1 << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
196         if (val1 != 0)
197         {
198           val = WIZCHIP_READ(Sn_TX_FSR(sn));
199           val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn),
    1));
200         }
201     }while (val != val1);
202     return val;
203 }
204
205
206 uint16_t getSn_RX_RSR(uint8_t sn)
207 {
208     uint16_t val=0,val1=0;
```

```
209
210     do
211     {
212        val1 = WIZCHIP_READ(Sn_RX_RSR(sn));
213        val1 = (val1 << 8) +
   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
   1));
214        if (val1 != 0)
215        {
216          val = WIZCHIP_READ(Sn_RX_RSR(sn));
217          val = (val << 8) +
   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn),
   1));
218        }
219     }while (val != val1);
220     return val;
221 }
222
223 void wiz_send_data(uint8_t sn, uint8_t
   *wizdata, uint16_t len)
224 {
225     uint16_t ptr = 0;
226     uint32_t addrsel = 0;
227
228     if(len == 0)  return;
229     ptr = getSn_TX_WR(sn);
230     //M20140501 : implict type casting ->
   explict type casting
231     //addrsel = (ptr << 8) +
   (WIZCHIP_TXBUF_BLOCK(sn) << 3);
232     addrsel = ((uint32_t)ptr << 8) +
   (WIZCHIP_TXBUF_BLOCK(sn) << 3);
233     //
234     WIZCHIP_WRITE_BUF(addrsel,wizdata, len);
235
236     ptr += len;
237     setSn_TX_WR(sn,ptr);
```

```
238 }
239
240 void wiz_recv_data(uint8_t sn, uint8_t
    *wizdata, uint16_t len)
241 {
242    uint16_t ptr = 0;
243    uint32_t addrsel = 0;
244
245    if(len == 0) return;
246    ptr = getSn_RX_RD(sn);
247    //M20140501 : implict type casting ->
    explict type casting
248    //addrsel = ((ptr << 8) +
    (WIZCHIP_RXBUF_BLOCK(sn) << 3);
249    addrsel = ((uint32_t)ptr << 8) +
    (WIZCHIP_RXBUF_BLOCK(sn) << 3);
250    //
251    WIZCHIP_READ_BUF(addrsel, wizdata, len);
252    ptr += len;
253
254    setSn_RX_RD(sn,ptr);
255 }
256
257
258 void wiz_recv_ignore(uint8_t sn, uint16_t
    len)
259 {
260    uint16_t ptr = 0;
261
262    ptr = getSn_RX_RD(sn);
263    ptr += len;
264    setSn_RX_RD(sn,ptr);
265 }
266
267 #endif
```