W5500 Interface Driver for PSoC 4/5LP

Version 1.0, 23-June-2015

Features

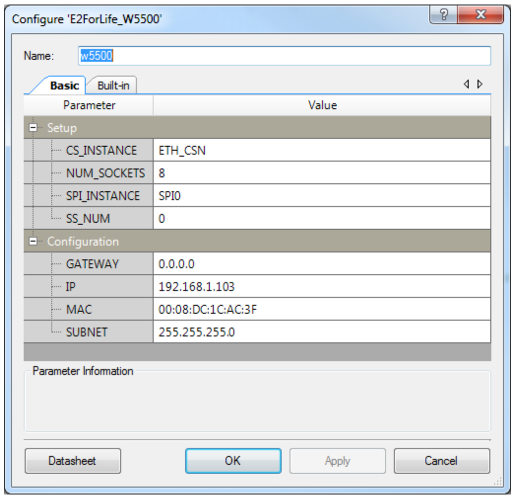
- Powerful API that is simple to use.
- Hardware independent design.
- Uses SPIM or SCB mode based on design.
- Support for up to 8 simultaneous Sockets.
- TCP & UDP Support.
- API functions for general utility support.
- Fully customizable through software or Customizer.

Description

This component provides the software interface for the initialization, control, and communication over Ethernet using the W5500 device from WizNET. The driver API supports up to 8 sockets at the same time, with a 2K Tx/Rx FIFO buffers of off-chip RAM (within the W5500). The API is designed to be simple to use, yet powerful enough to support complex applications without burdening the application designer with complex configurations when not required by the end-use application.

Component Parameters

Drag the E2ForLife_W5500 component onto your design and double-click to open the **Configure** dialog.



Name: w5500

Parameter	Value
Setup	
CS_INSTANCE	ETH_CSN
NUM_SOCKETS	8
SPI_INSTANCE	SPI0
SS_NUM	0
Configuration	
GATEWAY	0.0.0.0
IP	192.168.1.103
MAC	00:08:DC:1C:AC:3F
SUBNET	255.255.255.0

The configuration dialog allows the instance name of the component to be set as well as providing configuration options for connected peripherals as well as default configurations for the W5500 device.

Setup Section

The setup section contains the parameters used to connect the W5500 driver to the SPI hardware block and the chip select block.

CS_INSTANCE

This parameter must contain the instance name of the component or pin used to control the chip select of the W5500.

NUM_SOCKETS
This parameter is used to set the number of available sockets for the driver to allocate. The W5500 device has a maximum of 8 sockets.

SPI_INSTANCE
The SPI_INSTANCE parameter must contain the instance name of the SPI device used to communicate with the W5500. It can be either a SCB or a SPIM device within your design.

SS_NUM
The SS_NUM parameter contains the index of the output used for the generation of the chip select. When using only an output pin to control the chip select to the W5500, set this parameter to 0, otherwise set it to the control register output number used to connect to the chip select output pin.

Configuration Section
The configuration section contains parameters used by Start() to set the default configuration of the W5500 device. API calls are available to initialize the W5500 with a different configuration.

GATEWAY
This parameter is 4 decimal numbers between 0 and 255 separated by dots (.) and used to set the default network gateway address of the W5500 device. The default value is “0.0.0.0”.

IP
This parameter is 4 decimal numbers between 0 and 255 separated by dots (.) and used to set the default IPv4 address of the W5500 device (your board’s IP Address). The default value is “192.168.1.101”.

MAC
This parameter is 6 hexadecimal bytes separated by colons (:) and is used to set the default hardware address (MAC Address) of the W5500 device (your board’s MAC or NIC Address). The default value is “00:de:ad:be:ef:00”.

SUBNET
This parameter is 4 decimal numbers between 0 and 255 separated by dots (.) and used to set the default subnet mask of the W5500 device. The default value is “255.255.255.0”.

Component usage
This component contains no hardware and does not utilize UDB’s or fixed function peripherals, rather it is intended to be used in conjunction with hardware to promote SPI port sharing among multiple devices. Configuration of these devices is highly important, since the W5500 interface has no control over the implementation of the interfaces outside of the component. To add the driver component to your design, locate it in the component catalog and drag it on to the schematic sheet. The component can be found in the catalog under: **Community/Communications/Ethernet/E2ForLife_W5500**

Installation

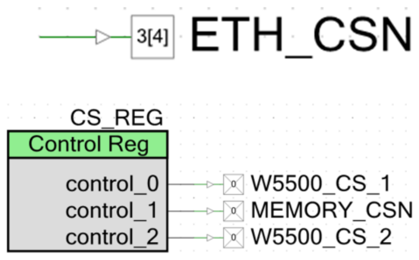
By default, this community component is not visible within the component catalog, thus it must be installed by adding it in to the application project in which it will be used. Alternatively, is can be added to a component library that is referenced by the application project. Once installed the component is available within the component catalog as specified above.

Installation in to a project

From PSoC Creator, select the component tab of the workspace explorer to work with the component of a project. Then, from the "Project" menu, select the drop-down option "Import Component…" to open the component import dialog.

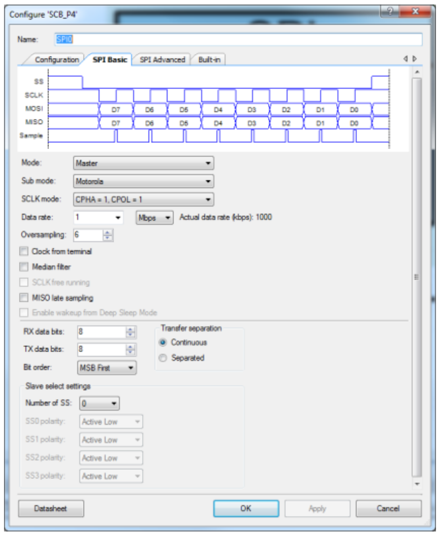
Once the dialog has opened, select the option "Import from Archive", and using the "browse" button select the W5500 component archive. Set the target project (library or application) for the import to be the project to which the component will be installed and click OK to import the component and make it available from the component catalog.

Configuration of Chip Select hardware



Chip selects can be configured as either single-pin outputs or as an output of a chip select register. **Do not connect the chip select for the W5500 to the SPI device generated chip select**. When configuring the component to interface with either the single-pin or register-based ship selects, set the **CS_INSTANCE** parameter to the **name** of the device used to control the chip select output. For single-pin usage this is the name of the pin, for register-based implementations this is the name of the **register**. The SS_NUM paramter should be **set to 0 for single-pin** implementations, or set to the number of the register output (bit number) when using a register.

Configration of SPI hardware



Name: SPI0
Mode: Master
Sub mode: Motorola
SCLK mode: CPHA = 1, CPOL = 1
Data rate: 1 Mbps Actual data rate (kbps): 1000
Oversampling: 6
☐ Clock from terminal
☐ Median filter
☐ SCLK free running
☐ MISO late sampling
☐ Enable wakeup from Deep Sleep Mode
RX data bits: 8
TX data bits: 8
Bit order: MSB First
Transfer separation: [✓ Continuous] Separated
Slave select settings
Number of SS: 0
SS0 polarity: Active Low
SS1 polarity: Active Low
SS2 polarity: Active Low
SS3 polarity: Active Low

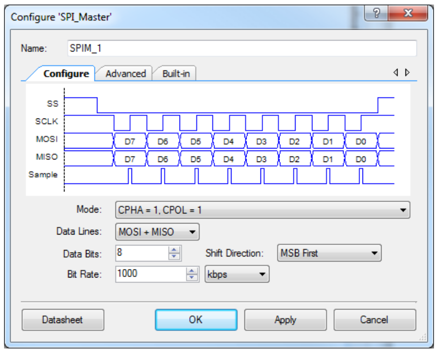
The W5500 driver can interface with the W5500 device using either a SPIM (SPI Master) or a SCB block. The type of interface is automatically determined by the driver interface and no configuration is required. Since the SPI component is not a portion of this driver, it must be configured properly to provide an interface compatible with the W5500 device.

SCB Configuration

Set the Mode to "Motorola", and the SCLK mode to (CPHA = 1, CPOL = 1) or (CPHA =0, CPOL = 0) to enable MODE 0 or MODE 3 SPI communications. Change your data rate to your applications requirements; however, the W5500 max operating SCLK frequency is 33 MHz (per datasheet). The SCB does not need to be configured to generate SS signals. In applications requiring the use of SCB generated SS signals, make sure that the W5500 device is allocated an SS to prevent issues when accessing the device.

SPIM Configuration

Set the Mode to (CPHA =1, CPOL = 1) or (CPHA = 0, CPOL = 0) to enable SPI MODE 0 or MODE 3 interfacing. Select full-duplex operation by setting the data lines to (MOSI + MISO). Use 8-bits per transfer, and select a data rate base on your applications requirements, keeping in mind that the max SCLK frequency supported by the W5500 is 33 MHz.



Name: SPIM_1
Mode: CPHA = 1, CPOL = 1
Data Lines: MOSI + MISO
Data Bits: 8
Shift Direction: MSB First
Bit Rate: 1000 kbps

Application Programming Interface

Application Programming Interface (API) routines allow your software to configure and control the W5500 device. The driver API has been functionally divided to provide more logical interfacing with the W5500.

Component API

The W5500 interface driver's component API has been divided in to sections for general/utility functions, socket functions, TCP functions and UDP functions. Where possible, the naming convention used for the API functions also identifies the type of function or section that is associated with the operation.

By default, PSoC Creator assigns the instance name "W5500_1" to the first instance of this component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "W5500".

The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

Function	Description
W5500_Send()	Chip-level Send/Receive API for direct register access
W5500_Start()	Initialize and enable the Software API and W5500 device
W5500_StartEx()	Initialize and enable the Software API and W5500 device using custom configuration parameters.
W5500_Init()	Initialize the registers of the W5500.
W5500_GetMAC()	Read the MAC (Source Hardware) Address from the W5500
W5500_GetIP()	Read the IPv4 Address from the W5500
W5500_GetTxLength()	Read the buffer length available for writing during transmit
W5500_WriteTxData()	Send data block to the W5500 transmit buffer memory.
W5500_RxDataReady()	Read the number of bytes waiting in the Socket receive buffer
W5500_TxBufferFree()	Read the number of free bytes in the socket Transmit buffer.

Function	Description
W5500_SocketOpen()	Open and initialize a socket.
W5500_SocketClose()	Close an open socket connection.
W5500_SocketDisconnect()	Disconnect and close an open socket connection.
W5500_SocketSendComplete()	Read the status of the W5500 transmit operation.
W5500_ExecuteSocketCommand()	Send a command to the W5500 and wait for completion.
W5500_TcpConnected()	Read the status of the TCP connection for a socket.
W5500_TcpOpenClient()	Open a socket and connect to a remote server using TCP.
W5500_TcpOpenServer()	Open a socket and enable the W5500 to receive connections.
W5500_TcpWaitForConnection()	Wait for a valid connection from a client or to a server.
W5500_TcpSend()	Send a data block using a TCP socket.
W5500_TcpPrint()	Send an ASCII-Z String using TCP.
W5500_TcpReceive()	Receive a block of data from a TCP Socket.
W5500_TcpGetChar()	Wait for, and receive a single character from a TCP socket.
W5500_TcpGetLine()	Read ASCII data from a TCP socket until a newline is received.
W5500_UdpOpen()	Open a socket and initialize it for UDP operation.
W5500_UdpSend()	Send a datagram to a port/destination using UDP
W5500_UdpReceive()	Receive a datagram on a port using UDP
W5500_ParseIP()	Parse an ASCII-Z string and extract and IP address
W5500_ParseMAC()	Parse an ASCII-Z string an extract a MAC address
W5500_StringMAC()	Convert a binary MAC Address in to an ASCII-Z string
W5500_StringIP()	Convert a binary IP address in to an ASCII-Z string
W5500_Base64Encode()	Base 64 encode a block of data.
W5500_Base64Decode()	Decode a base 64 encoded block of data.
W5500_IPADDRESS()	Convert 4-bytes in to a binary IP address.

void W5500_Send()

Description: Access the internal register or buffer memory for reading/writing data to the device. It is recommended that device access use other API calls when available, as this function directly interfaces with the W5500 device and may result in accidental register corruption that may disable Ethernet communications.

Parameters: *uint16* **offset** – The offset address to begin writing.

uint8 **block_select** – The memory block being accessed

W5500_BLOCK_COMMON	W5500 Common Registers
W5500_BLOCK_Sn_REG	W5500 Socket Registers, Socket n
W5500_BLOCK_Sn_TXB	W5500 Transmit buffer, Socket n
W5500_BLOCK_Sn_RXB	W5500 Receive Buffer, Socket n

uint8 **write** – Non-Zero == Write data to W5500, 0 == Read Data from W5500

*uint8** **buffer** – Pointer to the data to be written, or the buffer to hold data read from the W5500 device.

uint16 **length** – the total number of bytes to read from or write to the W5500 device.

Return Value: None

Side Effects: This function Reads/Writes from/to the internal chip registers of the W5500, improper usage could cause the device to operate improperly.

cystatus W5500_Start()

Description:	Initialize the W5500 device to the default component parameters specified in the configuration dialog window, and enables the device for normal operation.
Parameters:	None
Return Value:	**CYRET_TIMEOUT** : W5500 was not able to be properly initialized. **CYRET_SUCCESS** : W5500 device was successfully initialized.
Side Effects:	This function resets and closes all open sockets. All internal data and configurations are overwritten with the default configuration.

cystatus W5500_StartEx()

Description:	Initialize the W5500 device to the values specified as parameters when calling the function. For simplicity, the Ethernet parameters are passed as pointers to ASCII-Z strings.
Parameters:	*const char** **gateway** – The IP address of the gateway router (ASCII-Z) *const char** **subnet** – The Subnet Mask for the device (ASCII-Z) *const char** **mac** – The hardware address of the W5500 device (ASCII-Z) *const char** **ip** – The IP address to which the W5500 will be assigned (ASCII-Z)
Return Value:	**CYRET_TIMEOUT** : W5500 was not able to be properly initialized. **CYRET_SUCCESS** : W5500 device was successfully initialized.
Side Effects:	This function resets and closes all open sockets. All internal data and configurations are overwritten with the default configuration.

cystatus W5500_Init()

Description:	Initialize the W5500 device to the default values specified as parameters to the function.
Parameters:	uint8* **gateway** – pointer to a 4-byte array containing the gateway address uint8* **subnet** – pointer to a 4-byte array containing the subnet mask. uint8* **mac** – pointer to a 6-byte array containing the hardware address of the W5500. uint8* **ip** – pointer to a 4-byte array containing the IP address to assign to the W5500.
Return Value:	**CYRET_BAD_DATA** : Data was improperly written to the W5500 **CYRET_SUCCESS** : W5500 was successfully initialized.
Side Effects:	This function resets and closes all open sockets. All internal data and configurations are overwritten with the default configuration.

void W5500_GetMAC()

Description:	Read the hardware address programmed in to the internal source hardware address of the W5500.
Parameters:	*uint8** **mac** – pointer to a 6-byte buffer that will hold the MAC address read from the W5500 device. (*Output parameter*)
Return Value:	None
Side Effects:	None

uint32 W5500_GetIP()

Description: Read the Source IP that the W5500 is currently using and returns the value as a 32-bit IP Address.
Parameters: None
Return Value: *uint32* – 32-bit IP address (4-bytes)
Side Effects: None

uint16 W5500_GetTxLength()

Description: Request a block of data from the transmit buffer for a socket, and optionally wait for the room to be available.
Parameters: *uint8* **Socket** – The open socket buffer to access
uint16 **Length** – The length of data requested from the buffer
uint8 **flags** – configuration of the function operation.

W5500_TXRX_FLG_WAIT	Set this flag to wait until the requested data is available in the buffer.

Return Value: *uint16* – The number of bytes allocated for the transmit function.
Side Effects: None

cystatus W5500_WriteTxData()

Description: Write a block of data into the specified socket transmit buffer memory.
Parameters: *uint8* **Socket** – The socket on which data will be placed in the transmit buffer
*uint8** **Buffer** – Pointer to the block of data to transmit.
uint16 **Length** – Length of the data to send.
uint8 **flags** – Flags to control the operation of the function.

W5500_TXRX_FLG_WAIT	Wait until the complete data buffer can be transmitted.

Return Value: *cystatus* – function status

CYRET_BAD_PARAM	Invalid Socket
CYRET_STARTED	Transfer not yet complete
CYRET_CANCELED	Socket was closed before complete transfer
CYRET_FINISHED	Transfer is complete

Side Effects: W5500 Interrupt register is cleared after execution.

uint16 W5500_RxDataReady()

Description: Read the number of bytes received and waiting in the socket receive buffer.
Parameters: uint8 **Socket** – Socket to check received data length.
Return Value: *uint16* – The number of bytes in the Rx buffer memory for the socket.
Side Effects: None

uint16 W5500_TxBufferFree()

Description: Read the number of bytes available in the socket transmit buffer.
Parameters: *uint8* **Socket** – Socket to check the remaining length of Tx buffer.
Return Value: *uint16* – The number of bytes remaining in the Tx buffer memory for the socket.
Side Effects: None

cystatus W5500_SocketClose()

Description: Close an open socket, optionally sending a disconnect packet.

Parameters: *uint8* **Socket** – Socket to close/disconnect

uint8 **Discon** – Set to non-zero value to send a disconnect packet before closing the socket.

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_CANCELLED	Socket was already closed
CYRET_SUCCESS	Socket was closed successfully.

Side Effects: None

cystatus W5500_SocketDisconnect()

Description: Close an open socket, sending a disconnect packet.

Parameters: *uint8* **Socket** – Socket to close/disconnect

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_CANCELLED	Socket was already closed
CYRET_SUCCESS	Socket was closed successfully.

Side Effects: None

uint8 W5500_SocketOpen()

Description: Open and initialize a socket for communication.

Parameters: *uint16* **Port** – Ethernet port on which the socket will be opened.

uint8 **Flags** – Socket configuration and other flags.

W5500_PROTO_CLOSED	Closed Socket
W5500_PROTO_TCP	Open a TCP socket
W5500_PROTO_UDP	Open a UDP socket
W5500_PROTO_MACRAW	Open a socket with no protocol processing. Must be socket 0
W5500_FLG_SKT_UDP_MULTICAST_ENABLE	Enable Multicast transfers for UDP
W5500_FLG_SKT_MACRAW_MAC_FILT_ENABLE	
W5500_FLG_SKT_BLOCK_BROADCAST	Block broadcast messages
W5500_FLG_SKT_ND_ACK	
W5500_FLG_SKT_UDP_IGMP_V1	
W5500_FLG_SKT_MACRAW_MULTICAST_BLOCK	
W5500_FLG_SKT_UDP_BLOCK_UNICAST	
W5500_FLG_SKT_MACRAW_IPV6_BLOCKING	Block IPV6 transfers for MACRAW mode.

Return Value: *uint8* – The socket handle for the opened and initialized socket, or 0xFF for an invalid or failed socket open.

Side Effects: Opening a socket opens the next available socket, and will not release the socket allocation until a Close operation is performed. Use W5500_SocketClose to free socket allocations.

cystatus W5500_SocketSendComplete()

Description: Read the socket transmit state to determine if a SEND operation has completed.

Parameters: *uint8* **Socket** – Socket to check for transmit status

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_CANCELLED	Socket was closed while sending data
CYRET_FINISHED	Data transfer was complete

Side Effects: Socket Interrupt register is cleared while executing this function.

cystatus W5500_ExecuteSocketCommand()

Description: Execute a W5500 socket command and wait for completion.

Parameters: *uint8* **Socket** – Socket on which command will be executed.

uint8 **cmd** – Command to be executed.

W5500_CR_OPEN	Open and initialize socket with the options defined in the mode register.
W5500_CR_LISTEN	**TCP ONLY** – Start an open TCP mode socket listening for client connections (Start Server)
W5500_CR_CONNECT	**TCP ONLY** – Connect to a remote server using parameters initialized in socket registers.
W5500_CR_DISCON	**TCP ONLY** – Send a FIN/ACK packet and close connection.
W5500_CR_CLOSE	Close specified socket.
W5500_CR_SEND	Transmit all data in the socket Tx buffer.
W5500_CR_SEND_MAC	**UDP ONLY** – Transmit all data within the socket Tx buffer without using ARP.
W5500_CR_SEND_KEEP	**TCP ONLY** – Check the connection status by sending a 1-byte keep-alive packet.
W5500_CR_RECV	Complete the receive processing on a socket

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_TIMEOUT	Timeout occurred while executing the command
CYRET_SUCCESS	Command executed successfully.

Side Effects: W5500 internal socket state may change when executing some commands.

cystatus W5500_TcpConnected()

Description: Check the connection status of a TCP socket.

Parameters: *uint8* **Socket** – Socket on which connection status will be tested.

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_STARTED	Socket is still waiting for connection.
CYRET_SUCCESS	Socket connection is established.

Side Effects: None

uint8 W5500_TcpOpenClient()

Description: Open a TCP connection to a remote server. While waiting for the client connection to complete, this call will BLOCK until either the connection has been established or a timeout has occurred.

Parameters: *uint16* **Port** – Ethernet port on which the socket will be opened.
uint32 **remote_ip** – IP Address of remote server.
uint16 **remote_port** – port number for the remote server connection.

Return Value: *uint8* – The socket handle for the opened and initialized socket, or 0xFF for an invalid or failed socket open.

Side Effects: Opening a client connection will open an initialize a socket, and wait until the connection has been established or timed out.

uint8 W5500_TcpOpenServer()

Description: Open a socket using TCP and start a server listening.

Parameters: *uint16* **Port** – Ethernet port on which the socket will be opened.

Return Value: *uint8* – The socket handle for the opened and initialized socket, or 0xFF for an invalid or failed socket open.

Side Effects: This function opens and initializes a socket. See W5500_SocketOpen()

cystatus W5500_TcpWaitForConnection()

Description: Test a socket connection status and wait until a valid connection has been established. Note that this is a blocking call that will wait until the connection has been established. See W5500_TcpConnected() to determine the connection status without blocking.

Parameters: *uint8* **Socket** – a valid open socket to be used to wait for a connection.

Return Value: *cystatus – Operation status.*

CYRET_BAD_PARAM	Invalid or unallocated socket
CYRET_SUCCESS	Socket connection is established.

Side Effects: None

uint16 W5500_TcpSend()

Description: Send a block of data on a socket using TCP.

Parameters: *uint8* **Socket** – Valid open socket used for transmission
*uint8** **Buffer** – pointer to the block of data to transmit.
uint16 **Length** – the length size of the data block in bytes.
uint8 **flags** – Flags for altering the operation of the function

W5500_TXRX_FLG_WAIT	Wait for enough free tx buffer space to send the entire data block.

Return Value: *uint16* – The total number of bytes placed in the W5500 transmit buffer.

Side Effects: None.

void W5500_TcpPrint()

Description: Transmit an ASCII-Z string using TCP

Parameters: *uint8* **Socket** – A Valid, open socket to send data
*const char** **str** – the ASCII-Z string to send

Return Value: none

Side Effects:

uint16 W5500_TcpReceive()

Description: Receive a block of data from a socket using TCP.

Parameters: *uint8* **Socket** – Valid open socket
*uint8** **Buffer** – pointer to the buffer to hold the data.
uint16 **Length** – The maximum size of the received data block.
uint8 **flags** – Flags for altering the operation of the function

W5500_TXRX_FLG_WAIT	Wait until the requested maximum bytes are available before reading data from the W5500.

Return Value: *uint16* – The total number of bytes read from the W5500 receive buffer.

Side Effects: None.

char W5500_TcpGetChar()

Description: wait for data to be available and Read a character from the receive buffer

Parameters: *uint8* **Socket** – A Valid, open socket on which data is expected

Return Value: *char* – The data element read from the socket receive buffer.

Side Effects: Updates receive buffer read pointer.

int W5500_TcpGetLine()

Description: Read data from the W5500 and store in a buffer until a CR or LF has been read from the receive data stream.

Parameters: *uint8* **Socket** – A Valid, open socket on which data is expected
*char** **Buffer** – the memory space used to receive the data from the W5500

Return Value: *int* – The length of the data read from the W5500.

Side Effects: Updates receive buffer read pointer.

uint16 W5500_UdpSend()

Description: Send a block of data on a socket using UDP.

Parameters: *uint8* **Socket** – Valid open socket used for transmission
*uint8** **Buffer** – pointer to the block of data to transmit.
uint16 **Length** – the length size of the data block in bytes.
uint8 **flags** – Flags for altering the operation of the function

W5500_TXRX_FLG_WAIT	Wait for enough free tx buffer space to send the entire data block.

Return Value: *uint16* – The total number of bytes placed in the W5500 transmit buffer.

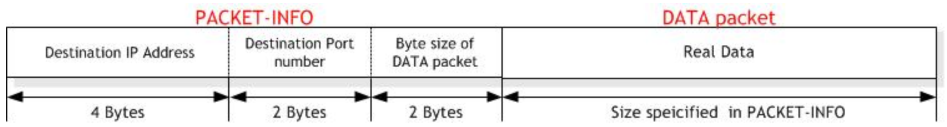
Side Effects: None.

uint16 W5500_UdpReceive()

Description:	Receive a block of data from a socket using UDP.
Parameters:	*uint8* **Socket** – Valid open socket
	*uint8** **Header** – pointer to an 8-Byte buffer to hold the UDP header
	*uint8** **Buffer** – pointer to the buffer to hold the data.
	uint16 **Length** – The maximum size of the received data block.
	uint8 **flags** – Flags for altering the operation of the function

W5500_TXRX_FLG_WAIT	Wait until the requested maximum bytes are available before reading data from the W5500.

Return Value:	*uint16* – The total number of bytes read from the W5500 receive buffer.
Side Effects:	None.
Notes:	



The UDP header consists of the 4-byte IP address of the sender, followed by the 2-byte port number, and the length of the data payload. “Real Data” is copied in to the buffer and is not part of the header.

uint8 W5500_UdpOpen()

Description:	Open and initialize a socket using unicast/broadcast UDP
Parameters:	*uint16* **Port** – Ethernet port on which the socket will be opened.
Return Value:	*uint8* – The socket handle for the opened and initialized socket, or 0xFF for an invalid or failed socket open.
Side Effects:	This function opens a socket. see W5500_SocketOpen()

uint32 W5500_ParseIP()

Description:	Convert an ASCII string to an IP address.
Parameters:	*const char** **ipString** – IP address specified using xxx.xxx.xxx.xxx formatting
Return Value:	*uint32* – The converted IP address or 0xFFFFFFFF upon error.
Side Effects:	none.

cystatus W5500_ParseMAC()

Description:	Convert an ASCII string into a MAC (hardware) address. This function convets the mac address in the format of hh:hh:hh:hh:hh:hh in to a 6-byte array containing the binary MAC address.
Parameters:	*const char** **macString** – ASCII string containing the MAC address
	*uint8** **mac** – 6-Byte array to contain the result of the conversion.
Return Value:	*cystatus* – The status of the conversion

CYRET_SUCCESS	Valid conversion, conversion successful
CYRET_BAD_DATA	MAC String was not converted properly.

Side Effects:	none.

void W5500_StringMAC()

Description:	Convert a 6-byte MAC (hardware) address in to an ASCII String
Parameters:	*uint8** **mac** – pointer to a 6-byte MAC address.
	*const char** **macString** – String to hold the ASCII MAC address
Return Value:	None
Side Effects:	macString will contain the converted value of the MAC address.

void W5500_StringIP()

Description:	Convert an IP address in to an ASCII String
Parameters:	*uint32* **ip** – IP address to be converted
	*const char** **ipString** – String to hold the ASCII IP address
Return Value:	None
Side Effects:	ipString will contain the converted value of the IP address.

Applications Information

Adding Ethernet support to your project using the PSoC + W5500 is as simple as adding the SPI component and then dragging the W5500 interface component on to the schematic. After some simple configuration to setup the interface between the W5500 driver and the SPI component, all that is required is to start the driver, open a socket, and send/receive some packets. The following sections contain some useful information for building Ethernet applications using this driver and the W5500 device.

Driver Initialization

Initializing the driver (and W5500 device) can be as simple as calling W5500_Start(), or for applications requiring a more flexible interface, calling the W5500_StartEx() with the configuration parameters set to the custom configuration.

Using the custom configuration can be helpful when building applications that load the IP, MAC, Gateway, and Subnet mask from internal or external non-volatile memory (NVM). The device configuration can be read from a software-defined source then passed to the driver initialization using W5500_StartEx().

Example 1: Basic Initialization of the W5500

```
/* SPI Interface Initialization  */
/* Initialize the W5500 device */
if ( W5500_Start() != CYRET_SUCCESS) {
   /* An Error has occurred */
}
```

Example 2: Custom Initialization of the W5500

```
/* SPI Interface Initialization */
/* Read parameters from NVM */
MAC = EEPROM_ReadString( MAC_ADDRESS );
IP = EEPROM_ReadString( IP_ADDRESS );
Subnet = EEPROM_ReadString( SUBNET_ADDRESS );
Gateway = EEPROM_ReadString( GATEWAY_ADDRESS );
/* Since the data was read from the NVM, Initialize the W5500 */
W5500_StartEx(Gateway,Subnet,MAC,IP);
```

Sockets

The W5500 device communicates using sockets. Sockets are opened with a protocol, and can be connectionless or client/server based, usually depending on the protocol being used by the socket. Presently, this driver supports UDP and TCP based communications using the built in protocol stack of the W5500. MAC RAW mode communications are possible, however at present, only the low-level chip access is available for this special mode of operation.

In order to access Ethernet messaging, an application must first open a socket and declare the source port and protocol for the connection. This can be achieved by using the Socket API function W5500_SocketOpen(), or by using the simpler protocol API functions W5500_UdpOpen(), W5500_TcpOpenServer() or W5500_TcpOpenClient.

Opening TCP Connections

Since TCP is a connection-based protocol, once a socket is open, something meaningful must be done in order to actually start using the socket. For this reason, it is recommended that sockets using TCP be opened with either W5500_TcpOpenClient or W5500_TcpOpenServer. Using these function will greatly simplify the software, since they are designed to open the socket, and initiate either a connection to a remote server, or to start listening for connections from a remote client.

Sending Data using TCP

TCP data is transmitted over the Ethernet using the built-in TCP protocol stack within the W5500. Applications transmitting data should use the W5500_TcpSend function from the TCP API, or one of the several helper functions that were designed to handle specific types of data. Please refer to the API documentation section for information regarding the use of any API calls to send data.

Using UDP Sockets

UDP is a connectionless protocol, meaning that there is no connection required between clients and sever, rather messages are just transmitted from a local source port to a remote destination port, thus, a socket must only be opened in order to use the socket for communications. The UDP API call W5500_UdpOpen is a shorthand and more straightforward version of using the more complex Socket API call to W5500_SocketOpen.

More Information

More information can be obtained by visiting http://www.e2forlife.com. Are you using this driver in a unique way? Do you have some ideas on additional features? We are interested in hearing from you… Please send all feature requests, bug reports, applications questions, and applications information to E2forlife.com.

Revision History

Name	Version	Description
C. Erhardt	1.0	Initial Release candidate of the W5500 driver & API documentation.

License

Made on a Mac

The component and this documentation were made using an Apple Macbook Pro