E2ForLife_W5100

Interface Driver for the WIZnet W5100 Device

V1.2

Features

- Compatible with PSoC 3, 4, 5, and 5LP
- SPI port independent (Uses Either SCB or SPIM)
- Supports TCP, UDP, and ARP
- 4 simultaneous protocol Sockets
- 2K off-processor packet buffer per socket
-

General Description

The W5100 interface driver provides a simple software driver for using the WIZnet W5100 iEthernet controller with a PSoC project. The driver can be customized to support many system configurations, and allows for SPI port sharing. Both the SPIM and the SCB interfaces are supported to allow the driver to support many hardware configurations of the application.

Using the W5100 Driver

Schematic Requirements

This driver is a software only driver, thus in order to effectively use the functions provided to access the W5100, there must be a hardware interface defined in the schematics to access the device. Once entered in the schematics, enter the instance name of the SPI component in to the customizer parameters for the driver component.

Using the Driver

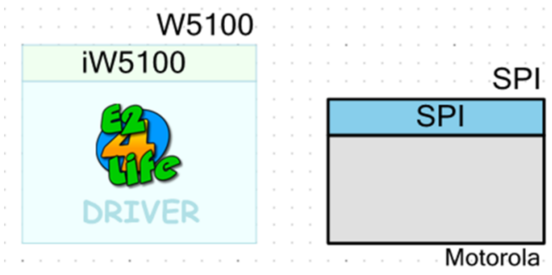
When using the driver, simply start the driver with the Start() function then open a socket using the protocol desired, and either start a server to wait for connections or connect to a remote server. The W5100 device and the software driver handle the interfacing, management, and data handling for the connections.

Schematic Macros

As part of the distribution of the driver component, two schematic macros have been provided to simplify the use of the component. Each macro is defined for the type of SPI interface used to communicate with the W5100 device, and have all of the options set so that they will correctly communicate with the W5100.

SCB mode (PSoC 4 Only)

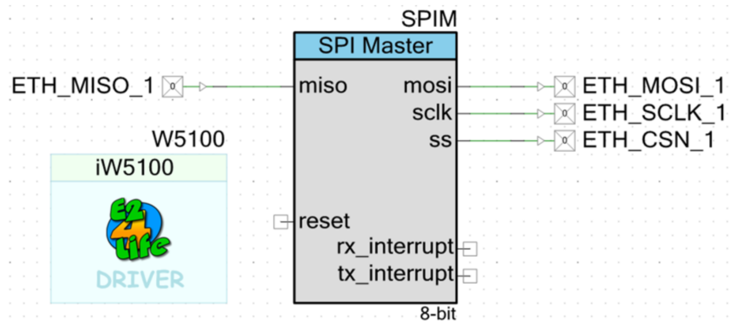
When using a SCB to communicate with the W5100 device, you should use the schematic macro "Community\Communications\Ethernet\E2ForLife - W5100 (SCB Mode)". This macro has the SCB and the W5100 device driver configured for proper operation with the W5100 device.



To add Ethernet support to a project which already contains an SPI port, just drop the W5100 driver component on to your schematics and set the SPI_INSTANCE configuration parameter to the instance name of the SPI device in your project. You might also wish to double-check the configuration of the SPI port to make sure that it will support the W5100 interface.

SPIM mode (SPI Master)

When using a SPI Master component (SPIM) as the communications interface with the W5100 device, you should start with the "Community\Communications\Ethernet\E2ForLife – W5100 (SPIM mode)" schematic macro. This macro contains the pre-configured device driver and SPI master mode device to support proper operation of the device driver with the W5100 device.



SPI Configuration

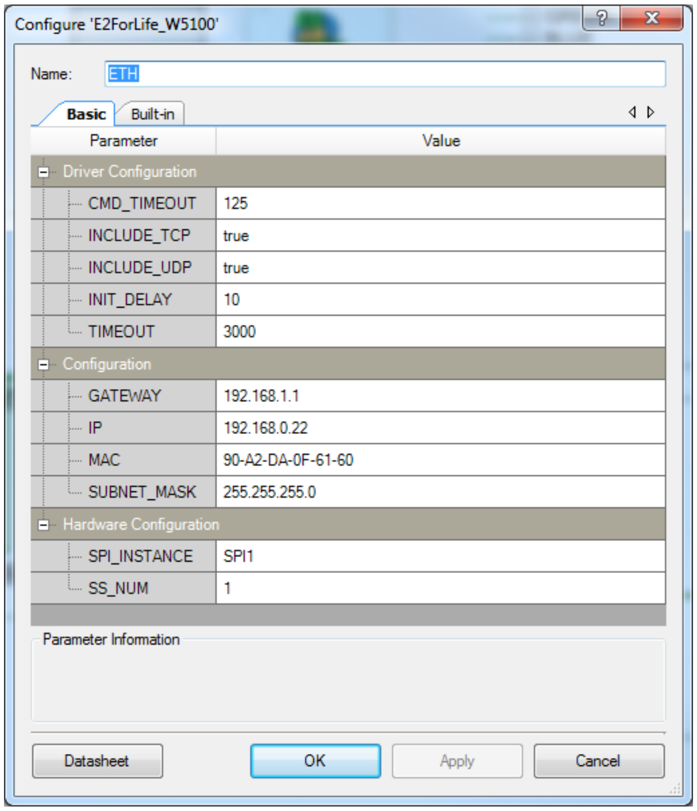
- Mode 0 or 3
- 8-bit data, MSB first
- At Least a 4-byte FIFO buffer
- Continuous SS mode (SCB Only)
- Motorola Style SS (SCB Only)
- Chip Select generated by SPI component.
- **NOTE**: In some instances when using the SPIM, the "*Enable High-Speed Mode*" option may be required for proper operation.

Input/output Connections

There are no Input or Output connections to this component.

Component Parameters

Drag a W5100 component on to your design and double-click to open the component configuration dialog.


Configure 'E2ForLife_W5100'

Name: ETH

Parameter	Value
Driver Configuration	
CMD_TIMEOUT	125
INCLUDE_TCP	true
INCLUDE_UDP	true
INIT_DELAY	10
TIMEOUT	3000
Configuration	
GATEWAY	192.168.1.1
IP	192.168.0.22
MAC	90-A2-DA-0F-61-60
SUBNET_MASK	255.255.255.0
Hardware Configuration	
SPI_INSTANCE	SPI1
SS_NUM	1

Parameters

Driver Configuration Parameters

This section contains parameters that modify the operation of the driver, or provide settings for options of the driver implementation.

CMD_TIMEOUT – The number of milliseconds to wait for a W5100 Command to execute

This parameter will allow you to set the amount of time that the internal driver function used to execute socket commands within the W5100 device will wait for a command to execute before declaring a timeout condition.

INCLUDE_TCP – Set to True to enable the TCP interface code

A True/False parameter used to specify if the TCP protocol interface code will be included when compiling the driver software. Set this to False when not using the TCP interface functions to save FLASH memory space. The default setting is to include this interface code.

INCLUDE_UDP – Set to True to enable the UDP interface Code

A True/False parameter used to specify if the UDP protocol interface code will be compiled when building the library. Set this to False if not using the UDP functions and you wish to save some FLASH memory space. The default setting for this parameter is true.

INIT_DELAY – the number of milliseconds to wait for the W5100 PLL to lock after a device reset or power on.

Usually this parameter will not need to be modified; however, it allows you to configure the amount of time that the driver will wait for the W5100 internal PLL to achieve lock. This might need to be adjusted if your power supply is noisy or you are experiencing a high amount of clock jitter at power on.

TIMEOUT – The number of milliseconds to wait before an operation declares a general timeout

This parameter will allow you to adjust the number of milliseconds that an operation will wait for an operation to complete before declaring a timeout condition. This does not affect every function.

Network Configuration Parameters

The "configuration" parameters section contains the parameters for initializing the default network configuration for use by the W5100. These parameters can be overridden within your application through the use of the API function calls.

GATEWAY – The IPv4 Address of the network gateway

This parameter will allow you to specify the IP address of the Ethernet gateway router. This value is a string specified in IPv4 format of www.xxx.yyy.zzz

IP – The IPv4 Address of the device

This parameter contains the network address of the W5100 device. This address is the configured address of the controller after the API Start() call executes. Setting this will change the fixed IP of the system.

MAC – The hardware (MAC) address of the Ethernet Controller

The MAC address contains the hardware address of the system. It is expressed as a 6 byte Dash (-) or Colon (:) delimited string containing the hardware address of the W5100.

If you're using an Arduino Ethernet shield, the MAC address is printed on the sticker on the bottom of the board

MAC Address Example: 04-1A-B4-33-02-00

SUBNET_MASK – The subnet mask used for Ethernet communications

Modifying this parameter will change the subnet that the MAC will use to communicate over the network. The default subnet mask is 255.255.255.0 , meaning that for a IP address of 192.168.1.100 the MAC can communicate directly with only other IP addresses that match 192.168.1.xxx. Setting any bit in the subnet mask to a zero defines that bit as “don't care” for communications.

Hardware Configuration

This parameter section is used to define the interface parameters for associating component instances with the driver, and for declaring design specific delays and configuration data.

SPI_INSTANCE – The Instance Name of the SPI component

Note: When using the SPIM or SPI mode SCB, this driver requires at least a 4-byte FIFO buffer

Enter the component (instance) name of the SPI component that is used to communicate with the W5100. This SPI port should be configured to use 8-bit data, MSB first transmission, and SPI mode 0. The data rate is dependent upon your board layout (EMI/Noise issues), and your processor and bus clock speeds.

SS_NUM – the slave select number used to connect to the W5100 (SCB Mode)

When using the SCB component, this parameter specifies the slave select (SS) number used to communicate with the W5100 device. Valid values are from 0 to 3; values outside of this range are assumed to be 0 and will use the "ss0" pin.

Note: This component uses the internal SPI chip select generation to select the W5100.

Application Programming Interface

The functions of the Application Programming interface (API) provide the ability to configure and operate the W5100 device using your software application. The following sections describe the driver API in detail.

By default, PSoC Creator assigns the name *W5100_1* to the first instance of the driver component within your project. You may rename the component to any unique name within your project, provided it follows the syntax rules defined within PSoC Creator. The name of the instance becomes the prefix for each global identifier within the driver so that no interface of the driver will interfere with your software project. For simplicity, API references within this document will use the instance name prefix of *W5100*.

API Function	Description
W5100_Start()	Startup and initialize the device using the creator defaults
W5100_Init()	initialize device parameters and memory setup
W5100_ParseIP()	Parse an ASCII Text IPv4 address to an IPv4 Address.
W5100_ParseMAC()	Parse and ASCII Text MAC address in to an internal MAC byte array **(v1.2)**
W5100_StringIP()	Convert the current IPv4 address in to an ASCII String. **(v1.2)**
W5100_StringMAC()	Convert the current device hardware address (MAC) to an ASCII String. **(v1.2)**
W5100_SetIP()	re-assign the local IP address of the device

API Function	Description
W5100_GetIP()	Read the current IP address of the device
W5100_SetMAC()	Re-assign the hardware address (MAC) of the device
W5100_GetMAC()	Retrieve the assigned Source hardware (MAC) address of the device
W5100_SocketOpen()	Open a socket using the specified protocol on the specified port
W5100_SocketClose()	Close a previously opened socket
W5100_SocketProcessConnections()	Process the socket connection to check for errors and remote closure
W5100_SocketEstablished()	Check the connection establishment status of the socket
W5100_SocketRxDataWaiting()	Retrieve the length of waiting Receive data
W5100_TcpOpen()	Open an port using the TCP protocol
W5100_TcpStartServer()	Start a server listening for connection on an open socket
W5100_TcpStartServerWait()	Start a TCP server listening for connections on the specified socket
W5100_TcpConnect()	Open a client connection to a specified IP and port
W5100_TcpConnected()	Return the connection status of the TCP socket
W5100_TcpDisconnect()	Terminate a connection with a remote client/server
W5100_TcpSend()	Transmit a byte packet using the built-in TCP
W5100_TcpReceive()	Receive a packet of data using the built-in TCP handler
W5100_TcpPrint()	Send a zero-terminated ASCII string using TCP
W5100_UdpOpen()	Open a Socket Port using the UDP protocol
W5100_UdpSend()	Transmit a byte packet using the built-in UDP
W5100_UdpReceive()	Receive a packet of data using the built-in p handler

W5100_Start()

Startup and initialize the device using the creator defaults.

Syntax

void W5100_Start(void)

Description

This function will initialize and startup the Ethernet device chip using the default parameters supplied in the configuration window of Creator. This is usually the main method for initializing the device

This function requires that the SPI interface is initialized, however it will attempt to discover if the initialization has been completed and initialize the interface if it has not yet been setup. It is highly recommended that your application initialize the SPI interface directly rather than depend on this, since every SPI implementation may be different and your port might not be correctly initialized.

See Also

W5100_Init(), W5100_ParseIP(), W5100_SetIP(), W5100_GetIP(), W5100_SetMAC(), W5100_GetMAC()

W5100_Init()

Initialize device parameters and memory setup.

Syntax

void W5100_Init(uint8* mac, uint32 ip, uint32 subnet, uint32 gateway)

Parameters

Parameter	Description
***mac**	Pointer to a 6-byte buffer holding the device MAC address
ip	The IP address to which the device will be configured
subnet	The subnet mask to be used for the device (usually 255.255.255.0)
gateway	The IP address of the network gateway

Description

This function will reset the device, and wait for the internal PLL to lock, then initialize the device registers to allow for correct operation in your application. It currently assumes that there will be a 2K buffer for both transmit and receive for each of the 4 sockets available.

The usual method of calling W5100_Init() is from W5100_Start(). No explicit user calls are required unless the modification of the network settings beyond the component default parameters is desired by the application.

Note: Calling this function will reset all open connections.

See Also

W5100_Start(), W5100_ParseIP(), W5100_SetIP(), W5100_GetIP(), W5100_SetMAC(), W5100_GetMAC()

W5100_ParseIP()

Parse an ASCII Text IPv4 address to an IPv4 Address.

Syntax

uint32 W5100_ParseIP(const char* ipString)

Parameters

Parameter	Description
***ipString**	ASCII Z-String containing the IP address to Parse

Returns

The parsed IP address

Description

This function will parse an ASCII String IP address in to a 32-bit IP address used by the device. If the address string contains an error, this function will return an IP address of 255.255.255.255, or 0xFFFFFFFF to indicate that an error has been detected.

See Also

W5100_Start(), W5100_Init(), W5100_ParseIP(), W5100_SetIP(), W5100_GetIP(), W5100_SetMAC(), W5100_GetMAC()

W5100_ParseMAC()

Parse a MAC Address string in to a 6-byte mac address

Syntax

cystatus W5100_ParseMAC(const char* macString, uint8* mac)

Parameters

Parameter	Description
***macString**	ASCII Z-String containing the MAC address to Parse
***mac**	Pointer to a 6-byte array to hold the parsed MAC address

Returns

CYRET_SUCCESS	MAC Address Parsed successfully
CYRET_BAD_DATA	MAC Address string was formatted incorrectly

Description

This function will parse an ASCII Z-String in to a 6-byte array for assignment as the W5100 hardware address. The ASCII String should be in the format of dash (-) or colon (:) delimited data, for example "00-12-34-56-78-90" or "00:12:34:56:78:90" are valid MAC strings. Incorrectly formatted string will stop processing and return CYRET_BAD_DATA to indicate a conversion failure.

Example

```
// Define a holder for the allocated socket
result = ETH_ParseMAC("90-A2-DA-0F-61-60", &ETH_MAC[0]);
if (result == CYRET_BAD_DATA) {
        // Assign a default MAC address
}
```

See Also

W5100_StringIP(), W5100_ParseIP(), W5100_StringMAC(), W5100_SetIP(), W5100_SetMAC()

W5100_StringIP()

Read the W5100 configured IPv4 address and convert it into an ASCII String

Syntax

void W5100_StringIP(char *ipString)

Returns

Nothing

Parameters

Parameter	Description
***ipString**	ASCII Z-String to contain the read IPv4 Address

Description

StringIP() will read the currently configured IP address and convert it in to a IPv4 ASCII String for logging, or user feedback on a display.

See Also

W5100_ParseMAC(), W5100_ParseIP(), W5100_StringMAC(), W5100_SetIP(), W5100_SetMAC()

W5100_StringMAC()

Read the current hardware address registers and return as a delimited ASCII String.

Syntax

void W5100_StringMAC(char *macString)

Returns

Nothing.

Parameters

Parameter	Description
***macString**	ASCII Z-String to contain the read MAC Address

Description

StringMAC() is a function used to get the currently configured MAC address of the W5100 as an ASCII String. This is useful for logging, user feedback, and debugging.

See Also

W5100_ParseMAC(), W5100_ParseIP(), W5100_StringIP(), W5100_SetIP(), W5100_SetMAC()

W5100_SetIP()

re-assign the local IP address of the device

Syntax

uint8 W5100_SetIP(uint32 ip)

Returns

0	IP Address specified was not valid
0xFF (255)	IP Address was successfully assigned to the device.

Parameters

Parameter	Description
ip	The new IP address to which the device will be assigned.

Description

This function will re-assign the IP address of the Ethernet device to the specified address. If the address to be assigned is invalid, a zero (0) is returned from the function to indicate that a bad IP address was specified. Otherwise, 255 will be returned.

See Also

W5100_Start(), W5100_Init(), W5100_ParseIP(),W5100_GetIP(), W5100_SetMAC(), W5100_GetMAC()

W5100_GetIP()

Read the current IP address of the device.

Syntax

uint32 W5100_GetIP(void)

Returns

This function returns the IP address read from the W5100 device.

Parameters

None.

Description

This function reads and returns the contents of the Source IP register of the W5100 device.

See Also

W5100_Start(), W5100_Init(), W5100_ParseIP(), W5100_SetIP(), W5100_SetMAC(), W5100_GetMAC()

W5100_SetMAC()

Re-assign the hardware address (MAC) of the W5100 device.

Syntax

void W5100_SetMAC(uint8* mac)

Parameters

Parameter	Description
*mac	Pointer to a 6-byte array that contains the MAC value to be written

Description

This function will store the contents of the specified MAC address to the source Hardware Address register (MAC address) for the W5100 device.

See Also

W5100_Start(), W5100_Init(), W5100_ParseIP(), W5100_SetIP(), W5100_GetIP(), W5100_GetMAC()

W5100_GetMAC()

Retrieve the assigned Source hardware (MAC) address of the device.

Syntax

void W5100_GetMAC(uint8* mac)

Parameters

Parameter	Description
***mac**	Pointer to a 6-byte array to hold the read MAC value.

Description

This function will read the assigned MAC address and store it within the specified array.

See Also

W5100_Start(), W5100_Init(), W5100_ParseIP(), W5100_SetIP(), W5100_GetIP(), W5100_SetMAC

W5100_SocketOpen()

Open a socket using the specified protocol on the specified port.

Syntax

uint8 W5100_SocketOpen(uint8 Protocol, uint16 port, uint8 flags)

Parameters

Parameter	Description
Protocol	The protocol type to use for socket communications. (See Description)
port	The port number with which the opened socket will be associated
flags	Socket configuration flags (presently not used)

Returns

The socket number (0-3) of the allocated socket, or 0xFF upon error.

Description

This function will allocate and initialize a socket from the socket table and return the socket number which was opened. If there are no sockets available, or there is an error opening the socket, a value of 0xFF will be returned.

When calling this function, you should use one of the defined constants for declaring the socket protocol.

Protocol Constant	Description
W5100_PROTO_TCP	Use the W5100 Native TCP implementation
W5100_PROTO_UDP	Use the W5100 Native UDP implementation
W5100_PROTO_IP	IP mode, Reserved for future use
W5100_PROTO_MAC	MAC mode, Reserved for future use

Example

```
// Define a holder for the allocated socket
uint8 socket;
// Open and initialize a socket
socket = W5100_SocketOpen(W5100_PROTO_TCP, 23, 0);
if ( socket < 4 ) {
    // The socket was allocated Correctly… Continue
} else {
   // there was an error allocating the socket, so handle the error
}
```

See Also

W5100_SocketClose(), W5100_SocketEstablished(), W5100_SocketProcessConnections(), W5100_TcpSend(), W5100_TcpReceive(), W5100_UdpSend(), W5100_UdpReceive()

W5100_SocketClose()

Close a previously opened socket.

Syntax

Void W5100_SocketClose(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket to be closed.

Description

This function will close (and disconnect) an open socket specified as a parameter. The socket allocation record will be flushed from memory and made available for further allocation using the W5100_SocketOpen() function. If an invalid socket is specified, the function will ignore the request. Closing an already closed socket has no effect.

See Also

W5100_SocketOpen(), W5100_SocketProcessConnections(), W5100_SocketEstablished()

W5100_SocketProcessConnections()

Process the socket connection to check for errors and remote closure.

Syntax

uint8 W5100_SocketProcessConnections(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Returns

TRUE	The socket was closed
FALSE	The socket is ready for communications

Description

This function is a helper function for handling remote socket closure status that can occur during a session. It will process the opened socket to look for socket closure errors, and other aspects which would require the software to reset the socket. Upon detection of the issue, the socket will be closed and a TRUE state will be returned. When no remote closure status is detected, no action is taken.

Example

```
// Define a holder for the allocated socket
uint8 socket;
// Open and initialize a socket
socket = W5100_SocketOpen(W5100_PROTO_TCP, 23, 0);
if ( socket < 4 ) {
   // The socket was allocated Correctly… Continue
} else {
  // there was an error allocating the socket, so handle the error
}
// Start a TCP Server and wait for connection establishment
W5100_TcpStartServerWait( socket );

while( W5100_SocketEstablished( socket )  ) {
   // Communications loop … Do stuff for comms
          … Insert Comms code …
   // Process the server socket, and reset connections if closed
   if ( W5100_SocketProcessConnections( socket ) ) {
      // The socket was close, so re-open the socket connection
      socket = W5100_TcpOpen(23);
   }
}
```

See Also

W5100_SocketOpen(), W5100_SocketClose()

W5100_SocketEstablished()

Check the connection establishment (connection) status of the socket.

Syntax

Uint8 W5100_SocketEstablished(uint8 socket)

Returns

TRUE	The socket has been established
FALSE	The socket has not yet been established

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Description

This function reads the socket status register of the W5100 device and returns the state of the socket establishment.

Example

```
// wait for a connection to be established
while ( !W5100_SocketEstablished( socket )  ) {
        // Process waiting for connection
        // delay a bit
        CyDelay(1);
}
```

See Also

W5100_SocketOpen(), W5100_SocketClose()

W5100_SocketRxDataWaiting()

Retrieve the length of waiting Receive data.

Syntax

uint16 W5100_SocketRxDataWaiting(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Returns

The number of bytes that have been received by the W5100 and are waiting in the receiver buffer memory.

Description

This function will read the waiting data length from the Receive buffer and return the read length of waiting data.

W5100_TcpOpen()

Open a port using the TCP protocol.

Syntax

uint8 W5100_TcpOpen(uint16 port)

Parameters

Parameter	Description

port	The port number that the socket will be associated with.

Returns

This function returns the socket number that was opened (0 – 3) or 0xFF when an error occurs.

Description

This function will open and initialize a socket using the W5100 built-in TCP protocol, and return the socket number for the opened TCP socket. When there are no sockets available, or there is an error opening the socket, 0xFF is returned.

See Also

W5100_SocketOpen(), W5100_SocketClose()

W5100_TcpStartServer()

Start a server listening for connection on an open socket.

Syntax

void W5100_TcpStartServer(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Description

This function will execute the socket command to begin listening for connections on the specified socket. If the socket specified is not a valid socket nothing will occur. After starting the listen operation, this function will return (NON-BLOCKING).

See Also

W5100_TcpOpen(), W5100_TcpStartServerWait(), W5100_SocketClose()

W5100_TcpStartServerWait()

Start a TCP server listening for connections on the specified socket.

Syntax

void W5100_TcpStartServerWait(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Description

This function will start a valid socket listening for TCP connections by executing the listen command on the specified socket. If the socket is invalid, no action is taken. After the socket server is started, this function will wait until a connection has been made to a client before continuing.

See Also

W5100_TcpOpen(), W5100_TcpStartServer(), W5100_socketClose()

W5100_TcpConnect()

Open a client connection to a specified IP and port.

Syntax

void W5100_TcpConnect(uint8 socket, uint32 ip, uint16 port);

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
ip	The IP Address of the server to attempt a connection
port	The port number of the server

Description

This function will attempt to open a connection between a W5100 device socket, and a remote server using TCP. This function will wait for the timeout specified in the component parameters within Creator for the connection to be made before terminating the wait. While waiting for the connection establishment, the function will block.

Example

```
// Open a socket for the connection
Socket = W5100_TcpOpen(80);

// Attempt a connection with a remote server
W5100_TcpConnect(Socket, W5100_ParseIP("192.161.1.100"), 80);

// <Insert Client Code Here>
```

See Also

W5100_TcpOpen(), W5100_TcpConnected(), W5100_ParseIP(), W5100_TcpDisconnect(), W5100_TcpSend(), W5100_TcpReceive()

W5100_TcpConnected()

Return the connection status of the TCP socket.

Syntax

uint8 W5100_TcpConnected(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Returns

TRUE	The socket connection has been established
FALSE	The socket connection is not established

Description

This function will check the establishment status of the specified socket, and return the state.

See Also

W5100_TcpOpen(), W5100_TcpConnect(), W5100_TcpDisconnect()

W5100_TcpDisconnect()

Terminate a connection with a remote client/server.

Syntax

void W5100_TcpDisconnect(uint8 socket)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket

Description

This function will issue the disconnect function to initiate a connection termination between the W5100 socket and the remote client/server.

See Also

W5100_TcpConnect(), W5100_TcpConnected(), W5100_StartServer(), W5100_TcpStartServerWait(), W5100_TcpOpen()

W5100_TcpSend()

Transmit a byte packet using the built-in TCP.

Syntax

Uint16 W5100_TcpSend(uint8 socket, uint8 *buffer, uint16 len)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
***buffer**	Pointer to the byte buffer holding the data to transmit
len	The number of bytes to transmit from the buffer

Returns

This function returns the number of bytes copied from the buffer memory to the internal transmit buffer of the W5100 device.

Description

This function will copy the specified packet buffer to the W5100 Transmitter buffer, then execute the commands to transmit the data packet using the built-in TCP handlers. Upon completion of the operation, this function will return the number of bytes transmitted.

When called, this function will verify that a socket connection has first been established and is opened with the correct socket protocol. Send operations to sockets that contain a different protocol or are not yet established are ignored and 0 is returned.

See Also

W5100_TcpOpen(), W5100_TcpConnect(), W5100_TcpStartServer(), W5100_TcpStartServerWait(), W5100_TcpPrint(), W5100_TcpReceive()

W5100_TcpReceive()

Receive a packet of data using the built-in TCP handler.

Syntax

uint16 W5100_TcpReceive(uint8 socket, uint8 *buffer, uint16 length)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
***buffer**	Pointer to the byte buffer to receive the data from the W5100 device
length	The maximum amount of data to be received in to the buffer

Returns

This function returns the length of data copied from the W5100 device's internal buffer to the buffer memory.

Description

This function will check for available received data, then copy the data from the internal W5100 buffer to the specified holding buffer for the received data. When there is more data waiting than available space in the buffer (specified by the length parameter), this function will only receive up to the maximum length specified.

Prior to receiving data, this function will verify that a valid connection has been established, and that the configured protocol is set to the internal TCP. When not properly configured, this function will return 0, otherwise, the number of bytes read from the W5100 receive buffer memory is returned.

See Also

W5100_TcpOpen(), W5100_TcpConnect(), W5100_TcpStartServer(), W5100_TcpStartServerWait(), W5100_TcpPrint(), W5100_TcpSend()

W5100_TcpPrint()

Send a zero-terminated ASCII string using TCP.

Syntax

Void W5100_TcpPrint(uint8 socket, const char *str)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
***str**	Pointer to the ASCII X-String to send

Description

This function is a shortcut to using the W5100_TcpSend() to transmit a zero-terminated ASCII (ASCII-Z) string to a remote client/server.

Calling this function is the same as:

*W1500_TcpSend(socket, (const char *) &str[0], strlen((char *) &str[0]);*

See Also

W5100_TcpOpen(), W5100_TcpConnect(), W5100_TcpStartServer(), W5100_TcpStartServerWait(), W5100_TcpSend()

W5100_UdpOpen()

Open a socket port using the UDP protocol.

Syntax

Uint8 W5100_UdpOpen(uint16 port)

Parameters

Parameter	Description
port	The port number that the socket will be associated with.

Returns

This function returns the socket number that was opened (0 – 3) or 0xFF when an error occurs.

Description

This function will open and initialize a socket using the W5100 built-in UDP protocol, and return the socket number for the opened UDP socket. When there are no sockets available, or there is an error opening the socket, 0xFF is returned.

See Also

W5100_SocketOpen(), W5100_SocketClose(), W5100_TcpOpen()

W5100_UdpSend()

Transmit a byte packet using the built-in UDP.

Syntax

Uint16 W5100_UdpSend(uint8 socket, uint32 ip, uint16 port, uint8 *buffer, uint16 length)

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
ip	The IP address of the destination
port	The port number of the datagram destination
***buffer**	Pointer to the byte buffer containing the datagram data
length	The number of bytes to transmit in the user datagram.

Returns

This function returns the number of bytes copied from the user datagram buffer to the internal transmit buffer of the W5100.

Description

This function will copy the specified packet buffer to the W5100 Transmitter buffer, then execute the commands to transmit the data packet using the built-in UDP handlers. Upon completion of the operation, this function will return the number of bytes transmitted.

When called, this function will verify that a socket connection has first been opened with the correct socket protocol. Send operations to sockets that contain a different protocol are ignored and 0 is returned.

See Also

W5100_UdpReceive(), W5100_UdpOpen()

W5100_UdpReceive()

Receive a packet of data using the built-in UDP handler.

Syntax

```
Uint16 W5100_UdpReceive( uin8 socket, uint32 *ip, uint16 *port,
                          uint8 *buffer, uint16 length)
```

Parameters

Parameter	Description
socket	The socket number (0-3) of the socket
***ip**	Pointer to a buffer to hold the sender's IP address
***port**	Pointer to a buffer to hold the sender's port number
***buffer**	Pointer to the byte buffer to receive the data from the W5100 device
length	The maximum amount of data to be received in to the buffer

Returns

This function returns the actual length of data copied from the W5100 device's internal receiver buffer to the buffer memory.

Description

This function will check for available received data, and then copy the data from the internal W5100 buffer to the specified holding buffer for the received data. When there is more data waiting than available space in the buffer (specified by the length parameter), this function will only receive up to the maximum length specified.

Prior to receiving data, this function will verify that the configured protocol is set to the internal UDP. When not properly configured, this function will return 0, otherwise, the number of bytes read from the W5100 receive buffer memory is returned.

See Also

W5100_UdpSend(), W5100_UdpOpen()

API Memory Usage

	PSoC 3		PSoC 5/5LP		PSoC 4	
W5100 Driver	**FLASH**	**SRAM**	**FLASH**	**SRAM**	**FLASH**	**SRAM**
Using SCB	N/A	N/A	N/A	N/A	4462	32
Using SPIM			3792	40	4110	20
TCP Functions			696	0	808	0
UDP Functions			432	0	568	0

- PSoC 4 Memory size determined using PSoC4 CY8C4245AXI-483
- PSoC5 Memory size determined using PSoC5LP CY8C5868AXI-LP035
- PSoC3 Memory size determined using PSoC3 CY8C3…

Additional Resources

Please refer to the documents listed below for more information related to the W5100 device. Additional information and application notes can be obtained from the WIZnet website http://www.wiznt.co.kr.

Document	Version	Location
W5100 Datasheet	1.2.4	W5100_Datasheet_v1.2.4.pdf
W5100 Reference Schematics		W5100_Ref_sch_MAG_R2.1.pdf
W5100 Errata Sheet	2.4	3150Aplus_5100_errata_en_v2.4.pdf

Hardware/Software Notes

When working with the W5100 it is important that the 25 MHz reference is applied to the device before the device is initialized using the W5100_Start() function. This can be important when sourcing the reference clock from another device such as an FPGA. The

W5100 initialization must be delayed until after the reference clock is stable to prevent communications issues on the Ethernet.

If the reference clock is removed from the W5100, it is advisable to re-initialize the W5100 device to prevent communication issues. Unfortunately, this action will cause the W5100 to disconnect any open connections, requiring this consideration to be me when writing application software in an environment where the clock may be removed from the device.

Project team, Credits and Thanks

The following individuals were involved in the development, testing and support of this component:

Name	Notes
Chuck Erhardt	Project lead, author of original component code and interface implementation of W5100 interface.

Component Versions

This section contains the versions of and major modifications to the W5100 interface driver component.

Version	Description of Changes	Reason for changes
1.0	Initial Release	N/A
1.1	Removed READ_WRITE_DELAY	Caused data rate to decrease when user did not manually configure to a value that represented the SPI data rate
1.1	Removed "inline" keywords	PSoC 3 Keil compiler does not support inline functions; GCC ignores them without a special command line option set that is not default.
1.1	Updated internal receive data process	W5100 RxDataPointer register was not updated correctly during a packet data read.
1.2	Added MAC Parsing & String IP/MAC Functions	Improve usability of the component API and configuration API. Also, added feedback and debugging capabilities.
1.2	Updated Tx/Rx Buffer Pointer address calculations	When using multiple sockets it was discovered that the buffer pointers could become corrupted for sockets 1 through 3.

Latest version

The latest version of the driver can be downloaded from the GIT repository at:

https://github.com/e2forlife/PSoC-W5100-Driver

Roadmap

Below are described features which are not yet included in the driver, but are desired to be included at a later date. Since there is limited time to work on this driver, it is difficult to forecast the time frame for the inclusion of the features. Check http://www.e2forlife.com for more information on the status of any of these implementations, or to provide feedback related to bug-fixes or suggestions of additional features or improvements to make the driver better.

Feature	Description
DHCP	Add Support for the Dynamic Host Configuration Protocol (ref: WIZnet app-note for implementation of DHCP)
Custom "Customizer"	Add a C# Customizer to replace the default Creator Customizer
HTTP (web) Server	Add the ability to serve embedded web sites

Copyright

Disclaimer of Warranty

Limitation of Liability

Interpretation