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_cbffunc

(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.

```c
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.

```c
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.

```c
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.

```c
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:

<table>
<thead>
<tr>
<th>#</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WIZnet socket APIs</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>WIZnet Extra Functions</td>
<td>These functions is optional function. It could be replaced at WIZCHIP I/O function because they were made by WIZCHIP I/O functions.</td>
</tr>
</tbody>
</table>

### 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**
<table>
<thead>
<tr>
<th>Socket register access functions</th>
<th>These are functions to access <strong>socket registers</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ WIZCHIP register</td>
<td>WIZCHIP register defines register group of <strong>W5100</strong></td>
</tr>
<tr>
<td>Common register</td>
<td>Common register group It set the basic for the networking It set the configuration such as interrupt, network information, ICMP, etc</td>
</tr>
<tr>
<td>Socket register</td>
<td>Socket register group Socket register configures and control SOCKETn which is necessary to data communication</td>
</tr>
<tr>
<td>▼ W5200</td>
<td>WHIZCHIP register defines and I/O functions of <strong>W5200</strong></td>
</tr>
<tr>
<td>▼ WIZCHIP I/O functions</td>
<td>This supports the basic I/O functions for <strong>WIZCHIP register</strong></td>
</tr>
<tr>
<td>Basic I/O function</td>
<td>These are basic input/output functions to read values from register or write values to register</td>
</tr>
<tr>
<td>Common register access functions</td>
<td>These are functions to access <strong>common registers</strong></td>
</tr>
<tr>
<td>Socket register access functions</td>
<td>These are functions to access <strong>socket registers</strong></td>
</tr>
<tr>
<td>▼ WIZCHIP register</td>
<td>WIZCHIP register defines register group of <strong>W5200</strong></td>
</tr>
<tr>
<td>Common register</td>
<td>Common register group It set the basic for the networking</td>
</tr>
<tr>
<td><strong>Socket register</strong></td>
<td>It set the configuration such as interrupt, network information, ICMP, etc.</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Socket register group</strong></td>
<td>Socket register configures and control SOCKETn which is necessary to data communication</td>
</tr>
<tr>
<td><strong>W5300</strong></td>
<td>WHIZCHIP register defines and I/O functions of <strong>W5300</strong></td>
</tr>
<tr>
<td><strong>WIZCHIP I/O functions</strong></td>
<td>This supports the basic I/O functions for <strong>WIZCHIP register</strong></td>
</tr>
<tr>
<td><strong>Basic I/O function</strong></td>
<td>These are basic input/output functions to read values from register or write values to register</td>
</tr>
<tr>
<td><strong>Common register access functions</strong></td>
<td>These are functions to access <strong>common registers</strong></td>
</tr>
<tr>
<td><strong>Socket register access functions</strong></td>
<td>These are functions to access <strong>socket registers</strong></td>
</tr>
<tr>
<td><strong>WIZCHIP register</strong></td>
<td>WHIZCHIP register defines register group of <strong>W5300</strong></td>
</tr>
<tr>
<td><strong>Common register</strong></td>
<td>Common register group. It set the configuration such as interrupt, network information, ICMP, etc</td>
</tr>
<tr>
<td><strong>Socket register</strong></td>
<td>Socket register group. Socket register</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W5500</td>
<td>WHIZCHIP register defines and I/O functions of W5500</td>
</tr>
<tr>
<td>WIZCHIP I/O functions</td>
<td>This supports the basic I/O functions for WIZCHIP register</td>
</tr>
<tr>
<td>Basic I/O function</td>
<td>These are basic input/output functions to read values from register or write values to register</td>
</tr>
<tr>
<td>Common register access functions</td>
<td>These are functions to access common registers</td>
</tr>
<tr>
<td>Socket register access functions</td>
<td>These are functions to access socket registers</td>
</tr>
<tr>
<td>WIZCHIP register</td>
<td>WHIZCHIP register defines register group of W5500</td>
</tr>
</tbody>
</table>
| 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          |
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8_t socket</td>
<td>(uint8_t sn, uint8_t protocol, uint16_t port, uint8_t flag) Open a socket.</td>
<td></td>
</tr>
<tr>
<td>int8_t close</td>
<td>(uint8_t sn) Close a socket.</td>
<td></td>
</tr>
<tr>
<td>int8_t listen</td>
<td>(uint8_t sn) Listen to a connection request from a client.</td>
<td></td>
</tr>
<tr>
<td>int8_t connect</td>
<td>(uint8_t sn, uint8_t *addr, uint16_t port) Try to connect a server.</td>
<td></td>
</tr>
<tr>
<td>int8_t disconnect</td>
<td>(uint8_t sn) Try to disconnect a connection socket.</td>
<td></td>
</tr>
<tr>
<td>int32_t send</td>
<td>(uint8_t sn, uint8_t *buf, uint16_t len) Send data to the connected peer in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TCP socket.</td>
<td></td>
</tr>
<tr>
<td>int32_t recv</td>
<td>(uint8_t sn, uint8_t *buf, uint16_t len) Receive data from the connected peer.</td>
<td></td>
</tr>
<tr>
<td>int32_t sendto</td>
<td>(uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t port) Sends</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datagram to the peer with destination IP address and port number passed as parameter.</td>
<td></td>
</tr>
<tr>
<td>int32_t recvfrom</td>
<td>(uint8_t sn, uint8_t *buf, uint16_t len, uint8_t *addr, uint16_t *port) Receive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datagram of UDP or MACRAW.</td>
<td></td>
</tr>
<tr>
<td>int8_t ctlsocket</td>
<td>(uint8_t sn, ctlsock_type cstype, void *arg) Control socket.</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>API</th>
<th>WIZnet</th>
<th>Berkeley</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>bind()</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>listen()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>connect()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>accept()</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>recv()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>send()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>recvfrom()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>sendto()</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>closesocket()</td>
<td>O</td>
<td>close() &amp; disconnect()</td>
</tr>
</tbody>
</table>

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.
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>

Server Socket 1

socket()

listen()

send(), recv(),

close()

Client Socket 1

socket()

connect()

send(), recv(),

close()
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 SFETHER_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.
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

int8_t listen ( uint8_t sn )

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


```c
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.


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.


```c
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.


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 \textit{len} in socket buffer. In non-block io mode, it return \textbf{SOCK_BUSY} immediately when \textit{len} is greater than data size in socket buffer.

\textbf{Parameters}

\begin{itemize}
  \item \textbf{sn} Socket number. It should be 0 ~ \texttt{WIZCHIP\_SOCK\_NUM}.
  \item \textbf{buf} Pointer buffer to read incoming data.
  \item \textbf{len} The max data length of data in buf.
\end{itemize}

\textbf{Returns}

\begin{itemize}
  \item \textbf{Success} : The real received data size
  \item \textbf{Fail} :
    \begin{itemize}
      \item \texttt{SOCKERR\_SOCKSTATUS} - Invalid socket status for socket operation
      \item \texttt{SOCKERR\_SOCKMODE} - Invalid operation in the socket
      \item \texttt{SOCKERR\_SOCKNUM} - Invalid socket number
      \item \texttt{SOCKERR\_DATALEN} - zero data length
      \item \textbf{SOCK\_BUSY} - Socket is busy.
    \end{itemize}
\end{itemize}

Definition at line \texttt{387} of file \texttt{socket.c}.

\textbf{References} \texttt{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().}

\begin{verbatim}
int32_t sendto ( uint8_t   sn,
    uint8_t *  buf,
    uint16_t   len,
    uint8_t *  addr,
    uint16_t   port
)
\end{verbatim}
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**.

Referenced by close().

```c
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 recvfrom 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.


```c
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**

<table>
<thead>
<tr>
<th>sn</th>
<th>socket number</th>
</tr>
</thead>
<tbody>
<tr>
<td>cstype</td>
<td>type of control socket. refer to <code>ctlsock_type</code>.</td>
</tr>
<tr>
<td>arg</td>
<td>Data type and value is determined according to <code>ctlsock_type</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>cstype</code></th>
<th><code>data type</code></th>
<th><code>value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_SET_IOMODE</td>
<td>uint8_t</td>
<td>SOCK_IO_BLOCK</td>
</tr>
<tr>
<td>CS_GET_IOMODE</td>
<td>uint8_t</td>
<td>SOCK_IO_NONBLOCK</td>
</tr>
<tr>
<td>CS_GET_MAXTXBUF</td>
<td>uint16_t</td>
<td>0 ~ 16K</td>
</tr>
<tr>
<td>CS_GET_MAXRXBUF</td>
<td>uint16_t</td>
<td>0 ~ 16K</td>
</tr>
<tr>
<td>CS_CLR_INTERRUPT</td>
<td>sockint_kind</td>
<td>SIK_CONNECTED</td>
</tr>
<tr>
<td>CS_GET_INTERRUPT</td>
<td>sockint_kind</td>
<td>SIK_CONNECTED</td>
</tr>
<tr>
<td>CS_SET_INTMASK</td>
<td>sockint_kind</td>
<td>SIK_CONNECTED</td>
</tr>
<tr>
<td>CS_GET_INTMASK</td>
<td>sockint_kind</td>
<td>SIK_CONNECTED</td>
</tr>
</tbody>
</table>

**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`.

```c
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`.

<table>
<thead>
<tr>
<th>sotype</th>
<th>data type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_TTL</td>
<td>uint8_t</td>
<td>0 ~ 255</td>
</tr>
<tr>
<td>SO_TOS</td>
<td>uint8_t</td>
<td>0 ~ 255</td>
</tr>
<tr>
<td>SO_MSS</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_DESTIP</td>
<td>uint8_t[4]</td>
<td></td>
</tr>
<tr>
<td>SO_DESTPORT</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_KEEPALIVESEND</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>SO_KEEPALIVEAUTO</td>
<td>uint8_t</td>
<td>0 ~ 255</td>
</tr>
</tbody>
</table>

**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`.

**SOCKERR_SOCKOPT**, and **SOCKERR_TIMEOUT**.

```c
int8_t getsockopt ( uint8_t sn, socket_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`.

<table>
<thead>
<tr>
<th>sotype</th>
<th>data type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_FLAG</td>
<td>uint8_t</td>
<td>SF_ETHER_OWN, etc...</td>
</tr>
<tr>
<td>SO_TOS</td>
<td>uint8_t</td>
<td>0 ~ 255</td>
</tr>
<tr>
<td>SO_MSS</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_DESTIP</td>
<td>uint8_t[4]</td>
<td></td>
</tr>
<tr>
<td>SO_DESTPORT</td>
<td>uint16_t</td>
<td></td>
</tr>
<tr>
<td>SO_KEEPALIVEAUTO</td>
<td>uint8_t</td>
<td>0 ~ 255</td>
</tr>
<tr>
<td>SO.SendBuf</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_RECVBUF</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_STATUS</td>
<td>uint8_t</td>
<td>SOCK_ESTABLISHED, etc...</td>
</tr>
<tr>
<td>SO_REMAINSIZE</td>
<td>uint16_t</td>
<td>0 ~ 65535</td>
</tr>
<tr>
<td>SO_PACKINFO</td>
<td>uint8_t</td>
<td>PACK_FIRST, etc...</td>
</tr>
</tbody>
</table>

**Returns**

- Success: **SOCK_OK**
• Fail
  o **SOCKERR_SOCKNUM** - Invalid Socket number
  o **SOCKERR_SOCKOPT** - Invalid socket option or its value
  o **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 Header 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int8_t ctlwizchip (ctlwizchip_type cwtype, void *arg)</code></td>
<td>Controls to the WIZCHIP. More...</td>
</tr>
<tr>
<td><code>int8_t ctlnetwork (ctlnetwork_type cnype, void *arg)</code></td>
<td>Controls to network. More...</td>
</tr>
<tr>
<td><code>void wizchip_sw_reset (void)</code></td>
<td>Reset WIZCHIP by softly. More...</td>
</tr>
<tr>
<td><code>int8_t wizchip_init (uint8_t *txsize, uint8_t *rxsize)</code></td>
<td>Initializes WIZCHIP with socket buffer size. More...</td>
</tr>
<tr>
<td><code>void wizchip_clrinterrupt (intr_kind intr)</code></td>
<td>Clear Interrupt of WIZCHIP. More...</td>
</tr>
<tr>
<td><code>intr_kind wizchip_getinterrupt (void)</code></td>
<td>Get Interrupt of WIZCHIP. More...</td>
</tr>
<tr>
<td><code>void wizchip_setinterruptmask (intr_kind intr)</code></td>
<td>Mask or Unmask Interrupt of WIZCHIP. More...</td>
</tr>
<tr>
<td><code>intr_kind wizchip_getinterruptmask (void)</code></td>
<td>Get Interrupt mask of WIZCHIP. More...</td>
</tr>
<tr>
<td><code>void wizphy_setphyconf (wiz_PhyConf *phyconf)</code></td>
<td>Set the phy information for WIZCHIP without power mode. More...</td>
</tr>
<tr>
<td><code>void wizphy_getphyconf (wiz_PhyConf *phyconf)</code></td>
<td>Get phy configuration information. More...</td>
</tr>
<tr>
<td><code>void wizphy_getphystat (wiz_PhyConf *phyconf)</code></td>
<td>Get phy status. More...</td>
</tr>
</tbody>
</table>
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

```c
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()`.

```c
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 : success
- -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()`.

```c
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 value of intr_kind. It can type-cast to uint16_t.

Definition at line 546 of file wizchip_conf.c.
References \texttt{getIMR}, and \texttt{getSIMR}.

Referenced by \texttt{ctlwizchip()}. 

\begin{verbatim}
void wizphy_setphyconf ( wiz_PhyConf * phyconf )
\end{verbatim}

Set the phy information for WIZCHIP without power mode.

\textbf{Parameters}

\texttt{phyconf} : \texttt{wiz\_PhyConf}

Definition at line 627 of file \texttt{wizchip\_conf.c}.

References \texttt{wiz\_PhyConf\_t::by}, \texttt{wiz\_PhyConf\_t::duplex}, \texttt{wiz\_PhyConf\_t::mode}, \texttt{PHY\_CONFBY\_SW}, \texttt{PHY\_DUPLEX\_FULL}, \texttt{PHY\_MODE\_AUTONEGO}, \texttt{PHY\_SPEED\_100}, \texttt{PHYCFGR\_OPMD}, \texttt{PHYCFGR\_OPMDC\_100F}, \texttt{PHYCFGR\_OPMDC\_100H}, \texttt{PHYCFGR\_OPMDC\_10F}, \texttt{PHYCFGR\_OPMDC\_10H}, \texttt{PHYCFGR\_OPMDC\_ALLA}, \texttt{setPHYCFGR}, \texttt{wiz\_PhyConf\_t::speed}, and \texttt{wizphy\_reset()}. 

Referenced by \texttt{ctlwizchip()}. 

\begin{verbatim}
void wizphy_getphyconf ( wiz_PhyConf * phyconf )
\end{verbatim}

Get phy configuration information.

\textbf{Parameters}

\texttt{phyconf} : \texttt{wiz\_PhyConf}

Definition at line 657 of file \texttt{wizchip\_conf.c}.

References \texttt{wiz\_PhyConf\_t::by}, \texttt{wiz\_PhyConf\_t::duplex}, \texttt{getPHYCFGR}, \texttt{wiz\_PhyConf\_t::mode}, \texttt{PHY\_CONFBY\_HW}, \texttt{PHY\_CONFBY\_SW}, \texttt{PHY\_DUPLEX\_FULL}, \texttt{PHY\_DUPLEX\_HALF}, \texttt{PHY\_MODE\_AUTONEGO}, \texttt{PHY\_MODE\_MANUAL}, \texttt{PHY\_SPEED\_10}, \texttt{PHY\_SPEED\_100}, \texttt{PHYCFGR\_OPMD},
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 Settg value of power down mode.

Definition at line 703 of file wizchip_conf.c.

References getPHYCFGR, PHY_POWER_DOWN, PHYCFGR_OPM, 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()`.

```c
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()`.

```c
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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

**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

```c
typedef struct __WIZCHIP __WIZCHIP
    The set of callback functions for W5500:WIZCHIP I/O functions
    W5200:WIZCHIP I/O functions
    More...
```

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Structure Name</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef</td>
<td>wiz_PhyConf_t</td>
<td>wiz_PhyConf</td>
</tr>
<tr>
<td>typedef</td>
<td>wiz_NetInfo_t</td>
<td>wiz_NetInfo</td>
</tr>
<tr>
<td>typedef</td>
<td>wiz_NetTimeout_t</td>
<td>wiz_NetTimeout</td>
</tr>
</tbody>
</table>
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 configuration.
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()

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIK_CONNECTED</td>
<td>connected</td>
</tr>
<tr>
<td>SIK_DISCONNECTED</td>
<td>disconnected</td>
</tr>
<tr>
<td>SIK_RECEIVED</td>
<td>data received</td>
</tr>
<tr>
<td>SIK_TIMEOUT</td>
<td>timeout occurred</td>
</tr>
<tr>
<td>SIK_SENT</td>
<td>send ok</td>
</tr>
<tr>
<td>SIK_ALL</td>
<td>all interrupt</td>
</tr>
</tbody>
</table>

Definition at line 345 of file socket.h.

**enum ctlsock_type**

The type of ctlsocket().

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_SET_IOMODE</td>
<td>set socket IO mode with SOCK_IO_BLOCK or SOCK_IO_NONBLOCK</td>
</tr>
<tr>
<td>CS_GET_IOMODE</td>
<td>get socket IO mode</td>
</tr>
<tr>
<td>CS_GET_MAXTXBUF</td>
<td>get the size of socket buffer allocated in TX memory</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CS_GET_MAXRXBUF</td>
<td>get the size of socket buffer allocated in RX memory</td>
</tr>
<tr>
<td>CS_CLR_INTERRUPT</td>
<td>clear the interrupt of socket with sockint_kind</td>
</tr>
<tr>
<td>CS_GET_INTERRUPT</td>
<td>get the socket interrupt. refer to sockint_kind</td>
</tr>
<tr>
<td>CS_SET_INTMASK</td>
<td>set the interrupt mask of socket with sockint_kind, Not supported in W5100</td>
</tr>
<tr>
<td>CS_GET_INTMASK</td>
<td>get the masked interrupt of socket. refer to sockint_kind, Not supported in W5100</td>
</tr>
</tbody>
</table>

Definition at line 361 of file socket.h.

enum sockopt_type

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

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_FLAG</td>
<td>Valid only in getsockopt(), For set flag of socket refer to flag in socket().</td>
</tr>
<tr>
<td>SO_TTL</td>
<td>Set TTL. Sn_TTL ( setSn_TTL(), getSn_TTL() )</td>
</tr>
<tr>
<td>SO_TOS</td>
<td>Set TOS. Sn_TOS ( setSn_TOS(), getSn_TOS() )</td>
</tr>
<tr>
<td>SO_MSS</td>
<td>Set MSS. Sn_MSSR ( setSn_MSSR(), getSn_MSSR() )</td>
</tr>
<tr>
<td>SO_DESTIP</td>
<td>Set the destination IP address. Sn_DIPR ( setSn_DIPR(), getSn_DIPR() )</td>
</tr>
<tr>
<td>SO_DESTPORT</td>
<td>Set the destination Port number. Sn_DPORT ( setSn_DPORT(), getSn_DPORT() )</td>
</tr>
<tr>
<td>SO_KEEPALIVESEND</td>
<td>Valid only in setsockopt. Manually send keep-alive packet in TCP mode, Not</td>
</tr>
</tbody>
</table>
supported in W5100.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_KEEPALIVEAUTO</td>
<td>Set/Get keep-alive auto transmission timer in TCP mode, Not supported in W5100, W5200.</td>
</tr>
<tr>
<td>SO_SEND_BUF</td>
<td>Valid only in getsockopt. Get the free data size of socket TX buffer. <strong>Sn_TX_FSR, getSn_TX_FSR()</strong></td>
</tr>
<tr>
<td>SO_RECV_BUF</td>
<td>Valid only in getsockopt. Get the received data size in socket RX buffer. <strong>Sn_RX_RSR, getSn_RX_RSR()</strong></td>
</tr>
<tr>
<td>SO_STATUS</td>
<td>Valid only in getsockopt. Get the socket status. <strong>Sn_SR, getSn_SR()</strong></td>
</tr>
<tr>
<td>SO_REMAIN_SIZE</td>
<td>Valid only in getsockopt. Get the remained packet size in other then TCP mode.</td>
</tr>
<tr>
<td>SO_PACKINFO</td>
<td>Valid only in getsockopt. Get the packet information as <strong>PACK_FIRST</strong>, <strong>PACK_REMAINED</strong>, and <strong>PACK_COMPLETED</strong> in other then TCP mode.</td>
</tr>
</tbody>
</table>

Definition at line 380 of file socket.h.

**enum ctlwizchip_type**

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

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW_RESET_WIZCHIP</td>
<td>Resets WIZCHIP by softly.</td>
</tr>
<tr>
<td>CW_INIT_WIZCHIP</td>
<td>Initializes to WIZCHIP with SOCKET buffer size 2 or 1 dimension array typed uint8_t.</td>
</tr>
<tr>
<td>CW_GET_INTERRUPT</td>
<td>Get Interrupt status of WIZCHIP.</td>
</tr>
<tr>
<td>CW_CLR_INTERRUPT</td>
<td>Clears interrupt.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CW_SET_INTRMASK</td>
<td>Masks interrupt.</td>
</tr>
<tr>
<td>CW_GET_INTRMASK</td>
<td>Get interrupt mask.</td>
</tr>
<tr>
<td>CW_SET_INTRTIME</td>
<td>Set interval time between the current and next interrupt.</td>
</tr>
<tr>
<td>CW_GET_INTRTIME</td>
<td>Set interval time between the current and next interrupt.</td>
</tr>
<tr>
<td>CW_GET_ID</td>
<td>Gets WIZCHIP name.</td>
</tr>
<tr>
<td>CW_RESET_PHY</td>
<td>Resets internal PHY. Valid Only W5500.</td>
</tr>
<tr>
<td>CW_SET_PHYCONF</td>
<td>When PHY configured by internal register, PHY operation mode (Manual/Auto, 10/100, Half/Full). Valid Only W5000.</td>
</tr>
<tr>
<td>CW_GET_PHYCONF</td>
<td>Get PHY operation mode in internal register. Valid Only W5500.</td>
</tr>
<tr>
<td>CW_GET_PHYSTATUS</td>
<td>Get real PHY status on operating. Valid Only W5500.</td>
</tr>
<tr>
<td>CW_SET_PHYPOWMODE</td>
<td>Set PHY power mode as normal and down when PHYSTATUS.OPMD == 1. Valid Only W5500.</td>
</tr>
<tr>
<td>CW_GET_PHYPOWMODE</td>
<td>Get PHY Power mode as down or normal, Valid Only W5100, W5200.</td>
</tr>
<tr>
<td>CW_GET_PHYLINK</td>
<td>Get PHY Link status, Valid Only W5100, W5200.</td>
</tr>
</tbody>
</table>

Definition at line 262 of file wizchip_conf.h.

**enum ctlnetwork_type**

Network control type enumeration used in `ctlnetwork()`.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN_SET_NETINFO</td>
<td>Set Network with <code>wiz_NetInfo</code>.</td>
</tr>
<tr>
<td>CN_GET_NETINFO</td>
<td>Get Network with <code>wiz_NetInfo</code>.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CN_SET_NETMODE</td>
<td>Set network mode as WOL, PPPoE, Ping Block, and Force ARP mode.</td>
</tr>
<tr>
<td>CN_GET_NETMODE</td>
<td>Get network mode as WOL, PPPoE, Ping Block, and Force ARP mode.</td>
</tr>
<tr>
<td>CN_SET_TIMEOUT</td>
<td>Set network timeout as retry count and time.</td>
</tr>
<tr>
<td>CN_GET_TIMEOUT</td>
<td>Get network timeout as retry count and time.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IK_WOL</td>
<td>Wake On Lan by receiving the magic packet. Valid in W500.</td>
</tr>
<tr>
<td>IK_PPPOE_TERMINATED</td>
<td>PPPoE Disconnected.</td>
</tr>
<tr>
<td>IK_DEST_UNREACH</td>
<td>Destination IP &amp; Port Unreachable, No use in W5200.</td>
</tr>
<tr>
<td>IK_IP_CONFLICT</td>
<td>IP conflict occurred.</td>
</tr>
<tr>
<td>IK_SOCKET_0</td>
<td>Socket 0 interrupt.</td>
</tr>
<tr>
<td>IK_SOCKET_1</td>
<td>Socket 1 interrupt.</td>
</tr>
<tr>
<td>IK_SOCKET_2</td>
<td>Socket 2 interrupt.</td>
</tr>
<tr>
<td>IK_SOCKET_3</td>
<td>Socket 3 interrupt.</td>
</tr>
<tr>
<td>IK_SOCKET_4</td>
<td>Socket 4 interrupt, No use in 5100.</td>
</tr>
<tr>
<td>IK_SOCKET_5</td>
<td>Socket 5 interrupt, No use in 5100.</td>
</tr>
<tr>
<td>IK_SOCKET_6</td>
<td>Socket 6 interrupt, No use in 5100.</td>
</tr>
<tr>
<td>IK_SOCKET_7</td>
<td>Socket 7 interrupt, No use in 5100.</td>
</tr>
<tr>
<td>IK_SOCKET_ALL</td>
<td>All Socket interrupt.</td>
</tr>
</tbody>
</table>
Definition at line 309 of file `wizchip_conf.h`.

**enum dhcp_mode**

It used in setting dhcp_mode of `wiz_NetInfo`.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETINFO_STATIC</td>
<td>Static IP configuration by manually.</td>
</tr>
<tr>
<td>NETINFO_DHCP</td>
<td>Dynamic IP configuration from a DHCP server.</td>
</tr>
</tbody>
</table>

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

**enum netmode_type**

Network mode

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NM_FORCEARP</td>
<td>Force to APP send whenever udp data is sent. Valid only in W5500.</td>
</tr>
<tr>
<td>NM_WAKEONLAN</td>
<td>Wake On Lan.</td>
</tr>
<tr>
<td>NM_PINGBLOCK</td>
<td>Block ping-request.</td>
</tr>
<tr>
<td>NM_PPPOE</td>
<td>PPPoE mode.</td>
</tr>
</tbody>
</table>

Definition at line 403 of file `wizchip_conf.h`. 
Socket APIs

__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

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>CRIS</td>
</tr>
<tr>
<td>struct</td>
<td>CS</td>
</tr>
<tr>
<td>union</td>
<td>IF</td>
</tr>
</tbody>
</table>
## Public Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16_t if_mode</code></td>
<td>Host interface mode</td>
</tr>
<tr>
<td><code>uint8_t id [6]</code></td>
<td>WIZCHIP ID such as 5100, 5200, 5500, and so on.</td>
</tr>
</tbody>
</table>

```c
struct __WIZCHIP::__CRIS CRIS
```

```c
struct __WIZCHIP::__CS CS
```

```c
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.
<table>
<thead>
<tr>
<th>Member Data Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint16_t</strong> _WIZCHIP::if_mode**</td>
</tr>
<tr>
<td>host interface mode</td>
</tr>
<tr>
<td>Definition at line <strong>203</strong> of file <strong>wizchip_conf.h</strong>.</td>
</tr>
<tr>
<td>Referenced by <strong>reg_wizchip_bus_cbfunc()</strong>, <strong>reg_wizchip_spi_cbfunc()</strong>, and <strong>reg_wizchip_spiburst_cbfunc()</strong>.</td>
</tr>
</tbody>
</table>

| **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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
__WIZCHIP::_IF

Reference

DATA TYPE

#include <wizchip_conf.h>
Public Attributes

```c
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_cbfnc().

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_cbfnc().

struct { ... } __WIZCHIP::IF::BUS

For BUS interface IO
Referenced by reg_wizchip_bus_cbfnc().

uint8_t(* __WIZCHIP::IF::read_byte) (void)

Definition at line 246 of file wizchip_conf.h.
Referenced by reg_wizchip_spi_cbfnc().

void(* __WIZCHIP::IF::write_byte) (uint8_t wb)

Definition at line 247 of file wizchip_conf.h.
Referenced by `reg_wizchip_spi_cbfunc()`.

```c
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()`.

```c
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()`.

```c
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

__WIZCHIP__::__CS

Struct Reference

DATA TYPE

#include <wizchip_conf.h>
## Public Attributes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void(* <em>select</em>)(void)</td>
<td><em>WIZCHIP</em> selected More...</td>
</tr>
<tr>
<td>void(* <em>deselect</em>)(void)</td>
<td><em>WIZCHIP</em> deselected More...</td>
</tr>
</tbody>
</table>
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

__WIZCHIP__: __CRIS

Struct Reference

DATA TYPE

#include <wizchip_conf.h>
### Public Attributes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void(* _enter )(void)</td>
<td>critical section enter More...</td>
</tr>
<tr>
<td>void(* _exit )(void)</td>
<td>critical section exit More...</td>
</tr>
</tbody>
</table>
Detailed Description

The set of critical section callback func.

Definition at line 208 of file \texttt{wizchip\_conf.h}.
Member Data Documentation

`void(* __WIZCHIP::__CRIS::_enter)(void)`

critical section enter

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

Referenced by `reg_wizchip_cris_cbffunc()`.

`void(* __WIZCHIP::__CRIS::_exit)(void)`

critical section exit

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

Referenced by `reg_wizchip_cris_cbffunc()`.

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

- Ethernet/wizchip_conf.h

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
## wiz_PhyConf_t Struct

**Reference**

**DATA TYPE**

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Setting</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>by</td>
<td>uint8_t</td>
<td>set by PHY_CONFBY_HW or PHY_CONFBY_SW</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>uint8_t</td>
<td>set by PHY_MODE_MANUAL or PHY_MODE_AUTONEGO</td>
<td></td>
</tr>
<tr>
<td>speed</td>
<td>uint8_t</td>
<td>set by PHY_SPEED_10 or PHY_SPEED_100</td>
<td></td>
</tr>
<tr>
<td>duplex</td>
<td>uint8_t</td>
<td>set by PHY_DUPLEX_HALF PHY_DUPLEX_FULL</td>
<td></td>
</tr>
</tbody>
</table>
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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by Doxygen 1.8.9.1
#include <wizchip_conf.h>
### Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>mac[6]</td>
<td>Source Mac Address. More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td>ip[4]</td>
<td>Source IP Address. More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td>sn[4]</td>
<td>Subnet Mask. More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td>gw[4]</td>
<td>Gateway IP Address. More...</td>
</tr>
<tr>
<td>uint8_t</td>
<td>dns[4]</td>
<td>DNS server IP Address. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dhcp</td>
<td>dhcp_mode 1 - Static, 2 - DHCP More...</td>
</tr>
</tbody>
</table>
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()`.

```c
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()`.

```c
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

**wiz_NetTimeout_t**

**Struct Reference**

**DATA TYPE**

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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>retry_cnt</td>
<td>retry count</td>
<td>More...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td>time_100us</td>
<td>time unit 100us</td>
<td>More...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Used in CN_SET_TIMEOUT or CN_GET_TIMEOUT of `ctlwizchip()` for timeout configuration.

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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
## Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W5100</strong></td>
<td></td>
<td></td>
<td></td>
<td>Modules</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIZCHIP I/O functions</strong></td>
<td>This supports the basic I/O functions for <strong>WIZCHIP register</strong>.</td>
</tr>
<tr>
<td><strong>WIZCHIP register</strong></td>
<td>WIZCHIP register defines register group of <strong>W5100</strong>.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th><strong>Basic I/O function</strong></th>
<th>These are basic input/output functions to read values from register or write values to register.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common register access functions</strong></td>
<td>These are functions to access common registers.</td>
</tr>
<tr>
<td><strong>Socket register access functions</strong></td>
<td>These are functions to access socket registers.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8_t WIZCHIP_READ (uint32_t AddrSel)</code></td>
<td>It reads 1 byte value from a register.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb)</code></td>
<td>It writes 1 byte value to a register.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It reads sequence data from registers.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE_BUF (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It writes sequence data to registers.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void wiz_send_data (uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to internal TX memory.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void wiz_recv_data (uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to your buffer from internal RX memory.  <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void wiz_recv_ignore (uint8_t sn, uint16_t len)</code></td>
<td>It discard the received data in RX memory.  <a href="#">More...</a></td>
</tr>
</tbody>
</table>
Detailed Description

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

```c
uint8_t WIZCHIP_READ ( uint32_t AddrSel )
```
It reads 1 byte value from a register.

**Parameters**
- **AddrSel** Register address

**Returns**
- The value of register

```c
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

```c
void WIZCHIP_READ_BUF ( uint32_t AddrSel, 
                        uint8_t * pBuf, 
                        uint16_t len )
```
It reads sequence data from registers.
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 read 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().

```c
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 \sim \_\_WIZCHIP\_SOCK\_NUM\_.
- **wizdata** Pointer buffer to read data
- **len** Data length

See also

    wiz_send_data()

Referenced by recv(), and recvfrom().

```c
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

\textbf{(uint8_t)sn} Socket number. It should be 0 - \_WIZCHIP\_SOCK\_NUM\__.

\textbf{len} Data length

Referenced by \textbf{recvfrom()}. 

Generated on Wed May 4 2016 16:44:00 for Socket APIs by \textit{doxygen} 1.8.9.1
# Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Macros</th>
</tr>
</thead>
</table>

## Common register access functions

W5100 » WIZCHIP I/O functions

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

**setMR**(mr)  
```c
*(uint8_t*)MR = mr)
```
Set Mode Register. More...

**getMR()**
```c
*(uint8_t*)MR)
```
Get MR. More...

**setGAR**(gar)
```c
WIZCHIP_WRITE_BUF(GAR,gar,4)
```
Set GAR. More...

**getGAR(gar)**
```c
WIZCHIP_READ_BUF(GAR,gar,4)
```
Get GAR. More...

**setSUBR**(subr)
```c
WIZCHIP_WRITE_BUF(SUBR,subr,4)
```
Set SUBR. More...

**getSUBR(subr)**
```c
WIZCHIP_READ_BUF(SUBR, subr, 4)
```
Get SUBR. More...

**setSHAR**(shar)
```c
WIZCHIP_WRITE_BUF(SHAR, shar, 6)
```
Set SHAR. More...

**getSHAR(shar)**
```c
WIZCHIP_READ_BUF(SHAR, shar, 6)
```
Get SHAR. More...

**setSIPR**(sipr)
```c
WIZCHIP_WRITE_BUF(SIPR, sipr, 4)
```
Set SIPR. More...

**getSIPR(sipr)**
```c
WIZCHIP_READ_BUF(SIPR, sipr, 4)
```
Get SIPR. More...

**setIR**(ir)
```c
WIZCHIP_WRITE(IR, (ir & 0xA0))
```
Set IR register. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define getIR()</code></td>
<td><code>(WIZCHIP_READ(IR) &amp; 0xA0)</code> Get IR register. More...</td>
</tr>
<tr>
<td><code>#define setIMR(imr)</code></td>
<td><code>WIZCHIP_WRITE(_IMR_, imr)</code> Set IMR register. More...</td>
</tr>
<tr>
<td><code>#define getIMR()</code></td>
<td><code>WIZCHIP_READ(_IMR_)</code> Get IMR register. More...</td>
</tr>
<tr>
<td><code>#define setRTR(rtr)</code></td>
<td><code>WIZCHIP_WRITE(_RTR_, rtr)</code> Set RTR register. More...</td>
</tr>
<tr>
<td><code>#define getRTR()</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(_RTR_) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))</code> Get RTR register. More...</td>
</tr>
<tr>
<td><code>#define setRCR(rcr)</code></td>
<td><code>WIZCHIP_WRITE(_RCR_, rcr)</code> Set RCR register. More...</td>
</tr>
<tr>
<td><code>#define getRCR()</code></td>
<td><code>WIZCHIP_READ(_RCR_)</code> Get RCR register. More...</td>
</tr>
<tr>
<td><code>#define setRMSR(rmsr)</code></td>
<td><code>WIZCHIP_WRITE(RMSR)</code> Get RMSR register. More...</td>
</tr>
<tr>
<td><code>#define getRMSR()</code></td>
<td><code>WIZCHIP_READ()</code> Get RMSR register. More...</td>
</tr>
<tr>
<td><code>#define setTMSR(rmsr)</code></td>
<td><code>WIZCHIP_WRITE(TMSR)</code> Get TMSR register. More...</td>
</tr>
<tr>
<td><code>#define getPATR()</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(PATR) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR,1)))</code> Get TMSR register. More...</td>
</tr>
<tr>
<td><code>#define getPPPALGO()</code></td>
<td><code>WIZCHIP_READ(PPPALGO)</code> Get PPPALGO register. More...</td>
</tr>
</tbody>
</table>

For more information, please refer to the documentation or source code.
Get **PPPALGO** register. More...

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

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

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

```c
#define getPMAGIC() WIZCHIP_READ(PMAGIC)
Get **PMAGIC** register. More...
```
Detailed Description

These are functions to access common registers.
Macro Definition Documentation

```c
#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()`.

```c
#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()`.

```c
#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()`.

```c
#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()`.

```c
#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.
Otherwise, 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 (    }


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()`.

```c
#define setRTR(rcr) WIZCHIP_WRITE(_RCR_, rcr)
```

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()`.

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

```c
#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().

```c
#define setRMSR(rmsr) WIZCHIP_WRITE(RMSR)
```

Get RMSR register.

**See also**

getRMSR()  

Definition at line 1276 of file w5100.h.

```c
#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*.

```c
#define setTMSR ( rmsr ) WIZCHIP_WRITE(TMSR)
```

Get **TMSR** register.

**See also**
getTMSR()

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

```c
#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`.

```c
#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`.  

Generated on Wed May 4 2016 16:44:00 for Socket APIs by **doxygen** 1.8.9.1
Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Macros</th>
<th>Functions</th>
</tr>
</thead>
</table>

**Socket register access functions**

W5100 » WIZCHIP I/O functions

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

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Description</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define setSn_MR(sn, mr)  WIZCHIP_WRITE(Sn_MR(sn),mr)</code></td>
<td>Set Sn_MR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_MR(sn)  WIZCHIP_READ(Sn_MR(sn))</code></td>
<td>Get Sn_MR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define setSn_CR(sn, cr)  WIZCHIP_WRITE(Sn_CR(sn), cr)</code></td>
<td>Set Sn_CR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_CR(sn)  WIZCHIP_READ(Sn_CR(sn))</code></td>
<td>Get Sn_CR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define setSn_IR(sn, ir)  WIZCHIP_WRITE(Sn_IR(sn), ir)</code></td>
<td>Set Sn_IR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_IR(sn)  WIZCHIP_READ(Sn_IR(sn))</code></td>
<td>Get Sn_IR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_SR(sn)  WIZCHIP_READ(Sn_SR(sn))</code></td>
<td>Get Sn_SR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define setSn_PORT(sn, port)</code></td>
<td>Set Sn_PORT register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_PORT(sn)  (((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)))</code></td>
<td>Get Sn_PORT register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define setSn_DHAR(sn, dhar)  WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6)</code></td>
<td>Set Sn_DHAR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>#define getSn_DHAR(sn, dhar)  WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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(sn)) << 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(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn), 1))
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(RMSR (WIZCHIP_READ(RMSR) & ~(0x03 << (2*sn))) | (rxmemsize << (2*sn)))
Set Sn_RXMEM_SIZE register. More...

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

#define setSn_TX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn) << 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 setSn_TX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1)))
Get Sn_TX_WR register. More...
```c
#define setSn_RX_RD(sn, rxrd)  
Set Sn_RX_RD register. More...

#define getSn_RX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 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(sn)) << 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), 1)))  
Get Sn_FRAG register. More...

#define getSn_RxMAX(sn) ((uint16_t)(1 << getSn_RXMEM_SIZE(sn) << 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...
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16_t getSn_TX_FSR (uint8_t sn)</code></td>
<td>Get <code>Sn_TX_FSR</code> register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>uint16_t getSn_RX_RSR (uint8_t sn)</code></td>
<td>Get <code>Sn_RX_RSR</code> register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>uint32_t getSn_RxBASE (uint8_t sn)</code></td>
<td>Get the base address of socket sn RX buffer.</td>
<td>More...</td>
</tr>
<tr>
<td><code>uint32_t getSn_TxBASE (uint8_t sn)</code></td>
<td>Get the base address of socket sn TX buffer.</td>
<td>More...</td>
</tr>
</tbody>
</table>
Detailed Description

These are functions to access socket registers.
#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

getSn_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().

```c
#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().

```c
#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)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn))))

Get Sn_PORT register.

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

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 ~
(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`.

```c
#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_SOCKET_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`.

```c
#define setSn_DIPR ( sn, dipr
    WIZCHIP_WRITE_BUF(Sn_DIPR(sn),
    ) 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()**.

```c
#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_NUM__

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

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)) <<
getSn_MSSR ( sn) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn)),1))}
Get `Sn_MSSR` register.

**Parameters**

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

**Returns**

- `uint16_t`. Value of `Sn_MSSR`.

**See also**

- `setSn_MSSR()`

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

Referenced by `getsockopt()`.

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

Set `SnPROTO` register.

**Parameters**

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

- `(uint8_t)proto` Value to set `SnPROTO`

**See also**

- `getSn_PROTO()`

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

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

Get `SnPROTO` register.

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

Returns
uint8_t. Value of SnPROTO.

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()`.

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

Set Sn_RXMEM_SIZE register.

**Parameters**

- `(uint8_t)sn` Socket number. It should be `0 ~ _WIZCHIP_SOCK_NUM_`
- `(uint8_t)rxmemsizesize` Value to set Sn_RXMEM_SIZE

See also
`getSn_RXMEM_SIZE()`

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

```c
#define getSn_RXMEM_SIZE ( sn ) 
                       ((WIZCHIP_READ(RMSR) & (0x03 
                       getSn_RXMEM_SIZE ( sn ) << (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.

```c
#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.

```c
#define getSn_TXMEM_SIZE (sn) (((WIZCHIP_READ(TMSR) & 0x03) << (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**.

```c
#define getSn_TX_RD ( sn ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn)))
```

Get **Sn_TX_RD** register.

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

**Returns**  
uint16_t. Value of **Sn_TX_RD**.

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

Referenced by **send()**.

```c
#define setSn_TX_WR ( sn, txwr )
```

**Value:**

```c
{ 
  WIZCHIP_WRITE(Sn_TX_WR(sn), (uint8_t)(txwr>>8)); \n  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1 (uint8_t) txwr)); \n}
```

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.

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

Get Sn_TX_WR register.

Parameters

- **(uint8_t)sn** Socket number. It should be 0 ~ \_WIZCHIP_SOCK_NUM\_.

Returns

uint16_t. Value of Sn_TX_WR.

See also

- setSn_TX_WR()

Definition at line 1675 of file w5100.h.

```c
#define setSn_RX_RD (sn, rxrd)
```

Value:

```c
{ 
    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.

```c
#define getSn_RX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn))))
```

Get Sn_RX_RD register.

Parameters
- (uint8_t)sn Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.
- uint16_t rxrd Value of Sn_RX_RD.

See also
- setSn_RX_RD()

Definition at line 1705 of file w5100.h.

```c
#define setSn_RX_WR(sn, rxwr)

Value:
```
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.

```c
#define getSn_RX_WR(sn)
    (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) +
     WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn))))
```

Get Sn_RX_WR register.

**Parameters**

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

**Returns**

uint16_t. Value of Sn_RX_WR.

Definition at line 1727 of file w5100.h.

```c
#define setSn_FRAG(sn, frag)
    {
        WIZCHIP_WRITE(Sn_FRAG(sn),
         (uint8_t)(frag >>8));
    }
```
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`.

```c
#define getSn_FRAG (sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn))))
```

Get `Sn_FRAG` register.

**Parameters**

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

**Returns**

- `uint16_t`. Value of `Sn_FRAG`.

See also
- `setSn_FRAG()`

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

```c
#define getSn_RxMAX (sn) (((uint16_t)(1 << getSn_RXMEM_SIZE(sn)) << 10) + getSn_RxMAX(sn))
```

Get the max RX buffer size of socket `sn`.
Parameters

\[
\text{sn} \quad \text{(uint8_t)}
\]
Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.

Returns

\[
\text{uint16_t}
\]
Max buffer size

Definition at line 1758 of file w5100.h.

Referenced by \text{ctlsocket()}, and \text{recv()}. 

\[
\text{#define getSn_TxMAX ( (\text{uint16_t})(1 << \text{getSn_TXMEM_SIZE(sn)}) << 10)}
\]
Get the max TX buffer size of socket \text{sn}.

Parameters

\[
\text{sn} \quad \text{(uint8_t)}
\]
Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.

Returns

\[
\text{uint16_t}
\]
Max buffer size

Definition at line 1768 of file w5100.h.

Referenced by \text{close()}, \text{ctlsocket()}, \text{recv()}, \text{send()}, and \text{sendto()}. 

\[
\text{#define getSn_RxMASK ( sn ) (getSn_RxMAX(sn) - 1)}
\]
Get the mask of socket \text{sn} RX buffer.

Parameters

\[
\text{sn} \quad \text{(uint8_t)}
\]
Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.

Returns

\[
\text{uint16_t}
\]
Mask value
#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 1777 of file w5100.h.

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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZCHIP register</td>
<td>W5100</td>
<td>Modules</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th><strong>Common register</strong></th>
<th>Common register group</th>
</tr>
</thead>
<tbody>
<tr>
<td>It set the basic for the networking</td>
<td></td>
</tr>
<tr>
<td>It set the configuration such as interrupt, network information, ICMP, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Socket register</strong></th>
<th>Socket register group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket register configures and control SOCKETn which is necessary to data communication.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

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

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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Macros</th>
</tr>
</thead>
</table>

**Common register**

W5100 » WIZCHIP register

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

<table>
<thead>
<tr>
<th>Define</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td><em>WIZCHIP_IO_BASE</em> + (0x0000)</td>
<td>Mode Register address(R/W) MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...</td>
</tr>
<tr>
<td>GAR</td>
<td><em>W5100_IO_BASE</em> + (0x0001)</td>
<td>Gateway IP Register address(R/W) More...</td>
</tr>
<tr>
<td>SUBR</td>
<td><em>W5100_IO_BASE</em> + (0x0005)</td>
<td>Subnet mask Register address(R/W) More...</td>
</tr>
<tr>
<td>SHAR</td>
<td><em>W5100_IO_BASE</em> + (0x0009)</td>
<td>Source MAC Register address(R/W) More...</td>
</tr>
<tr>
<td>SIPR</td>
<td><em>W5100_IO_BASE</em> + (0x000F)</td>
<td>Source IP Register address(R/W) More...</td>
</tr>
<tr>
<td>IR</td>
<td><em>W5100_IO_BASE</em> + (0x0015)</td>
<td>Interrupt Register(R/W) More...</td>
</tr>
<tr>
<td>IMR_</td>
<td><em>W5100_IO_BASE</em> + (0x0016)</td>
<td>Socket Interrupt Mask Register(R/W) More...</td>
</tr>
<tr>
<td>RTR_</td>
<td><em>W5100_IO_BASE</em> + (0x0017)</td>
<td>Timeout register address( 1 is 100us )(R/W) More...</td>
</tr>
<tr>
<td>RCR_</td>
<td><em>W5100_IO_BASE</em> + (0x0019)</td>
<td>Retry count register(R/W) More...</td>
</tr>
<tr>
<td>PATR</td>
<td><em>W5100_IO_BASE</em> + (0x001C)</td>
<td>PPP LCP Request Timer register in PPPoE mode(R) More...</td>
</tr>
<tr>
<td>PTIMER</td>
<td><em>W5100_IO_BASE</em> + (0x0028)</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>RST</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>WOL</td>
</tr>
<tr>
<td>4</td>
<td>PB</td>
</tr>
<tr>
<td>3</td>
<td>PPPoE</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>0</td>
<td>IND</td>
</tr>
</tbody>
</table>

- **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 will be written 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.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFLICT</td>
<td>UNREACH</td>
<td>PPPoE</td>
<td>Reserved</td>
<td>S3_INT</td>
<td>S2_INT</td>
<td>S1_INT</td>
</tr>
</tbody>
</table>

- **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.

```c
#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 set and the corresponding bit of IR is set, Interrupt will be issued.

Definition at line 264 of file w5100.h.

```c
#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.

```c
#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').
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.

PPP LCP Magic number register in PPPoE mode(R)

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

---

**#define PATR**  (*_W5100_IO_BASE_ + (0x001C))

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

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

---

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

##define PATR  (*_W5100_IO_BASE_ + (0x001C))

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

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

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

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

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

##define PMAGIC  (*_W5100_IO_BASE_ + (0x0029))

---

Generated on Wed May 4 2016 16:44:00 for Socket APIs by [Doxygen](https://www.doxygen.nl) 1.8.9.1
Socket APIs

Socket register

Socket register group
Socket register configures and control SOCKETn which is necessary to data communication. More...
<table>
<thead>
<tr>
<th>Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR</td>
<td>Socket Mode register(R/W)</td>
</tr>
<tr>
<td>Sn_CR</td>
<td>Socket command register(R/W)</td>
</tr>
<tr>
<td>Sn_IR</td>
<td>Socket interrupt register(R)</td>
</tr>
<tr>
<td>Sn_SR</td>
<td>Socket status register(R)</td>
</tr>
<tr>
<td>Sn_PORT</td>
<td>Source port register(R/W)</td>
</tr>
<tr>
<td>Sn_DHAR</td>
<td>Peer MAC register address(R/W)</td>
</tr>
<tr>
<td>Sn_DIPR</td>
<td>Peer IP register address(R/W)</td>
</tr>
<tr>
<td>Sn_DPORT</td>
<td>Peer port register address(R/W)</td>
</tr>
<tr>
<td>Sn_MSSR</td>
<td>Peer MAC Status register(R)</td>
</tr>
</tbody>
</table>
Maximum Segment Size(Sn_MSSR0) register address(R/W) More...

```c
#define Sn_PROTO(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0014))
IP Protocol(PROTO) Register(R/W) More...
```

```c
#define Sn_TOS(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + 0x0015)
IP Type of Service(TOS) Register(R/W) More...
```

```c
#define Sn_TTL(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0016))
IP Time to live(TTL) Register(R/W) More...
```

```c
#define Sn_TX_FSR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020))
Transmit free memory size register(R) More...
```

```c
#define Sn_TX_RD(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022))
Transmit memory read pointer register address(R) More...
```

```c
#define Sn_TX_WR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024))
Transmit memory write pointer register address(R/W) More...
```

```c
#define Sn_RX_RSR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0026))
Received data size register(R) More...
```

```c
#define Sn_RX_RD(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0028))
Read point of Receive memory(R/W) More...
```

```c
#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
#define (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK)
Sn_MR ( sn ) (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:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

- **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 multicasting**

**Protocol**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TCP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>UDP</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
</tbody>
</table>

- **In case of Socket 0**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PPPoE</td>
</tr>
</tbody>
</table>

- **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.

```c
#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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x23</td>
<td>PCON</td>
<td>PPPoE connection begins by transmitting PPPoE discovery packet</td>
</tr>
<tr>
<td>0x24</td>
<td>PDISCON</td>
<td>Closes PPPoE connection</td>
</tr>
<tr>
<td>0x25</td>
<td>PCR</td>
<td>In each phase, it transmits REQ message.</td>
</tr>
<tr>
<td>0x26</td>
<td>PCN</td>
<td>In each phase, it transmits NAK message.</td>
</tr>
</tbody>
</table>
In each phase, it transmits REJECT message.

Definition at line 380 of file w5100.h.

```c
#define Sn_IR(_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK + (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 IMR is 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 0
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.

```c
#define Sn_SR(_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.

```c
#define Sn_PORT (_W5100_IO_BASE_ + (sn)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.

```c
#define Sn_DHAR (_W5100_IO_BASE_ + (sn)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**.

```c
#define Sn_DHAR(_W5100_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 450 of file **w5100.h**.

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

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 client mode, it configures the listen port number of TCP server before CONNECT command. In TCP server mode, 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**.

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

```c
#define Sn_MSSR (_W5100_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 467 of file *w5100.h*.

### Sn_PROTO

```c
#define Sn_PROTO (_W5100_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 475 of file *w5100.h*.

### Sn_TOS

```c
#define Sn_TOS (_W5100_IO_BASE_ + 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 483 of file *w5100.h*.

### Sn_TTL

```c
#define Sn_TTL (_W5100_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 491 of file **w5100.h**.

```c
#define Sn_TX_FSR (_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**.

```c
#define Sn_TX_RD (_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
**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 0xFFF (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_TX_WR (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024))
```

**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 ( _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, 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 ( _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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5200</td>
<td></td>
<td></td>
<td></td>
<td>Modules</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th><strong>WIZCHIP I/O functions</strong></th>
<th>This supports the basic I/O functions for WIZCHIP register.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIZCHIP register</strong></td>
<td>WIZCHIP register defines register group of W5200.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZCHIP I/O functions</td>
<td>W5200</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Basic I/O function</th>
<th>These are basic input/output functions to read values from register or write values to register.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common register access functions</td>
<td>These are functions to access common registers.</td>
</tr>
<tr>
<td>Socket register access functions</td>
<td>These are functions to access socket registers.</td>
</tr>
</tbody>
</table>
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
W5200 » WIZCHIP I/O functions

These are basic input/output functions to read values from register or write values to register. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8_t WIZCHIP_READ(uint32_t AddrSel)</code></td>
<td>It reads 1 byte value from a register.</td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb)</code></td>
<td>It writes 1 byte value to a register.</td>
</tr>
<tr>
<td><code>void WIZCHIP_READ_BUF(uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It reads sequence data from registers.</td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE_BUF(uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It writes sequence data to registers.</td>
</tr>
<tr>
<td><code>void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to internal TX memory.</td>
</tr>
<tr>
<td><code>void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to your buffer from internal RX memory.</td>
</tr>
<tr>
<td><code>void wiz_recv_ignore(uint8_t sn, uint16_t len)</code></td>
<td>It discard the received data in RX memory.</td>
</tr>
</tbody>
</table>
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

```c
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

```c
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

Generated on Wed May 4 2016 16:44:00 for Socket APIs by doxygen 1.8.9.1
Socket APIs

Common register access functions
W5200 » WIZCHIP I/O functions

These are functions to access common registers. More...
### Macros

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```
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)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define setPMAGIC(pmagic) WIZCHIP_WRITE(PMAGIC, pmagic)</code></td>
<td>Set PMAGIC register. More...</td>
</tr>
<tr>
<td><code>#define getPMAGIC()</code></td>
<td>Get PMAGIC register. More...</td>
</tr>
<tr>
<td><code>#define setINTLEVEL(intlevel)</code></td>
<td>Set INTLEVEL register. More...</td>
</tr>
<tr>
<td><code>#define getIntLEVEL()</code></td>
<td>Get INTLEVEL register. More...</td>
</tr>
<tr>
<td><code>#define setIR2(ir2) WIZCHIP_WRITE(IR2, ir2)</code></td>
<td>Set IR2 register. More...</td>
</tr>
<tr>
<td><code>#define getIR2()</code></td>
<td>Get IR2 register. More...</td>
</tr>
<tr>
<td><code>#define getPHYSTATUS()</code></td>
<td>Get PHYSTATUS register. More...</td>
</tr>
<tr>
<td><code>#define setIMR2(imr2) WIZCHIP_WRITE(_IMR_, imr2)</code></td>
<td>Set IMR register. More...</td>
</tr>
<tr>
<td><code>#define getIMR2()</code></td>
<td>Get IMR register. More...</td>
</tr>
</tbody>
</table>
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()
#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 1307 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 1316 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
(\texttt{uint16\_t})\texttt{rtr} Value to set RTR register.

See also
getRTR()
Definition at line 1379 of file w5200.h.

#define getRTR ( ) (((uint16\_t)WIZCHIP_READ(_RTR_) \ll 8) +
WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))

Get RTR register.

Returns
\texttt{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 ) 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));
}
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()**.

```c
#define getINTLEVEL() (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL)))
```

Get **INTLEVEL** register.

**Returns**

- `uint16_t` Value of **INTLEVEL** register.

**See also**

- **setINTLEVEL()**

Definition at line 1488 of file **w5200.h**.

Referenced by **ctlwizchip()**.

```c
#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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Macros</th>
<th>Functions</th>
</tr>
</thead>
</table>

### Socket register access functions

*W5200 > WIZCHIP I/O functions*

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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setSn_MR(sn, mr)</td>
<td>Set Sn_MR register.</td>
</tr>
<tr>
<td>setSn_CR(sn, cr)</td>
<td>Set Sn_CR register.</td>
</tr>
<tr>
<td>getSn_CR(sn)</td>
<td>Get Sn_CR register.</td>
</tr>
<tr>
<td>setSn_IR(sn, ir)</td>
<td>Set Sn_IR register.</td>
</tr>
<tr>
<td>getSn_IR(sn)</td>
<td>Get Sn_IR register.</td>
</tr>
<tr>
<td>setSn_IMR(sn, imr)</td>
<td>Set Sn_IMR register.</td>
</tr>
<tr>
<td>getSn_IMR(sn)</td>
<td>Get Sn_IMR register.</td>
</tr>
<tr>
<td>getSn_SR(sn)</td>
<td>Get Sn_SR register.</td>
</tr>
</tbody>
</table>

More...
```c
#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(sn), dhar, 6)
    Set Sn_DHAR register. More...

#define getSn_DHAR(sn, dhar) WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar, 6)
    Get Sn_DHAR register. More...

#define setSn_DIPR(sn, dipr) WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)
    Set Sn_DIPR register. More...

#define getSn_DIPR(sn, dipr) WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 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(sn)) << 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(sn)) << 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), rxmemsize)
Set Sn_RXMEM_SIZE register. More...

#define getSn_RXMEM_SIZE(sn)   WIZCHIP_READ(Sn_RXMEM_SIZE(sn))
Get Sn_RXMEM_SIZE register. More...

#define setSn_TXMEM_SIZE(sn, txmemsize)   WIZCHIP_WRITE(Sn_TXMEM_SIZE(sn), txmemsize)
Set Sn_TXMEM_SIZE register. More...

#define getSn_TXMEM_SIZE(sn)   WIZCHIP_READ(Sn_TXMEM_SIZE(sn))
Get Sn_TXMEM_SIZE register. More...

#define getSn_TX_RD(sn)   (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn), 1)))
Get Sn_TX_RD register. More...
<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define setSn_TX_WR(sn, txwr)</code></td>
<td>Set <code>Sn_TX_WR</code> register. More...</td>
</tr>
<tr>
<td><code>#define getSn_TX_WR(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_TX_WR) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1)))</code>.Get <code>Sn_TX_WR</code> register. More...</td>
</tr>
<tr>
<td><code>#define setSn_RX_RD(sn, rxrd)</code></td>
<td>Set <code>Sn_RX_RD</code> register. More...</td>
</tr>
<tr>
<td><code>#define getSn_RX_RD(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1)))</code>Get <code>Sn_RX_RD</code> register. More...</td>
</tr>
<tr>
<td><code>#define setSn_RX_WR(sn, rxwr)</code></td>
<td>Set <code>Sn_RX_WR</code> register. More...</td>
</tr>
<tr>
<td><code>#define getSn_RX_WR(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_RX_WR) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1)))</code>Get <code>Sn_RX_WR</code> register. More...</td>
</tr>
<tr>
<td><code>#define setSn_FRAG(sn, frag)</code></td>
<td>Set <code>Sn_FRAG</code> register. More...</td>
</tr>
<tr>
<td><code>#define getSn_FRAG(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_FRAG) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)))</code>Get <code>Sn_FRAG</code> register. More...</td>
</tr>
<tr>
<td><code>#define getSn_RxMAX(sn)</code></td>
<td><code>((uint16_t)getSn_RXMEM_SIZE(sn) &lt;&lt; 10)</code>Get the max RX buffer size of socket sn. More...</td>
</tr>
<tr>
<td><code>#define getSn_TxMAX(sn)</code></td>
<td><code>((uint16_t)getSn_TXMEM_SIZE(sn) &lt;&lt; 10)</code>Get the max TX buffer size of socket sn. More...</td>
</tr>
</tbody>
</table>
#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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Function Name</th>
<th>Argument (uint8_t sn)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
<td>getSn_TX_FSR</td>
<td>(uint8_t sn)</td>
<td>Get Sn_TX_FSR register. More...</td>
</tr>
<tr>
<td>uint16_t</td>
<td>getSn_RX_RSR</td>
<td>(uint8_t sn)</td>
<td>Get Sn_RX_RSR register. More...</td>
</tr>
<tr>
<td>uint16_t</td>
<td>getSn_RxBASE</td>
<td>(uint8_t sn)</td>
<td>Get the base address of socket sn RX buffer. More...</td>
</tr>
<tr>
<td>uint16_t</td>
<td>getSn_TxBASE</td>
<td>(uint8_t sn)</td>
<td>Get the base address of socket sn TX buffer. More...</td>
</tr>
</tbody>
</table>
Detailed Description

These are functions to access **socket registers**.
Macro Definition Documentation

```c
#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`.

```c
#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`. 
```c
#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`.

```c
#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`.

```c
#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**.

```c
#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**.

```c
#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()
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 1963 of file w5200.h.

Referenced by ctlsocket().

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.

Get Sn_SR register.

Parameters

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

Returns

- uint8_t. Value of Sn_SR.

See also

- setSn_SR()
Get Sn_SR register.

**Parameters**

\[(\text{uint8_t})\text{sn}\] Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.

**Returns**

uint8_t. Value of Sn_SR.

Definition at line 1638 of file w5200.h.

```c
#define setSn_PORT (sn, port)
```

**Value:**

```c
{
    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**

\[(\text{uint8_t})\text{sn}\] Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.

\[(\text{uint16_t})\text{port}\] 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)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT)))

Get Sn_PORT register.

**Parameters**

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

**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.
```c
#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`.

```c
#define setSn_DPORT ( sn, dport )
```

**Value:**

```c
{
    \ WIZCHIP_WRITE(Sn_DPORT(sn),
    (uint8_t) (dport>>8)); \n
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1
    (uint8_t) dport); \n}
```

Set **Sn_DPORT** register.

**Parameters**

- `(uint8_t)sn` Socket number. It should be `0 ~`
(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)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_I

Get Sn_DPORT register.

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

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**.

```c
#define getSn_MSSR (sn) (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn))))
```

Get **Sn_MSSR** register.

**Parameters**

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

**Returns**

- `uint16_t`. Value of **Sn_MSSR**.

See also

- `setSn_MSSR()`

Definition at line **1744** of file **w5200.h**.

```c
#define setSn_PROTO (sn, proto) WIZCHIP_WRITE(Sn_PROTO(sn), proto)
```

Set **Sn_PROTO** register.

**Parameters**

- `sn`
- `proto`
(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
(u8 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),rxmemsize)

Set Sn_RXMEM_SIZE register.

Parameters
(u8 sn) Socket number. It should be 0 ~ _WIZCHIP_SOCK_NUM_.
(u8 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(sn))

Get Sn_RXMEM_SIZE register.
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*.

```c
#define setSn_RXMEM_SIZE (sn, txmemsize)  
    WIZCHIP_WRITE(Sn_RXMEM_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*.

```c
#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)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD( sn ))))

Get Sn_TX_RD register.

Parameters

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

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 \texttt{Sn\_TX\_WR} register.

\textbf{Parameters}

- \texttt{(uint8\_t)sn} Socket number. It should be \(0 \sim \_\text{WIZCHIP\_SOCK\_NUM}\).
- \texttt{(uint16\_t)txwr} Value to set \texttt{Sn\_TX\_WR}

\textbf{See also}

Get\texttt{Sn\_TX\_WR()}

Definition at line \textbf{1889} of file \texttt{w5200.h}.

\begin{verbatim}
#define getSn_TX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn)
\end{verbatim}

Get \texttt{Sn\_TX\_WR} register.

\textbf{Parameters}

- \texttt{(uint8\_t)sn} Socket number. It should be \(0 \sim \_\text{WIZCHIP\_SOCK\_NUM}\)

\textbf{Returns}

- \texttt{uint16\_t}. Value of \texttt{Sn\_TX\_WR}.

\textbf{See also}

set\texttt{Sn\_TX\_WR()}

Definition at line \textbf{1901} of file \texttt{w5200.h}.

\begin{verbatim}
#define setSn_RX_RD(sn, rxrd)
\end{verbatim}

\textbf{Value:}

\begin{verbatim}
{ \ WIZCHIP_WRITE(Sn_RX_RD(sn),
\end{verbatim}
(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))
getSn_RX_RD(sn)WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_F

Get Sn_RX_RD register.

Parameters

  (uint8_t)sn  Socket number. It should be 0 ~ \_WIZCHIP_SOCK_NUM\_.

Returns

  uint16_t. Value of Sn_RX_RD.

See also

getSn_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 ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_I
```

Get Sn_RX_WR register.

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

Returns
- `uint16_t`. Value of Sn_RX_WR.

Definition at line 1953 of file w5200.h.

```
#define setSn_FRAG ( sn, frag
```
Value:

```c
{
    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`.

```c
#define getSn_FRAG( sn ) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) <<
                   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FF
```

Get `Sn_FRAG` register.

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

**Returns**
- `uint16_t`. Value of `Sn_FRAG`.

See also
- `setSn_FRAG()`

Definition at line `1995` of file `w5200.h`. 
#define getSn_RxMAX ( (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 ( (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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZCHIP register</td>
<td>W5200</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Generated on Wed May 4 2016 16:44:00 for Socket APIs by doxygen 1.8.9.1
Socket APIs

Common register
W5200 » WIZCHIP register

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

<table>
<thead>
<tr>
<th>#define</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MR</code> (<em>W5200_IO_BASE</em> + (0x0000))</td>
<td>Mode Register address(R/W)</td>
<td><code>MR</code> is used for S/W reset, ping block mode, PPPoE mode and etc. More...</td>
</tr>
<tr>
<td><code>GAR</code> (<em>W5200_IO_BASE</em> + (0x0001))</td>
<td>Gateway IP Register address(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SUBR</code> (<em>W5200_IO_BASE</em> + (0x0005))</td>
<td>Subnet mask Register address(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SHAR</code> (<em>W5200_IO_BASE</em> + (0x0009))</td>
<td>Source MAC Register address(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SIPR</code> (<em>W5200_IO_BASE</em> + (0x000F))</td>
<td>Source IP Register address(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>IR</code> (<em>W5200_IO_BASE</em> + (0x0015))</td>
<td>Interrupt Register(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>_IMR_</code> (<em>W5200_IO_BASE</em> + (0x0016))</td>
<td>Socket Interrupt Mask Register(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>_RTR_</code> (<em>W5200_IO_BASE</em> + (0x0017))</td>
<td>Timeout register address( 1 is 100us ) (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>_RCR_</code> (<em>W5200_IO_BASE</em> + (0x0019))</td>
<td>Retry count register(R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>PATR</code> (<em>W5200_IO_BASE</em> + (0x001C))</td>
<td>PPP LCP Request Timer register in PPPoE mode(R)</td>
<td>More...</td>
</tr>
<tr>
<td><code>PPPALGO</code> (<em>W5200_IO_BASE</em> + (0x001E))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define Name</td>
<td>Address Expression</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VERSIONR</td>
<td>W5200_IO_BASE_ + 0x001F</td>
<td>Chip version register address (R)</td>
</tr>
<tr>
<td>PTIMER</td>
<td>W5200_IO_BASE_ + 0x0028</td>
<td>PPP LCP Request Timer register in PPPoE mode (R)</td>
</tr>
<tr>
<td>PMAGIC</td>
<td>W5200_IO_BASE_ + 0x0029</td>
<td>PPP LCP Magic number register in PPPoE mode (R)</td>
</tr>
<tr>
<td>INTLEVEL</td>
<td>W5200_IO_BASE_ + 0x0030</td>
<td>Set Interrupt low level timer register address (R/W)</td>
</tr>
<tr>
<td>IR2</td>
<td>W5200_IO_BASE_ + 0x0034</td>
<td>Socket Interrupt Register (R/W)</td>
</tr>
<tr>
<td>PHYSTATUS</td>
<td>W5200_IO_BASE_ + 0x0035</td>
<td>PHYSTATUS (R/W)</td>
</tr>
<tr>
<td>IMR2</td>
<td>W5200_IO_BASE_ + 0x0036</td>
<td>Interrupt mask register (R/W)</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>RST</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>WOL</td>
</tr>
<tr>
<td>4</td>
<td>PB</td>
</tr>
<tr>
<td>3</td>
<td>PPPoE</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>0</td>
<td>IND</td>
</tr>
</tbody>
</table>

- **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.*

```c
#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.*

```c
#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.*

```c
#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 writes to the host. If **IR** is not equal to x00 INTn PIN is asserted to low until it is x00.

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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFLICT</td>
<td>Reserved</td>
<td>PPPoE</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- **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**.

```c
#define PATR (_W5200_IO_BASE_ + (0x001C))
```

**PPPoE** 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**.

```c
#define PPPALGO (_W5200_IO_BASE_ + (0x001E))
```

**PPPoE** 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**.

```c
#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**.

```c
#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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>LINK</th>
<th>POWERSAVE</th>
<th>POWERDOWN</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>Reserved</td>
<td>Reserved</td>
<td>3</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- **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 bit of IMR2 is and the corresponding bit of IR is an interrupt will be issued. In other words, if a bit of IMR2 is an interrupt will not be issued even if the corresponding bit of IR is

Each bit of IMR2 defined as the following.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>IM_IR5</th>
<th>Reserved</th>
<th>IM_IR3</th>
<th>Reserved</th>
<th>Reserved</th>
<th>Reserved</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>IM_IR7</td>
<td>Reserved</td>
<td>IM_IR5</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- **IM_IR7** : IP Conflict Interrupt Mask
- **IM_IR5** : PPPoE Close Interrupt Mask

Note

This Register is same operated as _IMR_ of W5100, W5300 & W5550.

So, setIMR() set a value to IMR2 for integrating with ioLibrary.

Definition at line 389 of file w5200.h.
Socket register group
Socket register configures and control SOCKETn which is necessary to data communication. More...
## Macros

**Sn_MR**

```c
#define Sn_MR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0000))
socket Mode register(R/W) More...
```

**Sn_CR**

```c
#define Sn_CR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001))
Socket command register(R/W) More...
```

**Sn_IR**

```c
#define Sn_IR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0002))
Socket interrupt register(R) More...
```

**Sn_SR**

```c
#define Sn_SR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0003))
Socket status register(R) More...
```

**Sn_PORT**

```c
#define Sn_PORT(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004))
source port register(R/W) More...
```

**Sn_DHAR**

```c
#define Sn_DHAR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006))
Peer MAC register address(R/W) More...
```

**Sn_DIPR**

```c
#define Sn_DIPR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x000C))
Peer IP register address(R/W) More...
```

**Sn_DPORT**

```c
#define Sn_DPORT(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0010))
Peer port register address(R/W) More...
```

**Sn_MSSR**

```c
#define Sn_MSSR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0012))
```
Maxmum 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))
<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sn_RX_RD</strong>&lt;sup&gt;(sn)&lt;/sup&gt;</td>
<td>(<em>W5200_IO_BASE</em> + WIZCHIP_SREG_BLOCK&lt;sup&gt;(sn)&lt;/sup&gt; + (0x0028))</td>
<td>Read point of Receive memory(R/W)</td>
</tr>
<tr>
<td><strong>Sn_RX_WR</strong>&lt;sup&gt;(sn)&lt;/sup&gt;</td>
<td>(<em>W5200_IO_BASE</em> + WIZCHIP_SREG_BLOCK&lt;sup&gt;(sn)&lt;/sup&gt; + (0x002A))</td>
<td>Write point of Receive memory(R)</td>
</tr>
<tr>
<td><strong>Sn_IMR</strong>&lt;sup&gt;(sn)&lt;/sup&gt;</td>
<td>(<em>W5200_IO_BASE</em> + WIZCHIP_SREG_BLOCK&lt;sup&gt;(sn)&lt;/sup&gt; + (0x002C))</td>
<td>socket interrupt mask register(R)</td>
</tr>
<tr>
<td><strong>Sn_FRAG</strong>&lt;sup&gt;(sn)&lt;/sup&gt;</td>
<td>(<em>W5200_IO_BASE</em> + WIZCHIP_SREG_BLOCK&lt;sup&gt;(sn)&lt;/sup&gt; + (0x002D))</td>
<td>Fragment field value in IP header register(R/W)</td>
</tr>
</tbody>
</table>
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

```c
#define (_W5200_IO_BASE_ + WIZCHIP_SREG_BLO
Sn_MR ( sn ) (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.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

- **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 multicasting
- **Protocol**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TCP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>UDP</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
</tbody>
</table>

- **In case of Socket 0**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PPPoE</td>
</tr>
</tbody>
</table>

- **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.

```c
#define Sn_CR (_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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x23</td>
<td>PCON</td>
<td>PPPoE connection begins by transmitting PPPoE discovery packet</td>
</tr>
<tr>
<td>0x24</td>
<td>PDISCON</td>
<td>Closes PPPoE connection</td>
</tr>
<tr>
<td>0x25</td>
<td>PCR</td>
<td>In each phase, it transmits REQ message.</td>
</tr>
<tr>
<td>0x26</td>
<td>PCN</td>
<td>In each phase, it transmits NAK message.</td>
</tr>
<tr>
<td>0x27</td>
<td>PCJ</td>
<td>In each phase, it transmits REJECT</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRECV</td>
<td>PFAIL</td>
<td>PNEXT</td>
<td>SEND_OK</td>
<td>TIMEOUT</td>
<td>RECV</td>
<td>DISCON</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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

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`.

```c
#define Sn_PORT (_W5200_IO_BASE_ +
              ( _sn ) 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`.

```c
#define Sn_DHAR (_W5200_IO_BASE_ +
              ( _sn ) 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**.

```c
#define Sn_DHAR (_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**.

```c
#define Sn_DIPR (_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 client mode, it configures the listen port number of TCP server before CONNECT command. In TCP server mode, 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**.

```c
#define Sn_DPORT (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0010))
```
**Sn_MSSR** \( \text{( sn ) WIZCHIP_SREG_BLOCK(sn) + (0x0012)} \)

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*.

```c
#define Sn_PROTO ( _W5200_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 **552** of file *w5200.h*.

```c
#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*.

```c
#define Sn_TTL ( _W5200_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.
#define Sn_RXMEM_SIZE ( _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 568 of file **w5200.h**.

#define Sn_TXMEM_SIZE ( _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 588 of file **w5200.h**.

#define Sn_TX_FSR ( _W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020))


Transmit free memory size register(R)

\textbf{Sn\_TX\_FSR} indicates the free size of Socket n TX Buffer Block. It is initialized to the configured size by \textbf{Sn\_TXMEM\_SIZE}. Data bigger than \textbf{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 \texttt{w5200.h}.

\begin{verbatim}
#define Sn_TX_RD ( _W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022))
\end{verbatim}

Transmit memory read pointer register address(R)

\textbf{Sn\_TX\_RD} is initialized by OPEN command. However, if \textbf{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 \textbf{Sn\_TX\_RD} to the \textbf{Sn\_TX\_WR} in the Socket n TX Buffer. After transmitting the saved data, the SEND command increases the \textbf{Sn\_TX\_RD} as same as the \textbf{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 \texttt{w5200.h}.

\begin{verbatim}
#define Sn_TX_WR ( _W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024))
\end{verbatim}

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**.

```c
#define Sn_RX_RSR ( _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**.

```c
#define Sn_RX_RD ( _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 \texttt{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 \texttt{Sn\_RX\_RD} to W5200.

Definition at line 658 of file \texttt{w5200.h}.

\begin{verbatim}
define ( \_W5200\_IO\_BASE\_ +
Sn\_RX\_WR  (  sn ) WIZCHIP\_SREG\_BLOCK(sn) + (0x002A))
\end{verbatim}

Write point of Receive memory(R)

\texttt{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 \texttt{w5200.h}.

\begin{verbatim}
define ( \_W5200\_IO\_BASE\_ +
Sn\_IMR   (  sn ) WIZCHIP\_SREG\_BLOCK(sn) + (0x002C))
\end{verbatim}

socket interrupt mask register(R)

\texttt{Sn\_IMR} masks the interrupt of Socket n. Each bit corresponds to each bit of \texttt{Sn\_IR}. When a Socket n Interrupt is occurred and the corresponding bit of \texttt{Sn\_IMR} is the corresponding bit of \texttt{Sn\_IR} becomes When both the corresponding bit of \texttt{Sn\_IMR} and \texttt{Sn\_IR} are and the n-th bit of \texttt{IR} is Host is interrupted by asserted INTn PIN to low.

Definition at line 677 of file \texttt{w5200.h}. 
#define Sn_FRAG (_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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W5300</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHIZCHIP register defines and I/O functions of **W5300**. More...
## Modules

<table>
<thead>
<tr>
<th><strong>WIZCHIP I/O functions</strong></th>
<th>This supports the basic I/O functions for <strong>WIZCHIP register</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIZCHIP register</strong></td>
<td>WHIZCHIP register defines register group of <strong>W5300</strong>.</td>
</tr>
</tbody>
</table>
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

Generated on Wed May 4 2016 16:44:00 for Socket APIs by doxygen 1.8.9.1
Socket APIs

WIZCHIP I/O functions

W5300

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

<table>
<thead>
<tr>
<th><strong>Basic I/O function</strong></th>
<th>These are basic input/output functions to read values from register or write values to register.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common register access functions</strong></td>
<td>These are functions to access common registers.</td>
</tr>
<tr>
<td><strong>Socket register access functions</strong></td>
<td>These are functions to access socket registers.</td>
</tr>
</tbody>
</table>
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()

- **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()`, `setSn_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()`
## Basic I/O function

*W5300 > WIZCHIP I/O functions*

These are basic input/output functions to read values from register or write values to register. [More...](#)
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint16_t WIZCHIP_READ (uint32_t AddrSel)</code></td>
<td>It reads 1 byte value from a register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE (uint32_t AddrSel, uint16_t wb)</code></td>
<td>It writes 1 byte value to a register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void wiz_send_data (uint8_t sn, uint8_t *wizdata, uint32_t len)</code></td>
<td>It copies data to internal TX memory.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void wiz_recv_data (uint8_t sn, uint8_t *wizdata, uint32_t len)</code></td>
<td>It copies data to your buffer from internal RX memory.</td>
<td>More...</td>
</tr>
<tr>
<td><code>void wiz_recv_ignore (uint8_t sn, uint32_t len)</code></td>
<td>It discard the received data in RX memory.</td>
<td>More...</td>
</tr>
</tbody>
</table>
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 \textit{wizdata(pointer buffer)} of the length of \textit{len(variable)} bytes to internal TX memory and updates the Tx write pointer register. This function is being called by \textit{send()} and \textit{sendto()} function also.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{(uint8\_t)sn} Socket number. It should be \textit{0} \textit{~} \textit{7}.
  \item \texttt{wizdata} Pointer buffer to write data
  \item \texttt{len} Data length
\end{itemize}

See also

\textit{wiz\_recv\_data()}

\begin{verbatim}
void wiz_recv_data ( uint8_t sn,
                   uint8_t * wizdata,
                   uint32_t len
                     )
\end{verbatim}

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 \textit{wizdata(pointer variable)} of the length of \textit{len(variable)} bytes. This function is being called by \textit{recv()} also.

\textbf{Parameters}

\begin{itemize}
  \item \texttt{(uint8\_t)sn} Socket number. It should be \textit{0} \textit{~} \textit{7}.
  \item \texttt{wizdata} Pointer buffer to read data
  \item \texttt{len} Data length
\end{itemize}

See also

\textit{wiz\_send\_data()}

\begin{verbatim}
void wiz_recv_ignore ( uint8_t sn,
                      uint32_t len
                        )
\end{verbatim}
It discard the received data in RX memory.

It discards the data of the length of \textit{len}\textit{(variable)} bytes in internal RX memory.

\textbf{Parameters}

\begin{itemize}
  \item \textbf{\texttt{uint8_t}sn} Socket number. It should be 0 ~ 7.
  \item \textbf{len} Data length
\end{itemize}
# Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Common register access functions

W5300 » WIZCHIP I/O functions

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

<table>
<thead>
<tr>
<th>Define</th>
<th>Function</th>
<th>Description</th>
<th>More Link</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setIR(ir)</code></td>
<td><code>WIZCHIP_WRITE(IR, ir &amp; 0xF0FF)</code></td>
<td>Set Mode Register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>getIR()</code></td>
<td><code>WIZCHIP_READ(IR) &amp; 0xF0FF)</code></td>
<td>Get IR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>setIMR(imr)</code></td>
<td><code>WIZCHIP_WRITE(_IMR_, imr &amp; 0xF0FF)</code></td>
<td>Set IMR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>getIMR()</code></td>
<td><code>WIZCHIP_READ(_IMR_) &amp; 0xF0FF)</code></td>
<td>Get IMR register.</td>
<td>More...</td>
</tr>
<tr>
<td><code>setSHAR(shar)</code></td>
<td></td>
<td>Set local MAC address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>setGAR(gar)</code></td>
<td></td>
<td>Set gateway IP address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>getGAR()</code></td>
<td></td>
<td>Get gateway IP address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>setSUBR(subr)</code></td>
<td></td>
<td>Set subnet mask address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>getSUBR()</code></td>
<td></td>
<td>Get subnet mask address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>setSIPR(sipr)</code></td>
<td></td>
<td>Set local IP address.</td>
<td>More...</td>
</tr>
<tr>
<td><code>getSIPR()</code></td>
<td></td>
<td>Get local IP address.</td>
<td>More...</td>
</tr>
</tbody>
</table>
```c
#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

Set **PTIMER** register. More...

### Define getPTIMER()

```c
#define getPTIMER() ((uint8_t)(WIZCHIP_READ(PTIMER) & 0x00FF))
```

Get **PTIMER** register. More...

### Define setPMAGIC(pmagic)

```c
#define setPMAGIC(pmagic) WIZCHIP_WRITE(PMAGIC, ((uint16_t)pmagic) & 0x00FF)
```

Set **PMAGIC** register. More...

### Define getPMAGIC()

```c
#define getPMAGIC() ((uint8_t)(WIZCHIP_READ(PMAGIC) & 0x00FF))
```

Get **PMAGIC** register. More...

### Define getPSIDR()

```c
#define getPSIDR() WIZCHIP_READ(PSIDR)
```

Get **PSID** register. More...

### Define getPDHAR(pdhar)

```c
#define getPDHAR(pdhar)
```

Get **PDHAR** register. More...

### Define getUIPR(uipr)

```c
#define getUIPR(uipr)
```

Get unreachable IP address. **UIPR**. More...

### Define getUPORTTR()

```c
#define getUPORTTR() WIZCHIP_READ(UPORTR)
```

Get **UPORTR** register. More...

### Define getFMTUR()

```c
#define getFMTUR() WIZCHIP_READ(FMTUR)
```

Get **FMTUR** register. More...

### Define getPn_BRDYR(p)

```c
#define getPn_BRDYR(p) ((uint8_t)(WIZCHIP_READ(Pn_BRDYR(p)) & 0x00FF))
```

Get **Pn_BRDYR** register. More...

### Define setPn_BRDYR(p, brdyr)

```c
#define setPn_BRDYR(p, brdyr) WIZCHIP_WRITE(Pn_BRDYR(p), brdyr & 0x00E7)
```

Set **Pn_BRDYR** register. More...
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getPn_BDPTHR(p)</code></td>
<td>WIZCHIP_READ(Pn_BDPTHR(p))</td>
<td>Get Pn_BDPTHR register.</td>
</tr>
<tr>
<td><code>setPn_BDPTHR(p, bdpthr)</code></td>
<td>WIZCHIP_WRITE(Pn_BDPTHR(p),bdpthr)</td>
<td>Set Pn_BDPTHR register.</td>
</tr>
<tr>
<td><code>getIDR()</code></td>
<td>WIZCHIP_READ(IDR)</td>
<td>Get IDR register.</td>
</tr>
</tbody>
</table>
**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setTMSR (uint8_t sn, uint8_t tmsr)</code></td>
<td>Set <strong>TMSR0 ~ TMSR7</strong> register.</td>
<td></td>
</tr>
<tr>
<td><code>uint8_t getTMSR (uint8_t sn)</code></td>
<td>Get <strong>TMSR0 ~ TMSR7</strong> register.</td>
<td></td>
</tr>
<tr>
<td><code>void setRMSR (uint8_t sn, uint8_t rmsr)</code></td>
<td>Set <strong>RMS01R ~ RMS67R</strong> register.</td>
<td></td>
</tr>
<tr>
<td><code>uint8_t getRMSR (uint8_t sn)</code></td>
<td>Get <strong>RMS01R ~ RMS67R</strong> register.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

These are functions to access common registers.
**Macro Definition Documentation**

```c
#define setIR ( ir )   WIZCHIP_WRITE(IR, ir & 0xF0FF)
```

Set Mode Register.

**Parameters**

- `ir`

Definition at line 1370 of file `w5300.h`.

```c
#define getIR ( )   (WIZCHIP_READ(IR) & 0xF0FF)
```

Get IR register.

**Returns**

- `uint8_t`. Value of IR register.

**See also**

- `getIMR()`

Definition at line 1379 of file `w5300.h`.

```c
#define setIMR ( imr )   WIZCHIP_WRITE(_IMR_, imr & 0xF0FF)
```

Set IMR register.

**Parameters**

- `uint16_t` 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()
#define setGAR (gar)

Value:

```c
{ 
    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:

```c
{ 
    (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)
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**.

```c
#define setSUBR (subr)

Value:

```c
{
  
  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:

```c
{
    (subr)[0] = (uint8_t)(WIZCHIP_READ(SUBR) >> 8); \
    (subr)[1] = (uint8_t)(WIZCHIP_READ(SUBR)); \n    (subr)[2] = (uint8_t)(WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2)) >> 8); \n    (subr)[3] = (uint8_t)(WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2))); \n}
```

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:

```c
{
    WIZCHIP_WRITE(SIPR, (((uint16_t)((sipr)[0])) << 8) + (((uint16_t)((sipr)[1])) & 0x00FF)); \n    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(SIPR,2), (((uint16_t)((sipr)[2])) << 8) + (((uint16_t)((sipr)[3])) & 0x00FF)); \n```
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`

```c
#define getSIPR( sipr)

Value:

```c
{  
   (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()`
#define setRTR ( rtr ) WIZCHIP_WRITE(_RTR_, rtr)

Set RTR register.

Parameters
   (uint16_t)rtr Value to set RTR register.

See also
   getRTR()

#define getRTR ( ) WIZCHIP_READ(_RTR_)

Get RTR register.

Returns
   uint16_t. Value of RTR register.

See also
   setRTR()

#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()
#define getRCR ((uint8_t)(WIZCHIP_READ(_RCR_) & 0x00FF))

Get RCR register.

Returns
uint8_t. Value of RCR register.

See also
setRCR()

#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()

#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

\[(\text{uint16}_t)\text{tms45r}\] 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 ( ) \text{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 ) \text{WIZCHIP\_WRITE}(TMS67R,tms67r)

Set TMS67R register.

Parameters

\[(\text{uint16}_t)\text{tms67r}\] 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()


#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 1644 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 1653 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*.

```c
#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*.

```c
#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*.

```c
#define getPATR() WIZCHIP_READ(PATR)
```
Get RATR register.

**Returns**

`uint16_t`. Value of **PATR** register.

Definition at line 1744 of file **w5300.h**.

```c
#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**.

```c
#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**.

```c
#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`.

```c
#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`.

```c
#define getPSIDR() WIZCHIP_READ(PSIDR)
```

Get **PSID** register.

**Returns**

uint16_t. Value of **PSID** register.

Definition at line 1788 of file `w5300.h`.

```c
#define getPDHAR(pdhar)
```

**Value:**
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:**

```c
{ 
    (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))); \
    (uipr)[4] = (uint8_t)(WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,4)) >> 8); \
    (uipr)[5] = (uint8_t)(WIZCHIP_READ(WIZCHIP_OFFSET_INC(UIPR,4))); \
}
```
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.

```c
#define getUPORTR () WIZCHIP_READ(UPORTR)
```

Get **UPORTR** register.

**Returns**  
uint16_t. Value of **UPORTR** register.

Definition at line 1822 of file w5300.h.

```c
#define getFMTUR () WIZCHIP_READ(FMTUR)
```

Get **FMTUR** register.

**Returns**  
uint16_t. Value of **FMTUR** register.

Definition at line 1830 of file w5300.h.

```c
#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.

```c
#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.

```c
#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.

```c
#define setPn_BDPTHR ( p,
```
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`. 

```
WIZCHIP_WRITE(Pn_BDPTHR(p),bdpthr)
```
Function Documentation

```c
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()`

```c
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()`

```c
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()`

```c
uint8_t getRMSR ( uint8_t sn )
```

Get **RMS01R ~ RMS67R** register.

**Parameters**

- `(uint8_t)sn` Socket number. It should be 0 ~ 7.

**Returns**

- `uint8_t`. Value of **RMSR0 ~ RMSR7** register.

**See also**

- `setRMSR()`
Socket APIs

Socket register access functions
W5300 » WIZCHIP I/O functions

These are functions to access socket registers. More...
### Macros

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define setSn_MR(sn, mr) WIZCHIP_WRITE(Sn_MR(sn), mr)</code></td>
<td>Set Sn_MR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_MR(sn) WIZCHIP_READ(Sn_MR(sn))</code></td>
<td>Get Sn_MR register. More...</td>
</tr>
<tr>
<td><code>#define setSn_CR(sn, cr) WIZCHIP_WRITE(Sn_CR(sn), ((uint16_t)cr) &amp; 0x00FF)</code></td>
<td>Set Sn_CR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_CR(sn) ((uint8_t)WIZCHIP_READ(Sn_CR(sn)))</code></td>
<td>Get Sn_CR register. More...</td>
</tr>
<tr>
<td><code>#define setSn_IMR(sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), ((uint16_t)imr) &amp; 0x00FF)</code></td>
<td>Set Sn_IMR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_IMR(sn) ((uint8_t)WIZCHIP_READ(Sn_IMR(sn)))</code></td>
<td>Get Sn_IMR register. More...</td>
</tr>
<tr>
<td><code>#define setSn_IR(sn, ir) WIZCHIP_WRITE(Sn_IR(sn), ((uint16_t)ir) &amp; 0x00FF)</code></td>
<td>Set Sn_IR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_IR(sn) ((uint8_t)WIZCHIP_READ(Sn_IR(sn)))</code></td>
<td>Get Sn_IR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_SSR(sn) ((uint8_t)WIZCHIP_READ(Sn_SR(sn)))</code></td>
<td>Get Sn_SR register. More...</td>
</tr>
<tr>
<td><code>#define setSn_PORTR(sn, port) WIZCHIP_WRITE(Sn_PORTR(sn), port)</code></td>
<td>Set Sn_PORTR register. More...</td>
</tr>
<tr>
<td><code>#define getSn_PORTR(sn, port) WIZCHIP_READ(Sn_PORTR(sn))</code></td>
<td>Get Sn_PORTR register. More...</td>
</tr>
</tbody>
</table>
```c
#define setSn_DHAR(sn, dhar) Set Sn_DHAR register. More...
#define getSn_DHAR(sn, dhar) Get Sn_MMR register. More...
#define setSn_DPORTR(sn, dport) WIZCHIP_WRITE(Sn_DPORTR(sn), dport) 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), mss) 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_KPALVTR(sn), WIZCHIP_READ(Sn_KPALVTR(sn)) & 0x00FF) | (((uint16_t)kpalvt) & 0x00FF) Set Sn_KPALVTR register. More...
#define getSn_KPALVTR(sn) ((uint8_t)(WIZCHIP_READ(Sn_KPALVTR(sn)) & 0xFF) Get Sn_KPALVTR register. More...
#define setSn_PROTOR(sn, proto) WIZCHIP_WRITE(Sn_PROTOR(sn), WIZCHIP_READ(Sn_PROTOR(sn)) & 0xFF00) | (((uint16_t)proto) & 0x00FF) Set Sn_PROTOR register. More...
```
#define getSn_PROTOR(sn)  ((uint8_t)WIZCHIP_READ(Sn_PROTOR)
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_WRSR) << 16) +
((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WRSR & 0x0000FFFF)))
Get Sn_TX_WRSR register. More...

#define setSn_TX_FIFOR(sn, txfifo) WIZCHIP_WRITE(Sn_TX_FIFOR)
Set Sn_TX_FIFOR register. More...

#define getSn_RX_FIFOR(sn) WIZCHIP_READ(Sn_RX_FIFOR)
Get Sn_RX_FIFOR register. More...

#define setSn_TOSR(sn, tos) WIZCHIP_WRITE(Sn_TOSR(sn), ((uint16_t)tos) & 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), ((uint16_t)ttl) & 0x00FF)
Set Sn_TTLR register. More...

#define getSn_TTLR(sn) ((uint8_t)WIZCHIP_READ(Sn_TTLR(sn)))
Get Sn_TTLR register. More...

#define setSn_FRAGR(sn, frag) WIZCHIP_WRITE(Sn_FRAGR(sn), >>8))
Set Sn_FRAGR register. More...


```c
#define getSn_FRAGR(sn)  (WIZCHIP_READ(Sn_FRAG(sn)) << 8)
Get Sn_FRAGR register. More...
```
## Functions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint32_t</strong></td>
<td><strong>getSn_TX_FSR</strong> (uint8_t sn)</td>
</tr>
<tr>
<td></td>
<td>Get <em>Sn_TX_FSR</em> register. More...</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>uint32_t</strong></td>
<td><strong>getSn_RX_RSR</strong> (uint8_t sn)</td>
</tr>
<tr>
<td></td>
<td>Get <em>Sn_RX_RSR</em> register. More...</td>
</tr>
</tbody>
</table>
Detailed Description

These are functions to access *socket registers*.
Macro Definition Documentation

```c
#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.

```c
#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.
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*.

```c
#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*.

```c
#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.
setSn_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)

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,
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 2002 of file **w5300.h**.

```c
#define getSn_DHAR(   sn,
                    dhar
                )
```

Value:
```
{ \
    (dhar)[0] = (uint8_t)
```
Get $\text{Sn\_MR}$ register.

**Parameters**

- $(\text{uint8\_t})\text{sn}$ Socket number. It should be $0 \sim 7$.
- $(\text{uint8\_t}* )\text{dhar}$ Pointer variable to get socket $n$ destination hardware address. It should be allocated 6 bytes.

See also

- $\text{setSn\_DHAR()}$

Definition at line 2015 of file $\text{w5300.h}$.

```c
#define setSn_DPORTR (sn, dport)  
       WIZCHIP_WRITE(Sn_DPORTR(sn),dport)
```

Set $\text{Sn\_DPORTR}$ 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)
Set **Sn_DIPR** register.

**Parameters**
- **(uint8_t)**<em>sn</em>  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**.

```c
#define getSn_DIPR( sn, dipr )
{
    (dipr)[0] = (uint8_t)(WIZCHIP_READ(Sn_DIPR(sn)) >> 8);  \n    (dipr)[1] = (uint8_t)WIZCHIP_READ(Sn_DIPR(sn));  \n    (dipr)[2] = (uint8_t)(WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2)) >> 8);  \n    (dipr)[3] = (uint8_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DIPR(sn),2))  \n}
```
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().

#define getSn_KPALVTR ( (uint8_t) (((uint8_t) (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 ( (((uint32_t)WIZCHIP_READ(Sn_TX_WRSR(sn)) & 0x0000FFFF))

#define getSn_TX_WRSR ( (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn), (uint16_t)txwrs))) & 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.

```c
#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.

```c
#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*.

```c
#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*.

```c
#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()**
#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 2211 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 2222 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  getSn_FRAGR  ( (WIZCHIP_READ(Sn_FRAG(sn)) <<
      (  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

```c
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`.

```c
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`.

Generated on Wed May 4 2016 16:44:00 for Socket APIs by doxygen 1.8.9.1
### Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZCHIP register</td>
<td>W5300</td>
<td>Modules</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHIZCHIP register defines register group of **W5300**. More...
## Modules

<table>
<thead>
<tr>
<th>Common register</th>
<th>Common register group</th>
</tr>
</thead>
<tbody>
<tr>
<td>It set the basic for the networking.</td>
<td></td>
</tr>
<tr>
<td>It set the configuration such as interrupt, network information, ICMP, etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Socket register</th>
<th>Socket register group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket register configures and control SOCKETn which is necessary to data communication.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

WHIZCHIP register defines register group of **W5300**.

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

Common register

W5300 » WIZCHIP register

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

<table>
<thead>
<tr>
<th>#define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR (<em>WIZCHIP_IO_BASE</em>)</td>
<td>Mode Register address(R/W) MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...</td>
</tr>
<tr>
<td>IR (<em>W5300_IO_BASE</em> + 0x02)</td>
<td>Interrupt Register(R/W) More...</td>
</tr>
<tr>
<td><em>IMR</em> (<em>W5300_IO_BASE</em> + 0x04)</td>
<td>Socket Interrupt Mask Register(R/W) More...</td>
</tr>
<tr>
<td>SHAR (<em>W5300_IO_BASE</em> + 0x08)</td>
<td>Source MAC Register address(R/W) More...</td>
</tr>
<tr>
<td>GAR (<em>W5300_IO_BASE</em> + 0x10)</td>
<td>Gateway IP Register address(R/W) More...</td>
</tr>
<tr>
<td>SUBR (<em>W5300_IO_BASE</em> + 0x14)</td>
<td>Subnet mask Register address(R/W) More...</td>
</tr>
<tr>
<td>SIPR (<em>W5300_IO_BASE</em> + 0x18)</td>
<td>Source IP Register address(R/W) More...</td>
</tr>
<tr>
<td><em>RTR</em> (<em>W5300_IO_BASE</em> + 0x1C)</td>
<td>Timeout register address( 1 is 100us )(R/W) More...</td>
</tr>
<tr>
<td><em>RCR</em> (<em>W5300_IO_BASE</em> + 0x1E)</td>
<td>Retry count register(R/W) More...</td>
</tr>
<tr>
<td>TMS01R (<em>W5300_IO_BASE</em> + 0x20)</td>
<td>TX memory size of socket 0 &amp; 1. More...</td>
</tr>
<tr>
<td>TMS23R (TMS01R + 2)</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>Equation</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>TX memory size of socket 2 &amp; 3</td>
<td>More...</td>
</tr>
<tr>
<td>#define TMS45R (TMS01R + 4)</td>
<td>TX memory size of socket 4 &amp; 5</td>
</tr>
<tr>
<td>#define TMS67R (TMS01R + 6)</td>
<td>TX memory size of socket 6 &amp; 7</td>
</tr>
<tr>
<td>#define TMSR0 TMS01R</td>
<td>TX memory size of socket 0</td>
</tr>
<tr>
<td>#define TMSR1 (TMSR0 + 1)</td>
<td>TX memory size of socket 1</td>
</tr>
<tr>
<td>#define TMSR2 (TMSR0 + 2)</td>
<td>TX memory size of socket 2</td>
</tr>
<tr>
<td>#define TMSR3 (TMSR0 + 3)</td>
<td>TX memory size of socket 3</td>
</tr>
<tr>
<td>#define TMSR4 (TMSR0 + 4)</td>
<td>TX memory size of socket 4</td>
</tr>
<tr>
<td>#define TMSR5 (TMSR0 + 5)</td>
<td>TX memory size of socket 5</td>
</tr>
<tr>
<td>#define TMSR6 (TMSR0 + 6)</td>
<td>TX memory size of socket 6</td>
</tr>
<tr>
<td>#define TMSR7 (TMSR0 + 7)</td>
<td>TX memory size of socket 7</td>
</tr>
<tr>
<td>#define RMS01R (<em>W5300_IO_BASE</em> + 0x28)</td>
<td>RX memory size of socket 0 &amp; 1</td>
</tr>
</tbody>
</table>
#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)
<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPPoE Authentication Type register.</td>
<td>More...</td>
</tr>
<tr>
<td>#define PTIMER (<em>W5300_IO_BASE</em> + 0x36)</td>
<td>PPP Link Control Protocol Request Timer Register. More...</td>
</tr>
<tr>
<td>#define PMAGICR (<em>W5300_IO_BASE</em> + 0x38)</td>
<td>PPP LCP magic number register. More...</td>
</tr>
<tr>
<td>#define PSIDR (<em>W5300_IO_BASE</em> + 0x3C)</td>
<td>PPPoE session ID register. More...</td>
</tr>
<tr>
<td>#define PDHAR (<em>W5300_IO_BASE</em> + 0x40)</td>
<td>PPPoE destination hardware address register. More...</td>
</tr>
<tr>
<td>#define UIPR (<em>W5300_IO_BASE</em> + 0x48)</td>
<td>Unreachable IP address register. More...</td>
</tr>
<tr>
<td>#define UPORTR (<em>W5300_IO_BASE</em> + 0x4C)</td>
<td>Unreachable port number register. More...</td>
</tr>
<tr>
<td>#define FMTUR (<em>W5300_IO_BASE</em> + 0x4E)</td>
<td>Fragment MTU register. More...</td>
</tr>
<tr>
<td>#define Pn_BRDYR(n) (<em>W5300_IO_BASE</em> + 0x60 + n*4)</td>
<td>PIN 'BRDYn' configure register. More...</td>
</tr>
<tr>
<td>#define Pn_BDPTHTR(n) (<em>W5300_IO_BASE</em> + 0x60 + n*4 + 2)</td>
<td>PIN 'BRDYn' buffer depth Register. More...</td>
</tr>
<tr>
<td>#define IDR (<em>W5300_IO_BASE</em> + 0xFE)</td>
<td>W5300 identification register. More...</td>
</tr>
</tbody>
</table>
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.
- **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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Definition at line</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>MR_DBW : Data bus width</td>
<td>0</td>
<td>224</td>
</tr>
<tr>
<td>14</td>
<td>MR_MPF : Received a Pause</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Frame from MAC layer</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MR_WDF : Write Data</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Fetch time (When CS signal</td>
<td>FS</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>is low, W5300</td>
<td>RDF</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Fetch a written data by Host</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PLL_CLK * MR_WDF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MR_RDH : Read Data Hold</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>time (0 : No use data hold</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>time, 1 : Use data hold</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>time, 2 PLL_CLK)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MR_FS : FIFO Swap</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(0 : Disable Swap, 1 :</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Enable Swap)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>MR_RST : Reset</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MR_WOL : Wake on LAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MR_PB : Ping block</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MR_PPPOE : PPPoE mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MR_FARP : Force ARP mode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 written to by the host. If IR is not equal to 0x0000 INTn PIN is asserted to low until 0x0000

Each bit of IR defined as follows.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>IR_IPCF</td>
</tr>
<tr>
<td>14</td>
<td>IR_DPUR</td>
</tr>
<tr>
<td>13</td>
<td>IR_PPPT</td>
</tr>
<tr>
<td>12</td>
<td>IR_FMTU</td>
</tr>
<tr>
<td>11</td>
<td>S7_INT(n)</td>
</tr>
<tr>
<td>10</td>
<td>S6_INT(n)</td>
</tr>
<tr>
<td>9</td>
<td>S5_INT(n)</td>
</tr>
<tr>
<td>8</td>
<td>S4_INT(n)</td>
</tr>
<tr>
<td>7</td>
<td>S3_INT(n)</td>
</tr>
<tr>
<td>6</td>
<td>S2_INT(n)</td>
</tr>
<tr>
<td>5</td>
<td>S1_INT(n)</td>
</tr>
</tbody>
</table>

- 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.

```c
#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 set 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.

```c
#define _RCR_ (_W5300_IO_BASE_ + 0x1E)
```

Retry count register (R/W)

*RCR* configures the number of times 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.

```c
#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 transmission 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**.

```c
#define RMSR3  (RMSR0 + 3)
```
RX memory size of socket 3.
refer to **RMS01R**
Definition at line 452 of file **w5300.h**.

```c
#define RMSR4  (RMSR0 + 4)
```
RX memory size of socket 4.
refer to **RMS01R**
Definition at line 459 of file **w5300.h**.

```c
#define RMSR5  (RMSR0 + 5)
```
RX memory size of socket 5.
refer to **RMS01R**
Definition at line 466 of file **w5300.h**.

```c
#define RMSR6  (RMSR0 + 6)
```
RX memory size of socket 6.
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.
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 \texttt{Pn_BDPTHR}, or received buffer size of RX memory is same or bigger than the \texttt{Pn_BDPTHR}, PIN "BRDYn" is signaled.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved, Read as 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>PEN</td>
<td>MT</td>
<td>PPL</td>
<td>Reserved</td>
<td>SN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \texttt{Pn_PEN} Enable PIN 'BRDYn' (0 : Disable, 1 : Enable)
- \texttt{Pn_MT} Monitoring Memory type (0 : RX memory, 1 : TX Memory)
- \texttt{Pn_PPL} PIN Polarity bit of Pn_BRDYR. (0 : Low sensitive, 1 : High sensitive)
- \texttt{Pn_SN(n)} Monitoring SOCKET number of Pn_BRDYR

Definition at line 584 of file \texttt{w5300.h}.

PIN 'BRDYn' buffer depth Register.

It configures buffer depth of PIN "BRDYn". When monitoring TX memory and \texttt{Sn_TX_FSR} is same or bigger than \texttt{Pn_BDPTHR}, the PIN "BRDYn" is signaled. When monitoring RX memory and if \texttt{Sn_RX_RSR} is same or bigger than \texttt{Pn_BDPTHR}, PIN "BRDYn" is signaled. The value for \texttt{Pn_BDPTHR} can’t exceed TX/RX memory size allocated by TMSR or RMSR such like as \texttt{TMS01R} or \texttt{RMS01R}.

Definition at line 594 of file \texttt{w5300.h}. 

\begin{verbatim}
#define Pn_BRDYR ( n ) (_W5300_IO_BASE_ + 0x60 + n*4)
#define Pn_BDPTHR ( n ) (_W5300_IO_BASE_ + 0x60 + n*4 + 2)
\end{verbatim}
#define IDR (_W5300_IO_BASE_ + 0xFE)

W5300 identification register.

Read Only. 0x5300.

Definition at line 601 of file w5300.h.
Socket register

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

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Sn_MR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x00)</code>  Socket Mode register(R/W)</td>
</tr>
<tr>
<td><code>Sn_CR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x02)</code>  Socket command register(R/W)</td>
</tr>
<tr>
<td><code>Sn_IMR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x04)</code> socket interrupt mask register(R)</td>
</tr>
<tr>
<td><code>Sn_IR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x06)</code>  Socket interrupt register(R)</td>
</tr>
<tr>
<td><code>Sn_SSR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x08)</code>  Socket status register(R)</td>
</tr>
<tr>
<td><code>Sn_PORTR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0A)</code> source port register(R/W)</td>
</tr>
<tr>
<td><code>Sn_DHAR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0C)</code> Peer MAC register address(R/W)</td>
</tr>
<tr>
<td><code>Sn_DPORTR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x12)</code> Peer port register address(R/W)</td>
</tr>
<tr>
<td><code>Sn_DIPR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x14)</code></td>
</tr>
</tbody>
</table>

More information available...
<table>
<thead>
<tr>
<th>#define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MSSR(n)</td>
<td>Maximum Segment Size(Sn_MSSR0) register address(R/W)</td>
</tr>
<tr>
<td>Sn_KPALVTR(n)</td>
<td>Keep Alive Timer register(R/W)</td>
</tr>
<tr>
<td>Sn_PROTOR(n)</td>
<td>IP Protocol(PROTO) Register(R/W)</td>
</tr>
<tr>
<td>Sn_TOSR(n)</td>
<td>IP Type of Service(TOS) Register(R/W)</td>
</tr>
<tr>
<td>Sn_TTLR(n)</td>
<td>IP Time to live(TTL) Register(R/W)</td>
</tr>
<tr>
<td>Sn_TX_WRSR(n)</td>
<td>SOCKETn TX write size register(R/W)</td>
</tr>
<tr>
<td>Sn_TX_FSR(n)</td>
<td>Transmit free memory size register(R)</td>
</tr>
<tr>
<td>Sn_FRAGR(n)</td>
<td>Fragment field value in IP header register(R/W)</td>
</tr>
<tr>
<td>Sn_TX_FIFOR(n)</td>
<td>SOCKETn TX FIFO register.</td>
</tr>
</tbody>
</table>
#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 (_W5300_IO_BASE_ + Sn_MR ( n ) 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.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved. Read as 0</td>
<td>ALIGN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MULTI</td>
<td>MF</td>
<td>ND/IGMPv</td>
<td>Reserved</td>
<td>PROTOCOL[3:0]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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 multicasting. (0 : Version 2, 1 : Version 2)
- **PROTOCOL[3:0]**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TCP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>UDP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>IPCRAW</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PPPoE</td>
</tr>
</tbody>
</table>
- **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.

```c
#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(r**Server mode**)
- **Sn_CR_CONNECT**: Send connection request in TCP mode(r**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`.

```c
#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`.

```c
#define Sn_IR (n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 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

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved. Read as 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>PRECV</td>
<td>PFAIL</td>
<td>PNEXT</td>
<td>SENDOK</td>
<td>TIMEOUT</td>
<td>RECV</td>
<td>DISCON</td>
<td>C</td>
</tr>
</tbody>
</table>

- **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`.

```c
#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`.

```c
#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`.

```c
#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`.

```c
#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 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 753 of file w5300.h.

```c
#define Sn_DIPR (_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 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 765 of file w5300.h.

```c
#define Sn_MSSR (_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.

```c
#define Sn_KPALVTR (_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**.

```c
#define Sn_PROTOR(n) Sn_KPALVTR(n)
```

**Sn_PROTOR** 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**.

```c
#define Sn_TOSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1C)
```

**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**.

```c
#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**.

```c
#define Sn_TTLR (_W5300_IO_BASE_ + Sn_TX_WRSR ( n ) 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**.

```c
#define Sn_TX_WRSR (_W5300_IO_BASE_ + Sn_TX_FSR ( n ) 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 ( _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 ( _W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x2E)

SOCKET n TX FIFO regisiter.

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 ( _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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W5500</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHIZCHIP register defines and I/O functions of **W5500**. More...
## Modules

<table>
<thead>
<tr>
<th><strong>WIZCHIP I/O functions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This supports the basic I/O functions for <strong>WIZCHIP register</strong>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WIZCHIP register</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>WHIZCHIP register defines register group of <strong>W5500</strong>.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>
| **WIZCHIP I/O functions**

W5500

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

<table>
<thead>
<tr>
<th><strong>Basic I/O function</strong></th>
<th>These are basic input/output functions to read values from register or write values to register.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common register access functions</strong></td>
<td>These are functions to access common registers.</td>
</tr>
<tr>
<td><strong>Socket register access functions</strong></td>
<td>These are functions to access socket registers.</td>
</tr>
</tbody>
</table>
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(), setPMAGIC(),
       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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
</table>

## Basic I/O function

W5500 > WIZCHIP I/O functions

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td><strong>WIZCHIP_READ</strong> (uint32_t AddrSel)</td>
<td>It reads 1 byte value from a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE</strong> (uint32_t AddrSel, uint8_t wb)</td>
<td>It writes 1 byte value to a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_READ_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It reads sequence data from registers.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It writes sequence data to registers.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_send_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to internal TX memory.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_recv_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to your buffer from internal RX memory.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_recv_ignore</strong> (uint8_t sn, uint16_t len)</td>
<td>It discards the received data in RX memory.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

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

```c
uint8_t WIZCHIP_READ ( uint32_t AddrSel )
```

It reads 1 byte value from a register.

**Parameters**

- **AddrSel** Register address

**Returns**

- The value of register

```c
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

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

It reads sequence data from registers.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddrSel</code></td>
<td>Register address</td>
</tr>
<tr>
<td><code>pBuf</code></td>
<td>Pointer buffer to read data</td>
</tr>
<tr>
<td><code>len</code></td>
<td>Data length</td>
</tr>
</tbody>
</table>

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

It writes sequence data to registers.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddrSel</code></td>
<td>Register address</td>
</tr>
<tr>
<td><code>pBuf</code></td>
<td>Pointer buffer to write data</td>
</tr>
<tr>
<td><code>len</code></td>
<td>Data length</td>
</tr>
</tbody>
</table>

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sn</code></td>
<td>Socket number. It should be 0 ~ 7.</td>
</tr>
<tr>
<td><code>wizdata</code></td>
<td>Pointer buffer to write data</td>
</tr>
<tr>
<td><code>len</code></td>
<td>Data length</td>
</tr>
</tbody>
</table>

See also
### wiz_recv_data()

```c
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 reads 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()`

### wiz_recv_ignore()

```c
void wiz_recv_ignore ( uint8_t sn,
                       uint16_t len
                    )
```

It discards 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

W5500 » WIZCHIP I/O functions

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

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define getSHAR(shar)</td>
<td>Get local MAC address.</td>
</tr>
<tr>
<td>#define setMR(mr)</td>
<td>Set Mode Register.</td>
</tr>
<tr>
<td>#define getMR()</td>
<td>Get Mode Register.</td>
</tr>
<tr>
<td>#define setGAR(gar)</td>
<td>Set gateway IP address.</td>
</tr>
<tr>
<td>#define getGAR(gar)</td>
<td>Get gateway IP address.</td>
</tr>
<tr>
<td>#define setSUBR(subr)</td>
<td>Set subnet mask address.</td>
</tr>
<tr>
<td>#define getSUBR(subr)</td>
<td>Get subnet mask address.</td>
</tr>
<tr>
<td>#define setSHAR(shar)</td>
<td>Set local MAC address.</td>
</tr>
<tr>
<td>#define getSHAR(shar)</td>
<td>Get local MAC address.</td>
</tr>
<tr>
<td>#define setSIPR(sipr)</td>
<td>Set local IP address.</td>
</tr>
<tr>
<td>#define getSIPR(sipr)</td>
<td>Get local IP address.</td>
</tr>
</tbody>
</table>
#define setINTLEVEL(intlevel)
Set INTLEVEL register. More...

#define getINTLEVEL()

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...
<table>
<thead>
<tr>
<th>#define</th>
<th>getPMRU() (((uint16_t)WIZCHIP_READ(PMRU) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get PMRU register. More...</td>
</tr>
<tr>
<td>#define</td>
<td>getUIPR(uipr) WIZCHIP_READ_BUF(UIPR,uipr,4)</td>
</tr>
<tr>
<td></td>
<td>Get unreachable IP address. More...</td>
</tr>
<tr>
<td>#define</td>
<td>getUPORTR() (((uint16_t)WIZCHIP_READ(UPORTR) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1)))</td>
</tr>
<tr>
<td></td>
<td>Get UPORTR register. More...</td>
</tr>
<tr>
<td>#define</td>
<td>setPHYCFGR(phycfgr) WIZCHIP_WRITE(PHYCFGR, phycfgr)</td>
</tr>
<tr>
<td></td>
<td>Set PHYCFGR register. More...</td>
</tr>
<tr>
<td>#define</td>
<td>getPHYCFGR() WIZCHIP_READ(PHYCFGR)</td>
</tr>
<tr>
<td></td>
<td>Get PHYCFGR register. More...</td>
</tr>
<tr>
<td>#define</td>
<td>getVersionR() WIZCHIP_READ(VERSIONR)</td>
</tr>
<tr>
<td></td>
<td>Get VERSIONR register. More...</td>
</tr>
</tbody>
</table>
Detailed Description

These are functions to access common registers.
Macro Definition Documentation

```c
#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()
#define setSHAR(shar)  
WIZCHIP_WRITE_BUF(SHAR, 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()

#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

getSHAR()

#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:

```c
{
    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()
#define getINTLEVEL ( ) (((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL))

Get INTLEVEL register.

Returns
  uint16_t. Value of INTLEVEL register.

See also
  setINTLEVEL()

Definition at line 1339 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 1356 of file w5500.h.

#define getIR ( ) (WIZCHIP_READ(IR) & 0xF0)

Get IR register.

Returns
  uint8_t. Value of IR register.

See also
  setIR()
#define setIMR (imr)  WIZCHIP_WRITE(_IMR_, imr)

Set IMR register.

Parameters

\begin{verbatim}
(uint8_t)imr Value to set IMR register.
\end{verbatim}

See also

getIMR()

#define getIMR ()  WIZCHIP_READ(_IMR_)

Get IMR register.

Returns

\begin{verbatim}
uint8_t. Value of IMR register.
\end{verbatim}

See also

setIMR()

#define setSIR (sir)  WIZCHIP_WRITE(SIR, sir)

Set SIR register.

Parameters

\begin{verbatim}
(uint8_t)sir Value to set SIR register.
\end{verbatim}

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()
**#define setRTR ( rtr )**

**Value:**

```c
{
    WIZCHIP_WRITE(_RTR_, (uint8_t) (rtr >> 8)); \n
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1), (uint8_t) rtr); \n}
```

Set *RTR* register.

**Parameters**

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

**See also**

- `getRTR()`

Definition at line 1426 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 1435 of file `w5500.h`.
```c
#define setRCR ( rcr ) WIZCHIP_WRITE(_RCR_, rcr)
```

Set RCR register.

**Parameters**

`r8_t rcr` Value to set RCR register.

**See also**

`getRCR()`

Definition at line 1461 of file `w5500.h`.

```c
#define getRCR ( ) WIZCHIP_READ(_RCR_)
```

Get RCR register.

**Returns**

`r8_t`. Value of RCR register.

**See also**

`setRCR()`

Definition at line 1470 of file `w5500.h`.

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

Set PTIMER register.

**Parameters**

`r8_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, ( 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:

```c
{{
   WIZCHIP_WRITE(PSID, (uint8_t) (psid >> 8)); \\
   WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PSID, 1), (uint8_t) psid);
}}
```
Set **PSID** register.

**Parameters**

\( (\text{uint16}_t)psid \) 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**

\( \text{uint16}_t \). 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

\texttt{(uint16\_t)pmru} Value to set \texttt{PMRU} register.

See also

\texttt{getPMRU()}

Definition at line \texttt{1561} of file \texttt{w5500.h}.

\begin{verbatim}
#define getPMRU () (((uint16_t) WIZCHIP_READ(PMRU) << 8) +
 getPMRU ( ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))
\end{verbatim}

Get \texttt{PMRU} register.

Returns

\texttt{uint16\_t}. Value of \texttt{PMRU} register.

See also

\texttt{setPMRU()}

Definition at line \texttt{1577} of file \texttt{w5500.h}.

\begin{verbatim}
#define getUIPR ( uipr ) WIZCHIP_READ_BUF(UIPR,uipr,4)
\end{verbatim}

Get unreachable IP address.

Parameters

\texttt{(uint8\_t*)uipr} Pointer variable to get unreachable IP address. It should be allocated 4 bytes.

Definition at line \texttt{1590} of file \texttt{w5500.h}.

\begin{verbatim}
#define getUPORTR () (((uint16_t) WIZCHIP_READ(UPORTR) << 8) +
 getUPORTR ( ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1))
\end{verbatim}

Get \texttt{UPORTR} register.
Returns

`uint16_t` Value of **UPORTR** register.

Definition at line **1603** of file **w5500.h**.

```c
#define setPHYCFGR(phycfgr) WIZCHIP_WRITE(PHYCFGR, (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()**.

```c
#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
W5500 » WIZCHIP I/O functions

These are functions to access **socket registers**. 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 & 0x1F))
Set Sn_IMR register. More...

#define getSn_IMR(sn) (WIZCHIP_READ(Sn_IMR(sn)) & 0x1F)
Get Sn_IMR register. More...

#define setSn_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)) & 0x1F) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn))))
Get Sn_PORT register. More...
<table>
<thead>
<tr>
<th>Define</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setSn_DHAR(sn, dhar)</code></td>
<td><code>WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6)</code></td>
<td>Set Sn_DHAR register. More...</td>
</tr>
<tr>
<td><code>getSn_DHAR(sn, dhar)</code></td>
<td><code>WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar, 6)</code></td>
<td>Get Sn_DHAR register. More...</td>
</tr>
<tr>
<td><code>setSn_DIPR(sn, dipr)</code></td>
<td><code>WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)</code></td>
<td>Set Sn_DIPR register. More...</td>
</tr>
<tr>
<td><code>getSn_DIPR(sn, dipr)</code></td>
<td><code>WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 4)</code></td>
<td>Get Sn_DIPR register. More...</td>
</tr>
<tr>
<td><code>setSn_DPORT(sn, dport)</code></td>
<td></td>
<td>Set Sn_DPORT register. More...</td>
</tr>
<tr>
<td><code>getSn_DPORT(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn), 1)))</code></td>
<td>Get Sn_DPORT register. More...</td>
</tr>
<tr>
<td><code>setSn_MSSR(sn, mss)</code></td>
<td></td>
<td>Set Sn_MSSR register. More...</td>
</tr>
<tr>
<td><code>getSn_MSSR(sn)</code></td>
<td><code>(((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn), 1)))</code></td>
<td>Get Sn_MSSR register. More...</td>
</tr>
<tr>
<td><code>setSn_TOS(sn, tos)</code></td>
<td><code>WIZCHIP_WRITE(Sn_TOS(sn), tos)</code></td>
<td>Set Sn_TOS register. More...</td>
</tr>
<tr>
<td><code>getSn_TOS(sn)</code></td>
<td><code>WIZCHIP_READ(Sn_TOS(sn))</code></td>
<td>Get Sn_TOS register. More...</td>
</tr>
</tbody>
</table>
```c
#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_SIZE(sn))
    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(sn))
    Get Sn_TXBUF_SIZE register. More...

#define getSn_TX_RD(sn)  (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 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(sn)) << 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(sn))<<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(sn))<<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),1)))
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint16_t</td>
<td><code>getSn_TX_FSR (uint8_t sn)</code></td>
<td>Get <strong>Sn_TX_FSR</strong> register.</td>
<td>More...</td>
</tr>
<tr>
<td>uint16_t</td>
<td><code>getSn_RX_RSR (uint8_t sn)</code></td>
<td>Get <strong>Sn_RX_RSR</strong> register.</td>
<td>More...</td>
</tr>
</tbody>
</table>
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,`
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**.

```c
#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**.

```c
#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
(uuint8_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
(uuint8_t)sn  Socket number. It should be 0 ~ 7.
(uuint8_t)imr Value to set Sn_IMR

See also
getSn_IMR()
#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:

```c
{
    \n```
```c
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`.

```c
#define getSn_PORT(sn) (((uint16_t)WIZCHIP_READ(Sn_PORT(sn))  
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)))
```

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`.

```c
#define setSn_DHAR(sn, dhar
```

- **setSn_DHAR**
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`.

```c
#define getSn_DHAR
```

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`.

```c
#define
```
setSn_DIPR

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**.

```c
#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**.

```c
#define setSn_DPORT (sn, dipr)
WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)
```
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(  
    (uint16_t)WIZCHIP_READ(Sn_DPORT(sn))
getSn_DPORT(sn) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_I

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:

```c
{
  \n  WIZCHIP_WRITE(Sn_MSSR(sn),
  (uint8_t)(mss>>8)); \n
  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)
  (uint8_t) mss); \n}
```

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 
)

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.

```c
#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.

```c
#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.

```c
#define setSn_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**.

```c
#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**.

```c
#define setSn_RXBUF_SIZE(sn, rxbufsize) WIZCHIP_WRITE(Sn_RXBUF_SIZE(sn), rxbufsize)
```

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 getSn_TX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + 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:

```c
{
    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`.

```c
#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)  
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1  

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.

```c
#define getSn_RX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(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.

```c
#define setSn_FRAG(sn, frag)
{
  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()`


```c
#define getSn_FRAG ( sn ) WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FF + Sn_FRAG * 2) + Sn_FRAG * 8) + WIZCHIP_READ(Sn_FRAG(sn)) << 8
```

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*.

```c
#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*.
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`. 

```
getSn_KPALVTR ( sn )  WIZCHIP_READ(Sn_KPALVTR(sn))
```
Function Documentation

```c
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`.

```c
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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIZCHIP register</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>W5500</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WIZCHIP 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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Macros</th>
</tr>
</thead>
</table>

**Common register**

W5500 » WIZCHIP register

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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Address Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MR</code> (<code>_W5500_IO_BASE_ + (0x0000 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Mode Register address (R/W)</td>
<td>MR is used for S/W reset, ping block mode, PPPoE mode and etc. More...</td>
</tr>
<tr>
<td><code>GAR</code> (<code>_W5500_IO_BASE_ + (0x0001 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Gateway IP Register address (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SUBR</code> (<code>_W5500_IO_BASE_ + (0x0005 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Subnet mask Register address (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SHAR</code> (<code>_W5500_IO_BASE_ + (0x0009 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Source MAC Register address (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>SIPR</code> (<code>_W5500_IO_BASE_ + (0x000F &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Source IP Register address (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>INTLEVEL</code> (<code>_W5500_IO_BASE_ + (0x0013 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Set Interrupt low level timer register address (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>IR</code> (<code>_W5500_IO_BASE_ + (0x0015 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Interrupt Register (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td><code>IMR</code> (<code>_W5500_IO_BASE_ + (0x0016 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code>)</td>
<td>Interrupt mask register (R/W)</td>
<td>More...</td>
</tr>
<tr>
<td>Definition</td>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>#define</code></td>
<td>SIR</td>
<td><code>_W5500_IO_BASE_ + (0x0017 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>SIMR</td>
<td><code>_W5500_IO_BASE_ + (0x0018 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td><em>RTR</em></td>
<td><code>_W5500_IO_BASE_ + (0x0019 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td><em>RCR</em></td>
<td><code>_W5500_IO_BASE_ + (0x001B &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>PTIMER</td>
<td><code>_W5500_IO_BASE_ + (0x001C &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>PMAGIC</td>
<td><code>_W5500_IO_BASE_ + (0x001D &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>PHAR</td>
<td><code>_W5500_IO_BASE_ + (0x001E &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>PSID</td>
<td><code>_W5500_IO_BASE_ + (0x0024 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td><code>#define</code></td>
<td>PMRU</td>
<td><code>_W5500_IO_BASE_ + (0x0026 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3)</code></td>
</tr>
<tr>
<td>Define</td>
<td>Address Expression</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>#define</td>
<td>UIPR (<em>W5500_IO_BASE</em> + (0x0028 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3))</td>
<td>Unreachable IP register address in UDP mode (R)</td>
</tr>
<tr>
<td>#define</td>
<td>UPORTR (<em>W5500_IO_BASE</em> + (0x002C &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3))</td>
<td>Unreachable Port register address in UDP mode (R)</td>
</tr>
<tr>
<td>#define</td>
<td>PHYCFGR (<em>W5500_IO_BASE</em> + (0x002E &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3))</td>
<td>PHY Status Register (R/W)</td>
</tr>
<tr>
<td>#define</td>
<td>VERSIONR (<em>W5500_IO_BASE</em> + (0x0039 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3))</td>
<td>Chip version register address (R)</td>
</tr>
</tbody>
</table>
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.
- PHYCFG, VERSIONR : etc.
### Macro Definition Documentation

```c
#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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MR_RST</td>
<td>Reset</td>
</tr>
<tr>
<td>6</td>
<td>MR_WOL</td>
<td>Wake on LAN</td>
</tr>
<tr>
<td>5</td>
<td>MR_PB</td>
<td>Ping block</td>
</tr>
<tr>
<td>4</td>
<td>MR_PPPOE</td>
<td>PPPoE mode</td>
</tr>
<tr>
<td>3</td>
<td>MR_FARP</td>
<td>Force ARP mode</td>
</tr>
</tbody>
</table>

Definition at line 214 of file `w5500.h`.

```c
#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`.

```c
#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*.

```c
#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*.

```c
#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*.

```c
#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*.

```c
#define IR (_W5500_IO_BASE_ + (0x0015 << 8) + (WIZCHIP_CREG_BLOCK << 3))
```
Interrupt Register(R/W)

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

Each bit of IR defined as follows.

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFLICT</td>
<td>UNREACH</td>
<td>PPPoE</td>
<td>MP</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- 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 bit of IR. When a bit of IMR is and the corresponding bit of IR is an interrupt will be issued. In other words, if a bit of IMR is an interrupt will not be issued even if the corresponding bit of IR is

Each bit of IMR defined as the following.

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM_IR7</td>
<td>IM_IR6</td>
<td>IM_IR5</td>
<td>IM_IR4</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

- IM_IR7 : IP Conflict Interrupt Mask
- IM_IR6 : Destination unreachable Interrupt Mask
- IM_IR5 : PPPoE Close Interrupt Mask
- IM_IR4 : Magic Packet Interrupt Mask
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

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

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.

```c
#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.

```c
#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.

```c
#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...
#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))
<table>
<thead>
<tr>
<th>Define</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_TOS(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0015 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_TTL(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0016 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_RXBUF_SIZE(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x001E \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_TXBUF_SIZE(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x001F \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_TX_FSR(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0020 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_TX_RD(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0022 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_TX_WR(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0024 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_RX_RSR(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0026 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
<tr>
<td>Sn_RX_RD(N)</td>
<td>(<em>_W5500_IO_BASE</em> + (0x0028 \ll 8) + (WIZCHIP_SREG_BLOCK(N) \ll 3))</td>
</tr>
</tbody>
</table>
#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

```c
#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.

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTI/MFEN</td>
<td>BCASTB</td>
<td>ND/MC/MMB</td>
<td>UCASTB/MIP6B</td>
<td>Protocol[3]</td>
</tr>
</tbody>
</table>

- **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 multicasting**
- **Sn_MR_MMB** : Multicast Blocking **in Sn_MR_MACRAW mode**
- **Sn_MR_UCASTB** : Unicast Block **in UDP Multicasting**
- **Sn_MR_MIP6B** : IPv6 packet Blocking **in Sn_MR_MACRAW mod**
- **Protocol**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TCP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>UDP</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MACRAW</td>
</tr>
</tbody>
</table>

- **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**.


```c
#define (_W5500_IO_BASE_ + (0x0001 << 8) +
Sn_CR ( N ) (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.

```c
#define (_W5500_IO_BASE_ + (0x0002 << 8) +
Sn_IR ( N ) (WIZCHIP_SREG_BLOCK(N) << 3))
```

Socket interrupt register(R)

**Sn_IR** indicates the status of Socket Interrupt such as establishment, receiving data, timeout). When an interrupt occurs and the corresponding bit of Sn_IMR is the case, the bit of Sn_IR becomes
In order to clear the Sn_IR bit, the host should write the bit to

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
<td>SEND_OK</td>
<td>TIMEOUT</td>
<td>RECV</td>
<td>DIS</td>
<td></td>
</tr>
</tbody>
</table>

- **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.

```c
#define Sn_SR ( _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`.

```c
#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`.

```c
#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`.

```c
#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 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 523 of file *w5500.h*.

```c
#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 client mode, it configures the listen port number of TCP server before CONNECT command. In TCP Server mode, 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*.

```c
#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*.

```c
#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**.

```c
#define Sn_TOS (_W5500_IO_BASE_ + (0x0016 << 8) + Sn_TOS (N) (WIZCHIP_SREG_BLOCK(N) << 3))
```

**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**.

```c
#define Sn_TTL (_W5500_IO_BASE_ + (0x0016 << 8) + Sn_TTL (N) (WIZCHIP_SREG_BLOCK(N) << 3))
```

**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**.

```c
#define Sn_RXBUF_SIZE (_W5500_IO_BASE_ + (0x001E << 8) + Sn_RXBUF_SIZE (N) (WIZCHIP_SREG_BLOCK(N) << 3))
```

**Sn_TXBUF_SIZE** configures the Transmit memory size register(R/W)

```c
#define Sn_TXBUF_SIZE (_W5500_IO_BASE_ + (0x001F << 8) + Sn_TXBUF_SIZE (N) (WIZCHIP_SREG_BLOCK(N) << 3))
```
**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 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*.

```c
#define Sn_TX_FSR (N) (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*.

```c
#define Sn_TX_RD (N) (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**.

```c
#define Sn_TX_WR ( ( _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**.

```c
#define Sn_RX_RSR ( ( _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**.

```c
#define (_W5500_IO_BASE_ + (0x0028 << 8) + Sn_RX_RD ( N ) (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**.

```c
#define (_W5500_IO_BASE_ + (0x002A << 8) + Sn_RX_WR ( N ) (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.
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.

```c
define   ( _W5500_IO_BASE_ + (0x002D << 8) +
Sn_FRAG  (   N ) (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.

```c
define   ( _W5500_IO_BASE_ + (0x002F << 8) +
Sn_KPALVTR (   N ) (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

Here are the classes, structs, unions and interfaces with brief descriptions:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__WIZCHIP</td>
<td>The set of callback functions for W5500:WIZCHIP I/O functions W5200:WIZCHIP I/O functions</td>
</tr>
<tr>
<td>_CRIS</td>
<td></td>
</tr>
<tr>
<td>_CS</td>
<td></td>
</tr>
<tr>
<td>_IF</td>
<td></td>
</tr>
<tr>
<td>wiz_NetInfo_t</td>
<td></td>
</tr>
<tr>
<td>wiz_NetTimeout_t</td>
<td></td>
</tr>
<tr>
<td>wiz_PhyConf_t</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.stackoverflow.com) 1.8.9.1
## Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th><strong>Classes</strong></th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class List</td>
<td><strong>Class Index</strong></td>
<td>Class Members</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Class Index

<table>
<thead>
<tr>
<th>W</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WIZCHIP</strong>::__CRIS</td>
<td>wiz__NetTimeout_t</td>
</tr>
<tr>
<td><em>WIZCHIP</em>_::__CS</td>
<td>wiz__PhyConf_t</td>
</tr>
<tr>
<td><strong>WIZCHIP</strong>::__IF</td>
<td>wiz__NetInfo_t</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.stackoverflow.com) 1.8.9.1
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:

<table>
<thead>
<tr>
<th>Ethernet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W5100</td>
<td></td>
</tr>
<tr>
<td>w5100.c</td>
<td>W5100 HAL Interface</td>
</tr>
<tr>
<td>w5100.h</td>
<td>W5100 HAL Header File</td>
</tr>
<tr>
<td>W5200</td>
<td></td>
</tr>
<tr>
<td>w5200.c</td>
<td>W5200 HAL Interface</td>
</tr>
<tr>
<td>w5200.h</td>
<td>W5200 HAL Header File</td>
</tr>
<tr>
<td>W5300</td>
<td></td>
</tr>
<tr>
<td>w5300.c</td>
<td></td>
</tr>
<tr>
<td>w5300.h</td>
<td>W5300 HAL implement File</td>
</tr>
<tr>
<td>W5500</td>
<td></td>
</tr>
<tr>
<td>w5500.c</td>
<td>W5500 HAL Interface</td>
</tr>
<tr>
<td>w5500.h</td>
<td>W5500 HAL Header File</td>
</tr>
<tr>
<td>socket.c</td>
<td>SOCKET APIs Implements file</td>
</tr>
<tr>
<td>socket.h</td>
<td>SOCKET APIs Header file</td>
</tr>
<tr>
<td>wizchip_conf.c</td>
<td>WIZCHIP Config Header File</td>
</tr>
<tr>
<td>wizchip_conf.h</td>
<td>WIZCHIP Config Header File</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ethernet Directory Reference
<table>
<thead>
<tr>
<th>directory</th>
<th>W5100</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>W5200</td>
</tr>
<tr>
<td>directory</td>
<td>W5300</td>
</tr>
<tr>
<td>directory</td>
<td>W5500</td>
</tr>
</tbody>
</table>
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>socket.c</code></td>
<td>SOCKET APIs Implements file.</td>
</tr>
<tr>
<td><code>socket.h</code></td>
<td>SOCKET APIs Header file.</td>
</tr>
<tr>
<td><code>wizchip_conf.c</code></td>
<td>WIZCHIP Config Header File.</td>
</tr>
<tr>
<td><code>wizchip_conf.h</code></td>
<td>WIZCHIP Config Header File.</td>
</tr>
</tbody>
</table>
## Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>W5100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**W5100 Directory Reference**
### Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w5100.c</strong></td>
<td>W5100 HAL Interface.</td>
</tr>
<tr>
<td><strong>w5100.h</strong></td>
<td>W5100 HAL Header File.</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
### w5100.c File Reference

W5100 HAL Interface. More...

```c
#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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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

**w5100.h File Reference**

W5100 HAL Header File. More...

```c
#include <stdint.h> #include "wizchip_conf.h"
```

Go to the source code of this file.
### Macros

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define _WIZCHIP_SN_BASE_ (0x0400)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_SN_SIZE_ (0x0100)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_TXBUF_ (0x4000)</code></td>
<td>Internal Tx buffer address of the iinchip */</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_RXBUF_ (0x6000)</code></td>
<td>Internal Rx buffer address of the iinchip */</td>
</tr>
<tr>
<td><code>#define WIZCHIP_CREG_BLOCK 0x00</code></td>
<td>Common register block. More...</td>
</tr>
<tr>
<td><code>#define WIZCHIP_SREG_BLOCK(N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_ *N)</code></td>
<td>Socket N register block. More...</td>
</tr>
<tr>
<td><code>#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)</code></td>
<td>Increase offset address. More...</td>
</tr>
<tr>
<td><code>#define _W5100_IO_BASE_ _WIZCHIP_IO_BASE_</code></td>
<td></td>
</tr>
<tr>
<td><code>#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR,VAL)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_READ_BUF(ADDR, BUF, LEN) WIZCHIP_READ_BUF(ADDR,BUF,LEN)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE_BUF(ADDR, BUF, LEN) WIZCHIP_WRITE_BUF(ADDR,BUF,LEN)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td>#define</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>MR</code></td>
<td>Mode Register address (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>MR</code> is used for S/W reset, ping block mode, PPPoE mode and etc.</td>
</tr>
<tr>
<td><code>GAR</code></td>
<td>Gateway IP Register address (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>GAR</code> (W5100_IO_BASE + 0x0001)</td>
</tr>
<tr>
<td><code>SUBR</code></td>
<td>Subnet mask Register address (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>SUBR</code> (W5100_IO_BASE + 0x0005)</td>
</tr>
<tr>
<td><code>SHAR</code></td>
<td>Source MAC Register address (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>SHAR</code> (W5100_IO_BASE + 0x0009)</td>
</tr>
<tr>
<td><code>SIPR</code></td>
<td>Source IP Register address (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>SIPR</code> (W5100_IO_BASE + 0x000F)</td>
</tr>
<tr>
<td><code>IR</code></td>
<td>Interrupt Register (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>IR</code> (W5100_IO_BASE + 0x0015)</td>
</tr>
<tr>
<td><code>_IMR_</code></td>
<td>Socket Interrupt Mask Register (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>_IMR_</code> (W5100_IO_BASE + 0x0016)</td>
</tr>
<tr>
<td><code>_RTR_</code></td>
<td>Timeout register address (1 is 100us) (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>_RTR_</code> (W5100_IO_BASE + 0x0017)</td>
</tr>
<tr>
<td><code>_RCR_</code></td>
<td>Retry count register (R/W)</td>
</tr>
<tr>
<td></td>
<td><code>_RCR_</code> (W5100_IO_BASE + 0x0019)</td>
</tr>
<tr>
<td><code>RMSR</code></td>
<td>(W5100_IO_BASE + 0x001A)</td>
</tr>
<tr>
<td><code>TMSR</code></td>
<td>(W5100_IO_BASE + 0x001B)</td>
</tr>
<tr>
<td><code>PATR</code></td>
<td>(W5100_IO_BASE + 0x001C)</td>
</tr>
</tbody>
</table>
PPP LCP Request Timer register in PPPoE mode(R)

#define PTIMER (_W5100_IO_BASE_ + (0x0028))
PPP LCP Request Timer register in PPPoE mode(R)

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

#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)

#define Sn_CR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001))
Socket command register(R/W)

#define Sn_IR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0002))
Socket interrupt register(R)

#define Sn_SR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0003))
Socket status register(R)

#define Sn_PORT(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004))
source port register(R/W)

#define Sn_DHAR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006))
Peer MAC register address(R/W)

#define 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) (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. More...

#define IR_PPPoE 0x20
Get the PPPoE close message. More...

#define IR_SOCK(sn) (0x01 << sn)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR_CLOSE</td>
<td>0x00</td>
<td>Unused socket.</td>
</tr>
<tr>
<td>Sn_MR_TCP</td>
<td>0x01</td>
<td>TCP</td>
</tr>
<tr>
<td>Sn_MR_UDP</td>
<td>0x02</td>
<td>UDP</td>
</tr>
<tr>
<td>Sn_MR_IPRAW</td>
<td>0x03</td>
<td>IP LAYER RAW SOCK.</td>
</tr>
<tr>
<td>Sn_MR_MACRAW</td>
<td>0x04</td>
<td>MAC LAYER RAW SOCK.</td>
</tr>
<tr>
<td>Sn_MR_PPPoE</td>
<td>0x05</td>
<td>PPPoE</td>
</tr>
<tr>
<td>Sn_MR_ND</td>
<td>0x20</td>
<td>No Delayed Ack(TCP), Multicast flag.</td>
</tr>
<tr>
<td>Sn_MR_MC</td>
<td></td>
<td>Support UDP Multicasting.</td>
</tr>
<tr>
<td>Sn_MR_MF</td>
<td>0x40</td>
<td>MAC filter enable in Sn_MR_MACRAW mode.</td>
</tr>
<tr>
<td>Sn_MR_MULTI</td>
<td>0x80</td>
<td>Support UDP Multicasting.</td>
</tr>
<tr>
<td>Sn_CR_OPEN</td>
<td>0x01</td>
<td>Initialize or open socket.</td>
</tr>
<tr>
<td>Define</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Sn_CR_LISTEN</td>
<td>0x02</td>
<td>Wait connection request in TCP mode (Server mode)</td>
</tr>
<tr>
<td>Sn_CR_CONNECT</td>
<td>0x04</td>
<td>Send connection request in TCP mode (Client mode)</td>
</tr>
<tr>
<td>Sn_CR_DISCON</td>
<td>0x08</td>
<td>Send closing request in TCP mode.</td>
</tr>
<tr>
<td>Sn_CR_CLOSE</td>
<td>0x10</td>
<td>Close socket.</td>
</tr>
<tr>
<td>Sn_CR_SEND</td>
<td>0x20</td>
<td>Update TX buffer pointer and send data.</td>
</tr>
<tr>
<td>Sn_CR_SEND_MAC</td>
<td>0x21</td>
<td>Send data with MAC address, so without ARP process.</td>
</tr>
<tr>
<td>Sn_CR_SEND_KEEP</td>
<td>0x22</td>
<td>Send keep alive message.</td>
</tr>
<tr>
<td>Sn_CR_RECV</td>
<td>0x40</td>
<td>Update RX buffer pointer and receive data.</td>
</tr>
<tr>
<td>Sn_CR_PCON</td>
<td>0x23</td>
<td>PPPoE connection.</td>
</tr>
<tr>
<td>Sn_CR_PDISCON</td>
<td>0x24</td>
<td>Closes PPPoE connection.</td>
</tr>
<tr>
<td>Sn_CR_PCR</td>
<td>0x25</td>
<td>REQ message transmission.</td>
</tr>
<tr>
<td>Sn_CR_PCN</td>
<td>0x26</td>
<td>NAK massage transmission.</td>
</tr>
<tr>
<td>#define</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>Sn_CR_PCJ</code></td>
<td>0x27</td>
<td>REJECT message transmission.</td>
</tr>
<tr>
<td><code>Sn_IR_PRECV</code></td>
<td>0x80</td>
<td>PPP Receive Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_PFAIL</code></td>
<td>0x40</td>
<td>PPP Fail Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_PNEXT</code></td>
<td>0x20</td>
<td>PPP Next Phase Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_SENDOK</code></td>
<td>0x10</td>
<td>SEND_OK Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_TIMEOUT</code></td>
<td>0x08</td>
<td>TIMEOUT Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_RECV</code></td>
<td>0x04</td>
<td>RECV Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_DISCON</code></td>
<td>0x02</td>
<td>DISCON Interrupt.</td>
</tr>
<tr>
<td><code>Sn_IR_CON</code></td>
<td>0x01</td>
<td>CON Interrupt.</td>
</tr>
<tr>
<td><code>SOCK_CLOSED</code></td>
<td>0x00</td>
<td>Closed.</td>
</tr>
<tr>
<td><code>SOCK_INIT</code></td>
<td>0x13</td>
<td>Initiate state.</td>
</tr>
<tr>
<td><code>SOCK_LISTEN</code></td>
<td>0x14</td>
<td>Listen state.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td><code>SOCK_SYNSENT</code></td>
<td>0x15</td>
<td>Connection state. More...</td>
</tr>
<tr>
<td><code>SOCK_SYNRECV</code></td>
<td>0x16</td>
<td>Connection state. More...</td>
</tr>
<tr>
<td><code>SOCK_ESTABLISHED</code></td>
<td>0x17</td>
<td>Success to connect. More...</td>
</tr>
<tr>
<td><code>SOCK_FIN_WAIT</code></td>
<td>0x18</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_CLOSING</code></td>
<td>0x1A</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_TIME_WAIT</code></td>
<td>0x1B</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_CLOSE_WAIT</code></td>
<td>0x1C</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_LAST_ACK</code></td>
<td>0x1D</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_UDP</code></td>
<td>0x22</td>
<td>UDP socket. More...</td>
</tr>
<tr>
<td><code>SOCK_IPRAW</code></td>
<td>0x32</td>
<td>IP raw mode socket. More...</td>
</tr>
<tr>
<td><code>SOCK_MACRAW</code></td>
<td>0x42</td>
<td>MAC raw mode socket. More...</td>
</tr>
<tr>
<td><code>SOCK_PPPOE</code></td>
<td>0x5F</td>
<td>PPPoE mode socket. More...</td>
</tr>
<tr>
<td>Define</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>IPPROTO_IP</td>
<td>0</td>
<td>Dummy for IP.</td>
</tr>
<tr>
<td>IPPROTO_ICMP</td>
<td>1</td>
<td>Control message protocol.</td>
</tr>
<tr>
<td>IPPROTO_IGMP</td>
<td>2</td>
<td>Internet group management protocol.</td>
</tr>
<tr>
<td>IPPROTO_GGP</td>
<td>3</td>
<td>GW^2 (deprecated)</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>6</td>
<td>TCP.</td>
</tr>
<tr>
<td>IPPROTO_PUP</td>
<td>12</td>
<td>PUP.</td>
</tr>
<tr>
<td>IPPROTO_UDP</td>
<td>17</td>
<td>UDP.</td>
</tr>
<tr>
<td>IPPROTO_IDP</td>
<td>22</td>
<td>XNS idp.</td>
</tr>
<tr>
<td>IPPROTO_ND</td>
<td>77</td>
<td>UNOFFICIAL net disk protocol.</td>
</tr>
<tr>
<td>IPPROTO_RAW</td>
<td>255</td>
<td>Raw IP packet.</td>
</tr>
<tr>
<td>WIZCHIP_CRITICAL_ENTER()</td>
<td>WIZCHIP.CRIS._enter()</td>
<td>Enter a critical section.</td>
</tr>
<tr>
<td>WIZCHIP_CRITICAL_EXIT()</td>
<td>WIZCHIP.CRIS._exit()</td>
<td>Exit a critical section.</td>
</tr>
</tbody>
</table>
#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...
```c
#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...
```


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

```c
#define getPTIMER()  WIZCHIP_READ(PTIMER)
Get PTIMER register. More...
```

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

```c
#define getPMAGIC()  WIZCHIP_READ(PMAGIC)
Get PMAGIC register. More...
```

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

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

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

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

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

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

```c
#define setSn_SR(sn)  WIZCHIP_READ(Sn_SR(sn))
Get Sn_SR register. More...
```

```c
#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),1)))
Get Sn_PORT register. More...

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

#define getSn_DHAR(sn, dhar)   WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar, 6)
Get Sn_DHAR register. More...

#define setSn_DIPR(sn, dipr)   WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)
Set Sn_DIPR register. More...

#define getSn_DIPR(sn, dipr)   WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 4)
Get Sn_DIPR register. More...

#define setSn_DPORT(sn, dport)   WIZCHIP_WRITE(Sn_DPORT(sn), dport)
Set Sn_DPORT register. More...

#define getSn_DPORT(sn)   (((uint16_t)WIZCHIP_READ(Sn_DPORT(sn))<<8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1)))
Get Sn_DPORT register. More...

#define setSn_MSSR(sn, mss)   WIZCHIP_WRITE(Sn_MSSR(sn), mss)
Set Sn_MSSR register. More...

#define getSn_MSSR(sn)   (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn))<<8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)))
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) & 0x03 << (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(TMSR (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)
```c
#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(sn)) << 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(sn)) << 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(sn)) << 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(sn)) << 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),1)))
Get `Sn_FRAG` register. More...

#define getSn_RXMAX(sn) ((uint16_t)(1 << getSn_RXMEM_SIZE(sn)) << 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td><strong>WIZCHIP_READ</strong> (uint32_t AddrSel)</td>
<td>It reads 1 byte value from a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE</strong> (uint32_t AddrSel, uint8_t wb)</td>
<td>It writes 1 byte value to a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_READ_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It reads sequence data from registers.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It writes sequence data to registers.</td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td><strong>getSn_TX_FSR</strong> (uint8_t sn)</td>
<td>Get Sn_TX_FSR register.</td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td><strong>getSn_RX_RSR</strong> (uint8_t sn)</td>
<td>Get Sn_RX_RSR register.</td>
<td></td>
</tr>
<tr>
<td>uint32_t</td>
<td><strong>getSn_RxBASE</strong> (uint8_t sn)</td>
<td>Get the base address of socket sn RX buffer.</td>
<td></td>
</tr>
<tr>
<td>uint32_t</td>
<td><strong>getSn_TxBASE</strong> (uint8_t sn)</td>
<td>Get the base address of socket sn TX buffer.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_send_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to internal TX memory.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_recv_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to your buffer from internal RX memory.</td>
<td></td>
</tr>
</tbody>
</table>
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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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.h.
**Macro Definition Documentation**

```c
#define _WIZCHIP_SN_BASE_ (0x0400)
```

Definition at line 50 of file `w5100.h`.

```c
#define _WIZCHIP_SN_SIZE_ (0x0100)
```

Definition at line 51 of file `w5100.h`.

```c
#define _WIZCHIP_IO_TXBUF_ (0x4000) /* Internal Tx buffer address of the iiochip */
```

Definition at line 52 of file `w5100.h`.

```c
#define _WIZCHIP_IO_RXBUF_ (0x6000) /* Internal Rx buffer address of the iiochip */
```

Definition at line 53 of file `w5100.h`.

```c
#define WIZCHIP_CREG_BLOCK 0x00
```

Common register block.

Definition at line 56 of file `w5100.h`.

```c
#define WIZCHIP_SREG_BLOCK (_WIZCHIP_SN_BASE_ +
                         WIZCHIP_SREG_BLOCK (N) _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,
The defined for legacy chip driver.
Definition at line 78 of file w5100.h.

#define IINCHIP_WRITE_BUF ( 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))
#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 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.

Definition at line 586 of file w5100.h.

#define MR_PPPOE 0x08

Enable PPPoE.

0 : Disable PPPoE mode
1 : Enable PPPoE mode
If you use ADSL, this bit should be '1'.

Definition at line 594 of file w5100.h.
Address Auto-Increment in Indirect Bus Interface.

0 : Disable auto-increment  
1 : Enable auto-increment  
At the Indirect Bus Interface mode, if this bit is set as 0x1, 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 0x1, 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 conflict

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()`.

```c
#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()`.

```c
#define Sn_MR_IPRAW 0x03
```

IP LAYER RAW SOCK.

Definition at line 655 of file *w5100.h*.

Referenced by `recvfrom()`, and `socket()`.

```c
#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 1, it sends the ACK packet without delay as soon as a  
Data packet is received from a peer.  
When this bit is 0, 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 \texttt{w5100.h}.

\#define \texttt{Sn\_MR\_MF} \ 0x40

MAC filter enable in \texttt{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 \texttt{w5100.h}.

\#define \texttt{Sn\_MR\_MFEN} \ \texttt{Sn\_MR\_MF}

Definition at line 701 of file \texttt{w5100.h}.

\#define \texttt{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, \texttt{Sn\_DIPR} \ \texttt{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 \texttt{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.

<table>
<thead>
<tr>
<th>Sn_MR (P[3:0])</th>
<th>Sn_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR_CLOSE</td>
<td>–</td>
</tr>
<tr>
<td>Sn_MR_TCP</td>
<td>SOCK_INIT (0x13)</td>
</tr>
<tr>
<td>Sn_MR_UDP</td>
<td>SOCK_UDP (0x22)</td>
</tr>
<tr>
<td>S0_MR_IPRAW</td>
<td>SOCK_IPRAW (0x32)</td>
</tr>
<tr>
<td>S0_MR_MACRAW</td>
<td>SOCK_MACRAW (0x42)</td>
</tr>
<tr>
<td>S0_MR_PPPoE</td>
<td>SOCK_PPPoE (0x5F)</td>
</tr>
</tbody>
</table>

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 SOCKET_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 \texttt{w5100.h}.

Referenced by \texttt{listen()}. 

\begin{verbatim}
#define Sn_CR_CONNECT 0x04
\end{verbatim}

Send connection request in TCP mode (Client mode)

To connect, a connect-request (SYN packet) is sent to TCP server configured by \texttt{Sn_DIPR} & \texttt{Sn_DPORT}(destination address & port). If the connect-request is successful, the \texttt{Sn_SR} is changed to \texttt{SOCK_ESTABLISHED} and the \texttt{Sn_IR(0)} becomes

The connect-request fails in the following three cases.

1. When a ARPTO occurs (\texttt{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 (\texttt{Sn_IR(3)} = '1')
3. When a RST packet is received instead of a SYN/ACK packet. In these cases, \texttt{Sn_SR} is changed to \texttt{SOCK_CLOSED}.

\textbf{Note}\n
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 \texttt{w5100.h}.

Referenced by \texttt{connect()}. 

\begin{verbatim}
#define Sn_CR_DISCON 0x08
\end{verbatim}

Send closing request in TCP mode.

Regardless of TCP server or TCP client the DISCON command processes the disconnect-process \texttt{(Active close or Passive close}.

\textbf{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), \texttt{Sn\_SR} is changed to \texttt{SOCK\_CLOSED}. Otherwise, TCPTO occurs (\texttt{Sn\_IR(3)}=‘1’) and then \texttt{Sn\_SR} is changed to \texttt{SOCK\_CLOSED}.

**Note**
Valid only in TCP mode. send closing request in tcp mode
Definition at line 766 of file \texttt{w5100.h}.
Referenced by \texttt{disconnect()}. 

```c
#define Sn_CR_CLOSE 0x10
```
Close socket.
\texttt{Sn\_SR} is changed to \texttt{SOCK\_CLOSED}.
Definition at line 772 of file \texttt{w5100.h}.
Referenced by \texttt{close()}. 

```c
#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 (\texttt{Sn\_TX\_FSR}), Socket n, TX Write Pointer Register (\texttt{Sn\_TX\_WR}), and Socket n TX Read Pointer Register (\texttt{Sn\_TX\_RD}).
**#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.

**#define Sn_CR_SEND_KEEP 0x22**

Send keep alive message.

It checks the connection status by sending 1 byte 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.
#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.
\#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.

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.

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 \texttt{w5100.h}.

\begin{verbatim}
#define Sn_IR_DISCON 0x02
\end{verbatim}

DISCON Interrupt.
This is issued when FIN or FIN/ACK packet is received from a peer.

Definition at line 879 of file \texttt{w5100.h}.

\begin{verbatim}
#define Sn_IR_CON 0x01
\end{verbatim}

CON Interrupt.
This is issued one time when the connection with peer is successful and then \texttt{Sn_SR} is changed to \texttt{SOCK_ESTABLISHED}.

Definition at line 885 of file \texttt{w5100.h}.

\begin{verbatim}
#define SOCK_CLOSED 0x00
\end{verbatim}

Closed.
This indicates that Socket n is released.
When DICON, CLOSE command is ordered, or when a timeout occurs, it is changed to \texttt{SOCK_CLOSED} regardless of previous status.closed

Definition at line 893 of file \texttt{w5100.h}.

Referenced by \texttt{close()}, \texttt{connect()}, \texttt{disconnect()}, \texttt{listen()}, \texttt{recvfrom()}, \texttt{sendto()}, and \texttt{socket()}.

\begin{verbatim}
#define SOCK_INIT 0x13
\end{verbatim}
Initiate state.

This indicates Socket n is opened with TCP mode. It is changed to \texttt{SOCK\_INIT} when \texttt{Sn\_MR(P[3:0]) = 001} and OPEN command is ordered. After \texttt{SOCK\_INIT}, user can use LISTEN /CONNECT command.

Definition at line 901 of file \texttt{w5100.h}.

\begin{verbatim}
#define SOCK_LISTEN 0x14
\end{verbatim}

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 \texttt{SOCK\_ESTABLISHED} when the connection-request is successfully accepted. Otherwise it will change to \texttt{SOCK\_CLOSED} after TCPTO occurred (\texttt{Sn\_IR\_TIMEOUT} = '1').

Definition at line 909 of file \texttt{w5100.h}.

Referenced by \texttt{listen()}.

\begin{verbatim}
#define SOCK_SYNSENT 0x15
\end{verbatim}

Connection state.

This indicates Socket n sent the connect-request packet (SYN packet) to a peer. It is temporarily shown when \texttt{Sn\_SR} is changed from \texttt{SOCK\_INIT} to \texttt{SOCK\_ESTABLISHED} by CONNECT command. If connect-accept(SYN/ACK packet) is received from the peer at \texttt{SOCK\_SYNSENT}, it changes to \texttt{SOCK\_ESTABLISHED}. Otherwise, it changes to \texttt{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 \textbf{SOCK\_CLOSED}.

Definition at line 943 of file \texttt{w5100.h}.

\begin{verbatim}
define SOCK\_CLOSING 0x1A
\end{verbatim}

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 \textbf{SOCK\_CLOSED}.

Definition at line 951 of file \texttt{w5100.h}.

\begin{verbatim}
define SOCK\_TIME\_WAIT 0x1B
\end{verbatim}

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 \textbf{SOCK\_CLOSED}.

Definition at line 959 of file \texttt{w5100.h}.

\begin{verbatim}
define SOCK\_CLOSE\_WAIT 0x1C
\end{verbatim}

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 transferred 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.

The socket is opened in IPRAW mode. The SOCKET status is changed 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.

```c
#define IPPROTO_IP  0
```

Dummy for IP.

Definition at line 1010 of file w5100.h.

```c
#define IPPROTO_ICMP  1
```

Control message protocol.

Definition at line 1011 of file w5100.h.

```c
#define IPPROTO_IGMP  2
```

Internet group management protocol.

Definition at line 1012 of file w5100.h.

```c
#define IPPROTO_GGP  3
```

GW^2 (deprecated)

Definition at line 1013 of file w5100.h.

```c
#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


<table>
<thead>
<tr>
<th>Function</th>
<th>Definition Line</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>setSn_RXMEM_SIZE(sn,rxmemsize)</td>
<td>1604</td>
<td>wizchip_init()</td>
</tr>
<tr>
<td>getSn_RXBUF_SIZE(sn)</td>
<td>1615</td>
<td>wizchip_init()</td>
</tr>
<tr>
<td>setSn_TXBUF_SIZE(sn,txmemsiz)</td>
<td>1626</td>
<td>wizchip_init()</td>
</tr>
<tr>
<td>getSn_TXBUF_SIZE(sn)</td>
<td>1637</td>
<td>wizchip_init()</td>
</tr>
</tbody>
</table>
## Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>W5200</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### W5200 Directory Reference
## Files

<table>
<thead>
<tr>
<th>file</th>
<th>w5200.c [code]</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5200 HAL Interface.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>file</th>
<th>w5200.h [code]</th>
</tr>
</thead>
<tbody>
<tr>
<td>W5200 HAL Header File.</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](https://doxygen.org) 1.8.9.1
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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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

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

<table>
<thead>
<tr>
<th>Macro %</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define _WIZCHIP_SN_BASE_ (0x4000)</code></td>
<td>Macros</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_SN_SIZE_ (0x0100)</code></td>
<td>Macros</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_TXBUF_ (0x8000) /* Internal Tx buffer address of the iinchip */</code></td>
<td>Macros</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_RXBUF_ (0xC000) /* Internal Rx buffer address of the iinchip */</code></td>
<td>Macros</td>
</tr>
<tr>
<td><code>#define _W5200_SPI_READ_ (0x00 &lt;&lt; 7)</code></td>
<td>SPI interface Read operation in Control Phase. More...</td>
</tr>
<tr>
<td><code>#define _W5200_SPI_WRITE_ (0x01 &lt;&lt; 7)</code></td>
<td>SPI interface Write operation in Control Phase. More...</td>
</tr>
<tr>
<td><code>#define WIZCHIP_CREG_BLOCK 0x00</code></td>
<td>Common register block. More...</td>
</tr>
<tr>
<td><code>#define WIZCHIP_SREG_BLOCK(N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_ *N)</code></td>
<td>Socket N register block. More...</td>
</tr>
<tr>
<td><code>#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)</code></td>
<td>Increase offset address. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR,VAL)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_READ_BUF(ADDR, BUF, LEN) WIZCHIP_READ_BUF(ADDR,BUF,LEN)</code></td>
<td></td>
</tr>
</tbody>
</table>
The defined for legacy chip driver. **More...**

<table>
<thead>
<tr>
<th>#define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IINCHIP_WRITE_BUF(ADDR, BUF, LEN)</code></td>
<td><code>WIZCHIP_WRITE(ADDR, BUF, LEN)</code></td>
</tr>
<tr>
<td><code>MR (_W5200_IO_BASE_ + (0x0000))</code></td>
<td>Mode Register address(R/W)</td>
</tr>
<tr>
<td><code>GAR (_W5200_IO_BASE_ + (0x0001))</code></td>
<td>Gateway IP Register address(R/W)</td>
</tr>
<tr>
<td><code>SUBR (_W5200_IO_BASE_ + (0x0005))</code></td>
<td>Subnet mask Register address(R/W)</td>
</tr>
<tr>
<td><code>SHAR (_W5200_IO_BASE_ + (0x0009))</code></td>
<td>Source MAC Register address(R/W)</td>
</tr>
<tr>
<td><code>SIPR (_W5200_IO_BASE_ + (0x000F))</code></td>
<td>Source IP Register address(R/W)</td>
</tr>
<tr>
<td><code>IR (_W5200_IO_BASE_ + (0x0015))</code></td>
<td>Interrupt Register(R/W)</td>
</tr>
<tr>
<td><code>_IMR_ (_W5200_IO_BASE_ + (0x0016))</code></td>
<td>Socket Interrupt Mask Register(R/W)</td>
</tr>
<tr>
<td><code>_RTR_ (_W5200_IO_BASE_ + (0x0017))</code></td>
<td>Timeout register address( 1 is 100us )(R/W)</td>
</tr>
<tr>
<td><code>_RCR_ (_W5200_IO_BASE_ + (0x0019))</code></td>
<td>Retry count register(R/W)</td>
</tr>
<tr>
<td><code>PATR (_W5200_IO_BASE_ + (0x001C))</code></td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PPPALGO</td>
<td>PPP LCP Request Timer register in PPPoE mode</td>
</tr>
<tr>
<td>VERSIONR</td>
<td>Chip version register address</td>
</tr>
<tr>
<td>PTIMER</td>
<td>PPP LCP Request Timer register in PPPoE mode</td>
</tr>
<tr>
<td>PMAGIC</td>
<td>PPP LCP Magic number register in PPPoE mode</td>
</tr>
<tr>
<td>INTLEVEL</td>
<td>Set Interrupt low level timer register address</td>
</tr>
<tr>
<td>IR2</td>
<td>Socket Interrupt Register</td>
</tr>
<tr>
<td>PHYSTATUS</td>
<td>PHYSTATUS</td>
</tr>
<tr>
<td>IMR2</td>
<td>Interrupt mask register</td>
</tr>
<tr>
<td>Sn_MR(sn)</td>
<td>Socket Mode register</td>
</tr>
<tr>
<td>Sn_CR(sn)</td>
<td>Socket command register</td>
</tr>
<tr>
<td>Sn_IR(sn)</td>
<td>Socket control register</td>
</tr>
</tbody>
</table>
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 + (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...
<table>
<thead>
<tr>
<th>#define</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Sn_MR_CLOSE</code></td>
<td>0x00</td>
<td>Unused socket. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_TCP</code></td>
<td>0x01</td>
<td>TCP. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_UDP</code></td>
<td>0x02</td>
<td>UDP. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_IPRAW</code></td>
<td>0x03</td>
<td>IP LAYER RAW SOCK. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_MACRAW</code></td>
<td>0x04</td>
<td>MAC LAYER RAW SOCK. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_PPPOE</code></td>
<td>0x05</td>
<td>PPPoE. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_ND</code></td>
<td>0x20</td>
<td>No Delayed Ack(TCP), Multicast flag. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_MF</code></td>
<td>0x40</td>
<td>Multicast Blocking in <code>Sn_MR_MACRAW</code> mode. [More...]</td>
</tr>
<tr>
<td><code>Sn_MR_MFEN</code></td>
<td>Sn_MR_MF</td>
<td></td>
</tr>
<tr>
<td><code>Sn_MR_MULTI</code></td>
<td>0x80</td>
<td>Support UDP Multicasting. [More...]</td>
</tr>
<tr>
<td><code>Sn_CR_OPEN</code></td>
<td>0x01</td>
<td>Initialize or open socket. [More...]</td>
</tr>
<tr>
<td><code>Sn_CR_LISTEN</code></td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>Define</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>Sn_CR_CONNECT</td>
<td>0x04</td>
<td>Send connection request in TCP mode (Client mode)</td>
</tr>
<tr>
<td>Sn_CR_DISCON</td>
<td>0x08</td>
<td>Send closing request in TCP mode.</td>
</tr>
<tr>
<td>Sn_CR_CLOSE</td>
<td>0x10</td>
<td>Close socket.</td>
</tr>
<tr>
<td>Sn_CR_SEND</td>
<td>0x20</td>
<td>Update TX buffer pointer and send data.</td>
</tr>
<tr>
<td>Sn_CR_SEND_MAC</td>
<td>0x21</td>
<td>Send data with MAC address, so without ARP process.</td>
</tr>
<tr>
<td>Sn_CR_SEND_KEEP</td>
<td>0x22</td>
<td>Send keep alive message.</td>
</tr>
<tr>
<td>Sn_CR_RECV</td>
<td>0x40</td>
<td>Update RX buffer pointer and receive data.</td>
</tr>
<tr>
<td>Sn_CR_PCON</td>
<td>0x23</td>
<td>PPPoE connection.</td>
</tr>
<tr>
<td>Sn_CR_PDISCON</td>
<td>0x24</td>
<td>Closes PPPoE connection.</td>
</tr>
<tr>
<td>Sn_CR_PCR</td>
<td>0x25</td>
<td>REQ message transmission.</td>
</tr>
<tr>
<td>Sn_CR_PCN</td>
<td>0x26</td>
<td>NAK massage transmission.</td>
</tr>
<tr>
<td>Sn_CR_PCJ</td>
<td>0x27</td>
<td></td>
</tr>
<tr>
<td>Define Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Sn_IR_PRECV</td>
<td>0x80</td>
<td>PPP Receive Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_PFAIL</td>
<td>0x40</td>
<td>PPP Fail Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_PNEXT</td>
<td>0x20</td>
<td>PPP Next Phase Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_SENDOK</td>
<td>0x10</td>
<td>SEND_OK Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_TIMEOUT</td>
<td>0x08</td>
<td>TIMEOUT Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_RECV</td>
<td>0x04</td>
<td>RECV Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_DISCON</td>
<td>0x02</td>
<td>DISCON Interrupt.</td>
</tr>
<tr>
<td>Sn_IR_CON</td>
<td>0x01</td>
<td>CON Interrupt.</td>
</tr>
<tr>
<td>SOCK_CLOSED</td>
<td>0x00</td>
<td>Closed.</td>
</tr>
<tr>
<td>SOCK_INIT</td>
<td>0x13</td>
<td>Initiate state.</td>
</tr>
<tr>
<td>SOCK_LISTEN</td>
<td>0x14</td>
<td>Listen state.</td>
</tr>
<tr>
<td>SOCK_SYNSENT</td>
<td>0x15</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td><code>SOCK_SYNRECV</code></td>
<td>0x16</td>
<td>Connection state.</td>
</tr>
<tr>
<td><code>SOCK_ESTABLISHED</code></td>
<td>0x17</td>
<td>Success to connect.</td>
</tr>
<tr>
<td><code>SOCK_FIN_WAIT</code></td>
<td>0x18</td>
<td>Closing state.</td>
</tr>
<tr>
<td><code>SOCK_CLOSING</code></td>
<td>0x1A</td>
<td>Closing state.</td>
</tr>
<tr>
<td><code>SOCK_TIME_WAIT</code></td>
<td>0x1B</td>
<td>Closing state.</td>
</tr>
<tr>
<td><code>SOCK_CLOSE_WAIT</code></td>
<td>0x1C</td>
<td>Closing state.</td>
</tr>
<tr>
<td><code>SOCK_LAST_ACK</code></td>
<td>0x1D</td>
<td>Closing state.</td>
</tr>
<tr>
<td><code>SOCK_UDP</code></td>
<td>0x22</td>
<td>UDP socket.</td>
</tr>
<tr>
<td><code>SOCK_IPRAW</code></td>
<td>0x32</td>
<td>IP raw mode socket.</td>
</tr>
<tr>
<td><code>SOCK_MACRAW</code></td>
<td>0x42</td>
<td>MAC raw mode socket.</td>
</tr>
<tr>
<td><code>SOCK_PPPOE</code></td>
<td>0x5F</td>
<td>PPPoE mode socket.</td>
</tr>
<tr>
<td><code>IPPROTO_IP</code></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
#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...

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

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

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

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

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

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

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

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

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

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

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

```c
#define setIMR(imr) WIZCHIP_WRITE(IMR2, imr & 0xA0)
```
Set IMR2 register. More...

```c
#define getIMR() (WIZCHIP_READ(IMR2) & 0xA0)
```

Get IMR2 register. More...

```c
#define setRTR(rtr)
```

Set RTR register. More...

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

Get RTR register. More...

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

Set RCR register. More...

```c
#define getRCR() WIZCHIP_READ(_RCR_)
```

Get RCR register. More...

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

Get PATR register. More...

```c
#define getPPPALGO()
```

Get PPPALGO register. More...

```c
#define getVERSIONR()
```

Get VERSIONR register. More...

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

Set PTIMER register. More...

```c
#define getPTIMER()
```

Get PTIMER register. More...

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

Set PMAGIC register. More...
```c
#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),1)))
Get Sn_PORT register. More...

#define setSn_DHAR(sn, dhar) WIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6)
Set Sn_DHAR register. More...
#define getSn_DHAR(sn, dhar)  WIZCHIP_READ_BUF(Sn_DHAR(sn, dhar, 6)
Get Sn_DHAR register. More...

#define setSn_DIPR(sn, dipr)  WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)
Set Sn_DIPR register. More...

#define getSn_DIPR(sn, dipr)  WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 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(sn) << 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(sn) << 8) +
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn), 1))
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), rxmemsize)
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_SIZE(sn))
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), txmemsize)
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_SIZE(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(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn), 1)))
Get Sn_TX_RD register. More...

#define setSn_TX_WR(sn, txwr)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getSn_TX_WR(sn)</code></td>
<td>Get Sn_TX_WR register. More...</td>
<td><code>getSn_TX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_WR &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1)))</code></td>
</tr>
<tr>
<td><code>setSn_RX_RD(sn, rxrd)</code></td>
<td>Set Sn_RX_RD register. More...</td>
<td><code>setSn_RX_RD(sn, rxrd)</code></td>
</tr>
<tr>
<td><code>getSn_RX_RD(sn)</code></td>
<td>Get Sn_RX_RD register. More...</td>
<td><code>getSn_RX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_RD(&lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1)))</code></td>
</tr>
<tr>
<td><code>setSn_RX_WR(sn, rxwr)</code></td>
<td>Set Sn_RX_WR register. More...</td>
<td><code>setSn_RX_WR(sn, rxwr)</code></td>
</tr>
<tr>
<td><code>getSn_RX_WR(sn)</code></td>
<td>Get Sn_RX_WR register. More...</td>
<td><code>getSn_RX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(&lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1)))</code></td>
</tr>
<tr>
<td><code>setSn_IMR(sn, imr)</code></td>
<td>Set Sn_IMR register. More...</td>
<td><code>setSn_IMR(sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), imr)</code></td>
</tr>
<tr>
<td><code>getSn_IMR(sn)</code></td>
<td>Get Sn_IMR register. More...</td>
<td><code>getSn_IMR(sn) WIZCHIP_READ(Sn_IMR(sn))</code></td>
</tr>
<tr>
<td><code>setSn_FRAG(sn, frag)</code></td>
<td>Set Sn_FRAG register. More...</td>
<td><code>setSn_FRAG(sn, frag)</code></td>
</tr>
<tr>
<td><code>getSn_FRAG(sn)</code></td>
<td>Get Sn_FRAG register. More...</td>
<td><code>getSn_FRAG(sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn))))</code></td>
</tr>
<tr>
<td><code>getSn_RxMAX(sn)</code></td>
<td>Get Sn_RxMAX register. More...</td>
<td><code>getSn_RxMAX(sn) (((uint16_t)getSn_RXMEM_SIZE(sn) &lt;&lt; 1)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Equations</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td><code>getSn_RxMAX(sn)</code></td>
<td>Get the max RX buffer size of socket sn. More...</td>
<td><code>((uint16_t)getSn_TXMEM_SIZE(sn) &lt;&lt; 1)</code></td>
</tr>
<tr>
<td><code>getSn_RxMASK(sn)</code></td>
<td>Get the mask of socket sn RX buffer. More...</td>
<td><code>((uint16_t)getSn_RxMAX(sn) - 1)</code></td>
</tr>
<tr>
<td><code>getSn_TxMAX(sn)</code></td>
<td>Get the max TX buffer size of socket sn. More...</td>
<td><code>((uint16_t)getSn_TXMEM_SIZE(sn) &lt;&lt; 1)</code></td>
</tr>
<tr>
<td><code>getSn_TxMASK(sn)</code></td>
<td>Get the mask of socket sn TX buffer. More...</td>
<td><code>((uint16_t)getSn_TxMAX(sn) - 1)</code></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8_t WIZCHIP_READ (uint32_t AddrSel)</code></td>
<td>It reads 1 byte value from a register. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb)</code></td>
<td>It writes 1 byte value to a register. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It reads sequence data from registers. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void WIZCHIP_WRITE_BUF (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</code></td>
<td>It writes sequence data to registers. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint16_t getSn_TX_FSR (uint8_t sn)</code></td>
<td>Get Sn_TX_FSR register. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint16_t getSn_RX_RSR (uint8_t sn)</code></td>
<td>Get Sn_RX_RSR register. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint16_t getSn_RxBASE (uint8_t sn)</code></td>
<td>Get the base address of socket sn RX buffer. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>uint16_t getSn_TxBASE (uint8_t sn)</code></td>
<td>Get the base address of socket sn TX buffer. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void wiz_send_data (uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to internal TX memory. <a href="#">More...</a></td>
</tr>
<tr>
<td><code>void wiz_recv_data (uint8_t sn, uint8_t *wizdata, uint16_t len)</code></td>
<td>It copies data to your buffer from internal RX memory. <a href="#">More...</a></td>
</tr>
</tbody>
</table>
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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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.h.
Macro Definition Documentation

```c
#define _WIZCHIP_SN_BASE_  (0x4000)
```
Definition at line 50 of file `w5200.h`.

```c
#define _WIZCHIP_SN_SIZE_  (0x0100)
```
Definition at line 51 of file `w5200.h`.

```c
#define _WIZCHIP_IO_TXBUF_  (0x8000) /* Internal Tx buffer address of the iinchip */
```
Definition at line 52 of file `w5200.h`.

```c
#define _WIZCHIP_IO_RXBUF_  (0xC000) /* Internal Rx buffer address of the iinchip */
```
Definition at line 53 of file `w5200.h`.

```c
#define _W5200_SPI_READ_  (0x00 << 7)
```
SPI interface Read operation in Control Phase.
Definition at line 55 of file `w5200.h`.

```c
#define _W5200_SPI_WRITE_  (0x01 << 7)
```
SPI interface Write operation in Control Phase.

Definition at line 56 of file \texttt{w5200.h}.

\begin{verbatim}
#define WIZCHIP_CREG_BLOCK 0x00
\end{verbatim}

Common register block.

Definition at line 58 of file \texttt{w5200.h}.

\begin{verbatim}
#define WIZCHIP_SREG_BLOCK (N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_*N)
\end{verbatim}

Socket \textit{N} register block.

Definition at line 59 of file \texttt{w5200.h}.

\begin{verbatim}
#define WIZCHIP_OFFSET_INC (ADDR, N) (ADDR + N)
\end{verbatim}

Increase offset address.

Definition at line 61 of file \texttt{w5200.h}.

\begin{verbatim}
#define IINCHIP_READ (ADDR) WIZCHIP_READ(ADDR)
\end{verbatim}

The defined for legacy chip driver.

Definition at line 75 of file \texttt{w5200.h}.

\begin{verbatim}
#define IINCHIP_WRITE (ADDR, N)
\end{verbatim}
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.


**#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 694 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 705 of file **w5200.h**.

---

**#define MR_PPPOE  0x08**

Enable PPPoE.
0 : Disable PPPoE mode  
1 : Enable PPPoE mode  
If you use ADSL, this bit should be '1'.

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-increment  
At the Indirect Bus Interface mode, if this bit is set as 111, the address will be automatically increased by 1 whenever read and write are performed.

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 111, Indirect Bus Interface mode is set.

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.
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
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)

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

Support UDP Multicasting.

# define Sn_MR_MC Sn_MR_ND

Definition at line 814 of file `w5200.h`.

# define Sn_MR_MF 0x40

Definition at line 825 of file `w5200.h`.

# define Sn_MR_MFEN Sn_MR_MF

Definition at line 834 of file `w5200.h`.

# define Sn_MR_MULTI 0x80

Definition at line 835 of file `w5200.h`.

# define Sn_MR_MULTI 0x80

Definition at line 814 of file `w5200.h`. 
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`.

```c
#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`.

<table>
<thead>
<tr>
<th>Sn_MR (P[3:0])</th>
<th>Sn_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR_CLOSE (000)</td>
<td>–</td>
</tr>
<tr>
<td>Sn_MR_TCP (001)</td>
<td>SOCK_INIT (0x13)</td>
</tr>
<tr>
<td>Sn_MR_UDP (010)</td>
<td>SOCK_UDP (0x22)</td>
</tr>
<tr>
<td>S0_MR_IPRAW (011)</td>
<td>SOCK_IPRAW (0x32)</td>
</tr>
<tr>
<td>S0_MR_MACRAW (100)</td>
<td>SOCK_MACRAW (0x42)</td>
</tr>
<tr>
<td>S0_MR_PPPoE (101)</td>
<td>SOCK_PPPoE (0x5F)</td>
</tr>
</tbody>
</table>

initialize or open socket

Definition at line 863 of file `w5200.h`.

```c
#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 request 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.

```c
#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.

```c
#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.

```c
#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.
```
#define Sn_IR_SENDOK  0x10
SEND_OK Interrupt.
This is issued when SEND command is completed.
Definition at line 988 of file w5200.h.

#define Sn_IR_TIMEOUT  0x08
TIMEOUT Interrupt.
This is issued when ARPTO or TCPTO occurs.
Definition at line 994 of file w5200.h.

#define Sn_IR_RECV  0x04
RECV Interrupt.
This is issued whenever data is received from a peer.
Definition at line 1000 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 1006 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**.

```c
#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**.

```c
#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[TIMOUT] = '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 transferred without the connection-process.

udp socket

Definition at line 1115 of file w5200.h.

#define SOCK_IPRAW 0x32

IP raw mode socket.

The 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]) = 100 and 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
`#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.

`#define IPPROTO_IP 0`

Dummy for IP.

Definition at line 1140 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.

```c
#define setSIR(ir2) setIR2(ir2)
```

Definition at line 1499 of file w5200.h.

Referenced by wizchip_clrinterrupt().

```c
#define getSIR() getIR2()
```

Definition at line 1509 of file w5200.h.

Referenced by wizchip_getinterrupt().

```c
#define setSIMR(imr2) setIMR2(imr2)
```

Definition at line 1533 of file w5200.h.

Referenced by wizchip_setinterruptmask().

```c
#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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>W5300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

W5300 Directory Reference
## Files

<table>
<thead>
<tr>
<th>file</th>
<th>w5300.c [code]</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>w5300.h [code]</td>
</tr>
</tbody>
</table>

W5300 HAL implement File.

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.doxygen.org) 1.8.9.1
Socket APIs

w5300.c File Reference

#include <stdint.h> #include "wizchip_conf.h"

Go to the source code of this file.
Socket APIs

W5300 HAL implement File. More...

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

Go to the source code of this file.
### Macros

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define _WIZCHIP_SN_BASE_ (0x0200)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_SN_SIZE_ (0x0040)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_CREG_BLOCK 0x00</code></td>
<td>Common register block.</td>
</tr>
<tr>
<td><code>#define WIZCHIP_SREG_BLOCK (N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_ * N)</code></td>
<td>Socket N register block.</td>
</tr>
<tr>
<td><code>#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)</code></td>
<td>Increase offset address.</td>
</tr>
<tr>
<td><code>#define _W5300_IO_BASE_ _WIZCHIP_IO_BASE_</code></td>
<td></td>
</tr>
<tr>
<td><code>#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)</code></td>
<td>The defined for legacy chip driver.</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR, VAL)</code></td>
<td>The defined for legacy chip driver.</td>
</tr>
<tr>
<td><code>#define MR (_WIZCHIP_IO_BASE_)</code></td>
<td>Mode Register address(R/W)</td>
</tr>
<tr>
<td><code>#define IR (_W5300_IO_BASE_ + 0x02)</code></td>
<td>Interrupt Register(R/W)</td>
</tr>
<tr>
<td><code>#define _IMR_ (_W5300_IO_BASE_ + 0x04)</code></td>
<td>Socket Interrupt Mask Register(R/W)</td>
</tr>
<tr>
<td><code>#define SHAR (_W5300_IO_BASE_ + 0x08)</code></td>
<td>Source MAC Register address(R/W)</td>
</tr>
</tbody>
</table>
#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  **TMS0R**  TMS01R
TX memory size of socket 0.  More...

#define  **TMS1R**  (TMS0R + 1)
TX memory size of socket 1.  More...

#define  **TMS2R**  (TMS0R + 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...
<table>
<thead>
<tr>
<th>Define</th>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSR2</td>
<td>RMSR0 + 2</td>
<td>RX memory size of socket 2. More...</td>
</tr>
<tr>
<td>RMSR3</td>
<td>RMSR0 + 3</td>
<td>RX memory size of socket 3. More...</td>
</tr>
<tr>
<td>RMSR4</td>
<td>RMSR0 + 4</td>
<td>RX memory size of socket 4. More...</td>
</tr>
<tr>
<td>RMSR5</td>
<td>RMSR0 + 5</td>
<td>RX memory size of socket 5. More...</td>
</tr>
<tr>
<td>RMSR6</td>
<td>RMSR0 + 6</td>
<td>RX memory size of socket 6. More...</td>
</tr>
<tr>
<td>RMSR7</td>
<td>RMSR0 + 7</td>
<td>RX memory size of socket 7. More...</td>
</tr>
<tr>
<td>MTYPER</td>
<td><em>W5300_IO_BASE</em> + 0x30</td>
<td>Memory Type Register. More...</td>
</tr>
<tr>
<td>PATR</td>
<td><em>W5300_IO_BASE</em> + 0x32</td>
<td>PPPoE Authentication Type register. More...</td>
</tr>
<tr>
<td>PTIMER</td>
<td><em>W5300_IO_BASE</em> + 0x36</td>
<td>PPP Link Control Protocol Request Timer Register. More...</td>
</tr>
<tr>
<td>PMAGICR</td>
<td><em>W5300_IO_BASE</em> + 0x38</td>
<td>PPP LCP magic number register. More...</td>
</tr>
<tr>
<td>PSIDR</td>
<td><em>W5300_IO_BASE</em> + 0x3C</td>
<td>PPPoE session ID register. More...</td>
</tr>
<tr>
<td>PDHAR</td>
<td><em>W5300_IO_BASE</em> + 0x40</td>
<td>PPPoE destination hardware address register. More...</td>
</tr>
</tbody>
</table>
#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 + n*4)  
Socket Mode register(R/W) More...

#define Sn_CR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + n*4)  
Socket command register(R/W) More...

#define Sn_IMR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + n*4)  
Socket interrupt mask register(R) More...

#define Sn_IR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + n*4)  
Socket interrupt register(R) More...

#define Sn_SSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + n*4)  
Socket status register(R) More...
<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define Sn_SR(n) Sn_SSR(n)</code></td>
<td>For Compatible ioLibrary. Refer to <code>Sn_SSR(n)</code> More...</td>
</tr>
<tr>
<td><code>#define Sn_PORTR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + 0x0A)</code> source port register(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_PORT(n) Sn_PORTR(n)</code></td>
<td>For compatible ioLibrary. Refer to <code>Sn_PORTR(n)</code> More...</td>
</tr>
<tr>
<td><code>#define Sn_DHAR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK)</code> Peer MAC register address(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_DPORTR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + 0x12)</code> Peer port register address(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_DPORT(n) Sn_DPORTR(n)</code></td>
<td>For compatible ioLibrary. Refer to <code>Sn_DPORTR(n)</code> More...</td>
</tr>
<tr>
<td><code>#define Sn_DIPR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK)</code> Peer IP register address(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_MSSR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK)</code> Maximum Segment Size(Sn_MSSR0)</td>
</tr>
<tr>
<td></td>
<td>register address(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_KPALVTR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK + 0x1A)</code> Keep Alive Timer register(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
<tr>
<td><code>#define Sn_PROTOR(n) Sn_KPALVTR(n)</code></td>
<td>IP Protocol(PROTO) Register(R/W) More...</td>
</tr>
<tr>
<td><code>#define Sn_TOSR(n)</code></td>
<td><code>(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK)</code> IP Type of Service(TOS) Register(R/W)</td>
</tr>
<tr>
<td></td>
<td>More...</td>
</tr>
</tbody>
</table>
#define Sn_TOS(n) Sn_TOSR(n)
For compatible ioLibrary. Refer to Sn_TOSR. More...

#define Sn_TTLR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK_0x20)
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_BLOCK_0x20)
SOCKETn TX write size register(R/W) More...

#define Sn_TX_FSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK_0x024)
Transmit free memory size register(R) More...

#define Sn_RX_RSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK_0x028)
Received data size register(R) More...

#define Sn_FRAGR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK_0x02C)
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_BLOCK_0x2E)
SOCKET n TX FIFO regsiter. More...

#define Sn_RX_FIFOR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK_0x30)
SOCKET n RX FIFO register. More...

#define MR_DBW (1 << 15)
```c
#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)
```
```markdown
<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Sn_MR_ALIGN</code></td>
<td>1 &lt;&lt; 8</td>
<td>Alignment bit of <code>Sn_MR</code>. More...</td>
</tr>
<tr>
<td><code>Sn_MR_MULTI</code></td>
<td>1 &lt;&lt; 7</td>
<td>Multicasting bit of <code>Sn_MR</code>. More...</td>
</tr>
<tr>
<td><code>Sn_MR_MF</code></td>
<td>1 &lt;&lt; 6</td>
<td>MAC filter bit of <code>Sn_MR</code>. More...</td>
</tr>
<tr>
<td><code>Sn_MR_IGMPv</code></td>
<td>1 &lt;&lt; 5</td>
<td>IGMP version bit of <code>Sn_MR</code> details. It is valid in case of <code>Sn_MR</code> and UDP(<code>Sn_MR_UDP</code>). It configures IGMP version to send IGMP messages such as <code>Join/Leave/Report</code> to multicast-group. 0 : IGMPv2, 1 : IGMPv1. More...</td>
</tr>
<tr>
<td><code>Sn_MR_MC</code></td>
<td><code>Sn_MR_IGMPv</code></td>
<td>For compatible ioLibrary. More...</td>
</tr>
<tr>
<td><code>Sn_MR_ND</code></td>
<td>1 &lt;&lt; 5</td>
<td>No delayed ack bit of <code>Sn_MR</code>. More...</td>
</tr>
<tr>
<td><code>Sn_MR_CLOSE</code></td>
<td>0x00</td>
<td>No mode. More...</td>
</tr>
<tr>
<td><code>Sn_MR_TCP</code></td>
<td>0x01</td>
<td>TCP mode. More...</td>
</tr>
<tr>
<td><code>Sn_MR_UDP</code></td>
<td>0x02</td>
<td>UDP mode. More...</td>
</tr>
<tr>
<td><code>Sn_MR_IPRAW</code></td>
<td>0x03</td>
<td>IP LAYER RAW mode. More...</td>
</tr>
<tr>
<td><code>Sn_MR_MACRAW</code></td>
<td>0x04</td>
<td>MAC LAYER RAW mode. More...</td>
</tr>
</tbody>
</table>
```
#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
<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_CR_PDISCON</td>
<td>0x24</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_PCR</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_PCN</td>
<td>0x26</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_PCJ</td>
<td>0x27</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_PRECV</td>
<td>0x80</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_PFAIL</td>
<td>0x40</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_PNEXT</td>
<td>0x20</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_SENDOK</td>
<td>0x10</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_TIMEOUT</td>
<td>0x08</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_RECV</td>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_DISCON</td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_CON</td>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>SO_CLOSED</td>
<td>0x00</td>
<td>The state of SOCKET initialized or closed. <a href="#">More...</a></td>
</tr>
<tr>
<td>SO_ARP</td>
<td>0x01</td>
<td>The state of ARP process. <a href="#">More...</a></td>
</tr>
<tr>
<td>SO_INIT</td>
<td>0x13</td>
<td>Initiate state in TCP. <a href="#">More...</a></td>
</tr>
<tr>
<td>SO_LISTEN</td>
<td>0x14</td>
<td>Listen state. <a href="#">More...</a></td>
</tr>
<tr>
<td>Define Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SOCK_SYNSENT</td>
<td>0x15</td>
<td>Connection state.</td>
</tr>
<tr>
<td>SOCK_SYNRECV</td>
<td>0x16</td>
<td>Connection state.</td>
</tr>
<tr>
<td>SOCK_ESTABLISHED</td>
<td>0x17</td>
<td>Success to connect.</td>
</tr>
<tr>
<td>SOCK_FIN_WAIT</td>
<td>0x18</td>
<td>Closing state.</td>
</tr>
<tr>
<td>SOCK_CLOSING</td>
<td>0x1A</td>
<td>Closing state.</td>
</tr>
<tr>
<td>SOCK_TIME_WAIT</td>
<td>0x1B</td>
<td>Closing state.</td>
</tr>
<tr>
<td>SOCK_CLOSE_WAIT</td>
<td>0x1C</td>
<td>Closing state.</td>
</tr>
<tr>
<td>SOCK_LAST_ACK</td>
<td>0x1D</td>
<td>Closing state.</td>
</tr>
<tr>
<td>SOCK_UDP</td>
<td>0x22</td>
<td>UDP socket.</td>
</tr>
<tr>
<td>SOCK_IPRAW</td>
<td>0x32</td>
<td>IP raw mode socket.</td>
</tr>
<tr>
<td>SOCK_MACRAW</td>
<td>0x42</td>
<td>MAC raw mode socket.</td>
</tr>
<tr>
<td>SOCK_PPPoE</td>
<td>0x5F</td>
<td>PPPoE mode socket.</td>
</tr>
</tbody>
</table>
#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)&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 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...
<table>
<thead>
<tr>
<th>#define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define setRMS23R(rms23r) WIZCHIP_WRITE(RMS23R, rms23r)</code></td>
<td>Set RMS23R register. More...</td>
</tr>
<tr>
<td><code>#define getRMS23R() WIZCHIP_READ(RMS23R)</code></td>
<td>Get RMS23R register. More...</td>
</tr>
<tr>
<td><code>#define setRMS45R(rms45r) WIZCHIP_WRITE(RMS45R, rms45r)</code></td>
<td>Set RMS45R register. More...</td>
</tr>
<tr>
<td><code>#define getRMS45R() WIZCHIP_READ(RMS45R)</code></td>
<td>Get RMS45R register. More...</td>
</tr>
<tr>
<td><code>#define setRMS67R(rms67r) WIZCHIP_WRITE(RMS67R, rms67r)</code></td>
<td>Set RMS67R register. More...</td>
</tr>
<tr>
<td><code>#define getRMS67R() WIZCHIP_READ(RMS67R)</code></td>
<td>Get RMS67R register. More...</td>
</tr>
<tr>
<td><code>#define setSn_RXBUF_SIZE(sn, rmsr) setRMSR(sn, rmsr)</code></td>
<td>For compatible ioLibrary. More...</td>
</tr>
<tr>
<td><code>#define getSn_RXBUF_SIZE(sn) getRMSR(sn)</code></td>
<td>For compatible ioLibrary. More...</td>
</tr>
<tr>
<td><code>#define setMTYPER(mtype) WIZCHIP_WRITE(MTYPER, mtype)</code></td>
<td>Set MTYPER register. More...</td>
</tr>
<tr>
<td><code>#define getMTYPER() WIZCHIP_READ(MTYPER)</code></td>
<td>Get MTYPER register. More...</td>
</tr>
<tr>
<td><code>#define getPATR() WIZCHIP_READ(PATR)</code></td>
<td>Get RATR register. More...</td>
</tr>
<tr>
<td><code>#define setPTIMER(ptimer) WIZCHIP_WRITE(PTIMER, ((uint16_t)ptimer) &amp; 0x00FF)</code></td>
<td>For compatible ioLibrary. More...</td>
</tr>
</tbody>
</table>
Set PTIMER register. More...

```c
#define getPTIMER() ((uint8_t)(WIZCHIP_READ(PTIMER) & 0x00FF)
Get PTIMER register. More...
```

```c
#define setPMAGIC(pmagic) WIZCHIP_WRITE(PMAGIC, ((uint16_t)(pmagic)
Set PMAGIC register. More...
```

```c
#define getPMAGIC() ((uint8_t)(WIZCHIP_READ(PMAGIC) & 0x00FF))
Get PMAGIC register. More...
```

```c
#define getPSIDR() WIZCHIP_READ(PSIDR)
Get PSID register. More...
```

```c
#define getPDHAR(pdhar)
Get PDHAR register. More...
```

```c
#define getUIPR(uipr)
Get unreachable IP address. UIPR. More...
```

```c
#define getUPORT() WIZCHIP_READ(UPORT)
Get UPORTR register. More...
```

```c
#define getFMTUR() WIZCHIP_READ(FMTUR)
Get FMTUR register. More...
```

```c
#define getPn_BRDYR(p) ((uint8_t)(WIZCHIP_READ(Pn_BRDYR(p))
Get Pn_BRDYR register. More...
```

```c
#define setPn_BRDYR(p, brdyr) WIZCHIP_WRITE(Pn_BRDYR(p), brdyr & 0x00E7)
Set Pn_BRDYR register. More...
```

```c
#define getPn_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...

#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)cr) & 0x00FF)
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_t)imr) & 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) & 0x00FF)
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...
<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define setSn_PORTR(sn, port)</td>
<td><strong>WIZCHIP_WRITE</strong>(Sn_PORTR(sn))&lt;br&gt;Set <strong>Sn_PORTR</strong> register. More...</td>
</tr>
<tr>
<td>#define setSn_PORT(sn, port)</td>
<td><strong>setSn_PORTR</strong>(sn, port)&lt;br&gt;For compatible ioLibrary. More...</td>
</tr>
<tr>
<td>#define getSn_PORTR(sn, port)</td>
<td><strong>WIZCHIP_READ</strong>(Sn_PORTR(sn))&lt;br&gt;Get <strong>Sn_PORTR</strong> register. More...</td>
</tr>
<tr>
<td>#define getSn_PORT(sn)</td>
<td><strong>getSn_PORTR</strong>(sn)&lt;br&gt;For compatible ioLibrary. More...</td>
</tr>
<tr>
<td>#define setSn_DHAR(sn, dhar)</td>
<td>Set <strong>Sn_DHAR</strong> register. More...</td>
</tr>
<tr>
<td>#define getSn_DHAR(sn, dhar)</td>
<td>Get <strong>Sn_MR</strong> register. More...</td>
</tr>
<tr>
<td>#define setSn_DPORTR(sn, dport)</td>
<td><strong>WIZCHIP_WRITE</strong>(Sn_DPORTR(sn), dport)&lt;br&gt;Set <strong>Sn_DPORT</strong> register. More...</td>
</tr>
<tr>
<td>#define setSn_DPORT(sn, dport)</td>
<td><strong>setSn_DPORTR</strong>(sn, dport)&lt;br&gt;For compatible ioLibrary. Refer to <strong>Sn_DPORTR</strong>. More...</td>
</tr>
<tr>
<td>#define getSn_DPORTR(sn)</td>
<td><strong>WIZCHIP_READ</strong>(Sn_DPORTR(sn))&lt;br&gt;Get <strong>Sn_DPORT</strong> register. More...</td>
</tr>
<tr>
<td>#define getSn_DPORT(sn)</td>
<td><strong>getSn_DPORTR</strong>(sn)&lt;br&gt;For compatible ioLibrary. Refer to <strong>Sn_DPORTR</strong>. More...</td>
</tr>
<tr>
<td>#define setSn_DIPR(sn, dipr)</td>
<td>Set <strong>Sn_DIPR</strong> register. More...</td>
</tr>
<tr>
<td>#define getSn_DIPR(sn, dipr)</td>
<td>Get <strong>Sn_DIPR</strong> register. More...</td>
</tr>
</tbody>
</table>
```c
#define setSn_MSSR(sn, mss) WIZCHIP_WRITE(Sn_MSSR(sn), mss)
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_KPALVTR(sn),
(WIZCHIP_READ(Sn_KPALVTR(sn)) & 0x00FF) | (((uint16_t)kpalvt)<<8))
Set Sn_KPALVTR register. More...

#define getSn_KPALVTR(sn) ((uint8_t)(WIZCHIP_READ(Sn_KPALVTR(sn)))
Get Sn_KPALVTR register. More...

#define setSn_PROTOR(sn, proto) WIZCHIP_WRITE(Sn_PROTOR(sn),
(WIZCHIP_READ(Sn_PROTOR(sn)) & 0xFF00) | (((uint16_t)proto)& 0x00FF))
Set Sn_PROTOR register. More...

#define setSn_PROTO(sn, proto) setSn_PROTOR(sn,proto)
For compatible ioLibrary. More...

#define getSn_PROTO(sn) WIZCHIP_READ(Sn_PROTOR(sn))
Get Sn_PROTO register. More...

#define getSn_PROTOR(sn) ((uint8_t)WIZCHIP_READ(Sn_PROTOR(sn))
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_WRSR(sn))<16) +
(((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn)) & 0x0000FFFF))
Get Sn_TX_WRSR register. More...
```
```c
#define setSn_TX_FIFOR(sn, txfifo) WIZCHIP_WRITE(Sn_TX_FIFO, (uint16_t)txfifo)
Set Sn_TX_FIFOR register. More...

#define getSn_RX_FIFOR(sn) WIZCHIP_READ(Sn_RX_FIFOR, (uint16_t)sn)
Get Sn_RX_FIFOR register. More...

#define setSn_TOSR(sn, tos) WIZCHIP_WRITE(Sn_TOSR, (uint16_t)tos & 0x00FF)
Set Sn_TOSR register. More...

#define setSn_TOS(sn, tos) setSn_TOSR(sn, tos)
For compatible ioLibrary. More...

#define getSn_TOSR(sn) ((uint8_t)WIZCHIP_READ(Sn_TOSR, (uint16_t)sn))
Get Sn_TOSR register. More...

#define getSn_TOS(sn) getSn_TOSR(sn)
For compatible ioLibrary. More...

#define setSn_TTLR(sn, ttl) WIZCHIP_WRITE(Sn_TTLR, (uint16_t)ttl & 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_TTLR, (uint16_t)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, (uint16_t)frag >> 8)
Set Sn_FRAGR register. More...
```

More information on each macro can be found in the documentation.
```c
#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)) << 10)
Socket_register_access_function_W5300. More...
#define getSn_TxMAX(sn) (((uint32_t)getSn_TXBUF_SIZE(sn)) << 10)
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](https://github.com/Wiznet/ioLibrary_Driver)

**Author**

MidnightCow

**Copyright**

Copyright (c) 2015, WIZnet Co., LTD. All rights reserved.

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 above 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 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`.

```c
#define Sn_PORT ( n ) Sn_PORTR(n)
```
For compatible ioLibrary. Refer to `Sn_PORTR(n)`.
Definition at line 735 of file `w5300.h`.

```c
#define Sn_DPORT ( n ) Sn_DPORTR(n)
```
For compatible ioLibrary. Refer to `Sn_DPORTR`.
Definition at line 754 of file `w5300.h`.

```c
#define Sn_TOS ( n ) Sn_TOSR(n)
```
For compatible ioLibrary. Refer to `Sn_TOSR`.
Definition at line 803 of file `w5300.h`.

```c
#define Sn_TTL ( n ) Sn_TTLR(n)
```
For compatible ioLibrary. Refer to `Sn_TTLR`.
Definition at line 812 of file `w5300.h`.

```c
#define (_W5300_IO_BASE_ + Sn_RX_RSR ( n ) 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.

```c
#define Sn_FRAG (n) Sn_FRAGR(n)
```

Definition at line 849 of file w5300.h.

```c
#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.

```c
#define MR_MPF (1 << 14)
```

Mac layer pause frame bit of MR. (0 : Disable, 1 : Enable)

Definition at line 890 of file w5300.h.

```c
#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.

```c
#define MR_RDH (1 << 10)
```
Read data hold time bit of **MR**. Hold Data on DATA bus during \(2 \times \text{PLL}_{-}\text{CLK}\) after CS high

Definition at line 892 of file **w5300.h**.

```c
#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()`.

```c
#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**.

```c
#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**.

```c
#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.

```c
#define Pn_SN ( n ) ((n & 0x07) << 0)
```

What socket to monitor.

Definition at line 917 of file w5300.h.

```c
#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().

```c
#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.

```c
#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.
#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.

<table>
<thead>
<tr>
<th>Sn_MR (P[3:0])</th>
<th>Sn_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR_CLOSE (000)</td>
<td></td>
</tr>
<tr>
<td>Sn_MR_TCP (001)</td>
<td>SOCK_INIT (0x13)</td>
</tr>
<tr>
<td>Sn_MR_UDP (010)</td>
<td>SOCK_UDP (0x22)</td>
</tr>
<tr>
<td>Sn_MR_IPRAW (010)</td>
<td>SOCK_IPRAW (0x32)</td>
</tr>
<tr>
<td>Sn_MR_MACRAW (100)</td>
<td>SOCK_MACRAW (0x42)</td>
</tr>
<tr>
<td>Sn_MR_PPPoE (101)</td>
<td>SOCK_PPPoE (0x5F)</td>
</tr>
</tbody>
</table>

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 \((\text{Sn}_\text{MR}(P3:P0) = \text{Sn}_\text{MR}_{TCP})\). In this mode, Socket n operates as a TCP server and waits for connection-request (SYN packet) from any TCP client. The \(\text{Sn}_\text{SR}\) changes the state from \text{SOCK\_INIT} to \text{SOCKET\_LISTEN}. When a TCP client connection request is successfully established, the \(\text{Sn}_\text{SR}\) changes from \text{SOCKET\_LISTEN} to \text{SOCK\_ESTABLISHED} and the \(\text{Sn}_\text{IR}(0)\) becomes. But when a TCP client connection request is failed, \(\text{Sn}_\text{IR}(3)\) becomes and the status of \(\text{Sn}_\text{SR}\) changes to \text{SOCK\_CLOSED}.

Definition at line 1045 of file \texttt{w5300.h}.

\#define \texttt{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 \(\text{Sn}_\text{DIPR} \& \text{Sn}_\text{DPORT}\) (destination address & port). If the connect-request is successful, the \(\text{Sn}_\text{SR}\) is changed to \text{SOCK\_ESTABLISHED} and the \(\text{Sn}_\text{IR}(0)\) becomes.

The connect-request fails in the following three cases.

1. When a \text{ARPTO} occurs \((\text{Sn}_\text{IR}[3] = '1')\) because destination hardware address is not acquired through the ARP-process.
2. When a \text{SYN/ACK} packet is not received and \text{TCPTO} \((\text{Sn}_\text{IR}(3) = )\)
3. When a \text{RST} packet is received instead of a \text{SYN/ACK} packet. In these cases, \(\text{Sn}_\text{SR}\) is changed to \text{SOCK\_CLOSED}.

\textbf{Note}  
This is valid only in TCP mode and operates when Socket n acts as TCP client.

Definition at line 1057 of file \texttt{w5300.h}.

\#define \texttt{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 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.

Definition at line 1070 of file `w5300.h`.

```c
#define Sn_CR_CLOSE 0x10
```

Close socket.

`Sn_SR` is changed to **SOCK_CLOSED**.

Definition at line 1076 of file `w5300.h`.

```c
#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 occurred. 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 initialized 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.
### #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 1137 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 1144 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_ESTABLISHED 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**.

```c
#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**.

```c
#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**.

```c
#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*.

```c
#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*.

```c
#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*.

```c
#define SOCK_UDP 0x22
```
UDP socket.

This indicates Socket n is opened in UDP mode ($Sn_{\text{MR}}(P[3:0]) = '010'$). It changes to SOCK_UDP when $Sn_{\text{MR}}(P[3:0]) = '010'$ and $Sn_{\text{CR\_OPEN}}$ command is ordered. Unlike TCP mode, data can be transferred without the connection-process.

Definition at line 1233 of file w5300.h.

#define SOCK_IPRAW 0x32

IP raw mode socket.

The socket is opened in IPRAW mode. The SOCKET status is change to SOCK_IPRAW when $Sn_{\text{MR}}$ (P3:P0) is $Sn_{\text{MR\_IPRAW}}$ and $Sn_{\text{CR\_OPEN}}$ 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 ($Sn_{\text{MR}}(P[3:0]) = '100'$ and n = 0) and is valid only in Socket 0. It changes to SOCK_MACRAW when $Sn_{\text{MR}}(P[3:0] = 100)$and @ ref $Sn_{\text{CR\_OPEN}}$ 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.
For compatible ioLibrary.

Definition at line \textbf{1627} of file \texttt{w5300.h}.

\begin{verbatim}
#define setSn_RXBUF_SIZE ( sn, rmsr )
        setRMSR(sn, rmsr)
\end{verbatim}

For compatible ioLibrary.

Definition at line \textbf{1709} of file \texttt{w5300.h}.

\begin{verbatim}
#define getSn_RXBUF_SIZE ( sn )
        getRMSR(sn)
\end{verbatim}

For compatible ioLibrary.

Definition at line \textbf{1719} of file \texttt{w5300.h}.

\begin{verbatim}
#define getSn_SR ( sn )
        getSn_SSR(sn)
\end{verbatim}

For compatible ioLibrary. Refer to \texttt{getSn_SSR()}. 

Definition at line \textbf{1971} of file \texttt{w5300.h}.

\begin{verbatim}
#define setSn_PORT ( sn, port )
        setSn_PORTR(sn, port)
\end{verbatim}

For compatible ioLibrary.

Definition at line \textbf{1982} of file \texttt{w5300.h}.

\begin{verbatim}
#define getSn_PORT ( sn )
        getSn_PORTR(sn)
\end{verbatim}
For compatible ioLibrary.
Definition at line 1993 of file w5300.h.

```c
#define setSn_DPORT ( sn, dport )
    setSn_DPORTR(sn,dport)
```

For compatible ioLibrary. Refer to `Sn_DPORTR`.
Definition at line 2033 of file w5300.h.

```c
#define getSn_DPORT ( sn )
    getSn_DPORTR(sn)
```

For compatible ioLibrary. Refer to `Sn_DPORTR`.
Definition at line 2047 of file w5300.h.

```c
#define setSn_PROTO ( sn, proto )
    setSn_PROTOR(sn,proto)
```

For compatible ioLibrary.
Definition at line 2124 of file w5300.h.

```c
#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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet W5500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### W5500 Directory Reference
## Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w5500.c</code> [code]</td>
<td>W5500 HAL Interface.</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.stackexchange.com) 1.8.9.1
Socket APIs

W5500 HAL Interface. More...

#include "w5500.h"

Go to the source code of this file.
### Macros

<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define <em>W5500_SPI_VDM_OP</em></td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>#define <em>W5500_SPI_FDM_OP_LEN1</em></td>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>#define <em>W5500_SPI_FDM_OP_LEN2</em></td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>#define <em>W5500_SPI_FDM_OP_LEN4</em></td>
<td>0x03</td>
<td></td>
</tr>
</tbody>
</table>
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
2. WIZCHIP_READ_BUF WIZCHIP_WRITE_BUF in case WIZCHIP_IO_MODE_SPI_FDM for loop optimized(removed). refer to M20131220 <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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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
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

```c
#define _W5500_SPI_VDM_OP_ 0x00
```
Definition at line 57 of file `w5500.c`.

```c
#define _W5500_SPI_FDM_OP_LEN1_ 0x01
```
Definition at line 58 of file `w5500.c`.

```c
#define _W5500_SPI_FDM_OP_LEN2_ 0x02
```
Definition at line 59 of file `w5500.c`.

```c
#define _W5500_SPI_FDM_OP_LEN4_ 0x03
```
Definition at line 60 of file `w5500.c`.

Generated on Wed May 4 2016 16:43:59 for Socket APIs by `doxygen` 1.8.9.1
Socket APIs

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

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define _W5500_IO_BASE_ 0x00000000</code></td>
<td></td>
</tr>
<tr>
<td><code>#define _W5500_SPI_READ_ (0x00 &lt;&lt; 2)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define _W5500_SPI_WRITE_ (0x01 &lt;&lt; 2)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_CREG_BLOCK 0x00</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_SREG_BLOCK(N) (1+4*N)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_TXBUF_BLOCK(N) (2+4*N)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_RXBUF_BLOCK(N) (3+4*N)</code></td>
<td></td>
</tr>
<tr>
<td><code>#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + (N&lt;&lt;8))</code></td>
<td></td>
</tr>
<tr>
<td><code>#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR, VAL)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_READ_BUF(ADDR, BUF, LEN) WIZCHIP_READ_BUF(ADDR, BUF, LEN)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define IINCHIP_WRITE_BUF(ADDR, BUF, LEN) WIZCHIP_WRITE_BUF(ADDR, BUF, LEN)</code></td>
<td>The defined for legacy chip driver. More...</td>
</tr>
<tr>
<td><code>#define MR (_W5500_IO_BASE_ + (0x0000 &lt;&lt; 8) + (WIZCHIP_CREG_BLOCK &lt;&lt; 3))</code></td>
<td>Mode Register address(R/W)</td>
</tr>
<tr>
<td><strong>MR</strong></td>
<td><strong>MR</strong> is used for S/W reset, ping block mode, PPPoE mode and</td>
</tr>
</tbody>
</table>

The defined for legacy chip driver. More...
```c
#define GAR (_W5500_IO_BASE_ + (0x0001 << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Gateway IP Register address(R/W)

#define SUBR (_W5500_IO_BASE_ + (0x0005 << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Subnet mask Register address(R/W)

#define SHAR (_W5500_IO_BASE_ + (0x0009 << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Source MAC Register address(R/W)

#define SIPR (_W5500_IO_BASE_ + (0x000F << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Source IP Register address(R/W)

#define INTLEVEL (_W5500_IO_BASE_ + (0x0013 << 8) +
               (WIZCHIP_CREG_BLOCK << 3))
Set Interrupt low level timer register address(R/W)

#define IR (_W5500_IO_BASE_ + (0x0015 << 8) +
        (WIZCHIP_CREG_BLOCK << 3))
Interrupt Register(R/W)

#define _IMR_ (_W5500_IO_BASE_ + (0x0016 << 8) +
            (WIZCHIP_CREG_BLOCK << 3))
Interrupt mask register(R/W)

#define SIR (_W5500_IO_BASE_ + (0x0017 << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Register(R/W)

#define SIMR (_W5500_IO_BASE_ + (0x0018 << 8) +
          (WIZCHIP_CREG_BLOCK << 3))
Socket Interrupt Mask Register(R/W)
```

#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...
<table>
<thead>
<tr>
<th>#define</th>
<th>PHYCFGR</th>
<th>PHY Status Register(R/W) More...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VERSIONR</td>
<td>chip version register address(R) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_MR(N)</td>
<td>socket Mode register(R/W) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_CR(N)</td>
<td>Socket command register(R/W) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_IR(N)</td>
<td>Socket interrupt register(R) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_SR(N)</td>
<td>Socket status register(R) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_PORT(N)</td>
<td>source port register(R/W) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_DHAR(N)</td>
<td>Peer MAC register address(R/W) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_DIPR(N)</td>
<td>Peer IP register address(R/W) More...</td>
</tr>
<tr>
<td></td>
<td>Sn_DPORT(N)</td>
<td>More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_MSSR(N) = (<em>W5500_IO_BASE</em> + (0x0012 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Maximum Segment Size(Sn_MSSR) register address(R/W) More...</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TOS(N) = (<em>W5500_IO_BASE</em> + (0x0015 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>IP Type of Service(TOS) Register(R/W) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TTL(N) = (<em>W5500_IO_BASE</em> + (0x0016 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>IP Time to live(TTL) Register(R/W) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_RXBUF_SIZE(N) = (<em>W5500_IO_BASE</em> + (0x001E &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Receive memory size register(R/W) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TXBUF_SIZE(N) = (<em>W5500_IO_BASE</em> + (0x001F &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Transmit memory size register(R/W) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TX_FSR(N) = (<em>W5500_IO_BASE</em> + (0x0020 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Transmit free memory size register(R) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TX_RD(N) = (<em>W5500_IO_BASE</em> + (0x0022 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Transmit memory read pointer register address(R) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_TX_WR(N) = (<em>W5500_IO_BASE</em> + (0x0024 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td>Transmit memory write pointer register address(R/W) More...</td>
</tr>
<tr>
<td>#define</td>
<td>Sn_RX_RSR(N) = (<em>W5500_IO_BASE</em> + (0x0026 &lt;&lt; 8) + (WIZCHIP_SREG_BLOCK(N) &lt;&lt; 3))</td>
<td></td>
</tr>
</tbody>
</table>
(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...
```c
#define IR_CONFLICT 0x80
Check IP conflict. More...

#define IR_UNREACH 0x40
Get the destination unreachable message in UDP sending. More...

#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)
```
```c
#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
```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define Sn_MR_TCP</td>
<td>0x01</td>
<td>TCP. More...</td>
</tr>
<tr>
<td>#define Sn_MR_CLOSE</td>
<td>0x00</td>
<td>Unused socket. More...</td>
</tr>
<tr>
<td>#define Sn_MR_MFEN</td>
<td>Sn_MR_MULTI</td>
<td>MAC filter enable in Sn_MR_MACRAW mode. More...</td>
</tr>
<tr>
<td>#define Sn_MR_MMB</td>
<td>Sn_MR_ND</td>
<td>Multicast Blocking in Sn_MR_MACRAW mode. More...</td>
</tr>
<tr>
<td>#define Sn_MR_MIP6B</td>
<td>Sn_MR_UCASTB</td>
<td>IPv6 packet Blocking in Sn_MR_MACRAW mode. More...</td>
</tr>
<tr>
<td>#define Sn_MR_MC</td>
<td>Sn_MR_ND</td>
<td>IGMP version used in UDP multcasting. More...</td>
</tr>
<tr>
<td>#define SOCK_STREAM</td>
<td>Sn_MR_TCP</td>
<td>For Berkeley Socket API. More...</td>
</tr>
<tr>
<td>#define SOCK_DGRAM</td>
<td>Sn_MR_UDP</td>
<td>For Berkeley Socket API. More...</td>
</tr>
<tr>
<td>#define Sn_CR_OPEN</td>
<td>0x01</td>
<td>Initialize or open socket. More...</td>
</tr>
<tr>
<td>#define Sn_CR_LISTEN</td>
<td>0x02</td>
<td>Wait connection request in TCP mode(Server mode) More...</td>
</tr>
<tr>
<td>#define Sn_CR_CONNECT</td>
<td>0x04</td>
<td>Send connection request in TCP mode(Client mode) More...</td>
</tr>
<tr>
<td>#define Sn_CR_DISCON</td>
<td>0x08</td>
<td></td>
</tr>
<tr>
<td>Define Name</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_CLOSE</td>
<td>0x10</td>
<td></td>
</tr>
<tr>
<td>Close socket.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_SEND</td>
<td>0x20</td>
<td></td>
</tr>
<tr>
<td>Update TX buffer pointer and send data.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_SEND_MAC</td>
<td>0x21</td>
<td></td>
</tr>
<tr>
<td>Send data with MAC address, so without ARP process.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_SEND_KEEP</td>
<td>0x22</td>
<td></td>
</tr>
<tr>
<td>Send keep alive message.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_CR_RECV</td>
<td>0x40</td>
<td></td>
</tr>
<tr>
<td>Update RX buffer pointer and receive data.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_SENDOK</td>
<td>0x10</td>
<td></td>
</tr>
<tr>
<td>SEND_OK Interrupt.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_TIMEOUT</td>
<td>0x08</td>
<td></td>
</tr>
<tr>
<td>TIMEOUT Interrupt.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_RECV</td>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>RECV Interrupt.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_DISCON</td>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>DISCON Interrupt.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>Sn_IR_CON</td>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>CON Interrupt.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>SOCK_CLOSED</td>
<td>0x00</td>
<td></td>
</tr>
<tr>
<td>Closed.</td>
<td>More...</td>
<td></td>
</tr>
<tr>
<td>SOCK_INIT</td>
<td>0x13</td>
<td></td>
</tr>
<tr>
<td>Define Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td><code>SOCK_LISTEN</code></td>
<td>0x14</td>
<td>Listen state. More...</td>
</tr>
<tr>
<td><code>SOCK_SYNSENT</code></td>
<td>0x15</td>
<td>Connection state. More...</td>
</tr>
<tr>
<td><code>SOCK_SYNRECV</code></td>
<td>0x16</td>
<td>Connection state. More...</td>
</tr>
<tr>
<td><code>SOCK_ESTABLISHED</code></td>
<td>0x17</td>
<td>Success to connect. More...</td>
</tr>
<tr>
<td><code>SOCK_FIN_WAIT</code></td>
<td>0x18</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_CLOSING</code></td>
<td>0x1A</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_TIME_WAIT</code></td>
<td>0x1B</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_CLOSE_WAIT</code></td>
<td>0x1C</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_LAST_ACK</code></td>
<td>0x1D</td>
<td>Closing state. More...</td>
</tr>
<tr>
<td><code>SOCK_UDP</code></td>
<td>0x22</td>
<td>UDP socket. More...</td>
</tr>
<tr>
<td><code>SOCK_MACRAW</code></td>
<td>0x42</td>
<td>MAC raw mode socket. More...</td>
</tr>
<tr>
<td><code>IPPROTO_IP</code></td>
<td>0</td>
<td>IP protocol. More...</td>
</tr>
</tbody>
</table>
#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...
<table>
<thead>
<tr>
<th>#define</th>
<th>setSUBR(subr)</th>
<th>WIZCHIP_WRITE_BUF(SUBR, subr, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set subnet mask address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getSUBR(subr)</th>
<th>WIZCHIP_READ_BUF(SUBR, subr, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get subnet mask address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>setSHAR(shar)</th>
<th>WIZCHIP_WRITE_BUF(SHAR, shar, 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set local MAC address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getSHAR(shar)</th>
<th>WIZCHIP_READ_BUF(SHAR, shar, 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get local MAC address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>setSIPR(sipr)</th>
<th>WIZCHIP_WRITE_BUF(SIPR, sipr, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set local IP address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getSIPR(sipr)</th>
<th>WIZCHIP_READ_BUF(SIPR, sipr, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get local IP address. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>setINTLEVEL(intlevel)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set INTLEVEL register. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getINTLEVEL()</th>
<th>(((uint16_t)WIZCHIP_READ(INTLEVEL) &lt;&lt; WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL,1))))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get INTLEVEL register. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>setIR(ir)</th>
<th>WIZCHIP_WRITE(IR, (ir &amp; 0xF0))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set IR register. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getIR()</th>
<th>(WIZCHIP_READ(IR) &amp; 0xF0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get IR register. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>setIMR(imr)</th>
<th>WIZCHIP_WRITE(<em>IMR</em>, imr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set IMR register. More...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#define</th>
<th>getIMR()</th>
<th>WIZCHIP_READ(<em>IMR</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get IMR register. More...</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>setSIR(sir)</td>
<td>WIZCHIP_WRITE(SIR, sir)</td>
<td></td>
</tr>
<tr>
<td>getSIR()</td>
<td>WIZCHIP_READ(SIR)</td>
<td></td>
</tr>
<tr>
<td>setSIMR(simr)</td>
<td>WIZCHIP_WRITE(SIMR, simr)</td>
<td></td>
</tr>
<tr>
<td>getSIMR()</td>
<td>WIZCHIP_READ(SIMR)</td>
<td></td>
</tr>
<tr>
<td>setRTR(rtr)</td>
<td>WIZCHIP_WRITE(<em>RTR</em>, rtr)</td>
<td></td>
</tr>
<tr>
<td>getRTR()</td>
<td>(((uint16_t)WIZCHIP_READ(<em>RTR</em>) &lt;&lt; 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(<em>RTR</em>,1)))</td>
<td></td>
</tr>
<tr>
<td>setRCR(rcr)</td>
<td>WIZCHIP_WRITE(<em>RCR</em>, rcr)</td>
<td></td>
</tr>
<tr>
<td>getRCR()</td>
<td>WIZCHIP_READ(<em>RCR</em>)</td>
<td></td>
</tr>
<tr>
<td>setPTIMER(ptimer)</td>
<td>WIZCHIP_WRITE(PTIMER, ptimer)</td>
<td></td>
</tr>
<tr>
<td>getPTIMER()</td>
<td>WIZCHIP_READ(PTIMER)</td>
<td></td>
</tr>
<tr>
<td>setPMAGIC(pmagic)</td>
<td>WIZCHIP_WRITE(PMAGIC, pmagic)</td>
<td></td>
</tr>
<tr>
<td>getPMAGIC()</td>
<td>WIZCHIP_READ(PMAGIC)</td>
<td></td>
</tr>
</tbody>
</table>
Get `PMAGIC` register. More...

```c
#define setPHAR(phar) WIZCHIP_WRITE_BUF(PHAR, phar, 6)
Set `PHAR` address. More...
```

```c
#define getPHAR(phar) WIZCHIP_READ_BUF(PHAR, phar, 6)
Get `PHAR` address. More...
```

```c
#define setPSID(psid)
Set `PSID` register. More...
```

```c
#define getPSID() (((uint16_t)WIZCHIP_READ(PSID) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1)))
Get `PSID` register. More...
```

```c
#define setPMRU(pmru)
Set `PMRU` register. More...
```

```c
#define getPMRU() (((uint16_t)WIZCHIP_READ(PMRI) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRI,1)))
Get `PMRU` register. More...
```

```c
#define getUIPR(uipr) WIZCHIP_READ_BUF(UIPR,uipr,4)
Get unreachable IP address. More...
```

```c
#define getUPORTR() (((uint16_t)WIZCHIP_READ(UPORTR) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1)))
Get `UPORTR` register. More...
```

```c
#define setPHYCFGR(phycfgr) WIZCHIP_WRITE(PHYCFGR, phycfgr)
Set `PHYCFGR` register. More...
```

```c
#define getPHYCFGR() WIZCHIP_READ(PHYCFGR)
Get `PHYCFGR` register. More...
```

```c
#define getVERSIONR() WIZCHIP_READ(VERSIONR)
```

Get VERSIONR register. More...

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

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

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

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

```c
#define setSn_IR(sn, ir) WIZCHIP_WRITE(Sn_IR(sn), (ir & 0x1F))
Set Sn_IR register. More...
```

```c
#define getSn_IR(sn) (WIZCHIP_READ(Sn_IR(sn)) & 0x1F)
Get Sn_IR register. More...
```

```c
#define setSn_IMR(sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), (imr & 0x1F))
Set Sn_IMR register. More...
```

```c
#define getSn_IMR(sn) (WIZCHIP_READ(Sn_IMR(sn)) & 0x1F)
Get Sn_IMR register. More...
```

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

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

```c
#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(sn), dhar, 6)
Set Sn_DHAR register. More...

#define getSn_DHAR(sn, dhar) WIZCHIP_READ_BUF(Sn_DHAR(sn), dhar, 6)
Get Sn_DHAR register. More...

#define setSn_DIPR(sn, dipr) WIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4)
Set Sn_DIPR register. More...

#define getSn_DIPR(sn, dipr) WIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 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(sn)) << 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(sn)) << 8) +
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)))
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...
```c
#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_SIZE(sn))
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(sn))
Get Sn_TXBUF_SIZE register. More...

#define getSn_TX_RD(sn) (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 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(sn)) << 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(sn)) << 8) +
                       WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn), 1)))
Get Sn_RX_RD register. More...
```
```c
#define getSn_RX_WR(sn) (((uint16_t) WIZCHIP_READ(Sn_RX_WR(sn)) << 8) +
                        WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn), 1))
Get Sn_RX_WR register. More...
```

```c
#define setSn_FRAG(sn, frag) WIZCHIP_WRITE(Sn_FRAG(sn), frag)
Set Sn_FRAG register. More...
```

```c
#define getSn_FRAG(sn) (((uint16_t) WIZCHIP_READ(Sn_FRAG(sn)) << 8) +
                       WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn), 1))
Get Sn_FRAG register. More...
```

```c
#define setSn_KPALVTR(sn, kpalvt) WIZCHIP_WRITE(Sn_KPALVTR(sn), kpalvt)
Set Sn_KPALVTR register. More...
```

```c
#define getSn_KPALVTR(sn) WIZCHIP_READ(Sn_KPALVTR(sn))
Get Sn_KPALVTR register. More...
```

```c
#define getSn_RxMAX(sn) (((uint16_t) getSn_RXBUF_SIZE(sn)) << 10)
Socket_register_access_function. More...
```

```c
#define getSn_TxMAX(sn) (((uint16_t) getSn_TXBUF_SIZE(sn)) << 10)
Socket_register_access_function. More...
```
Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
<th>More...</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td><strong>WIZCHIP_READ</strong> (uint32_t AddrSel)</td>
<td>It reads 1 byte value from a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE</strong> (uint32_t AddrSel, uint8_t wb)</td>
<td>It writes 1 byte value to a register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_READ_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It reads sequence data from registers.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>WIZCHIP_WRITE_BUF</strong> (uint32_t AddrSel, uint8_t *pBuf, uint16_t len)</td>
<td>It writes sequence data to registers.</td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td><strong>getSn_TX_FSR</strong> (uint8_t sn)</td>
<td>Get Sn_TX_FSR register.</td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td><strong>getSn_RX_RSR</strong> (uint8_t sn)</td>
<td>Get Sn_RX_RSR register.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_send_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to internal TX memory.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_recv_data</strong> (uint8_t sn, uint8_t *wizdata, uint16_t len)</td>
<td>It copies data to your buffer from internal RX memory.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td><strong>wiz_recv_ignore</strong> (uint8_t sn, uint16_t len)</td>
<td>It discard the received data in RX memory.</td>
<td></td>
</tr>
</tbody>
</table>
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>

Author
MidnightCow

Copyright
Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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.h.
#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.

```c
#define WIZCHIP_OFFSET_INC ( ADDR, N ) (ADDR + (N<<8))
```

Definition at line 68 of file w5500.h.

```c
#define IINCHIP_READ ( ADDR ) WIZCHIP_READ(ADDR)
```

The defined for legacy chip driver.

Definition at line 74 of file w5500.h.

```c
#define IINCHIP_WRITE ( ADDR, VAL ) WIZCHIP_WRITE(ADDR,VAL)
```

The defined for legacy chip driver.

Definition at line 75 of file w5500.h.

```c
#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 (xFFFFFFF 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.


**#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 708 of file `w5500.h`.

**#define MR_PPPOE 0x08**

Enable PPPoE.

0 : Disable PPPoE mode
1 : Enable PPPoE mode
If you use ADSL, this bit should be

Definition at line 716 of file `w5500.h`.

**#define MR_FARP 0x02**

Enable UDP_FORCE_ARP CHECK.

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 724 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()`.

```c
#define PHYCFGR_OPMDC_100F  (3<<3)
```

Definition at line 768 of file `w5500.h`. Referenced by `wizphy_getphyconf()`, and `wizphy_setphyconf()`.

```c
#define PHYCFGR_OPMDC_100H  (2<<3)
```

Definition at line 769 of file `w5500.h`. Referenced by `wizphy_getphyconf()`, and `wizphy_setphyconf()`.

```c
#define PHYCFGR_OPMDC_10F   (1<<3)
```

Definition at line 770 of file `w5500.h`. Referenced by `wizphy_getphyconf()`, and `wizphy_setphyconf()`.

```c
#define PHYCFGR_OPMDC_10H   (0<<3)
```

Definition at line 771 of file `w5500.h`. Referenced by `wizphy_setphyconf()`.

```c
#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.

```c
#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.

```c
#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.

```c
#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.

```c
#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, \texttt{Sn_DIPR} & \texttt{Sn_DPORT} should be respectively configured with the multicast group IP address & port number before Socket n is opened by OPEN command of \texttt{Sn_CR}.

Definition at line 817 of file \texttt{w5500.h}.

\begin{verbatim}
#define Sn_MR_BCASTB 0x40
\end{verbatim}

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 \texttt{w5500.h}.

\begin{verbatim}
#define Sn_MR_ND 0x20
\end{verbatim}

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 \texttt{RTR}.

Definition at line 836 of file \texttt{w5500.h}.

\begin{verbatim}
#define Sn_MR_UCASTB 0x10
\end{verbatim}

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 multicasting.

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 \texttt{Sn\_MR(P3:P0)}. The table below shows the value of \texttt{Sn\_SR} corresponding to \texttt{Sn\_MR}.

<table>
<thead>
<tr>
<th>\texttt{Sn_MR (P[3:0])}</th>
<th>\texttt{Sn_SR}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_MR_CLOSE (000)</td>
<td></td>
</tr>
<tr>
<td>Sn_MR_TCP (001)</td>
<td>SOCK_INIT (0x13)</td>
</tr>
<tr>
<td>Sn_MR_UDP (010)</td>
<td>SOCK_UDP (0x22)</td>
</tr>
<tr>
<td>S0_MR_MACRAW (100)</td>
<td>SOCK_MACRAW (0x02)</td>
</tr>
</tbody>
</table>

Definition at line 937 of file \texttt{w5500.h}.

\texttt{#define Sn\_CR\_LISTEN 0x02}

Wait connection request in TCP mode (Server mode)

This is valid only in TCP mode (\texttt{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 \texttt{Sn\_SR} changes the state from \texttt{SOCK\_INIT} to \texttt{SOCKET\_LISTEN}. When a TCP client connection request is successfully established, the \texttt{Sn\_SR} changes from \texttt{SOCKET\_LISTEN} to \texttt{SOCK\_ESTABLISHED} and the \texttt{Sn\_IR(0)} becomes. But when a TCP client connection request is failed, \texttt{Sn\_IR(3)} becomes and the status of \texttt{Sn\_SR} changes to \texttt{SOCK\_CLOSED}.

Definition at line 948 of file \texttt{w5500.h}.

\texttt{#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 \texttt{Sn\_DIPR} & \texttt{Sn\_DPORT} (destination address & port). If the connect-request is successful, the \texttt{Sn\_SR} is changed to \texttt{SOCK\_ESTABLISHED} and the \texttt{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**.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is valid only in TCP mode and operates when Socket n acts as <strong>TCP client</strong></td>
</tr>
</tbody>
</table>

Definition at line 960 of file w5500.h.

```c
#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 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**.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid only in TCP mode.</td>
</tr>
</tbody>
</table>

Definition at line 973 of file w5500.h.
```c
#define Sn_CR_CLOSE 0x10

Close socket.

Sn_SR is changed to SO_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.
```
```c
#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*.

```c
#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*.

```c
#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 `Sn_IR(TIMEOUT) = '1'`) is occurred.

Definition at line 1068 of file *w5500.h*.

```c
#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 transferred 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]) = 100) and 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.

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.

```c
#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.

```c
#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.
#define getSn_TxMAX (((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

Socket.c File
Reference

SOCKET APIs Implements file. More...

#include "socket.h"

Go to the source code of this file.
### Macros

<table>
<thead>
<tr>
<th>Define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SOCK_ANY_PORT_NUM</code></td>
<td><code>0xC000</code></td>
</tr>
<tr>
<td><code>CHECK_SOCKNUM()</code></td>
<td></td>
</tr>
<tr>
<td><code>CHECK_SOCKMODE(mode)</code></td>
<td></td>
</tr>
<tr>
<td><code>CHECK_SOCKINIT()</code></td>
<td></td>
</tr>
<tr>
<td><code>CHECK_SOCKDATA()</code></td>
<td></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int8_t socket</code></td>
<td>Open a socket.</td>
</tr>
<tr>
<td><code>int8_t close</code></td>
<td>Close a socket.</td>
</tr>
<tr>
<td><code>int8_t listen</code></td>
<td>Listen to a connection request from a client.</td>
</tr>
<tr>
<td><code>int8_t connect</code></td>
<td>Try to connect a server.</td>
</tr>
<tr>
<td><code>int8_t disconnect</code></td>
<td>Try to disconnect a connection socket.</td>
</tr>
<tr>
<td><code>int32_t send</code></td>
<td>Send data to the connected peer in TCP socket.</td>
</tr>
<tr>
<td><code>int32_t recv</code></td>
<td>Receive data from the connected peer.</td>
</tr>
<tr>
<td><code>int32_t sendto</code></td>
<td>Sends datagram to the peer with destination IP address and port number passed as parameter.</td>
</tr>
<tr>
<td><code>int32_t recvfrom</code></td>
<td>Receive datagram of UDP or MACRAW.</td>
</tr>
<tr>
<td><code>int8_t ctlsocket</code></td>
<td>Control socket.</td>
</tr>
<tr>
<td>int8_t</td>
<td><code>setsockopt</code> (uint8_t sn, <code>sockopt_type</code> sotype, void *arg)</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>int8_t</td>
<td><code>getsockopt</code> (uint8_t sn, <code>sockopt_type</code> sotype, void *arg)</td>
</tr>
</tbody>
</table>
Variables

```c
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 uint32 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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are
Redistributions of source code must retain the above 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:

```
if(sn > _WIZCHIP_SOCK_NUM_) return SOCKERR_SOCKNUM;
```

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:

```
if((getSn_MR(sn) & 0x0F) != mode) return SOCKERR_SOCKMODE;
```

Definition at line 88 of file socket.c.
Referenced by connect(), disconnect(), getsockopt(), listen(),
...
recv(), send(), and setsockopt().

#define CHECK_SOCKINIT ( )
Value:
```c
    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:
```c
    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**

```c
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

socket.h File Reference

SOCKET APIs Header file. More...

```c
#include "wizchip_conf.h"
```

Go to the source code of this file.
<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET</td>
<td>uint8_t</td>
<td>SOCKET type define for legacy driver. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCK_OK</td>
<td>1</td>
<td>Result is OK about socket process. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCK_BUSY</td>
<td>0</td>
<td>Socket is busy on processing the operation. Valid only Non-block IO Mode. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCK_FATAL</td>
<td>-1000</td>
<td>Result is fatal error about socket process. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCK_ERROR</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SOCKERR_SOCKNUM</td>
<td>(SOCK_ERROR - 1)</td>
<td>Invalid socket number. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCKERR_SOCKOPT</td>
<td>(SOCK_ERROR - 2)</td>
<td>Invalid socket option. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCKERR_SOCKINIT</td>
<td>(SOCK_ERROR - 3)</td>
<td>Socket is not initialized or SIPR is Zero IP address when Sn_MR_TCP. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCKERR_SOCKCLOSED</td>
<td>(SOCK_ERROR - 4)</td>
<td>Socket unexpectedly closed. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCKERR_SOCKMODE</td>
<td>(SOCK_ERROR - 5)</td>
<td>Invalid socket mode for socket operation. <a href="#">More...</a></td>
</tr>
<tr>
<td>SOCKERR_SOCKFLAG</td>
<td>(SOCK_ERROR - 6)</td>
<td>Invalid socket flag. <a href="#">More...</a></td>
</tr>
<tr>
<td>Predefined Error Constants</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_SOCKSTATUS</strong></td>
<td>Invalid socket status for socket operation. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_ARG</strong></td>
<td>Invalid argument. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_PORTZERO</strong></td>
<td>Port number is zero. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_IPINVALID</strong></td>
<td>Invalid IP address. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_TIMEOUT</strong></td>
<td>Timeout occurred. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_DATALEN</strong></td>
<td>Data length is zero or greater than buffer max size. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKERR_BUFFER</strong></td>
<td>Socket buffer is not enough for data communication. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SOCKFATAL_PACKLEN</strong></td>
<td>Invalid packet length. Fatal Error. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SF_ETHER_OWN</strong></td>
<td>In Sn_MR_MACRAW, Receive only the packet as broadcast, multicast and own packet. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SF_IGMP_VER2</strong></td>
<td>In Sn_MR_UDP with SF_MULTI_ENABLE, Select IGMP version 2. More...</td>
<td></td>
</tr>
<tr>
<td><strong>SF_TCP_NODELAY</strong></td>
<td>In Sn_MR_TCP, Use to nodelayed ack. More...</td>
<td></td>
</tr>
</tbody>
</table>
#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
#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

<table>
<thead>
<tr>
<th>sockint_kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIK_CONNECTED = (1 &lt;&lt; 0), SIK_DISCONNECTED = (1 &lt;&lt; 1), SIK_RECEIVED = (1 &lt;&lt; 2), SIK_TIMEOUT = (1 &lt;&lt; 3), SIK_SENT = (1 &lt;&lt; 4), SIK_ALL = 0x1F</td>
</tr>
<tr>
<td>The kind of Socket Interrupt. More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ctlsock_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_SET_IOMODE, CS_GET_IOMODE, CS_GET_MAXTXBUF, CS_GET_MAXRXBUF, CS_CLR_INTERRUPT, CS_GET_INTERRUPT, CS_SET_INTMASK, CS_GET_INTMASK</td>
</tr>
<tr>
<td>The type of ctlsocket(). More...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sockopt_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>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</td>
</tr>
<tr>
<td>The type of socket option in setsockopt() or getsockopt()</td>
</tr>
<tr>
<td>More...</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8_t socket</td>
<td>Open a socket.</td>
</tr>
<tr>
<td>int8_t close</td>
<td>Close a socket.</td>
</tr>
<tr>
<td>int8_t listen</td>
<td>Listen to a connection request from a client.</td>
</tr>
<tr>
<td>int8_t connect</td>
<td>Try to connect a server.</td>
</tr>
<tr>
<td>int8_t disconnect</td>
<td>Try to disconnect a connection socket.</td>
</tr>
<tr>
<td>int32_t send</td>
<td>Send data to the connected peer in TCP socket.</td>
</tr>
<tr>
<td>int32_t recv</td>
<td>Receive data from the connected peer.</td>
</tr>
<tr>
<td>int32_t sendto</td>
<td>Sends datagram to the peer with destination IP address and port number passed as parameter.</td>
</tr>
<tr>
<td>int32_t recvfrom</td>
<td>Receive datagram of UDP or MACRAW.</td>
</tr>
<tr>
<td>int8_t ctlsocket</td>
<td>Control socket.</td>
</tr>
</tbody>
</table>
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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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.h.
#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()`.

```c
#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()`.

```c
#define SOCKERR_SOCKFLAG    (SOCK_ERROR - 6)
```

Invalid socket flag.

Definition at line 102 of file `socket.h`.

Referenced by `socket()`.

```c
#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()`.

```c
#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()`.

```c
#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`.

```c
#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`.

```c
#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`.

```c
#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_FIFO.

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

wizchip_conf.c File
Reference

WIZCHIP Config Header File. More...

#include <stdio.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...](#)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void reg_wizchip_cris_cbfunc</code></td>
<td>Registers call back function for critical section of I/O functions such as</td>
</tr>
<tr>
<td></td>
<td>WIZCHIP_READ, WIZCHIP_WRITE, WIZCHIP_READ_BUF and WIZCHIP_WRITE_BUF. More...</td>
</tr>
<tr>
<td><code>void reg_wizchip_cs_cbfunc</code></td>
<td>Registers call back function for WIZCHIP select &amp; deselect. More...</td>
</tr>
<tr>
<td><code>void reg_wizchip_bus_cbfunc</code></td>
<td>Registers call back function for bus interface. More...</td>
</tr>
<tr>
<td><code>void reg_wizchip_spi_cbfunc</code></td>
<td>Registers call back function for SPI interface. More...</td>
</tr>
<tr>
<td><code>void reg_wizchip_spiburst_cbfunc</code></td>
<td>Registers call back function for SPI interface. More...</td>
</tr>
<tr>
<td><code>int8_t ctlwizchip</code></td>
<td>Controls to the WIZCHIP. More...</td>
</tr>
<tr>
<td><code>int8_t ctlnetwork</code></td>
<td>Controls to network. More...</td>
</tr>
<tr>
<td><code>void wizchip_sw_reset</code></td>
<td></td>
</tr>
</tbody>
</table>
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)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int8_t wizphy_setphypmode(uint8_t pmode)</code></td>
<td>Set the power mode of phy inside WIZCHIP. Refer to PHYCFGRI in W5500, PHYSTATUS in W5200.</td>
</tr>
<tr>
<td><code>void wizchip_setnetinfo(wiz_NetInfo *pnetinfo)</code></td>
<td>Set the network information for WIZCHIP.</td>
</tr>
<tr>
<td><code>void wizchip_getnetinfo(wiz_NetInfo *pnetinfo)</code></td>
<td>Get the network information for WIZCHIP.</td>
</tr>
<tr>
<td><code>int8_t wizchip_setnetmode(netmode_type netmode)</code></td>
<td>Set the network mode such as WOL, PPPoE, Ping Block, and etc.</td>
</tr>
<tr>
<td><code>netmode_type wizchip_getnetmode(void)</code></td>
<td>Get the network mode such as WOL, PPPoE, Ping Block, and etc.</td>
</tr>
<tr>
<td><code>void wizchip_settimeout(wiz_NetTimeout *nettime)</code></td>
<td>Set retry time value(RTR) and retry count(RCR).</td>
</tr>
<tr>
<td><code>void wizchip_gettimeout(wiz_NetTimeout *nettime)</code></td>
<td>Get retry time value(RTR) and retry count(RCR).</td>
</tr>
<tr>
<td>Variables</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>_WIZCHIP  WIZCHIP</td>
<td></td>
</tr>
</tbody>
</table>
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 recovering 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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

Redistribution and use in source and binary forms, with or without modi
permitted provided that the following conditions are met:

* Redistributions of source code must retain the above
notice, this list of conditions and the following disclaimer.

  • Redistributions in binary form must reproduce the above copyright
    conditions and the following disclaimer in the documentation and/o
    provided with the distribution.
  • Neither the name of the <ORGANIZATION> nor the names of its co
    used to endorse or promote products derived from this software wi
    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.
IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTOR
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.

**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 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()`.

```c
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()`.

```c
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()`.

```c
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_cbfnc()`. 

```c
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_cbfnc()`. 

```c
void reg_wizchip_cris_cbfnc ( 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()*.

```c
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 function 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_rb,
    void(*)(uint32_t addr, iodata_t wb) bus_wb )

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 function 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_BUS__
    __WIZCHIP::__IF::__write_data, __WIZCHIP::__IF::BUS, __WIZCHIP::IF__
    __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::_IF::_write_byte**, **__WIZCHIP::IF**, **__WIZCHIP::if_mode**, **__WIZCHIP::_IF::SPI**, **wizchip_spi_readbyte()**, and **wizchip_spi_writebyte()**.

```c
void reg_wizchip_spiburst_cbf FUNC（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::IF**, **__WIZCHIP::if_mode**, **__WIZCHIP::SPI**, **wizchip_spi_readburst()**, and **wizchip_spi_writeburst()**.

```c
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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### wizchip_conf.h File

Reference

WIZCHIP Config Header File. More...

```c
#include <stdint.h> #include "W5500/W5500.h"
```

Go to the source code of this file.
# Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>__WIZCHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The set of callback functions for W5500: <strong>WIZCHIP I/O functions</strong> W5200: <strong>WIZCHIP I/O functions</strong>. More...</td>
</tr>
<tr>
<td>struct</td>
<td>__WIZCHIP::__CRIS</td>
</tr>
<tr>
<td>struct</td>
<td>__WIZCHIP::__CS</td>
</tr>
<tr>
<td>union</td>
<td>__WIZCHIP::__IF</td>
</tr>
<tr>
<td>struct</td>
<td>wiz_PhyConf_t</td>
</tr>
<tr>
<td>struct</td>
<td>wiz_NetInfo_t</td>
</tr>
<tr>
<td>struct</td>
<td>wiz_NetTimeout_t</td>
</tr>
</tbody>
</table>
### Macros

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define _WIZCHIP_ 5500</code></td>
<td></td>
<td>Select WIZCHIP. More...</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_NONE_ 0x0000</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_BUS_ 0x0100</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_SPI_ 0x0200</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_BUS_DIR_ (_WIZCHIP_IO_MODE_BUS_ + 1)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_BUS_INDIR_ (_WIZCHIP_IO_MODE_BUS_ + 2)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_SPI_VDM_ (_WIZCHIP_IO_MODE_SPI_ + 1)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_SPI_FDM_ (_WIZCHIP_IO_MODE_SPI_ + 2)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_ID_ &quot;W5500\0&quot;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_SPI_VDM_</code></td>
<td></td>
<td>Define interface mode. More...</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_IO_BASE_ 0x00000000</code></td>
<td></td>
<td>Define I/O base address when BUS IF mode. More...</td>
</tr>
<tr>
<td><code>#define _WIZCHIP_SOCK_NUM_ 8</code></td>
<td></td>
<td>The count of independant socket of WIZCHIP. More...</td>
</tr>
<tr>
<td><code>#define PHY_CONFBY_HW 0</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Configured PHY operation mode by HW pin.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_CONFBY_SW 1</td>
<td>Configured PHY operation mode by SW register.</td>
</tr>
</tbody>
</table>

### Configured PHY operation mode by SW register.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_CONFBY_SW 1</td>
<td>Configured PHY operation mode by SW register.</td>
</tr>
</tbody>
</table>

### Configured PHY operation mode with user setting.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_CONFBY_SW 1</td>
<td>Configured PHY operation mode by SW register.</td>
</tr>
</tbody>
</table>

### Configured PHY operation mode with auto-negotiation.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_CONFBY_SW 1</td>
<td>Configured PHY operation mode by SW register.</td>
</tr>
</tbody>
</table>

### Link Speed 10.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_SPEED_10 0</td>
<td>Link Speed 10.</td>
</tr>
</tbody>
</table>

### Link Speed 100.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_SPEED_100 1</td>
<td>Link Speed 100.</td>
</tr>
</tbody>
</table>

### Link Half-Duplex.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_DUPLEX_HALF 0</td>
<td>Link Half-Duplex.</td>
</tr>
</tbody>
</table>

### Link Full-Duplex.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_DUPLEX_FULL 1</td>
<td>Link Full-Duplex.</td>
</tr>
</tbody>
</table>

### Link Off.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_LINK_OFF 0</td>
<td>Link Off.</td>
</tr>
</tbody>
</table>

### Link On.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_LINK_ON 1</td>
<td>Link On.</td>
</tr>
</tbody>
</table>

### PHY power normal mode.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_POWER_NORM 0</td>
<td>PHY power normal mode.</td>
</tr>
</tbody>
</table>

### PHY power down mode.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHY_POWER_DOWN 1</td>
<td>PHY power down mode.</td>
</tr>
</tbody>
</table>
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

#### ctlwizchip_type
```c
enum ctlwizchip_type {
    CW_RESET_WIZCHIP, CW_INIT_WIZCHIP,
    CW_GET_INTERRUPT, CW_CLR_INTERRUPT,
    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
}
```

#### cntlnetwork_type
```c
enum cntlnetwork_type {
    CN_SET_NETINFO, CN_GET_NETINFO,
    CN_SET_NETMODE, CN_GET_NETMODE,
    CN_SET_TIMEOUT, CN_GET_TIMEOUT
}
```

#### intr_kind
```c
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)
}
```

#### dhcp_mode
```c
enum dhcp_mode { NETINFO_STATIC = 1, NETINFO_DHCP }
```

#### netmode_type
```c
enum netmode_type { NM_FORCEARP = (1<<1),
    NM_WAKEONLAN = (1<<5), NM_PINGBLOCK = (1<<4),
    NM_PPPOE = (1<<3) }
```
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reg_wizchip_cris_cbfunc</code></td>
<td>Registers call back function for critical section of I/O functions such as <code>WIZCHIP_READ</code>, <code>WIZCHIP_WRITE</code>, <code>WIZCHIP_READ_BUF</code> and <code>WIZCHIP_WRITE_BUF</code>. More...</td>
</tr>
<tr>
<td><code>reg_wizchip_cs_cbfunc</code></td>
<td>Registers call back function for WIZCHIP select &amp; deselect. More...</td>
</tr>
<tr>
<td><code>reg_wizchip_bus_cbfunc</code></td>
<td>Registers call back function for bus interface. More...</td>
</tr>
<tr>
<td><code>reg_wizchip_spi_cbfunc</code></td>
<td>Registers call back function for SPI interface. More...</td>
</tr>
<tr>
<td><code>reg_wizchip_spiburst_cbfunc</code></td>
<td>Registers call back function for SPI interface. More...</td>
</tr>
<tr>
<td><code>ctlwizchip (ctlwizchip_type cwtype, void *arg)</code></td>
<td>Controls to the WIZCHIP. More...</td>
</tr>
<tr>
<td><code>ctlnetwork (ctlnetwork_type cntype, void *arg)</code></td>
<td>Controls to network. More...</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void  <strong>wizchip_sw_reset</strong>(void)</td>
<td>Reset WIZCHIP by softly.</td>
</tr>
<tr>
<td>int8_t <strong>wizchip_init</strong>(uint8_t *txsize, uint8_t *rxsize)</td>
<td>Initializes WIZCHIP with socket buffer size.</td>
</tr>
<tr>
<td>void  <strong>wizchip_clrinterrupt</strong>(intr_kind intr)</td>
<td>Clear Interrupt of WIZCHIP.</td>
</tr>
<tr>
<td>intr_kind <strong>wizchip_getinterrupt</strong>(void)</td>
<td>Get Interrupt of WIZCHIP.</td>
</tr>
<tr>
<td>void  <strong>wizchip_setinterruptmask</strong>(intr_kind intr)</td>
<td>Mask or Unmask Interrupt of WIZCHIP.</td>
</tr>
<tr>
<td>intr_kind <strong>wizchip_getinterruptmask</strong>(void)</td>
<td>Get Interrupt mask of WIZCHIP.</td>
</tr>
<tr>
<td>int8_t <strong>wizphy_getphylink</strong>(void)</td>
<td>get the link status of phy in WIZCHIP. No use in W5100</td>
</tr>
<tr>
<td>int8_t <strong>wizphy_getphypmode</strong>(void)</td>
<td>get the power mode of PHY in WIZCHIP. No use in W5100</td>
</tr>
<tr>
<td>void  <strong>wizphy_reset</strong>(void)</td>
<td>Reset phy. Vailid only in W5500.</td>
</tr>
<tr>
<td>void  <strong>wizphy_setphyconf</strong>(wiz_PhyConf *phyconf)</td>
<td>Set the phy information for WIZCHIP without power mode.</td>
</tr>
<tr>
<td>void  <strong>wizphy_getphyconf</strong>(wiz_PhyConf *phyconf)</td>
<td>Get phy configuration information.</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void wizphy_getphystat (wiz_PhyConf *phyconf)</code></td>
<td>Get phy status. More...</td>
</tr>
<tr>
<td><code>int8_t wizphy_setphypmode (uint8_t pmode)</code></td>
<td>Set the power mode of phy inside WIZCHIP. Refer to PHYCFGR in W5500, PHYSTATUS in W5200 More...</td>
</tr>
<tr>
<td><code>void wizchip_setnetinfo (wiz_NetInfo *pnetinfo)</code></td>
<td>Set the network information for WIZCHIP. More...</td>
</tr>
<tr>
<td><code>void wizchip_getnetinfo (wiz_NetInfo *pnetinfo)</code></td>
<td>Get the network information for WIZCHIP. More...</td>
</tr>
<tr>
<td><code>int8_t wizchip_setnetmode (netmode_type netmode)</code></td>
<td>Set the network mode such WOL, PPPoE, Ping Block, and etc. More...</td>
</tr>
<tr>
<td><code>netmode_type wizchip_getnetmode (void)</code></td>
<td>Get the network mode such WOL, PPPoE, Ping Block, and etc. More...</td>
</tr>
<tr>
<td><code>void wizchip_settimeout (wiz_NetTimeout *nettime)</code></td>
<td>Set retry time value(RTR) and retry count(RCR). More...</td>
</tr>
<tr>
<td><code>void wizchip_gettimeout (wiz_NetTimeout *nettime)</code></td>
<td>Get retry time value(RTR) and retry count(RCR). More...</td>
</tr>
</tbody>
</table>
## 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

Copyright (c) 2013, WIZnet Co., LTD. All rights reserved.

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 above 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 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_
        o _WIZCHIP_IO_MODE_BUS_DIR_
        o _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_  0x00000000
```
The count of independent 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 \texttt{wizchip\_conf.h}.

Referenced by \texttt{wizphy\_getphypmode()}, and \texttt{wizphy\_setphypmode()}. 
typedef uint8_t iodata_t

Definition at line 131 of file wizchip_conf.h.
Function Documentation

```c
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()`.

```c
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_deselect**: callback function 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()**.

```c
void reg_wizchip_bus_cbfuc ( iodata_t(*)(uint32_t addr) bus_rb,
                              void(*)(uint32_t addr, iodata_t wb) bus_wb )
```

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** function 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_BUS**, **__WIZCHIP::_IF::_write_data**, **__WIZCHIP::IF::BUS**, **__WIZCHIP::IF::BUS**, **__WIZCHIP::if_mode**, **wizchip_bus_readdata()**, and
wizchip_bus_writedata().

```c
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::IF::write_byte`, `__WIZCHIP::IF`, `__WIZCHIP::if_mode`, `__WIZCHIP::IF::SPI`, `wizchip_spi_readbyte()`, and `wizchip_spi_writebyte()`.

```c
void reg_wizchip_spiburst_cbfunc ( void(*)(uint8_t *pBuf, uint16_t len) spi_rb,
                                      void(*)(uint8_t *pBuf, uint16_t len) spi_wb)
```

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` 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_mode`, `__WIZCHIP::_IF::SPI`, `wizchip_spi_readburst()`, and `wizchip_spi_writeburst()`.

```c
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`, `PHYCFGFR_LNK_ON`, and `PHYSTATUS_LINK`.

Referenced by `ctlwizchip()`.

```c
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`, `PHYCFGFR_OPMDC_PDOWN`, and `PHYSTATUS_POWERDOWN`.

Referenced by `ctlwizchip()`.

```c
void wizphy_reset ( void )
```
Reset phy. Valid only in W5500.

Definition at line 617 of file wizchip_conf.c.

References getPHYCFG, PHYCFG_RST, and setPHYCFG.

Referenced by ctlwizchip(), wizphy_setphyconf(), and wizphy_setphympmode().
Variable Documentation

_WIZCHIP WIZCHIP

\ref _WIZCHIP instance

Definition at line 165 of file wizchip_conf.c.
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
- ctslock_type : socket.h
- ctssocket() : 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 INTERRUPTION : 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
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](#)

---

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](#) 1.8.9.1
### 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](#)
  - getPHSTATUS : [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
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_BUF_SIZE : w5300.h
getSn_RXBUF_SIZE : w5100.h, w5200.h, w5300.h, w5500.h
getSn_RX_FIFOR : w5300.h
getSn_RX_RS : w5500.h, w5100.h, w5200.h
getSn_RX_RS() : 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_RxDBASE : 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_TXBUFF_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
• getUPOTR : 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: `w5100.h`, `w5200.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宋OCK: `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:

- I -

  - 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
- MTYPEPER : 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
- PHYCFG_OPMDC_100F : w5500.h
- PHYCFG_OPMDC_100FA : w5500.h
- PHYCFG_OPMDC_100H : w5500.h
- PHYCFG_OPMDC_10F : w5500.h
- PHYCFG_OPMDC_10H : w5500.h
- PHYCFG_OPMDC_ALLA : w5500.h
- PHYCFG_OPMDC_NA : w5500.h
- PHYCFG_OPMDC_PDOWN : w5500.h
- PHYCFG_RST : w5500.h
- PHYCFG_SPD_10 : w5500.h
- PHYCFG_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_BDPTH : 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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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`
- `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
setSnPROTO : 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_RXBBUF : w5100.h , w5200.h , w5300.h , w5500.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_ISC_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, w5500.h
- Sn_CR_RECV : w5100.h, w5200.h, w5300.h, w5500.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_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 : 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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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](#)

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.stackoverflow.com) 1.8.9.1
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

---

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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.c , 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
  - getssockopt() : 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_getnetmask() : 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`

---

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](http://www.doxygen.org) 1.8.9.1
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`

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](https://doxygen.nl) 1.8.9.1
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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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_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
- c -
  - 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_SOCKET_0 : wizchip_conf.h
  - IK_SOCKET_1 : wizchip_conf.h
  - IK_SOCKET_2 : wizchip_conf.h
  - IK_SOCKET_3 : wizchip_conf.h
  - IK_SOCKET_4 : wizchip_conf.h
  - IK_SOCKET_5 : wizchip_conf.h
  - IK_SOCKET_6 : wizchip_conf.h
  - IK_SOCKET_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

<table>
<thead>
<tr>
<th>Macros</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>IMR</em> : w5100.h, w5200.h, w5500.h, w5300.h</td>
</tr>
<tr>
<td><em>RCR</em> : w5500.h, w5100.h, w5200.h, w5300.h</td>
</tr>
<tr>
<td><em>RTR</em> : w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td><em>W5100_IO_BASE</em> : w5100.h</td>
</tr>
<tr>
<td><em>W5200_SPI_READ</em> : w5200.h</td>
</tr>
<tr>
<td><em>W5200_SPI_WRITE</em> : w5200.h</td>
</tr>
<tr>
<td><em>W5300_IO_BASE</em> : w5300.h</td>
</tr>
<tr>
<td><em>W5500_IO_BASE</em> : w5500.h</td>
</tr>
<tr>
<td><em>W5500_SPI_FDM_OP_LEN1</em> : w5500.c</td>
</tr>
<tr>
<td><em>W5500_SPI_FDM_OP_LEN2</em> : w5500.c</td>
</tr>
<tr>
<td><em>W5500_SPI_FDM_OP_LEN4</em> : w5500.c</td>
</tr>
<tr>
<td><em>W5500_SPI_READ</em> : w5500.h</td>
</tr>
<tr>
<td><em>W5500_SPI_VDM_OP</em> : w5500.c</td>
</tr>
<tr>
<td><em>W5500_SPI_WRITE</em> : w5500.h</td>
</tr>
<tr>
<td><em>WIZCHIP</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_ID</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_BASE</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_BUS</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_BUS_DIR</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_BUS_INDIR</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_NONE</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_SPI</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_SPI_FDM</em> : wizchip_conf.h</td>
</tr>
<tr>
<td><em>WIZCHIP_IO_MODE_SPI_VDM</em> : wizchip_conf.h</td>
</tr>
</tbody>
</table>
- _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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
Socket APIs

- **FMTUR**: `w5300.h`

---

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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

Generated on Wed May 4 2016 16:44:02 for Socket APIs by doxygen 1.8.9.1
- 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*
- INLEVEL : *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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>Functions</td>
<td>Variables</td>
<td>Typedefs</td>
<td>Enumerations</td>
</tr>
</tbody>
</table>

### Macros

- **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_FAR : w5500.h
- MR_F : 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

---

Generated on Wed May 4 2016 16:44:02 for Socket APIs by [Doxygen](http://www.doxygen.org) 1.8.9.1
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
- PHYCFGFR_OPMDC_100F : w5500.h
- PHYCFGFR_OPMDC_100FA : w5500.h
- PHYCFGFR_OPMDC_100H : w5500.h
- PHYCFGFR_OPMDC_10F : w5500.h
- PHYCFGFR_OPMDC_10H : w5500.h
- PHYCFGFR_OPMDC_ALLA : w5500.h
- PHYCFGFR_OPMDC_NA : w5500.h
- PHYCFGFR_OPMDC_PDOWN : w5500.h
- PHYCFGFR_RST : w5500.h
- PHYCFGFR_SPD_10 : w5500.h
- PHYCFGFR_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
<table>
<thead>
<tr>
<th>Symbol</th>
<th>File Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>setSIPR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSIR</td>
<td>w5200.h, w5500.h</td>
</tr>
<tr>
<td>setSn_CR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_DHAR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_DIPIR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_DIPR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_DPORT</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_DPORTR</td>
<td>w5500.h</td>
</tr>
<tr>
<td>setSn_FRAG</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_FRAGR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_IMR</td>
<td>w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_IR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_KPALVTR</td>
<td>w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_MR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_MSSR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_PORT</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_PORTR</td>
<td>w5500.h</td>
</tr>
<tr>
<td>setSn_PROTO</td>
<td>w5100.h, w5200.h, w5300.h</td>
</tr>
<tr>
<td>setSn_PROTOR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_RX_RD</td>
<td>w5100.h, w5200.h, w5500.h</td>
</tr>
<tr>
<td>setSn_RX_WR</td>
<td>w5100.h, w5200.h</td>
</tr>
<tr>
<td>setSn_RXBUF_SIZE</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_RXMEM_SIZE</td>
<td>w5100.h, w5200.h</td>
</tr>
<tr>
<td>setSn_TOS</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_TOSR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_TTL</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_TTLR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_TX_FIFOR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_TX_WR</td>
<td>w5100.h, w5200.h, w5500.h</td>
</tr>
<tr>
<td>setSn_TX_WRSR</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setSn_TXBUF_SIZE</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setSn_TXMEM_SIZE</td>
<td>w5100.h, w5200.h</td>
</tr>
<tr>
<td>setSUBR</td>
<td>w5100.h, w5200.h, w5300.h, w5500.h</td>
</tr>
<tr>
<td>setTMS01R</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setTMS23R</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setTMS45R</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setTMS67R</td>
<td>w5300.h</td>
</tr>
<tr>
<td>setTMSR</td>
<td>w5100.h</td>
</tr>
<tr>
<td>SF_BROAD_BLOCK</td>
<td>socket.h</td>
</tr>
<tr>
<td>SF_ETHER_OWN</td>
<td>socket.h</td>
</tr>
<tr>
<td>SF_IGMP_VER2</td>
<td>socket.h</td>
</tr>
</tbody>
</table>
- 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_PDJ : 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_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_FGRA : 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
- Sn_IR_RECV : w5100.h , w5200.h , w5300.h , w5500.h
- Sn_IR_SENDOK : w5100.h , w5200.h , w5300.h , w5500.h
• Sn_IR_TIMEOUT : w5100.h , w5200.h , w5300.h , w5500.h
• Sn_KPÄLVTR : 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_MACRAW : 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_PORTR : w5300.h
• Sn_PROTO : w5100.h , w5200.h
• Sn_PROTOR : w5500.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_PPOE : 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

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>File Members</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>All</th>
<th>Functions</th>
<th>Variables</th>
<th>Typedefs</th>
<th>Enumerations</th>
<th>Enumerator</th>
</tr>
</thead>
</table>

### Macros

- c
- f
- g
- i
- m
- p
- r
- s
- t
- u
- v
- w

- 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

---

Generated on Wed May 4 2016 16:44:02 for Socket APIs by doxygen 1.8.9.1
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

Generated on Wed May 4 2016 16:44:02 for Socket APIs by doxygen 1.8.9.1
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

Generated on Wed May 4 2016 16:44:02 for Socket APIs by doxygen 1.8.9.1
Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
</table>

### Related Pages

Here is a list of all related documentation pages:

<table>
<thead>
<tr>
<th>Todo List</th>
</tr>
</thead>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by [doxygen](https://www.doxygen.org) 1.8.9.1
Go to the documentation of this file.

```c
#include "socket.h"

//M20150401 : Typing Error
#define SOCK_ANY_PORT_NUM 0xC000;

static uint16_t sock_any_port = SOCK_ANY_PORT_NUM;
static uint16_t sock_io_mode = 0;
static uint16_t sock_is_sending = 0;

static uint16_t sock_remained_size[_WIZCHIP_SOCK_NUM_] = {0,0,};

//M20150601 : For extern decleation
static uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_] = {0,};
uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_] = {0,};
```
```c
#define CHECK_SOCKNUM() \
    do{
        if(sn > _WIZCHIP_SOCK_NUM_) return \
          SOCKERR_SOCKNUM; \n    }while(0); \n
#define CHECK_SOCKMODE(mode) \n    do{
        if((getSn_MR(sn) & 0x0F) != mode) \
          return SOCKERR_SOCKMODE; \n    }while(0); \n
#define CHECK_SOCKINIT() \n    do{
        if((getSn_SR(sn) != SOCK_INIT)) return \
          SOCKERR_SOCKINIT; \n    }while(0); \n
#define CHECK_SOCKDATA() \n    do{
        if(len == 0) return SOCKERR_DATALEN; \n```
int8_t socket(uint8_t sn, uint8_t protocol, uint16_t port, uint8_t flag) {
    CHECK_SOCKNUM();
    switch(protocol) {
    case Sn_MR_TCP:
        { //M20150601: Fixed the warning - taddr will never be NULL
            /* uint8_t taddr[4];
            getSIPR(taddr);
            */
            uint32_t taddr;
            getSIPR((uint8_t*)&taddr);
            if(taddr == 0) return SOCKERR_SOCKINIT;
        }
        
        case Sn_MR_UDP:
        case Sn_MR_MACRAW:
        break;
        
        #if (_WIZCHIP_ < 5200)
        case Sn_MR_IPRAW:
        case Sn_MR_PPPoE:
        break;
        #endif
    
        default:
        return SOCKERR_SOCKMODE;
    }
    
    //M20150601: For SF_TCP_ALIGN & W5300
    //if((flag & 0x06) != 0) return SOCKERR_SOCKFLAG;
if((flag & 0x04) != 0) return SOCKERR_SOCKFLAG;

#if _WIZCHIP_ == 5200
  if(flag & 0x10) return SOCKERR_SOCKFLAG;
#endif

if(flag != 0)
{
  switch(protocol)
  {
  case Sn_MR_TCP:
      //M20150601 : For SF_TCP_ALIGN & W5300
      #if _WIZCHIP_ == 5300
        if((flag & (SF_TCP_NODELAY|SF_IO_NONBLOCK|SF_TCP_ALIGN))!=0) return SOCKERR_SOCKFLAG;
      #else
        if((flag & (SF_TCP_NODELAY|SF_IO_NONBLOCK))==0) return SOCKERR_SOCKFLAG;
      #endif
    #endif
    break;
  case Sn_MR_UDP:
    if(flag & SF_IGMP_VER2)
    {
      if((flag & SF_MULTI_ENABLE)==0) return SOCKERR_SOCKFLAG;
    }
    #if _WIZCHIP_ == 5500
    if(flag & SF_UNI_BLOCK)
    {
      if((flag & SF_MULTI_ENABLE)==0) return SOCKERR_SOCKFLAG;
    }
    #endif
break;
default:
break;
} 
close(sn);

//M20150601
#if _WIZCHIP_ == 5300
setSn_MR(sn, ((uint16_t)(protocol | (flag & 0xF0))) | (((uint16_t)(flag & 0x02)) << 7));
#else
setSn_MR(sn, (protocol | (flag & 0xF0)));
#endif
if(!port) {
    port = sock_any_port++; 
    if(sock_any_port == 0xFFF0) sock_any_port = SOCK_ANY_PORT_NUM;
}
setSn_PORT(sn, port);
setSn_CR(sn, Sn_CR_OPEN);
while(getSn_CR(sn));

//A20150401 : For release the previous sock_io_mode
sock_io_mode &= ~(1 << sn);
//
sock_io_mode |= ((flag & SF_IO_NONBLOCK) << sn);
sock_is_sending &= ~(1<<sn);
sock_Remained_size[sn] = 0;
//M20150601 : repalce 0 with PACK_COMPLETED
//sock_pack_info[sn] = 0;
sock_pack_info[sn] = PACK_COMPLETED;
//
while(getSn_SR(sn) == SOCK_CLOSED);
return (int8_t)sn;
}

int8_t close(uint8_t sn)
{
    CHECK_SOCKNUM();

    // Applied the erratum 1 of W5300
    #if (_WIZCHIP_ == 5300)
    // Wrong socket parameter. sn -
    //if((getSn_MR(s)&0x0F) == Sn_MR_TCP)
    if( ((getSn_MR(sn)&0x0F) == Sn_MR_TCP)
        && (getSn_TX_FSR(sn) != getSn_TxMAX(sn)) )
    {
        uint8 destip[4] = {0, 0, 0, 1};
        // You can wait for completing to
        // sending data;
        // wait about 1 second;
        // if you have completed to send data,
        // skip the code of erratum 1
        // ex> wait_1s();
        // if (getSn_TX_FSR(s) ==
        getSn_TxMAX(s)) continue;
        //
        // The socket() of close()
    // calls close() itself again. It occurs a
    // infinite loop - close()-socket()-close()-
    //socket()~
    //socket(s,Sn_MR_UDP,0x3000,0);
    //sendto(s,destip,1,destip,0x3000); //
    // send the dummy data to an unknown
    // destination(0.0.0.1).
    setSn_MR(sn,Sn_MR_UDP);
    setSn_PORTR(sn, 0x3000);
setSn_CR(sn, Sn_CR_OPEN);
while(getSn_CR(sn) != 0);
while(getSn_SR(sn) != SOCK_UDP);
sendto(sn, destip, 1, destip, 0x3000); // send the dummy data to an unknown destination(0.0.0.1).
}
#endif
setSn_CR(sn, Sn_CR_CLOSE);
/* wait to process the command... */
while( getSn_CR(sn) );
/* clear all interrupt of the socket. */
setSn_IR(sn, 0xFF);
//A20150401 : Release the sock_io_mode of socket n.
sock_io_mode &= ~(1<<sn);
//
sock_is_sending &= ~(1<<sn);
sock_remained_size[sn] = 0;
sock_pack_info[sn] = 0;
while(getSn_SR(sn) != SOCK_CLOSED);
return SOCK_OK;
}
int8_t listen(uint8_t sn)
{
    CHECK_SOCKNUM();
    CHECK_SOCKMODE(Sn_MR_TCP);
    CHECK_SOCKINIT();
    setSn_CR(sn, Sn_CR_LISTEN);
    while(getSn_CR(sn));
    while(getSn_SR(sn) != SOCK_LISTEN)
    {
        if(getSn_CR(sn) == SOCK_CLOSED)
        {
            close(sn);
            return SOCKERR_SOCKCLOSED;
int8_t connect(uint8_t sn, uint8_t * addr, uint16_t port)
{
    CHECK_SOCKNUM();
    CHECK_SOCKMODE(Sn_MR_TCP);
    CHECK_SOCKINIT();
    //M20140501 : For avoiding fatal error on memory align mismatched
    //if( *((uint32_t*)addr) == 0xFFFFFFFF || *((uint32_t*)addr) == 0) return SOCKERR_IPINVALID;
    {
        uint32_t taddr;
        taddr = ((uint32_t)addr[0] & 0x000000FF);
        taddr = (taddr << 8) + ((uint32_t)addr[1] & 0x000000FF);
        taddr = (taddr << 8) + ((uint32_t)addr[2] & 0x000000FF);
        taddr = (taddr << 8) + ((uint32_t)addr[3] & 0x000000FF);
        if( taddr == 0xFFFFFFFF || taddr == 0) return SOCKERR_IPINVALID;
    }
    //
    if(port == 0) return SOCKERR_PORTZERO;
    setSn_DIPR(sn,addr);
    setSn_DPORT(sn,port);
    setSn_CR(sn,Sn_CR_CONNECT);
    while(getSn_CR(sn));
if (sock_io_mode & (1<<sn)) return SOCK_BUSY;

while (getSn_SR(sn) != SOCK_ESTABLISHED)
{
    if (getSn_IR(sn) & Sn_IR_TIMEOUT)
    {
        setSn_IR(sn, Sn_IR_TIMEOUT);
        return SOCKERR_TIMEOUT;
    }

    if (getSn_SR(sn) == SOCK_CLOSED)
    {
        return SOCKERR_SOCKCLOSED;
    }
}

return SOCK_OK;

int8_t disconnect(uint8_t sn)
{
    CHECK_SOCKNUM();
    CHECK_SOCKMODE(Sn_MR_TCP);
    setSn_CR(sn,Sn_CR_DISCON);
    /* wait to process the command... */
    while (getSn_CR(sn));
    sock_is_sending &= ~(1<<sn);
    if (sock_io_mode & (1<<sn)) return SOCK_BUSY;

    while (getSn_SR(sn) != SOCK_CLOSED)
    {
        if (getSn_IR(sn) & Sn_IR_TIMEOUT)
        {
            close(sn);
            return SOCKERR_TIMEOUT;
        }
    }
}
int32_t send(uint8_t sn, uint8_t * buf, uint16_t len) {
    uint8_t tmp = 0;
    uint16_t freesize = 0;

    CHECK_SOCKNUM();
    CHECK_SOCKMODE(Sn_MR_TCP);
    CHECK_SOCKDATA();
    tmp = getSn_SR(sn);
    if (tmp != SOCK_ESTABLISHED && tmp != SOCK_CLOSE_WAIT)
        return SOCKERR_SOCKSTATUS;

    if (sock_is_sending & (1<<sn)) {
        tmp = getSn_IR(sn);
        if (tmp & Sn_IR_SENDOK) {
            setSn_IR(sn, Sn_IR_SENDOK);
            //M20150401 : Typing Error
            //if _WZICHIP_ == 5200
            #if _WZICHIP_ == 5200
                if (getSn_TX_RD(sn) != sock_next_rd[sn])
                    sock_next_rd[sn] = buf;
            #endif
            setSn_CR(sn, Sn_CR_SEND);
            while (getSn_CR(sn));
            return SOCK_BUSY;
        }
        else if (tmp & Sn_IR_TIMEOUT)
            close(sn);
    }
}
return SOCKERR_TIMEOUT;
}
else return SOCK_BUSY;
}
freesize = getSn_TxMAX(sn);
if (len > freesize) len = freesize; // check size not to exceed MAX size.
while(1)
{
    freesize = getSn_TX_FSR(sn);
    tmp = getSn_SR(sn);
    if ((tmp != SOCK_ESTABLISHED) && (tmp != SOCK_CLOSE_WAIT))
    {
        close(sn);
        return SOCKERR_SOCKSTATUS;
    }
    if((sock_io_mode & (1<<sn)) && (len > freesize)) return SOCK_BUSY;
    if(len <= freesize) break;
}
wiz_send_data(sn, buf, len);
#if _WIZCHIP_ == 5200
    sock_next_rd[sn] = getSn_TX_RD(sn) + len;
#else
#endif
#if _WIZCHIP_ == 5300
    setSn_TX_WRSR(sn,len);
#else
#endif
setSn_CR(sn,Sn_CR_SEND);
/* wait to process the command... */
while(getSn_CR(sn));
sock_is_sending |= (1 << sn);
//M20150409 : Explicit Type Casting
//return len;
return (int32_t)len;
}

int32_t recv(uint8_t sn, uint8_t * buf, uint16_t len)
{
  uint8_t tmp = 0;
  uint16_t recvsize = 0;
  //A20150601 : For integrating with W5300
  #if _WIZCHIP_ == 5300
    uint8_t head[2];
    uint16_t mr;
  #endif
  //
  CHECK_SOCKNUM();
  CHECK_SOCKMODE(Sn_MR_TCP);
  CHECK_SOCKDATA();
  recvsize = getSn_RxMAX(sn);
  if(rerecvsize < len) len = recvsize;
  //A20150601 : For Integrating with W5300
  #if _WIZCHIP_ == 5300
    //sock_pack_info[sn] = PACK_COMPLETED;
    // for clear
    if(sock_remained_size[sn] == 0)
    {
      #endif
      //
      while(1)
      {
        recvsize = getSn_RX_RSR(sn);
        tmp = getSn_SR(sn);
        if (tmp != SOCK_ESTABLISHED)
        {
          if(tmp == SOCK_CLOSE_WAIT)
418    if (recvsize != 0) break;
419    else if (getSn_TX_FSR(sn) ==
420              getSn_TxMAX(sn))
421    {
422        close(sn);
423        return SOCKERR_SOCKSTATUS;
424    }
425    else
426    {
427        close(sn);
428        return SOCKERR_SOCKSTATUS;
429    }
430
431    if( (sock_io_mode & (1<<sn)) &&
432        (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) ||
442        (getSn_MR(sn) & Sn_MR_ALIGN))
443    {
444        mr = getMR();
445        if(( (getSn_MR(sn) & Sn_MR_ALIGN)==0)
446           { 
447                wiz_recv_data(sn,head,2);
448                if(mr & MR_FS)
449                    recvsize = (((uint16_t)head[1])
450                        << 8) | (((uint16_t)head[0]));
451                else
452                    recvsize = (((uint16_t)head[0])
453                        << 8) | (((uint16_t)head[1]));
454            }
455        }
456    }
457
<< 8) | ((uint16_t)head[1]);

    sock_pack_info[sn] = PACK_FIRST;
}
    sock_remained_size[sn] = recvsize;
}

    } if(len > sock_remained_size[sn]) len =
    sock_remained_size[sn];
    recvsize = len;
    if(sock_pack_info[sn] & PACK_FIFOBYTE)
    {
        *buf = sock_remained_byte[sn];
        buf++;
        sock_pack_info[sn] &= ~(PACK_FIFOBYTE);
        recvsize -= 1;
        sock_remained_size[sn] -= 1;
    }
    if(recvsize != 0)
    {
        wiz_recv_data(sn, buf, recvsize);
        setSn_CR(sn, Sn_CR_RECV);
        while(getSn_CR(sn));
    }
    sock_remained_size[sn] -= recvsize;
    if(sock_remained_size[sn] != 0)
    {
        sock_pack_info[sn] |= PACK_REMAINED;
        if(recvsize & 0x1) sock_pack_info[sn]
            |= PACK_FIFOBYTE;
    }
} else sock_pack_info[sn] = PACK_COMPLETED;
    if(getSn_MR(sn) & Sn_MR_ALIGN)
        sock_remained_size[sn] = 0;
    //len = recvsize;
#else
    if(recvsize < len) len = recvsize;
    wiz_recv_data(sn, buf, len);
```c
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  setSn_DIPR(sn,addr);
524  setSn_DPORT(sn,port);
525  freesize = getSn_TxMAX(sn);
526  if (len > freesize) len = freesize; //check size not to exceed MAX size.
527  while(1)
528  {
529    if(len <= freesize) break;
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);
setSUBR((uint8_t*)"\x00\x00\x00\x00");

else  taddr = 0;

#endif

//A20150601 : For W5300
#if _WIZCHIP_ == 5300
    setSn_TX_WRSR(sn, len);
#endif

// setSn_CR(sn,Sn_CR_SEND);
/* wait to process the command... */
while(getSn_CR(sn));
while(1)
{
    tmp = getSn_IR(sn);
    if(tmp & Sn_IR_SENDOK)
    {
        setSn_IR(sn, Sn_IR_SENDOK);
        break;
    }
    //M:20131104
    //else if(tmp & Sn_IR_TIMEOUT) return SOCKERR_TIMEOUT;
    else if(tmp & Sn_IR_TIMEOUT)
    {
        setSn_IR(sn, Sn_IR_TIMEOUT);
        //M20150409 : Fixed the lost of sign bits by type casting.
        //len = (uint16_t)SOCKERR_TIMEOUT;
        //break;
#if _WIZCHIP_ < 5500  //M20150401 : for WIZCHIP Errata #4, #5 (ARP errata)
    if(taddr)
        setSUBR((uint8_t*)&taddr);
#endif
return SOCKERR_TIMEOUT;
}

#if _WIZCHIP_ < 5500  //M20150401 : for WIZCHIP Errata #4, #5 (ARP errata)
    if(taddr) setSUBR((uint8_t*)taddr);
#endif

//M20150409 : Explicit Type Casting
//return len;
return (int32_t)len;

int32_t recvfrom(uint8_t sn, uint8_t * buf,
    uint16_t len, uint8_t * addr, uint16_t *port)
{
    //M20150601 : For W5300
#if _WIZCHIP_ == 5300
    uint16_t mr;
    uint16_t mr1;
#else
    uint8_t mr;
#endif
    //
    uint8_t head[8];
    uint16_t pack_len=0;

    CHECK_SOCKNUM();
    //CHECK_SOCKMODE(Sn_MR_UDP);
    //A20150601
    #if _WIZCHIP_ == 5300
        mr1 = getMR();
    #endif
    //
    switch((mr=getSn_MR(sn)) & 0x0F)
    {

case Sn_MR_UDP:
    case Sn_MR_MACRAW:
        break;
#if (_WIZCHIP_ < 5200 )
    case Sn_MR_IPRAW:
    case Sn_MR_PPPoE:
        break;
#endif
    #endif
    default:
        return SOCKERR_SOCKMODE;
    }
CHECK_SOCKDATA();
if(sock_remained_size[sn] == 0)
{
while(1)
{
    pack_len = getSn_RX_RSR(sn);
    if(getSn_SR(sn) == SOCK_CLOSED)
        return SOCKERR_SOCKCLOSED;
    if( (sock_io_mode & (1<<sn)) &&
        (pack_len == 0) ) return SOCK_BUSY;
    if(pack_len != 0) break;
}
//D20150601 : Move it to bottom
// sock_pack_info[sn] = PACK_COMPLETED;
switch (mr & 0x07)
{
    case Sn_MR_UDP :
        if(sock_remained_size[sn] == 0)
            { 
wiz_recv_data(sn, head, 8);
            setSn_CR(sn,Sn_CR_RECV);
            while(getSn_CR(sn));
        // read peer's IP address, port
            number & packet length
        //A20150601 : For W5300
#if _WIZCHIP_ == 5300
    if (mr1 & MR_FS)
    {
        addr[0] = head[1];
        addr[1] = head[0];
        addr[2] = head[3];
        addr[3] = head[2];
        *port = head[5];
        *port = (*port << 8) + head[4];
        sock_remained_size[sn] = head[7];
        sock_remained_size[sn] = (sock_remained_size[sn] << 8) + head[6];
    }
    else
    {
        #endif
        addr[0] = head[0];
        addr[1] = head[1];
        addr[2] = head[2];
        addr[3] = head[3];
        *port = head[4];
        *port = (*port << 8) + head[5];
        sock_remained_size[sn] = head[6];
        sock_remained_size[sn] = (sock_remained_size[sn] << 8) + head[7];
        #if _WIZCHIP_ == 5300
    }
    #endif
    sock_pack_info[sn] = PACK_FIRST;
}
if (len < sock_remained_size[sn])
    pack_len = len;
else pack_len =
sock_remained_size[sn];

    //A20150601 : For W5300
    len = pack_len;
    #if _WIZCHIP_ == 5300
        if(sock_pack_info[sn] & PACK_FIFOBYTE)
            {
                *buf++ =
                sock_remained_byte[sn];
                pack_len -= 1;
                sock_remained_size[sn] -= 1;
                sock_pack_info[sn] &= ~PACK_FIFOBYTE;
            }
    #endif
    //
    // Need to packet length check (default 1472)
    //
    wiz_recv_data(sn, buf, pack_len); // data copy.
    break;
    case Sn_MR_MACRAW :
        if(sock_remained_size[sn] == 0)
        {
            wiz_recv_data(sn, head, 2);
            setSn_CR(sn, Sn_CR_RECV);
            while(getSn_CR(sn));
            // read peer's IP address, port number & packet length
            sock_remained_size[sn] = head[0];
            sock_remained_size[sn] =
            (sock_remained_size[sn] <<8) + head[1];
            if(sock_remained_size[sn] > 1514)
    { close(sn);
        return SOCKFATAL_PACKLEN;
    }

    sock_pack_info[sn] = PACK_FIRST;

    if(len < sock_remained_size[sn])
        pack_len = len;
    else pack_len =
        sock_remained_size[sn];

    wiz_recv_data(sn,buf,pack_len);
    break;

#if (_WIZCHIP_ < 5200 )
    case Sn_MR_IPRAW:
        if(sock_remained_size[sn] == 0)
            {
                wiz_recv_data(sn, head, 6);
                setSn_CR(sn, Sn_CR_RECV);
                while(getSn_CR(sn));
                addr[0] = head[0];
                addr[1] = head[1];
                addr[2] = head[2];
                addr[3] = head[3];
                sock_remained_size[sn] =
                    head[4];
                //M20150401 : For Typing Error
                //sock_remaianed_size[sn] =
                    (sock_remained_size[sn] << 8) + head[5];
                sock_remained_size[sn] =
                    (sock_remained_size[sn] << 8) + head[5];
                sock_pack_info[sn] = PACK_FIRST;
            }
        //
        // Need to packet length check
        //
        if(len < sock_remained_size[sn])
            pack_len = len;
else  pack_len = 
    sock_remained_size[sn];
    wiz_recv_data(sn, buf, pack_len);  //
    data copy.
break;
#endif
default:
    wiz_recv_ignore(sn, pack_len);  //
    data copy.
    sock_remained_size[sn] = pack_len;
    break;
}
setSn_CR(sn,Sn_CR_RECV);
/* wait to process the command... */
while(getSn_CR(sn)) {
    sock_remained_size[sn] -= pack_len;
    //M20150601:
    //if(sock_remained_size[sn] != 0)
    sock_pack_info[sn] |= 0x01;
    if(sock_remained_size[sn] != 0)
    {
        sock_pack_info[sn] |= PACK_REMAINED;
        #if _WIZCHIP_ == 5300
        if(pack_len & 0x01)
            sock_pack_info[sn] |= PACK_FIFOBYTE;
        #endif
    }
    else sock_pack_info[sn] =
        PACK_COMPLETED;
    #if _WIZCHIP_ == 5300
    pack_len = len;
    #endif
    //
    //M20150409 : Explicit Type Casting
    //return pack_len;
    return (int32_t)pack_len;
}
int8_t ctlsocket(uint8_t sn, ctlsock_type cstype, void* arg) {
    uint8_t tmp = 0;
    CHECK_SOCKNUM();
    switch(cstype) {
    case CS_SET_IOMODE:
        tmp = *((uint8_t*)arg);
        if(tmp == SOCK_IO_NONBLOCK)
            sock_io_mode |= (1<<sn);
        else if(tmp == SOCK_IO_BLOCK)
            sock_io_mode &= ~(1<<sn);
        else return SOCKERR_ARG;
        break;
    case CS_GET_IOMODE:
        /*((uint8_t*)arg) = (sock_io_mode >> sn) & 0x0001;
        *((uint8_t*)arg) = (uint8_t)((sock_io_mode >> sn) & 0x0001);*/
        break;
    case CS_GET_MAXTXBUF:
        *((uint16_t*)arg) = getSn_TxMAX(sn);
        break;
    case CS_GET_MAXRXBUF:
        *((uint16_t*)arg) = getSn_RxMAX(sn);
        break;
    case CS_CLR_INTERRUPT:
        if( *((uint8_t*)arg) > SIK_ALL )
            return SOCKERR_ARG;
        break;
    }
}
```c
    setSn_IR(sn, *(uint8_t*)arg);
    break;
  case CS_GET_INTERRUPT:
    *((uint8_t*)arg) = getSn_IR(sn);
    break;
  #if _WIZCHIP_ != 5100
    case CS_SET_INTMASK:
      if ( *((uint8_t*)arg) > SIK_ALL )
        return SOCKERR_ARG;
      setSn_IMR(sn, *(uint8_t*)arg);
      break;
  #endif
  case CS_GET_INTMASK:
    *((uint8_t*)arg) = getSn_IMR(sn);
    break;
  default:
    return SOCKERR_ARG;
  }
  return SOCK_OK;
}

int8_t setsockopt(uint8_t sn, sockopt_type sotype, void* arg)
{
  // M20131220 : Remove warning
  //uint8_t tmp;
  CHECK_SOCKNUM();
  switch(sotype)
  {
    case SO_TTL:
      setSn_TTL(sn, *(uint8_t*)arg);
      break;
    case SO_TOS:
      setSn_TOS(sn, *(uint8_t*)arg);
      break;
    case SO_MSS:
      setSn_MSSR(sn, *(uint16_t*)arg);
  }
```
break;

case SO_DESTIP:
  setSn_DIPR(sn, (uint8_t*)arg);
  break;

case SO_DESTPORT:
  setSn_DPORT(sn, *(uint16_t*)arg);
  break;

#if _WIZCHIP_ != 5100
  case SO_KEEPALIVESEND:
    CHECK_SOCKMODE(Sn_MR_TCP);
    #if _WIZCHIP_ > 5200
      if(getSn_KPALVTR(sn) != 0)
        return SOCKERR_SOCKOPT;
    #endif
    setSn_CR(sn,Sn_CR_SEND_KEEP);
    while(getSn_CR(sn) != 0)
      {
        // M20131220
        //if ((tmp = getSn_IR(sn)) &
        Sn_IR_TIMEOUT)
        if (getSn_IR(sn) &
          Sn_IR_TIMEOUT)
          {
            setSn_IR(sn,
                     Sn_IR_TIMEOUT);
            return SOCKERR_TIMEOUT;
          }
      }
  break;
#else
  case SO_KEEPALIVEAUTO:
    CHECK_SOCKMODE(Sn_MR_TCP);
    setSn_KPALVTR(sn,* (uint8_t*) arg);
    break;
#endif
#endif

default:
```c
int8_t getsockopt(uint8_t sn, sockopt_type sotype, void* arg)
{
    CHECK_SOCKNUM();
    switch(sotype)
    {
        case SO_FLAG:
            *(uint8_t*)arg = getSn_MR(sn) & 0xF0;
            break;
        case SO_TTL:
            *(uint8_t*)arg = getSn_TTL(sn);
            break;
        case SO_TOS:
            *(uint8_t*)arg = getSn_TOS(sn);
            break;
        case SO_MSS:
            *(uint8_t*)arg = getSn_MSSR(sn);
            break;
        case SO_DESTIP:
            getSn_DIPR(sn, (uint8_t*)arg);
            break;
        case SO_DESTPORT:
            *(uint16_t*)arg = getSn_DPORT(sn);
            break;
        #if _WIZCHIP_ > 5200
        case SO_KEEPALIVEAUTO:
            CHECK_SOCKMODE(Sn_MR_TCP);
            *(uint16_t*)arg = getSn_KPALVTR(sn);
            break;
        #endif
    }
    return SOCK_OK;
}
```
case SO_SENDBUF:
    *(uint16_t*) arg = getSn_TX_FSR(sn);
    break;

case SO_RECVBUF:
    *(uint16_t*) arg = getSn_RX_RSR(sn);
    break;

case SO_STATUS:
    *(uint8_t*) arg = getSn_SR(sn);
    break;

case SO_REMAINSIZE:
    if (getSn_MR(sn) == Sn_MR_TCP)
        *(uint16_t*) arg = getSn_RX_RSR(sn);
    else
        *(uint16_t*) arg = sock_remained_size[sn];
    break;

case SO_PACKINFO:
    CHECK_SOCKMODE(Sn_MR_TCP);
    *(uint8_t*) arg = sock_pack_info[sn];
    break;

default:
    return SOCKERR_SOCKOPT;

} return SOCK_OK;
Socket APIs

Go to the documentation of this file.

```
1  //*
   **********************************************
   ******************************
   //******************************************
   ***********************************
   38  //
   39 ঙ/******************************
   **********************************************

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  
56  define WIZCHIP_CREG_BLOCK
57  0x00
58  define WIZCHIP_SREG_BLOCK(N)
      (_WIZCHIP_SN_BASE_+ _WIZCHIP_SN_SIZE_*N)
```
#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define _W5100_IO_BASE_ _WIZCHIP_IO_BASE_

#define IDM_OR ((_WIZCHIP_IO_BASE_ + 0x0000))
#define IDM_AR0 ((_WIZCHIP_IO_BASE_ + 0x0001))
#define IDM_AR1 ((_WIZCHIP_IO_BASE_ + 0x0002))
#define IDM_DR ((_WIZCHIP_IO_BASE_ + 0x0003))
#define _W5100_IO_BASE_ 0x0000
#define IDM_DR (_WIZCHIP_IO_BASE_ + 0x0003)
#define IDM_AR1 (_WIZCHIP_IO_BASE_ + 0x0002)
#define IDM_AR0 (_WIZCHIP_IO_BASE_ + 0x0001)
#define IDM_OR (_WIZCHIP_IO_BASE_ + 0x0000)
#define _W5100_IO_BASE_ 0x0000

// Definition For Legacy Chip Driver //
#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)
#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR, VAL)
#define IINCHIP_READ_BUF(ADDR, BUF, LEN) WIZCHIP_READ_BUF(ADDR, BUF, LEN)
#define IINCHIP_WRITE_BUF(ADDR, BUF, LEN) WIZCHIP_WRITE(ADDR, BUF, LEN)

//-----------			
group -----------------------------
------------------------
#if _WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_
    #define MR (_WIZCHIP_IO_BASE_ + (0x0000)) // Mode
#else
    #define MR (_W5100_IO_BASE_ + (0x0000)) // Mode
#endif
#define GAR (_W5100_IO_BASE_ + (0x0001)) // GW Address
#define SUBR (_W5100_IO_BASE_ + (0x0005)) // SN Mask Address
#define SHAR (_W5100_IO_BASE_ + (0x0009)) // Source Hardware Address
#define SIPR (_W5100_IO_BASE_ + (0x000F)) // Source IP Address
#define IR (_W5100_IO_BASE_ + (0x0015)) // Interrupt
#define _IMR_ (_W5100_IO_BASE_ + (0x0016)) // Socket Interrupt Mask
```c
#define _RTR_ (_W5100_IO_BASE_ + (0x0017)) // Retry Time
#define _RCR_ (_W5100_IO_BASE_ + (0x0019)) // Retry Count
#define RMSR (_W5100_IO_BASE_ + (0x001A)) // Receive Memory Size
#define TMSR (_W5100_IO_BASE_ + (0x001B)) // Transmit Memory Size
#define PATR (_W5100_IO_BASE_ + (0x001C))
#define PTIMER (_W5100_IO_BASE_ + (0x0028)) // PPP LCP Request Timer
#define PMAGIC (_W5100_IO_BASE_ + (0x0029)) // PPP LCP Magic number
#define UIPR0 (_W5100_IO_BASE_ + (0x002A))
#define UPORT0 (_W5100_IO_BASE_ + (0x002E))

//------------------------------ W5100 Socket Registers ------------------------------

// For Backward Compatibility -------------------------------
#define Sn_MR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0000)) // socket
```
Mode register

#define Sn_CR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001)) // channel Sn_CR register

#define Sn_IR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0002)) // channel interrupt register

#define Sn_SR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0003)) // channel status register

#define Sn_PORT(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004)) // source port register

#define Sn_DHAR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006)) // Peer MAC register address

#define Sn_DIPR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x000C)) // Peer IP register address

#define Sn_DPORT(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0010)) // Peer port register address

#define Sn_MSSR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0012)) // Maximum Segment Size(Sn_MSSR0) register address

#define Sn_PROTO(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0014)) // Protocol of IP Header field register in IP raw
```c
#define Sn_TOS(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + 0x0015) // IP Type of Service(TOS) Register
#define Sn_TTL(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0016)) // IP Time to live(TTL) Register
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0017))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0018))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0019))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001A))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001B))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001C))
// Reserved (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001D))
#define Sn_TX_FSR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020)) // Transmit free memory size register
#define Sn_TX_RD(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022)) // Transmit memory read pointer register address
#define Sn_TX_WR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024)) // Transmit memory write pointer register address
```
```c
#define Sn_RX_RSR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0026)) // Received data size register

#define Sn_RX_RD(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0028)) // Read point of Receive memory

#define Sn_RX_WR(sn) (_W5100_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002A)) // Write point of Receive memory

 '<%=W5100%'

// W5100 Register values

/* MODE register values */
#define MR_RST 0x80
#define MR_PB 0x10
#define MR_PPPOE 0x08
#define MR_AI 0x02
#define MR_IND 0x01

/* IR register values */
#define IR_CONFLICT 0x80
#define IR_UNREACH 0x40
```
#define IR_PPPoE 0x20

#define IR_SOCK(sn) (0x01 << sn)

// Sn_MR values
/* Sn_MR Default values */
#define Sn_MR_CLOSE 0x00

#define Sn_MR_TCP 0x01

#define Sn_MR_UDP 0x02
#define Sn_MR_IPRAW 0x03

#define Sn_MR_MACRAW 0x04

#define Sn_MR_PPPoE 0x05

#define Sn_MR_ND 0x20
#define Sn_MR_MC Sn_MR_ND

#define Sn_MR_MF 0x40
#define Sn_MR_MFEN Sn_MR_MF

#define Sn_MR_MULTI 0x80

/* Sn_MR Default values */
/* Sn_CR values */

#define Sn_CR_OPEN    0x01
#define Sn_CR_LISTEN   0x02
#define Sn_CR_CONNECT  0x04
#define Sn_CR_DISCON   0x08
#define Sn_CR_CLOSE    0x10
#define Sn_CR_SEND     0x20
#define Sn_CR_SEND_MAC 0x21
#define Sn_CR_SEND_KEEP 0x22
#define Sn_CR_RECV     0x40
#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

/* Sn_IR values */

#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

/* Sn_SR values */
#define SOCK_CLOSED           0x00
#define SOCK_INIT             0x13
#define SOCK_LISTEN           0x14
#define SOCK_SYNSENT          0x15
#define SOCK_SYNRECV          0x16
#define SOCK_ESTABLISHED      0x17
#define SOCK_FIN_WAIT         0x18
#define SOCK_CLOSING          0x1A
#define SOCK_TIME_WAIT        0x1B
#define SOCK_CLOSE_WAIT       0x1C
#define SOCK_LAST_ACK 0x1D
#define SOCK_UDP 0x22
#define SOCK_IPRAW 0x32
#define SOCK_MACRAW 0x42
#define SOCK_PPPOE 0x5F

// IP PROTOCOL
#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()
#endif

#define WIZCHIP_CRITICAL_EXIT()
    WIZCHIP.CRIS._exit()
// Basic I/O Function //

//M20150601 :  uint16_t AddrSel  -->  uint32_t AddrSel

//

uint8_t  WIZCHIP_READ (uint32_t AddrSel);

void  WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb);

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

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

// Common Register IO function //

#if (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
#define setMR(mr) WIZCHIP_WRITE(MR,mr)
#else
#define setMR(mr) (*((uint8_t*)MR) = mr)
#endif

#if (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
#define getMR() WIZCHIP_READ(MR)
#else
#define getMR() (*((uint8_t*)MR)
#endif

#define setGAR(gar) \

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

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

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

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

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

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

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

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

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

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

#define getIMR() \  WIZCHIP_READ(_IMR_)

#define setRTR(rtr) \  WIZCHIP_WRITE(_RTR_, (uint8_t)(rtr


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

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

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

#define getRCR() \ 
    WIZCHIP_READ(_RCR_)

#define setRMSR(rmsr) \ 
    WIZCHIP_WRITE(RMSR) // Receive Memory Size

#define getRMSR() \ 
    WIZCHIP_READ() // Receive Memory Size

#define setTMSR(rmsr) \ 
    WIZCHIP_WRITE(TMSR) // Receive Memory Size

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

#define getPPPALGO() \ 
    WIZCHIP_READ(PPPALGO)

#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 setSn_SR(sn) \
1427     WIZCHIP_READ(Sn_SR(sn))
1428
1436  #define setSn_PORT(sn, port) { \
1437     WIZCHIP_WRITE(Sn_PORT(sn),
1438             (uint8_t)(port >> 8)); \
1439   }
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn), 1), (uint8_t) port); "
}  

#define getSn_PORT(sn) "
(((uint16_t) WIZCHIP_READ(Sn_PORT(sn)) << 8) +  
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn), 1)) 
)  

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

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

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

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

#define setSn_DPORT(sn, dport) { "
WIZCHIP_WRITE(Sn_DPORT(sn),  
(uint8_t) (dport >> 8)); "
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn), 1), (uint8_t) dport);  "
}  

#define getSn_DPORT(sn) "
((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1))

#define setSn_MSSR(sn, mss) {
    WIZCHIP_WRITE(Sn_MSSR(sn), (uint8_t)(mss>>8));
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1), (uint8_t) mss);
}

#define getSn_MSSR(sn) 
    (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)))

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

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

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

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

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

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

#define setSn_RXMEM_SIZE(sn, rxmemsize) 

WIZCHIP_WRITE(RMSR, 
(WIZCHIP_READ(RMSR) & ~(0x03 << (2*sn))) | 
(rxmemsize << (2*sn)))
#define setSn_RXBUF_SIZE(sn,rxmemsize) 
setSn_RXMEM_SIZE(sn,rxmemsize)
#define getSn_RXMEM_SIZE(sn) 
((WIZCHIP_READ(RMSR) & (0x03 << 
(2*sn))) >> (2*sn))
#define getSn_RXBUF_SIZE(sn) 
getSn_RXMEM_SIZE(sn)
#define setSn_TXMEM_SIZE(sn,txmemsize) 
WIZCHIP_WRITE(TMSR, 
(WIZCHIP_READ(TMSR) & ~(0x03 << (2*sn))) | 
(txmemsize << (2*sn)))
#define setSn_TXBUF_SIZE(sn,txmemsize) 
setSn_TXMEM_SIZE(sn,txmemsize)
#define getSn_TXMEM_SIZE(sn) 
((WIZCHIP_READ(TMSR) & (0x03 << 
(2*sn))) >> (2*sn))
#define getSn_TXBUF_SIZE(sn) 
getSn_TXMEM_SIZE(sn)
uint16_t getSn_TX_FSR(uint8_t sn);
#define getSn_TX_RD(sn) 
(((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + 
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1 )))
#define setSn_TX_WR(sn, txwr) {
 WIZCHIP_WRITE(Sn_TX_WR(sn), 
(uint8_t)(txwr>>8)); 
}
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn), 1), (uint8_t) txwr); \
}

#define getSn_TX_WR(sn) 
((((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + 
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn), 1 ))))

uint16_t getSn_RX_RSR(uint8_t sn);

#define setSn_RX_RD(sn, rxrd) { \
WIZCHIP_WRITE(Sn_RX_RD(sn), (uint8_t)(rxrd>>8)); \
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn), 1), (uint8_t) rxrd); \
}

#define getSn_RX_RD(sn) 
((((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + 
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn), 1 ))))

#define setSn_RX_WR(sn, rxwr) { \
WIZCHIP_WRITE(Sn_RX_WR(sn), (uint8_t)(rxwr>>8)); \
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn), 1), (uint8_t) rxwr); \
}

#define getSn_RX_WR(sn) 

(((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn), 1)))

#define setSn_FRAG(sn, frag) {
	WIZCHIP_WRITE(Sn_FRAG(sn), (uint8_t)(frag >> 8));
	WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn), 1), (uint8_t)frag);
}

#define getSn_FRAG(sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn), 1)))

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

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

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

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

uint32_t getSn_RxBASE(uint8_t sn);

uint32_t getSn_TxBASE(uint8_t sn);
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

Generated on Wed May 4 2016 16:43:58 for Socket APIs by doxygen 1.8.9.1
Socket APIs

socket.h

Go to the documentation of this file.

```c
//******************************************
//***********************************
//******************************************
//***********************************

#ifndef _SOCKET_H_
#define _SOCKET_H_

#include "wizchip_conf.h"

#define SOCKET uint8_t

#define SOCK_OK 1
#define SOCK_BUSY 0
#define SOCK_FATAL -1000

#define SOCK_ERROR 0
#define SOCKERR_SOCKNUM (SOCK_ERROR - 1)
#define SOCKERR_SOCKOPT (SOCK_ERROR - 2)
#define SOCKERR_SOCKINIT (SOCK_ERROR - 3)
#define SOCKERR_SOCKCLOSED (SOCK_ERROR - 4)
```
#define SOCKERR_SOCKMODE (SOCK_ERROR - 5)
#define SOCKERR_SOCKFLAG (SOCK_ERROR - 6)
#define SOCKERR_SOCKSTATUS (SOCK_ERROR - 7)
#define SOCKERR_ARG (SOCK_ERROR - 10)
#define SOCKERR_PORTZERO (SOCK_ERROR - 11)
#define SOCKERR_IPINVALID (SOCK_ERROR - 12)
#define SOCKERR_TIMEOUT (SOCK_ERROR - 13)
#define SOCKERR_DATALEN (SOCK_ERROR - 14)
#define SOCKERR_BUFFER (SOCK_ERROR - 15)

#define SF_ETHER_OWN (Sn_MR_MFEN)
#define SF_IGMP_VER2 (Sn_MR_MC)
#define SF_TCP_NODELAY (Sn_MR_ND)
#define SF_MULTI_ENABLE (Sn_MR_MULTI)

#if _WIZCHIP_ == 5500
  #define SF_BROAD_BLOCK (Sn_MR_BCASTB)
  #define SF_MULTI_BLOCK (Sn_MR_MMB)
  #define SF_IPv6_BLOCK (Sn_MR_MIP6B)
#endif
#define SF_UNI_BLOCK
(Sn_MR_UCASTB)
#endif

//A201505 : For W5300
#if _WIZCHIP_ == 5300
    #define SF_TCP_ALIGN 0x02
#endif

#define SF_I0_NONBLOCK 0x01

/*
 * UDP & MACRAW Packet Information
 */
#define PACK_FIRST 0x80
#define PACK_REMAINED 0x01
#define PACK_COMPLETED 0x00
//A20150601 : For Integrating with W5300
#define PACK_FIFOBYTE 0x02

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

int8_t close(uint8_t sn);

int8_t listen(uint8_t sn);

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

int8_t disconnect(uint8_t sn);

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

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

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

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

// SOCKET CONTROL & OPTION //
#define SOCK_IO_BLOCK      0
#define SOCK_IO_NONBLOCK   1

typedef enum {
    SIK_CONNECTED = (1 << 0),
    SIK_DISCONNECTED = (1 << 1),
    SIK_RECEIVED = (1 << 2),
    SIK_TIMEOUT = (1 << 3),
    SIK_SENT = (1 << 4),
    //M20150410 : Remove the comma of last member
    //SIK_ALL = 0x1F,    //<
    SIK_ALL = 0x1F
} sockint_kind;

typedef enum {
    CS_SET_IOMODE,
    CS_GET_IOMODE,
    CS_GET_MAXTXBUF,
    CS_GET_MAXRXBUF,
    CS_CLR_INTERRUPT,
    CS_GET_INTERRUPT,
    #if _WIZCHIP_ > 5100
370     CS_SET_INTMASK,
371     CS_GET_INTMASK
372 #endif
373 }ctlsock_type;
374
375 typedef enum
376 {
377     SO_FLAG,
378     SO_TTL,
379     SO_TOS,
380     SO_MSS,
381     SO_DESTIP,
382     SO_DESTPORT,
383     #if _WIZCHIP_ != 5100
384     SO_KEEPALIVESEND,
385     #if _WIZCHIP_ > 5200
386     SO_KEEPALIVEAUTO,
387     #endif
388     #endif
389    SO_SENDBUF,
390    SO_RECVBUF,
391    SO_STATUS,
392    SO_REMAINSIZE,
393    SO_PACKINFO
394 }sockopt_type;
395
396 int8_t  ctlsocket(uint8_t sn, ctlsock_type cstype, void* arg);
397
398 int8_t setsockopt(uint8_t sn, sockopt_type sotype, void* arg);
399
400 int8_t getsockopt(uint8_t sn, sockopt_type sotype, void* arg);
401
402 #endif  // _SOCKET_H_
Socket APIs

w5300.h

Go to the documentation of this file.

```c
#define _W5300_H_

#include <stdint.h>
#include "wizchip_conf.h"

#if (_WIZCHIP_ == 5300)
#define _WIZCHIP_SN_BASE_ (0x0200)
#define _WIZCHIP_SN_SIZE_ (0x0040)
#define WIZCHIP_CREG_BLOCK
#define WIZCHIP_SREG_BLOCK(N) (_WIZCHIP_SN_BASE_ + _WIZCHIP_SN_SIZE_*N)
#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)
```
#if (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
    #define _W5300_IO_BASE_ _WIZCHIP_IO_BASE_
#elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
    #define IDM_AR ((_WIZCHIP_IO_BASE_ + 0x0002))
    #define IDM_DR ((_WIZCHIP_IO_BASE_ + 0x0004))
    #define _W5300_IO_BASE_ 0x0000
#elif (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
    #error "Unkonw _WIZCHIP_IO_MODE_"
#endif

// Definition For Legacy Chip Driver //
#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)
#define IINCHIP_WRITE(ADDR,VAL) WIZCHIP_WRITE(ADDR,VAL)
#define IINCHIP_READ_BUF(ADDR,BUF,LEN) WIZCHIP_READ_BUF(ADDR,BUF,LEN)
#define IINCHIP_WRITE_BUF(ADDR,BUF,LEN) WIZCHIP_WRITE(ADDR,BUF,LEN)

//--------------------------
defgroup W5300 Common Registers
--------------------------
#define MR (_WIZCHIP_IO_BASE_)
#define IR (_W5300_IO_BASE_ +
```c
#define _IMR_ (_W5300_IO_BASE_ + 0x02)

#define ICFGR (_W5300_IO_BASE_ + 0x04)

#define INTLEVEL ICFGR

#define SHAR (_W5300_IO_BASE_ + 0x06)

#define GAR (_W5300_IO_BASE_ + 0x08)

#define SUBR (_W5300_IO_BASE_ + 0x10)

#define SIPR (_W5300_IO_BASE_ + 0x14)

#define _RTR_ (_W5300_IO_BASE_ + 0x18)

#define _RCR_ (_W5300_IO_BASE_ + 0x1C)

#define TMS01R (_W5300_IO_BASE_ + 0x20)

#define TMS23R (TMS01R + 2)

#define TMS45R (TMS01R + 4)

#define TMS67R (TMS01R + 6)
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Definition</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>344</td>
<td><code>#define TMSR0</code></td>
<td>TMS01R</td>
</tr>
<tr>
<td>351</td>
<td><code>#define TMSR1</code></td>
<td><code>(TMSR0 + 1)</code></td>
</tr>
<tr>
<td>358</td>
<td><code>#define TMSR2</code></td>
<td><code>(TMSR0 + 2)</code></td>
</tr>
<tr>
<td>365</td>
<td><code>#define TMSR3</code></td>
<td><code>(TMSR0 + 3)</code></td>
</tr>
<tr>
<td>372</td>
<td><code>#define TMSR4</code></td>
<td><code>(TMSR0 + 4)</code></td>
</tr>
<tr>
<td>379</td>
<td><code>#define TMSR5</code></td>
<td><code>(TMSR0 + 5)</code></td>
</tr>
<tr>
<td>386</td>
<td><code>#define TMSR6</code></td>
<td><code>(TMSR0 + 6)</code></td>
</tr>
<tr>
<td>393</td>
<td><code>#define TMSR7</code></td>
<td><code>(TMSR0 + 7)</code></td>
</tr>
<tr>
<td>403</td>
<td><code>#define RMS01R</code></td>
<td><code>(_W5300_IO_BASE_ + 0x28)</code></td>
</tr>
<tr>
<td>410</td>
<td><code>#define RMS23R</code></td>
<td><code>(RMS01R + 2)</code></td>
</tr>
<tr>
<td>417</td>
<td><code>#define RMS45R</code></td>
<td><code>(RMS01R + 4)</code></td>
</tr>
<tr>
<td>424</td>
<td><code>#define RMS67R</code></td>
<td><code>(RMS01R + 6)</code></td>
</tr>
<tr>
<td>431</td>
<td><code>#define RMSR0</code></td>
<td>RMS01R</td>
</tr>
<tr>
<td>438</td>
<td><code>#define RMSR1</code></td>
<td><code>(RMSR0 + 1)</code></td>
</tr>
<tr>
<td>445</td>
<td><code>#define RMSR2</code></td>
<td><code>(RMSR0 + 2)</code></td>
</tr>
<tr>
<td>452</td>
<td><code>#define RMSR3</code></td>
<td><code>(RMSR0 + 3)</code></td>
</tr>
<tr>
<td>459</td>
<td><code>#define RMSR4</code></td>
<td><code>(RMSR0 + 4)</code></td>
</tr>
</tbody>
</table>
#define RMSR5 (RMSR0 + 5)
#define RMSR6 (RMSR0 + 6)
#define RMSR7 (RMSR0 + 7)

#define MTYPER (_W5300_IO_BASE_ + 0x30)
#define PATR (_W5300_IO_BASE_ + 0x32)
// #define PPPALGOR (_W5300_IO_BASE_ + 0x34)
#define PTIMER (_W5300_IO_BASE_ + 0x36)
#define PMAGICR (_W5300_IO_BASE_ + 0x38)
// #define PSTATER (_W5300_IO_BASE_ + 0x3A)
#define PSIDR (_W5300_IO_BASE_ + 0x3C)
#define PDHAR (_W5300_IO_BASE_ + 0x40)
#define UIPR (_W5300_IO_BASE_ + 0x48)
#define UPORTR (_W5300_IO_BASE_ + 0x4C)
```
#define FMTUR (_W5300_IO_BASE_ + 0x4E)

//
#define Sn_RTCR(n) (_W5300_IO_BASE_ + 0x50 + n*2)

#define Pn_BRDYR(n) (_W5300_IO_BASE_ + 0x60 + n*4)

#define Pn_BDPTHR(n) (_W5300_IO_BASE_ + 0x60 + n*4 + 2)

#define IDR (_W5300_IO_BASE_ + 0xFE)

#define VERSIONR IDR

//------------------------------- W5300 SOCKET Registers -----------------------------

#define Sn_MR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x00)

#define Sn_CR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x02)

#define Sn_IMR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x04)

#define Sn_IR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x06)

#define Sn_SSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x08)

#define Sn_SR(n) Sn_SSR(n)
```
#define Sn_PORTR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0A)
#define Sn_PORT(n)  Sn_PORTR(n)

#define Sn_DHAR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0C)
#define Sn_DPORTR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x12)
#define Sn_DPORT(n)  Sn_DPORTR(n)

#define Sn_DIPR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x14)
#define Sn_MSSR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x18)
#define Sn_KPALVTR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1A)
#define Sn_PROTOR(n)  Sn_KPALVTR(n)

#define Sn_TOSR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1C)
#define Sn_TOS(n)  Sn_TOSR(n)

#define Sn_TTLR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x1E)
#define Sn_TTL(n)  Sn_TTLR(n)

#define Sn_TX_WRSR(n)  (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x20)
#define Sn_TX_FSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0024)

#define Sn_RX_RSR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0028)

#define Sn_FRAG(n) Sn_FRAGR(n)

#define Sn_TX_FIFOR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x002E)

#define Sn_RX_FIFOR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0030)

#define Sn_TX_SADR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0032)

#define Sn_RX_SADR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0034)

#define Sn_TX_RD(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0036)

#define Sn_TX_WR(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x0038)

#define Sn_TX_ACK(n) (_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) + 0x003A)
```c
#define Sn_RX_RD(n) 
(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
0x3C)
#define Sn_RX_WR(n) 
(_W5300_IO_BASE_ + WIZCHIP_SREG_BLOCK(n) +
0x3E)

/************************************
/* The bit of MR regsiter defintion */
 ************************************/
#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)

/************************************
/* The bit of IR regsiter defintion */
 ************************************/
#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)

/************************************
/* The bit of Pn_BRDYR regsiter defintion*/
 ************************************/
#define Pn_PEN 												(1 << 7)
```
#define Pn_MT (1 << 6)
#define Pn_PPL (1 << 5)
#define Pn_SN(n) ((n & 0x07) << 0)

/*******************************************************************************/
/* The bit of Sn_MR regsiter defintion */
/*******************************************************************************/
#define Sn_MR_ALIGN (1 << 8)
#define Sn_MR_MULTI (1 << 7)
#define Sn_MR MF (1 << 6)
#define Sn_MR IGMPv (1 << 5)
#define Sn_MR MC Sn_MR IGMPv
#define Sn_MR ND (1 << 5)
#define Sn_MR CLOSE 0x00
#define Sn_MR TCP 0x01
#define Sn_MR UDP 0x02
#define Sn_MR IPRAW 0x03
#define Sn_MR MACRAW 0x04
#define Sn_MR PPPoE 0x05
#define SOCK_STREAM Sn_MR TCP
#define SOCK_DGRAM Sn_MR UDP

/*******************************************************************************/
/* The values of CR defintion */
/*******************************************************************************/
#define Sn_CR OPEN 0x01
#define Sn_CR LISTEN 0x02
#define Sn_CR CONNECT 0x04
#define Sn_CR_DISCON    0x08
#define Sn_CR_CLOSE     0x10
#define Sn_CR_SEND      0x20
#define Sn_CR_SEND_MAC  0x21
#define Sn_CR_SEND_KEEP 0x22
#define Sn_CR_RECV      0x40
#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
#define SOCK_ARP        0x01
#define SOCK_INIT       0x13
#define SOCK_LISTEN     0x14
#define SOCK_SYNSENT 0x15
#define SOCK_SYNRECV 0x16
#define SOCK_ESTABLISHED 0x17
#define SOCK_FIN_WAIT 0x18
#define SOCK_CLOSING 0x1A
#define SOCK_TIME_WAIT 0x1B
#define SOCK_CLOSE_WAIT 0x1C
#define SOCK_LAST_ACK 0x1D
#define SOCK_UDP 0x22
#define SOCK_IPRAW 0x32
#define SOCK_MACRAW 0x42
#define SOCK_PPPoE 0x5F

/* IP PROTOCOL */
#define IPPROTO_IP 0
    //< Dummy for IP
#define IPPROTO_ICMP 1
    //< Control message protocol
#define IPPROTO_IGMP 2
    //< Internet group management protocol
#define IPPROTO_GGP 3
    //< Gateway^2 (deprecated)
#define IPPROTO_TCP 6
    //< TCP
#define IPPROTO_PUP 12
    //< PUP
#define IPPROTO_UDP 17
/< UDP
#define IPPROTO_IDP 22
/ < XNS idp
#define IPPROTO_ND 77
/ < UNOFFICIAL net disk protocol
#define IPPROTO_RAW 255
/ < Raw IP packet

#define WIZCHIP_CRITICAL_ENTER()
WIZCHIP.CRIS._enter()

#define WIZCHIP_CRITICAL_EXIT()
WIZCHIP.CRIS._exit()

// Basic I/O Function //
uint16_t WIZCHIP_READ (uint32_t AddrSel);
void WIZCHIP_WRITE (uint32_t AddrSel,
                     uint16_t wb );

/******************************
* COMMON Register Access Function *
*******************************/
#if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
    #if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
        #define setMR(mr) \/*((uint8_t*)MR) = (uint8_t)((mr)  
                      >> 8); /*((uint8_t*)WIZCHIP_OFFSET_INC(MR,1))  
                      = (uint8_t)((mr) & 0xFF))

    #else
        #define setMR(mr) \/*((uint32_t*)MR) = (uint32_t)((mr)  
                      >> 32); /*((uint32_t*)WIZCHIP_OFFSET_INC(MR,4))  
                      = (uint32_t)((mr) & 0xFFFFFFFF)*/
    #endif
#else
    #define setMR(mr) \/*((uint8_t*)MR) = (uint8_t)((mr)  
                      >> 8); /*((uint8_t*)WIZCHIP_OFFSET_INC(MR,1))  
                      = (uint8_t)((mr) & 0xFF))*/
#endif
elif (_WIZCHIP_IO_BUS_WIDTH_ == 16)
    
define setMR(mr) (*
        ((uint16_t*)MR)) = (uint16_t)((mr) & 0xFFFF))
else
    #error "Unknown
    _WIZCHIP_IO_BUS_WIDTH_. You should be define
    _WIZCHIP_IO_BUS_WIDTH_ as 8 or 16."
endif

else
    #error "Unknown _WIZCHIP_IO_MODE_."
endif

if (_WIZCHIP_IO_MODE_ &
    _WIZCHIP_IO_MODE_BUS_)
    if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
        #define getMR() (((uint16_t)*
            ((uint8_t*)MR)) << 8) + (((uint16_t)*
            ((uint8_t*)WIZCHIP_OFFSET_INC(MR,1))) &
            0x00FF))
    elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
        #define getMR() (*((uint16_t*)MR))
    else
        #error "Unknown
        _WIZCHIP_IO_BUS_WIDTH_. You should be define
        _WIZCHIP_IO_BUS_WIDTH_ as 8 or 16."
    endif
else
    #error "Unknown _WIZCHIP_IO_MODE_."
endif

#define setIR(ir) \n    WIZCHIP_WRITE(IR, ir & 0xF0FF)

#define getIR() \n    (WIZCHIP_READ(IR) & 0xF0FF)
#define  setIMR(imr) \  WIZCHIP_WRITE(_IMR_, imr & 0xF0FF)

#define  getIMR() \  (WIZCHIP_READ(_IMR_) & 0xF0FF)

#define  setSHAR(shar) { \  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)); \  }

#define  getSHAR(shar) { \  (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))); \  }

}
```c
#define setGAR(gar) {
    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));
}

#define getGAR(gar) {
    (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)));
}

#define setSUBR(subr) {
    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));
}

#define getSUBR(subr) {
    (subr)[0] = (uint8_t) (WIZCHIP_READ(SUBR) >> 8);
    (subr)[1] = (uint8_t) (WIZCHIP_READ(SUBR));
```

```c
(subr)[2] = (uint8_t) (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2)) >> 8); \
(subr)[3] = (uint8_t) (WIZCHIP_READ(WIZCHIP_OFFSET_INC(SUBR,2))); \
}

#define setSIPR(sipr) { \
    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)); \
}

#define getSIPR(sipr) { \
    (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))); \
}

#define setRTR(rtr) WIZCHIP_WRITE(_RTR_, rtr)

#define getRTR() WIZCHIP_READ(_RTR_)

#define setRCR(rcr) 
```
WIZCHIP_WRITE(_RCR_, ((uint16_t)rcr)&0x00FF)

#define getRCR() ((uint8_t)(WIZCHIP_READ(_RCR_) & 0x00FF))

#define setTMS01R(tms01r) 
  WIZCHIP_WRITE(TMS01R,tms01r)

#define getTMS01R() 
  WIZCHIP_READ(TMS01R)

#define setTMS23R(tms23r) 
  WIZCHIP_WRITE(TMS23R,tms23r)

#define getTMS23R() 
  WIZCHIP_READ(TMS23R)

#define setTMS45R(tms45r) 
  WIZCHIP_WRITE(TMS45R,tms45r)

#define getTMS45R() 
  WIZCHIP_READ(TMS45R)

#define setTMS67R(tms67r) 
  WIZCHIP_WRITE(TMS67R,tms67r)

#define getTMS67R() 
  WIZCHIP_READ(TMS67R)

void setTMSR(uint8_t sn,uint8_t tmsr);
#define setSn_TXBUF_SIZE(sn, tmsr) 
  setTMSR(sn, tmsr)

uint8_t getTMSR(uint8_t sn);
```c
#define getSn_TXBUF_SIZE(sn)      getTMSR(sn)
#define setRMS01R(rms01r)   
    WIZCHIP_WRITE(RMS01R,rms01r)
#define getRMS01R()        
    WIZCHIP_READ(RMS01R)
#define setRMS23R(rms23r)  
    WIZCHIP_WRITE(RMS23R,rms23r)
#define getRMS23R()        
    WIZCHIP_READ(RMS23R)
#define setRMS45R(rms45r)  
    WIZCHIP_WRITE(RMS45R,rms45r)
#define getRMS45R()        
    WIZCHIP_READ(RMS45R)
#define setRMS67R(rms67r)  
    WIZCHIP_WRITE(RMS67R,rms67r)
#define getRMS67R()        
    WIZCHIP_READ(RMS67R)

void setRMSR(uint8_t sn, uint8_t rmsr);
#define setSn_RXBUF_SIZE(sn,rmsr)  
    setRMSR(sn, rmsr)

uint8_t getRMSR(uint8_t sn);
#define getSn_RXBUF_SIZE(sn)      getRMSR(sn)

#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, 
1755 ((uint16_t)ptimer) & 0x00FF)
1755
1762 #define getPTIMER() \
1763 ((uint8_t)(WIZCHIP_READ(PTIMER) & 0x00FF))
1764
1771 #define setPMAGIC(pmagic) \
1772 WIZCHIP_WRITE(PMAGIC, 
1773 ((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) { \n1797 (pdhar)[0] = (uint8_t) 
1798 (WIZCHIP_READ(PDHAR) >> 8); \n1799 (pdhar)[1] = (uint8_t) 
1800 (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR, 2)) >> 8); \n1801 (pdhar)[2] = (uint8_t) 
1802 (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR, 2))); \n1803 (pdhar)[3] = (uint8_t) 
1804 (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR, 2))); \n1805 }
(pdhar)[4] = (uint8_t) (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4)) >> 8); \
(pdhar)[5] = (uint8_t) (WIZCHIP_READ(WIZCHIP_OFFSET_INC(PDHAR,4))); \\
}

#define getUIPR(uipr) { \\
    (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))); \\
}

#define getUPORTR() \n    WIZCHIP_READ(UPORTR)

#define getFMTUR() \n    WIZCHIP_READ(FMTUR)

#define getPn_BRDYM(p) \n    ((uint8_t)(WIZCHIP_READ(Pn_BRDYM(p)) & 0x00FF))

#define setPn_BRDYM(p, brdym) \n    WIZCHIP_WRITE(Pn_BRDYM(p), brdym & 0x00E7)

#define getPn_BDPTHR(p) \n    WIZCHIP_READ(Pn_BDPTHR(p))
```c
#define setPn_BDPTHR(p, bdpthr) \  
    WIZCHIP_WRITE(Pn_BDPTH(p), bdpthr)

#define getIDR() \  
    WIZCHIP_READ(IDR)

/***********************************
 * SOCKET Register Access Function *
 ***********************************/
#define setSn_MR(sn, mr) \  
    WIZCHIP_WRITE(Sn_MR(sn), mr)
#define getSn_MR(sn) \  
    WIZCHIP_READ(Sn_MR(sn))
#define setSn_CR(sn, cr) \  
    WIZCHIP_WRITE(Sn_CR(sn), ((uint16_t)cr) & 0x00FF)
#define getSn_CR(sn) \  
    ((uint8_t)WIZCHIP_READ(Sn_CR(sn)))
#define setSn_IMR(sn, imr) \  
    WIZCHIP_WRITE(Sn_IMR(sn), ((uint16_t)imr) & 0x00FF)
#define getSn_IMR(sn) \  
    ((uint8_t)WIZCHIP_READ(Sn_IMR(sn)))
#define setSn_IR(sn, ir) \  
    WIZCHIP_WRITE(Sn_IR(sn), ((uint16_t)ir) & 0x00FF)
#define getSn_IR(sn) \  
```
```c
#define getSn_SSR(sn)  
(uint8_t)WIZCHIP_READ(Sn_SR(sn))
#define getSn_SR(sn)  
getSn_SSR(sn)
#define setSn_PORTR(sn, port)  
WIZCHIP_WRITE(Sn_PORTR(sn), port)
#define setSn_PORT(sn, port)  
setSn_PORTR(sn, port)
#define getSn_PORTR(sn, port)  
WIZCHIP_READ(Sn_PORTR(sn))
#define getSn_PORT(sn)  
getSn_PORTR(sn)
#define setSn_DHAR(sn, dhar)  
{
  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));
}
#define getSn_DHAR(sn, dhar)  
{
  (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));
}
#define setSn_DPORTR(sn, dport) 
WIZCHIP_WRITE(Sn_DPORTR(sn), dport)
#define setSn_DPORT(sn, dport)
setSn_DPORTR(sn, dport)
#define getSn_DPORTR(sn) 
WIZCHIP_READ(Sn_DPORTR(sn))
#define getSn_DPORT(sn) getSn_DPORTR(sn)
#define setSn_DIPR(sn, dipr) {
    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));
}
#define getSn_DIPR(sn, dipr) {
    (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)); \ 
    } 

#define setSn_MSSR(sn, mss) \  WIZCHIP_WRITE(Sn_MSSR(sn), mss) 
#define getSn_MSSR(sn) \  WIZCHIP_READ(Sn_MSSR(sn)) 

#define setSn_KPALVTR(sn, kpalvt) \  WIZCHIP_WRITE(Sn_KPALVTR(sn), (WIZCHIP_READ(Sn_KPALVTR(sn)) & 0x00FF) | (((uint16_t)kpalvt)<<8)) 
#define getSn_KPALVTR(sn) \  ((uint8_t)WIZCHIP_READ(Sn_KPALVTR(sn)) >> 8)) 

#define setSn_PROTOR(sn, proto) \  WIZCHIP_WRITE(Sn_PROTOR(sn), (WIZCHIP_READ(Sn_PROTOR(sn)) & 0xFF00) | (((uint16_t)proto) & 0x00FF)) 
#define setSn_PROTO(sn,proto) \  setSn_PROTOR(sn,proto) 

#define getSn_PROTO(sn) \  ((uint8_t)WIZCHIP_READ(Sn_PROTOR(sn)))
```c
#define setSn_TX_WRSR(sn, txwrs) {
    WIZCHIP_WRITE(Sn_TX_WRSR(sn), (uint16_t)(((uint32_t)txwrs) >> 16));
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn), 2), (uint16_t)txwrs);
}

#define getSn_TX_WRSR(sn) ((((uint32_t)WIZCHIP_READ(Sn_TX_WRSR(sn))) << 16) + (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WRSR(sn), 1))) & 0x0000FFFF))

uint32_t getSn_TX_FSR(uint8_t sn);
uint32_t getSn_RX_RSR(uint8_t sn);

#define setSn_TX_FIFOR(sn, txfifo) WIZCHIP_WRITE(Sn_TX_FIFOR(sn), txfifo);
#define getSn_TX_FIFOR(sn) WIZCHIP_READ(Sn_TX_FIFOR(sn));

#define setSn_TOSR(sn, tos) WIZCHIP_WRITE(Sn_TOS(sn), ((uint16_t)tos) & 0x0000FF)
#define setSn_TOS(sn, tos) setSn_TOSR(sn, tos)

#define getSn_TOSR(sn) ((uint8_t)WIZCHIP_READ(Sn_TOSR(sn)))
#define getSn_TOS(sn) getSn_TOSR(sn)
```
```c
#define setSn_TTLR(sn, ttl) \
    WIZCHIP_WRITE(Sn_TTLR(sn), \
                  ((uint16_t)ttl) & 0x00FF)
#define setSn_TTL(sn, ttl) 
    setSn_TTLR(sn, ttl)

#define getSn_TTLR(sn) \
    ((uint8_t)WIZCHIP_READ(Sn_TTL(sn)))
#define getSn_TTL(sn) 
    getSn_TTLR(sn)

#define setSn_FRAGR(sn, frag) \
    WIZCHIP_WRITE(Sn_FRAGR(sn), (uint16_t) \
                  (frag >> 8))
#define setSn_FRAG(sn, frag) 
    setSn_FRAGR(sn, flag)

#define getSn_FRAGR(sn) 
    (WIZCHIP_READ(Sn_FRAG(sn)) << 8)
#define getSn_FRAG(sn) 
    getSn_FRAGR(sn)

// Sn_TXBUF & Sn_RXBUF IO function //

#define getSn_RxMAX(sn) \
    (((uint32_t)getSn_RXBUF_SIZE(sn)) << 10)
#define getSn_TxMAX(sn) \
    (((uint32_t)getSn_TXBUF_SIZE(sn)) << 10)

void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint32_t len);
```
void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint32_t len);

void wiz_recv_ignore(uint8_t sn, uint32_t len);

#include/_W5300_H_
Socket APIs

Go to the documentation of this file.

```c
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)
```
#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + N)
```c
#define _WIZCHIP_IO_MODE_ 
#define _WIZCHIP_IO_MODE_BUS_ 
#define _WIZCHIP_IO_BASE_ 
#define _WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_BUS_ 
#define MR (_WIZCHIP_IO_BASE_ + (0x0000)) // Mode 
#define MR (_W5200_IO_BASE_ + (0x0000)) // Mode 
#endif 
#define GAR (_W5200_IO_BASE_ + (0x0001)) // GW Address 
#define SUBR (_W5200_IO_BASE_ + (0x0005)) // SN Mask Address 
#define SHAR (_W5200_IO_BASE_ + (0x0009)) // Source Hardware Address 
#define SIPR (_W5200_IO_BASE_ + (0x000F)) // Source IP Address 
#define IR (_W5200_IO_BASE_ + (0x0015)) // Interrupt 
#define _IMR_ (_W5200_IO_BASE_ + (0x0016)) // Socket Interrupt Mask 
#define _RTR_ (_W5200_IO_BASE_ + (0x0017)) // Retry Time 
#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
```c
#define PMAGIC (_W5200_IO_BASE_ + (0x0029)) // PPP LCP Magic number

#define INTLEVEL (_W5200_IO_BASE_ + (0x0030)) // Interrupt Low Level Timer

#define IR2 (_W5200_IO_BASE_ + (0x0034)) // Socket Interrupt

#define PHYSTATUS (_W5200_IO_BASE_ + (0x0035)) // PHY Status

#define IMR2 (_W5200_IO_BASE_ + (0x0036)) // Interrupt Mask
```

//------------------------------- W5200 Socket Registers -----------------------------
#define Sn_MR(sn)    (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0000)) // socket Mode register
#define Sn_CR(sn)    (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0001)) // channel Sn_CR register
#define Sn_IR(sn)    (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0002)) // channel interrupt register
#define Sn_SR(sn)    (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0003)) // channel status register
#define Sn_PORT(sn)  (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0004)) // source port register
#define Sn_DHAR(sn)  (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0006)) // Peer MAC register address
#define Sn_DIPR(sn)  (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x000C)) // Peer IP register address
#define Sn_DPORT(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0010)) // Peer port register address
#define Sn_MSSR(sn)  (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0012)) // Maximum
Segment Size(Sn_MSSR0) register address

#define Sn_PROTO(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0014)) // Protocol of IP Header field register in IP raw mode

#define Sn_TOS(sn) (WIZCHIP_SREG_BLOCK(sn) + 0x0015) // IP Type of Service(TOS) Register

#define Sn_TTL(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0016)) // IP Time to live(TTL) Register

#define Sn_RXMEM_SIZE(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001E)) // Receive memory size register

#define Sn_TXMEM_SIZE(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x001F)) // Transmit memory size register
#define Sn_TX_FSR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0020)) // Transmit free memory size register

#define Sn_TX_RD(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0022)) // Transmit memory read pointer register address

#define Sn_TX_WR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0024)) // Transmit memory write pointer register address

#define Sn_RX_RSR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0026)) // Received data size register

#define Sn_RX_RD(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x0028)) // Read point of Receive memory

#define Sn_RX_WR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002A)) // Write point of Receive memory

#define Sn_IMR(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002C)) // socket interrupt mask register

#define Sn_FRAG(sn) (_W5200_IO_BASE_ + WIZCHIP_SREG_BLOCK(sn) + (0x002D)) // frag field value in IP header register

//----------------------------- W5200 Register values -----------------------------
/* MODE register values */
#define MR_RST 0x80
#define MR_WOL 0x20
#define MR_PB 0x10
#define MR_PPPOE 0x08
#define MR_AI 0x02
#define MR_IND 0x01

/* IR register values */
#define IR_CONFLICT 0x80
#define IR_PPPoE 0x20
#define PHYSTATUS_LINK 0x20
#define PHYSTATUS_POWERSAVE 0x10
#define PHYSTATUS_POWERDOWN 0x08

// Sn_MR values
/* Sn_MR Default values */
#define Sn_MR_CLOSE 0x00
#define Sn_MR_TCP 0x01
#define Sn_MR_UDP      0x02
#define Sn_MR_IPRAW    0x03
#define Sn_MR_MACRAW   0x04
#define Sn_MR_PPPOE    0x05
#define Sn_MR_ND       0x20
#define Sn_MR_MC       Sn_MR_ND
#define Sn_MR_MF       0x40
#define Sn_MR_MFEN     Sn_MR_MF
#define Sn_MR_MULTI    0x80

/* Sn_MR Default values */
#define Sn_MR_OPEN     0x01
#define Sn_MR_LISTEN   0x02
#define Sn_MR_CONNECT  0x04
#define Sn_CR_OPEN     0x01
#define Sn_CR_LISTEN   0x02
#define Sn_CR_CONNECT  0x04
#define Sn_CR_DISCON   0x08
#define Sn_CR_CLOSE     0x10
#define Sn_CR_SEND     0x20
#define Sn_CR_SEND_MAC  0x21
#define Sn_CR_SEND_KEEP 0x22
#define Sn_CR_RECV     0x40
#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

/* Sn_IR values */
#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
<table>
<thead>
<tr>
<th>Line</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1018</td>
<td><code>#define Sn_IR_CON</code></td>
<td>0x01</td>
</tr>
<tr>
<td>1019</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1020</td>
<td><code>/* Sn_SR values */</code></td>
<td></td>
</tr>
<tr>
<td>1026</td>
<td><code>#define SOCK_CLOSED</code></td>
<td>0x00</td>
</tr>
<tr>
<td>1027</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1034</td>
<td><code>#define SOCK_INIT</code></td>
<td>0x13</td>
</tr>
<tr>
<td>1035</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1036</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1042</td>
<td><code>#define SOCK_LISTEN</code></td>
<td>0x14</td>
</tr>
<tr>
<td>1043</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1049</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1051</td>
<td><code>#define SOCK_SYNSENT</code></td>
<td>0x15</td>
</tr>
<tr>
<td>1052</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1059</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1068</td>
<td><code>#define SOCK_ESTABLISHED</code></td>
<td>0x17</td>
</tr>
<tr>
<td>1069</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1076</td>
<td><code>#define SOCK_FIN_WAIT</code></td>
<td>0x18</td>
</tr>
<tr>
<td>1077</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1084</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1085</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1092</td>
<td><code>#define SOCK_TIME_WAIT</code></td>
<td>0x1B</td>
</tr>
<tr>
<td>1093</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td><code>#define SOCK_CLOSE_WAIT</code></td>
<td>0x1C</td>
</tr>
<tr>
<td>1101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1107</td>
<td><code>#define SOCK_LAST_ACK</code></td>
<td>0x1D</td>
</tr>
<tr>
<td>1108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1115</td>
<td><code>#define SOCK_UDP</code></td>
<td>0x22</td>
</tr>
<tr>
<td>1116</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1117</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1123</td>
<td><code>#define SOCK_IPRAW</code></td>
<td>0x32</td>
</tr>
<tr>
<td>1124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1131</td>
<td><code>#define SOCK_MACRAW</code></td>
<td>0x42</td>
</tr>
<tr>
<td>1132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1140</td>
<td><code>#define SOCK_PPPOE</code></td>
<td>0x5F</td>
</tr>
<tr>
<td>1141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1147</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1141 // 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

1165 #define WIZCHIP_CRITICAL_ENTER()
    WIZCHIP.CRIS._enter()

1167 #ifdef _exit
1168 #undef _exit
1169 #endif

1182 #define WIZCHIP_CRITICAL_EXIT()
    WIZCHIP.CRIS._exit()

1187 // Basic I/O Function //
1195 uint8_t WIZCHIP_READ (uint32_t AddrSel);
1204 void WIZCHIP_WRITE (uint32_t AddrSel, uint8_t wb);
1213 void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len);
1222 void WIZCHIP_WRITE_BUF (uint32_t AddrSel,
uint8_t* pBuf, uint16_t len);

// Common Register IO function //
#if (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
#define setMR(mr) WIZCHIP_WRITE(MR,mr)
#else
#define setMR(mr) (*((uint8_t*)MR) = mr)
#endif

#if (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
#define getMR() WIZCHIP_READ(MR)
#else
#define getMR() (*((uint8_t*)MR)
#endif

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

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

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

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

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

#define getSHAR(shar) 
           WIZCHIP_READ_BUF(SHAR,shar,6)
```c
#define setSIPR(sipr)       
   WIZCHIP_WRITE_BUF(SIPR, sipr, 4)

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

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

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

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

#define getIMR()             
   (WIZCHIP_READ(IMR2) & 0xA0)

#define setRTR(rtr)         
   {                           
      WIZCHIP_WRITE(_RTR_, (uint8_t)(rtr >> 8));   
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),  
                     (uint8_t) rtr);   
   }
```

//M20150410 : Replace _IMR_ with IMR2 for integrating with ioLibrary

/ *
#define setIMR(imr)         
   WIZCHIP_WRITE(_IMR_, imr)
 *
#define setIMR(imr)         
   WIZCHIP_WRITE(IMR2, imr & 0xA0)

//M20150410 : Replace _IMR_ with IMR2 for integrating with ioLibrary

/ *
#define getIMR()             
   WIZCHIP_READ(_IMR_)
 *
#define getIMR()             
   (WIZCHIP_READ(IMR2) & 0xA0)

#define setRTR(rtr)         
   {                           
      WIZCHIP_WRITE(_RTR_, (uint8_t)(rtr >> 8));   
      WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1),  
                     (uint8_t) rtr);   
   }
```c
#define getRTR() 
    (((uint16_t)WIZCHIP_READ(_RTR_) << 8) + 
     WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_, 1)))
#define setRCR(rcr) 
    WIZCHIP_WRITE(_RCR_, rcr)
#define getRCR() 
    WIZCHIP_READ(_RCR_)
#define getPATR() 
    (((uint16_t)WIZCHIP_READ(PATR) << 8) 
     + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PATR, 1)))
#define getPPPALGO() 
    WIZCHIP_READ(PPPALGO)
#define getVERSIONR() 
    WIZCHIP_READ(VERSIONR)
#define setPTIMER(ptimer) 
    WIZCHIP_WRITE(PTIMER, ptimer)
#define getPTIMER() 
    WIZCHIP_READ(PTIMER)
#define setPMAGIC(pmagic) 
    WIZCHIP_WRITE(PMAGIC, pmagic)
#define getPMAGIC() 
    WIZCHIP_READ(PMAGIC)
#define setINTLEVEL(intlevel) {
```
WIZCHIP_WRITE(INTLEVEL, (uint8_t) (intlevel >> 8)); \
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL, 1), (uint8_t) intlevel); \
}

#define getINTLEVEL() \                  
((uint16_t)WIZCHIP_READ(INTLEVEL) << 8) + 
WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL, 1)))
#define setIR2(ir2) \                   
WIZCHIP_WRITE(IR2, ir2)
#define setSIR(ir2) setIR2(ir2)
#define getIR2() \                      
WIZCHIP_READ(IR2)
#define getSIR() getIR2()
#define getPHYSTATUS() \                
WIZCHIP_READ(PHYSTATUS)

#define setIMR2(imr2) \                
WIZCHIP_WRITE(_IMR_, imr2)
#define setSIMR(imr2) setIMR2(imr2)

//M20150410 : Replace IMR2 with _IMR_ for integrating with ioLibrary
/*
#define setIMR2(imr2) \                
WIZCHIP_WRITE(IMR2, (imr2 & 0xA0))
*/
#define setIMR2(imr2) \                
WIZCHIP_WRITE(_IMR_, imr2)
#define setSIMR(imr2) setIMR2(imr2)

//M20150410 : Replace IMR2 with _IMR_ for integrating with ioLibrary
/*
#define getIMR2() \                   
*/
(WIZCHIP_READ(IMR2) & 0xA0)
*/
#define getIMR2() WIZCHIP_READ(_IMR_)
#define getSIMR() getIMR2()
// Socket N register I/O function //
#define setSn_MR(sn, mr) WIZCHIP_WRITE(Sn_MR(sn), mr)
#define getSn_MR(sn) WIZCHIP_READ(Sn_MR(sn))
#define setSn_CR(sn, cr) WIZCHIP_WRITE(Sn_CR(sn), cr)
#define getSn_CR(sn) WIZCHIP_READ(Sn_CR(sn))
#define setSn_IR(sn, ir) WIZCHIP_WRITE(Sn_IR(sn), ir)
#define getSn_IR(sn) WIZCHIP_READ(Sn_IR(sn))
#define setSn_IMR(sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), imr)
#define getSn_IMR(sn) WIZCHIP_READ(Sn_IMR(sn))
#define getSn_SR(sn) WIZCHIP_READ(Sn_SR(sn))
#define setSn_PORT(sn, port) {
    WIZCHIP_WRITE(Sn_PORT(sn), (uint8_t)(port >> 8));
}
```c
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1), (uint8_t) port); \
} 

#define getSn_PORT(sn) \n((uint16_t)WIZCHIP_READ(Sn_PORT(sn)) << 8) + 
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1)) \n
#define setSn_DHAR(sn, dhar) \nWIZCHIP_WRITE_BUF(Sn_DHAR(sn), dhar, 6) \n
#define getSn_DHAR(sn, dhar) \nWIZCHIP_READ_BUF(Sn_DHAR(sn), dhar, 6) \n
#define setSn_DIPR(sn, dipr) \nWIZCHIP_WRITE_BUF(Sn_DIPR(sn), dipr, 4) \n
#define getSn_DIPR(sn, dipr) \nWIZCHIP_READ_BUF(Sn_DIPR(sn), dipr, 4) \n
#define setSn_DPORT(sn, dport) { \
WIZCHIP_WRITE(Sn_DPORT(sn), (uint8_t) (dport>>8)); \nWIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_DPORT(sn), 1), (uint8_t) dport); \n} 

#define getSn_DPORT(sn) 
```
#define setSn_MSSR(sn, mss) { 
    WIZCHIP_WRITE(Sn_MSSR(sn), (uint8_t)(mss>>8)); 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1), (uint8_t) mss); 
}

#define getSn_MSSR(sn) 
    (((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)))

#define setSn_PROTO(sn, proto) 
    WIZCHIP_WRITE(Sn_PROTO(sn), proto)

#define getSn_PROTO(sn) 
    WIZCHIP_READ(Sn_PROTO(sn))

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

//M20150601 : Fixed Wrong Register address

/*
#define setSn_PROTO(sn, proto) 
    WIZCHIP_WRITE(Sn_PROTO(sn), proto)
 */

//M20150601 : Fixed Wrong Register address

/*
#define getSn_PROTO(sn) 
    WIZCHIP_READ(Sn_PROTO(sn))
 */

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

#define setSn_TOS(sn, tos) 
    WIZCHIP_WRITE(Sn_TOS(sn), tos)
#define getSn_TOS(sn)  
    WIZCHIP_READ(Sn_TOS(sn))
#define setSn_TTL(sn, ttl)  
    WIZCHIP_WRITE(Sn_TTL(sn), ttl)
#define getSn_TTL(sn)  
    WIZCHIP_READ(Sn_TTL(sn))
#define setSn_RXMEM_SIZE(sn, rxmembsize)  
    WIZCHIP_WRITE(Sn_RXMEM_SIZE(sn), rxmembsize)
#define setSn_RXBUF_SIZE(sn, rxmembsize)  
    setSn_RXMEM_SIZE(sn, rxmembsize)
#define getSn_RXMEM_SIZE(sn)  
    WIZCHIP_READ(Sn_RXMEM_SIZE(sn))
#define getSn_RXBUF_SIZE(sn)  
    getSn_RXMEM_SIZE(sn)
#define setSn_TXMEM_SIZE(sn, txmembsize)  
    WIZCHIP_WRITE(Sn_TXMEM_SIZE(sn), txmembsize)
#define setSn_TXBUF_SIZE(sn, txmembsize)  
    setSn_TXMEM_SIZE(sn, txmembsize)
#define getSn_TXMEM_SIZE(sn)  
    WIZCHIP_READ(Sn_TXMEM_SIZE(sn))
#define getSn_TXBUF_SIZE(sn)  
    getSn_TXMEM_SIZE(sn)

tuint16_t getSn_TX_FSR(uint8_t sn);
#define getSn_TX_RD(sn) \\
(((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn),1 )))

#define setSn_TX_WR(sn, txwr) { \\
    WIZCHIP_WRITE(Sn_TX_WR(sn), (uint8_t)(txwr>>8)); \\
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn), 1), (uint8_t)txwr); \\
} 

#define getSn_TX_WR(sn) \\
(((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1 )))

uint16_t getSn_RX_RSR(uint8_t sn);

#define setSn_RX_RD(sn, rxrd) { \\
    WIZCHIP_WRITE(Sn_RX_RD(sn), (uint8_t)(rxrd>>8)); \\
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn), 1), (uint8_t)rxrd); \\
} 

#define getSn_RX_RD(sn) \\
(((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1 )))

#define setSn_RX_WR(sn, rxwr) {

WIZCHIP_WRITE(Sn_RX_WR(sn), (uint8_t)(rxwr>>8));

WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn), 1), (uint8_t) rxwr);

#define getSn_RX_WR(sn) (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1)))

#define setSn_IMR(sn, imr) WIZCHIP_WRITE(Sn_IMR(sn), imr)

#define getSn_IMR(sn) WIZCHIP_READ(Sn_IMR(sn))

#define setSn_FRAG(sn, frag) {
    WIZCHIP_WRITE(Sn_FRAG(sn), (uint8_t)(frag>>8));
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1), (uint8_t) frag);
}

#define getSn_FRAG(sn) (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)))

#define getSn_RxMAX(sn) (((uint16_t)getSn_RXMEM_SIZE(sn) << 10)
#define getSn_TxMAX(sn) 
    ((uint16_t)getSn_TXMEM_SIZE(sn) << 10)

#define getSn_RxMASK(sn) 
    ((uint16_t)getSn_RxMAX(sn) - 1)

#define getSn_TxMASK(sn) 
    ((uint16_t)getSn_TxMAX(sn) - 1)

uint16_t getSn_RxBASE(uint8_t sn);

uint16_t getSn_TxBASE(uint8_t sn);

// Sn_TXBUF & Sn_RXBUF IO function //

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

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

void wiz_recv_ignore(uint8_t sn, uint16_t len);

#endif

#endif //_W5200_H_
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 - ptdiff_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)
```
void wizchip_cs_select(void) {};

//void wizchip_cs_deselect(void) {};

void wizchip_cs_deselect(void) {};

//M20150601 : Rename the function for integrating with W5300

//uint8_t wizchip_bus_readbyte(uint32_t AddrSel) { return * ((volatile uint8_t *) ((ptrdiff_t) AddrSel)); }

iodata_t wizchip_bus_readdata(uint32_t AddrSel) { return * ((volatile iodata_t *) ((ptrdiff_t) AddrSel)); }

//M20150601 : Rename the function for integrating with W5300

//void wizchip_bus_writebyte(uint32_t AddrSel, uint8_t wb) { *((volatile uint8_t*) ((ptrdiff_t)AddrSel)) = wb; }

void wizchip_bus_writedata(uint32_t AddrSel, iodata_t wb) { *((volatile iodata_t*) ((ptrdiff_t)AddrSel)) = wb; }

//uint8_t wizchip_spi_readbyte(void) {return 0;};

uint8_t wizchip_spi_readbyte(void) {
    return 0;
}

//void wizchip_spi_writebyte(uint8_t wb) {};

void wizchip_spi_writebyte(uint8_t wb) {};

//void wizchip_spi_readburst(uint8_t* pBuf, uint16_t len) {};

void wizchip_spi_readburst(uint8_t* pBuf,
uint16_t len) {}

//void wizchip_spi_writeburst(uint8_t* pBuf, uint16_t len) {};
void wizchip_spi_writeburst(uint8_t* pBuf, uint16_t len) {}

//M20150401 : For a compiler didnot support a member of structure
// Replace the assignment of struct members with the assingment of array

/*
_WIZCHIP =
{
    .id = _WIZCHIP_ID_,
    .if_mode = _WIZCHIP_IO_MODE_,
    .CRIS._enter = wizchip_cris_enter,
    .CRIS._exit = wizchip_cris_exit,
    .CS._select = wizchip_cs_select,
    .CS._deselect = wizchip_cs_deselect,
    .IF.BUS._read_byte = wizchip_bus_readbyte,
    .IF.BUS._write_byte = wizchip_bus_writebyte
};
*/
_WIZCHIP WIZCHIP =
{
  _WIZCHIP_IO_MODE_,
  _WIZCHIP_ID_,
  wizchip_cris_enter,
  wizchip_cris_exit,
  wizchip_cs_select,
  wizchip_cs_deselect,
  //M20150601 : Rename the function
  //wizchip_bus_readbyte,
  //wizchip_bus_writebyte
  wizchip_bus_readdata,
  wizchip_bus_writedata,
  // wizchip_spi_readbyte,
  // wizchip_spi_writebyte
};

static uint8_t _DNS_[4]; // DNS server ip address
static dhcp_mode _DHCP_; // DHCP mode

void reg_wizchip_cris_cbfunc(void(*cris_en)(void), void(*cris_ex)(void))
{
  if(!cris_en || !cris_ex)
  {
    WIZCHIP.CRIS._enter = wizchip_cris_enter;
    WIZCHIP.CRIS._exit   = wizchip_cris_exit;
  }
  else
  {
    WIZCHIP.CRIS._enter = cris_en;
    WIZCHIP.CRIS._exit   = cris_ex;
void reg_wizchip_cs_cbfunc(void(*cs_sel)(void), void(*cs_desel)(void))
{
    if(!cs_sel || !cs_desel)
    {
        WIZCHIP.CS._select = wizchip_cs_select;
        WIZCHIP.CS._deselect = wizchip_cs_deselect;
    }
    else
    {
        WIZCHIP.CS._select = cs_sel;
        WIZCHIP.CS._deselect = cs_desel;
    }
}

//M20150515 : For integrating with W5300
void reg_wizchip_bus_cbfunc(uint8_t(*bus_rb)(uint32_t addr), void (*bus_wb)(uint32_t addr, uint8_t wb))
{
    while(!(WIZCHIP.if_mode & _WIZCHIP_IO_MODE_BUS_));
    //M20150601 : Rename call back function for integrating with W5300
    /*
    if(!bus_rb || !bus_wb)
    {
        
    */
WIZCHIP.IF.BUS._read_byte = wizchip_bus_readbyte;
WIZCHIP.IF.BUS._write_byte = wizchip_bus_writebyte;
}
else {
    WIZCHIP.IF.BUS._read_byte = bus_rb;
    WIZCHIP.IF.BUS._write_byte = bus_wb;
}
*/
if(!bus_rb || !bus_wb) {
    WIZCHIP.IF.BUS._read_data = wizchip_bus_readdata;
    WIZCHIP.IF.BUS._write_data = wizchip_bus_writedata;
} else {
    WIZCHIP.IF.BUS._read_data = bus_rb;
    WIZCHIP.IF.BUS._write_data = bus_wb;
}
}

void reg_wizchip_spi_cbfunc(uint8_t (*spi_rb)(void), void (*spi_wb)(uint8_t *wb)) {
    while(!(WIZCHIP.if_mode & _WIZCHIP_IO_MODE_SPI_));
    if(!spi_rb || !spi_wb) {
        WIZCHIP.IF.SPI._read_byte = wizchip_spi_readbyte;
        WIZCHIP.IF.SPI._write_byte = wizchip_spi_writebyte;
    }
```c
} else {
    WIZCHIP.IF.SPI._read_byte = spi_rb;
    WIZCHIP.IF.SPI._write_byte = spi_wb;
}
}

// 20140626 Eric Added for SPI burst operations
void reg_wizchip_spiburst_cbfunc(void (*spi_rb)(uint8_t* pBuf, uint16_t len), void (*spi_wb)(uint8_t* pBuf, uint16_t len)) {
    while(!(WIZCHIP.IF._mode & _WIZCHIP_IO_MODE_SPI_));
    if(!spi_rb || !spi_wb) {
        WIZCHIP.IF.SPI._read_burst = wizchip_spi_readburst;
        WIZCHIP.IF.SPI._write_burst = wizchip_spi_writeburst;
    } else {
        WIZCHIP.IF.SPI._read_burst = spi_rb;
        WIZCHIP.IF.SPI._write_burst = spi_wb;
    }
}

int8_t ctlwizchip(ctlwizchip_type cwtype, void* arg) {
    #if _WIZCHIP_ == 5200 || _WIZCHIP_ == 5500
        uint8_t tmp = 0;
    #endif
```
uint8_t* ptmp[2] = {0, 0};

switch(cwtype)
{
    case CW_RESET_WIZCHIP:
        wizchip_sw_reset();
        break;
    case CW_INIT_WIZCHIP:
        if(arg != 0)
        {
            ptmp[0] = (uint8_t*)arg;
            ptmp[1] = ptmp[0] + _WIZCHIP_SOCK_NUM_;
        }
        return wizchip_init(ptmp[0], ptmp[1]);
    case CW_CLR_INTERRUPT:
        wizchip_clrinterrupt (*((intr_kind*)arg));
        break;
    case CW_GET_INTERRUPT:
        *((intr_kind*)arg) = wizchip_getinterrupt();
        break;
    case CW_SET_INTRMASK:
        wizchip_setinterruptmask (*((intr_kind*)arg));
        break;
    case CW_GET_INTRMASK:
        *((intr_kind*)arg) = wizchip_getinterruptmask();
        break;
    //M20150601 : This can be supported by W5200, W5500
    //#if _WIZCHIP_ > 5100
    #if (_WIZCHIP_ == 5200 || _WIZCHIP_ == 5500)
    case CW_SET_INTRTIME:
setINTLEVEL(*(uint16_t*)arg);
break;
case CW_GET_INTRTIME:
    *(uint16_t*)arg = getINTLEVEL();
break;
#endif

case CW_GET_ID:
    ((uint8_t*)arg)[0] = WIZCHIP.id[0];
    ((uint8_t*)arg)[1] = WIZCHIP.id[1];
    ((uint8_t*)arg)[2] = WIZCHIP.id[2];
    ((uint8_t*)arg)[3] = WIZCHIP.id[3];
    ((uint8_t*)arg)[4] = WIZCHIP.id[4];
    ((uint8_t*)arg)[5] = 0;
break;
#if _WIZCHIP_ == 5500
    case CW_RESET_PHY:
        wizphy_reset();
    break;
    case CW_SET_PHYCONF:
        wizphy_setphyconf((wiz_PhyConf*)arg);
    break;
    case CW_GET_PHYCONF:
        wizphy_getphyconf((wiz_PhyConf*)arg);
    break;
    case CW_GET_PHYSTATUS:
        break;
    case CW_SET_PHYPOWMODE:
        return wizphy_setphympmode(*(uint8_t*)arg);
#endif
#if _WIZCHIP_ == 5200 || _WIZCHIP_ == 5500
    case CW_GET_PHYPOWMODE:
        tmp = wizphy_getphympmode();
        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;

int8_t ctlnetwork(ctlnetwork_type cntype, void* arg)
{
    switch(cntype)
    {
    case CN_SET_NETINFO:
        wizchip_setnetinfo((wiz_NetInfo*)arg);
        break;
    case CN_GET_NETINFO:
        wizchip_getnetinfo((wiz_NetInfo*)arg);
        break;
    case CN_SET_NETMODE:
        return wizchip_setnetmode(*((netmode_type*)arg));
    case CN_GET_NETMODE:
        *(netmode_type*)arg = wizchip_getnetmode();
        break;
    case CN_SET_TIMEOUT:

wizchip_settimeout((wiz_NetTimeout*)arg);

break;

wizchip_gettimeout((wiz_NetTimeout*)arg);

break;

default:
    return -1;

}

return 0;

void wizchip_sw_reset(void)
{
    uint8_t gw[4], sn[4], sip[4];
    uint8_t mac[6];
    //A20150601
    #if _WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_
        uint16_t mr = (uint16_t)getMR();
        setMR(mr | MR_IND);
    #endif
    //
    getSHAR(mac);
    getGAR(gw); getSUBR(sn); getSIPR(sip);
    setMR(MR_RST);
    getMR(); // for delay
    //A2015051 : For indirect bus mode
    #if _WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_
        setMR(mr | MR_IND);
    #endif
    //
    setSHAR(mac);
    setGAR(gw);
    setSUBR(sn);
```c
int8_t wizchip_init(uint8_t* txsize, uint8_t* rxsize)
{
    int8_t i;
    int8_t tmp = 0;
    wizchip_sw_reset();
    if(txsize)
    {
        tmp = 0;
        //M20150601 : For integrating with W5300
        #if _WIZCHIP_ == 5300
            for(i = 0 ; i < _WIZCHIP_SOCK_NUM_; i++)
            {
                if(txsize[i] >= 64) return -1;
                //No use 64KB even if W5300 support max 64KB memory allocation
                tmp += txsize[i];
                if(tmp > 128) return -1;
            }
        #endif
        for(i = 0 ; i < _WIZCHIP_SOCK_NUM_; i++)
        {
            tmp += txsize[i];
            if(tmp > 16) return -1;
        }
    }
    else  //else
    for(i = 0 ; i < _WIZCHIP_SOCK_NUM_; i++)
    {
        if(rxsize[i] >= 64) return -1;
        tmp += txsize[i];
        if(tmp > 16) return -1;
    }
    setSn_TXBUF_SIZE(i, txsize[i]);
    if(rxsize)
    {  
```
```c
{ tmp = 0;
#if _WIZCHIP_ == 5300
   for(i = 0 ; i < _WIZCHIP_SOCK_NUM_; i++)
   {
      if(rxsize[i] >= 64) return -1;
      //No use 64KB even if W5300 support max 64KB memory allocation
      tmp += rxsize[i];
      if(tmp > 128) return -1;
   }
#endif
#if _WIZCHIP_ < 5500
   ir |= (1<<4); // IK_WOL
#endif
   if(tmp % 8) return -1;
#endif
   for(i = 0 ; i < _WIZCHIP_SOCK_NUM_; i++)
   {
      tmp += rxsize[i];
      if(tmp > 16) return -1;
   }
#endif
   return 0;
}
```

```c
void wizchip_clrinterrupt(intr_kind intr)
{
   uint8_t ir = (uint8_t)intr;
   uint8_t sir = (uint8_t)((uint16_t)intr >> 8);
#if _WIZCHIP_ < 5500
   ir |= (1<<4); // IK_WOL
#endif
   setSn_RXBUF_SIZE(i, rxsize[i]);
}
```
//A20150601 : For integrating with W5300

//Wizchip get interrupt functions

intr_kind wizchip_getinterrupt(void)
{
    uint8_t ir = 0;
    uint8_t sir = 0;
    uint16_t ret = 0;
    //A20150601 : For integrating with W5300
    //Wizchip get interrupt functions

    if (_WIZCHIP_ == 5100)
    {
        ir = getIR();
        sir = ir & 0x0F;
    }
    //A20150601 : For integrating with W5300
    //Wizchip get interrupt functions

    else
    {
        ir = getIR();
        sir = getSIR();
    }
}
#endif

//M20150601 : For Integrating with W5300
#elif _WIZCHIP_ == 5100
    simr &= 0x0F;
    imr |= simr;
    setIMR(imr);
#endif

//A20150601 : For integrating with W5300
#else
    setIMR ( (((uint16_t)imr) << 8) | (((uint16_t)simr) & 0x00FF) );
#endif

//WIZCHIP_ == 5100
    simr &= 0x0F;
    imr |= simr;
    setIMR(imr);
#endif

//A20150601 : For integrating with W5300
#elif _WIZCHIP_ == 5300
    setIMR ( (((uint16_t)imr) << 8) | (((uint16_t)simr) & 0x00FF) );
#endif

//else
setIMR(imr);
setSIMR(simr);
#endif
}

intr_kind wizchip_getinterruptmask(void)
{
uint8_t imr = 0;
uint8_t simr = 0;
uint16_t ret = 0;
#if _WIZCHIP_ == 5100
imr = getIMR();
simr = imr & 0x0F;
#else
#endif
	
#if _WIZCHIP_ == 5300
ret = getIMR();
imr = (uint8_t)(ret >> 8);
simr = (uint8_t)ret;
#else
#endif
	
#if _WIZCHIP_ < 5500
imr &= ~(1<<4); // IK_WOL
#endif
#if _WIZCHIP_ == 5200
imr &= ~(1 << 6); // IK_DEST_UNREACH
#endif
ret = simr;
ret = (ret << 8) + imr;
return (intr_kind)ret;
}

int8_t wizphy_getphylink(void)
{
int8_t tmp;
#if _WIZCHIP_ == 5200
    if(getPHYSTATUS() & PHYSTATUS_LINK)
        tmp = PHY_LINK_ON;
    else
        tmp = PHY_LINK_OFF;
#elif _WIZCHIP_ == 5500
    if(getPHYCFGR() & PHYCFGR_LNK_ON)
        tmp = PHY_LINK_ON;
    else
        tmp = PHY_LINK_OFF;
#else
    tmp = -1;
#endif
return tmp;

#if _WIZCHIP_ > 5100

int8_t wizphy_getphypmode(void)
{
    int8_t tmp = 0;
#if _WIZCHIP_ == 5200
    if(getPHYSTATUS() & PHYSTATUS_POWERDOWN)
        tmp = PHY_POWER_DOWN;
#else
    tmp = PHY_POWER_NORM;
#endif
#if _WIZCHIP_ == 5500
    if(getPHYCFGR() & PHYCFGR_OPMDC_PDOWN)
        tmp = PHY_POWER_DOWN;
#else
    tmp = PHY_POWER_NORM;
#endif
    return tmp;
}
#endif

#if _WIZCHIP_ == 5500

void wizphy_reset(void)
{
    uint8_t tmp = getPHYCFGR();
    tmp &= PHYCFGR_RST;
    setPHYCFGR(tmp);
    tmp = getPHYCFGR();
    tmp |= ~PHYCFGR_RST;
    setPHYCFGR(tmp);
}

void wizphy_setphyconf(wiz_PhyConf* phyconf)
{
    uint8_t tmp = 0;
    if(phyconf->by == PHY_CONFBY_SW)
        tmp |= PHYCFGR_OPMD;
    else
        tmp &= ~PHYCFGR_OPMD;
    if(phyconf->mode == PHY_MODE_AUTONEGO)
        tmp |= PHYCFGR_OPMDC_ALLA;
    else
    {
        if(phyconf->duplex == PHY_DUPLEX_FULL)
        {
            if(phyconf->speed == PHY_SPEED_100)
                tmp |= PHYCFGR_OPMDC_100F;
            else
                tmp |= PHYCFGR_OPMDC_10F;
        }
        else
        {
            if(phyconf->speed == PHY_SPEED_100)
                tmp |= PHYCFGR_OPMDC_100H;
            else
                tmp |= PHYCFGR_OPMDC_10H;
    
}
void wizphy_getphyconf(wiz_PhyConf* phyconf) {
    uint8_t tmp = 0;
    tmp = getPHYCFGR();
    phyconf->by = (tmp & PHYCFGR_OPMD) ? PHY_CONFBY_SW : PHY_CONFBY_HW;
    switch(tmp & PHYCFGR_OPMDC_ALLA) {
        case PHYCFGR_OPMDC_ALLA:
            phyconf->mode = PHY_MODE_AUTONEGO;
            break;
        case PHYCFGR_OPMDC_100FA:
            phyconf->mode = PHY_MODE_AUTONEGO;
            break;
        case PHYCFGR_OPMDC_100F:
            phyconf->mode = PHY_MODE_AUTONEGO;
            break;
        default:
            phyconf->mode = PHY_MODE_MANUAL;
            break;
    }
    switch(tmp & PHYCFGR_OPMDC_ALLA) {
        case PHYCFGR_OPMDC_100FA:
            phyconf->speed = PHY_SPEED_100;
            break;
        case PHYCFGR_OPMDC_100F:
            phyconf->speed = PHY_SPEED_100;
            break;
        case PHYCFGR_OPMDC_100H:
            phyconf->speed = PHY_SPEED_100;
            break;
        default:
            phyconf->speed = PHY_SPEED_10;
            break;
    }
    switch(tmp & PHYCFGR_OPMDC_ALLA) {
        case PHYCFGR_OPMDC_100FA:
            phyconf->speed = PHY_SPEED_100;
            break;
    }
    switch(tmp & PHYCFGR_OPMDC_ALLA) {
        case PHYCFGR_OPMDC_100F:
            phyconf->speed = PHY_SPEED_100;
            break;
    }
}
case PHYCFGR_OPMDC_10F:
    phyconf->duplex = PHY_DUPLEX_FULL;
    break;
default:
    phyconf->duplex = PHY_DUPLEX_HALF;
    break;
}

void wizphy_getphystat(wiz_PhyConf* phyconf) {
    uint8_t tmp = getPHYCFGR();
    phyconf->duplex = (tmp & PHYCFGR_DPX_FULL) ? PHY_DUPLEX_FULL : PHY_DUPLEX_HALF;
    phyconf->speed = (tmp & PHYCFGR_SPD_100) ? PHY_SPEED_100 : PHY_SPEED_10;
}

int8_t wizphy_setphypmode(uint8_t pmode) {
    uint8_t tmp = 0;
    tmp = getPHYCFGR();
    if((tmp & PHYCFGR_OPMD)== 0) return -1;
    tmp &= ~PHYCFGR_OPMDC_ALLA;
    if( pmode == PHY_POWER_DOWN)
        tmp |= PHYCFGR_OPMDC_PDOWN;
    else
        tmp |= PHYCFGR_OPMDC_ALLA;
    setPHYCFGR(tmp);
    wizphy_reset();
    tmp = getPHYCFGR();
    if( pmode == PHY_POWER_DOWN)
    {
        if(tmp & PHYCFGR_OPMDC_PDOWN) return 0;
    }
}
else
{
    if(tmp & PHYCFGR_OPMDC_ALLA) return 0;
}
return -1;
#endif

void wizchip_setnetinfo(wiz_NetInfo* pnetinfo)
{
    setSHAR(pnetinfo->mac);
    setGAR(pnetinfo->gw);
    setSUBR(pnetinfo->sn);
    setSIPR(pnetinfo->ip);
    _DNS_[0] = pnetinfo->dns[0];
    _DNS_[1] = pnetinfo->dns[1];
    _DNS_[2] = pnetinfo->dns[2];
    _DNS_[3] = pnetinfo->dns[3];
    _DHCP_ = pnetinfo->dhcp;
}

void wizchip_getnetinfo(wiz_NetInfo* pnetinfo)
{
    getSHAR(pnetinfo->mac);
    getGAR(pnetinfo->gw);
    getSUBR(pnetinfo->sn);
    getSIPR(pnetinfo->ip);
    pnetinfo->dns[0] = _DNS_[0];
    pnetinfo->dns[1] = _DNS_[1];
    pnetinfo->dns[2] = _DNS_[2];
    pnetinfo->dns[3] = _DNS_[3];
    pnetinfo->dhcp = _DHCP_;
```c
int8_t wizchip_setnetmode(netmode_type netmode)
{
    uint8_t tmp = 0;
    #if __WIZCHIP__ != 5500
        if((netmode & ~(NM_WAKEONLAN | NM_PPPOE | NM_PINGBLOCK)) return -1;
    #else
        if((netmode & ~(NM_WAKEONLAN | NM_PPPOE | NM_PINGBLOCK | NM_FORCEARP)) return -1;
    #endif
    tmp = getMR();
tmp |= (uint8_t)netmode;
setMR(tmp);
return 0;
}

netmode_type wizchip_getnetmode(void)
{
    return (netmode_type) getMR();
}

void wizchip_settimeout(wiz_NetTimeout* nettime)
{
    setRCR((nettime->retry_cnt);
setRTR((nettime->time_100us);
}

void wizchip_gettimeout(wiz_NetTimeout* nettime)
{
    nettime->retry_cnt = getRCR();
    nettime->time_100us = getRTR();
}
```
Socket APIs

wizchip_conf.h

Go to the documentation of this file.

```
1  //**************************************************************************************
2  //******************************************************************************
3  //
4  // 43  //******************************************************************************
5  //**************************************************************************************

64  #define _WIZCHIP_ 0x5500

65  // 5100, 5200, 5300, 5500
```

```
66  #endif

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
```
// Add to

//

#define _WIZCHIP_IO_MODE_BUS_DIR_  (_WIZCHIP_IO_MODE_BUS_ + 1)
#define _WIZCHIP_IO_MODE_BUS_INDIR_  (_WIZCHIP_IO_MODE_BUS_ + 2)
#define _WIZCHIP_IO_MODE_SPI_VDM_  (_WIZCHIP_IO_MODE_SPI_ + 1)
#define _WIZCHIP_IO_MODE_SPI_FDM_  (_WIZCHIP_IO_MODE_SPI_ + 2)

#if (_WIZCHIP_ == 5100)
#define _WIZCHIP_ID_ "W5100\0"
#endif

// #define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_BUS_DIR_
// #define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_BUS_INDIR_
#define _WIZCHIP_IO_MODE_SPI_ _WIZCHIP_IO_MODE_SPI_

//A20150601 : Define the unit of IO DATA.
typedef uint8_t iodata_t;

//A20150401 : Indclude W5100.h file
#include "W5100/w5100.h"
#endif

#elif (_WIZCHIP_ == 5200)
#define _WIZCHIP_ID_ "W5200\0"
#define _WIZCHIP_IO_MODE.Bus_DIR_ _WIZCHIP_IO_MODE_BUS_DIR_
#define _WIZCHIP_IO_MODE_BUS_INDIR_
#define _WIZCHIP_IO_MODE_SPI_ _WIZCHIP_IO_MODE_SPI_
#define _WIZCHIP_IO_MODE_SPI_FDM_
#endif
//A20150601 : Define the unit of IO DATA.
typedef uint8_t iodata_t;
#include "W5200/w5200.h"

#elif (_WIZCHIP_ == 5500)
#define _WIZCHIP_ID_ "W5500\0"

#ifndef _WIZCHIP_IO_MODE_
#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_SPI_FDM_
#endif

//A20150601 : Define the unit of IO DATA.
typedef uint8_t iodata_t;
#include "W5500/w5500.h"

#elif (_WIZCHIP_ == 5300)
#define _WIZCHIP_ID_ "W5300\0"

#ifndef _WIZCHIP_IO_MODE_
#define _WIZCHIP_IO_MODE_ _WIZCHIP_IO_MODE_BUS_DIR_
#endif

//A20150601 : Define the unit and bus width of IO DATA.
#ifndef _WIZCHIP_IO_BUS_WIDTH_
#define _WIZCHIP_IO_BUS_WIDTH_ 8 // 16
#endif

#if _WIZCHIP_IO_BUS_WIDTH_ == 8
typedef uint8_t iodata_t;
#elif _WIZCHIP_IO_BUS_WIDTH_ == 16
typedef uint16_t iodata_t;
#endif
#else
#error "Unknown _WIZCHIP_IO_BUS_WIDTH_. It should be 8 or 16."
#endif

//
#include "W5300/w5300.h"
#else
#error "Unknown defined _WIZCHIP_. You should define one of 5100, 5200, and 5500 !!!"
#endif

#ifndef _WIZCHIP_IO_MODE_
#error "Undefined _WIZCHIP_IO_MODE_. You should define it !!!"
#endif

#ifndef _WIZCHIP_IO_BASE_
#error "You should be define _WIZCHIP_IO_BASE to fit your system memory map."
#endif
#else
#endif

#ifndef _WIZCHIP_IO_MODE_BASE_
#define _WIZCHIP_IO_BASE_ 0x00000000 // 0x8000
#endif

//M20150401 : Typing Error
#ifndef _WIZCHIP_IO_MODE_BASE_
#define _WIZCHIP_IO_BASE_ _WIZCHIP_IO_MODE_BASE_
#endif
#endif

#ifndef _WIZCHIP_IO_MODE_BASE_
#error "You should be define _WIZCHIP_IO_BASE to fit your system memory map."
#endif
#else
#endif

#define _WIZCHIP_SOCK_NUM_ 8
#else
#define _WIZCHIP_SOCK_NUM_ 4
typedef struct __WIZCHIP
{
    uint16_t if_mode;
    uint8_t id[6];
}

struct _CRIS
{
    void (*_enter) (void);
    void (*_exit) (void);
} CRIS;

struct _CS
{
    void (*_select) (void);
    void (*_deselect)(void);
} CS;

union _IF
{
    //M20156501 : Modify the function name for integrating with W5300
    //struct
    //{
    //    uint8_t (*_read_byte) (uint32_t AddrSel);
    //    void (*_write_byte) (uint32_t AddrSel, uint8_t wb);
    //}BUS;
    struct
    {

237 | iodata_t (*_read_data) (uint32_t AddrSel);
238 | void (*_write_data) (uint32_t AddrSel, iodata_t wb);
239 | } BUS;
240 |
244 | struct {
245 |   uint8_t (*_read_byte) (void);
246 |   void (*_write_byte) (uint8_t wb);
247 |   void (*_read_burst) (uint8_t* pBuf, uint16_t len);
248 |   void (*_write_burst) (uint8_t* pBuf, uint16_t len);
249 | } SPI;
250 |
251 | // To be added
252 | //
253 | } IF;
254 | } _WIZCHIP;
255 |
256 | extern _WIZCHIP WIZCHIP;
257 |
258 | typedef enum {
259 |   CW_RESET_WIZCHIP,
260 |   CW_INIT_WIZCHIP,
261 |   CW_GET_INTERRUPT,
262 |   CW_CLR_INTERRUPT,
263 |   CW_SET_INTRMASK,
264 |   CW_GET_INTRMASK,
265 |   CW_SET_INTRTIME,
266 |   CW_GET_INTRTIME,
267 |   CW_GET_ID,
268 |   //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
283      application code
284  //if _WIZCHIP_ == 5200 || _WIZCHIP_ == 5500
285      CW_GET_PHYPOWMODE,
286      CW_GET_PHYLINK
287  //endif
288 }ctlwizchip_type;
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;
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      IK_PPPOE_TERMINATED    = (1 << 5),
317      #if  _WIZCHIP_ != 5200
318      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)

#define PHY_CONFBY_HW 0
#define PHY_CONFBY_SW 1
#define PHY_MODE_MANUAL 0
#define PHY_MODE_AUTONEGO 1
#define PHY_SPEED_10 0
#define PHY_SPEED_100 1
#define PHY_DUPLEX_HALF 0
#define PHY_DUPLEX_FULL 1
#define PHY_LINK_OFF 0
#define PHY_LINK_ON 1
#define PHY_POWER_NORM 0
#define PHY_POWER_DOWN 1

#if _WIZCHIP_ == 5500
typedef struct wiz_PhyConf_t {
    uint8_t by;
    uint8_t mode;
    uint8_t speed;
    uint8_t duplex;
    //uint8_t power;  ///< set by @ref PHY_POWER_NORM or @ref PHY_POWER_DOWN
    //uint8_t link;   ///< Valid only in CW_GET_PHYSTATUS. set by @ref PHY_LINK_ON or PHY_DUPLEX_OFF
} wiz_PhyConf;

typedef enum {
    NETINFO_STATIC = 1,
    NETINFO_DHCP
} dhcp_mode;

typedef struct wiz_NetInfo_t {
    uint8_t mac[6];
    uint8_t ip[4];
    uint8_t sn[4];
    uint8_t gw[4];
    uint8_t dns[4];
    dhcp_mode dhcp;
} wiz_NetInfo;

typedef enum {
    #if  _WIZCHIP_  ==  5500
    NM_FORCEARP   =  (1<<1),
    #endif
    NM_WAKEONLAN  =  (1<<5),
}
NM_PINGBLOCK  = (1<<4),
NM_PPPOE       = (1<<3),
}

typedef struct wiz_NetTimeout_t {
    uint8_t  retry_cnt;
    uint16_t time_100us;
} wiz_NetTimeout;

void reg_wizchip_cris_cbfunc(void(*cris_en)(void), void(*cris_ex)(void));

void reg_wizchip_cs_cbfunc(void(*cs_sel)(void), void(*cs_desel)(void));

//M20150601 : For integrating with W5300
//void reg_wizchip_bus_cbfunc(uint8_t (*bus_rb)(uint32_t addr), void (*bus_wb)(uint32_t addr, uint8_t wb));
void reg_wizchip_bus_cbfunc(iodata_t (*bus_rb)(uint32_t addr), void (*bus_wb)(uint32_t addr, iodata_t wb));

void reg_wizchip_spi_cbfunc(uint8_t (*spi_rb)(void), void (*spi_wb)(uint8_t wb));

void reg_wizchip_spiburst_cbfunc(void (*spi_rb)(uint8_t* pBuf, uint16_t len), void (*spi_wb)(uint8_t* pBuf, uint16_t len));

int8_t ctlwizchip(ctlwizchip_type cwtype, void* arg);

int8_t ctlnetwork(ctlnetwork_type cntype, void* arg);
The following functions are implemented for internal use. but You can call these functions for code size reduction instead of ctlwizchip() and ctlnetwork().

```c
void wizchip_sw_reset(void);
int8_t wizchip_init(uint8_t* txsize,
                     uint8_t* rxsize);
void wizchip_clrinterrupt(intr_kind intr);
intr_kind wizchip_getinterrupt(void);
void wizchip_setinterruptmask(intr_kind intr);
intr_kind wizchip_getinterruptmask(void);
#if _WIZCHIP_ > 5100
    int8_t wizphy_getphylink(void);
    int8_t wizphy_getphypmode(void);
#endif
#if _WIZCHIP_ == 5500
    void wizphy_reset(void);
    void wizphy_setphyconf(wiz_PhyConf* phyconf);
    void wizphy_getphyconf(wiz_PhyConf* phyconf);
    void wizphy_getphystat(wiz_PhyConf* phyconf);
```
int8_t wizphy_setphypmode(uint8_t pmode);

#define

void wizchip_setnetinfo(wiz_NetInfo* pnetinfo);

void wizchip_getnetinfo(wiz_NetInfo* pnetinfo);

int8_t wizchip_setnetmode(netmode_type netmode);

netmode_type wizchip_getnetmode(void);

void wizchip_settimeout(wiz_NetTimeout* nettime);

void wizchip_gettimeout(wiz_NetTimeout* nettime);

#endif // _WIZCHIP_CONF_H_

Generated on Wed May 4 2016 16:43:58 for Socket APIs by doxygen 1.8.9.1
Socket APIs

w5500.h

Go to the documentation of this file.

```c
1  /**************************************************************************************************
2  //**********************************************************************************************
3  //
4  //
5  //**************************************************************************************************
6  //******************************************************************************
7  //
8  //
9  //**************************************************************************************************
10 #ifndef _W5500_H_
11 #define _W5500_H_
12
13 #include <stdint.h>
14 #include "wizchip_conf.h"
15
16 #if (_WIZCHIP_ == 5500)
17
18 #define _W5500_IO_BASE_ 0x00000000
19
20 #define _W5500_SPI_READ_ (0x00 << 2) // SPI interface Read operation in Control Phase
21 #define _W5500_SPI_WRITE_ (0x01 << 2) // SPI interface Write operation in Control Phase
```
#define WIZCHIP_CREG_BLOCK 0x00  // Common register block

#define WIZCHIP_SREG_BLOCK(N) (1+4*N)  // Socket N register block

#define WIZCHIP_TXBUF_BLOCK(N) (2+4*N)  // Socket N Tx buffer address block

#define WIZCHIP_RXBUF_BLOCK(N) (3+4*N)  // Socket N Rx buffer address block

#define WIZCHIP_OFFSET_INC(ADDR, N) (ADDR + (N<<8)) // Increase offset address

// Definition For Legacy Chip Driver //

#define IINCHIP_READ(ADDR) WIZCHIP_READ(ADDR)
#define IINCHIP_WRITE(ADDR, VAL) WIZCHIP_WRITE(ADDR, VAL)
#define IINCHIP_READ_BUF(ADDR, BUF, LEN) WIZCHIP_READ_BUF(ADDR, BUF, LEN)
#define IINCHIP_WRITE_BUF(ADDR, BUF, LEN) WIZCHIP_WRITE(ADDR, BUF, LEN)

#define MR (_W5500_IO_BASE_ + (0x0000 << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define GAR (_W5500_IO_BASE_ + (0x0001 << 8) + (WIZCHIP_CREG_BLOCK << 3))
```c
// M20150401 : Rename SYMBOL (Re-define error in a compile)
#define SUBR (_W5500_IO_BASE_ + (0x0005 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define SHAR (_W5500_IO_BASE_ + (0x0009 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define SIPR (_W5500_IO_BASE_ + (0x000F << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define INTLEVEL (_W5500_IO_BASE_ + (0x0013 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define IR (_W5500_IO_BASE_ + (0x0015 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define IMR (_W5500_IO_BASE_ + (0x0016 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define_SIR(_W5500_IO_BASE_ + (0x0017 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define SIMR (_W5500_IO_BASE_ + (0x0018 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define RTR (_W5500_IO_BASE_ + (0x0019 << 8) + (WIZCHIP_CREG_BLOCK << 3))
#define_RTR_(_W5500_IO_BASE_ + (0x0019 << 8) + (WIZCHIP_CREG_BLOCK << 3))
```
// M20150401 : Rename SYMBOE (Re-define error in a compile)

#define RCR (_W5500_IO_BASE_ + (0x001B << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define _RCR_ (_W5500_IO_BASE_ + (0x001B << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PTIMER (_W5500_IO_BASE_ + (0x001C << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PMAGIC (_W5500_IO_BASE_ + (0x001D << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PHAR (_W5500_IO_BASE_ + (0x001E << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PSID (_W5500_IO_BASE_ + (0x0024 << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PMRU (_W5500_IO_BASE_ + (0x0026 << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define UIPR (_W5500_IO_BASE_ + (0x0028 << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define UPORTR (_W5500_IO_BASE_ + (0x002C << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define PHYCFGR (_W5500_IO_BASE_ + (0x002E << 8) + (WIZCHIP_CREG_BLOCK << 3))

#define 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) <<
```c
#define Sn_CR(N) (_W5500_IO_BASE_ + (0x0001 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_IR(N) (_W5500_IO_BASE_ + (0x0002 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_SR(N) (_W5500_IO_BASE_ + (0x0003 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_PORT(N) (_W5500_IO_BASE_ + (0x0004 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_DHAR(N) (_W5500_IO_BASE_ + (0x0006 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_DIPR(N) (_W5500_IO_BASE_ + (0x000C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_DPORT(N) (_W5500_IO_BASE_ + (0x0010 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_MSSR(N) (_W5500_IO_BASE_ + (0x0012 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

// Reserved
(_W5500_IO_BASE_ + (0x0014 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
```
```c
#define Sn_TOS(N) (_W5500_IO_BASE_ + (0x0015 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_TTL(N) (_W5500_IO_BASE_ + (0x0016 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

// Reserved
(_W5500_IO_BASE_ + (0x0017 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
(_W5500_IO_BASE_ + (0x0018 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
(_W5500_IO_BASE_ + (0x0019 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
(_W5500_IO_BASE_ + (0x001A << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
(_W5500_IO_BASE_ + (0x001B << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
(_W5500_IO_BASE_ + (0x001C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))

#define Sn_RXBUF_SIZE(N) (_W5500_IO_BASE_ + (0x001E << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_TXBUF_SIZE(N) (_W5500_IO_BASE_ + (0x001F << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
```
#define Sn_TX_FSR(N)     (_W5500_IO_BASE_ + (0x0020 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_TX_RD(N)      (_W5500_IO_BASE_ + (0x0022 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_TX_WR(N)      (_W5500_IO_BASE_ + (0x0024 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_RX_RSR(N)     (_W5500_IO_BASE_ + (0x0026 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_RX_RD(N)      (_W5500_IO_BASE_ + (0x0028 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_RX_WR(N)      (_W5500_IO_BASE_ + (0x002A << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_IMR(N)        (_W5500_IO_BASE_ + (0x002C << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_FRAG(N)       (_W5500_IO_BASE_ + (0x002E << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
#define Sn_KPALVTR(N)    (_W5500_IO_BASE_ + (0x0030 << 8) + (WIZCHIP_SREG_BLOCK(N) << 3))
//#define Sn_TSR(N)
(_W5500_IO_BASE_ + (0x0030 << 8) +
(WIZCHIP_SREG_BLOCK(N) << 3))

//------------------------------- W5500
Register values -------------------------------

/* MODE register values */
#define MR_RST 0x80
#define MR_WOL 0x20
#define MR_PB 0x10
#define MR_PPPOE 0x08
#define MR_FARP 0x02

/* IR register values */
#define IR_CONFLICT 0x80
#define IR_UNREACH 0x40
#define IR_PPPOE 0x20
#define IR_MP 0x10

/* PHYCFGR register value */
#define PHYCFGR_RST ~(1<<7)
    // For PHY reset, must operate AND mask.
#define PHYCFGR_OPMD (1<<6)
    // Configure PHY with OPMDC value
#define PHYCFGR_OPMDC_ALLA (7<<3)
#define PHYCFGR_OPMDC_PDOWN (6<<3)
#define PHYCFGR_OPMDC_NA (5<<3)
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>767</td>
<td>#define PHYCFGR_OPMDC_100FA (4&lt;&lt;3)</td>
<td></td>
</tr>
<tr>
<td>768</td>
<td>#define PHYCFGR_OPMDC_100F (3&lt;&lt;3)</td>
<td></td>
</tr>
<tr>
<td>769</td>
<td>#define PHYCFGR_OPMDC_100H (2&lt;&lt;3)</td>
<td></td>
</tr>
<tr>
<td>770</td>
<td>#define PHYCFGR_OPMDC_10F (1&lt;&lt;3)</td>
<td></td>
</tr>
<tr>
<td>771</td>
<td>#define PHYCFGR_OPMDC_10H (0&lt;&lt;3)</td>
<td></td>
</tr>
<tr>
<td>772</td>
<td>#define PHYCFGR_DPX_FULL (1&lt;&lt;2)</td>
<td></td>
</tr>
<tr>
<td>773</td>
<td>#define PHYCFGR_DPX_HALF (0&lt;&lt;2)</td>
<td></td>
</tr>
<tr>
<td>774</td>
<td>#define PHYCFGR_SPD_100 (1&lt;&lt;1)</td>
<td></td>
</tr>
<tr>
<td>775</td>
<td>#define PHYCFGR_SPD_10 (0&lt;&lt;1)</td>
<td></td>
</tr>
<tr>
<td>776</td>
<td>#define PHYCFGR_LNK_ON (1&lt;&lt;0)</td>
<td></td>
</tr>
<tr>
<td>777</td>
<td>#define PHYCFGR_LNK_OFF (0&lt;&lt;0)</td>
<td></td>
</tr>
<tr>
<td>778</td>
<td>/* IMR register values */</td>
<td></td>
</tr>
<tr>
<td>779</td>
<td>#define IM_IR7 0x80</td>
<td></td>
</tr>
<tr>
<td>780</td>
<td>#define IM_IR6 0x40</td>
<td></td>
</tr>
<tr>
<td>781</td>
<td>#define IM_IR5 0x20</td>
<td></td>
</tr>
<tr>
<td>782</td>
<td>#define IM_IR4 0x10</td>
<td></td>
</tr>
<tr>
<td>783</td>
<td>/* Sn_MR Default values */</td>
<td></td>
</tr>
<tr>
<td>784</td>
<td>#define Sn_MR_MULTI 0x80</td>
<td></td>
</tr>
<tr>
<td>785</td>
<td>#define Sn_MR_BCASTB 0x40</td>
<td></td>
</tr>
<tr>
<td>786</td>
<td>#define Sn_MR_ND 0x20</td>
<td></td>
</tr>
<tr>
<td>787</td>
<td>#define Sn_MR_UCASTB 0x10</td>
<td></td>
</tr>
<tr>
<td>788</td>
<td>#define Sn_MR_MACRAW 0x04</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td>//define Sn_MR_IPRAW 0x03</td>
<td></td>
</tr>
<tr>
<td>790</td>
<td>/**&lt; IP LAYER RAW SOCK */</td>
<td></td>
</tr>
<tr>
<td>791</td>
<td>#define Sn_MR_IPRAW /*&lt; IP LAYER RAW SOCK */</td>
<td></td>
</tr>
<tr>
<td>792</td>
<td>#define Sn_MR_UDP 0x02</td>
<td></td>
</tr>
</tbody>
</table>
#define Sn_MR_TCP        0x01
#define Sn_MR_CLOSE       0x00

/* Sn_MR values used with Sn_MR_MACRAW */
#define Sn_MR_MFEN
   Sn_MR_MULTI
#define Sn_MR_MMB
   Sn_MR_ND

#define Sn_MR_MIP6B
   Sn_MR_UCASTB

/* Sn_MR value used with Sn_MR_UDP & Sn_MR_MULTI */
#define Sn_MR_MC
   Sn_MR_ND

/* Sn_MR alternate values */
#define SOCK_STREAM
   Sn_MR_TCP
#define SOCK_DGRAM
   Sn_MR_UDP

/* Sn_CR values */
#define Sn_CR_OPEN
   0x01
#define Sn_CR_LISTEN
   0x02
#define Sn_CR_CONNECT
   0x04
#define Sn_CR_DISCON
   0x08
#define Sn_CR_CLOSE
   0x10
```c
#define Sn_CR_SEND_MAC 0x21
#define Sn_CR_SEND_KEEP 0x22
#define Sn_CR_RECV 0x40

#include "Sn_IR_values"
#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

#include "Sn_SR_values"
#define SOCK_CLOSED 0x00
#define SOCK_INIT 0x13
#define SOCK_LISTEN 0x14
#define SOCK_SYNSENTR 0x15
#define SOCK_SYNRECV 0x16
#define SOCK_ESTABLISHED 0x17
#define SOCK_FIN_WAIT 0x18
#define SOCK_CLOSING 0x1A
```
```c
#define SOCK_TIME_WAIT      0x1B
#define SOCK_CLOSE_WAIT     0x1C
#define SOCK_LAST_ACK       0x1D
#define SOCK_UDP           0x22
// #define SOCK_IPRAW          0x32
// * IP raw mode socket */
#define SOCK_MACRAW         0x42
// #define SOCK_PPPOE          0x5F
/* IP PROTOCOL */
#define IPPROTO_IP          0
   // Dummy for IP
#define IPPROTO_ICMP         1
   // Control message protocol
#define IPPROTO_IGMP         2
   // Internet group management protocol
#define IPPROTO_GGP          3
   // Gateway^2 (deprecated)
#define IPPROTO_TCP          6
   // TCP
#define IPPROTO_PUP          12
   // PUP
#define IPPROTO_UDP          17
   // UDP
#define IPPROTO_IDP          22
   // XNS idp
#define IPPROTO_HC           77
   // UNOFFICIAL net disk protocol
#define IPPROTO_RAW          255
   // Raw IP packet
```
```c
#define WIZCHIP_CRITICAL_ENTER()  
    WIZCHIP.CRIS._enter()

#ifndef _exit  
#undef _exit
#endif

#define WIZCHIP_CRITICAL_EXIT()  
    WIZCHIP.CRIS._exit()

// Basic I/O Function //

uint8_t WIZCHIP_READ(uint32_t AddrSel);

void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb);

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

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

// Common Register I/O function //

#define setMR(mr)  
    WIZCHIP_WRITE(MR,mr)

#define getMR()  
    WIZCHIP_READ(MR)

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

```c
#define getGAR(gar) \  
WIZCHIP_READ_BUF(GAR, gar, 4)

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

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

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

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

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

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

#define setINTLEVEL(intlevel) {  
  WIZCHIP_WRITE(INTLEVEL, (uint8_t)(intlevel >> 8));  
  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(INTLEVEL, 1),  
                (uint8_t)intlevel);  
}

#define getINTLEVEL() \  
((WIZCHIP_READ(INTLEVEL) << 8) +  
  WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL, 1)))
```

//M20150401 : Type explicit declaration
/*
#define getINTLEVEL() \  
((WIZCHIP_READ(INTLEVEL) << 8) +  
  WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL, 1)))
*/
#define getINTLEVEL() \
(((uint16_t)WIZCHIP_READ(INTLEVEL) \
  << 8) + 
  WIZCHIP_READ(WIZCHIP_OFFSET_INC(INTLEVEL,1))))

#define setIR(ir) \
WIZCHIP_WRITE(IR, (ir & 0xF0))

#define getIR() \
(WIZCHIP_READ(IR) & 0xF0)

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

#define getIMR() \
WIZCHIP_READ(_IMR_)

#define setSIR(sir) \
WIZCHIP_WRITE(SIR, sir)

#define getSIR() \
WIZCHIP_READ(SIR)

#define setSIMR(simr) \
WIZCHIP_WRITE(SIMR, simr)

#define getSIMR() \
WIZCHIP_READ(SIMR)

#define setRTR(rtr)  \
{ \
  WIZCHIP_WRITE(_RTR_, (uint8_t)(rtr >> 8)); \
  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(_RTR_,1), 
  (uint8_t) rtr); \
}
/*
#define getRTR() ((WIZCHIP_READ(_RTR_) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(_RTR_,1)))
*/

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

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

//==========================================
======== test done
==============================================

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

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

#define setPHAR(phar) WIZCHIP_WRITE_BUF(PHAR, phar, 6)
#define getPHAR(phar) 
    WIZCHIP_READ_BUF(PHAR, phar, 6)

#define setPSID(psid) {
    WIZCHIP_WRITE(PSID, (uint8_t)(psid >> 8)); \
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PSID,1), (uint8_t) psid); \
}

//uint16_t getPSID(void);
//M20150401 : Type explicit declaration
/*
#define getPSID() 
    ((WIZCHIP_READ(PSID) << 8) + 
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1)))
*/
#endif

#define getPMRU() 
    ((uint16_t)WIZCHIP_READ(PSID) << 8) 
    + WIZCHIP_READ(WIZCHIP_OFFSET_INC(PSID,1))

#define setPMRU(pmru) {
    WIZCHIP_WRITE(PMRU, (uint8_t) (pmru>>8)); \
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(PMRU,1), (uint8_t) pmru); \
}

//M20150401 : Type explicit declaration
/*
#define getPMRU() 
    ((WIZCHIP_READ(PMRU) << 8) + 
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))
*/
*/
```c
#define getPMRU() \  
((uint16_t)WIZCHIP_READ(PMRU) << 8)  
+ WIZCHIP_READ(WIZCHIP_OFFSET_INC(PMRU,1)))

//M20150401 : Size Error of UIPR (6 -> 4)

#define getUIPR(uipr) \  
WIZCHIP_READ_BUF(UIPR,uipr,6)

//M20150401 : Type explict declaration

#define getUPORTR() \  
((WIZCHIP_READ(UPORTR) << 8)  
+ WIZCHIP_READ(WIZCHIP_OFFSET_INC(UPORTR,1)))

#define setPHYCFGR(phycfgr) \  
WIZCHIP_WRITE(PHYCFGR, phycfgr)

#define getPHYCFGR() \  
WIZCHIP_READ(PHYCFGR)

#define getVersionR() \  
WIZCHIP_READ(VERSIONR)

// Socket N register I/O function //

#define setSn_MR(sn, mr) \  
WIZCHIP_WRITE(Sn_MR(sn),mr)
```
```c
#define getSn_MR(sn) \    
    WIZCHIP_READ(Sn_MR(sn))
#define setSn_CR(sn, cr) \    
    WIZCHIP_WRITE(Sn_CR(sn), cr)
#define getSn_CR(sn) \    
    WIZCHIP_READ(Sn_CR(sn))
#define setSn_IR(sn, ir) \    
    WIZCHIP_WRITE(Sn_IR(sn), (ir & 0x1F))
#define getSn_IR(sn) \    
    (WIZCHIP_READ(Sn_IR(sn)) & 0x1F)
#define setSn_IMR(sn, imr) \    
    WIZCHIP_WRITE(Sn_IMR(sn), (imr & 0x1F))
#define getSn_IMR(sn) \    
    (WIZCHIP_READ(Sn_IMR(sn)) & 0x1F)
#define getSn_SR(sn) \    
    WIZCHIP_READ(Sn_SR(sn))
#define setSn_PORT(sn, port) { \    
    WIZCHIP_WRITE(Sn_PORT(sn), (uint8_t)(port >> 8)); \    
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_PORT(sn),1 ), (uint8_t) port); \    
}    
#define getSn_PORT(sn) \    
```

//M20150401 : Type explicit declaration
/*
#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 explicit declaration
1813 */
1814 #define getSn_DPORT(sn) \
#define getSn_DPORT(sn)  
(((uint16_t)WIZCHIP_READ(Sn_DPORT(sn)) << 8) +  
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_DPORT(sn),1)))

#define getSn_MSSR(sn)  
(((uint16_t)WIZCHIP_READ(Sn_MSSR(sn)) << 8) +  
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1)))

#define setSn_MSSR(sn, mss) {  
WIZCHIP_WRITE(Sn_MSSR(sn),  
(uint8_t)(mss>>8));  
WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_MSSR(sn),1),  
(uint8_t) mss);  
}

#define getSn_TOS(sn)  
WIZCHIP_READ(Sn_TOS(sn))
```c
#define setSn_TTL(sn, ttl) \    
    WIZCHIP_WRITE(Sn_TTL(sn), ttl)
#define getSn_TTL(sn) \    
    WIZCHIP_READ(Sn_TTL(sn))
#define setSn_RXBUF_SIZE(sn, rxbufsize) \    
    WIZCHIP_WRITE(Sn_RXBUF_SIZE(sn), rxbufsize)
#define getSn_RXBUF_SIZE(sn) \    
    WIZCHIP_READ(Sn_RXBUF_SIZE(sn))
#define setSn_TXBUF_SIZE(sn, txbufsize) \    
    WIZCHIP_WRITE(Sn_TXBUF_SIZE(sn), txbufsize)
#define getSn_TXBUF_SIZE(sn) \    
    WIZCHIP_READ(Sn_TXBUF_SIZE(sn))

uint16_t getSn_TX_FSR(uint8_t sn);

//M20150401 : Type explicit declaration
/*
#define getSn_TX_RD(sn) \    
    ((WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + 
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn), 1 )))
*/
#define getSn_TX_RD(sn) \    
    (((uint16_t)WIZCHIP_READ(Sn_TX_RD(sn)) << 8) + 
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_RD(sn), 1 )))
```
```c
#define setSn_TX_WR(sn, txwr) { 
    WIZCHIP_WRITE(Sn_TX_WR(sn), (uint8_t)(txwr>>8)); 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn), 1), (uint8_t)txwr); 
} 

//M20150401 : Type explicit declaration
/*
#define getSn_TX_WR(sn) 
    ((WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1 )))
*/
#define getSn_TX_WR(sn) 
    (((uint16_t)WIZCHIP_READ(Sn_TX_WR(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn),1 )))

uint16_t getSn_RX_RSR(uint8_t sn);

#define setSn_RX_RD(sn, rxrd) { 
    WIZCHIP_WRITE(Sn_TX_RD(sn), (uint8_t)(rxrd>>8)); 
    WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_TX_WR(sn), 1), (uint8_t)rxrd); 
} 

//M20150401 : Type explicit declaration
/*
#define getSn_RX_RD(sn) 
    ((WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1 )))
*/
#define getSn_RX_RD(sn) 
```

```
#define getSn_RX_RD(sn) \
  (((uint16_t)WIZCHIP_READ(Sn_RX_RD(sn)) << 8) + 
   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RD(sn),1 ))

#define getSn_RX_WR(sn) \
  (((uint16_t)WIZCHIP_READ(Sn_RX_WR(sn)) << 8) + 
   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_WR(sn),1 ))

#define setSn_FRAG(sn, frag) { \
  WIZCHIP_WRITE(Sn_FRAG(sn), (uint8_t)(frag >>8)); \
  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1 ), (uint8_t) frag); \
}
```
```c
2053 */
2054 #define getSn_FRAG(sn) \n2055 (((uint16_t)WIZCHIP_READ(Sn_FRAG(sn)) << 8) + \n     WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_FRAG(sn),1)))
2056
2064 #define setSn_KPALVTR(sn, kpalvt) \n2065 WIZCHIP_WRITE(Sn_KPALVTR(sn), kpalvt)
2066
2074 #define getSn_KPALVTR(sn) \n2075 WIZCHIP_READ(Sn_KPALVTR(sn))
2076
2080 // Sn_TXBUF & Sn_RXBUF IO function //
2082
2088 //M20150401 : Type explicit declaration
2089 */
2090 #define getSn_RxMAX(sn) \n2091 (getSn_RXBUF_SIZE(sn) << 10)
2092 */
2093 #define getSn_RxMAX(sn) \n2094 (((uint16_t)getSn_RXBUF_SIZE(sn)) << 10)
2095
2102 //M20150401 : Type explicit declaration
2103 */
2104 #define getSn_TxMAX(sn) \n2105 (getSn_TXBUF_SIZE(sn) << 10)
2106 */
2107 #define getSn_TxMAX(sn) \n2108 (((uint16_t)getSn_TXBUF_SIZE(sn)) << 10)
2109
2124 void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len);
```
void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint16_t len);

void wiz_recv_ignore(uint8_t sn, uint16_t len);

#endif

#endif // _W5500_H_
This is the complete list of members for __WIZCHIP, including all inherited members.

<table>
<thead>
<tr>
<th>CRIS</th>
<th>__WIZCHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>__WIZCHIP</td>
</tr>
<tr>
<td>id</td>
<td>__WIZCHIP</td>
</tr>
<tr>
<td>IF</td>
<td>__WIZCHIP</td>
</tr>
<tr>
<td>if_mode</td>
<td>__WIZCHIP</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
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

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
__WIZCHIP::_CS Member List

This is the complete list of members for __WIZCHIP::_CS, including all inherited members.

| __deselect | __WIZCHIP::_CS |
| __select   | __WIZCHIP::_CS |

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
Socke\n
 Socket APIs

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class List</td>
<td>Class Index</td>
<td>Class Members</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__WIZCHIP::__CRIS Member List

This is the complete list of members for __WIZCHIP::__CRIS, including all inherited members.

```cpp
__enter__ __WIZCHIP::__CRIS
__exit__ __WIZCHIP::__CRIS
```

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
Socket APIs

wiz_PhyConf_t Member List

This is the complete list of members for \texttt{wiz\_PhyConf\_t}, including all inherited members.

\begin{itemize}
  \item \texttt{by} \quad \texttt{wiz\_PhyConf\_t}
  \item \texttt{duplex} \quad \texttt{wiz\_PhyConf\_t}
  \item \texttt{mode} \quad \texttt{wiz\_PhyConf\_t}
  \item \texttt{speed} \quad \texttt{wiz\_PhyConf\_t}
\end{itemize}

Generated on Wed May 4 2016 16:44:01 for Socket APIs by \texttt{doxygen} 1.8.9.1
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
wiz_NetTimeout_t Member List

This is the complete list of members for wiz_NetTimeout_t, including all inherited members.

<table>
<thead>
<tr>
<th>retry_cnt</th>
<th>wiz_NetTimeout_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_100us</td>
<td>wiz_NetTimeout_t</td>
</tr>
</tbody>
</table>

Generated on Wed May 4 2016 16:44:01 for Socket APIs by doxygen 1.8.9.1
Socket APIs

w5100.c

Go to the documentation of this file.

```c
1  /************************************************************************************
2  /************************************************************************************
3  #include "w5100.h"
4  
5  #if  (_WIZCHIP_ == 5100)
6  
7  void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb )
8  {
9      WIZCHIP_CRITICAL_ENTER();
10     WIZCHIP_CS._select();
11     
12     #if( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_) )
13     WIZCHIP_IF.SPI._write_byte(0xF0);
14     WIZCHIP_IF.SPI._write_byte((AddrSel & 0xFF00) >> 8);
15     WIZCHIP_IF.SPI._write_byte((AddrSel & 0x00FF) >> 0);
16     WIZCHIP_IF.SPI._write_byte(wb);  // Data write (write 1byte data)
```
#elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
  //M20150601: Rename the function for integrating with ioLibrary
  //WIZCHIP.IF.BUS._write_byte(AddrSel,wb);
  WIZCHIP.IF.BUS._write_data(AddrSel,wb);
  #elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
  //add indirect bus
  //M20150601: Rename the function for integrating with ioLibrary
  //WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0xFF00) >> 8);
  //WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x00FF));
  //WIZCHIP.IF.BUS._write_byte(IDM_DR,wb);
  WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0xFF00) >> 8);
  WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x00FF));
  WIZCHIP.IF.BUS._write_data(IDM_DR,wb);
  #else
  #error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!"
  #endif

WIZCHIP.CS._deselect();
WIZCHIP_CRITICAL_EXIT();
}

uint8_t WIZCHIP_READ(uint32_t AddrSel)
{
  uint8_t ret;

  WIZCHIP_CRITICAL_ENTER();
  WIZCHIP.CS._select();
#if(_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
WIZCHIP.IF.SPI._write_byte(0x0F);
WIZCHIP.IF.SPI._write_byte((AddrSel & 0xFF00) >> 8);
WIZCHIP.IF.SPI._write_byte((AddrSel & 0x00FF) >> 0);
ret = WIZCHIP.IF.SPI._read_byte();
#elsif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
  //M20150601: Rename the function for integrating with ioLibrary
  //ret = WIZCHIP.IF.BUS._read_byte(AddrSel);
  ret = WIZCHIP.IF.BUS._read_data(AddrSel);
#elsif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
  //add indirect bus
  //M20150601: Rename the function for integrating with ioLibrary
  //WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0xFF00) >> 8);
  //WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x00FF));
  //ret = WIZCHIP.IF.BUS._read_byte(IDM_DR);
  WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0xFF00) >> 8);
  WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x00FF));
  ret = WIZCHIP.IF.BUS._read_data(IDM_DR);
#else
  #error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!"
#endif
111 WIZCHIP.CS._deselect();
112 WIZCHIP_CRITICAL_EXIT();
113 return ret;
114 }
115
116
117
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(); //M20150601: Moved here.
123     #if(_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
124         for(i = 0; i < len; i++)
125         {
126             //M20160715: Deprecated "M20150601: Remove _select() to top-side"
127             //     CS should be controlled every SPI frames
128             WIZCHIP.CS._select();
129             WIZCHIP.IF.SPI._write_byte(0xF0);
130             WIZCHIP.IF.SPI._write_byte(((uint16_t)(AddrSel+i)) & 0xFF00) >> 8);
131             WIZCHIP.IF.SPI._write_byte(((uint16_t)(AddrSel+i)) & 0x00FF) >> 0);
132             WIZCHIP.IF.SPI._write_byte(pBuf[i]); // Data write (write 1byte data)
133         }   //M20160715: Deprecated "M20150601: Remove _select() to top-side"
134     WIZCHIP.CS._deselect();
135 #elif (_WIZCHIP_IO_MODE_ ==
for(i = 0; i < len; i++)

// M20150601: Rename the function for integrating with ioLibrary

WIZCHIP.IO.BUS._write_byte(AddrSel+i,pBuf[i]);

WIZCHIP.IO.BUS._write_data(AddrSel+i,pBuf[i]);

#elif ((__WIZCHIP_IO_MODE__ == _WIZCHIP_IO_MODE_BUS_INDIR__) )

// M20150601: Rename the function for integrating with ioLibrary

/*
 * WIZCHIP_WRITE(MR,WIZCHIP_READ(MR) | MR_AI);
 * WIZCHIP.IO.BUS._write_byte(IDM_AR0, (AddrSel & 0xFF00) >> 8);
 * WIZCHIP.IO.BUS._write_byte(IDM_AR1, (AddrSel & 0x00FF));
 * for(i = 0 ; i < len; i++)
 * WIZCHIP.IO.BUS._write_byte(IDM_DR,pBuf[i]);
 * WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
 * */

setMR(getMR()|MR_AI);

WIZCHIP.IO.BUS._write_data(IDM_AR0, (AddrSel & 0xFF00) >> 8);

WIZCHIP.IO.BUS._write_data(IDM_AR1, (AddrSel & 0x00FF));

for(i = 0 ; i < len; i++)

WIZCHIP.IO.BUS._write_data(IDM_DR,pBuf[i]);

setMR(getMR() & ~MR_AI);
#else
    #error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!!"
#endif

WIZCHIP_CS._deselect();  //M20150601: Moved here.
WIZCHIP_CRITICAL_EXIT();
}

void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len)
{
    uint16_t i = 0;
    WIZCHIP_CRITICAL_ENTER();
    WIZCHIP_CS._select();  //M20150601: Moved here.
    #if( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_) )
        for(i = 0; i < len; i++)
        {
            //M20160715: Deprecated "M20150601: Remove _select() to top-side"
            //     CS should be controlled every SPI frames
            WIZCHIP_CS._select();
            WIZCHIP_IF.SPI._write_byte(0x0F);
            WIZCHIP_IF.SPI._write_byte((uint16_t)((AddrSel+i) & 0xFF00) >> 8);
            WIZCHIP_IF.SPI._write_byte((uint16_t)((AddrSel+i) & 0x00FF) >> 0);
            pBuf[i] = WIZCHIP_IF.SPI._read_byte();
            //M20160715: Deprecated "M20150601: Remove _select() to top-side"
        }
    }
    WIZCHIP_CS._deselect();
```c
#elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
    for(i = 0; i < len; i++)
        //M20150601 : Rename the function for integrating with ioLibrary
        pBuf[i] = WIZCHIP.IF.BUS._read_byte(AddrSel+i);
    pBuf[i] = WIZCHIP.IF.BUS._read_data(AddrSel+i);
#elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
    //M20150601 : Rename the function for integrating with ioLibrary
    /*
    WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) | MR_AI);
    WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0xFF00) >> 8);
    WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x00FF));
    for(i = 0; i < len; i++)
        pBuf[i] = WIZCHIP.IF.BUS._read_byte(IDM_DR);
    WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
    */
    setMR(getMR() | MR_AI);
    WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0xFF00) >> 8);
    WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x00FF));
    for(i = 0; i < len; i++)
        pBuf[i] = WIZCHIP.IF.BUS._read_data(IDM_DR);
    setMR(getMR() & ~MR_AI);
#else
```

#error "Unknown _WIZCHIP_IO_MODE_ in W5100. !!!!!"

WIZCHIP.CS._deselect(); //M20150601 : Moved Here.

WIZCHIP_CRITICAL_EXIT();
}

// Socket N regsiter IO function //

uint16_t getSn_TX_FSR(uint8_t sn)
{
    uint16_t val=0,val1=0;
    do
    {
        val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
        val1 = (val1 << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
        if (val1 != 0)
        {
            val = WIZCHIP_READ(Sn_TX_FSR(sn));
            val = (val << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
        }
    }while (val != val1);
    return val;
}

uint16_t getSn_RX_RSR(uint8_t sn)
{
    uint16_t val=0,val1=0;
    do
    {
    
}
val1 = WIZCHIP_READ(Sn_RX_RSR(sn));

val1 = (val1 << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));

if (val1 != 0)
{
    val = WIZCHIP_READ(Sn_RX_RSR(sn));
    val = (val << 8) + WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));
}

while (val != val1);
return val;

// Sn_TXBUF & Sn_RXBUF IO function //

guts
uint32_t getSn_RxBASE(uint8_t sn)
{
    int8_t i;
    #if (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
        uint32_t rxbase = _W5100_IO_BASE_ + _WIZCHIP_IO_TXBUF_
    #else
        uint32_t rxbase = _WIZCHIP_IO_TXBUF_;
    #endif
    for(i = 0; i < sn; i++)
        rxbase += getSn_RxMAX(i);
    return rxbase;
}


guts
uint32_t getSn_TxBASE(uint8_t sn)
{
    int8_t i;
    #if (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
        uint32_t rxbase = _W5100_IO_BASE_ + _WIZCHIP_IO_TXBUF_
    #else
        uint32_t rxbase = _WIZCHIP_IO_TXBUF_;
    #endif
    for(i = 0; i < sn; i++)
        rxbase += getSn_RxMAX(i);
    return rxbase;
}
```c
uint32_t txbase = _W5100_IO_BASE_ + _WIZCHIP_IO_TXBUF_;  

#ifndef
    uint32_t txbase = _WIZCHIP_IO_TXBUF_;  
#endif

for(i = 0; i < sn; i++)
    txbase += getSn_TxMAX(i);

return txbase;

void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len)
{
    uint16_t ptr;
    uint16_t size;
    uint16_t dst_mask;
    uint16_t dst_ptr;

    ptr = getSn_TX_WR(sn);

    dst_mask = ptr & getSn_TxMASK(sn);
    dst_ptr = getSn_TxBASE(sn) + dst_mask;

    if (dst_mask + len > getSn_TxMAX(sn))
    {
        size = getSn_TxMAX(sn) - dst_mask;
        WIZCHIP_WRITE_BUF(dst_ptr, wizdata, size);
        wizdata += size;
        size = len - size;
        dst_ptr = getSn_TxBASE(sn);
        WIZCHIP_WRITE_BUF(dst_ptr, wizdata, size);
    }
    else
    {
        WIZCHIP_WRITE_BUF(dst_ptr, wizdata,
```

void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint16_t len) {
    uint16_t ptr;
    uint16_t size;
    uint16_t src_mask;
    uint16_t src_ptr;

    ptr = getSn_RX_RD(sn);
    src_mask = (uint32_t)ptr & getSn_RxMASK(sn);
    src_ptr = (getSn_RxBASE(sn) + src_mask);

    if ((src_mask + len) > getSn_RxMAX(sn)) {
        size = getSn_RxMAX(sn) - src_mask;
        WIZCHIP_READ_BUF((uint32_t)src_ptr, (uint8_t*)wizdata, size);
        wizdata += size;
        size = len - size;
        src_ptr = getSn_RxBASE(sn);
        WIZCHIP_READ_BUF(src_ptr, (uint8_t*)wizdata, size);
    } else {
        ...
    }
}
WIZCHIP_READ_BUF(src_ptr,
    (uint8_t*)wizdata, len);
}

ptr += len;

setSn_RX_RD(sn, ptr);
}

void wiz_recv_ignore(uint8_t sn, uint16_t len)
{
    uint16_t ptr;
    ptr = getSn_RX_RD(sn);
    ptr += len;
    setSn_RX_RD(sn,ptr);
}

#define
Socket APIs

w5200.c

Go to the documentation of this file.

```
1  //***************************************************************************************
2  //***************************************************************************************
3  //
4  //***************************************************************************************
5  //***************************************************************************************

6  #include "w5200.h"
7
8  #if (_WIZCHIP_ == 5200)

9  void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb )
10  {
11      WIZCHIP_CRITICAL_ENTER();
12      WIZCHIP_CS._select();
13
14      #if( (_WIZCHIP_IO_MODE_ &
15            _WIZCHIP_IO_MODE_SPI_))
16          WIZCHIP_IF.SPI._write_byte((AddrSel &
17                                          0x0000FF00) >> 8);
18          WIZCHIP_IF.SPI._write_byte((AddrSel &
19                                          0x000000FF) >> 0);
20          WIZCHIP_IF.SPI._write_byte(_W5200_SPI_WRITE_);
21     // 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 }
77
80 uint8_t WIZCHIP_READ(uint32_t AddrSel)
81 {
82    uint8_t ret;
83
84    WIZCHIP_CRITICAL_ENTER();
85    WIZCHIP_CS._select();
#if(_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_)
WIZCHIP.IF.SPI._write_byte((AddrSel & 0x0000FF00) >> 8);
WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF) >> 0);
WIZCHIP.IF.SPI._write_byte(_W5200_SPI_READ_);
// Read data length upper
WIZCHIP.IF.SPI._write_byte(0x01);
// Data length lower
ret = WIZCHIP.IF.SPI._read_byte();
#elif (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_BUS_)
//add indirect bus
//M20150601 : Rename the function for integrating with W5300
//WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0x0000FF00) >> 8);
//WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x000000FF));
//ret = WIZCHIP.IF.BUS._read_byte(IDM_DR);
WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0x0000FF00) >> 8);
WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x000000FF));
ret = WIZCHIP.IF.BUS._read_data(IDM_DR);
#else
#error "Unknown _WIZCHIP_IO_MODE_ in W5200. !!!"
#endif
109         WIZCHIP.CS._deselect();
110         WIZCHIP_CRITICAL_EXIT();
111         return ret;
112     }
113     
114     void WIZCHIP_WRITE_BUF(uint32_t AddrSel,
115         uint8_t* pBuf, uint16_t len)
116     {
117         uint16_t i = 0;
118         WIZCHIP_CRITICAL_ENTER();
119         WIZCHIP.CS._select();
120         
121         #if( (_WIZCHIP_IO_MODE_ &
122             _WIZCHIP_IO_MODE_SPI_))
123         WIZCHIP.IF.SPI._write_byte((AddrSel &
124             0x0000FF00) >> 8);
125         WIZCHIP.IF.SPI._write_byte((AddrSel &
126             0x000000FF) >> 0);
127         WIZCHIP.IF.SPI._write_byte(_W5200_SPI_WRITE_ |
128             ((len & 0x7F00) >> 8));               // Write data
129         WIZCHIP.IF.SPI._write_byte((len & 0x00FF)
130             >> 0);                                 // length lower
131         for(i = 0; i < len; i++)
132             WIZCHIP.IF.SPI._write_byte(pBuf[i]);
133         
134         #elif ( (_WIZCHIP_IO_MODE_ &
135             _WIZCHIP_IO_MODE_BUS_) )
136         //M20150601 : Rename the function for
137         // integrating with W5300
138         /*
139         WIZCHIP_WRITE(MR,WIZCHIP_READ(MR) |
140             MR_AI);
141         WIZCHIP.IF.BUS._write_byte(IDM_AR0,
142             (AddrSel & 0x0000FF00) >> 8);
137 | WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x000000FF));
138 | for(i = 0 ; i < len; i++)
139 |
140 | WIZCHIP.IF.BUS._write_byte(IDM_DR,pBuf[i]);
141 | WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
142 | */
143 | setMR(getMR() | MR_AI);
144 | WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0x0000FF00) >> 8);
145 | WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x000000FF));
146 | for(i = 0 ; i < len; i++)
147 | WIZCHIP.IF.BUS._write_data(IDM_DR,pBuf[i]);
148 | WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
149 | #else
150 | #error "Unknown _WIZCHIP_IO_MODE_ in W5200. !!!!"
151 | #endif
152 | WIZCHIP_CS._deselect();
153 | WIZCHIP_CRITICAL_EXIT();
154 |
155 | void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len)
156 | {
157 | uint16_t i = 0;
158 | WIZCHIP_CRITICAL_ENTER();
159 | WIZCHIP_CS._select();
160 | #if( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_SPI_) )
161 | WIZCHIP.IF.SPI._write_byte((AddrSel &
162 | (AddrSel & 0x000000FF) >> 8);
163 | WIZCHIP.IF.SPI._write_byte((AddrSel & 0x0000FF00) >> 8);
164 | WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF));
165 | WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
166 | WIZCHIP_CS._deselect();
167 | WIZCHIP_CRITICAL_EXIT();
168 | }
0x0000FF00) >> 8);
WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF) >> 0);
WIZCHIP.IF.SPI._write_byte(_W5200_SPI_READ_ | ((len & 0x7F00) >> 8));  // Write data op code and length upper
WIZCHIP.IF.SPI._write_byte((len & 0x00FF) >> 0);  // length lower
for(i = 0; i < len; i++)
pBuf[i] = WIZCHIP.IF.SPI._read_byte();

#elif ( (_WIZCHIP_IO_MODE_ & _WIZCHIP_IO_MODE_BUS_ ) )
//M20150601 : Rename the function for integrating with W5300
/*
WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) | MR_AI);
WIZCHIP.IF.BUS._write_byte(IDM_AR0, (AddrSel & 0x0000FF00) >> 8);
WIZCHIP.IF.BUS._write_byte(IDM_AR1, (AddrSel & 0x000000FF));
for(i = 0 ; i < len; i++)
pBuf[i] = WIZCHIP.IF.BUS._read_byte(IDM_DR);
WIZCHIP_WRITE(MR, WIZCHIP_READ(MR) & ~MR_AI);
*/
setMR(getMR() | MR_AI);
WIZCHIP.IF.BUS._write_data(IDM_AR0, (AddrSel & 0x0000FF00) >> 8);
WIZCHIP.IF.BUS._write_data(IDM_AR1, (AddrSel & 0x000000FF));
for(i = 0 ; i < len; i++)
pBuf[i] = WIZCHIP.IF.BUS._read_data(IDM_DR);
setMR(getMR() & ~MR_AI);
#else
    #error "Unknown _WIZCHIP_IO_MODE_ in W5200. !!!!"
#endif

WIZCHIP_CS._deselect();
WIZCHIP_CRITICAL_EXIT();
}

// Socket N registers IO function //

uint16_t getSn_TX_FSR(uint8_t sn)
{
    uint16_t val=0,val1=0;
    do
    {
        val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
        val1 = (val1 << 8) +
               WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
        if (val1 != 0)
            {
                val = WIZCHIP_READ(Sn_TX_FSR(sn));
                val = (val << 8) +
                       WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
            }
    }while (val != val1);
    return val;
}

uint16_t getSn_RX_RSR(uint8_t sn)
{
    uint16_t val=0,val1=0;
    do
    {
    

val1 = WIZCHIP_READ(Sn_RX_RSR(sn));
val1 = (val1 << 8) +
WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));

if (val1 != 0)
{
    val = WIZCHIP_READ(Sn_RX_RSR(sn));
    val = (val << 8) +
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));
}

while (val != val1);
return val;

// Sn_TXBUF & Sn_RXBUF IO function //

uint16_t getSn_RxBASE(uint8_t sn)
{
    int8_t i;
    uint16_t rxbase = _WIZCHIP_IO_RXBUF_;
    for(i = 0; i < sn; i++)
        rxbase += getSn_RxMAX(i);
    return rxbase;
}

uint16_t getSn_TxBASE(uint8_t sn)
{
    int8_t i;
    uint16_t txbase = _WIZCHIP_IO_TXBUF_;
    for(i = 0; i < sn; i++)
        txbase += getSn_TxMAX(i);
    return txbase;
}

void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len)
{
    uint16_t ptr;
    uint16_t size;
    uint16_t dst_mask;
    uint8_t * dst_ptr;

    ptr = getSn_TX_WR(sn);

    dst_mask = (uint32_t)ptr & getSn_TxMASK(sn);
    dst_ptr = (uint8_t*)((uint32_t)getSn_TxBASE(sn) + dst_mask);

    if (dst_mask + len > getSn_TxMAX(sn)) {
        size = getSn_TxMAX(sn) - dst_mask;
        WIZCHIP_WRITE_BUF((uint32_t)dst_ptr, wizdata, size);
        wizdata += size;
        size = len - size;
        dst_ptr = (uint8_t*)((uint32_t)getSn_TxBASE(sn));
        WIZCHIP_WRITE_BUF((uint32_t)dst_ptr, wizdata, size);
    } else {
        WIZCHIP_WRITE_BUF((uint32_t)dst_ptr, wizdata, len);
    }

    ptr += len;
    setSn_TX_WR(sn, ptr);
}
void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint16_t len)
{
    uint16_t ptr;
    uint16_t size;
    uint16_t src_mask;
    uint8_t * src_ptr;

    ptr = getSn_RX_RD(sn);

    src_mask = (uint32_t)ptr & getSn_RxMASK(sn);
    src_ptr = (uint8_t *)((uint32_t)getSn_RxBASE(sn) + src_mask);

    if( (src_mask + len) > getSn_RxMAX(sn) )
    {
        size = getSn_RxMAX(sn) - src_mask;
        WIZCHIP_READ_BUF((uint32_t)src_ptr, (uint8_t*)wizdata, size);
        wizdata += size;
        size = len - size;
        src_ptr = (uint8_t*)((uint32_t)getSn_RxBASE(sn));
        WIZCHIP_READ_BUF((uint32_t)src_ptr, (uint8_t*)wizdata, size);
    }
    else
    {
        WIZCHIP_READ_BUF((uint32_t)src_ptr, (uint8_t*)wizdata, len);
    }

    ptr += len;
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.

```c
#include <stdint.h>
#include "wizchip_conf.h"

#if _WIZCHIP_ == 5300
extern uint8_t sock_remained_byte[_WIZCHIP_SOCK_NUM_];
extern uint8_t sock_pack_info[_WIZCHIP_SOCK_NUM_];

/***********************
* Basic I/O Function *
*************************/

void WIZCHIP_WRITE(uint32_t AddrSel, uint16_t wb )
{
    WIZCHIP_CRITICAL_ENTER();
```
WIZCHIP_CS._select();

#if (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
 #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
 WIZCHIP_IF.BUS._write_data(AddrSel, (uint8_t)(wb>>8));
 WIZCHIP_IF.BUS._write_data(WIZCHIP_OFFSET_INC(AddrSel,1),(uint8_t)wb);
 #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
 WIZCHIP_IF.BUS._write_data(IDM_AR,(uint8_t)(AddrSel >> 8));
 WIZCHIP_IF.BUS._write_data(IDM_DR,(uint8_t)(wb>>8));
 #else
 #error "Abnormal _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
 #endif
#endif

#elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
 #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
 WIZCHIP_IF.BUS._write_data(IDM_AR,(uint8_t)(AddrSel >> 8));
 WIZCHIP_IF.BUS._write_data(IDM_DR,(uint8_t)(wb>>8));
 WIZCHIP_IF.BUS._write_data(WIZCHIP_OFFSET_INC(IDM_AR,1),(uint8_t)AddrSel);
 WIZCHIP_IF.BUS._write_data(WIZCHIP_OFFSET_INC(IDM_DR,1),(uint8_t)wb);
 #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
 WIZCHIP_IF.BUS._write_data(IDM_AR,(uint16_t)AddrSel);
 WIZCHIP_IF.BUS._write_data(IDM_DR,wb);
 #else
 #error "Abnormal"
#if (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_DIR_)
    #if (_WIZCHIP_IO_BUS_WIDTH_ == 8)
        ret = (((uint16_t)WIZCHIP_IF.BUS._read_data(AddrSel)) << 8) |
              (((uint16_t)WIZCHIP_IF.BUS._read_data(WIZCHIP_OFFSET_INC(AddrSel,1))) & 0x00FF);
    #elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
        ret = WIZCHIP_IF.BUS._read_data(AddrSel);
    #else
        #error "Abnormal _WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
    #endif
#else
    #elif (_WIZCHIP_IO_MODE_ == _WIZCHIP_IO_MODE_BUS_INDIR_)
        #if(_WIZCHIP_IO_BUS_WIDTH_ == 8)
WIZCHIP_IF.BUS._write_data(IDM_AR, (uint8_t)(AddrSel >> 8));

WIZCHIP_IF.BUS._write_data(WIZCHIP_OFFSET_INC(IDM_AR, 1), (uint8_t)AddrSel);

ret = (((uint16_t)WIZCHIP_IF.BUS._read_data(IDM_DR)) << 8) |

(((uint16_t)WIZCHIP_IF.BUS._read_data(WIZCHIP_OFFSET_INC(IDM_DR, 1))) & 0x00FF);

#elif(_WIZCHIP_IO_BUS_WIDTH_ == 16)
WIZCHIP_IF.BUS._write_data(IDM_AR, (uint16_t)AddrSel);
ret = WIZCHIP_IF.BUS._read_data(IDM_DR);

#else
#error "Abnorma1
_WIZCHIP_IO_BUS_WIDTH_. Should be 8 or 16"
#endif
#else
#error "Unknown _WIZCHIP_IO_MODE_ in
W5300. !!!"
#endif

WIZCHIP_CS._deselect();
WIZCHIP_CRITICAL_EXIT();
return ret;

void setTMSR(uint8_t sn, uint8_t tmsr) {
  uint16_t tmem;
  tmem = WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn & 0xFE)));
if(sn & 0x01)  
tmem = (tmem & 0xFF00) |  
  (((uint16_t)tmsr ) & 0x00FF) ;  
else  
tmem = (tmem & 0x00FF) |  
  (((uint16_t)tmsr) << 8) ;

WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(TMS01R,  
  (sn & 0xFE)),tmem);
}

uint8_t getTMSR(uint8_t sn)
{
  if(sn & 0x01)
    return (uint8_t)
      (WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn & 0xFE))) & 0x00FF);
  return (uint8_t)
      (WIZCHIP_READ(WIZCHIP_OFFSET_INC(TMS01R, (sn & 0xFE))) >> 8);
}

void setRMSR(uint8_t sn,uint8_t rmsr)
{
  uint16_t rmem;
  rmem =  
    WIZCHIP_READ(WIZCHIP_OFFSET_INC(RMS01R, (sn & 0xFE)));  
  if(sn & 0x01)  
    rmem = (rmem & 0xFF00) |  
      (((uint16_t)rmsr ) & 0x00FF) ;  
  else  
    rmem = (rmem & 0x00FF) |  
      (((uint16_t)rmsr) << 8) ;
  WIZCHIP_WRITE(WIZCHIP_OFFSET_INC(RMS01R,  
    (sn & 0xFE)),rmem);
}

uint8_t getRMSR(uint8_t sn)
{
  if(sn & 0x01)
    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 {
   uint32_t free_tx_size=0;
   uint32_t free_tx_size1=1;
   while(1)
   {
      free_tx_size = (((uint32_t)WIZCHIP_READ(Sn_TX_FSR(sn))) << 16) |
      (((uint32_t)WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn _TX_FSR(sn),2))) & 0x0000FFFF);
      // read
      if(free_tx_size == free_tx_size1)
         break; // if first == second, Sn_TX_FSR value is valid.
      free_tx_size1 = free_tx_size;
      // save second value into first
   }
   return free_tx_size;

173 }

174

175 uint32_t getSn_RX_RSR(uint8_t sn)
176 {
   uint32_t received_rx_size=0;
   uint32_t received_rx_size1=1;
   while(1)
   {
      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 == second, 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 void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint32_t len)
190 | {
191 | uint32_t i = 0;
192 | if(len == 0) return;
193 | for(i = 0; i < len ; i += 2)
194 | setSn_TX_FIFOR(sn,
195 | (((uint16_t)wizdata[i]) << 8) |
196 | (((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 | if(len == 0) return;
204 | for(i = 0; i < len; i++)
if((i & 0x01)==0)
{
    rd = getSn_RX_FIFOR(sn);
    wizdata[i] = (uint8_t)(rd >> 8);
}
else wizdata[i] = (uint8_t)rd; // For checking the memory access violation

sock_remained_byte[sn] = (uint8_t)rd; // back up the remaind fifo byte.

void wiz_recv_ignore(uint8_t sn, uint32_t len)
{
    uint32_t i = 0;
    for(i = 0; i < len ; i += 2)
        getSn_RX_FIFOR(sn);
}

#endif
Socket APIs

w5500.c

Go to the documentation of this file.

```c
1  //******************************************
   //******************************************
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
64  uint8_t  WIZCHIP_READ(uint32_t AddrSel)
65  {
66     uint8_t ret;
67     uint8_t spi_data[3];
68
69     WIZCHIP_CRITICAL_ENTER();
70     WIZCHIP_CS._select();
71
72     AddrSel |= (_W5500_SPI_READ_ | _W5500_SPI_VDM_OP_);
if(!WIZCHIP.IF.SPI._read_burst || !WIZCHIP.IF.SPI._write_burst) // byte operation
{
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x00FF0000) >> 16);
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x0000FF00) >> 8);
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF) >> 0);
}
else // burst operation
{
    spi_data[0] = (AddrSel & 0x00FF0000) >> 16;
    spi_data[1] = (AddrSel & 0x0000FF00) >> 8;
    spi_data[2] = (AddrSel & 0x000000FF) >> 0;
    WIZCHIP.IF.SPI._write_burst(spi_data, 3);
}
ret = WIZCHIP.IF.SPI._read_byte();
WIZCHIP.CS._deselect();
WIZCHIP_CRITICAL_EXIT();
return ret;

void WIZCHIP_WRITE(uint32_t AddrSel, uint8_t wb)
{
    uint8_t spi_data[4];
    WIZCHIP_CRITICAL_ENTER();
    WIZCHIP.CS._select();
AddrSel |= (_W5500_SPI_WRITE_ | _W5500_SPI_VDM_OP_);

// if(!WIZCHIP.IF.SPI._read_burst || !WIZCHIP.IF.SPI._write_burst) // byte operation
if(!WIZCHIP.IF.SPI._write_burst) // byte operation
{
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x00FF0000) >> 16);
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x0000FF00) >> 8);
    WIZCHIP.IF.SPI._write_byte((AddrSel & 0x000000FF) >> 0);
    WIZCHIP.IF.SPI._write_byte(wb);
}
else // burst operation
{
    spi_data[0] = (AddrSel & 0x00FF0000) >> 16;
    spi_data[1] = (AddrSel & 0x0000FF00) >> 8;
    spi_data[2] = (AddrSel & 0x000000FF) >> 0;
    spi_data[3] = wb;
    WIZCHIP.IF.SPI._write_burst(spi_data, 4);
}

WIZCHIP.CS._deselect();
WIZCHIP_CRITICAL_EXIT();

void WIZCHIP_READ_BUF (uint32_t AddrSel, uint8_t* pBuf, uint16_t len)
```c
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          }
```
void WIZCHIP_WRITE_BUF(uint32_t AddrSel, uint8_t* pBuf, uint16_t len) {
    uint8_t spi_data[3];
    uint16_t i;

    WIZCHIP_CRITICAL_ENTER();
    WIZCHIP_CS_.select();

    AddrSel |= (_W5500_SPI_WRITE_ | _W5500_SPI_VDM_OP_);

    if(!WIZCHIP_IF_.SPI_.write_burst) // byte operation
        
        WIZCHIP_IF_.SPI_.write_byte((AddrSel & 0x00FF0000) >> 16);
        WIZCHIP_IF_.SPI_.write_byte((AddrSel & 0x0000FF00) >> 8);
        WIZCHIP_IF_.SPI_.write_byte((AddrSel & 0x000000FF) >> 0);
        for(i = 0; i < len; i++)
            WIZCHIP_IF_.SPI_.write_byte(pBuf[i]);
    
    else // burst operation
        
        spi_data[0] = (AddrSel & 0x00FF0000) >> 16;
        spi_data[1] = (AddrSel & 0x0000FF00) >> 8;
        spi_data[2] = (AddrSel & 0x000000FF)
WIZCHIP_IF.SPI._write_burst(spi_data, 3);
WIZCHIP_IF.SPI._write_burst(pBuf, len);
}

WIZCHIP_CS._deselect();
WIZCHIP_CRITICAL_EXIT();

uint16_t getSn_TX_FSR(uint8_t sn) {
    uint16_t val=0,val1=0;
    do {
        val1 = WIZCHIP_READ(Sn_TX_FSR(sn));
        val1 = (val1 << 8) +
               WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
        if (val1 != 0) {
            val = WIZCHIP_READ(Sn_TX_FSR(sn));
            val = (val << 8) +
                   WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_TX_FSR(sn), 1));
        }
    }while (val != val1);
    return val;
}

uint16_t getSn_RX_RSR(uint8_t sn) {
    uint16_t val=0,val1=0;
do
{
    val1 = WIZCHIP_READ(Sn_RX_RSR(sn));
    val1 = (val1 << 8) +
          WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));
    if (val1 != 0)
    {
        val = WIZCHIP_READ(Sn_RX_RSR(sn));
        val = (val << 8) +
              WIZCHIP_READ(WIZCHIP_OFFSET_INC(Sn_RX_RSR(sn), 1));
    }
} while (val != val1);
return val;
}

void wiz_send_data(uint8_t sn, uint8_t *wizdata, uint16_t len)
{
    uint16_t ptr = 0;
    uint32_t addrsel = 0;
    if(len == 0) return;
    ptr = getSn_TX_WR(sn);
    //M20140501 : implicit type casting -> explicit type casting
    //addrsel = (ptr << 8) +
               (WIZCHIP_TXBUF_BLOCK(sn) << 3);
    addrsel = ((uint32_t)ptr << 8) +
              (WIZCHIP_TXBUF_BLOCK(sn) << 3);
    //
    WIZCHIP_WRITE_BUF(addrsel,wizdata, len);
    ptr += len;
    setSn_TX_WR(sn,ptr);
void wiz_recv_data(uint8_t sn, uint8_t *wizdata, uint16_t len) {
    uint16_t ptr = 0;
    uint32_t addrsel = 0;
    if(len == 0) return;
    ptr = getSn_RX_RD(sn);
    //M20140501 : implicit type casting -> explicit type casting
    //addrsel = ((ptr << 8) +
    (WIZCHIP_RXBUF_BLOCK(sn) << 3);
    addrsel = ((uint32_t)ptr << 8) +
    (WIZCHIP_RXBUF_BLOCK(sn) << 3);
    WIZCHIP_READ_BUF(addrsel, wizdata, len);
    ptr += len;
    setSn_RX_RD(sn,ptr);
}

void wiz_recv_ignore(uint8_t sn, uint16_t len) {
    uint16_t ptr = 0;
    ptr = getSn_RX_RD(sn);
    ptr += len;
    setSn_RX_RD(sn,ptr);
}
#endif