MC9S12NE64 OpenTCP Reference Manual

Rev. July 19, 2004

TABLE OF CONTENTS

OpenTCP TCP/IP Stack for MC9S12NE64 Data Structure Documentation

arp_entry Struct Reference

```
#include <arp.h>
```

Detailed Description

ARP packet header fields.

This structure contains various fields used for managing ARP cache.

Definition at line 81 of file arp.h.

Data Fields

- UINT8 state
  State of this ARP cache entry.

- UINT8 type
  Type of this ARP cache entry.

- UINT8 retries
- UINT8 ttl
- UINT8 hwadr [MAXHWALEN]
- UINT32 pradr

Field Documentation

UINT8 arp_entry::hwadr[MAXHWALEN]

Hardware Address that is received either as an ARP reply or by caching the address of a received IP packet

Definition at line 110 of file arp.h.

UINT32 arp_entry::pradr

Protocol Address (IPv4 protocol assumed)

Definition at line 114 of file arp.h.

UINT8 arp_entry::retries

Number of retries left (how many ARP requests more will be sent in order to try to resolve the IP address)

Definition at line 103 of file arp.h.

UINT8 arp_entry::state

State of this ARP cache entry.

Holds information about the state of this ARP cache entry. Can hold one of the following values:

- ARP_FREE - entry unused and available
- ARP_RESERVED - entry reserved by arp_alloc call
- ARP_PENDING - waiting for ARP reply to get the HW address
- ARP_RESOLVED - entry resolved and HW address available

Definition at line 92 of file arp.h.

UINT8 arp_entry::ttl

Time To Live value for this cache entry

Definition at line 107 of file arp.h.

UINT8 arp_entry::type

Type of this ARP cache entry.

Type of the entry defines what the ARP cache manager will do after the TTL period. Can be one of the following:

- ARP_FIXED_IP - ARP cache entry is refreshed after TTL
- ARP_TEMP_IP - ARP cache entry is deleted after TTL

Definition at line 102 of file arp.h.

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

- arp.h

ethernet_frame Struct Reference

```
#include <ethernet.h>
```

Detailed Description

Ethernet packet header fields.

This structure holds information about the Ethernet packets. In addition to standard Ethernet header (destination HW address, source HW address, frame size and protocol), buff_index is added. This variable is used by higher level protocols (IP, ARP or other) to initialize reading of the Ethernet packet by invoking NETWORK_RECEIVE_INITIALIZE macro to initialize reading of the data carried in the Ethernet packet (not the Ethernet header itself!).

Definition at line 36 of file ethernet.h.

Data Fields

- UINT8 destination [ETH_ADDRESS_LEN]
- UINT8 source [ETH_ADDRESS_LEN]
- UINT16 frame_size
- UINT16 protocol
- UINT16 buf_index

Field Documentation

UINT16 ethernet_frame::buf_index

Address in the Ethernet controllers buffer where data can be read from

Definition at line 55 of file ethernet.h.

UINT8 ethernet_frame::destination[ETH_ADDRESS_LEN]

destination hardware address as read from the received ethernet packet

Definition at line 38 of file ethernet.h.

UINT16 ethernet_frame::frame_size

size of the received Ethernet packet

Definition at line 46 of file ethernet.h.

UINT16 ethernet_frame::protocol

protocol field of the Ethernet header. For now we work with:

- PROTOCOL_IP - 0x0800
- PROTOCOL_ARP - 0x0806

Definition at line 49 of file ethernet.h.

UINT8 ethernet_frame::source[ETH_ADDRESS_LEN]

source hardware address as read from the received ethernet packet

Definition at line 42 of file ethernet.h.

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

- ethernet.h

---

http_server_state Struct Reference

```
#include <http_server.h>
```

Detailed Description

Structure that holds all the necessary state information for session management.

All the necessary information for HTTP session state management by the HTTP server is stored here. See individual field documentation for more info.

Definition at line 42 of file http_server.h.

Data Fields

- UINT8 state
  Session state.

- UINT8 ownersocket
  TCP socket used for TCP communication.

- UINT32 fstart
  File start.

- UINT32 flen
  File length.

- UINT32 fpoint
  File pointer.

- UINT16 funacked
  Number of unacknowledged HTTP bytes previously sent.

Field Documentation

UINT32 http_server_state::flen

File length.

This variable holds file length information. It is used by the HTTP server to determine when the entire file has been sent.

Definition at line 80 of file http_server.h.

UINT32 http_server_state::fpoint

File pointer.

Pointer to a current position inside the file that is beeing sent over the appropriate HTTP session.

Definition at line 87 of file http_server.h.

UINT32 http_server_state::fstart

File start.

This variable holds information about the file start address. This is highlyconfiguration-dependant (file system chosen, etc..)

File address can not start from zero!!! (Data won't be sent by HTTP server in this case)

Definition at line 72 of file http_server.h.

UINT16 http_server_state::funacked

Number of unacknowledged HTTP bytes previously sent.

This variable holds information about the number of previously sent and still unacknowledged bytes. This is needed to reliably determine, in case data needs to be regenerated, how much bytes to regenerate or, in case data has been acknowledged, how much to advance the fpoint variable.

Definition at line 97 of file http_server.h.

UINT8 http_server_state::ownersocket

TCP socket used for TCP communication.

This variable holds a handle to TCP socket that is used to achieve data transfer.

Definition at line 59 of file http_server.h.

UINT8 http_server_state::state

Session state.

This variable holds current sessions' state which can be one of the following:

- HTTPS_STATE_FREE
- HTTPS_STATE_RESERVED
- HTTPS_STATE_ACTIVE

Definition at line 52 of file http_server.h.

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

- http_server.h

---

ip_frame Struct Reference

```
#include <ip.h>
```

Detailed Description

IP datagram header fields.

This structure is used for holding information about various fields of the IPv4 header. In addition to standard IP header, buf_index variable has been added to store the information about the buffer address in the Ethernet controller from where upper layer protocols (such as TCP, UDP or some other) can start reading their data. This is initialized by invoking NETWORK_RECEIVED_INITIALIZE macro with appropriate buf_index value.

For detailed explanation of the IPv4 header fields refer to RFC791.

Definition at line 110 of file ip.h.

Data Fields

- UINT8 vihl

- UINT8 tos
- UINT16 tlen
- UINT16 id
- UINT16 frags
- UINT8 ttl
- UINT8 protocol
- UINT16 checksum
- UINT32 sip
- UINT32 dip
- UINT8 opt [MAX_IP_OPTLEN+1]
- UINT16 buf_index

Field Documentation

UINT16 ip_frame::buf_index

Next offset from the start of network buffer

Definition at line 123 of file ip.h.

UINT16 ip_frame::checksum

Header Checksum

Definition at line 119 of file ip.h.

UINT32 ip_frame::dip

Destination IP address

Definition at line 121 of file ip.h.

UINT16 ip_frame::frags

Flags & Fragment offsett

Definition at line 116 of file ip.h.

UINT16 ip_frame::id

IP Identification number

Definition at line 115 of file ip.h.

UINT8 ip_frame::opt[MAX_IP_OPTLEN + 1]

Option field

Definition at line 122 of file ip.h.

UINT8 ip_frame::protocol

Protocol over IP

Definition at line 118 of file ip.h.

UINT32 ip_frame::sip

Source IP address

Definition at line 120 of file ip.h.

UINT16 ip_frame::tlen

Total Length

Definition at line 114 of file ip.h.

UINT8 ip_frame::tos

Type Of Service

Definition at line 113 of file ip.h.

UINT8 ip_frame::ttl

Time to live

Definition at line 117 of file ip.h.

UINT8 ip_frame::vihl

Version & Header Length field

Definition at line 112 of file ip.h.

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

- ip.h

MBUF Struct Reference

```
#include <mBuf.h>
```

Data Fields

- INT16 status

- INT16 len
- UINT8 * working_ptr
- UINT8 * data

Field Documentation

UINT8* MBUF::data

Definition at line 28 of file mBuf.h.

INT16 MBUF::len

Definition at line 26 of file mBuf.h.

INT16 MBUF::status

Definition at line 25 of file mBuf.h.

UINT8* MBUF::working_ptr

Definition at line 27 of file mBuf.h.

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

- mBuf.h

---

netif Struct Reference

```
#include <system.h>
```

Detailed Description

Network Interface declaration.

This structure holds information about the network interface. This means that all of the network-related information are stored in this kind of structure.

Definition at line 42 of file system.h.

Data Fields

- LWORD localip
  IP address of a device.

- BYTE localHW [6]
  Ethernet address given to a device.

- LWORD defgw
  Default network gateway.

- LWORD netmask
  Network submask.

Field Documentation

LWORD netif::defgw

Default network gateway.

IP address of a default network gateway. This is needed if the device is to communicate with the outside network (Internet) and not only intranet.

Definition at line 79 of file system.h.

BYTE netif::localHW[6]

Ethernet address given to a device.

This array holds an Ethernet address assigned to a device. Note that these must be unique so if you're shipping your product to outside world you must purchase sufficient address range.

Definition at line 71 of file system.h.

LWORD netif::localip

IP address of a device.

IP address of a happy device using OpenTCP :-). This must hold proper-value IP address in order for the networking stuff to work.

Possible scenarios for filling this field are:

- By assigning static IP address to a device always after reset
- By allowing user to choose IP address by some tool (e.g. through serial communication, storing that information to some external flash,...)
- By using BOOTP or DHCP clients for obtaining dynamically assigned address
- By obtaining the IP address from the first ICMP packet the device receives

First three approaches can also be used for obtaining gateway and subnet-mask information.

Definition at line 62 of file system.h.

LWORD netif::netmask

Network submask.

Network submask. Also needed if the the device is to communicate with the outside network. Used when determining whether the host we're sending some data to is on the local network (send data directly) or not (send through gateway).

Definition at line 88 of file system.h.

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

- system.h

---

pop3c_struct Struct Reference

```
#include <pop3_client.h>
```

Detailed Description

POP3 client structure.

As expected this structure holds the fields that are needed for proper operation of the POP3 client. Refer to documentation of the fields to get more information about them.

Definition at line 85 of file pop3_client.h.

Data Fields

- UINT8 state
- UINT32 remip
- UINT16 remport
- INT8 sochandle
- UINT8 tmrhandle
- UINT8 unacked
- UINT16 msgtotal
- UINT16 curmsgindex
- UINT32 curmsgtotlen
- UINT16 curmsghlen
- UINT8 headerbuf [9]
- UINT8 charsinheaderbuf

- UINT8 from [POP3C_SENDERMAXLEN]
- UINT8 subject [POP3C_SUBJECTMAXLEN]

Field Documentation

UINT8 pop3c_struct::charsinheaderbuf

Number of valid chars in headerbuf

Definition at line 98 of file pop3_client.h.

UINT16 pop3c_struct::curmsghlen

Header length of current message

Definition at line 96 of file pop3_client.h.

UINT16 pop3c_struct::curmsgindex

Index of current message

Definition at line 94 of file pop3_client.h.

UINT32 pop3c_struct::curmsgtotlen

Total length of current message

Definition at line 95 of file pop3_client.h.

UINT8 pop3c_struct::from[POP3C_SENDERMAXLEN]

Sender of E-mail

Definition at line 99 of file pop3_client.h.

UINT8 pop3c_struct::headerbuf[9]

Used to parse from,to,subject

Definition at line 97 of file pop3_client.h.

UINT16 pop3c_struct::msgtotal

Number of messages in message box

Definition at line 93 of file pop3_client.h.

UINT32 pop3c_struct::remip

Remote IP of POP3 server

Definition at line 88 of file pop3_client.h.

UINT16 pop3c_struct::remport

Remote port of POP3 server

Definition at line 89 of file pop3_client.h.

INT8 pop3c_struct::sochandle

Handle to TCP socket

Definition at line 90 of file pop3_client.h.

UINT8 pop3c_struct::state

State of POP3 client state machine

Definition at line 87 of file pop3_client.h.

UINT8 pop3c_struct::subject[POP3C_SUBJECTMAXLEN]

Subject of E-mail

Definition at line 100 of file pop3_client.h.

UINT8 pop3c_struct::tmrhandle

Handle to timer

Definition at line 91 of file pop3_client.h.

UINT8 pop3c_struct::unacked

Do we have unacked data or not?

Definition at line 92 of file pop3_client.h.

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

- pop3_client.h

---

tcb Struct Reference

```
#include <tcp_ip.h>
```

Detailed Description

TCP transmission control block.

This structure holds various fields used to keep track of TCP socket states, settings and event listener function. It is needed to ensure proper operation of TCP state machine and TCP connections based on it.

Definition at line 589 of file tcp_ip.h.

Data Fields

- UINT8 state
  State of the TCP socket [entry].

- UINT8 type
  type of the TCP socket

- UINT8 flags
- UINT32 rem_ip
- UINT16 remport
- UINT16 locport
- UINT32 send_unacked
- UINT8 myflags
- UINT32 send_next
- UINT16 send_mtu
- UINT16 tout
- UINT8 tos
- UINT32 receive_next
- UINT16 persist_timerh
- UINT16 retransmit_timerh
- UINT8 retries_left
- INT32(* event_listener)(INT8, UINT8, UINT32, UINT32)
  TCP socket application event listener.

Field Documentation

INT32(* tcb::event_listener)(INT8, UINT8, UINT32, UINT32)

TCP socket application event listener.

Pointer to an event listener - a callback function used by TCP/IP stack to notify application about certain events.

UINT8 tcb::flags

State machine flags

Definition at line 622 of file tcp_ip.h.

UINT16 tcb::locport

Local TCP port

Definition at line 625 of file tcp_ip.h.

UINT8 tcb::myflags

My flags to be Txed

Definition at line 627 of file tcp_ip.h.

UINT16 tcb::persist_timerh

Persistent timers' handle

Definition at line 633 of file tcp_ip.h.

UINT32 tcb::receive_next

Definition at line 632 of file tcp_ip.h.

UINT32 tcb::rem_ip

Remote IP address

Definition at line 623 of file tcp_ip.h.

UINT16 tcb::remport

Remote TCP port

Definition at line 624 of file tcp_ip.h.

UINT16 tcb::retransmit_timerh

Retransmission timers' handle

Definition at line 634 of file tcp_ip.h.

UINT8 tcb::retries_left

Number of retries left before aborting

Definition at line 635 of file tcp_ip.h.

UINT16 tcb::send_mtu

Definition at line 629 of file tcp_ip.h.

UINT32 tcb::send_next

Definition at line 628 of file tcp_ip.h.

UINT32 tcb::send_unacked

Definition at line 626 of file tcp_ip.h.

UINT8 tcb::state

State of the TCP socket [entry].

This variable holds information used by the OpenTCP to manage sockets as well as information needed to manage TCP connection. Possible values are:

- TCP_STATE_FREE
- TCP_STATE_RESERVED
- TCP_STATE_CLOSED
- TCP_STATE_LISTENING
- TCP_STATE_SYN_RECEIVED
- TCP_STATE_SYN_SENT
- TCP_STATE_FINW1
- TCP_STATE_FINW2
- TCP_STATE_CLOSING
- TCP_STATE_LAST_ACK
- TCP_STATE_TIMED_WAIT
- TCP_STATE_CONNECTED

Definition at line 609 of file tcp_ip.h.

UINT8 tcb::tos

Type of service allocated

Definition at line 631 of file tcp_ip.h.

UINT16 tcb::tout

Socket idle timeout (seconds)

Definition at line 630 of file tcp_ip.h.

UINT8 tcb::type

type of the TCP socket

Defines type of the TCP socket allocated. This determines how connection is established/closed in some cases. Possible values are:

- TCP_TYPE_NONE
- TCP_TYPE_SERVER
- TCP_TYPE_CLIENT
- TCP_TYPE_CLIENT_SERVER

Definition at line 621 of file tcp_ip.h.

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

- tcp_ip.h

---

tcp_frame Struct Reference

```
#include <tcp_ip.h>
```

Detailed Description

TCP header information.

This structure holds header fields from the received TCP packet.

In addition to standard header fields, buf_index field has been added allowing applications to re-read the received data many times by reinitializing reading based on the address stored in this field.

Definition at line 564 of file tcp_ip.h.

Data Fields

- UINT16 sport
- UINT16 dport
- UINT32 seqno
- UINT32 ackno
- UINT16 hlen_flags
- UINT16 window
- UINT16 checksum
- UINT16 urgent
- UINT8 opt [MAX_TCP_OPTLEN+1]
- UINT16 buf_index

Field Documentation

UINT32 tcp_frame::ackno

Acknowledgement number

Definition at line 569 of file tcp_ip.h.

UINT16 tcp_frame::buf_index

Next offset from the start of network buffer

Definition at line 575 of file tcp_ip.h.

UINT16 tcp_frame::checksum

TCP packet checksum

Definition at line 572 of file tcp_ip.h.

UINT16 tcp_frame::dport

Destination port

Definition at line 567 of file tcp_ip.h.

UINT16 tcp_frame::hlen_flags

Header length and flags

Definition at line 570 of file tcp_ip.h.

UINT8 tcp_frame::opt[MAX_TCP_OPTLEN + 1]

Option field

Definition at line 574 of file tcp_ip.h.

UINT32 tcp_frame::seqno

Sequence number

Definition at line 568 of file tcp_ip.h.

UINT16 tcp_frame::sport

Source port

Definition at line 566 of file tcp_ip.h.

UINT16 tcp_frame::urgent

Urgent pointer

Definition at line 573 of file tcp_ip.h.

UINT16 tcp_frame::window

Size of window

Definition at line 571 of file tcp_ip.h.

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

- tcp_ip.h

TEthernetFrame Struct Reference

```
#include <ne64api.h>
```

Data Fields

- UINT8 destination [ETH_ADDRS_LEN]
- UINT8 source [ETH_ADDRS_LEN]
- UINT16 frame_size
- UINT16 protocol
- UINT16 buf_index

Field Documentation

UINT16 TEthernetFrame::buf_index

Address in the Ethernet controllers buffer where data can be read from

Definition at line 37 of file ne64api.h.

UINT8 TEthernetFrame::destination[ETH_ADDRS_LEN]

destination hardware address as read from the received ethernet packet

Definition at line 24 of file ne64api.h.

UINT16 TEthernetFrame::frame_size

size of the received Ethernet packet

Definition at line 30 of file ne64api.h.

UINT16 TEthernetFrame::protocol

protocol field of the Ethernet header. For now we work with:

- PROTOCOL_IP - 0x0800
- PROTOCOL_ARP - 0x0806

Definition at line 32 of file ne64api.h.

UINT8 TEthernetFrame::source[ETH_ADDRS_LEN]

source hardware address as read from the received ethernet packet

Definition at line 27 of file ne64api.h.

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

- ne64api.h

TFileEntry Struct Reference

```
#include <FileSys.h>
```

Data Fields

- unsigned char hash
- const unsigned char * file_start_address
- unsigned short file_length

Field Documentation

unsigned short TFileEntry::file_length

Definition at line 20 of file FileSys.h.

const unsigned char* TFileEntry::file_start_address

Definition at line 19 of file FileSys.h.

unsigned char TFileEntry::hash

Definition at line 18 of file FileSys.h.

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

- FileSys.h

---

tFRHEAD Struct Reference

Data Fields

- tU08 da [6]
- tU08 sa [6]
- tU16 ft

Field Documentation

tU08 tFRHEAD::da[6]

destination address

Definition at line 39 of file ne64driver.c.

tU16 tFRHEAD::ft

frame type

Definition at line 41 of file ne64driver.c.

tU08 tFRHEAD::sa[6]

source address

Definition at line 40 of file ne64driver.c.

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

- ne64driver.c

---

ucb Struct Reference

```
#include <tcp_ip.h>
```

Detailed Description

UDP control block.

This structure holds various fields used to keep track of UDP socket states, settings and event listener function.

Definition at line 506 of file tcp_ip.h.

Data Fields

- UINT8 state
  State of socket entry.

- UINT8 tos
  Type of service allocated for a socket.

- UINT16 locport
- UINT8 opts
  Socket options.

- INT8
  UDP socket application event listener.

- UINT8
  UDP socket application event listener.

- UINT32
  UDP socket application event listener.

- UINT16
  UDP socket application event listener.

Field Documentation

ucb::INT8

UDP socket application event listener.

Pointer to a event listener - a callback function used by TCP/IP stack to notify application about certain events.

Definition at line 547 of file tcp_ip.h.

UINT16 ucb::locport

Local UDP port of Socket

Definition at line 525 of file tcp_ip.h.

UINT8 ucb::opts

Socket options.

Currently, this holds information about checksum calculation options. Can be one of the following:

- UDP_OPT_NONE - cheksum calculation not performed
- UDP_OPT_SEND_CS - checksum is calculated for outgoing UDP packets
- UDP_OPT_CHECK_CS - checksum is checked for incoming UDP packets
- UDP_OPT_SEND_CS | UDP_OPT_CHECK_CS - both checksum calculations are enabled

Definition at line 539 of file tcp_ip.h.

UINT8 ucb::state

State of socket entry.

This variable holds state of a particular UDP socket entry in the UDP socket table. Following values are possible:

- UDP_STATE_FREE
- UDP_STATE_CLOSED
- UDP_STATE_OPENED

Definition at line 516 of file tcp_ip.h.

UINT8 ucb::tos

Type of service allocated for a socket.

For now no services implemented so this value is not important.

Definition at line 523 of file tcp_ip.h.

ucb::UINT16

UDP socket application event listener.

Pointer to a event listener - a callback function used by TCP/IP stack to notify application about certain events.

Definition at line 547 of file tcp_ip.h.

ucb::UINT32

UDP socket application event listener.

Pointer to a event listener - a callback function used by TCP/IP stack to notify application about certain events.

Definition at line 547 of file tcp_ip.h.

ucb::UINT8

UDP socket application event listener.

Pointer to a event listener - a callback function used by TCP/IP stack to notify application about certain events.

Definition at line 547 of file tcp_ip.h.

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

- tcp_ip.h

---

udp_frame Struct Reference

```
#include <tcp_ip.h>
```

Detailed Description

UDP header information.

This structures' fields are used to hold information about the headers of the received UDP packet.

In addition to standard UDP header fields, buf_index field has been added allowing applications to re-read the received data many times by reinitializing reading based on the address stored in this field.

Definition at line 488 of file tcp_ip.h.

Data Fields

- UINT16 sport
- UINT16 dport
- UINT16 tlen
- UINT16 checksum
- UINT16 buf_index

Field Documentation

UINT16 udp_frame::buf_index

Data offsett from the start of network buffer

Definition at line 494 of file tcp_ip.h.

UINT16 udp_frame::checksum

UDP checksum

Definition at line 493 of file tcp_ip.h.

UINT16 udp_frame::dport

Destination port

Definition at line 491 of file tcp_ip.h.

UINT16 udp_frame::sport

Source port

Definition at line 490 of file tcp_ip.h.

UINT16 udp_frame::tlen

total len (UDP part)

Definition at line 492 of file tcp_ip.h.

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

- tcp_ip.h

---

uMACADUnion Union Reference

Data Fields

- tU16 Word [3]
- tU08 Byte [6]

Field Documentation

tU08 uMACADUnion::Byte[6]

Definition at line 34 of file ne64driver.c.

tU16 uMACADUnion::Word[3]

Definition at line 33 of file ne64driver.c.

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

- ne64driver.c

---

uMCHASHUnion Union Reference

Data Fields

- tU16 Word [4]
- tU08 Byte [8]

Field Documentation

tU08 uMCHASHUnion::Byte[8]

Definition at line 28 of file ne64driver.c.

tU16 uMCHASHUnion::Word[4]

Definition at line 27 of file ne64driver.c.

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

- ne64driver.c

OpenTCP TCP/IP Stack for MC9S12NE64 File Documentation

address.c File Reference

```
#include "MOTTYPES.h"
```

Variables

- const tU08 hard_addr [6] = { 0x01, 0x23, 0x45, 0x56, 0x78, 0x9a }
- const tU08 prot_addr [4] = { 192, 168, 2, 3 }
- const tU08 netw_mask [4] = { 255, 255, 255, 0 }
- const tU08 dfgw_addr [4] = { 192, 168, 2, 1 }
- const tU08 brcs_addr [4] = { 192, 168, 2, 255 }

Variable Documentation

const tU08 brcs_addr[4] = { 192, 168, 2, 255 }

IP subnet broadcast

Definition at line 22 of file address.c.

const tU08 dfgw_addr[4] = { 192, 168, 2, 1 }

subnet default gateway IP addr

Definition at line 21 of file address.c.

const tU08 hard_addr[6] = { 0x01, 0x23, 0x45, 0x56, 0x78, 0x9a }

HW addr of our device

Definition at line 17 of file address.c.

const tU08 netw_mask[4] = { 255, 255, 255, 0 }

network mask

Definition at line 20 of file address.c.

const tU08 prot_addr[4] = { 192, 168, 2, 3 }

IP addr of our device

Definition at line 19 of file address.c.

---

address.h File Reference

Defines

- #define ip_address prot_addr
- #define ip_netmask netw_mask
- #define ip_gateway dfgw_addr

Variables

- const tU08 hard_addr []
- const tU08 prot_addr []
- const tU08 brcs_addr []
- const tU08 netw_mask []
- const tU08 dfgw_addr []

Define Documentation

#define ip_address prot_addr

Definition at line 25 of file address.h.

#define ip_gateway dfgw_addr

Definition at line 27 of file address.h.

#define ip_netmask netw_mask

Definition at line 26 of file address.h.

Variable Documentation

const tU08 brcs_addr[]

IP subnet broadcast

Definition at line 21 of file address.h.

const tU08 dfgw_addr[]

subnet default gateway IP addr

Definition at line 23 of file address.h.

const tU08 hard_addr[]

HW addr of our device

Definition at line 19 of file address.h.

const tU08 netw_mask[]

network mask

Definition at line 22 of file address.h.

const tU08 prot_addr[]

IP addr of our device

Definition at line 20 of file address.h.

---

arp.c File Reference

```
#include "debug.h"
#include "ethernet.h"
#include "arp.h"
#include "timers.h"
```

Functions

- UINT8 process_arp (struct ethernet_frame *frame)

Process and analyze the received ARP packet.

- void arp_send_response (void)
  Send response to an ARP request.

- void arp_get_response (void)
  Extract data from the received ARP packet.

- void arp_send_req (UINT8 entry)
  Send ARP request based on information in an ARP cache table.

- INT8 arp_alloc (UINT8 type)
  Allocate ARP entry in ARP cache table.

- INT8 arp_add (UINT32 pra, UINT8 *hwadr, UINT8 type)
  Add given IP address and MAC address to ARP cache.

- arp_entry * arp_find (LWORD pra, struct netif *machine, UINT8 type)
  Find an ARP entry given a protocol address.

- void arp_manage (void)
  Manage ARP cache periodically.

- void arp_init (void)
  Initialize data structures for ARP processing.

- BYTE is_subnet (LWORD ipadr, struct netif *machine)
  Checks if a given IP address belongs to the subnet of a given machine.

Variables

- arp_entry arp_table [ARP_TSIZE]
  ARP cache table holding ARP_TSIZE cache values.

- UINT8 arp_timer
  ARP timer handle used for measuring timeouts, doing retransmissions,..

Function Documentation

INT8 arp_add (UINT32 *pra*, UINT8 * *hwadr*, UINT8 *type*)

Add given IP address and MAC address to ARP cache.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Parameters:

pra - protocol address (assumed IPv4)
hwadr - pointer to Ethernet MAC address (6 bytes)
type - type of address allocated if not found. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

- 0 - Address already in cache. Refreshed.
- 1 - New entry in ARP cache created

New IP address is added to ARP cache based on the information supplied to function as parameters.

Definition at line 501 of file arp.c.

INT8 arp_alloc (UINT8 *type*)

Allocate ARP entry in ARP cache table.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

1.11.2001

Parameters:

type Type of ARP cache entry beeing allocated. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

>=0 - pointer to allocated ARP entry (actaully index in the ARP cache table)

Allocate arp entry for given type. Chooses the unused entry if one exists. Otherwice deletes entries in round-robin fashion.

Definition at line 415 of file arp.c.

struct arp_entry* arp_find (LWORD *pra*, struct netif * *machine*, UINT8 *type*)

Find an ARP entry given a protocol address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

01.11.2001

Parameters:

pra - Protocol address (IPv4)
machine - Pointer to configuration of network interface used
type - Type of address allocated if not found. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

- 0 - ARP entry not found or not ready yet (waiting for ARP response)
- struct arp_entry* - pointer to solved entry of ARP cache table

This function tries to resolve IPv4 address by checking the ARP cache table and sending ARP requested if needed.

Definition at line 588 of file arp.c.

void arp_get_response (void)

Extract data from the received ARP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Warning:

- This function starts reading data from Ethernet controller without initializing it for reading it first, so NIC must already be initialized for reading from correct address (it expects ar$sha field from ARP packet immediately)

This function is invoked from process_arp() function when ARP reply packet is detected. Basic checking is performed to see if the packet is intended for us, and if it is, ARP cache table is checked and corresponding entry is refreshed (resolved).

Definition at line 245 of file arp.c.

void arp_init (void)

Initialize data structures for ARP processing.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

01.11.2001

Warning:

- Invoke this function at start-up to properly initialize ARP cache subsystem.

Call this function to properly initialize ARP cache table and so that ARP allocates and initializes a timer for it's use.

Definition at line 859 of file arp.c.

void arp_manage (void)

Manage ARP cache periodically.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

04.11.2001

Warning:

- Invoke this function periodically to ensure proper ARP cache behaviour

Iterate through ARP cache aging entries. If timed-out entry is found, remove it (dynamic address) or update it (static address). This function must be called periodically by the system.

Definition at line 734 of file arp.c.

void arp_send_req (UINT8 *entry*)

Send ARP request based on information in an ARP cache table.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

1.11.2001

Parameters:

entry Index of ARP cache entry that is beeing resolved

Invoked from arp_find() and arp_manage() functions, arp_send_request creates ARP request packet based on data stored in the ARP cache entry who's index is given as a parameter.

Definition at line 332 of file arp.c.

void arp_send_response (void)

Send response to an ARP request.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Warning:

- This function starts reading data from Ethernet controller without initializing it for reading it first, so NIC must already be initialized for reading from correct address (it expects ar$sha field from ARP packet immediately)

This function is invoked from process_arp() function in order to send a reply to an ARP request. First, incoming packet is checked to see if it is intended for us or not. If not, function does not do anything. Otherwise, ARP reply packet is formed and sent.

Definition at line 120 of file arp.c.

BYTE is_subnet (LWORD *ipadr*, struct netif * *machine*)

Checks if a given IP address belongs to the subnet of a given machine.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

05.11.2001

Parameters:

ipadr - IP address under check
machine - pointer to configuration of network parameters used

Returns:

- TRUE - ipadr belongs to subnet of given machine
- FALSE - ipadr is NOT a part of subnet of given machine

Based on information supplied in ipadr and machine parameters this function performs basic check if IP address is on the same subnet as the one defined for the machine.

Definition at line 909 of file arp.c.

UINT8 process_arp (struct ethernet_frame * *frame*)

Process and analyze the received ARP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Parameters:

frame Pointer to ethernet_frame structure containing information about the received frame

Returns:

Return TRUE if Ethernet frame processed held ARP packet, otherwise FALSE.

Invoke process_arp function whenever ARP packet is received (see main_demo.c for an example loop). This function will process the received packet, analyze it'c content briefly and perform on of the two possible actions:

- If the received packet is ARP request it will invoke arp_send_reply in order to send ARP reply back
- If the received packet is ARP response it will iterate through the cache table and try to find ARP entry that is beeing resolved or refreshed

Definition at line 45 of file arp.c.

Variable Documentation

struct arp_entry arp_table[ARP_TSIZE]

ARP cache table holding ARP_TSIZE cache values.

ARP cache table is an array of arp_entry structures holding all of the necessary information about the state, timeouts and hardware/IP addresses of individual entries. By modifying the ARP_TSIZE, cache size can be changed and thus RAM memory occupied by the ARP cache significantly reduced or increased. See arp_entry definition for more information about struct fields.

Definition at line 15 of file arp.c.

UINT8 arp_timer

ARP timer handle used for measuring timeouts, doing retransmissions,..

ARP module uses this timer handle to detect that a certain period of time has expired (defined by the value of ARP_MANG_TOUT) and that cache entries should be examined to see what to do with them.

Definition at line 23 of file arp.c.

---

arp.h File Reference

```
#include "datatypes.h"
```

Data Structures

- struct arp_entry
  ARP packet header fields.

Defines

- #define MAXHWALEN 6
- #define MAXPRALEN 4
- #define ARP_TSIZE 10
  ARP cache size (number of entries).

- #define ARP_TIMEOUT 60
  ARP cache entry refresh period (in seconds).

- #define ARP_RESEND 2
  ARP Request resend period (in seconds).

- #define ARP_MAXRETRY 5
  Number of IP address resolving retires.

- #define AR_HARDWARE 0x0001
- #define ARP_ETHCODE 0x0806
- #define ARP_REQUEST 1
- #define ARP_REPLY 2
- #define ARP_MANG_TOUT 1
- #define ARP_FREE 0
- #define ARP_RESERVED 1
- #define ARP_PENDING 2
- #define ARP_RESOLVED 3
- #define ARP_REFRESHING 4
- #define ARP_FIXED_IP 0
- #define ARP_TEMP_IP 1

Functions

- void arp_init (void)
  Initialize data structures for ARP processing.

- arp_entry * arpfind (LWORD, struct netif *, UINT8)
- INT8 arp_alloc (UINT8)
  Allocate ARP entry in ARP cache table.

- void arp_send_req (UINT8)
  Send ARP request based on information in an ARP cache table.

- arp_entry * arp_find (LWORD, struct netif *, UINT8)
  Find an ARP entry given a protocol address.

- void arp_manage (void)
  Manage ARP cache periodically.

- BYTE is_subnet (LWORD, struct netif *)
  Checks if a given IP address belongs to the subnet of a given machine.

- BYTE process_arp (struct ethernet_frame *)
  Process and analyze the received ARP packet.

- void arp_send_response (void)
  Send response to an ARP request.

- void arp_get_response (void)
  Extract data from the received ARP packet.

- void arp_send_request (void)
- INT8 arp_add (UINT32, UINT8 *, UINT8)
  Add given IP address and MAC address to ARP cache.

Define Documentation

#define AR_HARDWARE 0x0001

Definition at line 66 of file arp.h.

#define ARP_ETHCODE 0x0806

Definition at line 67 of file arp.h.

#define ARP_FIXED_IP 0

For Fixed addresses like GW. Entry is refreshed after ttl

Definition at line 130 of file arp.h.

#define ARP_FREE 0

Entry is Unused (initial value)

Definition at line 120 of file arp.h.

#define ARP_MANG_TOUT 1

Definition at line 72 of file arp.h.

#define ARP_MAXRETRY 5

Number of IP address resolving retires.

Change this number to change number of times ARP module will resend ARP requests before giving up (if no ARP reply is received).

Definition at line 61 of file arp.h.

#define ARP_PENDING 2

Entry is used but incomplete

Definition at line 122 of file arp.h.

#define ARP_REFRESHING 4

Entry is being refreshed

Definition at line 124 of file arp.h.

#define ARP_REPLY 2

Definition at line 70 of file arp.h.

#define ARP_REQUEST 1

Definition at line 69 of file arp.h.

#define ARP_RESEND 2

ARP Request resend period (in seconds).

Change this number to determine how quickly will ARP module issue ARP requests.

Changing this values changes the amount time that will elapse before ARP module resends it's ARP request in case no response has been received. Change this number according to expected network latency and desired resolving speed.

Definition at line 52 of file arp.h.

#define ARP_RESERVED 1

Entry is reserved (allocated)

Definition at line 121 of file arp.h.

#define ARP_RESOLVED 3

Entry has been resolved

Definition at line 123 of file arp.h.

#define ARP_TEMP_IP 1

For Temporary addresses. Entry is removed after ttl

Definition at line 131 of file arp.h.

#define ARP_TIMEOUT 60

ARP cache entry refresh period (in seconds).

Change this number to change refresh period of ARP cache entries.

Changing this values changes the amount of ARP refreshes performed and thus can change processing power used when refreshing the entries.

Definition at line 37 of file arp.h.

#define ARP_TSIZE 10

ARP cache size (number of entries).

Change this number to change ARP cache size (number of cached ARP:IP address pairs).

Changing this values affects memory consumption as well as processing power needed to manage the ARP cache. If a device communicates with a limited number of hosts, this cache size may be smaller, thus reducing memory requirements. Small cache size may, however, reduce performance when communicating with more hosts than there are cache entries available.

Definition at line 26 of file arp.h.

#define MAXHWALEN 6

Maximum HW address Length (6=Eth)

Definition at line 9 of file arp.h.

#define MAXPRALEN 4

Maximum Protocol adr.len (4=IPv4)

Definition at line 10 of file arp.h.

Function Documentation

INT8 arp_add (UINT32 *pra*, UINT8 * *hwadr*, UINT8 *type*)

Add given IP address and MAC address to ARP cache.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Parameters:

pra - protocol address (assumed IPv4)
hwadr - pointer to Ethernet MAC address (6 bytes)
type - type of address allocated if not found. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

- 0 - Address already in cache. Refreshed.
- 1 - New entry in ARP cache created

New IP address is added to ARP cache based on the information supplied to function as parameters.

Definition at line 501 of file arp.c.

INT8 arp_alloc (UINT8 *type*)

Allocate ARP entry in ARP cache table.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

1.11.2001

Parameters:

type Type of ARP cache entry beeing allocated. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

>=0 - pointer to allocated ARP entry (actaully index in the ARP cache table)

Allocate arp entry for given type. Chooses the unused entry if one exists. Otherwice deletes entries in round-robin fashion.

Definition at line 415 of file arp.c.

struct arp_entry* arp_find (LWORD *pra*, struct netif * *machine*, UINT8 *type*)

Find an ARP entry given a protocol address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

01.11.2001

Parameters:

pra - Protocol address (IPv4)
machine - Pointer to configuration of network interface used
type - Type of address allocated if not found. Can be one of the following:

- ARP_FIXED_IP
- ARP_TEMP_IP

Returns:

- 0 - ARP entry not found or not ready yet (waiting for ARP response)
- struct arp_entry* - pointer to solved entry of ARP cache table

This function tries to resolve IPv4 address by checking the ARP cache table and sending ARP requested if needed.

Definition at line 588 of file arp.c.

void arp_get_response (void)

Extract data from the received ARP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Warning:

- This function starts reading data from Ethernet controller without initializing it for reading it first, so NIC must already be initialized for reading from correct address (it expects ar$sha field from ARP packet immediately)

This function is invoked from process_arp() function when ARP reply packet is detected. Basic checking is performed to see if the packet is intended for us, and if it is, ARP cache table is checked and corresponding entry is refreshed (resolved).

Definition at line 245 of file arp.c.

void arp_init (void)

Initialize data structures for ARP processing.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

01.11.2001

Warning:

- Invoke this function at start-up to properly initialize ARP cache subsystem.

Call this function to properly initialize ARP cache table and so that ARP allocates and initializes a timer for it's use.

Definition at line 859 of file arp.c.

void arp_manage (void)

Manage ARP cache periodically.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

04.11.2001

Warning:

- Invoke this function periodically to ensure proper ARP cache behaviour

Iterate through ARP cache aging entries. If timed-out entry is found, remove it (dynamic address) or update it (static address). This function must be called periodically by the system.

Definition at line 734 of file arp.c.

void arp_send_req (UINT8 *entry*)

Send ARP request based on information in an ARP cache table.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

1.11.2001

Parameters:

entry Index of ARP cache entry that is beeing resolved

Invoked from arp_find() and arp_manage() functions, arp_send_request creates ARP request packet based on data stored in the ARP cache entry who's index is given as a parameter.

Definition at line 332 of file arp.c.

void arp_send_request (void)

void arp_send_response (void)

Send response to an ARP request.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Warning:

- This function starts reading data from Ethernet controller without initializing it for reading it first, so NIC must already be initialized for reading from correct address (it expects ar$sha field from ARP packet immediately)

This function is invoked from process_arp() function in order to send a reply to an ARP request. First, incoming packet is checked to see if it is intended for us or not. If not, function does not do anything. Otherwise, ARP reply packet is formed and sent.

Definition at line 120 of file arp.c.

struct arp_entry* arpfind (LWORD, struct netif *, UINT8)

BYTE is_subnet (LWORD *ipadr*, struct netif * *machine*)

Checks if a given IP address belongs to the subnet of a given machine.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

05.11.2001

Parameters:

ipadr - IP address under check
machine - pointer to configuration of network parameters used

Returns:

- TRUE - ipadr belongs to subnet of given machine
- FALSE - ipadr is NOT a part of subnet of given machine

Based on information supplied in ipadr and machine parameters this function performs basic check if IP address is on the same subnet as the one defined for the machine.

Definition at line 909 of file arp.c.

BYTE process_arp (struct ethernet_frame * *frame*)

Process and analyze the received ARP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.07.2002

Parameters:

frame Pointer to ethernet_frame structure containing information about the received frame

Returns:

Return TRUE if Ethernet frame processed held ARP packet, otherwise FALSE.

Invoke process_arp function whenever ARP packet is received (see main_demo.c for an example loop). This function will process the received packet, analyze it'c content briefly and perform on of the two possible actions:

- If the received packet is ARP request it will invoke arp_send_reply in order to send ARP reply back

- If the received packet is ARP response it will iterate through the cache table and try to find ARP entry that is beeing resolved or refreshed

Definition at line 45 of file arp.c.

---

bootp.c File Reference

Detailed Description

OpenTCP BOOTP client implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
10.7.2002

Bug:

Warning:

Todo:

- Offer callback for once the BOOTP client is definitely finished (BOOTPC_STATE_REPLY_GET state)

OpenTCP BOOTP client protocol implementation. Function declarations can be found in bootp.h
Definition in file bootp.c.

```
#include "datatypes.h"
#include "system.h"
#include "timers.h"
#include "tcp_ip.h"
#include "bootp.h"
```

Functions

- INT8 bootpc_init (UINT8 mode)
  Initializes BOOTP client.

- void bootpc_stop (void)
  Stop BOOTP client operation.

- INT8 bootpc_enable (void)
  Enable BOOTP client operation.

- void bootpc_run (void)
  BOOTP client main loop.

- INT32 bootpc_eventlistener (INT8 cbhandle, UINT8 event, UINT32 remip, UINT16 remport, UINT16 bufindex, UINT16 dlen)
  BOOTP event listener.

Variables

- UINT8 bootp_app_init = 0
- struct {
- UINT8 **state**
- UINT8 **mode**
- INT8 **sochandle**
- UINT16 **tmrhandle**
- UINT16 **bootsecs**
- } bootp
  BOOTP client information.

Function Documentation

INT8 bootpc_enable (void)

Enable BOOTP client operation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Invoke this function to enable BOOTP client operation.

Definition at line 164 of file bootp.c.

INT32 bootpc_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *remip*, UINT16 *remport*, UINT16 *bufindex*, UINT16 *dlen*)

BOOTP event listener.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

07.10.2002

Parameters:

cbhandle handle of the socket this packet is intended for.
event event that is notified. For UDP, only UDP_EVENT_DATA.
ipaddr IP address of remote host who sent the UDP datagram
port port number of remote host who sent the UDP datagram
buffindex buffer index in RTL8019AS

Returns:

- - 1 - error in processing
- >0 - BOOTP reply successfully processed

Note:

- Event listeners are NOT to be invoked directly. They are callback functions invoked by the TCP/IP stack to notify events.

Analyze received UDP packet and see if it contains what we need. If yes, get new network settings.

Definition at line 334 of file bootp.c.

INT8 bootpc_init (UINT8 *mode*)

Initializes BOOTP client.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.07.2002

Returns:

- -1 - Error during initialization
- >=0 - OK

Invoke this function to initialize BOOTP client. This will also trigger BOOTP address-fetching procedure.

Definition at line 107 of file bootp.c.

void bootpc_run (void)

BOOTP client main loop.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

07.10.2002

Main thread of the BOOTP client that should be invoked periodically.

Definition at line 182 of file bootp.c.

void bootpc_stop (void)

Stop BOOTP client operation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Invoke this function to disable BOOTP client operation once it is not needed any more or just to temporarily suspend it's operation.

Definition at line 149 of file bootp.c.

Variable Documentation

struct { ... } bootp

BOOTP client information.

bootp variable holds various information about the BOOTP client and also information needed by the BOOTP client to function properly.

UINT8 bootp_app_init = 0

Defines whether bootpc_init has already been invoked or not

Definition at line 76 of file bootp.c.

UINT16 bootsecs

Definition at line 90 of file bootp.c.

UINT8 mode

Definition at line 87 of file bootp.c.

INT8 sochandle

Definition at line 88 of file bootp.c.

UINT8 state

Definition at line 86 of file bootp.c.

UINT16 tmrhandle

Definition at line 89 of file bootp.c.

bootp.h File Reference

Detailed Description

OpenTCP BOOTP client interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
10.7.2002

OpenTCP BOOTP client function declarations, constants, etc.
Definition in file bootp.h.

```
#include "datatypes.h"
```

Defines

- #define BOOTP_RETRY_TOUT 5
- #define BOOTP_CLIENTPORT 68
- #define BOOTP_SERVERPORT 67
- #define BOOTPC_STATE_DISABLED 0
- #define BOOTPC_STATE_ENABLED 1
- #define BOOTPC_STATE_REQUEST_NEEDED 2
- #define BOOTPC_STATE_WAITING_REPLY 3
- #define BOOTPC_STATE_REPLY_GET 4
- #define BOOTP_OPTION_SUBNETMASK 1
- #define BOOTP_OPTION_DEFGW 3
- #define BOOTP_REPLY 2
- #define BOOTP_HWLEN_ETHERNET 6
- #define BOOTP_HTYPE_ETHERNET 1

Functions

- INT8 init_bootpc (UINT8)
- INT8 bootpc_enable (void)
  Enable BOOTP client operation.

- void bootpc_stop (void)
  Stop BOOTP client operation.

- void bootpc_run (void)
  BOOTP client main loop.

- INT32 bootpc_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)
  BOOTP event listener.

Define Documentation

#define BOOTP_CLIENTPORT 68

Local BOOTP client port that will be used for sending requests

Definition at line 71 of file bootp.h.

#define BOOTP_HTYPE_ETHERNET 1

Definition at line 104 of file bootp.h.

#define BOOTP_HWLEN_ETHERNET 6

Definition at line 103 of file bootp.h.

#define BOOTP_OPTION_DEFGW 3

Default gateway option BOOTP client is waiting for in the reply from the BOOTP server

Definition at line 97 of file bootp.h.

#define BOOTP_OPTION_SUBNETMASK 1

Subnet mask option BOOTP client is waiting for in the reply from the BOOTP server

Definition at line 93 of file bootp.h.

#define BOOTP_REPLY 2

Definition at line 102 of file bootp.h.

#define BOOTP_RETRY_TOUT 5

How many seconds to pass before retrying

Definition at line 69 of file bootp.h.

#define BOOTP_SERVERPORT 67

BOOTP server's port

Definition at line 73 of file bootp.h.

#define BOOTPC_STATE_DISABLED 0

BOOTP client intentionally disabled

Definition at line 75 of file bootp.h.

#define BOOTPC_STATE_ENABLED 1

BOOTP initialized and waiting to send initial BOOTP request

Definition at line 76 of file bootp.h.

#define BOOTPC_STATE_REPLY_GET 4

Once we get into this state, proper reply has been received from the BOOTP server

Definition at line 87 of file bootp.h.

#define BOOTPC_STATE_REQUEST_NEEDED 2

New (or first) BOOTP requests must be issued

Definition at line 79 of file bootp.h.

#define BOOTPC_STATE_WAITING_REPLY 3

After issuing the request BOOTP is in this state waiting either for timeout or a response from the BOOTP server

Definition at line 82 of file bootp.h.

Function Documentation

INT8 bootpc_enable (void)

Enable BOOTP client operation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Invoke this function to enable BOOTP client operation.

Definition at line 164 of file bootp.c.

INT32 bootpc_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *remip*, UINT16 *remport*, UINT16 *bufindex*, UINT16 *dlen*)

BOOTP event listener.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

07.10.2002

Parameters:

cbhandle handle of the socket this packet is intended for.
event event that is notified. For UDP, only UDP_EVENT_DATA.
ipaddr IP address of remote host who sent the UDP datagram
port port number of remote host who sent the UDP datagram
buffindex buffer index in RTL8019AS

Returns:

- - 1 - error in processing
- >0 - BOOTP reply successfully processed

Note:

- Event listeners are NOT to be invoked directly. They are callback functions invoked by the TCP/IP stack to notify events.

Analyze received UDP packet and see if it contains what we need. If yes, get new network settings.

Definition at line 334 of file bootp.c.

void bootpc_run (void)

BOOTP client main loop.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

07.10.2002

Main thread of the BOOTP client that should be invoked periodically.

Definition at line 182 of file bootp.c.

void bootpc_stop (void)

Stop BOOTP client operation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Invoke this function to disable BOOTP client operation once it is not needed any more or just to temporarily suspend it's operation.

Definition at line 149 of file bootp.c.

INT8 init_bootpc (UINT8)

---

datatypes.h File Reference

Detailed Description

OpenTCP definitions of datatypes of certain length.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

10.7.2002

This file holds #defines of data types used in the OpenTCP sources so that recompiling for another MCU is easier even when the other MCU is using different size default values.

Constants that need to be defined in this file for every microcontroller and/or compiler are:

- BYTE - unsigned 8 bit value

- WORD - unsigned 16 bit value
- LWORD - unsigned 32 bit value
- UINT8 - unsigned 8 bit value
- INT8 - signed 8 bit value
- UINT16 - unsigned 16 bit value
- INT16 - signed 16 bit value
- UINT32 - unsigned 32 bit value
- INT32 - signed 32 bit value

Definition in file datatypes.h.

```
#include "MOTTYPES.h"
```

Defines

- #define LWORD unsigned long
- #define BYTE unsigned char
- #define WORD unsigned short
- #define UINT8 unsigned char
- #define INT8 signed char
- #define UINT16 unsigned short
- #define INT16 short

Typedefs

- typedef unsigned long uint32
- typedef signed long int32
- typedef unsigned char uint8
- typedef unsigned short uint16
- typedef signed char int8
- typedef signed short int16

Define Documentation

#define BYTE unsigned char

Definition at line 90 of file datatypes.h.

#define INT16 short

16 bit signed

Definition at line 106 of file datatypes.h.

#define INT8 signed char

8 bit signed

Definition at line 98 of file datatypes.h.

#define LWORD unsigned long

32 bit unsigned

Definition at line 84 of file datatypes.h.

#define UINT16 unsigned short

16 bit unsigned

Definition at line 102 of file datatypes.h.

#define UINT8 unsigned char

8 bit unsigned

Definition at line 94 of file datatypes.h.

#define WORD unsigned short

16 bit unsigned

Definition at line 91 of file datatypes.h.

Typedef Documentation

typedef signed short int16

Definition at line 113 of file datatypes.h.

typedef signed long int32

Definition at line 87 of file datatypes.h.

typedef signed char int8

Definition at line 112 of file datatypes.h.

typedef unsigned short uint16

Definition at line 110 of file datatypes.h.

typedef unsigned long uint32

Definition at line 86 of file datatypes.h.

typedef unsigned char uint8

Definition at line 109 of file datatypes.h.

---

debug.h File Reference

Detailed Description

OpenTCP file for debug options.

Author:

- Jari Lahti (jari.lahti@violasystems.com)
- Vladan Jovanovic (vladan.jovanovic@violasytems.com)

Version:

1.0

Date:

10.9.2002

This file contains debug settings for OpenTCP and it's modules. Debugging in this case only assumes a function (named mputs) that sends a null-terminated string over a serial port.
In order for the debugging to work this function **must be** implemented separately (this greately depends on your applications and hardware configuration so it was not implemented here). Empty mputs function is provided in system.c (not much help ;-)
Definition in file debug.h.

```
#include "datatypes.h"
#include "system.h"
```

Defines

- #define DEBUG 0
- #define ETHERNET_DEBUG 1
- #define IP_DEBUG 1
- #define ICMP_DEBUG 1

- #define ARP_DEBUG 1
- #define TCP_DEBUG 1
- #define UDP_DEBUG 1
- #define TIMERS_DEBUG 1
- #define DEBUGOUT(c) ; {};
- #define ETH_DEBUGOUT(c) ; {};
- #define IP_DEBUGOUT(c) ; {};
- #define ICMP_DEBUGOUT(c) ; {};
- #define ARP_DEBUGOUT(c) ; {};
- #define TCP_DEBUGOUT(c) ; {};
- #define UDP_DEBUGOUT(c) ; {};
- #define TMR_DEBUGOUT(c) ; {};

Define Documentation

#define ARP_DEBUG 1

enable/disable ARP-level debug messages

Definition at line 96 of file debug.h.

#define ARP_DEBUGOUT(c) ; {};

Definition at line 158 of file debug.h.

#define DEBUG 0

Controls debugging on a global level and also enables DEBUGOUT. Possible values are:

- 0 - debugging messages disabled globally
- 1 - debugging messages enabled globally. DEBUGOUT will print messages. TCP/IP layers that will print message are chosen separately.

Definition at line 80 of file debug.h.

#define DEBUGOUT(c) ; {};

Definition at line 153 of file debug.h.

#define ETH_DEBUGOUT(c) ; {};

Definition at line 155 of file debug.h.

#define ETHERNET_DEBUG 1

enable/disable Ethernet-level debug messages

Definition at line 93 of file debug.h.

#define ICMP_DEBUG 1

enable/disable ICMP-level debug messages

Definition at line 95 of file debug.h.

#define ICMP_DEBUGOUT(c) ; {};

Definition at line 157 of file debug.h.

#define IP_DEBUG 1

enable/disable IP-level debug messages

Definition at line 94 of file debug.h.

#define IP_DEBUGOUT(c) ; {};

Definition at line 156 of file debug.h.

#define TCP_DEBUG 1

enable/disable TCP-level debug messages

Definition at line 97 of file debug.h.

#define TCP_DEBUGOUT(c) ; {};

Definition at line 159 of file debug.h.

#define TIMERS_DEBUG 1

enable/disable Timer-level debug messages

Definition at line 99 of file debug.h.

#define TMR_DEBUGOUT(c) ; {};

Definition at line 161 of file debug.h.

#define UDP_DEBUG 1

enable/disable UDP-level debug messages

Definition at line 98 of file debug.h.

#define UDP_DEBUGOUT(c) ; {};

Definition at line 160 of file debug.h.

---

dhcpc.c File Reference

Detailed Description

OpenTCP DHCP client implementation.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:

1.03

Date:

23.5.2003

Bug:

Warning:

Todo:

- We SHOULD first test (with PING or ARP) assigned IP address to see if it's in use already.

OpenTCP DHCP client protocol implementation. Features a complete DHCP state machine. Function declarations can be found in dhcpc.h
Definition in file dhcpc.c.

```
#include "datatypes.h"
#include "debug.h"
#include "system.h"
#include "tcp_ip.h"
#include "timers.h"
#include "dhcpc.h"
```

Functions

- INT32 dhcpc_eventlistener (INT8 cbhandle, UINT8 event, UINT32 ipaddr, UINT16 port, UINT16 buffindex, UINT16 datalen)
  DHCP event listener, parses all DHCP replies.

- INT8 dhcpc_send_message (UINT8 msg_type)
  Sends DHCP messages.

- INT8 dhcpc_init (void)
  Initializes DHCP client.

- void dhcpc_run (void)
  DHCP client main state machine.

- UINT32 dhcpc_read_n_bytes (UINT8 n)
  Processes received parameter from DHCP server.

Variables

- UINT8 dhcpc_state
  Holds DHCP clients' state information.

- UINT8 dhcpc_timer_handle
  DHCP client's timer handle.

- INT8 dhcpc_soc_handle
  DHCP client's UDP socket handle.

- UINT8 dhcpc_initialized = 0
  Holds information if DHCP client is initialized.

- UINT32 dhcpc_t1
  DHCP renew timer.

- UINT32 dhcpc_t2
  DHCP rebind timer.

- UINT32 dhcpc_server_identifier
  DHCP server identifier as received from DHCP server.

- UINT32 dhcpc_requested_ip

Holds offered IP address or IP address that we're requesting.

Function Documentation

INT32 dhcpc_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *ipaddr*, UINT16 *port*, UINT16 *buffindex*, UINT16 *datalen*)

DHCP event listener, parses all DHCP replies.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

This is internal function invoked by OpenTCP UDP module when DHCP reply on a given UDP port is received. This function parses the response, checks for correctnes and performs certain actions based on the current state of DHCP client.

Definition at line 565 of file dhcpc.c.

INT8 dhcpc_init (void)

Initializes DHCP client.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

This function should be called once when system starts to initialize and start DHCP client. Before this function is invoked, localmachine.localip MUST be se to either zero (in which case DHCP client will request any IP address) or a previously assigned IP address (which doesn't mean DHCP server will allow us to continue using this address) in which case DHCP client will first try to obtain that existing IP address.

Definition at line 155 of file dhcpc.c.

UINT32 dhcpc_read_n_bytes (UINT8 *n*)

Processes received parameter from DHCP server.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

Parameters:

n Number of bytes to read&process.

Returns:

Returns last for 4 bytes that were read as a 32-bit variable

This is internal function that get's invoked to read a received parameter in DHCP message. Introduced to optimize code a little as 4 byte parameters are often returned by DHCP server (netmask, gateway, server identifier, T1, T2, lease expiration time, DNS IP,..)

Definition at line 546 of file dhcpc.c.

void dhcpc_run (void)

DHCP client main state machine.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

Call this function periodically from main loop to ensure proper operation. This function holds the main state machine of DHCP client that ensures proper operation.

Definition at line 198 of file dhcpc.c.

INT8 dhcpc_send_message (UINT8 *msg_type*)

Sends DHCP messages.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

Parameters:

msg_type Type of DHCP message to be sent. This implementation can send only DHCP_DISCOVER, DHCP_REQUEST and #DCHP_DECLINE messages.

Returns:

Returns result of udp_send() function.

This is internal function invoked to send appropriate DHCP message.

Definition at line 386 of file dhcpc.c.

Variable Documentation

UINT8 dhcpc_initialized = 0

Holds information if DHCP client is initialized.

Holds information if DHCP client is initialized

Definition at line 103 of file dhcpc.c.

UINT32 dhcpc_requested_ip

Holds offered IP address or IP address that we're requesting.

This variable holds the IP address that DHCP server offered to us during address request procedure and this is the address that we will be requesting in all future requests untill DHCP server disallows us to use it any more.

Definition at line 137 of file dhcpc.c.

UINT32 dhcpc_server_identifier

DHCP server identifier as received from DHCP server.

This variable will hold DHCP server identifier (which will actually be server's IP address).

Definition at line 128 of file dhcpc.c.

INT8 dhcpc_soc_handle

DHCP client's UDP socket handle.

DHCP client's UDP socket handle

Definition at line 97 of file dhcpc.c.

UINT8 dhcpc_state

Holds DHCP clients' state information.

This variable holds DHCP clients' current state information. Possible states are DHCP_STATE_INIT_REBOOT, DHCP_STATE_REBOOTING, DHCP_STATE_INIT,

DHCP_STATE_SELECTING, DHCP_STATE_REQUESTING, DHCP_STATE_BOUND, DHCP_STATE_RENEWING, DHCP_STATE_REBINDING.

Definition at line 84 of file dhcpc.c.

UINT32 dhcpc_t1

DHCP renew timer.

This variable holds renew time (in seconds) after which we'll start the renewing process. While obtaining the parameters from DHCP server (thus before we know of the renew time) this is used also to time retransmissions.

Definition at line 112 of file dhcpc.c.

UINT32 dhcpc_t2

DHCP rebind timer.

This variable holds rebind time (in seconds) after which we'll start the rebinding process. While obtaining the paramters from DHCP server (thus before we know of the renew time) this is also used to time retransmissions as well as timeout detection

Definition at line 121 of file dhcpc.c.

UINT8 dhcpc_timer_handle

DHCP client's timer handle.

Hold DHCP clients' timer handle. We'll use only one timer from timer pool and take care of the rest by ourselves manually

Definition at line 91 of file dhcpc.c.

---

dhcpc.h File Reference

Detailed Description

OpenTCP DHCP client interface file.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:

1.0

Date:

23.5.2003

OpenTCP DHCP protocol function declarations, constants, etc.
Definition in file dhcpc.h.

```
#include "datatypes.h"
```

Defines

- #define DHCP_SERVER_PORT 67
- #define DHCP_CLIENT_PORT 68
- #define BOOT_REQUEST 1
- #define BOOT_REPLY 2
- #define DHCP_STATE_INIT_REBOOT 0
- #define DHCP_STATE_REBOOTING 1
- #define DHCP_STATE_INIT 2
- #define DHCP_STATE_SELECTING 3
- #define DHCP_STATE_REQUESTING 4
- #define DHCP_STATE_BOUND 5
- #define DHCP_STATE_RENEWING 6
- #define DHCP_STATE_REBINDING 7
- #define DHCP_DISCOVER 1
- #define DHCP_OFFER 2
- #define DHCP_REQUEST 3
- #define DHCP_DECLINE 4
- #define DHCP_ACK 5
- #define DHCP_NAK 6
- #define DHCP_RELEASE 7
- #define DHCP_INFORM 8
- #define DHCP_OPT_PAD 0
- #define DHCP_OPT_END 255
- #define DHCP_OPT_SUBNET_MASK 1
- #define DHCP_OPT_TIME_OFFSET 2
- #define DHCP_OPT_ROUTER 3
- #define DHCP_OPT_TIME_SERVER 4
- #define DHCP_OPT_NAME_SERVER 5
- #define DHCP_OPT_DNS_SERVER 6
- #define DHCP_OPT_HOST_NAME 12
- #define DHCP_OPT_POP3_SERVER 70
- #define DHCP_OPT_REQUESTED_IP 50
- #define DHCP_OPT_LEASE_TIME 51
- #define DHCP_OPT_OVERLOAD 52
- #define DHCP_OPT_MSG_TYPE 53
- #define DHCP_OPT_SERV_IDENT 54
- #define DHCP_OPT_PARAM_REQUEST 55
- #define DHCP_OPT_T1_VALUE 58
- #define DHCP_OPT_T2_VALUE 59

Functions

- INT8 dhcpc_init (void)
  Initializes DHCP client.

- void dhcpc_run (void)
  DHCP client main state machine.

Define Documentation

#define BOOT_REPLY 2

Definition at line 73 of file dhcpc.h.

#define BOOT_REQUEST 1

Definition at line 72 of file dhcpc.h.

#define DHCP_ACK 5

Definition at line 90 of file dhcpc.h.

#define DHCP_CLIENT_PORT 68

Definition at line 70 of file dhcpc.h.

#define DHCP_DECLINE 4

Definition at line 89 of file dhcpc.h.

#define DHCP_DISCOVER 1

Definition at line 86 of file dhcpc.h.

#define DHCP_INFORM 8

Definition at line 93 of file dhcpc.h.

#define DHCP_NAK 6

Definition at line 91 of file dhcpc.h.

#define DHCP_OFFER 2

Definition at line 87 of file dhcpc.h.

#define DHCP_OPT_DNS_SERVER 6

Definition at line 103 of file dhcpc.h.

#define DHCP_OPT_END 255

Definition at line 97 of file dhcpc.h.

#define DHCP_OPT_HOST_NAME 12

Definition at line 104 of file dhcpc.h.

#define DHCP_OPT_LEASE_TIME 51

Definition at line 107 of file dhcpc.h.

#define DHCP_OPT_MSG_TYPE 53

Definition at line 109 of file dhcpc.h.

#define DHCP_OPT_NAME_SERVER 5

Definition at line 102 of file dhcpc.h.

#define DHCP_OPT_OVERLOAD 52

Definition at line 108 of file dhcpc.h.

#define DHCP_OPT_PAD 0

Definition at line 96 of file dhcpc.h.

#define DHCP_OPT_PARAM_REQUEST 55

Definition at line 111 of file dhcpc.h.

#define DHCP_OPT_POP3_SERVER 70

Definition at line 105 of file dhcpc.h.

#define DHCP_OPT_REQUESTED_IP 50

Definition at line 106 of file dhcpc.h.

#define DHCP_OPT_ROUTER 3

Definition at line 100 of file dhcpc.h.

#define DHCP_OPT_SERV_IDENT 54

Definition at line 110 of file dhcpc.h.

#define DHCP_OPT_SUBNET_MASK 1

Definition at line 98 of file dhcpc.h.

#define DHCP_OPT_T1_VALUE 58

Definition at line 112 of file dhcpc.h.

#define DHCP_OPT_T2_VALUE 59

Definition at line 113 of file dhcpc.h.

#define DHCP_OPT_TIME_OFFSET 2

Definition at line 99 of file dhcpc.h.

#define DHCP_OPT_TIME_SERVER 4

Definition at line 101 of file dhcpc.h.

#define DHCP_RELEASE 7

Definition at line 92 of file dhcpc.h.

#define DHCP_REQUEST 3

Definition at line 88 of file dhcpc.h.

#define DHCP_SERVER_PORT 67

Definition at line 69 of file dhcpc.h.

#define DHCP_STATE_BOUND 5

Definition at line 81 of file dhcpc.h.

#define DHCP_STATE_INIT 2

Definition at line 78 of file dhcpc.h.

#define DHCP_STATE_INIT_REBOOT 0

Definition at line 76 of file dhcpc.h.

#define DHCP_STATE_REBINDING 7

Definition at line 83 of file dhcpc.h.

#define DHCP_STATE_REBOOTING 1

Definition at line 77 of file dhcpc.h.

#define DHCP_STATE_RENEWING 6

Definition at line 82 of file dhcpc.h.

#define DHCP_STATE_REQUESTING 4

Definition at line 80 of file dhcpc.h.

#define DHCP_STATE_SELECTING 3

Definition at line 79 of file dhcpc.h.

Function Documentation

INT8 dhcpc_init (void)

Initializes DHCP client.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

This function should be called once when system starts to initialize and start DHCP client. Before this function is invoked, localmachine.localip MUST be se to either zero (in which case DHCP client will request any IP address) or a previously assigned IP address (which doesn't mean DHCP server will allow us to continue using this address) in which case DHCP client will first try to obtain that existing IP address.

Definition at line 155 of file dhcpc.c.

void dhcpc_run (void)

DHCP client main state machine.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

23.05.2003

Call this function periodically from main loop to ensure proper operation. This function holds the main state machine of DHCP client that ensures proper operation.

Definition at line 198 of file dhcpc.c.

---

dns.c File Reference

Detailed Description

OpenTCP DNS client implementation.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:

1.0

Date:

10.10.2002

Bug:

Warning:

Todo:

- Probably should implement sending different ID with requests.
- Maybe create similar cache as for ARP ?
- **Definitely** implement DNS's IP address use as a parameter to get_host_by_name function. This would allow more flexible manipulation

OpenTCP DNS client implementation. API functions, data structures and constants may be found in dns.h
Definition in file dns.c.

```
#include "debug.h"
#include "datatypes.h"
#include "globalvariables.h"
#include "system.h"
#include "timers.h"
#include "tcp_ip.h"
#include "dns.h"
```

Defines

- #define DNS_STATE_READY 0
- #define DNS_STATE_BUSY 1
- #define DNS_STATE_RESEND 2

Functions

- void dns_init (void)
  Initialize resources needed for the DNS client.

- void dns_retransmit (void)
  Retransmits requests towards the DNS server.

- void dns_run (void)
  DNS client main loop.

- INT32 dns_eventlistener (INT8 cbhandle, UINT8 event, UINT32 ipaddr, UINT16 port, UINT16 buffindex, UINT16 datalen)
  DNS client event listener.

- INT16 get_host_by_name (UINT8 *host_name_ptr, void(*listener)(UINT8, UINT32))
  Invokes DNS resolver.

Variables

- UINT8 dns_state
- INT8 dns_socket
- UINT8 dns_timer
- UINT8 dns_retries
- UINT32 dns_tmp_ip
- UINT8 * dns_hostptr
- void(* dns_event_listener)(UINT8 event, UINT32 data)

Define Documentation

#define DNS_STATE_BUSY 1

Definition at line 85 of file dns.c.

#define DNS_STATE_READY 0

Definition at line 84 of file dns.c.

#define DNS_STATE_RESEND 2

Definition at line 86 of file dns.c.

Function Documentation

INT32 dns_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *ipaddr*, UINT16 *port*, UINT16 *buffindex*, UINT16 *datalen*)

DNS client event listener.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Parameters:

cbhandle handle of the socket this packet is intended for.
event event that is notified. For UDP, only UDP_EVENT_DATA.
ipaddr IP address of remote host who sent the UDP datagram
port port number of remote host who sent the UDP datagram
buffindex buffer index in RTL8019AS

Returns:

- - 1 - error in processing
- 0 - DNS reply successfully processed

Note:

- Event listeners are NOT to be invoked directly. They are callback functions invoked by the TCP/IP stack to notify events.

This, of course, is where responses from DNS server are processed and checked whether they contain the IP address we requested or if they contain authorative name server to which we should proceed.

If we received the IP address we requested, DNS_EVENT_SUCCESS is reported to application DNS event listener. Otherwise DNS_EVENT_ERROR is reported.

Definition at line 205 of file dns.c.

void dns_init (void)

Initialize resources needed for the DNS client.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Invoke this function at startup to properly initialize DNS resources.

Definition at line 108 of file dns.c.

void dns_retransmit (void)

Retransmits requests towards the DNS server.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

This is internal function and IS NOT INTENDED to be invoked by the user application. It simply checks if retransmissions should be done (when retransmissions not used yet) and if yes, sends one. Otherwise timeout error is sent to the event listener.

Definition at line 142 of file dns.c.

void dns_run (void)

DNS client main loop.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Simple main loop that checks whether DNS requests should be sent or not (based on timer timeout). If yes, and DNS is in appropriate state, dns_retransmit() function is invoked.

Definition at line 168 of file dns.c.

INT16 get_host_by_name (UINT8 * *host_name_ptr*, void(* *listener*)(UINT8, UINT32))

Invokes DNS resolver.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Parameters:

host_name_ptr Pointer to null-terminated host name to be resolved
listener Pointer to DNS listener function that listens to events from DNS client. This function takes two parameters: first one can take a value of DNS_EVENT_SUCCESS or DNS_EVENT_ERROR and thus determine the meaning of the second parameter. If first parameter is DNS_EVENT_SUCCESS, second parameter represents requested IP address. In case of DNS_EVENT_ERROR, second parameter can be one of the: DNS_ERROR_FORMAT, DNS_ERROR_SERVER_FAILURE, DNS_ERROR_NAME_ERROR, DNS_ERROR_NOT_IMPLEMENTED, DNS_ERROR_REFUSED, DNS_ERROR_TIMEOUT, DNS_ERROR_GENERAL

Returns:

- DNS_ERROR_BUSY - Signals that DNS is currently processing another request so it is not possible to process a new one
- DNS_ERROR_OVERFLOW - Network transmit buffer too small to hold DNS request
- DNS_ERROR_LABEL - Label in host name longer than 63 bytes. Error
- DNS_ERROR_NAME - Host name longer than 264 bytes. Error

Invoke this function to start name-resolving process. Note that currently DNS client can process only one request at a time and will not allow multiple requests.

Definition at line 448 of file dns.c.

Variable Documentation

void(* dns_event_listener)(UINT8 event, UINT32 data)

Definition at line 98 of file dns.c.

UINT8* dns_hostptr

Pointer to hostname that is beeing resolved. Needed for retransmissions.

Definition at line 95 of file dns.c.

UINT8 dns_retries

DNS retry counter used for detecting timeouts

Definition at line 91 of file dns.c.

INT8 dns_socket

UDP socket used by the DNS resolver

Definition at line 89 of file dns.c.

UINT8 dns_state

Current DNS state. Used to prevent multiple requests, issue retransmissions,... See DNS_STATE_* for possible values.

Definition at line 88 of file dns.c.

UINT8 dns_timer

DNS timer handle used for retransmissions

Definition at line 90 of file dns.c.

UINT32 dns_tmp_ip

Used in many ways: as an IP address holder, for issuing requests to authorative name servers,..

Definition at line 93 of file dns.c.

---

dns.h File Reference

Detailed Description

OpenTCP DNS interface file.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:

1.0

Date:

10.9.2002

OpenTCP DNS protocol function declarations, constants, etc.
Definition in file dns.h.

```
#include "datatypes.h"
```

Defines

- #define DNS_UDP_PORT 53
- #define DNS_SERVER_IP 0xac100201
- #define DNS_RESEND_PERIOD 2
- #define DNS_NUM_RETRIES 5
- #define DNS_EVENT_ERROR 0
- #define DNS_EVENT_SUCCESS 1
- #define DNS_ERROR_FORMAT 1
- #define DNS_ERROR_SERVER_FAILURE 2
- #define DNS_ERROR_NAME_ERROR 3
- #define DNS_ERROR_NOT_IMPLEMENTED 4
- #define DNS_ERROR_REFUSED 5
- #define DNS_ERROR_TIMEOUT 16
- #define DNS_ERROR_GENERAL 17

- #define DNS_ERROR_BUSY -4
- #define DNS_ERROR_LABEL -5
- #define DNS_ERROR_NAME -6
- #define DNS_ERROR_OVERFLOW -7

Functions

- INT16 get_host_by_name (UINT8 *host_name_ptr, void(*listener)(UINT8, UINT32))
  Invokes DNS resolver.

- void dns_init (void)
  Initialize resources needed for the DNS client.

- void dns_run (void)
  DNS client main loop.

- INT32 dns_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)
  DNS client event listener.

Define Documentation

#define DNS_ERROR_BUSY -4

Returned from get_host_by_name(): DNS client is currently busy with another request and is unable the process a new one

Definition at line 135 of file dns.h.

#define DNS_ERROR_FORMAT 1

The name server was unable to interpret the query (RFC1035)

Definition at line 95 of file dns.h.

#define DNS_ERROR_GENERAL 17

General (not specific) error occured while resolving host name.

Definition at line 124 of file dns.h.

#define DNS_ERROR_LABEL -5

Returned from get_host_by_name(): Part of the host name (label) consists of more than 63 characters.

Definition at line 140 of file dns.h.

#define DNS_ERROR_NAME -6

Returned from get_host_by_name(): Host name too long (more than 263 bytes)

Definition at line 144 of file dns.h.

#define DNS_ERROR_NAME_ERROR 3

Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist (RFC1035)

Definition at line 103 of file dns.h.

#define DNS_ERROR_NOT_IMPLEMENTED 4

The name server does not support the requested kind of query (RFC1035)

Definition at line 110 of file dns.h.

#define DNS_ERROR_OVERFLOW -7

net_buf too small for the entire DNS request to be stored in it.

Definition at line 147 of file dns.h.

#define DNS_ERROR_REFUSED 5

The name server refuses to perform the specified operation for policy reasons. (RFC 1035)

Definition at line 114 of file dns.h.

#define DNS_ERROR_SERVER_FAILURE 2

The name server was unable to process this query due to a problem with the name server (RFC1035)

Definition at line 98 of file dns.h.

#define DNS_ERROR_TIMEOUT 16

Timeout occured while DNS was trying to resolve the host name. New request should be issued if the address is needed

Definition at line 119 of file dns.h.

#define DNS_EVENT_ERROR 0

Error event reported by DNS client to event_listener

Definition at line 84 of file dns.h.

#define DNS_EVENT_SUCCESS 1

Resolving successfull event reported by DNS client to event_listener

Definition at line 87 of file dns.h.

#define DNS_NUM_RETRIES 5

Number of retries that DNS client will perform before aborting name resolving

Definition at line 78 of file dns.h.

#define DNS_RESEND_PERIOD 2

Period in seconds for resending DNS requests

Definition at line 75 of file dns.h.

#define DNS_SERVER_IP 0xac100201

DNS server's IP address

Definition at line 73 of file dns.h.

#define DNS_UDP_PORT 53

DNS client will use this port for sending and receiving of DNS packets

Definition at line 69 of file dns.h.

Function Documentation

INT32 dns_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *ipaddr*, UINT16 *port*, UINT16 *buffindex*, UINT16 *datalen*)

DNS client event listener.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Parameters:

cbhandle handle of the socket this packet is intended for.
event event that is notified. For UDP, only UDP_EVENT_DATA.
ipaddr IP address of remote host who sent the UDP datagram
port port number of remote host who sent the UDP datagram

buffindex buffer index in RTL8019AS

Returns:

- - 1 - error in processing
- 0 - DNS reply successfully processed

Note:

- Event listeners are NOT to be invoked directly. They are callback functions invoked by the TCP/IP stack to notify events.

This, of course, is where responses from DNS server are processed and checked whether they contain the IP address we requested or if they contain authorative name server to which we should proceed.

If we received the IP address we requested, DNS_EVENT_SUCCESS is reported to application DNS event listener. Otherwise DNS_EVENT_ERROR is reported.

Definition at line 205 of file dns.c.

void dns_init (void)

Initialize resources needed for the DNS client.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Invoke this function at startup to properly initialize DNS resources.

Definition at line 108 of file dns.c.

void dns_run (void)

DNS client main loop.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Simple main loop that checks whether DNS requests should be sent or not (based on timer timeout). If yes, and DNS is in appropriate state, dns_retransmit() function is invoked.

Definition at line 168 of file dns.c.

INT16 get_host_by_name (UINT8 * *host_name_ptr*, void(* *listener*)(UINT8, UINT32))

Invokes DNS resolver.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

10.10.2002

Parameters:

host_name_ptr Pointer to null-terminated host name to be resolved
listener Pointer to DNS listener function that listens to events from DNS client. This function takes two parameters: first one can take a value of DNS_EVENT_SUCCESS or DNS_EVENT_ERROR and thus determine the meaning of the second parameter. If first parameter is DNS_EVENT_SUCCESS, second parameter represents requested IP address. In case of DNS_EVENT_ERROR, second parameter can be one of the: DNS_ERROR_FORMAT, DNS_ERROR_SERVER_FAILURE, DNS_ERROR_NAME_ERROR, DNS_ERROR_NOT_IMPLEMENTED, DNS_ERROR_REFUSED, DNS_ERROR_TIMEOUT, DNS_ERROR_GENERAL

Returns:

- DNS_ERROR_BUSY - Signals that DNS is currently processing another request so it is not possible to process a new one
- DNS_ERROR_OVERFLOW - Network transmit buffer too small to hold DNS request
- DNS_ERROR_LABEL - Label in host name longer than 63 bytes. Error
- DNS_ERROR_NAME - Host name longer than 264 bytes. Error

Invoke this function to start name-resolving process. Note that currently DNS client can process only one request at a time and will not allow multiple requests.

Definition at line 448 of file dns.c.

---

ethernet.h File Reference

Data Structures

- struct ethernet_frame
  Ethernet packet header fields.

Defines

- #define ETH_ADDRESS_LEN 6
- #define ETH_HEADER_LEN 14
- #define ETH_CHIP_HEADER_LEN 4
- #define ETH_MTU 1500
- #define PROTOCOL_IP 0x0800
- #define PROTOCOL_IPv6 0x86DD
- #define PROTOCOL_ARP 0x0806
- #define ARP_BUFFER 0x5F
- #define ICMP_BUF 0x4D
- #define TCP_BUF 0x53
- #define UDP_BUF 0x59

Define Documentation

#define ARP_BUFFER 0x5F

256 byte Tx for ARP

Definition at line 20 of file ethernet.h.

#define ETH_ADDRESS_LEN 6

Definition at line 7 of file ethernet.h.

#define ETH_CHIP_HEADER_LEN 4

Definition at line 12 of file ethernet.h.

#define ETH_HEADER_LEN 14

Definition at line 11 of file ethernet.h.

#define ETH_MTU 1500

Definition at line 13 of file ethernet.h.

#define ICMP_BUF 0x4D

1536 byte Tx for ICMP

Definition at line 21 of file ethernet.h.

#define PROTOCOL_ARP 0x0806

ARP over Ethernet

Definition at line 17 of file ethernet.h.

#define PROTOCOL_IP 0x0800

IP over Ethernet

Definition at line 15 of file ethernet.h.

#define PROTOCOL_IPv6 0x86DD

IPv6 over Ethernet

Definition at line 16 of file ethernet.h.

#define TCP_BUF 0x53

1536 byte Tx for TCP

Definition at line 22 of file ethernet.h.

#define UDP_BUF 0x59

1536 byte Tx for UDP

Definition at line 23 of file ethernet.h.

FileSys.c File Reference

```
#include "FileSys.h"
#include "index.h"
#include "fs_anilogo.h"
```

Variables

- const TFileEntry FAT []

Variable Documentation

const TFileEntry FAT[]

Initial value:

```
{
    {  115,    index_file,      INDEX_FILE_LEN      },
    {          4,    fs_anilogo_file,      FS_ANILOGO_FILE_LEN  },

    {  0,      (unsigned char *)0,  0  }
}
```

Definition at line 20 of file FileSys.c.

FileSys.h File Reference

Data Structures

- struct TFileEntry

Typedefs

- typedef TFileEntry TFileEntry

Variables

- const TFileEntry FAT []

Typedef Documentation

typedef struct TFileEntry TFileEntry

Variable Documentation

const TFileEntry FAT[]

Definition at line 23 of file FileSys.h.

---

globalvariables.h File Reference

Detailed Description

OpenTCP global variables declarations.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

10.7.2002

Here are declarations of global variables that are commonly used in other OpenTCP modules as well as OpenTCP applications in general. Basically just a bunch of externs.
Definition in file globalvariables.h.

```
#include "ip.h"
```

Variables

- UINT32 base_timer
- UINT8 net_buf []
  Transmit buffer used by all OpenTCP applications.

- ethernet_frame received_frame
- ethernet_frame send_frame
- netif localmachine
- ip_frame received_ip_packet
  Used for storing various information about the incoming IP packet.

- ip_frame send_ip_packet
  Used for storing various information about the outgoing IP packet.

- udp_frame received_udp_packet
  Used for storing field information about the received UDP packet.

- tcp_frame received_tcp_packet
  Used for storing field information about the received TCP packet.

Variable Documentation

UINT32 base_timer

System 1.024 msec timer

Definition at line 71 of file globalvariables.h.

struct netif localmachine

MUST BE PUT SOMEWHERE

Definition at line 80 of file globalvariables.h.

UINT8 net_buf[]

Transmit buffer used by all OpenTCP applications.

This buffer is the transmit buffer used by all OpenTCP applications for sending of data. Please note the warnings below for correct usage of this buffer that ensures proper operation of the applications.

Warning:

- **Transmit buffer start** - to avoid data copying, the TCP/IP stack will use first part of the net_buf buffer to add it's data. This means that applications using TCP and/or UDP **must not** write application-level data from the beginning of the buffer but from certain offset. This offset depends on the transport-layer protocol (it's header size that is). For TCP this value is defined by the TCP_APP_OFFSET and for the UDP it is UDP_APP_OFFSET.
- **Buffer sharing** - since all applications share this buffer among each other, and with the TCP/IP stack as well, care must be taken not to overwrite other applications' data before it is sent. This is best achieved if all applications work in the main loop and when they wish to send data they fill in the buffer and send it immediately.

Definition at line 74 of file globalvariables.h.

struct ethernet_frame received_frame

See ethernet.h

Definition at line 78 of file globalvariables.h.

struct ip_frame received_ip_packet

Used for storing various information about the incoming IP packet.

Various fields from the IP packet are stored in this structure. These values are later used from other upper layer protocols (ICMP, UDP, TCP and possibly others) to extract needed information about the received packet. See ip_frame definition for struct information.

Definition at line 81 of file globalvariables.h.

struct tcp_frame received_tcp_packet

Used for storing field information about the received TCP packet.

Various fields from the TCP packet are stored in this variable. These values are then used to perform the necessary actions as defined by the TCP specification: correctnes of the received TCP packet is checked by analyzing these fields, appropriate socket data is adjusted and/or control packet is sent based on it. See tcp_frame definition for struct information.

Definition at line 84 of file globalvariables.h.

struct udp_frame received_udp_packet

Used for storing field information about the received UDP packet.

Various fields from the received UDP packet are stored in this variable. See udp_frame definition for struct information.

Definition at line 83 of file globalvariables.h.

struct ethernet_frame send_frame

See ethernet.h

Definition at line 79 of file globalvariables.h.

struct ip_frame send_ip_packet

Used for storing various information about the outgoing IP packet.

Various fields from the IP packet are stored in this structure. These values are filled based on the information supplied by the upper layer protocols (ICMP, UDP, TCP and possibly others) and used to form a correct IP packet (correct filed values, checksum,..). See ip_frame definition for struct information.

Definition at line 82 of file globalvariables.h.

http_server.c File Reference

```
#include "datatypes.h"
#include "globalvariables.h"
#include "debug.h"
#include "system.h"
#include "tcp_ip.h"
#include "http_server.h"
```

Functions

- INT8 https_init (void)
  Initialize HTTP server variables.

- void https_run (void)
- INT32 https_eventlistener (INT8 cbhandle, UINT8 event, UINT32 par1, UINT32 par2)
- void https_deletesession (UINT8 ses)
- INT16 https_searchsession (UINT8 soch)
- INT16 https_bindsession (UINT8 soch)
- void https_activatesession (UINT8 ses)
- UINT8 https_read_encoded (void)
- INT16 https_calculatehash (UINT32 len)

Variables

- UINT8 https_enabled = 0
- http_server_state https [NO_OF_HTTP_SESSIONS]
  Used for storing state information about different HTTP sessions.

Function Documentation

void https_activatesession (UINT8 *ses*)

Definition at line 408 of file http_server.c.

INT16 https_bindsession (UINT8 *soch*)

Definition at line 386 of file http_server.c.

INT16 https_calculatehash (UINT32 *len*)

Definition at line 457 of file http_server.c.

void https_deletesession (UINT8 *ses*)

Definition at line 361 of file http_server.c.

INT32 https_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *par1*, UINT32 *par2*)

Definition at line 202 of file http_server.c.

INT8 https_init (void)

Initialize HTTP server variables.

Author:

- Jari Lahti (jari.lahti@violasysems.com)

Date:

13.10.2002

This function should be called before the HTTP Server application is used to set the operating parameters of it

Definition at line 27 of file http_server.c.

UINT8 https_read_encoded (void)

Definition at line 417 of file http_server.c.

void https_run (void)

Definition at line 87 of file http_server.c.

INT16 https_searchsession (UINT8 *soch*)

Definition at line 371 of file http_server.c.

Variable Documentation

struct http_server_state https[NO_OF_HTTP_SESSIONS]

Used for storing state information about different HTTP sessions.

This is an array of http_server_state structures holding various state information about the HTTP sessions. HTTP server uses this information to determine actions that need to be taken on sockets.

Definition at line 17 of file http_server.c.

UINT8 https_enabled = 0

Defines whether https_init has already been invoked or not

Definition at line 9 of file http_server.c.

---

http_server.h File Reference

```
#include "datatypes.h"
```

Data Structures

- struct http_server_state
  Structure that holds all the necessary state information for session management.

Defines

- #define NO_OF_HTTP_SESSIONS 3
  Defines number of simultaneous HTTP sessions.

- #define HTTPS_SERVERPORT 80
  HTTP server port on which we'll listen.

- #define HTTPS_STATE_FREE 1
- #define HTTPS_STATE_RESERVED 2
- #define HTTPS_STATE_ACTIVE 3

Functions

- INT32 https_eventlistener (INT8, UINT8, UINT32, UINT32)
- INT8 https_init (void)
  Initialize HTTP server variables.

- void https_run (void)
- void https_deletesession (UINT8)

- INT16 https_searchsession (UINT8)
- INT16 https_bindsession (UINT8)
- void https_activatesession (UINT8)
- INT16 https_calculatehash (UINT32)
- INT16 https_findfile (UINT8, UINT8)
  Brief function description here.

- INT16 https_loadbuffer (UINT8, UINT8 *, UINT16)
  Fill network transmit buffer with HTTP headers&data.

Variables

- http_server_state https []
  Used for storing state information about different HTTP sessions.

Define Documentation

#define HTTPS_SERVERPORT 80

HTTP server port on which we'll listen.

This defines on what TCP port the HTTP server will listen for incoming connections/requests. For HTTP standard port is 80.

Definition at line 21 of file http_server.h.

#define HTTPS_STATE_ACTIVE 3

HTTP Server state: session entry (and the session itself) are active.

Definition at line 30 of file http_server.h.

#define HTTPS_STATE_FREE 1

HTTP Server state: session entry free and available

Definition at line 24 of file http_server.h.

#define HTTPS_STATE_RESERVED 2

HTTP Server state: session entry is reserved and therefore not available

Definition at line 27 of file http_server.h.

#define NO_OF_HTTP_SESSIONS 3

Defines number of simultaneous HTTP sessions.

Change this define to change how many simultaneous HTTP sessions will be possible at any given time. Note that this will require at least as much TCP sockets, so change NO_OF_TCPSOCKETS also!

Definition at line 13 of file http_server.h.

Function Documentation

void https_activatesession (UINT8)

Definition at line 408 of file http_server.c.

INT16 https_bindsession (UINT8)

Definition at line 386 of file http_server.c.

INT16 https_calculatehash (UINT32)

Definition at line 457 of file http_server.c.

void https_deletesession (UINT8)

Definition at line 361 of file http_server.c.

INT32 https_eventlistener (INT8, UINT8, UINT32, UINT32)

Definition at line 202 of file http_server.c.

INT16 https_findfile (UINT8 *hash*, UINT8 *ses*)

Brief function description here.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Parameters:

hash Calculated file-name hash value. Used so that the whole file name doesn't need to be stored in RAM
ses HTTP session identifier

Returns:

- -1 - This function should return -1 if no file has been found
- 1 - This function should return 1 if a file with appropriate hash value has been found.

Warning:

- This function **MUST** be implemented by user application to work with local configuration

This function is invoked by the HTTP server once a hash value of a requested file name has been calculated. User application uses this hash value to check if appropriate file is available to web server. Appropriate https session entry is then filled accordingly.

Definition at line 37 of file https_callbacks.c.

INT8 https_init (void)

Initialize HTTP server variables.

Author:

- Jari Lahti (jari.lahti@violasysems.com)

Date:

13.10.2002

This function should be called before the HTTP Server application is used to set the operating parameters of it

Definition at line 27 of file http_server.c.

INT16 https_loadbuffer (UINT8 *ses*, UINT8 * *buf*, UINT16 *buflen*)

Fill network transmit buffer with HTTP headers&data.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Parameters:

ses HTTP session identifier

buf Pointer to buffer where data is to be stored
buflen Length of the buffer in bytes

Returns:

- >=0 - Number of bytes written to buffer

Warning:

- This function **MUST** be implemented by user application to work with local configuration

This handlers' job is to fill the buffer with the data that web server should return back through the TCP connection. This is accomplished based session identifer and values of variables in appropriate https entry.

Definition at line 105 of file https_callbacks.c.

void https_run (void)

Definition at line 87 of file http_server.c.

INT16 https_searchsession (UINT8)

Definition at line 371 of file http_server.c.

Variable Documentation

struct http_server_state https[]

Used for storing state information about different HTTP sessions.

This is an array of http_server_state structures holding various state information about the HTTP sessions. HTTP server uses this information to determine actions that need to be taken on sockets.

Definition at line 101 of file http_server.h.

---

https_callbacks.c File Reference

```
#include "datatypes.h"
#include "debug.h"
```

```
#include "globalvariables.h"
#include "system.h"
#include "http_server.h"
#include "FileSys.h"
```

Functions

- INT16 https_findfile (UINT8 hash, UINT8 ses)
  Brief function description here.

- INT16 https_loadbuffer (UINT8 ses, UINT8 *buf, UINT16 buflen)
  Fill network transmit buffer with HTTP headers&data.

Variables

- const char https_not_found_page [] = "HTTP/1.0 200 OK\r\nLast-modified: Mon, 17 May 2004 15:02:45 GMT\r\nServer: ESERV-10/1.0\nContent-type: text/html\r\nContent-length: 400\r\n\r\n<HEAD><TITLE>Viola Systems Embedded WEB Server</TITLE></HEAD><BODY><H2>HTTP 1.0 404 Error. File Not Found</H2>The requested URL was not found on this server.<HR>
<I>Viola Systems Embedded WEB Server 2.01, 2004
<A HREF=http://192.168.2.3/index.htm>Did you wish 192.168.2.3/index.htm?</I></A>
<A HREF=http://www.violasystems.com>www.violasystems.com - Embedding The Internet</A></BODY>"
  File not found message.

Function Documentation

INT16 https_findfile (UINT8 *hash*, UINT8 *ses*)

Brief function description here.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Parameters:

hash Calculated file-name hash value. Used so that the whole file name doesn't need to be stored in RAM
ses HTTP session identifier

Returns:

- -1 - This function should return -1 if no file has been found
- 1 - This function should return 1 if a file with appropriate hash value has been found.

Warning:

- This function **MUST** be implemented by user application to work with local configuration

This function is invoked by the HTTP server once a hash value of a requested file name has been calculated. User application uses this hash value to check if appropriate file is available to web server. Appropriate https session entry is then filled accordingly.

Definition at line 37 of file https_callbacks.c.

INT16 https_loadbuffer (UINT8 *ses*, UINT8 * *buf*, UINT16 *buflen*)

Fill network transmit buffer with HTTP headers&data.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

09.10.2002

Parameters:

ses HTTP session identifier
buf Pointer to buffer where data is to be stored
buflen Length of the buffer in bytes

Returns:

- >=0 - Number of bytes written to buffer

Warning:

- This function **MUST** be implemented by user application to work with local configuration

This handlers' job is to fill the buffer with the data that web server should return back through the TCP connection. This is accomplished based session identifer and values of variables in appropriate https entry.

Definition at line 105 of file https_callbacks.c.

Variable Documentation

**const char https_not_found_page[] = "HTTP/1.0 200 OK\r\nLast-modified: Mon, 17 May 2004 15:02:45 GMT\r\nServer: ESERV-10/1.0\nContent-type: text/html\r\nContent-length: 400\r\n\r\n<HEAD><TITLE>Viola Systems Embedded WEB Server</TITLE></HEAD><BODY><H2>HTTP 1.0 404 Error. File Not Found</H2>The requested URL was not found on this server.<HR>
<I>Viola Systems Embedded WEB Server 2.01, 2004
<A HREF=http://192.168.2.3/index.htm>Did you wish 192.168.2.3/index.htm?</I></A>
<A HREF=http://www.violasystems.com>www.violasystems.com - Embedding The Internet</A></BODY>"**

File not found message.

Message that will be displayed if a file with appropriate name (hash value) was not found.

Definition at line 14 of file https_callbacks.c.

---

icmp.c File Reference

Detailed Description

OpenTCP ICMP implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

8.7.2002

Bug:

Warning:

Todo:

- Add more functionality, not just ICMP Echo request/reply (possibly Destination unreachable processing)
- IP address setting option should be runtime or #define configurable

OpenTCP ICMP implementation. Functions and other ICMP-related stuff is declared in tcp_ip.h.
Definition in file icmp.c.

```
#include "debug.h"
#include "datatypes.h"
#include "ethernet.h"
#include "ip.h"
#include "tcp_ip.h"
#include "system.h"
```

Functions

- INT16 process_icmp_in (struct ip_frame *frame, UINT16 len)
  Process recieved ICMP datagram.

Function Documentation

INT16 process_icmp_in (struct ip_frame * *frame*, UINT16 *len*)

Process recieved ICMP datagram.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

08.07.2002

Parameters:

frame - pointer to received IP frame structure
len - length of the received IP datagram (in bytes)

Returns:

- -1 - packet not OK (not proper ICMP or not ICMP at all)
- >=0 - packet OK

Invoke process_icmp_in whenever IP datagram containing ICMP message is detected (see main_demo.c for example main loop implementing this).

This function simply checks correctnes of received ICMP message and send ICMP replies when requested.

Definition at line 97 of file icmp.c.

Init.c File Reference

```
#include "datatypes.h"
#include "ne64debug.h"
```

Functions

- void init (void)

Function Documentation

void init (void)

Definition at line 9 of file Init.c.

---

ip.c File Reference

Detailed Description

OpenTCP IP protocol implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)
- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:
1.0

Date:
11.6.2002

Bug:

Warning:

Todo:

- Implement stub handler for supporting fragmented datagrams (may be usefull on MCUs with lots of available RAM)

OpenTCP IP protocol implementation functions. For declaration, constants and data structures refer to ip.h.
Definition in file ip.c.

```
#include "debug.h"
#include "datatypes.h"
#include "ethernet.h"
#include "arp.h"
#include "ip.h"
#include "system.h"
```

Functions

- INT16 process_ip_in (struct ethernet_frame *frame)
  Process received IP frame.

- INT16 process_ip_out (UINT32 ipadr, UINT8 pcol, UINT8 tos, UINT8 ttl, UINT8 *dat, UINT16 len)
  Try to send out IP frame.

- UINT32 ip_construct_cs (struct ip_frame *frame)
  Construct checksum of the IP header.

- UINT8 ip_check_cs (struct ip_frame *frame)
  Check IP frame's checksum.

- UINT16 ip_checksum (UINT16 cs, UINT8 dat, UINT8 count)
  Used for constructuing IP checksum.

- UINT32 ip_checksum_buf (UINT16 cs, UINT8 *buf, UINT16 len)
  Used for constructuing IP checksum of a data buffer.

Variables

- ip_frame received_ip_packet
  Used for storing various information about the incoming IP packet.

- ip_frame send_ip_packet
  Used for storing various information about the outgoing IP packet.

- UINT16 ip_id

Function Documentation

UINT8 ip_check_cs (struct ip_frame * *frame*)

Check IP frame's checksum.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

frame pointer to IP frame to be checked

Returns:

- 0 - checksum corrupted
- 1 - checksum OK

Checksum of an IP packet is calculated and compared with the received checksum. Error is signaled if there is discrepancy between them.

Definition at line 453 of file ip.c.

UINT16 ip_checksum (UINT16 *cs*, UINT8 *dat*, UINT8 *count*)

Used for constructuing IP checksum.

Author:

- Jari Lahti

Date:

24.02.2002

Parameters:

cs last checksum value
dat byte to be added to checksum
count byte indicating whether dat is MSB or LSB byte

Returns:

new checksum value

Based on count value, dat byte is added to checksum either as a MSB or a LSB byte and the new checksum value is then returned.

Definition at line 518 of file ip.c.

UINT32 ip_checksum_buf (UINT16 *cs*, UINT8 * *buf*, UINT16 *len*)

Used for constructuing IP checksum of a data buffer.

Author:

- Jari Lahti

Date:

03.08.2003

Parameters:

cs last checksum value
buf buffer who's checksum we're calculating
len length of data in buffer

Returns:

new checksum value

Calculates checksum of the data in buffer and returns new checksum value.

Definition at line 563 of file ip.c.

UINT32 ip_construct_cs (struct ip_frame * *frame*)

Construct checksum of the IP header.

Author:

- Jari Lahti

Date:

08.07.2002

Parameters:

frame pointer to ip_frame structure holding header information based on which checksum is calculated

Returns:

Calculated checksum

Checksum of the supplied IP datagram is calculated.

Definition at line 396 of file ip.c.

INT16 process_ip_in (struct ethernet_frame * *frame*)

Process received IP frame.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

frame pointer to ethernet_frame structure holding information about the received frame that carries IP datagram.

Returns:

- -1 - IP packet not OK
- >0 - Length of next layer data (IP packet OK)

Process received IP packet by checking necessary header information and storing it accordingly to received_ip_packet variable. If everything checks out, return length of the data carried in the IP datagram (for higher-level protocols), otherwise return -1.

Definition at line 115 of file ip.c.

INT16 process_ip_out (UINT32 *ipadr*, UINT8 *pcol*, UINT8 *tos*, UINT8 *ttl*, UINT8 * *dat*, UINT16 *len*)

Try to send out IP frame.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

ipadr remote IP address
pcol protocol over IP used. Can be one of the following:

- IP_ICMP
- IP_UDP
- IP_TCP

tos type of service required
ttl time to live header field of IP packet
dat pointer to data buffer
len length of data to be sent in IP datagram

Returns:

- -1 - general error
- -2 - ARP cache not ready
- >0 - number of data bytes sent (packet OK)

Invoke this function to perform all of the necessary preparation in order to send out an IP packet. These include:

- Consulting ARP cache for HW address to send the packet to
- Filling send_ip_packet variable with correct values
- Calculating checksum for the IP packet
- Adding datalink header information
- Sending IP header and data
- Instructing NIC to send the data

Definition at line 286 of file ip.c.

Variable Documentation

UINT16 ip_id

ID field in the next IP packet that will be sent

Definition at line 97 of file ip.c.

struct ip_frame received_ip_packet

Used for storing various information about the incoming IP packet.

Various fields from the IP packet are stored in this structure. These values are later used from other upper layer protocols (ICMP, UDP, TCP and possibly others) to extract needed information about the received packet. See ip_frame definition for struct information.

Definition at line 85 of file ip.c.

struct ip_frame send_ip_packet

Used for storing various information about the outgoing IP packet.

Various fields from the IP packet are stored in this structure. These values are filled based on the information supplied by the upper layer protocols (ICMP, UDP, TCP and possibly others) and used to form a correct IP packet (correct filed values, checksum,..). See ip_frame definition for struct information.

Definition at line 95 of file ip.c.

---

ip.h File Reference

Detailed Description

OpenTCP IP interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

11.6.2002

OpenTCP IP function declarations, constants, etc.
Definition in file ip.h.

Data Structures

- struct ip_frame
  IP datagram header fields.

Defines

- #define PHY_ADR_LEN ETH_ADDRESS_LEN
- #define IP_ICMP 0x01
- #define IP_UDP 17
- #define IP_TCP 6
- #define IP_HLEN 20
- #define IP_MIN_HLEN 20
- #define IP_DEF_VIHL 0x45
- #define IP_DEF_TTL 100
- #define MAX_IP_OPTLEN 40
- #define IP_MAX_HLEN IP_MIN_HLEN + MAX_IP_OPTLEN
- #define IP_DONT_FRAGMENT 0x4000
- #define IP_FRAGOFF 0x1FFF
- #define IP_MOREFRAGS 0x2000
- #define IP_GOOD_CS 0
- #define IPO_COPY 0x80
- #define IPO_NOP 0x01
- #define IPO_EOOP 0x00
- #define IP_BROADCAST_ADDRESS 0xFFFFFFFF

Functions

- INT16 process_ip_in (struct ethernet_frame *)
  Process received IP frame.

- INT16 process_ip_out (UINT32, UINT8, UINT8, UINT8, UINT8 *, UINT16)
  Try to send out IP frame.

- UINT8 ip_check_cs (struct ip_frame *)
  Check IP frame's checksum.

- UINT16 ip_checksum (UINT16, UINT8, UINT8)
  Used for constructuing IP checksum.

- UINT32 ip_checksum_buf (UINT16 cs, UINT8 *buf, UINT16 len)

Used for constructuing IP checksum of a data buffer.

- UINT32 ip_construct_cs (struct ip_frame *)
  Construct checksum of the IP header.

Define Documentation

#define IP_BROADCAST_ADDRESS 0xFFFFFFFF

Definition at line 94 of file ip.h.

#define IP_DEF_TTL 100

Definition at line 76 of file ip.h.

#define IP_DEF_VIHL 0x45

Definition at line 75 of file ip.h.

#define IP_DONT_FRAGMENT 0x4000

Definition at line 80 of file ip.h.

#define IP_FRAGOFF 0x1FFF

Definition at line 81 of file ip.h.

#define IP_GOOD_CS 0

Definition at line 84 of file ip.h.

#define IP_HLEN 20

Definition at line 73 of file ip.h.

#define IP_ICMP 0x01

ICMP over IP

Definition at line 69 of file ip.h.

#define IP_MAX_HLEN IP_MIN_HLEN + MAX_IP_OPTLEN

Definition at line 78 of file ip.h.

#define IP_MIN_HLEN 20

Definition at line 74 of file ip.h.

#define IP_MOREFRAGS 0x2000

Definition at line 82 of file ip.h.

#define IP_TCP 6

TCP over IP

Definition at line 71 of file ip.h.

#define IP_UDP 17

UDP over IP

Definition at line 70 of file ip.h.

#define IPO_COPY 0x80

Definition at line 88 of file ip.h.

#define IPO_EOOP 0x00

Definition at line 90 of file ip.h.

#define IPO_NOP 0x01

Definition at line 89 of file ip.h.

#define MAX_IP_OPTLEN 40

Definition at line 77 of file ip.h.

#define PHY_ADR_LEN ETH_ADDRESS_LEN

Lower-layer physical address length

Definition at line 67 of file ip.h.

Function Documentation

UINT8 ip_check_cs (struct ip_frame * *frame*)

Check IP frame's checksum.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

frame pointer to IP frame to be checked

Returns:

- 0 - checksum corrupted
- 1 - checksum OK

Checksum of an IP packet is calculated and compared with the received checksum. Error is signaled if there is discrepancy between them.

Definition at line 453 of file ip.c.

UINT16 ip_checksum (UINT16 *cs*, UINT8 *dat*, UINT8 *count*)

Used for constructuing IP checksum.

Author:

- Jari Lahti

Date:

24.02.2002

Parameters:

cs last checksum value
dat byte to be added to checksum
count byte indicating whether dat is MSB or LSB byte

Returns:

new checksum value

Based on count value, dat byte is added to checksum either as a MSB or a LSB byte and the new checksum value is then returned.

Definition at line 518 of file ip.c.

UINT32 ip_checksum_buf (UINT16 *cs*, UINT8 * *buf*, UINT16 *len*)

Used for constructuing IP checksum of a data buffer.

Author:

- Jari Lahti

Date:

03.08.2003

Parameters:

cs last checksum value
buf buffer who's checksum we're calculating
len length of data in buffer

Returns:

new checksum value

Calculates checksum of the data in buffer and returns new checksum value.

Definition at line 563 of file ip.c.

UINT32 ip_construct_cs (struct ip_frame * *frame*)

Construct checksum of the IP header.

Author:

- Jari Lahti

Date:

08.07.2002

Parameters:

frame pointer to ip_frame structure holding header information based on which checksum is calculated

Returns:

Calculated checksum

Checksum of the supplied IP datagram is calculated.

Definition at line 396 of file ip.c.

INT16 process_ip_in (struct ethernet_frame * *frame*)

Process received IP frame.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

frame pointer to ethernet_frame structure holding information about the received frame that carries IP datagram.

Returns:

- -1 - IP packet not OK
- >0 - Length of next layer data (IP packet OK)

Process received IP packet by checking necessary header information and storing it accordingly to received_ip_packet variable. If everything checks out, return length of the data carried in the IP datagram (for higher-level protocols), otherwise return -1.

Definition at line 115 of file ip.c.

INT16 process_ip_out (UINT32 *ipadr*, UINT8 *pcol*, UINT8 *tos*, UINT8 *ttl*, UINT8 * *dat*, UINT16 *len*)

Try to send out IP frame.

Author:

- Jari Lahti

Date:

11.06.2002

Parameters:

ipadr remote IP address
pcol protocol over IP used. Can be one of the following:

- IP_ICMP
- IP_UDP
- IP_TCP

tos type of service required
ttl time to live header field of IP packet
dat pointer to data buffer
len length of data to be sent in IP datagram

Returns:

- -1 - general error
- -2 - ARP cache not ready
- >0 - number of data bytes sent (packet OK)

Invoke this function to perform all of the necessary preparation in order to send out an IP packet. These include:

- Consulting ARP cache for HW address to send the packet to
- Filling send_ip_packet variable with correct values
- Calculating checksum for the IP packet
- Adding datalink header information
- Sending IP header and data
- Instructing NIC to send the data

Definition at line 286 of file ip.c.

---

license.txt File Reference

Functions

- Copyright (c) 2000-2002 Viola Systems Ltd.All rights reserved.Redistribution and use in source and binary forms
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY 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

- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE

Variables

- with or without modification
- with or without are permitted provided that the following conditions are met
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright notice
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the redistribution
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if any
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following acknowledgment
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software itself
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written permission
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote

products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called OpenTCP

- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their name
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED WARRANTIES
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED INCLUDING
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED TO
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT

- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY INDIRECT
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY INCIDENTAL
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY SPECIAL
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY EXEMPLARY
- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this

acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN CONTRACT

- with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT LIABILITY

Function Documentation

Copyright (c)

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED *TO*, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF *USE*, DATA, OR PROFITS;OR BUSINESS *INTERRUPTION*)

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if

and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR *OTHERWISE*)

Variable Documentation

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following acknowledgment

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if any

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN CONTRACT

Definition at line 40 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY EXEMPLARY

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY INCIDENTAL

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the

above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED INCLUDING

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY INDIRECT

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software itself

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT LIABILITY

Definition at line 40 of file license.txt.

with or without are permitted provided that the following conditions are met

Definition at line 5 of file license.txt.

with or without modification

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their name

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright notice

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP

and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called OpenTCP

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written permission

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the redistribution

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL VIOLA SYSTEMS LTD OR ITS CONTRIBUTORS BE LIABLE FOR ANY SPECIAL

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their

without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED BUT NOT LIMITED TO

Definition at line 5 of file license.txt.

with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution The end user documentation included with the if must include the following this acknowledgment may appear in the software if and wherever such third party acknowledgments normally appear The names OpenTCP and Viola Systems must not be used to endorse or promote products derived from this software without prior written permission For written please contact opentcp opentcp org Products derived from this software may not be called nor may OpenTCP appear in their without prior written permission of the Viola Systems Ltd THIS SOFTWARE IS PROVIDED AS IS AND ANY EXPRESSED OR IMPLIED WARRANTIES

Definition at line 5 of file license.txt.

---

main.c File Reference

```
#include "debug.h"
#include "datatypes.h"
#include "timers.h"
#include "system.h"
#include "ethernet.h"
#include "arp.h"
#include "ip.h"
#include "tcp_ip.h"
#include "http_server.h"
#include "smtp_client.h"
#include "ne64driver.h"
#include "ne64api.h"
#include "mBuf.h"
#include "ne64config.h"
#include "udp_demo.h"
#include "address.h"
#include "MC9S12NE64.h"
```

Functions

- void RTI_Enable (void)
- void main (void)
- interrupt void PortHInterrupt (void)

Variables

- netif localmachine
- tU16 gotxflowc
- tU08 gotlink
- tU08 ourbuffer [1518]

Function Documentation

void main (void)

Definition at line 55 of file main.c.

interrupt void PortHInterrupt (void)

Definition at line 192 of file main.c.

void RTI_Enable (void)

Definition at line 32 of file RTI.c.

Variable Documentation

tU08 gotlink

Definition at line 42 of file main.c.

tU16 gotxflowc

Global Variable For Determination of Flow Control Packets are sent in Full Duplex defined in "main.c"

Definition at line 41 of file main.c.

struct netif localmachine

MUST BE PUT SOMEWHERE

Definition at line 37 of file main.c.

tU08 ourbuffer[1518]

Space for packet temporary storage if zero copy not used

Definition at line 46 of file main.c.

---

mBuf.c File Reference

```
#include "mBuf.h"
```

Defines

- #define MBUFS_NUM 2

Functions

- void mBufInit (void)
- INT16 mENQUEUE (MBUF *m)
- MBUF * mDEQUEUE (void)

Variables

- MBUF mBufs [MBUFS_NUM]
- MBUF mBufTx

Define Documentation

#define MBUFS_NUM 2

Definition at line 14 of file mBuf.c.

Function Documentation

void mBufInit (void)

Definition at line 24 of file mBuf.c.

MBUF* mDEQUEUE (void)

Definition at line 75 of file mBuf.c.

INT16 mENQUEUE (MBUF * *m*)

Definition at line 47 of file mBuf.c.

Variable Documentation

MBUF mBufs[MBUFS_NUM]

Definition at line 16 of file mBuf.c.

MBUF mBufTx

Definition at line 17 of file mBuf.c.

---

mBuf.h File Reference

```
#include "MotTypes.h"
```

Data Structures

- struct MBUF

Defines

- #define MBUF_NOTEMPTY 0x0001
- #define mBUFTOPTR(n, t) ((t)((n)))

Typedefs

- typedef MBUF MBUF

Functions

- void mBufInit (void)
- INT16 mENQUEUE (MBUF *m)
- MBUF * mDEQUEUE (void)

Define Documentation

#define MBUF_NOTEMPTY 0x0001

Definition at line 17 of file mBuf.h.

#define mBUFTOPTR(n, t) ((t)((n)))

Definition at line 32 of file mBuf.h.

Typedef Documentation

typedef struct MBUF MBUF

Function Documentation

void mBufInit (void)

Definition at line 24 of file mBuf.c.

MBUF* mDEQUEUE (void)

Definition at line 75 of file mBuf.c.

INT16 mENQUEUE (MBUF * *m*)

Definition at line 47 of file mBuf.c.

---

ne64api.c File Reference

```
#include "MotTypes.h"
#include <string.h>
#include "MC9S12NE64.h"
#include "ne64api.h"
#include "mBuf.h"
```

Functions

- void NE64InitializeOffsetToReadRxBuffer (UINT16 offset)
- UINT8 NE64ReadByte (void)
  NE64ReadByte: Read a byte from the RX buffer.
- UINT16 NE64ReadWord (void)
- void NE64ReadBytes (UINT8 *buf, UINT16 len)
- void NE64InitializeTransmissionBuffer (UINT8 page)
- void NE64WriteEthernetHeaderToTxBuffer (struct TEthernetFrame *frame)
- void NE64StartFrameTransmission (UINT16 len)
- void NE64WriteByte (UINT8 dat)
- void NE64WriteWord (UINT16 dat)
- void NE64WriteBytes (UINT8 *buf, UINT16 len)
- UINT16 NE64Receive (void *PktBuffer, UINT16 len, UINT16 flags)
- UINT16 NE64ValidFrameReception (void)
  NE64ValidFrameReception: Checks for valid reception of data in the RX buffers.
- void NE64FreeReceiveBuffer (void)
  NE64FreeReceiveBuffer: Clear RX receive flag.
- void * NE64GetCurrentReceivedFrame (void)
  NE64FreeReceiveBuffer: Clear RX receive flag.

Variables

- void * emacFIFOa []
- void * emacFIFOb []
- void * emacFIFOtx []
- MBUF mBufTx
- MBUF * pCurrentMBuf = (void *)0
- MBUF * pCurrentSendMBuf = (void *)0
- TEthernetFrame received_frame
- TEthernetFrame send_frame

Function Documentation

void NE64FreeReceiveBuffer (void)

NE64FreeReceiveBuffer: Clear RX receive flag.

This function clears the RX receive flag and makes the buffer availible for the next packet to be recieved. This function also initializes the RX buffer pointers

Parameters:

None

Returns:

None

Todo:

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the RX buffer

Definition at line 199 of file ne64api.c.

void* NE64GetCurrentReceivedFrame (void)

NE64FreeReceiveBuffer: Clear RX receive flag.

Parameters:

None

Returns:

None

Todo:

This function needs more testing

Bug:

Warning:

None

Definition at line 218 of file ne64api.c.

void NE64InitializeOffsetToReadRxBuffer (UINT16 *offset*)

Definition at line 34 of file ne64api.c.

void NE64InitializeTransmissionBuffer (UINT8 *page*)

Definition at line 72 of file ne64api.c.

UINT8 NE64ReadByte (void)

NE64ReadByte: Read a byte from the RX buffer.

This function reads a byte from the RX buffer and increments the RX buffer pointer. The buffer read is buffer that triggered reception of valid data.

Parameters:

none

Returns:

Data read from buffer

Todo:

This function need more testing

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the RX buffer

Definition at line 41 of file ne64api.c.

void NE64ReadBytes (UINT8 * *buf*, UINT16 *len*)

Definition at line 64 of file ne64api.c.

UINT16 NE64ReadWord (void)

Definition at line 51 of file ne64api.c.

UINT16 NE64Receive (void * *PktBuffer*, UINT16 *len*, UINT16 *flags*)

Definition at line 154 of file ne64api.c.

void NE64StartFrameTransmission (UINT16 *len*)

Definition at line 109 of file ne64api.c.

UINT16 NE64ValidFrameReception (void)

NE64ValidFrameReception: Checks for valid reception of data in the RX buffers.

This function checks for valid reception of data in the RX buffers. This function also records which buffer has the valid data in a global variable.

Moreover, this function stores the packet data including: the destination MAC HW address, the source MAC HW address, the packet length/type field, and the size in bytes of the Ethernet MAC header. This information is stored in a global variable/received_frame structure .

Parameters:

None

Returns:

Returns TRUE of the RX buffers have valid data in the buffers

Todo:

This function need more testing

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the receive buffer

Definition at line 189 of file ne64api.c.

void NE64WriteByte (UINT8 *dat*)

Definition at line 118 of file ne64api.c.

void NE64WriteBytes (UINT8 * *buf*, UINT16 *len*)

Definition at line 138 of file ne64api.c.

void NE64WriteEthernetHeaderToTxBuffer (struct TEthernetFrame * *frame*)

Definition at line 87 of file ne64api.c.

void NE64WriteWord (UINT16 *dat*)

Definition at line 126 of file ne64api.c.

Variable Documentation

void* emacFIFOa[]

Definition at line 18 of file ne64api.c.

void* emacFIFOb[]

Definition at line 19 of file ne64api.c.

void* emacFIFOtx[]

Definition at line 20 of file ne64api.c.

MBUF mBufTx

Definition at line 21 of file ne64api.c.

MBUF* pCurrentMBuf = (void *)0

Definition at line 24 of file ne64api.c.

MBUF* pCurrentSendMBuf = (void *)0

Definition at line 25 of file ne64api.c.

struct TEthernetFrame received_frame

See ethernet.h

Definition at line 29 of file ne64api.c.

struct TEthernetFrame send_frame

See ethernet.h

Definition at line 30 of file ne64api.c.

---

ne64api.h File Reference

```
#include "MotTypes.h"
```

Data Structures

- struct TEthernetFrame

Defines

- #define ETH_ADDRS_LEN 6
- #define ETH_HDR_LEN 14

Typedefs

- typedef TEthernetFrame TEthernetFrame

Functions

- void NE64WriteByte (tU08 dat)
  NE64WriteByte: Writes a byte to the TX buffer.

- void NE64WriteWord (tU16 dat)
- void NE64WriteBytes (tU08 *buf, tU16 len)
  NE64WriteBytes: Writes a given number of data, len, to the TX buffer.

- UINT8 NE64ReadByte (void)
  NE64ReadByte: Read a byte from the RX buffer.

- UINT16 NE64ReadWord (void)
- void NE64ReadBytes (tU08 *buf, tU16 len)
  NE64ReadBytes: Reads a specified number of bytes from the RX buffer.

- void NE64InitializeTransmissionBuffer (tU08 page)
  NE64InitializeTransmissionBuffer: Initializes the TX buffer pointer so that it points to the first byte location in the buffer.

- void NE64InitializeOffsetToReadRxBuffer (tU16 pos)
  NE64InitializeOffsetToReadRxBuffer: Initializes the offset in the receieve buffer to start reading data from.

- void NE64StartFrameTransmission (tU16 len)
  NE64StartFrameTransmission: Start Transmission of data in TX buffer.

- void NE64WriteEthernetHeaderToTxBuffer (struct ethernet_frame *frame)
  NE64WriteEthernetHeaderToTxBuffer: Writes the Ethernet Header into the TX buffer.

- UINT16 NE64ValidFrameReception (void)
  NE64ValidFrameReception: Checks for valid reception of data in the RX buffers.

- void NE64FreeReceiveBuffer (void)
  NE64FreeReceiveBuffer: Clear RX receive flag.

- void * NE64GetCurrentReceivedFrame (void)
  NE64FreeReceiveBuffer: Clear RX receive flag.

Define Documentation

#define ETH_ADDRS_LEN 6

Definition at line 17 of file ne64api.h.

#define ETH_HDR_LEN 14

Definition at line 18 of file ne64api.h.

Typedef Documentation

typedef struct TEthernetFrame TEthernetFrame

Function Documentation

void NE64FreeReceiveBuffer (void)

NE64FreeReceiveBuffer: Clear RX receive flag.

This function clears the RX receive flag and makes the buffer availible for the next packet to be recieved. This function also initializes the RX buffer pointers

Parameters:

None

Returns:

None

Todo:

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the RX buffer

Definition at line 199 of file ne64api.c.

void* NE64GetCurrentReceivedFrame (void)

NE64FreeReceiveBuffer: Clear RX receive flag.

Parameters:

None

Returns:

None

Todo:

This function needs more testing

Bug:

Warning:

None

Definition at line 218 of file ne64api.c.

void NE64InitializeOffsetToReadRxBuffer (tU16 *pos*)

NE64InitializeOffsetToReadRxBuffer: Initializes the offset in the receieve buffer to start reading data from.

This function initializes the offset in the receieve buffer to start reading data from by incrementing the RX buffer pointer to the desired location. The buffer initilized is set to the last buffer that triggered reception of valid data

Parameters:

pos - offset for receive buffer pointer

Returns:

No return value

Todo:

This function need more testing

Bug:

Warning:

Do not clear the data in the RX buffer until the data is not needed. This function needs a global variable to point to the RX buffer

void NE64InitializeTransmissionBuffer (tU08 *page*)

NE64InitializeTransmissionBuffer: Initializes the TX buffer pointer so that it points to the first byte location in the buffer.

This function initializes the TX buffer pointer to point so that it points to the first byte location in the buffer. This function needs a global variable to point to the Tx buffer.

Parameters:

page

Returns:

No return value

Todo:

This function need more testing

Bug:

Warning:

This function needs a global variable to point to the Tx buffer

UINT8 NE64ReadByte (void)

NE64ReadByte: Read a byte from the RX buffer.

This function reads a byte from the RX buffer and increments the RX buffer pointer. The buffer read is buffer that triggered reception of valid data.

Parameters:

none

Returns:

Data read from buffer

Todo:

This function need more testing

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the RX buffer

Definition at line 41 of file ne64api.c.

void NE64ReadBytes (tU08 * *buf*, tU16 *len*)

NE64ReadBytes: Reads a specified number of bytes from the RX buffer.

This function reads a specified number of bytes, length, from the RX buffer and increments the RX buffer pointer. The buffer read is buffer that triggered reception of valid data.

Parameters:

buf - pointer to a buffer were the read data is placed

len - number of bytes to read

Returns:

None

Todo:

This function needs more testing Use word access instead of byte access to speed performance

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the RX buffer

UINT16 NE64ReadWord (void)

Definition at line 51 of file ne64api.c.

void NE64StartFrameTransmission (tU16 *len*)

NE64StartFrameTransmission: Start Transmission of data in TX buffer.

This function start Transmission of data in TX buffer

Parameters:

len - Length of data to transmit (NOTE: this value does NOT include the nuber of byte in the MAC Ethernet Header.

Returns:

No return value

Todo:

This function need more testing

Bug:

Warning:

This function need a global variable to point to the TX buffer

UINT16 NE64ValidFrameReception (void)

NE64ValidFrameReception: Checks for valid reception of data in the RX buffers.

This function checks for valid reception of data in the RX buffers. This function also records which buffer has the valid data in a global variable.

Moreover, this function stores the packet data including: the destination MAC HW address, the source MAC HW address, the packet length/type field, and the size in bytes of the Ethernet MAC header. This information is stored in a global variable/received_frame structure .

Parameters:

None

Returns:

Returns TRUE of the RX buffers have valid data in the buffers

Todo:

This function need more testing

Bug:

Warning:

Do not clear the data in the recieve buffer until the data is not needed. This function needs a global variable to point to the receive buffer

Definition at line 189 of file ne64api.c.

void NE64WriteByte (tU08 *dat*)

NE64WriteByte: Writes a byte to the TX buffer.

This function writes a byte to the TX buffer This function also increments the pointer of to the next byte in the TX buffer.

Parameters:

dat - data byte to write to the TX buffer

Returns:

No return value

Todo:

This function need more testing Use word access instead of byte access to speed performance

Bug:

Warning:

This functions need a global variable to point to the TX buffer

void NE64WriteBytes (tU08 * *buf*, tU16 *len*)

NE64WriteBytes: Writes a given number of data, len, to the TX buffer.

This function writes a given number of data, len, to the TX buffer. This function also increments the pointer of to the next byte in the TX buffer.

Parameters:

buf - pointer to data to write to the TX buffer
len - number of bytes to write to the TX buffer

Returns:

No return value

Todo:

This function need more testing Use word access instead of byte access to speed performance

Bug:

Warning:

This function need a global variable to point to the TX buffer

void NE64WriteEthernetHeaderToTxBuffer (struct ethernet_frame * *frame*)

NE64WriteEthernetHeaderToTxBuffer: Writes the Ethernet Header into the TX buffer.

This function Writes the MAC Ethernet Header into the TX buffer. The Ethernet Header includes the MAC destination addres, MAC source addres, and the length/type fields. This function also increments the pointer of to the next byte in the TX buffer after the MAC Ethernet Header.

Parameters:

frame - Pointer to ethernet_frame

Returns:

No return value

Todo:

This function need more testing

Bug:

Warning:

NE64InitializeTransmissionBuffer needs to be called before NE64WriteEthernetHeaderToTxBuffer

void NE64WriteWord (tU16 *dat*)

---

ne64config.h File Reference

Defines

- #define WORD_ACCESS 1
- #define ZERO_COPY 1
- #define RX_POLL_MODE 0
- #define AUTO_NEG 1
- #define HALF100 1
- #define FULL100 1
- #define HALF10 1
- #define FULL10 1
- #define AUTO_NEG_TIMEOUT 0
- #define BUFMAP 4
- #define EMAC_RX_SZ 1536
- #define EMAC_TX_SZ 1536
- #define BRODC_REJ 0
- #define CON_MULTIC 0
- #define PROM_MODE 0
- #define ETYPE_PET 0
- #define ETYPE_EMW 0
- #define ETYPE_IPV6 0
- #define ETYPE_ARP 0

- #define ETYPE_IPV4 0
- #define ETYPE_IEEE 0
- #define ETYPE_ALL 1
- #define ETYPE_PRG 0
- #define RX_MAX_FL 1536
- #define DELETE_BFRAMES 0
- #define XFLOWC 0
- #define PAUSE_TIME 5
- #define SEND_PAUSE 1
- #define READ_PTIME 0
- #define PHY_ADDRESS 0
- #define BUS_CLOCK 25000000
- #define USE_SWLED 1
- #define ACTLED 1
- #define LNKLED 1
- #define SPDLED 1
- #define DUPLED 1
- #define COLLED 1
- #define USE_EXTBUS 0
- #define IEEE_PKT 0
- #define ON_OFF_AUTONEG 0
- #define READ_PHY_ID 0
- #define READ_ALL_REGS 0

Define Documentation

#define ACTLED 1

If USE_SWLED=1, use software to drive an EPHY activity LED

Definition at line 128 of file ne64config.h.

#define AUTO_NEG 1

1 - enable AUTO_NEG / 0 - disable AUTO_NEG

Definition at line 34 of file ne64config.h.

#define AUTO_NEG_TIMEOUT 0

1 - NOT IMPLEMENTED

Definition at line 45 of file ne64config.h.

#define BRODC_REJ 0

1 = All broadcast address frames are rejected.

Definition at line 86 of file ne64config.h.

#define BUFMAP 4

User select BUFMAP based on application buffer requirements

Definition at line 60 of file ne64config.h.

#define BUS_CLOCK 25000000

Busclock setting set be the CRG/PLL

Definition at line 119 of file ne64config.h.

#define COLLED 1

If USE_SWLED=1, use software to drive an EPHY collision LED

Definition at line 132 of file ne64config.h.

#define CON_MULTIC 0

1 = Multicast hash table is used for checking multicast addresses.

Definition at line 87 of file ne64config.h.

#define DELETE_BFRAMES 0

set to 1 to delete packets larger the maxiuim frame length (babbling error)

Definition at line 104 of file ne64config.h.

#define DUPLED 1

If USE_SWLED=1, use software to drive an EPHY duplex LED

Definition at line 131 of file ne64config.h.

#define EMAC_RX_SZ 1536

BUFMAP == 4

Definition at line 77 of file ne64config.h.

#define EMAC_TX_SZ 1536

BUFMAP == 4

Definition at line 78 of file ne64config.h.

#define ETYPE_ALL 1

1 = accept Accept all ethertypes. THIS OVERRIDES OTHER SETTINGS

Definition at line 97 of file ne64config.h.

#define ETYPE_ARP 0

1 = accept Address Resolution Protocol (ARP) Ethertype

Definition at line 94 of file ne64config.h.

#define ETYPE_EMW 0

1 = accept Emware Ethertype

Definition at line 92 of file ne64config.h.

#define ETYPE_IEEE 0

1 = accept IEEE802.3 Length Field Ethertype

Definition at line 96 of file ne64config.h.

#define ETYPE_IPV4 0

1 = accept Internet IP version 4 (IPV6) Ethertype

Definition at line 95 of file ne64config.h.

#define ETYPE_IPV6 0

1 = accept Internet IP version (IPV6) Ethertype

Definition at line 93 of file ne64config.h.

#define ETYPE_PET 0

1 = accept Programmable Ethertype, 'etype' parameter is used

Definition at line 91 of file ne64config.h.

#define ETYPE_PRG 0

Enter Value if ETYPE_PET is set for filter target

Definition at line 100 of file ne64config.h.

#define FULL10 1

Configure mode that the device should advertise in auto negotiation (advertise=1)

Definition at line 42 of file ne64config.h.

#define FULL100 1

Configure mode that the device should advertise in auto negotiation (advertise=1)

Definition at line 40 of file ne64config.h.

#define HALF10 1

Configure mode that the device should advertise in auto negotiation (advertise=1)

Definition at line 41 of file ne64config.h.

#define HALF100 1

Configure mode that the device should advertise in auto negotiation (advertise=1)

Definition at line 39 of file ne64config.h.

#define IEEE_PKT 0

Mode for UNH IOL testing (Do not use for gerneral use)

Definition at line 151 of file ne64config.h.

#define LNKLED 1

If USE_SWLED=1, use software to drive an EPHY link LED

Definition at line 129 of file ne64config.h.

#define ON_OFF_AUTONEG 0

Mode for UNH IOL testing (Do not use for gerneral use)

Definition at line 152 of file ne64config.h.

#define PAUSE_TIME 5

Enter value for PAUSE duration parameter in units of slot times (512 bit times)

Definition at line 111 of file ne64config.h.

#define PHY_ADDRESS 0

PHY address used by the MII serial management interface

Definition at line 118 of file ne64config.h.

#define PROM_MODE 0

1 = All frames are received regardless of address.

Definition at line 88 of file ne64config.h.

#define READ_ALL_REGS 0

Mode for UNH IOL testing (Do not use for gerneral use)

Definition at line 154 of file ne64config.h.

#define READ_PHY_ID 0

Mode for UNH IOL testing (Do not use for gerneral use)

Definition at line 153 of file ne64config.h.

#define READ_PTIME 0

define for EtherPause function ptrc variable

Definition at line 113 of file ne64config.h.

#define RX_MAX_FL 1536

Recieve maxiuim frame length

Definition at line 103 of file ne64config.h.

#define RX_POLL_MODE 0

1 = polling; Set to 0 to make RX interrupt driven >> Not implemented into Viola Stack

Definition at line 27 of file ne64config.h.

#define SEND_PAUSE 1

define for EtherPause function ptrc variable

Definition at line 112 of file ne64config.h.

#define SPDLED 1

If USE_SWLED=1, use software to drive an EPHY speed LED

Definition at line 130 of file ne64config.h.

#define USE_EXTBUS 0

1 = External Bus will be used. Change PLL setting to Force bus clock configuration to maximum 16 MHz bus clock Forces Configuration EPHY speed to 10 Mbps maximum

Definition at line 144 of file ne64config.h.

#define USE_SWLED 1

1 = use user software to drive EPHY status indicators on port L 0 = use EPHY hardware LED drive function to drive 5 EPHY status indicators

Definition at line 124 of file ne64config.h.

#define WORD_ACCESS 1

Word Acess Mode. This mode allows word access to buffers instead of byte acess to increase overall system preformance.

Definition at line 22 of file ne64config.h.

#define XFLOWC 0

1 - enable flow control in full dueplex / 0 - disable

Definition at line 110 of file ne64config.h.

#define ZERO_COPY 1

Zero Copy Mode. Setting this mode not only can conserve RAM but can also increase performance. In this mode, no copy of the recieve buffer is created, so the data is processed in the buffer

Definition at line 24 of file ne64config.h.

---

ne64debug.c File Reference

```
#include <stdio.h>
#include "ne64debug.h"
#include "MC9S12NE64.h"
```

---

ne64debug.h File Reference

```
#include "MOTTYPES.h"
```

Defines

- #define _INIT_DEBUG()
- #define _DEBUGT(a)
- #define _DEBUGI(a)
- #define _DEBUGC(a)
- #define _DEBUGNL

Define Documentation

#define _DEBUGC(a)

Definition at line 43 of file ne64debug.h.

#define _DEBUGI(a)

Definition at line 42 of file ne64debug.h.

#define _DEBUGNL

Definition at line 44 of file ne64debug.h.

#define _DEBUGT(a)

Definition at line 41 of file ne64debug.h.

#define _INIT_DEBUG()

Definition at line 40 of file ne64debug.h.

---

ne64driver.c File Reference

```
#include "MOTTYPES.h"
#include "address.h"
#include "ne64config.h"
#include "ne64driver.h"
#include "ne64debug.h"
#include "MC9S12NE64.h"
```

Data Structures

- struct tFRHEAD
- union uMACADUnion
- union uMCHASHUnion

Defines

- #define RAM_START 0x2000
- #define CRC32_POLY 0x04c11db7UL
- #define SET_MCAST_LIST 0x01
- #define SET_ALL_MCAST 0x02

Typedefs

- typedef uMCHASHUnion tMCHASHStr
- typedef uMACADUnion tMACADStr
- typedef tFRHEAD * pFRHEAD

Functions

- UINT16 NE64Receive (void *PktBuffer, UINT16 len, UINT16 flags)
- tREG16 emacFIFOb (RAM_START+EMAC_RX_SZ)
- tREG16 emacFIFOtx (RAM_START+2 *EMAC_RX_SZ)
- void EtherInit (void)
  EtherInit: Start up EPHY and EMAC base on etherinit.h user configuration.

- tU08 MIIwrite (tU08 _mpadr, tU08 _mradr, tU16 _mwdata)
  MIIwrite: write data to PHY function.

- tU08 MIIread (tU08 _mpadr, tU08 _mradr, tU16 *_mrdata)
  MIIread: read data from PHY function.

- void EtherSend (void *databuf, tU16 datalen)
  EtherSend: send one frame.

- void EtherOpen (tU08 miisetup, tU08 bufmap, tU16 maxfl, void *pmacad, tU08 control, tU16 etype, tU08 rxmode, tU08 netctl)
  EtherOpen: prepare the EMAC for normal operation.

- void EtherClose (void)
  EtherClose: switch off the EMAC and clear interrupt flags.

- void EmacDisable (void)
  EmacDisable: switch off the EMAC.

- void EmacEnable (void)

EmacEnable: switch on the EMAC.

- void EmacControl (tU08 netctl)
  EmacControl: Set control bits in EMAC netctl register.

- void EtherIoctl (tU08 flag, void *optionPtr, tU08 optionLen)
  EtherIoctl: Setting of the multicast hash table.

- void EtherGetPhysAddr (void *ethaddr)
  EtherGetPhysAddr: return EMAC current physical address.

- void EtherType (tU08 control, tU16 etype)
  EtherType: set the Ethertye acceptance registers.

- void EtherAbortTx (void)
  EtherAbortTx: abort TX in progress.

- tU16 EtherPause (tU08 ptrc, tU16 ptime)
  EtherOtherTx: EtherPause: send PAUSE frame.

- void EtherOtherTx (tU08 txpar)
  EtherOtherTx: setup of other TX parameters.

- void EtherStartFrameTransmission (tU16 datalen)
  ExternalBusCfg: external bus mode configuration.

- void UseSWLedRun ()
  ExternalBusCfg: external bus mode configuration UseSWLedRun: Turn off EPHY indicator LED.

- interrupt void ephy_isr (void)
  EPHY ISR - Type of EPHY interrupt detemined by MII read of PHY_REG_IR register.

- interrupt void emac_rx_fc_isr (void)
  RX flow control ISR.

- interrupt void emac_b_rx_error_isr (void)
  Babbling Receive Error ISR.

- interrupt void emac_rx_error_isr (void)
  Receive Error ISR.

- interrupt void emac_rx_b_a_o_isr (void)
  RXAOIF — Receive Buffer A Overrun ISR.

- interrupt void emac_rx_b_b_o_isr (void)
  RXAOIF — Receive Buffer B Overrun ISR.

- interrupt void emac_rx_b_a_c_isr (void)
  Valid Frame Reception to Receive Buffer A Complete ISR.

- interrupt void emac_rx_b_b_c_isr (void)
  Valid Frame Reception to Receive Buffer B Complete ISR.

- interrupt void emac_mii_mtc_isr (void)
  Management Transfer Complete ISR - MMCIF — MII Interrupt Flag.

- interrupt void emac_lc_isr (void)
  late collisions ISR - LCIF — Late Collision Interrupt Flag

- interrupt void emac_ec_isr (void)
  excess collisions ISR - ECIF — Excessive Collision Interrupt Flag

- interrupt void emac_f_tx_c_isr (void)
  transmit complete ISR - TXCIF — Frame Transmission Complete Interrupt Flag

Variables

- tREG16 emacFIFOa[EMAC_RX_SZ/2] RAM_START
- tU08 gotlink
- tREG16 * rxa_pointer
- tREG16 * rxb_pointer
- tREG16 * tx_pointer
- tU16 LEDcounter
- tU16 gotxflowc

Define Documentation

#define CRC32_POLY 0x04c11db7UL

Definition at line 655 of file ne64driver.c.

#define RAM_START 0x2000

RAM block starting address

Definition at line 48 of file ne64driver.c.

#define SET_ALL_MCAST 0x02

Definition at line 657 of file ne64driver.c.

#define SET_MCAST_LIST 0x01

Definition at line 656 of file ne64driver.c.

Typedef Documentation

typedef tFRHEAD* pFRHEAD

Definition at line 44 of file ne64driver.c.

typedef union uMACADUnion tMACADStr

typedef union uMCHASHUnion tMCHASHStr

Function Documentation

interrupt void emac_b_rx_error_isr (void)

Babbling Receive Error ISR.

Definition at line 1171 of file ne64driver.c.

interrupt void emac_ec_isr (void)

excess collisions ISR - ECIF — Excessive Collision Interrupt Flag

Definition at line 1292 of file ne64driver.c.

interrupt void emac_f_tx_c_isr (void)

transmit complete ISR - TXCIF — Frame Transmission Complete Interrupt Flag

Definition at line 1306 of file ne64driver.c.

interrupt void emac_lc_isr (void)

late collisions ISR - LCIF — Late Collision Interrupt Flag

Definition at line 1278 of file ne64driver.c.

interrupt void emac_mii_mtc_isr (void)

Management Transfer Complete ISR - MMCIF — MII Interrupt Flag.

Definition at line 1268 of file ne64driver.c.

interrupt void emac_rx_b_a_c_isr (void)

Valid Frame Reception to Receive Buffer A Complete ISR.

Definition at line 1226 of file ne64driver.c.

interrupt void emac_rx_b_a_o_isr (void)

RXAOIF — Receive Buffer A Overrun ISR.

Definition at line 1204 of file ne64driver.c.

interrupt void emac_rx_b_b_c_isr (void)

Valid Frame Reception to Receive Buffer B Complete ISR.

Definition at line 1246 of file ne64driver.c.

interrupt void emac_rx_b_b_o_isr (void)

RXAOIF — Receive Buffer B Overrun ISR.

Definition at line 1215 of file ne64driver.c.

interrupt void emac_rx_error_isr (void)

Receive Error ISR.

Definition at line 1193 of file ne64driver.c.

interrupt void emac_rx_fc_isr (void)

RX flow control ISR.

Definition at line 1160 of file ne64driver.c.

void EmacControl (tU08 *netctl*)

EmacControl: Set control bits in EMAC netctl register.

Set control bits in EMAC netctl register

Parameters:

NONE

Returns:

NONE

Definition at line 641 of file ne64driver.c.

void EmacDisable (void)

EmacDisable: switch off the EMAC.

switch off the EMAC

Parameters:

NONE

Returns:

NONE

Definition at line 627 of file ne64driver.c.

void EmacEnable (void)

EmacEnable: switch on the EMAC.

switch on the EMAC

Parameters:

NONE

Returns:

NONE

Definition at line 634 of file ne64driver.c.

tREG16 emacFIFOb (RAM_START+ *EMAC_RX_SZ*)

Emac RX buffer B definition

tREG16 emacFIFOtx (RAM_START+2 * *EMAC_RX_SZ*)

Emac TX buffer definition

interrupt void ephy_isr (void)

EPHY ISR - Type of EPHY interrupt detemined by MII read of PHY_REG_IR register.

Definition at line 909 of file ne64driver.c.

void EtherAbortTx (void)

EtherAbortTx: abort TX in progress.

This function aborts TX in progress

Parameters:

NONE

Returns:

NONE

Todo:

This function need more testing

Bug:

Definition at line 779 of file ne64driver.c.

void EtherClose (void)

EtherClose: switch off the EMAC and clear interrupt flags.

This function switches off the EMAC and clear interrupt flags

Parameters:

NONE

Returns:

NONE

Definition at line 617 of file ne64driver.c.

void EtherGetPhysAddr (void * *ethaddr*)

EtherGetPhysAddr: return EMAC current physical address.

return EMAC current physical address

Parameters:

ethaddr - pointer to place (6 bytes) where the physical address will be stored to No return value

See also:

ne64api.h

Todo:

Bug:

Definition at line 737 of file ne64driver.c.

void EtherInit (void)

EtherInit: Start up EPHY and EMAC base on etherinit.h user configuration.

This function initializes the NE64 EMAC and EPHY and sets speed and duplex based on the users configuation in the "etherinit.h" file

Parameters:

NONE

Returns:

NONE

See also:

ne64config.c

Todo:

- Validate Pause Resolution after lost link and re-autonegation

- Validate Duplex Resolution after lost link and re-autonegation
- Provide workaround for auto-negoitiation link issue

Bug:

- Dummy MII read required after restart auto-negotiation
- Dummy MII read required after MII write to PHY interrupt registers

Definition at line 87 of file ne64driver.c.

void EtherIoctl (tU08 *flag*, void * *optionPtr*, tU08 *optionLen*)

EtherIoctl: Setting of the multicast hash table.

Setting of the multicast hash table

Parameters:

flag - either MC_ALL or MC_LIST
listPtr - pointer to address list (valid only if flag == MC_LIST)
listLen - number of addresses in list (valid only if flag == MC_LIST)

Returns:

No return value

See also:

ne64api.h

Todo:

This function need more testing

Bug:

Definition at line 659 of file ne64driver.c.

void EtherOpen (tU08 *miisetup*, tU08 *bufmap*, tU16 *maxfl*, void * *pmacad*, tU08 *control*, tU16 *etype*, tU08 *rxmode*, tU08 *netctl*)

EtherOpen: prepare the EMAC for normal operation.

This function initializes the NE64 EMAC

Parameters:

miisetup - mii preamble & clock setup
bufmap - buffer configuration (see tables 3-5&3-6 in EMAC doc.)
maxfl - initial max.frame length for receive
pmacad - pointer to MAC address definition
control - the acceptance mask (same as in EtherType function)
etype - programmable ethertype (16bit value)
rxmode - reception mode settings (see RXCT_X possible values)
netctl - network control setup (see NETCT_X possible values)

Returns:

NONE

See also:

ne64api.h

Todo:

Bug:

Definition at line 549 of file ne64driver.c.

void EtherOtherTx (tU08 *txpar*)

EtherOtherTx: setup of other TX parameters.

This function sets up of other TX parameters

Parameters:

txpar - acceptable values (see TXCT_X possible values)

Returns:

No return value

Todo:

This function need more testing

Bug:

Definition at line 809 of file ne64driver.c.

tU16 EtherPause (tU08 *ptrc*, tU16 *ptime*)

EtherOtherTx: EtherPause: send PAUSE frame.

This function sends PAUSE frame

Parameters:

ptrc - if 1 ptime used for PAUSE time setting, if 0 ptime not used
ptime - value of PAUSE timer

Returns:

returns current value of PAUSE timer when ptrc=0

Todo:

This function need more testing

Bug:

Definition at line 787 of file ne64driver.c.

void EtherSend (void * *databuf*, tU16 *datalen*)

EtherSend: send one frame.

This function sends one frame

Parameters:

databuf - pointer to data which should be sent
datalen - length of the data to be sent

Returns:

No return value

See also:

ne64api.h

Todo:

Bug:

Definition at line 472 of file ne64driver.c.

void EtherStartFrameTransmission (tU16 *datalen*)

ExternalBusCfg: external bus mode configuration.

This function sets up external bus mode and forces the bus clock to 16 Mhz which is the external bus mode maximum. The function needs to be located in main().

Parameters:

None

Returns:

None

Todo:

Bug:

Definition at line 831 of file ne64driver.c.

void EtherType (tU08 *control*, tU16 *etype*)

EtherType: set the Ethertye acceptance registers.

This function set the Ethertye acceptance registers based on the users configuation in the "ne64config.h" file

Parameters:

control - the acceptance mask (see definitons)
etype - programmable ethertype (16bit value)

Returns:

No return value

Warning:

It is recommend not to change the Ethertype when the EMAC is enabled

See also:

ne64api.h

Todo:

Bug:

Definition at line 756 of file ne64driver.c.

tU08 MIIread (tU08 *_mpadr*, tU08 *_mradr*, tU16 * *_mrdata*)

MIIread: read data from PHY function.

Read internal EPHY registers through EMAC MII seiral management interface.

Parameters:

_mpadr - address of the device
_mradr - address of the register within the device
_mwdata - pointer to where to store the received contents of PHY register

Returns:

0xff = operation completed OK

- 0x00 = MII busy

Warning:

MII clock must be configured correctly and PHY PLLs must not be disabled for MII seiral management communication

See also:

ne64api.h

Todo:

Make this function repeat until it is sucessful

Bug:

Definition at line 375 of file ne64driver.c.

tU08 MIIwrite (tU08 *_mpadr*, tU08 *_mradr*, tU16 *_mwdata*)

MIIwrite: write data to PHY function.

Write internal EPHY registers through EMAC MII seiral management interface.

Parameters:

_mpadr - address of the device
_mradr - address of the register within the device
_mwdata - data to write to the PHY register

Returns:

0xff = operation completed OK

- 0x00 = MII busy

Todo:

Make this function repeat until it is sucessful

Bug:

Before the MII write take affect a dummy read via the MII is required. This function could be made to do the dummy read

Warning:

MII clock must be configured correctly and PHY PLLs must not be disabled for MII seiral management communication

Definition at line 350 of file ne64driver.c.

UINT16 NE64Receive (void * *PktBuffer*, UINT16 *len*, UINT16 *flags*)

Definition at line 154 of file ne64api.c.

void UseSWLedRun (void)

ExternalBusCfg: external bus mode configuration UseSWLedRun: Turn off EPHY indicator LED.

This function turns off EPHY indicator LED when driven by software. The function need to be located in main() and required a global counter called, LEDcounter. Alternatively, this code can be can be placed in a timer function

Parameters:

None

Returns:

None

Definition at line 891 of file ne64driver.c.

Variable Documentation

tU08 gotlink

Global Variable For Determination if link is active (1=active) defined in "main.c"

Definition at line 80 of file ne64driver.c.

tU16 gotxflowc

Global Variable For Determination of Flow Control Packets are sent in Full Duplex defined in "main.c"

Definition at line 76 of file ne64driver.c.

tU16 LEDcounter

Definition at line 72 of file ne64driver.c.

tREG16 emacFIFOa [EMAC_RX_SZ/2] RAM_START

Emac RX buffer A definition

Definition at line 51 of file ne64driver.c.

tREG16* rxa_pointer

Definition at line 63 of file ne64driver.c.

tREG16* rxb_pointer

Definition at line 64 of file ne64driver.c.

tREG16* tx_pointer

Definition at line 65 of file ne64driver.c.

---

ne64driver.h File Reference

```
#include "MOTTYPES.h"
#include "ne64config.h"
```

Defines

- #define T_PET 0x80
- #define T_EMW 0x10
- #define T_IPV6 0x08
- #define T_ARP 0x04
- #define T_IPV4 0x02
- #define T_IEEE 0x01
- #define T_ALL 0x00
- #define MC_ALL 0
- #define MC_LIST 1
- #define MII_NO_PREAM 0x10
- #define MII_C20 0x04
- #define MII_C25 0x05
- #define MII_C33 0x07
- #define MII_C40 0x08
- #define MII_C50 0x0a
- #define NETCT_ESWAI 0x10
- #define NETCT_EXTPHY 0x08
- #define NETCT_MLB 0x04
- #define NETCT_FDX 0x02
- #define RXCT_RFCE 0x10
- #define RXCT_PROM 0x04
- #define RXCT_CONMC 0x02
- #define RXCT_BCREJ 0x01
- #define TXCT_PTRC 0x10
- #define TXCT_SSB 0x08
- #define MII_MDCSEL(x) x/5000000
- #define MII_WRITE 0x01
- #define MII_READ 0x02
- #define TCMD_START 0x01
- #define TCMD_PAUSE 0x02
- #define TCMD_ABORT 0x03
- #define PHY_REG_CR 0x00
- #define PHY_REG_SR 0x01
- #define PHY_REG_ID1 0x02
- #define PHY_REG_ID2 0x03
- #define PHY_REG_ANAR 0x04
- #define PHY_REG_ANLPAR 0x05
- #define PHY_REG_ER 0x06
- #define PHY_REG_NPTR 0x07
- #define PHY_REG_IR 0x10
- #define PHY_REG_PSR 0x11
- #define PHY_REG_PCR 0x12
- #define PHY_REG_10BTBC 0x13
- #define PHY_REG_100BXBC 0x14
- #define PHY_REG_ADDR 0x15
- #define PHY_REG_DSPRC 0x17
- #define PHY_REG_DSPRR1 0x18
- #define PHY_REG_DSPRR2 0x19

- #define PHY_REG_DSPRR3 0x1A
- #define PHY_REG_DSPWR1 0x1B
- #define PHY_REG_DSPWR2 0x1C
- #define PHY_REG_DSPWR3 0x1D
- #define PHY_R0_RESET 0x8000
- #define PHY_R0_LB 0x4000
- #define PHY_R0_DR 0x2000
- #define PHY_R0_ANE 0x1000
- #define PHY_R0_PD 0x0800
- #define PHY_R0_ISOLATE 0x0400
- #define PHY_R0_RAN 0x0200
- #define PHY_R0_DPLX 0x0100
- #define PHY_R0_CT 0x0080
- #define PHY_R1_100T4 0x8000
- #define PHY_R1_100F 0x4000
- #define PHY_R1_100H 0x2000
- #define PHY_R1_10F 0x1000
- #define PHY_R1_10H 0x0800
- #define PHY_R1_SUP 0x0040
- #define PHY_R1_ANC 0x0020
- #define PHY_R1_RF 0x0010
- #define PHY_R1_ANA 0x0008
- #define PHY_R1_LS 0x0004
- #define PHY_R1_JD 0x0002
- #define PHY_R1_EC 0x0001
- #define PHY_R4_NP 0x8000
- #define PHY_R4_RF 0x2000
- #define PHY_R4_FC 0x0400
- #define PHY_R4_100F 0x0100
- #define PHY_R4_100H 0x0080
- #define PHY_R4_10F 0x0040
- #define PHY_R4_10H 0x0020
- #define PHY_R5_FCTL 0x0400
- #define PHY_R16_ACKIE 0x4000
- #define PHY_R16_PRIE 0x2000
- #define PHY_R16_LCIE 0x1000
- #define PHY_R16_ANIE 0x0800
- #define PHY_R16_PDFIE 0x0400
- #define PHY_R16_RFIE 0x0200
- #define PHY_R16_JABIE 0x0100
- #define PHY_R16_ACKR 0x0040
- #define PHY_R16_PGR 0x0020
- #define PHY_R16_LKC 0x0010
- #define PHY_R16_ANC 0x0008
- #define PHY_R16_PDF 0x0004
- #define PHY_R16_RMTF 0x0002
- #define PHY_R16_JABI 0x0001
- #define PHY_R17_LNK 0x4000
- #define PHY_R17_DPM 0x2000
- #define PHY_R17_SPD 0x1000
- #define PHY_R17_ANNC 0x0400
- #define PHY_R17_PRCVD 0x0200
- #define PHY_R17_ANCM 0x0100

- #define PHY_R17_PLR 0x0020
- #define BUFA_FULL 99
- #define BUFB_FULL 66

Functions

- void EtherInit (void)
  EtherInit: Start up EPHY and EMAC base on etherinit.h user configuration.

- tU08 MIIwrite (tU08 _mpadr, tU08 _mradr, tU16 _mwdata)
  MIIwrite: write data to PHY function.

- tU08 MIIread (tU08 _mpadr, tU08 _mradr, tU16 *_mrdata)
  MIIread: read data from PHY function.

- void EtherType (tU08 control, tU16 etype)
  EtherType: set the Ethertye acceptance registers.

- void EtherIoctl (tU08 flag, void *optionPtr, tU08 optionLen)
  EtherIoctl: Setting of the multicast hash table.

- void EtherGetPhysAddr (void *ethaddr)
  EtherGetPhysAddr: return EMAC current physical address.

- void EtherSend (void *databuf, tU16 datalen)
  EtherSend: send one frame.

- void ProcessPacket ()
  EtherReceive: read received frame.

- void EtherOpen (tU08 miisetup, tU08 bufmap, tU16 maxfl, void *pmacad, tU08 control, tU16 etype, tU08 rxmode, tU08 netctl)
  EtherOpen: prepare the EMAC for normal operation.

- void EtherClose (void)
  EtherClose: switch off the EMAC and clear interrupt flags.

- void EmacDisable (void)
  EmacDisable: switch off the EMAC.

- void EmacEnable (void)
  EmacEnable: switch on the EMAC.

- void EmacControl (tU08 netctl)
  EmacControl: Set control bits in EMAC netctl register.

- void EtherAbortTx (void)
  EtherAbortTx: abort TX in progress.

- tU16 EtherPause (tU08 ptrc, tU16 ptime)
  EtherOtherTx: EtherPause: send PAUSE frame.

- void EtherOtherTx (tU08 txpar)
  EtherOtherTx: setup of other TX parameters.

- void EtherStartFrameTransmission (tU16 datalen)
  ExternalBusCfg: external bus mode configuration.

- void UseSWLedRun (void)
  ExternalBusCfg: external bus mode configuration UseSWLedRun: Turn off EPHY indicator LED.

Define Documentation

#define BUFA_FULL 99

BUFA_FULL constant

Definition at line 467 of file ne64driver.h.

#define BUFB_FULL 66

BUFB_FULL constant

Definition at line 468 of file ne64driver.h.

#define MC_ALL 0

set hash to accept all multicast frames

Definition at line 124 of file ne64driver.h.

#define MC_LIST 1

set hash to accept list of addresses

Definition at line 125 of file ne64driver.h.

#define MII_C20 0x04

20Mhz IP Bus 2.5MHz MDC clock

Definition at line 203 of file ne64driver.h.

#define MII_C25 0x05

25Mhz IP Bus

Definition at line 204 of file ne64driver.h.

#define MII_C33 0x07

33Mhz IP Bus

Definition at line 205 of file ne64driver.h.

#define MII_C40 0x08

40Mhz IP Bus

Definition at line 206 of file ne64driver.h.

#define MII_C50 0x0a

50Mhz IP Bus

Definition at line 207 of file ne64driver.h.

#define MII_MDCSEL(x) x/5000000

Definition at line 362 of file ne64driver.h.

#define MII_NO_PREAM 0x10

No preamble

Definition at line 202 of file ne64driver.h.

#define MII_READ 0x02

Definition at line 365 of file ne64driver.h.

#define MII_WRITE 0x01

Definition at line 364 of file ne64driver.h.

#define NETCT_ESWAI 0x10

EMAC disabled during WAIT

Definition at line 210 of file ne64driver.h.

#define NETCT_EXTPHY 0x08

external PHY mode

Definition at line 211 of file ne64driver.h.

#define NETCT_FDX 0x02

full duplex mode

Definition at line 213 of file ne64driver.h.

#define NETCT_MLB 0x04

MAC loopback mode

Definition at line 212 of file ne64driver.h.

#define PHY_R0_ANE 0x1000

Definition at line 400 of file ne64driver.h.

#define PHY_R0_CT 0x0080

Definition at line 405 of file ne64driver.h.

#define PHY_R0_DPLX 0x0100

Definition at line 404 of file ne64driver.h.

#define PHY_R0_DR 0x2000

Definition at line 399 of file ne64driver.h.

#define PHY_R0_ISOLATE 0x0400

Definition at line 402 of file ne64driver.h.

#define PHY_R0_LB 0x4000

Definition at line 398 of file ne64driver.h.

#define PHY_R0_PD 0x0800

Definition at line 401 of file ne64driver.h.

#define PHY_R0_RAN 0x0200

Definition at line 403 of file ne64driver.h.

#define PHY_R0_RESET 0x8000

Definition at line 397 of file ne64driver.h.

#define PHY_R16_ACKIE 0x4000

Definition at line 441 of file ne64driver.h.

#define PHY_R16_ACKR 0x0040

Definition at line 449 of file ne64driver.h.

#define PHY_R16_ANC 0x0008

Definition at line 452 of file ne64driver.h.

#define PHY_R16_ANIE 0x0800

Definition at line 444 of file ne64driver.h.

#define PHY_R16_JABI 0x0001

Definition at line 455 of file ne64driver.h.

#define PHY_R16_JABIE 0x0100

Definition at line 447 of file ne64driver.h.

#define PHY_R16_LCIE 0x1000

Definition at line 443 of file ne64driver.h.

#define PHY_R16_LKC 0x0010

Definition at line 451 of file ne64driver.h.

#define PHY_R16_PDF 0x0004

Definition at line 453 of file ne64driver.h.

#define PHY_R16_PDFIE 0x0400

Definition at line 445 of file ne64driver.h.

#define PHY_R16_PGR 0x0020

Definition at line 450 of file ne64driver.h.

#define PHY_R16_PRIE 0x2000

Definition at line 442 of file ne64driver.h.

#define PHY_R16_RFIE 0x0200

Definition at line 446 of file ne64driver.h.

#define PHY_R16_RMTF 0x0002

Definition at line 454 of file ne64driver.h.

#define PHY_R17_ANCM 0x0100

Definition at line 463 of file ne64driver.h.

#define PHY_R17_ANNC 0x0400

Definition at line 461 of file ne64driver.h.

#define PHY_R17_DPM 0x2000

Definition at line 459 of file ne64driver.h.

#define PHY_R17_LNK 0x4000

Definition at line 458 of file ne64driver.h.

#define PHY_R17_PLR 0x0020

Definition at line 464 of file ne64driver.h.

#define PHY_R17_PRCVD 0x0200

Definition at line 462 of file ne64driver.h.

#define PHY_R17_SPD 0x1000

Definition at line 460 of file ne64driver.h.

#define PHY_R1_100F 0x4000

Definition at line 409 of file ne64driver.h.

#define PHY_R1_100H 0x2000

Definition at line 410 of file ne64driver.h.

#define PHY_R1_100T4 0x8000

Definition at line 408 of file ne64driver.h.

#define PHY_R1_10F 0x1000

Definition at line 411 of file ne64driver.h.

#define PHY_R1_10H 0x0800

Definition at line 412 of file ne64driver.h.

#define PHY_R1_ANA 0x0008

Definition at line 416 of file ne64driver.h.

#define PHY_R1_ANC 0x0020

Definition at line 414 of file ne64driver.h.

#define PHY_R1_EC 0x0001

Definition at line 419 of file ne64driver.h.

#define PHY_R1_JD 0x0002

Definition at line 418 of file ne64driver.h.

#define PHY_R1_LS 0x0004

Definition at line 417 of file ne64driver.h.

#define PHY_R1_RF 0x0010

Definition at line 415 of file ne64driver.h.

#define PHY_R1_SUP 0x0040

Definition at line 413 of file ne64driver.h.

#define PHY_R4_100F 0x0100

Definition at line 430 of file ne64driver.h.

#define PHY_R4_100H 0x0080

Definition at line 431 of file ne64driver.h.

#define PHY_R4_10F 0x0040

Definition at line 432 of file ne64driver.h.

#define PHY_R4_10H 0x0020

Definition at line 433 of file ne64driver.h.

#define PHY_R4_FC 0x0400

Definition at line 429 of file ne64driver.h.

#define PHY_R4_NP 0x8000

Definition at line 427 of file ne64driver.h.

#define PHY_R4_RF 0x2000

Definition at line 428 of file ne64driver.h.

#define PHY_R5_FCTL 0x0400

Definition at line 438 of file ne64driver.h.

#define PHY_REG_100BXBC 0x14

Definition at line 385 of file ne64driver.h.

#define PHY_REG_10BTBC 0x13

Definition at line 384 of file ne64driver.h.

#define PHY_REG_ADDR 0x15

Definition at line 386 of file ne64driver.h.

#define PHY_REG_ANAR 0x04

Definition at line 377 of file ne64driver.h.

#define PHY_REG_ANLPAR 0x05

Definition at line 378 of file ne64driver.h.

#define PHY_REG_CR 0x00

Definition at line 373 of file ne64driver.h.

#define PHY_REG_DSPRC 0x17

Definition at line 387 of file ne64driver.h.

#define PHY_REG_DSPRR1 0x18

Definition at line 388 of file ne64driver.h.

#define PHY_REG_DSPRR2 0x19

Definition at line 389 of file ne64driver.h.

#define PHY_REG_DSPRR3 0x1A

Definition at line 390 of file ne64driver.h.

#define PHY_REG_DSPWR1 0x1B

Definition at line 391 of file ne64driver.h.

#define PHY_REG_DSPWR2 0x1C

Definition at line 392 of file ne64driver.h.

#define PHY_REG_DSPWR3 0x1D

Definition at line 393 of file ne64driver.h.

#define PHY_REG_ER 0x06

Definition at line 379 of file ne64driver.h.

#define PHY_REG_ID1 0x02

Definition at line 375 of file ne64driver.h.

#define PHY_REG_ID2 0x03

Definition at line 376 of file ne64driver.h.

#define PHY_REG_IR 0x10

Definition at line 381 of file ne64driver.h.

#define PHY_REG_NPTR 0x07

Definition at line 380 of file ne64driver.h.

#define PHY_REG_PCR 0x12

Definition at line 383 of file ne64driver.h.

#define PHY_REG_PSR 0x11

Definition at line 382 of file ne64driver.h.

#define PHY_REG_SR 0x01

Definition at line 374 of file ne64driver.h.

#define RXCT_BCREJ 0x01

all broadcast frames will be rejected

Definition at line 219 of file ne64driver.h.

#define RXCT_CONMC 0x02

multicast hash table used for incoming frames chk

Definition at line 218 of file ne64driver.h.

#define RXCT_PROM 0x04

promiscuous mode

Definition at line 217 of file ne64driver.h.

#define RXCT_RFCE 0x10

PAUSE frame supported

Definition at line 216 of file ne64driver.h.

#define T_ALL 0x00

Accept all ethertypes

Definition at line 107 of file ne64driver.h.

#define T_ARP 0x04

Address Resolution Protocol (ARP) Ethertype

Definition at line 104 of file ne64driver.h.

#define T_EMW 0x10

Emware Ethertype

Definition at line 102 of file ne64driver.h.

#define T_IEEE 0x01

IEEE802.3 Length Field Ethertype

Definition at line 106 of file ne64driver.h.

#define T_IPV4 0x02

Internet IP version 4 (IPV6) Ethertype

Definition at line 105 of file ne64driver.h.

#define T_IPV6 0x08

Internet IP version (IPV6) Ethertype

Definition at line 103 of file ne64driver.h.

#define T_PET 0x80

Programmable Ethertype, 'etype' parameter is used

Definition at line 101 of file ne64driver.h.

#define TCMD_ABORT 0x03

Definition at line 369 of file ne64driver.h.

#define TCMD_PAUSE 0x02

Definition at line 368 of file ne64driver.h.

#define TCMD_START 0x01

Definition at line 367 of file ne64driver.h.

#define TXCT_PTRC 0x10

Definition at line 306 of file ne64driver.h.

#define TXCT_SSB 0x08

Definition at line 307 of file ne64driver.h.

Function Documentation

void EmacControl (tU08 *netctl*)

EmacControl: Set control bits in EMAC netctl register.

Set control bits in EMAC netctl register

Parameters:

NONE

Returns:

NONE

Definition at line 641 of file ne64driver.c.

void EmacDisable (void)

EmacDisable: switch off the EMAC.

switch off the EMAC

Parameters:

NONE

Returns:

NONE

Definition at line 627 of file ne64driver.c.

void EmacEnable (void)

EmacEnable: switch on the EMAC.

switch on the EMAC

Parameters:

NONE

Returns:

NONE

Definition at line 634 of file ne64driver.c.

void EtherAbortTx (void)

EtherAbortTx: abort TX in progress.

This function aborts TX in progress

Parameters:

NONE

Returns:

NONE

Todo:

This function need more testing

Bug:

Definition at line 779 of file ne64driver.c.

void EtherClose (void)

EtherClose: switch off the EMAC and clear interrupt flags.

This function switches off the EMAC and clear interrupt flags

Parameters:

NONE

Returns:

NONE

Definition at line 617 of file ne64driver.c.

void EtherGetPhysAddr (void * *ethaddr*)

EtherGetPhysAddr: return EMAC current physical address.

return EMAC current physical address

Parameters:

ethaddr - pointer to place (6 bytes) where the physical address will be stored to No return value

See also:

ne64api.h

Todo:

Bug:

Definition at line 737 of file ne64driver.c.

void EtherInit (void)

EtherInit: Start up EPHY and EMAC base on etherinit.h user configuration.

This function initializes the NE64 EMAC and EPHY and sets speed and duplex based on the users configuation in the "etherinit.h" file

Parameters:

NONE

Returns:

NONE

See also:

ne64config.c

Todo:

- Validate Pause Resolution after lost link and re-autonegation
- Validate Duplex Resolution after lost link and re-autonegation
- Provide workaround for auto-negoitiation link issue

Bug:

- Dummy MII read required after restart auto-negotiation
- Dummy MII read required after MII write to PHY interrupt registers

Definition at line 87 of file ne64driver.c.

void EtherIoctl (tU08 *flag*, void * *optionPtr*, tU08 *optionLen*)

EtherIoctl: Setting of the multicast hash table.

Setting of the multicast hash table

Parameters:

flag - either MC_ALL or MC_LIST
listPtr - pointer to address list (valid only if flag == MC_LIST)
listLen - number of addresses in list (valid only if flag == MC_LIST)

Returns:

No return value

See also:

ne64api.h

Todo:

This function need more testing

Bug:

Definition at line 659 of file ne64driver.c.

void EtherOpen (tU08 *miisetup*, tU08 *bufmap*, tU16 *maxfl*, void * *pmacad*, tU08 *control*, tU16 *etype*, tU08 *rxmode*, tU08 *netctl*)

EtherOpen: prepare the EMAC for normal operation.

This function initializes the NE64 EMAC

Parameters:

miisetup - mii preamble & clock setup
bufmap - buffer configuration (see tables 3-5&3-6 in EMAC doc.)
maxfl - initial max.frame length for receive
pmacad - pointer to MAC address definition
control - the acceptance mask (same as in EtherType function)
etype - programmable ethertype (16bit value)
rxmode - reception mode settings (see RXCT_X possible values)
netctl - network control setup (see NETCT_X possible values)

Returns:

NONE

See also:

ne64api.h

Todo:

Bug:

Definition at line 549 of file ne64driver.c.

void EtherOtherTx (tU08 *txpar*)

EtherOtherTx: setup of other TX parameters.

This function sets up of other TX parameters

Parameters:

txpar - acceptable values (see TXCT_X possible values)

Returns:

No return value

Todo:

This function need more testing

Bug:

Definition at line 809 of file ne64driver.c.

tU16 EtherPause (tU08 *ptrc*, tU16 *ptime*)

EtherOtherTx: EtherPause: send PAUSE frame.

This function sends PAUSE frame

Parameters:

ptrc - if 1 ptime used for PAUSE time setting, if 0 ptime not used
ptime - value of PAUSE timer

Returns:

returns current value of PAUSE timer when ptrc=0

Todo:

This function need more testing

Bug:

Definition at line 787 of file ne64driver.c.

void EtherSend (void * *databuf*, tU16 *datalen*)

EtherSend: send one frame.

This function sends one frame

Parameters:

databuf - pointer to data which should be sent
datalen - length of the data to be sent

Returns:

No return value

See also:

ne64api.h

Todo:

Bug:

Definition at line 472 of file ne64driver.c.

void EtherStartFrameTransmission (tU16 *datalen*)

ExternalBusCfg: external bus mode configuration.

This function sets up external bus mode and forces the bus clock to 16 Mhz which is the external bus mode maximum. The function needs to be located in main().

Parameters:

None

Returns:

None

Todo:

Bug:

Definition at line 831 of file ne64driver.c.

void EtherType (tU08 *control*, tU16 *etype*)

EtherType: set the Ethertye acceptance registers.

This function set the Ethertye acceptance registers based on the users configuation in the "ne64config.h" file

Parameters:

control - the acceptance mask (see definitons)
etype - programmable ethertype (16bit value)

Returns:

No return value

Warning:

It is recommend not to change the Ethertype when the EMAC is enabled

See also:

ne64api.h

Todo:

Bug:

Definition at line 756 of file ne64driver.c.

tU08 MIIread (tU08 *_mpadr*, tU08 *_mradr*, tU16 * *_mrdata*)

MIIread: read data from PHY function.

Read internal EPHY registers through EMAC MII seiral management interface.

Parameters:

_mpadr - address of the device
_mradr - address of the register within the device
_mwdata - pointer to where to store the received contents of PHY register

Returns:

0xff = operation completed OK

- 0x00 = MII busy

Warning:

MII clock must be configured correctly and PHY PLLs must not be disabled for MII seiral management communication

See also:

ne64api.h

Todo:

Make this function repeat until it is sucessful

Bug:

Definition at line 375 of file ne64driver.c.

tU08 MIIwrite (tU08 *_mpadr*, tU08 *_mradr*, tU16 *_mwdata*)

MIIwrite: write data to PHY function.

Write internal EPHY registers through EMAC MII seiral management interface.

Parameters:

_mpadr - address of the device
_mradr - address of the register within the device
_mwdata - data to write to the PHY register

Returns:

0xff = operation completed OK

- 0x00 = MII busy

Todo:

Make this function repeat until it is sucessful

Bug:

Before the MII write take affect a dummy read via the MII is required. This function could be made to do the dummy read

Warning:

MII clock must be configured correctly and PHY PLLs must not be disabled for MII seiral management communication

Definition at line 350 of file ne64driver.c.

void ProcessPacket ()

EtherReceive: read received frame.

This function reads the received frame

Parameters:

buffer - pointer to place where the physical address will be stored to

Returns:

length of the received data (if 0 - then no data received)

void UseSWLedRun (void)

ExternalBusCfg: external bus mode configuration UseSWLedRun: Turn off EPHY indicator LED.

This function turns off EPHY indicator LED when driven by software. The function need to be located in main() and required a global counter called, LEDcounter. Alternatively, this code can be can be placed in a timer function

Parameters:

None

Returns:

None

Definition at line 891 of file ne64driver.c.

---

os.c File Reference

```
#include "os.h"
```

Functions

- void os_enter_critical_section (void)
- void os_exit_critical_section (void)

Function Documentation

void os_enter_critical_section (void)

Definition at line 21 of file os.c.

void os_exit_critical_section (void)

Definition at line 30 of file os.c.

---

os.h File Reference

Defines

- #define OS_ENTER_CRITICAL() os_enter_critical_section()
- #define OS_EXIT_CRITICAL() os_exit_critical_section()

Functions

- void os_enter_critical_section (void)
- void os_exit_critical_section (void)

Define Documentation

#define OS_ENTER_CRITICAL() os_enter_critical_section()

Definition at line 20 of file os.h.

#define OS_EXIT_CRITICAL() os_exit_critical_section()

Definition at line 21 of file os.h.

Function Documentation

void os_enter_critical_section (void)

Definition at line 21 of file os.c.

void os_exit_critical_section (void)

Definition at line 30 of file os.c.

---

pop3_client.c File Reference

Detailed Description

OpenTCP POP3 client implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
20.08.2002

Bug:

Warning:

Todo:

OpenTCP implementation of POP3 client that uses TCP api. For interface functions declarations see /pop3/pop3_client.h.
Definition in file pop3_client.c.

```
#include "debug.h"
#include "datatypes.h"
#include "globalvariables.h"
#include "system.h"
#include "timers.h"
#include "tcp_ip.h"
#include "pop3_client.h"
```

Functions

- INT8 pop3c_connect (UINT32 ip, UINT16 port)
  Start E-mail reading procedure.

- void pop3c_init (void)
  Initialize POP3 client.

- UINT8 pop3c_getstate (void)
  Get current POP3 client state.

- INT32 pop3c_eventlistener (INT8 cbhandle, UINT8 event, UINT32 par1, UINT32 par2)
- void pop3c_run (void)

- void pop3c_senduser (void)
- void pop3c_sendpassword (void)
- void pop3c_sendstat (void)
- void pop3c_sendlist (UINT16 msgnbr)
- void pop3c_sendtop (UINT16 msgnbr)
- void pop3c_sendretr (UINT16 msgnbr)
- void pop3c_senddele (UINT16 msgnbr)
- void pop3c_sendquit (void)
- INT16 pop3c_parsestat (void)
- INT16 pop3c_parselist (void)
- void pop3c_changestate (UINT8 nstate)

Variables

- UINT8 pop3c_init_done = 0
- pop3c_struct pop3_client
  Holds information needed by the POP3 client for successful operation.

Function Documentation

void pop3c_changestate (UINT8 *nstate*)

Definition at line 1377 of file pop3_client.c.

INT8 pop3c_connect (UINT32 *ip*, UINT16 *port*)

Start E-mail reading procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

ip IP address of POP3 server from which to read the e-mails
port Port on the server

Returns:

- -1 - Error
- >0 - Connection procedure started (OK)

This function is called by user when she wants to start E-mail reading procedure. The function is responsible of establishing connection to POP3 server. After connection is established the POP3 client engine starts to make callbacks to user functions in order to get username information, data etc.

Definition at line 114 of file pop3_client.c.

INT32 pop3c_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *par1*, UINT32 *par2*)

Definition at line 220 of file pop3_client.c.

UINT8 pop3c_getstate (void)

Get current POP3 client state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.10.2002

Returns:

Current POP3 client state

Invoke this function to get current state of the POP3 client

Definition at line 194 of file pop3_client.c.

void pop3c_init (void)

Initialize POP3 client.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This function should be called once when system starts. Make sure that system services e.g. timers, TCP are initialized before initializing applications!

Definition at line 148 of file pop3_client.c.

INT16 pop3c_parselist (void)

Definition at line 1317 of file pop3_client.c.

INT16 pop3c_parsestat (void)

Definition at line 1276 of file pop3_client.c.

void pop3c_run (void)

Definition at line 831 of file pop3_client.c.

void pop3c_senddele (UINT16 *msgnbr*)

Definition at line 1222 of file pop3_client.c.

void pop3c_sendlist (UINT16 *msgnbr*)

Definition at line 1122 of file pop3_client.c.

void pop3c_sendpassword (void)

Definition at line 1066 of file pop3_client.c.

void pop3c_sendquit (void)

Definition at line 1256 of file pop3_client.c.

void pop3c_sendretr (UINT16 *msgnbr*)

Definition at line 1189 of file pop3_client.c.

void pop3c_sendstat (void)

Definition at line 1102 of file pop3_client.c.

void pop3c_sendtop (UINT16 *msgnbr*)

Definition at line 1155 of file pop3_client.c.

void pop3c_senduser (void)

Definition at line 1030 of file pop3_client.c.

Variable Documentation

struct pop3c_struct pop3_client

Holds information needed by the POP3 client for successful operation.

All of the information that the POP3 client is using for operation are stored here. See pop3c_struct definition for more information about the structure fields.

Definition at line 85 of file pop3_client.c.

UINT8 pop3c_init_done = 0

Defines whether pop3c_init has already been invoked or not

Definition at line 77 of file pop3_client.c.

---

pop3_client.h File Reference

Detailed Description

OpenTCP POP3 client interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

20.8.2002

OpenTCP POP3 function declarations, constants, etc.
Definition in file pop3_client.h.

```
#include "datatypes.h"
```

Data Structures

- struct pop3c_struct
  POP3 client structure.

Defines

- #define POP3C_SENDERMAXLEN 30
- #define POP3C_SUBJECTMAXLEN 30
- #define POP3C_TOUT 20
- #define POP3C_UNINITIALIZED 1
- #define POP3C_CLOSED 2
- #define POP3C_OPEN_REQUESTED 3
- #define POP3C_CONNECTIONOPEN_SENT 4
- #define POP3C_CONNECTION_OPENED 5
- #define POP3C_SERVER_READY 6
- #define POP3C_USERNAME_SENT 7
- #define POP3C_USERNAME_ACKED 8
- #define POP3C_PASSWORD_SENT 9
- #define POP3C_PASSWORD_ACKED 10
- #define POP3C_STAT_SENT 11
- #define POP3C_STAT_GET 12
- #define POP3C_LIST_SENT 13
- #define POP3C_LIST_GET 14
- #define POP3C_TOP0_SENT 15
- #define POP3C_RECEIVING_HEADER 16
- #define POP3C_RECEIVING_HDR_FROM 17
- #define POP3C_RECEIVING_HDR_SUBJ 18
- #define POP3C_TOP0_GET 19
- #define POP3C_RETR_SENT 20
- #define POP3C_RECEIVING_MSG_HEADER 21
- #define POP3C_RECEIVING_MSG 22
- #define POP3C_MESSAGE_RECEIVED 23
- #define POP3C_DELE_SENT 24
- #define POP3C_DELE_ACKED 25
- #define POP3C_QUIT_SENT 26
- #define POP3C_QUIT_ACKED 27
- #define POP3C_OK '+'

Functions

- INT8 pop3c_connect (UINT32, UINT16)
  Start E-mail reading procedure.

- void pop3c_init (void)
  Initialize POP3 client.

- UINT8 pop3c_getstate (void)
  Get current POP3 client state.

- INT32 pop3c_eventlistener (INT8, UINT8, UINT32, UINT32)
- void pop3c_run (void)
- void pop3c_senduser (void)

- void pop3c_sendpassword (void)
- void pop3c_sendstat (void)
- void pop3c_sendlist (UINT16)
- void pop3c_sendtop (UINT16)
- void pop3c_sendretr (UINT16)
- void pop3c_senddele (UINT16)
- void pop3c_sendquit (void)
- void pop3c_changestate (UINT8)
- INT16 pop3c_parsestat (void)
- INT16 pop3c_parselist (void)
- void pop3c_error (void)
  POP3 client error handler.

- void pop3c_data (UINT8)
  Receives E-mail data.

- void pop3c_allok (void)
  Indicates succesfull reading of E-mails.

- void pop3c_messages (UINT16)
  Invoked to inform user app about the number of new e-mails.

- INT16 pop3c_msgoffer (UINT16, UINT32, UINT8 *, UINT8 *)
  Offers e-mail message to the user app.

- INT8 pop3c_getusername (UINT8 *)
  Get user name that is to be used for loging to the server.

- INT8 pop3c_getpassword (UINT8 *)
  Get password that is to be used for loging to the server.

Define Documentation

#define POP3C_CLOSED 2

POP3 state: TCP connection closed

Definition at line 108 of file pop3_client.h.

#define POP3C_CONNECTION_OPENED 5

POP3 state: TCP Connection opened

Definition at line 111 of file pop3_client.h.

#define POP3C_CONNECTIONOPEN_SENT 4

POP3 state: TCP connection request sent

Definition at line 110 of file pop3_client.h.

#define POP3C_DELE_ACKED 25

POP3 state: Server has replied dele +OK

Definition at line 131 of file pop3_client.h.

#define POP3C_DELE_SENT 24

POP3 state: DELE sent by us

Definition at line 130 of file pop3_client.h.

#define POP3C_LIST_GET 14

POP3 state: Server has repld. with the len of msg

Definition at line 120 of file pop3_client.h.

#define POP3C_LIST_SENT 13

POP3 state: LIST sent by us

Definition at line 119 of file pop3_client.h.

#define POP3C_MESSAGE_RECEIVED 23

POP3 state: Received the message

Definition at line 129 of file pop3_client.h.

#define POP3C_OK '+'

Definition at line 135 of file pop3_client.h.

#define POP3C_OPEN_REQUESTED 3

POP3 state: User has requested mail read

Definition at line 109 of file pop3_client.h.

#define POP3C_PASSWORD_ACKED 10

POP3 state: Server answered password +OK

Definition at line 116 of file pop3_client.h.

#define POP3C_PASSWORD_SENT 9

POP3 state: PASS sent by us

Definition at line 115 of file pop3_client.h.

#define POP3C_QUIT_ACKED 27

POP3 state: Server has replied quit +OK

Definition at line 133 of file pop3_client.h.

#define POP3C_QUIT_SENT 26

POP3 state: QUIT sent by us

Definition at line 132 of file pop3_client.h.

#define POP3C_RECEIVING_HDR_FROM 17

POP3 state: We are parsing 'from:'

Definition at line 123 of file pop3_client.h.

#define POP3C_RECEIVING_HDR_SUBJ 18

POP3 state: We are parsing 'subject:'

Definition at line 124 of file pop3_client.h.

#define POP3C_RECEIVING_HEADER 16

POP3 state: We are receiving header

Definition at line 122 of file pop3_client.h.

#define POP3C_RECEIVING_MSG 22

POP3 state: Receiving the message

Definition at line 128 of file pop3_client.h.

#define POP3C_RECEIVING_MSG_HEADER 21

POP3 state: We are reading the message header

Definition at line 127 of file pop3_client.h.

#define POP3C_RETR_SENT 20

POP3 state: RETR sent by us

Definition at line 126 of file pop3_client.h.

#define POP3C_SENDERMAXLEN 30

Maximum length for senders' e-mail address including "

Definition at line 69 of file pop3_client.h.

#define POP3C_SERVER_READY 6

POP3 state: POP3 server has indicated +OK

Definition at line 112 of file pop3_client.h.

#define POP3C_STAT_GET 12

POP3 state: Server has answered how many messages

Definition at line 118 of file pop3_client.h.

#define POP3C_STAT_SENT 11

POP3 state: STAT sent by us

Definition at line 117 of file pop3_client.h.

#define POP3C_SUBJECTMAXLEN 30

Maximum length of the subject field including "

Definition at line 72 of file pop3_client.h.

#define POP3C_TOP0_GET 19

POP3 state: Server has replied with header

Definition at line 125 of file pop3_client.h.

#define POP3C_TOP0_SENT 15

POP3 state: TOP x 0 sent by us

Definition at line 121 of file pop3_client.h.

#define POP3C_TOUT 20

POP3 client timeout in secs

Definition at line 76 of file pop3_client.h.

#define POP3C_UNINITIALIZED 1

POP3 state: Not initialized yet

Definition at line 107 of file pop3_client.h.

#define POP3C_USERNAME_ACKED 8

POP3 state: Server answered username +OK

Definition at line 114 of file pop3_client.h.

#define POP3C_USERNAME_SENT 7

POP3 state: USER sent by us

Definition at line 113 of file pop3_client.h.

Function Documentation

void pop3c_allok (void)

Indicates succesfull reading of E-mails.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

14.10.2002

This callback function is called by POP3 Client to indicate succesfull reading of E-mails

Definition at line 173 of file pop3c_callbacks.c.

void pop3c_changestate (UINT8)

Definition at line 1377 of file pop3_client.c.

INT8 pop3c_connect (UINT32 *ip*, UINT16 *port*)

Start E-mail reading procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

ip IP address of POP3 server from which to read the e-mails
port Port on the server

Returns:

- -1 - Error
- >0 - Connection procedure started (OK)

This function is called by user when she wants to start E-mail reading procedure. The function is responsible of establishing connection to POP3 server. After connection is established the POP3 client engine starts to make callbacks to user functions in order to get username information, data etc.

Definition at line 114 of file pop3_client.c.

void pop3c_data (UINT8 *data*)

Receives E-mail data.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

14.10.2002

Parameters:

data E-mail data as received by POP3 client

This callback function is called by POP3 Client in order to give data to application

Definition at line 187 of file pop3c_callbacks.c.

void pop3c_error (void)

POP3 client error handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

This callback function is called by POP3 Client when there happens error of some kind (timeout, losing of connection etc.).

Definition at line 82 of file pop3c_callbacks.c.

INT32 pop3c_eventlistener (INT8, UINT8, UINT32, UINT32)

Definition at line 220 of file pop3_client.c.

INT8 pop3c_getpassword (UINT8 * *dbuf*)

Get password that is to be used for loging to the server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

dbuf Pointer to buffer to which the password will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by POP3 Client when it wants to know the password of us. The user is responsible of storing that name to destbuf without NULL termination (") and returning number of bytes on the password.

Definition at line 159 of file pop3c_callbacks.c.

UINT8 pop3c_getstate (void)

Get current POP3 client state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

10.10.2002

Returns:

Current POP3 client state

Invoke this function to get current state of the POP3 client

Definition at line 194 of file pop3_client.c.

INT8 pop3c_getusername (UINT8 * *dbuf*)

Get user name that is to be used for loging to the server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

dbuf Pointer to buffer to which the username will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by POP3 Client when it wants to know the username of us. The user is responsible of storing that name to destbuf without NULL termination (") and returning number of bytes on that username.

Definition at line 139 of file pop3c_callbacks.c.

void pop3c_init (void)

Initialize POP3 client.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This function should be called once when system starts. Make sure that system services e.g. timers, TCP are initialized before initializing applications!

Definition at line 148 of file pop3_client.c.

void pop3c_messages (UINT16 *msgs*)

Invoked to inform user app about the number of new e-mails.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

msgs Number of new e-mails waiting on the server

This callback function is called by POP3 Client in order to indicate the number of new E-mails on server.

Definition at line 96 of file pop3c_callbacks.c.

INT16 pop3c_msgoffer (UINT16 *index*, UINT32 *msglen*, UINT8 * *from*, UINT8 * *subject*)

Offers e-mail message to the user app.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

index index number of message
msglen Length of message data
from Buffer containing sender (null terminated string)
subject Buffer containing subject (null terminated string)

Returns:

- -2 - Reject the e-mail (delete from server)
- -1 - Skip the e-mail (leave it on server)
- >=0 - Read and delete the mail from server

This callback function is called by POP3 Client in order to offer the e-mail message to user. User can reject this mail, skip this, mail or read it as indicated with return value

Definition at line 117 of file pop3c_callbacks.c.

INT16 pop3c_parselist (void)

Definition at line 1317 of file pop3_client.c.

INT16 pop3c_parsestat (void)

Definition at line 1276 of file pop3_client.c.

void pop3c_run (void)

Definition at line 831 of file pop3_client.c.

void pop3c_senddele (UINT16)

Definition at line 1222 of file pop3_client.c.

void pop3c_sendlist (UINT16)

Definition at line 1122 of file pop3_client.c.

void pop3c_sendpassword (void)

Definition at line 1066 of file pop3_client.c.

void pop3c_sendquit (void)

Definition at line 1256 of file pop3_client.c.

void pop3c_sendretr (UINT16)

Definition at line 1189 of file pop3_client.c.

void pop3c_sendstat (void)

Definition at line 1102 of file pop3_client.c.

void pop3c_sendtop (UINT16)

Definition at line 1155 of file pop3_client.c.

void pop3c_senduser (void)

Definition at line 1030 of file pop3_client.c.

pop3c_callbacks.c File Reference

Detailed Description

OpenTCP POP3 callback functions.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
11.9.2002

Bug:

Warning:

Todo:

This file holds empty callback functions needed by the POP3 client to get user-specific e-mail data from the application. Add your own code to perform the requested tasks.
Definition in file pop3c_callbacks.c.

```
#include "debug.h"
#include "datatypes.h"
#include "pop3_client.h"
```

Functions

- void pop3c_error (void)
  POP3 client error handler.

- void pop3c_messages (UINT16 msgs)
  Invoked to inform user app about the number of new e-mails.

- INT16 pop3c_msgoffer (UINT16 index, UINT32 msglen, UINT8 *from, UINT8 *subject)

Offers e-mail message to the user app.

- INT8 pop3c_getusername (UINT8 *dbuf)
  Get user name that is to be used for loging to the server.

- INT8 pop3c_getpassword (UINT8 *dbuf)
  Get password that is to be used for loging to the server.

- void pop3c_allok (void)
  Indicates succesfull reading of E-mails.

- void pop3c_data (UINT8 data)
  Receives E-mail data.

Function Documentation

void pop3c_allok (void)

Indicates succesfull reading of E-mails.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

14.10.2002

This callback function is called by POP3 Client to indicate succesfull reading of E-mails

Definition at line 173 of file pop3c_callbacks.c.

void pop3c_data (UINT8 *data*)

Receives E-mail data.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

14.10.2002

Parameters:

data E-mail data as received by POP3 client

This callback function is called by POP3 Client in order to give data to application

Definition at line 187 of file pop3c_callbacks.c.

void pop3c_error (void)

POP3 client error handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

This callback function is called by POP3 Client when there happens error of some kind (timeout, losing of connection etc.).

Definition at line 82 of file pop3c_callbacks.c.

INT8 pop3c_getpassword (UINT8 * *dbuf*)

Get password that is to be used for loging to the server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

dbuf Pointer to buffer to which the password will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by POP3 Client when it wants to know the password of us. The user is responsible of storing that name to destbuf without NULL termination (") and returning number of bytes on the password.

Definition at line 159 of file pop3c_callbacks.c.

INT8 pop3c_getusername (UINT8 * *dbuf*)

Get user name that is to be used for loging to the server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

dbuf Pointer to buffer to which the username will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by POP3 Client when it wants to know the username of us. The user is responsible of storing that name to destbuf without NULL termination (") and returning number of bytes on that username.

Definition at line 139 of file pop3c_callbacks.c.

void pop3c_messages (UINT16 *msgs*)

Invoked to inform user app about the number of new e-mails.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

msgs Number of new e-mails waiting on the server

This callback function is called by POP3 Client in order to indicate the number of new E-mails on server.

Definition at line 96 of file pop3c_callbacks.c.

INT16 pop3c_msgoffer (UINT16 *index*, UINT32 *msglen*, UINT8 * *from*, UINT8 * *subject*)

Offers e-mail message to the user app.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

11.09.2002

Parameters:

index index number of message
msglen Length of message data
from Buffer containing sender (null terminated string)
subject Buffer containing subject (null terminated string)

Returns:

- -2 - Reject the e-mail (delete from server)
- -1 - Skip the e-mail (leave it on server)
- >=0 - Read and delete the mail from server

This callback function is called by POP3 Client in order to offer the e-mail message to user. User can reject this mail, skip this, mail or read it as indicated with return value

Definition at line 117 of file pop3c_callbacks.c.

RTI.c File Reference

```
#include "timers.h"
#include "MC9S12NE64.h"
```

Functions

- void RTI_Init (void)
- void RTI_Enable (void)
- void RTI_Disable (void)
- interrupt void RealTimeInterrupt (void)

Function Documentation

interrupt void RealTimeInterrupt (void)

Definition at line 54 of file RTI.c.

void RTI_Disable (void)

Definition at line 43 of file RTI.c.

void RTI_Enable (void)

Definition at line 32 of file RTI.c.

void RTI_Init (void)

Definition at line 21 of file RTI.c.

---

smtp_client.c File Reference

Detailed Description

OpenTCP SMTP client implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

9.8.2002

Bug:

Warning:

Todo:

OpenTCP implementation of SMTP client that uses TCP api. For interface functions declarations see smtp_client.h.
Definition in file smtp_client.c.

```
#include "datatypes.h"
#include "debug.h"
#include "globalvariables.h"
#include "system.h"
#include "timers.h"
#include "tcp_ip.h"
#include "smtp_client.h"
```

Functions

- INT8 smtpc_connect (UINT32 ip, UINT16 port)
  Start E-mail sending procedure.

- void smtpc_init (void)
  Initializes SMTP client.

- UINT8 smtpc_getstate (void)
  Retrieves SMTP clients' state.

- INT32 smtpc_eventlistener (INT8 cbhandle, UINT8 event, UINT32 par1, UINT32 par2)
- void smtpc_run (void)
- void smtpc_sendhelo (void)
- void smtpc_sendmailfrom (void)
- void smtpc_sendrcptto (void)
- void smtpc_senddatareq (void)
- void smtpc_sendbody (void)
- void smtpc_senddataend (void)
- void smtpc_sendquit (void)
- INT16 smtpc_senddata (void)
- void smtpc_changestate (UINT8 nstate)

Variables

- UINT8 smtpc_init_done = 0
- struct {
- UINT8 **state**
- UINT32 **remip**
- UINT16 **remport**
- INT8 **sochandle**
- UINT8 **tmrhandle**
- UINT16 **unacked**
- UINT16 **bufindex**
- } smtp_client

SMTP client state information.

Function Documentation

void smtpc_changestate (UINT8 *nstate*)

Definition at line 900 of file smtp_client.c.

INT8 smtpc_connect (UINT32 *ip*, UINT16 *port*)

Start E-mail sending procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

ip IP address of SMTP server
port Port number on server (remote port)

Returns:

- - 1 - Error
- >=0 - Connection procedure started (OK)

This function is called by user when she wants to start E-mail sending procedure. The function is responsible of establishing connection to SMTP server. After connection is established the SMTP client engine starts to make callbacks to user functions in order to get E-mail address information, data etc.

Definition at line 120 of file smtp_client.c.

INT32 smtpc_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *par1*, UINT32 *par2*)

Definition at line 219 of file smtp_client.c.

UINT8 smtpc_getstate (void)

Retrieves SMTP clients' state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

6.10.2002

Returns the state of SMTP client

Definition at line 193 of file smtp_client.c.

void smtpc_init (void)

Initializes SMTP client.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This function should be called once when system starts. Make sure that system services e.g. timers, TCP are initialized before initializing applications!

Definition at line 155 of file smtp_client.c.

void smtpc_run (void)

Definition at line 453 of file smtp_client.c.

void smtpc_sendbody (void)

Definition at line 752 of file smtp_client.c.

INT16 smtpc_senddata (void)

Definition at line 879 of file smtp_client.c.

void smtpc_senddataend (void)

Definition at line 837 of file smtp_client.c.

void smtpc_senddatareq (void)

Definition at line 730 of file smtp_client.c.

void smtpc_sendhelo (void)

Definition at line 610 of file smtp_client.c.

void smtpc_sendmailfrom (void)

Definition at line 646 of file smtp_client.c.

void smtpc_sendquit (void)

Definition at line 856 of file smtp_client.c.

void smtpc_sendrcptto (void)

Definition at line 690 of file smtp_client.c.

Variable Documentation

UINT16 bufindex

Definition at line 91 of file smtp_client.c.

UINT32 remip

Definition at line 86 of file smtp_client.c.

UINT16 remport

Definition at line 87 of file smtp_client.c.

struct { ... } smtp_client

SMTP client state information.

smtp_client variable holds various information about the smtp client needed for proper operation.

UINT8 smtpc_init_done = 0

Defines whether smtpc_init has already been invoked or not

Definition at line 75 of file smtp_client.c.

INT8 sochandle

Definition at line 88 of file smtp_client.c.

UINT8 state

Definition at line 85 of file smtp_client.c.

UINT8 tmrhandle

Definition at line 89 of file smtp_client.c.

UINT16 unacked

Definition at line 90 of file smtp_client.c.

---

smtp_client.h File Reference

Detailed Description

OpenTCP SMTP client interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

9.8.2002

OpenTCP SMTP client function declarations, constants, etc.
Definition in file smtp_client.h.

```
#include "datatypes.h"
```

Defines

- #define SMTPC_TOUT 20
- #define SMTP_UNINITIALIZED 1

- #define SMTP_CLOSED 2
- #define SMTP_OPEN_REQUESTED 3
- #define SMTP_CONNECTIONOPEN_SENT 4
- #define SMTP_CONNECTION_OPENED 5
- #define SMTP_SERVER_READY 6
- #define SMTP_HELO_SENT 7
- #define SMTP_HELO_ACKED 8
- #define SMTP_MAILFROM_SENT 9
- #define SMTP_MAILFROM_ACKED 10
- #define SMTP_RCPTTO_SENT 11
- #define SMTP_RCPTTO_ACKED 12
- #define SMTP_DATAREQ_SENT 13
- #define SMTP_DATAREQ_ACKED 14
- #define SMTP_BODY_SENT 15
- #define SMTP_SENDING_DATA 16
- #define SMTP_DATAEND_REACHED 17
- #define SMTP_DATAEND_SENT 18
- #define SMTP_DATAEND_ACKED 19
- #define SMTP_QUIT_SENT 20
- #define SMTP_QUIT_ACKED 21
- #define SMTP_CMD_SERVER_READY '2' + '2' + '0'
- #define SMTP_CMD_OK '2' + '5' + '0'
- #define SMTP_CMD_DATAOK '3' + '5' + '4'
- #define SMTP_CMD_QUITOK '2' + '2' + '1'

Functions

- INT8 smtpc_connect (UINT32, UINT16)
  Start E-mail sending procedure.

- void smtpc_init (void)
  Initializes SMTP client.

- INT32 smtpc_eventlistener (INT8, UINT8, UINT32, UINT32)
- void smtpc_run (void)
- UINT8 smtpc_getstate (void)
  Retrieves SMTP clients' state.

- void smtpc_sendhelo (void)
- void smtpc_sendmailfrom (void)
- void smtpc_sendrcptto (void)
- void smtpc_senddatareq (void)
- void smtpc_sendbody (void)
- void smtpc_senddataend (void)
- void smtpc_sendquit (void)
- INT16 smtpc_senddata (void)
- void smtpc_changestate (UINT8)
- INT8 smtpc_getdomain (UINT8 *)
  Fills in local domain information.

- INT8 smtpc_getsender (UINT8 *)
  Returns senders' e-mail address.

- INT8 smtpc_getreceiver (UINT8 *)
  Returns receivers' e-mail address.

- INT8 smtpc_getsubject (UINT8 *)
  Returns subject of the E-mail.

- INT16 smtpc_getdata (UINT8 *, UINT16)
  Returns e-mail data (message) to be sent.

- void smtpc_dataacked (void)
  Last data received by remote host.

- void smtpc_error (void)
  SMTP client error handler.

- void smtpc_allok (void)
  SMTP client success handler.

Define Documentation

#define SMTP_BODY_SENT 15

SMTP Client state: We have sent RFC822 body

Definition at line 92 of file smtp_client.h.

#define SMTP_CLOSED 2

SMTP Client state: TCP connection closed

Definition at line 79 of file smtp_client.h.

#define SMTP_CMD_DATAOK '3' + '5' + '4'

OK to send data

Definition at line 105 of file smtp_client.h.

#define SMTP_CMD_OK '2' + '5' + '0'

Command executed OK

Definition at line 104 of file smtp_client.h.

#define SMTP_CMD_QUITOK '2' + '2' + '1'

OK to quit, close connection

Definition at line 106 of file smtp_client.h.

#define SMTP_CMD_SERVER_READY '2' + '2' + '0'

Server outputs when connected

Definition at line 103 of file smtp_client.h.

#define SMTP_CONNECTION_OPENED 5

SMTP Client state: TCP Connection opened

Definition at line 82 of file smtp_client.h.

#define SMTP_CONNECTIONOPEN_SENT 4

SMTP Client state: TCP connection request sent

Definition at line 81 of file smtp_client.h.

#define SMTP_DATAEND_ACKED 19

SMTP Client state: Server has acked CRLF.CRLF by 250

Definition at line 96 of file smtp_client.h.

#define SMTP_DATAEND_REACHED 17

SMTP Client state: We have no more data

Definition at line 94 of file smtp_client.h.

#define SMTP_DATAEND_SENT 18

SMTP Client state: CRLF.CRLF sent by us

Definition at line 95 of file smtp_client.h.

#define SMTP_DATAREQ_ACKED 14

SMTP Client state: Server has acked DATA by 354

Definition at line 91 of file smtp_client.h.

#define SMTP_DATAREQ_SENT 13

SMTP Client state: DATA sent by us

Definition at line 90 of file smtp_client.h.

#define SMTP_HELO_ACKED 8

SMTP Client state: Server has acked HELO by 250

Definition at line 85 of file smtp_client.h.

#define SMTP_HELO_SENT 7

SMTP Client state: HELO sent by us

Definition at line 84 of file smtp_client.h.

#define SMTP_MAILFROM_ACKED 10

SMTP Client state: Server has acked MAIL FROM by 250

Definition at line 87 of file smtp_client.h.

#define SMTP_MAILFROM_SENT 9

SMTP Client state: MAIL FROM sent by us

Definition at line 86 of file smtp_client.h.

#define SMTP_OPEN_REQUESTED 3

SMTP Client state: User has requested mail read

Definition at line 80 of file smtp_client.h.

#define SMTP_QUIT_ACKED 21

SMTP Client state: Server has acked quit by 221

Definition at line 98 of file smtp_client.h.

#define SMTP_QUIT_SENT 20

SMTP Client state: QUIT sent by us

Definition at line 97 of file smtp_client.h.

#define SMTP_RCPTTO_ACKED 12

SMTP Client state: Server has acked RCPT TO by 250

Definition at line 89 of file smtp_client.h.

#define SMTP_RCPTTO_SENT 11

SMTP Client state: RCPT To sent by us

Definition at line 88 of file smtp_client.h.

#define SMTP_SENDING_DATA 16

SMTP Client state: We are sending data...

Definition at line 93 of file smtp_client.h.

#define SMTP_SERVER_READY 6

SMTP Client state: SMTP server has indicated 220

Definition at line 83 of file smtp_client.h.

#define SMTP_UNINITIALIZED 1

SMTP Client state: Not initialized yet

Definition at line 78 of file smtp_client.h.

#define SMTPC_TOUT 20

SMTP clients' timeout in seconds

Definition at line 71 of file smtp_client.h.

Function Documentation

void smtpc_allok (void)

SMTP client success handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This callback function is called by SMTP Client when the packet is succesfully delivered to E-mail server.

Definition at line 38 of file smtpc_callbacks.c.

void smtpc_changestate (UINT8)

Definition at line 900 of file smtp_client.c.

INT8 smtpc_connect (UINT32 *ip*, UINT16 *port*)

Start E-mail sending procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

ip IP address of SMTP server
port Port number on server (remote port)

Returns:

- - 1 - Error
- >=0 - Connection procedure started (OK)

This function is called by user when she wants to start E-mail sending procedure. The function is responsible of establishing connection to SMTP server. After connection is established the SMTP client engine starts to make callbacks to user functions in order to get E-mail address information, data etc.

Definition at line 120 of file smtp_client.c.

void smtpc_dataacked (void)

Last data received by remote host.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This callback function is called by SMTP Client when TCP has ensured that the last packet was transmitted succesfully and next time when smtpc_getdata callback is made new data should be assembled

Definition at line 167 of file smtpc_callbacks.c.

void smtpc_error (void)

SMTP client error handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

20.08.2002

This callback function is called by SMTP Client when there happens error of some kind (timeout, losing of connection etc.). It indicates that e-mail was not delivered to server.

Definition at line 22 of file smtpc_callbacks.c.

INT32 smtpc_eventlistener (INT8, UINT8, UINT32, UINT32)

Definition at line 219 of file smtp_client.c.

INT16 smtpc_getdata (UINT8 * *dbuf*, UINT16 *buflen*)

Returns e-mail data (message) to be sent.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the data will be stored
buflen length of data buffer

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to get mail plain data from user. The user is responsible of filling dbuf and returning number of bytes assembled. When data end is

reached the function must return (-1) without storing any bytes to buffer (so just send data untill you don't have any bytes to sent when callback is made to that function and return -1). Do not move read pointer of your data forward before SMTP makes callback to smtpc_dataacked!

Definition at line 149 of file smtpc_callbacks.c.

INT8 smtpc_getdomain (UINT8 * *dbuf*)

Fills in local domain information.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the domain name will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the local domain. The user is responsible of storing that domain to destbuf without NULL termination (") and returning number of bytes on domain.

Definition at line 57 of file smtpc_callbacks.c.

INT8 smtpc_getreceiver (UINT8 * *dbuf*)

Returns receivers' e-mail address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the receiver will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the E-mail address of receiver. The user is responsible of storing that address to destbuf without NULL termination (") and returning number of bytes on E-mail address.

Definition at line 100 of file smtpc_callbacks.c.

INT8 smtpc_getsender (UINT8 * *dbuf*)

Returns senders' e-mail address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the sender will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the E-mail address of sender. The user is responsible of storing that address to destbuf without NULL termination (") and returning number of bytes on E-mail address.

Definition at line 78 of file smtpc_callbacks.c.

UINT8 smtpc_getstate (void)

Retrieves SMTP clients' state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

6.10.2002

Returns the state of SMTP client

Definition at line 193 of file smtp_client.c.

INT8 smtpc_getsubject (UINT8 * *dbuf*)

Returns subject of the E-mail.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the subject will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the subject of E-mail to be sent. The user is responsible of storing subject to destbuf without NULL termination (") and returning number of bytes inserted.

Definition at line 122 of file smtpc_callbacks.c.

void smtpc_init (void)

Initializes SMTP client.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This function should be called once when system starts. Make sure that system services e.g. timers, TCP are initialized before initializing applications!

Definition at line 155 of file smtp_client.c.

void smtpc_run (void)

Definition at line 453 of file smtp_client.c.

void smtpc_sendbody (void)

Definition at line 752 of file smtp_client.c.

INT16 smtpc_senddata (void)

Definition at line 879 of file smtp_client.c.

void smtpc_senddataend (void)

Definition at line 837 of file smtp_client.c.

void smtpc_senddatareq (void)

Definition at line 730 of file smtp_client.c.

void smtpc_sendhelo (void)

Definition at line 610 of file smtp_client.c.

void smtpc_sendmailfrom (void)

Definition at line 646 of file smtp_client.c.

void smtpc_sendquit (void)

Definition at line 856 of file smtp_client.c.

void smtpc_sendrcptto (void)

Definition at line 690 of file smtp_client.c.

---

smtpc_callbacks.c File Reference

```
#include "datatypes.h"
#include "smtp_client.h"
#include <string.h>
```

Functions

- void smtpc_error (void)
  SMTP client error handler.

- void smtpc_allok (void)
  SMTP client success handler.

- INT8 smtpc_getdomain (UINT8 *dbuf)
  Fills in local domain information.

- INT8 smtpc_getsender (UINT8 *dbuf)
  Returns senders' e-mail address.

- INT8 smtpc_getreceiver (UINT8 *dbuf)
  Returns receivers' e-mail address.

- INT8 smtpc_getsubject (UINT8 *dbuf)
  Returns subject of the E-mail.

- INT16 smtpc_getdata (UINT8 *dbuf, UINT16 buflen)
  Returns e-mail data (message) to be sent.

- void smtpc_dataacked (void)
  Last data received by remote host.

Variables

- char * my_domain = "Freescale.com"
- char * my_sender = "NE64EVB@Freescale.com"
- char * my_receiver = "Somebody@freescale.com"
- char * my_subject = "This is a test:"
- char * my_data = "TEST!!!!!.\r\n\r\nTHIS IS AN AUTOMATIC NOTIFICATION, DO NOT REPLY TO THIS EMAIL.\r\n.\r\n"

Function Documentation

void smtpc_allok (void)

SMTP client success handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This callback function is called by SMTP Client when the packet is succesfully delivered to E-mail server.

Definition at line 38 of file smtpc_callbacks.c.

void smtpc_dataacked (void)

Last data received by remote host.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

This callback function is called by SMTP Client when TCP has ensured that the last packet was transmitted succesfully and next time when smtpc_getdata callback is made new data should be assembled

Definition at line 167 of file smtpc_callbacks.c.

void smtpc_error (void)

SMTP client error handler.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

20.08.2002

This callback function is called by SMTP Client when there happens error of some kind (timeout, losing of connection etc.). It indicates that e-mail was not delivered to server.

Definition at line 22 of file smtpc_callbacks.c.

INT16 smtpc_getdata (UINT8 * *dbuf*, UINT16 *buflen*)

Returns e-mail data (message) to be sent.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the data will be stored
buflen length of data buffer

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to get mail plain data from user. The user is responsible of filling dbuf and returning number of bytes assembled. When data end is reached the function must return (-1) without storing any bytes to buffer (so just send data untill you don't have any bytes to sent when callback is made to that function and return -1). Do not move read pointer of your data forward before SMTP makes callback to smtpc_dataacked!

Definition at line 149 of file smtpc_callbacks.c.

INT8 smtpc_getdomain (UINT8 * *dbuf*)

Fills in local domain information.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the domain name will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the local domain. The user is responsible of storing that domain to destbuf without NULL termination (') and returning number of bytes on domain.

Definition at line 57 of file smtpc_callbacks.c.

INT8 smtpc_getreceiver (UINT8 * *dbuf*)

Returns receivers' e-mail address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the receiver will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the E-mail address of receiver. The user is responsible of storing that address to destbuf without NULL termination (') and returning number of bytes on E-mail address.

Definition at line 100 of file smtpc_callbacks.c.

INT8 smtpc_getsender (UINT8 * *dbuf*)

Returns senders' e-mail address.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the sender will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the E-mail address of sender. The user is responsible of storing that address to destbuf without NULL termination (') and returning number of bytes on E-mail address.

Definition at line 78 of file smtpc_callbacks.c.

INT8 smtpc_getsubject (UINT8 * *dbuf*)

Returns subject of the E-mail.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.08.2002

Parameters:

dbuf pointer to buffer to which the subject will be stored

Returns:

- -1 - Error
- >0 - Number of bytes inserted

This callback function is called by SMTP Client when it wants to know the subject of E-mail to be sent. The user is responsible of storing subject to destbuf without NULL termination (') and returning number of bytes inserted.

Definition at line 122 of file smtpc_callbacks.c.

Variable Documentation

char* my_data = "TEST!!!!!.\r\n\r\nTHIS IS AN AUTOMATIC NOTIFICATION, DO NOT REPLY TO THIS EMAIL.\r\n.\r\n"

Definition at line 10 of file smtpc_callbacks.c.

char* my_domain = "Freescale.com"

Definition at line 6 of file smtpc_callbacks.c.

char* my_receiver = "Somebody@freescale.com"

Definition at line 8 of file smtpc_callbacks.c.

char* my_sender = "NE64EVB@Freescale.com"

Definition at line 7 of file smtpc_callbacks.c.

char* my_subject = "This is a test:"

Definition at line 9 of file smtpc_callbacks.c.

---

system.c File Reference

```
#include "datatypes.h"
#include "system.h"
#include "debug.h"
#include "ne64debug.h"
#include "MC9S12NE64.h"
```

Functions

- INT16 __strlen (UINT8 *buf, UINT16 len)
- INT16 bufsearch (UINT8 *startadr, UINT16 len, UINT8 *str)
- UINT8 __tolower (UINT8 ch)
- UINT8 __toupper (UINT8 ch)
- UINT8 isnumeric (UINT8 ch)
- UINT16 hextoascii (UINT8 c)
- UINT8 asciitohex (UINT8 ch)
- void __ltoa (UINT32 nmbr, UINT8 *ch)
- void __itoa (UINT16 nmbr, UINT8 *ch)
- INT16 __atoi (UINT8 *buf, UINT8 buflen)
- void mputs (INT8 *msg)
- void mputhex (UINT8 nbr)
- void kick_WD (void)
- void wait (INT16 i)
- UINT32 random (void)
- void dummy (void)
- void enter_power_save (void)
- void exit_power_save (void)

Variables

- UINT32 base_timer
- UINT8 sleep_mode = 0
- UINT8 net_buf [NETWORK_TX_BUFFER_SIZE]
  Transmit buffer used by all OpenTCP applications.

Function Documentation

INT16 __atoi (UINT8 * *buf*, UINT8 *buflen*)

Definition at line 391 of file system.c.

void __itoa (UINT16 *nmbr*, UINT8 * *ch*)

Definition at line 329 of file system.c.

void __ltoa (UINT32 *nmbr*, UINT8 * *ch*)

Definition at line 269 of file system.c.

INT16 __strlen (UINT8 * *buf*, UINT16 *len*)

Definition at line 50 of file system.c.

UINT8 __tolower (UINT8 *ch*)

Definition at line 181 of file system.c.

UINT8 __toupper (UINT8 *ch*)

Definition at line 204 of file system.c.

UINT8 asciitohex (UINT8 *ch*)

Definition at line 259 of file system.c.

INT16 bufsearch (UINT8 * *startadr*, UINT16 *len*, UINT8 * *str*)

Definition at line 84 of file system.c.

void dummy (void)

Definition at line 460 of file system.c.

void enter_power_save (void)

Definition at line 467 of file system.c.

void exit_power_save (void)

Definition at line 481 of file system.c.

UINT16 hextoascii (UINT8 *c*)

Definition at line 227 of file system.c.

UINT8 isnumeric (UINT8 *ch*)

Definition at line 216 of file system.c.

void kick_WD (void)

Definition at line 438 of file system.c.

void mputhex (UINT8 *nbr*)

Definition at line 430 of file system.c.

void mputs (INT8 * *msg*)

Definition at line 423 of file system.c.

UINT32 random (void)

Definition at line 452 of file system.c.

void wait (INT16 *i*)

Definition at line 444 of file system.c.

Variable Documentation

UINT32 base_timer

System 1.024 msec timer

Definition at line 8 of file system.c.

UINT8 net_buf[NETWORK_TX_BUFFER_SIZE]

Transmit buffer used by all OpenTCP applications.

This buffer is the transmit buffer used by all OpenTCP applications for sending of data. Please note the warnings below for correct usage of this buffer that ensures proper operation of the applications.

Warning:

- **Transmit buffer start** - to avoid data copying, the TCP/IP stack will use first part of the net_buf buffer to add it's data. This means that applications using TCP and/or UDP **must not** write application-level data from the beginning of the buffer but from certain offset. This offset depends on the transport-layer protocol (it's header size that is). For TCP this value is defined by the TCP_APP_OFFSET and for the UDP it is UDP_APP_OFFSET.
- **Buffer sharing** - since all applications share this buffer among each other, and with the TCP/IP stack as well, care must be taken not to overwrite other applications' data before it is sent. This is best achieved if all applications work in the main loop and when they wish to send data they fill in the buffer and send it immediately.

Definition at line 33 of file system.c.

UINT8 sleep_mode = 0

Used to store information about power-saving state we're in (if any)

Definition at line 10 of file system.c.

---

system.h File Reference

```
#include "os.h"
#include "ne64api.h"
#include "datatypes.h"
#include "globalvariables.h"
```

Data Structures

- struct netif
  Network Interface declaration.

Defines

- #define OPENTCP_VERSION "1.0.4"
  OpenTCP major version number.

- #define TRUE 1
- #define FALSE 0
- #define NETWORK_TX_BUFFER_SIZE 1024
  Transmit buffer size.

- #define MASTER_MS_CLOCK base_timer
- #define TXBUF net_buf
- #define RESET_SYSTEM() for(;;) {}
  Macro used to reset the MCU.

- #define OS_EnterCritical OS_ENTER_CRITICAL
  Macro used to enter critical sections.

- #define OS_ExitCritical OS_EXIT_CRITICAL
  Macro used to exit critical sections.

- #define RECEIVE_NETWORK_B() NE64ReadByte()
  Use this macro to read data from Ethernet controller.

- #define RECEIVE_NETWORK_W() NE64ReadWord()
  Use this macro to read data from Ethernet controller.

- #define RECEIVE_NETWORK_BUF(c, d) NE64ReadBytes(c,d)
  Use this macro to read data from Ethernet controller to a buffer.

- #define SEND_NETWORK_B(c) NE64WriteByte(c)
  Use this macro to write data to Ethernet controller.

- #define SEND_NETWORK_W(c) NE64WriteWord(c)
  Use this macro to write data to Ethernet controller.

- #define SEND_NETWORK_BUF(c, d) NE64WriteBytes(c,d)
  Use this macro to write data from buffer to Ethernet controller.

- #define NETWORK_CHECK_IF_RECEIVED() NE64ValidFrameReception()
  Use this macro to check if there is recieved data in Ethernet controller.

- #define NETWORK_RECEIVE_INITIALIZE(c) NE64InitializeOffsetToReadRxBuffer(c)
  Initialize reading from a given address.

- #define NETWORK_RECEIVE_END() NE64FreeReceiveBuffer()
  Dump received packet in the Ethernet controller.

- #define NETWORK_COMPLETE_SEND(c) NE64StartFrameTransmission(c)
  Send the Ethernet packet that was formed in the Ethernet controller.

- #define NETWORK_SEND_INITIALIZE(c) NE64InitializeTransmissionBuffer(c)
  Initialize sending of Ethernet packet from a given address.

- #define NETWORK_ADD_DATALINK(c) NE64WriteEthernetHeaderToTxBuffer((struct TEthernetFrame*)c)
  Add lower-level datalink information.

Functions

- void kick_WD (void)
- void wait (INT16)
- void enter_power_save (void)
- void exit_power_save (void)
- INT16 __strlen (UINT8 *, UINT16)
- INT16 bufsearch (UINT8 *, UINT16, UINT8 *)
- UINT16 hextoascii (UINT8)
- void __itoa (UINT16, UINT8 *)
- void __ltoa (UINT32, UINT8 *)
- INT16 __atoi (UINT8 *, UINT8)
- UINT8 asciitohex (UINT8)
- UINT8 isnumeric (UINT8)
- void mputs (INT8 *)
- void mputhex (UINT8)
- UINT32 random (void)
- void dummy (void)
- UINT8 __tolower (UINT8)
- UINT8 __toupper (UINT8)
- void init (void)

Define Documentation

#define FALSE 0

Boolean FALSE value as used in the OpenTCP

Definition at line 21 of file system.h.

#define MASTER_MS_CLOCK base_timer

Interrupt driven msec free-running clock

Definition at line 93 of file system.h.

#define NETWORK_ADD_DATALINK(c) NE64WriteEthernetHeaderToTxBuffer((struct TEthernetFrame*)c)

Add lower-level datalink information.

This implementation adds Ethernet data-link information by invoking WriteEthernetHeader() function that writes Ethernet header based on information provided (destination and source ethernet address and protocol field).

Definition at line 293 of file system.h.

#define NETWORK_CHECK_IF_RECEIVED() NE64ValidFrameReception()

Use this macro to check if there is recieved data in Ethernet controller.

Invoke this macro periodically (see main_demo.c for example) to check if there is new data in the Ethernet controller.

If there is new data in the Ethernet controller, this macro (function that it points to that is) will return a value of TRUE and fill in the appropriate values in the received_frame variable. Otherwise it returns FALSE.

Definition at line 249 of file system.h.

#define NETWORK_COMPLETE_SEND(c) NE64StartFrameTransmission(c)

Send the Ethernet packet that was formed in the Ethernet controller.

After the data has been written to the Ethernet controller, use this function to instruct the Ethernet controller that data is in it's internal buffer and should be sent.

Definition at line 275 of file system.h.

#define NETWORK_RECEIVE_END() NE64FreeReceiveBuffer()

Dump received packet in the Ethernet controller.

Invoke this macro when the received Ethernet packet is not needed any more and can be discarded.

Definition at line 266 of file system.h.

#define NETWORK_RECEIVE_INITIALIZE(c) NE64InitializeOffsetToReadRxBuffer(c)

Initialize reading from a given address.

This macro initializes reading of the received Ethernet frame from a given address in the Ethernet controller.

Definition at line 257 of file system.h.

#define NETWORK_SEND_INITIALIZE(c) NE64InitializeTransmissionBuffer(c)

Initialize sending of Ethernet packet from a given address.

Use this function to initialize sending (or creating) of an Ethernet packet from a given address in the Ethernet controller.

Definition at line 283 of file system.h.

#define NETWORK_TX_BUFFER_SIZE 1024

Transmit buffer size.

NETWORK_TX_BUFFER_SIZE defines the size of the network buffer used for data transmission by ICMP as well as TCP and UDP applications.

See net_buf documentation for more reference on the shared transmit buffer.

Definition at line 33 of file system.h.

#define OPENTCP_VERSION "1.0.4"

OpenTCP major version number.

This define represents OpenTCP version information. Version is in the format MAJOR.MINOR.PATCH.

Definition at line 17 of file system.h.

#define OS_EnterCritical OS_ENTER_CRITICAL

Macro used to enter critical sections.

Todo:

- Move this to other arch-dependant place

This is highly dependant on the architecture that is used and/or possible operating system beeing used so it will be moved to some other place in the future.

Usually disabling globally interrupts works just fine :-)

Definition at line 119 of file system.h.

#define OS_ExitCritical OS_EXIT_CRITICAL

Macro used to exit critical sections.

Todo:

- Move this to other arch-dependant place

This is highly dependant on the architecture that is used and/or possible operating system beeing used so it will be moved to some other place in the future.

For now this only globally enables interrupts

Definition at line 132 of file system.h.

#define RECEIVE_NETWORK_B() NE64ReadByte()

Use this macro to read data from Ethernet controller.

This macro should be used to read data from the Ethernet controller. Procedure for doing this would be as follows:

- Initialize reading of data from certain address in the Ethernet controller (usually you will do that based on buf_index value of ip_frame, udp_frame or tcp_frame type of variables; in certain special situations you can also use buf_index from ethernet_frame type of var.
- Keep invoking RECEIVE_NETWORK_B() to read one byte at a time from the ethernet controller. Take care not to read more data than actually received
- If needed, reinitialize reading of data again and start all over again
- When finished discard the current frame in the Ethernet controller by invoking NETWORK_RECEIVE_END() macro

Definition at line 153 of file system.h.

#define RECEIVE_NETWORK_BUF(c, d) NE64ReadBytes(c,d)

Use this macro to read data from Ethernet controller to a buffer.

This macro should be used to read data from the Ethernet controller to a buffer in memory. Procedure for using this macro is the same as for using RECEIVE_NETWORK_B() macro.

Definition at line 185 of file system.h.

#define RECEIVE_NETWORK_W() NE64ReadWord()

Use this macro to read data from Ethernet controller.

This macro should be used to read data from the Ethernet controller. Procedure for doing this would be as follows:

- Initialize reading of data from certain address in the Ethernet controller (usually you will do that based on buf_index value of ip_frame, udp_frame or tcp_frame type of variables; in certain special situations you can also use buf_index from ethernet_frame type of var.
- Keep invoking RECEIVE_NETWORK_W() to read one word at a time from the ethernet controller. Take care not to read more data than actually received
- If needed, reinitialize reading of data again and start all over again
- When finished discard the current frame in the Ethernet controller by invoking NETWORK_RECEIVE_END() macro

Definition at line 174 of file system.h.

#define RESET_SYSTEM() for(;;) {}

Macro used to reset the MCU.

By default this macro is only an infinite loop and the system is reset by the (presumably) running watchdog timer.

Change this if another form of reset is desired/needed.

Definition at line 106 of file system.h.

#define SEND_NETWORK_B(c) NE64WriteByte(c)

Use this macro to write data to Ethernet controller.

This macro should be used to write data to Ethernet controller. Procedure for doing this would be as follows:

- Initialize writing of data to certain address in the Ethernet controller. Buffer space in Ethernet controller is divided among the protocols in the following way:
- 256 byte Tx for ARP (see ARP_BUFFER)
- 1536 byte Tx for ICMP (see ICMP_BUF)
- 1536 byte Tx for TCP (see TCP_BUF)
- 1536 byte Tx for UDP (see UDP_BUF)
- Write the data by using SEND_NETWORK_B() macro
- When all of the data is written instruct the Ethernet controller to send the data by calling the NETWORK_COMPLETE_SEND() macro with number of bytes to send as a parameter

Definition at line 205 of file system.h.

#define SEND_NETWORK_BUF(c, d) NE64WriteBytes(c,d)

Use this macro to write data from buffer to Ethernet controller.

This macro should be used to write data from a buffer to Ethernet controller. Usage is the same as for the SEND_NETWORK_B() macro.

Definition at line 235 of file system.h.

#define SEND_NETWORK_W(c) NE64WriteWord(c)

Use this macro to write data to Ethernet controller.

This macro should be used to write data to Ethernet controller. Procedure for doing this would be as follows:

- Initialize writing of data to certain address in the Ethernet controller. Buffer space in Ethernet controller is divided among the protocols in the following way:
- 256 byte Tx for ARP (see ARP_BUFFER)
- 1536 byte Tx for ICMP (see ICMP_BUF)
- 1536 byte Tx for TCP (see TCP_BUF)
- 1536 byte Tx for UDP (see UDP_BUF)
- Write the data by using SEND_NETWORK_W() macro
- When all of the data is written instruct the Ethernet controller to send the data by calling the NETWORK_COMPLETE_SEND() macro with number of bytes to send as a parameter

Definition at line 226 of file system.h.

#define TRUE 1

Boolean TRUE value as used in the OpenTCP

Definition at line 20 of file system.h.

#define TXBUF net_buf

TXBUF points to transmit network buffer

Definition at line 94 of file system.h.

Function Documentation

INT16 __atoi (UINT8 *, UINT8)

Definition at line 391 of file system.c.

void __itoa (UINT16, UINT8 *)

Definition at line 329 of file system.c.

void __ltoa (UINT32, UINT8 *)

Definition at line 269 of file system.c.

INT16 __strlen (UINT8 *, UINT16)

Definition at line 50 of file system.c.

UINT8 __tolower (UINT8)

Definition at line 181 of file system.c.

UINT8 __toupper (UINT8)

Definition at line 204 of file system.c.

UINT8 asciitohex (UINT8)

Definition at line 259 of file system.c.

INT16 bufsearch (UINT8 *, UINT16, UINT8 *)

Definition at line 84 of file system.c.

void dummy (void)

Definition at line 460 of file system.c.

void enter_power_save (void)

Definition at line 467 of file system.c.

void exit_power_save (void)

Definition at line 481 of file system.c.

UINT16 hextoascii (UINT8)

Definition at line 227 of file system.c.

void init (void)

Definition at line 9 of file Init.c.

UINT8 isnumeric (UINT8)

Definition at line 216 of file system.c.

void kick_WD (void)

Definition at line 438 of file system.c.

void mputhex (UINT8)

Definition at line 430 of file system.c.

void mputs (INT8 *)

Definition at line 423 of file system.c.

UINT32 random (void)

Definition at line 452 of file system.c.

void wait (INT16)

Definition at line 444 of file system.c.

tcp.c File Reference

```
#include "debug.h"
#include "datatypes.h"
#include "timers.h"
#include "ethernet.h"
#include "ip.h"
#include "tcp_ip.h"
#include "system.h"
```

Functions

- INT8 tcp_getsocket (UINT8 soctype, UINT8 tos, UINT16 tout, INT32(*listener)(INT8, UINT8, UINT32, UINT32))
  Allocate a free socket in TCP socket pool.

- INT8 tcp_releasesocket (INT8 sochandle)
  Release a TCP socket.

- INT8 tcp_listen (INT8 sochandle, UINT16 port)
  Put TCP socket to listen on a given port.

- INT8 tcp_connect (INT8 sochandle, UINT32 ip, UINT16 rport, UINT16 myport)
  Initialize connection establishment towards remote IP&port.

- INT16 tcp_send (INT8 sockethandle, UINT8 *buf, UINT16 blen, UINT16 dlen)
  Send user data over TCP using given TCP socket.

- INT8 tcp_close (INT8 sochandle)
  Initiate TCP connection closing procedure.

- INT8 tcp_getstate (INT8 sochandle)
  Get current state of the socket.

- INT16 tcp_checksend (INT8 sochandle)
  Checks if it's possible to send data using given socket.

- INT8 tcp_abort (INT8 sochandle)
  Reset connection and place socket to closed state.

- void tcp_poll (void)
  Poll TCP sockets periodically.

- INT8 tcp_init (void)
  Initialize TCP module.

- INT16 process_tcp_in (struct ip_frame *frame, UINT16 len)
  Check and process the received TCP frame.

- INT16 process_tcp_out (INT8 sockethandle, UINT8 *buf, UINT16 blen, UINT16 dlen)
  Create and send TCP packet.

- void tcp_sendcontrol (INT8 sockethandle)
  Send a TCP control packet (no data).

- void tcp_sendreset (struct tcp_frame *frame, UINT32 remip)
  Send a reset (RST) packet to remote host.

- UINT32 tcp_initseq (void)
  Get and return initial sequence number.

- INT8 tcp_mapsocket (struct ip_frame *ipframe, struct tcp_frame *tcpframe)
  Try to match received TCP packet to a socket.

- void tcp_newstate (struct tcb *soc, UINT8 nstate)
  Change TCP socket state and reinitialize timers.

- UINT16 tcp_getfreeport (void)
  Returns next free (not used) local port number.

- UINT8 tcp_check_cs (struct ip_frame *ipframe, UINT16 len)
  Check if TCP checksum check's out.

Variables

- tcp_frame received_tcp_packet
  Used for storing field information about the received TCP packet.

- tcb tcp_socket [NO_OF_TCPSOCKETS+1]
  TCP table holding connection parameters for every TCP socket.

- UINT8 tcp_tempbuf [MIN_TCP_HLEN+1]

Function Documentation

INT16 process_tcp_in (struct ip_frame * *frame*, UINT16 *len*)

Check and process the received TCP frame.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.07.2002

Parameters:

frame pointer to received ip_frame structure
len length of data contained in IP datagram (in bytes)

Returns:

- -1 - Error (packet not OK, or not TCP,or something else)
- >0 - Packet OK

Invoke this function to process received TCP frames. See main_demo.c for an example on how to accomplish this.

Definition at line 1118 of file tcp.c.

INT16 process_tcp_out (INT8 *sockethandle*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Create and send TCP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

16.07.2002

Parameters:

sockethandle handle to processed socket
buf pointer to data buffer (where TCP header will be stored)
blen buffer length in bytes
dlen length of data in bytes

Returns:

- -1 - Error
- >0 - Packet OK

Based on data supplied as function parameters and data stored in socket's tcb, TCP header is created in buffer, checksum is calculated and packet is forwarded to lower layers (IP).

Definition at line 1982 of file tcp.c.

INT8 tcp_abort (INT8 *sochandle*)

Reset connection and place socket to closed state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be aborted

Returns:

- -1 - error
- >=0 - OK (value represents handle to aborted socket)

Use this function in cases when TCP connection must be immediately closed. Note that the preffered (more elegant) way of closing the TCP connection is to invoke tcp_close() which starts a proper closing procedure. tcp_abort should be used only in cases when it is really necessary to immediately and quickly close the connection.

Definition at line 616 of file tcp.c.

UINT8 tcp_check_cs (struct ip_frame * *ipframe*, UINT16 *len*)

Check if TCP checksum check's out.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

16.07.2002

Parameters:

ipframe pointer to IP frame that carried TCP message
len length of TCP portion

Returns:

- 0 - checksum corrupted
- 1 - checksum OK

Function recalculates TCP checksum (pseudoheader+header+data) and compares it to received checksum to see if everything is OK or there is a problem with the checksum.

Definition at line 2425 of file tcp.c.

INT16 tcp_checksend (INT8 *sochandle*)

Checks if it's possible to send data using given socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

23.07.2002

Parameters:

sochandle handle to the socket to be inspected

Returns:

- -1 - not possible to send over a socket (previously sent data is still not akcnowledged)
- >0 - it is possible to send data over a socket

Invoke this function to get information whether it is possible to send data or not. This may, sometimes, be preffered way of getting this type of information to waiting for TCP_EVENT_ACK in event_listener function.

Definition at line 576 of file tcp.c.

INT8 tcp_close (INT8 *sochandle*)

Initiate TCP connection closing procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket on which TCP connection is to be closed

Returns:

- -2 - there is unacked data on this socket. Try again later.
- -1 - Error
- >=0 - OK (connection closing procedure started. Handle to socket returned)

Invoke this function to start connetion closing procedure over a given socket. Note that connection is not immediately closed. It may take some time for that to happen. Event_listener function will be invoked with appropriate event when that really happens.

Definition at line 447 of file tcp.c.

INT8 tcp_connect (INT8 *sochandle*, UINT32 *ip*, UINT16 *rport*, UINT16 *myport*)

Initialize connection establishment towards remote IP&port.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be used for connection establishment
ip remote IP address to connect to
rport remote port number to connect to
myport local port to use for connection. This value can be specified directly or, if a value of 0 is given, TCP module will determine local TCP port automatically.

Returns:

- -1 - Error
- >=0 - OK (Connection establishment procedure started. Socket handle returned.)

Invoke this function to start connection establishment procedure towards remote host over some socket. This is only possible if socket was defined as either TCP_TYPE_CLIENT or TCP_TYPE_CLIENT_SERVER. Function will make some basic checks and if everything is OK, corresponding tcb socket entry will be initialized and connection procedure started.

Definition at line 283 of file tcp.c.

UINT16 tcp_getfreeport (void)

Returns next free (not used) local port number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Returns:

- 0 - no free ports!
- >0 - free local TCP port number

Function attempts to find new local port number that can be used to establish a connection.

Definition at line 2369 of file tcp.c.

INT8 tcp_getsocket (UINT8 *soctype*, UINT8 *tos*, UINT16 *tout*, INT32(* *listener*)(INT8, UINT8, UINT32, UINT32))

Allocate a free socket in TCP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

soctype type of socket wanted. Can take one of the following values:

- TCP_TYPE_NONE
- TCP_TYPE_SERVER
- TCP_TYPE_CLIENT
- TCP_TYPE_CLIENT_SERVER

tos type of service for socket. For now only TCP_TOS_NORMAL.
tout Timeout of socket in seconds. Defines after how many seconds of inactivity (application not sending and/or receiving any data over TCP connection) will the TCP socket automatically be closed.
listener pointer to callback function that will be invoked by the TCP/IP stack to inform socket application of certain events. See tcpc_demo_eventlistener() and tcps_demo_eventlistener() for more information on events and possible actions.

Returns:

- -1 - Error getting requested socket
- >=0 - Handle to reserved socket

Invoke this function to try to obtain a free socket from TCP socket pool. Function returns a handle to the free socket that is later used for accessing the allocated socket (opening, connecting, sending data, closing, aborting, etc.).

Definition at line 79 of file tcp.c.

INT8 tcp_getstate (INT8 *sochandle*)

Get current state of the socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to the socket to be queried

Returns:

- -1 - Error
- >0 - Socket state

Use this function for querying socket state. This is usually not needed directly, but could be usefull for some special purposes.

Definition at line 540 of file tcp.c.

INT8 tcp_init (void)

Initialize TCP module.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Returns:

- -1 - error
- >0 - number of sockets initialized

Warning:

- This function **must** be invoked at startup before any other TCP functions are invoked. This function initializes all sockets and corresponding tcbs to known state. Timers are also allocated for each socket and everything is brought to a predefined state.

Definition at line 1028 of file tcp.c.

UINT32 tcp_initseq (void)

Get and return initial sequence number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

17.07.2002

Returns:

UINT32 number containing initial sequence number to be used

This function returns initial sequence number to be used in a TCP connection. For now, initial sequence number is selected based on base_timer value, which should be solid enough choice.

Definition at line 2203 of file tcp.c.

INT8 tcp_listen (INT8 *sochandle*, UINT16 *port*)

Put TCP socket to listen on a given port.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be placed to listen state
port TCP port number on which it should listen

Returns:

- -1 - Error
- >=0 - OK (Socket put to listening state. Handle to socket returned)

This function will attempt to put socket to listening state. This is only possible if socket was defined as either TCP_TYPE_SERVER or TCP_TYPE_CLIENT_SERVER. If basic correctness checks pass, socket is put to listening mode and corresponding tcb entry is initialized.

Definition at line 202 of file tcp.c.

INT8 tcp_mapsocket (struct ip_frame * *ipframe*, struct tcp_frame * *tcpframe*)

Try to match received TCP packet to a socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.07.2002

Parameters:

ipframe pointer to received IP frame
tcpframe pointer to received TCP frame to be mapped

Returns:

- -1 - Error (no resources or no socket found)
- >=0 - Handle to mapped socket

Function iterates through socket table trying to find a socket for whom this TCP packet is intended.

Definition at line 2226 of file tcp.c.

void tcp_newstate (struct tcb * *soc*, UINT8 *nstate*)

Change TCP socket state and reinitialize timers.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2002

Parameters:

soc pointer to socket structure we're working with
nstate new socket state

This function is used for every state-change that occurs in the TCP sockets so as to provide correct timers/retransmittions that ensure TCP connection is lasting.

Definition at line 2313 of file tcp.c.

void tcp_poll (void)

Poll TCP sockets periodically.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.07.2002

Warning:

- This function **must be** invoked periodically from the main loop. See main_demo.c for an example.

This function checks all TCP sockets and performs various actions if timeouts occur. What kind of action is performed is defined by the state of the TCP socket.

Definition at line 682 of file tcp.c.

INT8 tcp_releasesocket (INT8 *sochandle*)

Release a TCP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be released

Returns:

- -1 - Error releasing the socket (Wrong socket handle or socket not in proper state to be released)
- >=0 - handle of the released socket (can not be used any more untill allocated again with tcp_getsocket()).

Once the application does not need the TCP socket any more it can invoke this function in order to release it. This is usefull if there is a very limited number of sockets (in order to save some memory) shared among several applications.

Definition at line 146 of file tcp.c.

INT16 tcp_send (INT8 *sockethandle*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Send user data over TCP using given TCP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

25.07.2002

Parameters:

sockethandle handle to TCP socket to be used for sending data
buf pointer to data buffer (start of user data)
blen buffer length in bytes (without space reserved at the beginning of buffer for headers)
dlen length of user data to be sent (in bytes)

Returns:

- -1 - Error
- >0 - OK (number represents number of bytes actually sent)

Warning:

- *buf* parameter is a pointer to data to be sent in user buffer. But note that there **MUST** be sufficient free buffer space before that data for TCP header (of MIN_TCP_HLEN size).

Invoke this function to initiate data sending over TCP connection established over a TCP socket. Since data is not buffered (in order to reduce RAM memory consumption) new data can not be sent until data that was previously sent is acknowledged. So, application knows when it can send new data either by:

- waiting for TCP_EVENT_ACK in event_listener function
- invoking tcp_check_send() function to check if it is possible to send data

Definition at line 381 of file tcp.c.

void tcp_sendcontrol (INT8 *sockethandle*)

Send a TCP control packet (no data).

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

17.07.2002

Parameters:

sockethandle handle to socket

This function is used to initiate sending of a control (no data) TCP packet. Important thing in these packets are the flags and sequence numbers they carry.

Definition at line 2113 of file tcp.c.

void tcp_sendreset (struct tcp_frame * *frame*, UINT32 *remip*)

Send a reset (RST) packet to remote host.

Author:

- Jari Lahti (jari.lahti@violasystems.com)
- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

20.08.2002

Parameters:

frame pointer to received TCP packet
remip remote IP address of packet

Uses socket NO_OF_TCPSOCKETS to send a RESET packet to peer. This function is used when we are establishing connection but we receive something else than SYN or SYN+ACK when it's possible that the peer has still old connection on which needs to be resetted without canceling the connection establishment on process.

Definition at line 2152 of file tcp.c.

Variable Documentation

struct tcp_frame received_tcp_packet

Used for storing field information about the received TCP packet.

Various fields from the TCP packet are stored in this variable. These values are then used to perform the necessary actions as defined by the TCP specification: correctnes of the received TCP packet is checked by analyzing these fields, appropriate socket data is adjusted and/or control packet is sent based on it. See tcp_frame definition for struct information.

Definition at line 20 of file tcp.c.

struct tcb tcp_socket[NO_OF_TCPSOCKETS + 1]

TCP table holding connection parameters for every TCP socket.

TCP table is an array of tcp_socket structures holding all of the necessary information about the state, timers, timeouts and sequence and port numbers of the TCP sockets opened. Number of TCP sockets that can be opened at any given time is defined by the NO_OF_TCPSOCKETS and may be changed in order to change the amount of used RAM memory. See tcb definition for more information about the structure itself.

Note:

As seen in the code, an array size is actually bigger for one than the NO_OF_TCPSOCKETS defines. The last entry is used for sending control packets as answers to incoming TCP packets that do not map to any existing TCP sockets.

Definition at line 36 of file tcp.c.

UINT8 tcp_tempbuf[MIN_TCP_HLEN + 1]

Temporary buffer used for sending TCP control packets

Definition at line 38 of file tcp.c.

---

tcp_ip.h File Reference

Detailed Description

OpenTCP ARP interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
1.2.2002

OpenTCP TCP and UDP protocol function declarations, constants, etc.
Definition in file tcp_ip.h.

```
#include "datatypes.h"
#include "ethernet.h"
#include "ip.h"
```

Data Structures

- struct tcb
  TCP transmission control block.

- struct tcp_frame
  TCP header information.

- struct ucb
  UDP control block.

- struct udp_frame
  UDP header information.

Defines

- #define NO_OF_TCPSOCKETS 8
  Defines number of TCP sockets available.

- #define NO_OF_UDPSOCKETS 4
  Defines number of UDP sockets available.

- #define TCP_PORTS_END 1023
  Define reserved-ports space.

- #define UDP_PORTS_END 1023
  Define reserved-ports space.

- #define UDP_OPT_NONE 0
  Disable checksum calculation for UDP socket.

- #define UDP_OPT_SEND_CS 1
  Enable checksum calculation for outgoing UDP packets.

- #define UDP_OPT_CHECK_CS 2
  Enable checksum checking for received UDP packets.

- #define UDP_SEND_MTU ETH_MTU - ETH_HEADER_LEN - UDP_HLEN - IP_MAX_HLEN
- #define UDP_HLEN 8
- #define MIN_TCP_HLEN 20
- #define MAX_TCP_OPTLEN 40
- #define TCP_DEF_MTU 512
- #define TCP_DEF_RETRIES 7
  Number of attempted TCP retransmissions before giving up.

- #define TCP_CON_ATTEMPTS 7
- #define TCP_DEF_KEEPALIVE 4
  Defines a number of seconds after which an empty ACK packet is sent.

- #define TCP_DEF_RETRY_TOUT 4
  Default data-retransmission period (in seconds).

- #define TCP_INIT_RETRY_TOUT 1
  Initial retransmission period (in seconds).

- #define TCP_SYN_RETRY_TOUT 2
  Retranmission period for SYN packet.

- #define TCP_TOS_NORMAL 0
  Defines normal type of service for TCP socket.

- #define TCP_DEF_TOUT 120
  Default idle timeout.

- #define TCP_HALF_SEQ_SPACE 0x0000FFFF
- #define ICMP_TYPE_DEST_UNREACHABLE 3
- #define ICMP_ECHO_REQUEST 8
- #define ICMP_ECHO_REPLY 0
- #define ICMP_CODE_FRAGMENTATION_NEEDED_DF_SET 4
- #define ICMP_MTUMSG_LEN 16
- #define ICMP_ECHOREQ_HLEN 8
- #define ICMP_TEMPIPSET_DATALEN 102
- #define UDP_STATE_FREE 1
  Defines that UDP socket is free and available.

- #define UDP_STATE_CLOSED 2
  Defines that UDP socket is allocated but closed.

- #define UDP_STATE_OPENED 3
  Defines that UDP socket is allocated and opened.

- #define UDP_EVENT_DATA 64
  Only UDP event notified to UDP socket event listener.

- #define TCP_FLAG_ACK 0x10
- #define TCP_FLAG_PUSH 0x08
- #define TCP_FLAG_RESET 0x04
- #define TCP_FLAG_SYN 0x02
- #define TCP_FLAG_FIN 0x01
- #define TCP_INTFLAGS_CLOSEPENDING 0x01
- #define TCP_TYPE_NONE 0x00
  TCP socket is nor a client nor a server.

- #define TCP_TYPE_SERVER 0x01
  TCP socket represents a server application.

- #define TCP_TYPE_CLIENT 0x02
  TCP socket represents a client application.

- #define TCP_TYPE_CLIENT_SERVER 0x03
  TCP socket can act as client or as server.

- #define TCP_STATE_FREE 1
- #define TCP_STATE_RESERVED 2
- #define TCP_STATE_CLOSED 3
- #define TCP_STATE_LISTENING 4
- #define TCP_STATE_SYN_RECEIVED 5
- #define TCP_STATE_SYN_SENT 6
- #define TCP_STATE_FINW1 7
- #define TCP_STATE_FINW2 8
- #define TCP_STATE_CLOSING 9
- #define TCP_STATE_LAST_ACK 10
- #define TCP_STATE_TIMED_WAIT 11
- #define TCP_STATE_CONNECTED 12
- #define TCP_EVENT_CONREQ 1
  Connection request event.

- #define TCP_EVENT_CONNECTED 2
  Connection established event.

- #define TCP_EVENT_CLOSE 4

Connection closed event.

- #define TCP_EVENT_ABORT 8
  Connection aborted event.

- #define TCP_EVENT_ACK 16
  Data acknowledged event.

- #define TCP_EVENT_REGENERATE 32
  Regenerate data event.

- #define TCP_EVENT_DATA 64
  Data arrival event.

- #define TCP_APP_OFFSET MIN_TCP_HLEN
  Transmit buffer offset for TCP applications.

- #define UDP_APP_OFFSET UDP_HLEN
  Transmit buffer offset for UDP applications.

Functions

- INT16 process_icmp_in (struct ip_frame *, UINT16)
  Process recieved ICMP datagram.

- INT8 udp_init (void)
  Initialize UDP socket pool.

- INT8 udp_getsocket (UINT8, INT32(*)(INT8, UINT8, UINT32, UINT16, UINT16, UINT16), UINT8)
  Allocate a free socket in UDP socket pool.

- INT8 udp_releasesocket (INT8)
  Release a given socket.

- INT8 udp_open (INT8, UINT16)
  Open a given UDP socket for communication.

- INT8 udp_close (INT8)
  Close given socket for communication.

- INT16 udp_send (INT8, UINT32, UINT16, UINT8 *, UINT16, UINT16)
  Send data to remote host using given UDP socket.

- INT16 process_udp_in (struct ip_frame *, UINT16)
  Process received UDP frame.

- UINT16 udp_getfreeport (void)
  Returns next free (not used) local port number.

- INT16 process_tcp_in (struct ip_frame *, UINT16)
  Check and process the received TCP frame.

- INT16 process_tcp_out (INT8, UINT8 *, UINT16, UINT16)
  Create and send TCP packet.

- INT8 tcp_init (void)
  Initialize TCP module.

- INT8 tcp_listen (INT8, UINT16)
  Put TCP socket to listen on a given port.

- INT8 tcp_mapsocket (struct ip_frame *, struct tcp_frame *)
  Try to match received TCP packet to a socket.

- UINT8 tcp_check_cs (struct ip_frame *, UINT16)
  Check if TCP checksum check's out.

- void tcp_sendcontrol (INT8)
  Send a TCP control packet (no data).

- UINT32 tcp_initseq (void)
  Get and return initial sequence number.

- void tcp_poll (void)
  Poll TCP sockets periodically.

- void tcp_newstate (struct tcb *, UINT8)
  Change TCP socket state and reinitialize timers.

- INT8 tcp_getsocket (UINT8, UINT8, UINT16, INT32(*)(INT8, UINT8, UINT32, UINT32))
  Allocate a free socket in TCP socket pool.

- INT8 tcp_releasesocket (INT8)
  Release a TCP socket.

- INT8 tcp_connect (INT8 sochandle, UINT32 ip, UINT16 rport, UINT16 myport)

Initialize connection establishment towards remote IP&port.

- INT16 tcp_send (INT8, UINT8 *, UINT16, UINT16)
  Send user data over TCP using given TCP socket.

- INT8 tcp_close (INT8)
  Initiate TCP connection closing procedure.

- void tcp_sendreset (struct tcp_frame *, UINT32)
  Send a reset (RST) packet to remote host.

- INT8 tcp_getstate (INT8)
  Get current state of the socket.

- UINT16 tcp_getfreeport (void)
  Returns next free (not used) local port number.

- INT16 tcp_checksend (INT8)
  Checks if it's possible to send data using given socket.

- INT8 tcp_abort (INT8)
  Reset connection and place socket to closed state.

Define Documentation

#define ICMP_CODE_FRAGMENTATION_NEEDED_DF_SET 4

Definition at line 235 of file tcp_ip.h.

#define ICMP_ECHO_REPLY 0

Definition at line 234 of file tcp_ip.h.

#define ICMP_ECHO_REQUEST 8

Definition at line 233 of file tcp_ip.h.

#define ICMP_ECHOREQ_HLEN 8

Definition at line 237 of file tcp_ip.h.

#define ICMP_MTUMSG_LEN 16

Definition at line 236 of file tcp_ip.h.

#define ICMP_TEMPIPSET_DATALEN 102

Definition at line 238 of file tcp_ip.h.

#define ICMP_TYPE_DEST_UNREACHABLE 3

Definition at line 232 of file tcp_ip.h.

#define MAX_TCP_OPTLEN 40

Definition at line 147 of file tcp_ip.h.

#define MIN_TCP_HLEN 20

Definition at line 146 of file tcp_ip.h.

#define NO_OF_TCPSOCKETS 8

Defines number of TCP sockets available.

Change this number to change maximum number of TCP sockets available to the application.

Definition at line 81 of file tcp_ip.h.

#define NO_OF_UDPSOCKETS 4

Defines number of UDP sockets available.

Change this number to change maximum number of UDP sockets available to the application.

Definition at line 91 of file tcp_ip.h.

#define TCP_APP_OFFSET MIN_TCP_HLEN

Transmit buffer offset for TCP applications.

This value defines offset that TCP applications must use when writing to transmit buffer. This many bytes will be used **before** the first byte of applications data in the transmit buffer to store TCP header.

Definition at line 459 of file tcp_ip.h.

#define TCP_CON_ATTEMPTS 7

Definition at line 169 of file tcp_ip.h.

#define TCP_DEF_KEEPALIVE 4

Defines a number of seconds after which an empty ACK packet is sent.

If for TCP_DEF_KEEPALIVE seconds nothing is received/sent over the TCP connection (this includes received empty TCP packets) an empty (keep-alive) TCP packet will be sent to check if the other side is still replying (and able to reply).

Definition at line 180 of file tcp_ip.h.

#define TCP_DEF_MTU 512

Definition at line 148 of file tcp_ip.h.

#define TCP_DEF_RETRIES 7

Number of attempted TCP retransmissions before giving up.

This number defines how many times will TCP module try to retransmit the data before recognizing that connection was broken. Increase this value for high-latency, low-throughput networks with substantial packet loss.

Definition at line 160 of file tcp_ip.h.

#define TCP_DEF_RETRY_TOUT 4

Default data-retransmission period (in seconds).

If data was not acknowledged during the time-frame defined by this value (in seconds) retransmission procedure will occur.

Definition at line 189 of file tcp_ip.h.

#define TCP_DEF_TOUT 120

Default idle timeout.

This period defines idle timeout in seconds - this feature allows TCP/IP stack to close the TCP connection if no data has been exchanged over it during this period of time. This relates ONLY to data. Empty keep-alive TCP packets are not included.

Definition at line 225 of file tcp_ip.h.

#define TCP_EVENT_ABORT 8

Connection aborted event.

Connection is, for some reason, aborted. This can happen for a number of reasons:

- Data retransmissions performed sufficient number of times but no acknowledgment was received
- No response for some time to keep-alive packets
- Remote host forcefully closed the connection for some reason
- Application invoked tcp_abort() function

Definition at line 418 of file tcp_ip.h.

#define TCP_EVENT_ACK 16

Data acknowledged event.

TCP/IP stack has received correct acknowledgment packet for the previously sent data and is informing the application about it. After this event, application can send new data packet to remote host.

Definition at line 428 of file tcp_ip.h.

#define TCP_EVENT_CLOSE 4

Connection closed event.

TCP connection was properly closed (either by calling tcp_close() by application or remote host initialized closing sequence).

Definition at line 405 of file tcp_ip.h.

#define TCP_EVENT_CONNECTED 2

Connection established event.

Applications' event listener is informed by this event that connection is established and that it may start sending/receiving data.

Definition at line 397 of file tcp_ip.h.

#define TCP_EVENT_CONREQ 1

Connection request event.

Connection request event is notified to TCP server applications' event listener when SYN packet is received for it's socket. Event listener can then, if it wants to, inspect IP addres and port number of the remote host, or some other internal parameters, to decide whether it should allow connection establishment or not. One of the following values must then be returned from the event listener:

- -1 - do not allow connection to be established. RST packet will be sent to remote host.
- -2 - do not respond to this particular SYN packet (keep quiet). This may be used if device is somehow busy and not yet ready to establish a connection, but doesn't wan't to forcefully reject the connection with a RST packet.
- 1 - allow connection to be established

Definition at line 387 of file tcp_ip.h.

#define TCP_EVENT_DATA 64

Data arrival event.

TCP received some data from remote host and is informing application that it is available for reading from the Ethernet controller.

Definition at line 445 of file tcp_ip.h.

#define TCP_EVENT_REGENERATE 32

Regenerate data event.

Previously sent data packet was not acknowledged (or the acknowledgment packet did not arrive) so retransmission needs to be peformed. Application must resend the data that was sent in the previous packet.

Definition at line 437 of file tcp_ip.h.

#define TCP_FLAG_ACK 0x10

Definition at line 281 of file tcp_ip.h.

#define TCP_FLAG_FIN 0x01

Definition at line 285 of file tcp_ip.h.

#define TCP_FLAG_PUSH 0x08

Definition at line 282 of file tcp_ip.h.

#define TCP_FLAG_RESET 0x04

Definition at line 283 of file tcp_ip.h.

#define TCP_FLAG_SYN 0x02

Definition at line 284 of file tcp_ip.h.

#define TCP_HALF_SEQ_SPACE 0x0000FFFF

Definition at line 227 of file tcp_ip.h.

#define TCP_INIT_RETRY_TOUT 1

Initial retransmission period (in seconds).

Initial retransmission is made a little faster, which helps with connection establishment if ARP cache didn't contain remote IP address.

Definition at line 198 of file tcp_ip.h.

#define TCP_INTFLAGS_CLOSEPENDING 0x01

Definition at line 289 of file tcp_ip.h.

#define TCP_PORTS_END 1023

Define reserved-ports space.

TCP socket numbers will only be assigned to be lower than this number.

Definition at line 100 of file tcp_ip.h.

#define TCP_STATE_CLOSED 3

Entry allocated, socket still closed

Definition at line 335 of file tcp_ip.h.

#define TCP_STATE_CLOSING 9

Received FIN independently of our FIN

Definition at line 353 of file tcp_ip.h.

#define TCP_STATE_CONNECTED 12

Connection established and data flowing freely to both sides :-)

Definition at line 362 of file tcp_ip.h.

#define TCP_STATE_FINW1 7

User issued tcp_close request issued so FIN packet was sent

Definition at line 346 of file tcp_ip.h.

#define TCP_STATE_FINW2 8

Received ACK of our FIN, now waiting for other side to send FIN

Definition at line 349 of file tcp_ip.h.

#define TCP_STATE_FREE 1

Entry is free and unused

Definition at line 333 of file tcp_ip.h.

#define TCP_STATE_LAST_ACK 10

Waiting for last ACK packet as a response to our FIN

Definition at line 356 of file tcp_ip.h.

#define TCP_STATE_LISTENING 4

Socket in listening state, waiting for incoming connections

Definition at line 336 of file tcp_ip.h.

#define TCP_STATE_RESERVED 2

Entry is reserved for use

Definition at line 334 of file tcp_ip.h.

#define TCP_STATE_SYN_RECEIVED 5

SYN packet received (either first SYN packet or response to SYN that we have previously sent)

Definition at line 339 of file tcp_ip.h.

#define TCP_STATE_SYN_SENT 6

SYN packet sent as an attempt to establish a connection

Definition at line 343 of file tcp_ip.h.

#define TCP_STATE_TIMED_WAIT 11

Waiting for 2MSL to prevent erroneous connection duplication

Definition at line 359 of file tcp_ip.h.

#define TCP_SYN_RETRY_TOUT 2

Retranmission period for SYN packet.

Controls SYN packet (segment) retransmit period.

Definition at line 206 of file tcp_ip.h.

#define TCP_TOS_NORMAL 0

Defines normal type of service for TCP socket.

This defines normal (and for now the only one implemented) type of service for the TCP socket.

Definition at line 214 of file tcp_ip.h.

#define TCP_TYPE_CLIENT 0x02

TCP socket represents a client application.

If TCP socket entry is of client type, application using it can establish connection through it towards other Internet hosts but can not accept any incoming connections on the port (execute tcp_listen() on it that is).

Definition at line 320 of file tcp_ip.h.

#define TCP_TYPE_CLIENT_SERVER 0x03

TCP socket can act as client or as server.

If TCP socket entry is of this type, application using it can both listen on a given socket or establish connection towards an outside host.

Definition at line 329 of file tcp_ip.h.

#define TCP_TYPE_NONE 0x00

TCP socket is nor a client nor a server.

If TCP socket entry is of this type it can not be used for anything. This may only be used for testing, debugging, etc. purposes or if application is not sure what it wants to be it can reserve a TCP socket by getting a socket of type TCP_TYPE_NONE.

Definition at line 300 of file tcp_ip.h.

#define TCP_TYPE_SERVER 0x01

TCP socket represents a server application.

If TCP socket entry is of server type, application using it can only listen on a given socket for incoming connections. No connections can be opened towards some outside host unless the outside host initiates the connection.

Definition at line 310 of file tcp_ip.h.

#define UDP_APP_OFFSET UDP_HLEN

Transmit buffer offset for UDP applications.

This value defines offset that UDP applications must use when writing to transmit buffer. This many bytes will be used **before** the first byte of applications data in the transmit buffer to store UDP header.

Definition at line 471 of file tcp_ip.h.

#define UDP_EVENT_DATA 64

Only UDP event notified to UDP socket event listener.

For now, this is the only UDP event that is notified to the UDP sockets' event listener.

Definition at line 276 of file tcp_ip.h.

#define UDP_HLEN 8

UDP Header Length

Definition at line 144 of file tcp_ip.h.

#define UDP_OPT_CHECK_CS 2

Enable checksum checking for received UDP packets.

When this option is enabled, checksum is checked for all received UDP packets to check for transmission errors.

Definition at line 140 of file tcp_ip.h.

#define UDP_OPT_NONE 0

Disable checksum calculation for UDP socket.

By choosing only this option for UDP socket, checksum calculation will be disabled for both incoming/outgoing UDP packets. This will make UDP extremely fast, but also more prone to errors. This is usually not a big limitation considering that checksum is not so good in error-detection anyway.

Definition at line 122 of file tcp_ip.h.

#define UDP_OPT_SEND_CS 1

Enable checksum calculation for outgoing UDP packets.

When only this option is chosen, checksum is calculated for outgoing UDP packets. This is sometimes neccessary since certain applications can disable reception of UDP packets without a calculated checksum.

Definition at line 132 of file tcp_ip.h.

#define UDP_PORTS_END 1023

Define reserved-ports space.

UDP socket numbers will only be assigned to be lower than this number.

Definition at line 109 of file tcp_ip.h.

#define UDP_SEND_MTU ETH_MTU - ETH_HEADER_LEN - UDP_HLEN - IP_MAX_HLEN

Definition at line 142 of file tcp_ip.h.

#define UDP_STATE_CLOSED 2

Defines that UDP socket is allocated but closed.

Entries of ucb type that have this as their state value were allocated by udp_getsocket() but they are in the closed state, so no data can be received/sent through the socket.

Definition at line 258 of file tcp_ip.h.

#define UDP_STATE_FREE 1

Defines that UDP socket is free and available.

If an entry of ucb type has this as a state value, then it is free and can be allocated by the udp_getsocket().

Definition at line 249 of file tcp_ip.h.

#define UDP_STATE_OPENED 3

Defines that UDP socket is allocated and opened.

Corresponding UDP socket was allocated and opened so data can be transmitted/received through it.

Definition at line 266 of file tcp_ip.h.

Function Documentation

INT16 process_icmp_in (struct ip_frame * *frame*, UINT16 *len*)

Process recieved ICMP datagram.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

08.07.2002

Parameters:

frame - pointer to received IP frame structure
len - length of the received IP datagram (in bytes)

Returns:

- -1 - packet not OK (not proper ICMP or not ICMP at all)
- >=0 - packet OK

Invoke process_icmp_in whenever IP datagram containing ICMP message is detected (see main_demo.c for example main loop implementing this).

This function simply checks correctnes of received ICMP message and send ICMP replies when requested.

Definition at line 97 of file icmp.c.

INT16 process_tcp_in (struct ip_frame * *frame*, UINT16 *len*)

Check and process the received TCP frame.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.07.2002

Parameters:

frame pointer to received ip_frame structure
len length of data contained in IP datagram (in bytes)

Returns:

- -1 - Error (packet not OK, or not TCP,or something else)
- >0 - Packet OK

Invoke this function to process received TCP frames. See main_demo.c for an example on how to accomplish this.

Definition at line 1118 of file tcp.c.

INT16 process_tcp_out (INT8 *sockethandle*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Create and send TCP packet.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

16.07.2002

Parameters:

sockethandle handle to processed socket
buf pointer to data buffer (where TCP header will be stored)
blen buffer length in bytes
dlen length of data in bytes

Returns:

- -1 - Error
- >0 - Packet OK

Based on data supplied as function parameters and data stored in socket's tcb, TCP header is created in buffer, checksum is calculated and packet is forwarded to lower layers (IP).

Definition at line 1982 of file tcp.c.

INT16 process_udp_in (struct ip_frame * *frame*, UINT16 *len*)

Process received UDP frame.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

15.07.2002

Parameters:

frame pointer to received IP frame structure
len length of data in bytes

Returns:

- -1 - Error (packet not UDP, header errror or no socket for it)
- >0 - Packet OK

Invoke this function to process received UDP frames. See main_demo.c for an example on how to accomplish this.

Definition at line 515 of file udp.c.

INT8 tcp_abort (INT8 *sochandle*)

Reset connection and place socket to closed state.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be aborted

Returns:

- -1 - error
- >=0 - OK (value represents handle to aborted socket)

Use this function in cases when TCP connection must be immediately closed. Note that the preffered (more elegant) way of closing the TCP connection is to invoke tcp_close() which starts a proper closing procedure. tcp_abort should be used only in cases when it is really necessary to immediately and quickly close the connection.

Definition at line 616 of file tcp.c.

UINT8 tcp_check_cs (struct ip_frame * *ipframe*, UINT16 *len*)

Check if TCP checksum check's out.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

16.07.2002

Parameters:

ipframe pointer to IP frame that carried TCP message
len length of TCP portion

Returns:

- 0 - checksum corrupted
- 1 - checksum OK

Function recalculates TCP checksum (pseudoheader+header+data) and compares it to received checksum to see if everything is OK or there is a problem with the checksum.

Definition at line 2425 of file tcp.c.

INT16 tcp_checksend (INT8 *sochandle*)

Checks if it's possible to send data using given socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

23.07.2002

Parameters:

sochandle handle to the socket to be inspected

Returns:

- -1 - not possible to send over a socket (previously sent data is still not akcnowledged)
- >0 - it is possible to send data over a socket

Invoke this function to get information whether it is possible to send data or not. This may, sometimes, be preffered way of getting this type of information to waiting for TCP_EVENT_ACK in event_listener function.

Definition at line 576 of file tcp.c.

INT8 tcp_close (INT8 *sochandle*)

Initiate TCP connection closing procedure.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket on which TCP connection is to be closed

Returns:

- -2 - there is unacked data on this socket. Try again later.
- -1 - Error
- >=0 - OK (connection closing procedure started. Handle to socket returned)

Invoke this function to start connetion closing procedure over a given socket. Note that connection is not immediately closed. It may take some time for that to happen. Event_listener function will be invoked with appropriate event when that really happens.

Definition at line 447 of file tcp.c.

INT8 tcp_connect (INT8 *sochandle*, UINT32 *ip*, UINT16 *rport*, UINT16 *myport*)

Initialize connection establishment towards remote IP&port.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be used for connection establishment
ip remote IP address to connect to
rport remote port number to connect to
myport local port to use for connection. This value can be specified directly or, if a value of 0 is given, TCP module will determine local TCP port automatically.

Returns:

- -1 - Error
- >=0 - OK (Connection establishment procedure started. Socket handle returned.)

Invoke this function to start connection establishment procedure towards remote host over some socket. This is only possible if socket was defined as either TCP_TYPE_CLIENT or TCP_TYPE_CLIENT_SERVER. Function will make some basic checks and if everything is OK, corresponding tcb socket entry will be initialized and connection procedure started.

Definition at line 283 of file tcp.c.

UINT16 tcp_getfreeport (void)

Returns next free (not used) local port number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Returns:

- 0 - no free ports!
- >0 - free local TCP port number

Function attempts to find new local port number that can be used to establish a connection.

Definition at line 2369 of file tcp.c.

INT8 tcp_getsocket (UINT8 *soctype*, UINT8 *tos*, UINT16 *tout*, INT32(* *listener*)(INT8, UINT8, UINT32, UINT32))

Allocate a free socket in TCP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

soctype type of socket wanted. Can take one of the following values:

- TCP_TYPE_NONE

- TCP_TYPE_SERVER
- TCP_TYPE_CLIENT
- TCP_TYPE_CLIENT_SERVER

tos type of service for socket. For now only TCP_TOS_NORMAL.
tout Timeout of socket in seconds. Defines after how many seconds of inactivity (application not sending and/or receiving any data over TCP connection) will the TCP socket automatically be closed.
listener pointer to callback function that will be invoked by the TCP/IP stack to inform socket application of certain events. See tcpc_demo_eventlistener() and tcps_demo_eventlistener() for more information on events and possible actions.

Returns:

- -1 - Error getting requested socket
- >=0 - Handle to reserved socket

Invoke this function to try to obtain a free socket from TCP socket pool. Function returns a handle to the free socket that is later used for accessing the allocated socket (opening, connecting, sending data, closing, aborting, etc.).

Definition at line 79 of file tcp.c.

INT8 tcp_getstate (INT8 *sochandle*)

Get current state of the socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to the socket to be queried

Returns:

- -1 - Error
- >0 - Socket state

Use this function for querying socket state. This is usually not needed directly, but could be usefull for some special purposes.

Definition at line 540 of file tcp.c.

INT8 tcp_init (void)

Initialize TCP module.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Returns:

- -1 - error
- >0 - number of sockets initialized

Warning:

- This function **must** be invoked at startup before any other TCP functions are invoked.

This function initializes all sockets and corresponding tcbs to known state. Timers are also allocated for each socket and everything is brought to a predefined state.

Definition at line 1028 of file tcp.c.

UINT32 tcp_initseq (void)

Get and return initial sequence number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

17.07.2002

Returns:

UINT32 number containing initial sequence number to be used

This function returns initial sequence number to be used in a TCP connection. For now, initial sequence number is selected based on base_timer value, which should be solid enough choice.

Definition at line 2203 of file tcp.c.

INT8 tcp_listen (INT8 *sochandle*, UINT16 *port*)

Put TCP socket to listen on a given port.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be placed to listen state
port TCP port number on which it should listen

Returns:

- -1 - Error
- >=0 - OK (Socket put to listening state. Handle to socket returned)

This function will attempt to put socket to listening state. This is only possible if socket was defined as either TCP_TYPE_SERVER or TCP_TYPE_CLIENT_SERVER. If basic correctness checks pass, socket is put to listening mode and corresponding tcb entry is initialized.

Definition at line 202 of file tcp.c.

INT8 tcp_mapsocket (struct ip_frame * *ipframe*, struct tcp_frame * *tcpframe*)

Try to match received TCP packet to a socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

12.07.2002

Parameters:

ipframe pointer to received IP frame
tcpframe pointer to received TCP frame to be mapped

Returns:

- -1 - Error (no resources or no socket found)
- >=0 - Handle to mapped socket

Function iterates through socket table trying to find a socket for whom this TCP packet is intended.

Definition at line 2226 of file tcp.c.

void tcp_newstate (struct tcb * *soc*, UINT8 *nstate*)

Change TCP socket state and reinitialize timers.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2002

Parameters:

soc pointer to socket structure we're working with
nstate new socket state

This function is used for every state-change that occurs in the TCP sockets so as to provide correct timers/retransmittions that ensure TCP connection is lasting.

Definition at line 2313 of file tcp.c.

void tcp_poll (void)

Poll TCP sockets periodically.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.07.2002

Warning:

- This function **must be** invoked periodically from the main loop. See main_demo.c for an example.

This function checks all TCP sockets and performs various actions if timeouts occur. What kind of action is performed is defined by the state of the TCP socket.

Definition at line 682 of file tcp.c.

INT8 tcp_releasesocket (INT8 *sochandle*)

Release a TCP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

21.07.2002

Parameters:

sochandle handle to socket to be released

Returns:

- -1 - Error releasing the socket (Wrong socket handle or socket not in proper state to be released)
- >=0 - handle of the released socket (can not be used any more untill allocated again with tcp_getsocket()).

Once the application does not need the TCP socket any more it can invoke this function in order to release it. This is usefull if there is a very limited number of sockets (in order to save some memory) shared among several applications.

Definition at line 146 of file tcp.c.

INT16 tcp_send (INT8 *sockethandle*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Send user data over TCP using given TCP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

25.07.2002

Parameters:

sockethandle handle to TCP socket to be used for sending data
buf pointer to data buffer (start of user data)
blen buffer length in bytes (without space reserved at the beginning of buffer for headers)
dlen length of user data to be sent (in bytes)

Returns:

- -1 - Error
- >0 - OK (number represents number of bytes actually sent)

Warning:

- *buf* parameter is a pointer to data to be sent in user buffer. But note that there **MUST** be sufficient free buffer space before that data for TCP header (of MIN_TCP_HLEN size).

Invoke this function to initiate data sending over TCP connection established over a TCP socket. Since data is not buffered (in order to reduce RAM memory consumption) new data can not be sent until data that was previously sent is acknowledged. So, application knows when it can send new data either by:

- waiting for TCP_EVENT_ACK in event_listener function
- invoking tcp_check_send() function to check if it is possible to send data

Definition at line 381 of file tcp.c.

void tcp_sendcontrol (INT8 *sockethandle*)

Send a TCP control packet (no data).

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

17.07.2002

Parameters:

sockethandle handle to socket

This function is used to initiate sending of a control (no data) TCP packet. Important thing in these packets are the flags and sequence numbers they carry.

Definition at line 2113 of file tcp.c.

void tcp_sendreset (struct tcp_frame * *frame*, UINT32 *remip*)

Send a reset (RST) packet to remote host.

Author:

- Jari Lahti (jari.lahti@violasystems.com)
- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

20.08.2002

Parameters:

frame pointer to received TCP packet
remip remote IP address of packet

Uses socket NO_OF_TCPSOCKETS to send a RESET packet to peer. This function is used when we are establishing connection but we receive something else than SYN or SYN+ACK when it's possible that the peer has still old connection on which needs to be resetted without canceling the connection establishment on process.

Definition at line 2152 of file tcp.c.

INT8 udp_close (INT8 *sochandle*)

Close given socket for communication.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to socket to be closed

Returns:

- -1 - Error
- >=0 - handle to closed socket

Closes a given socket in order to disable further communication over it.

Definition at line 319 of file udp.c.

UINT16 udp_getfreeport (void)

Returns next free (not used) local port number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.10.2002

Returns:

- 0 - no free ports!
- >0 - free local TCP port number

Function attempts to find new local port number that can be used to establish a connection.

Definition at line 654 of file udp.c.

INT8 udp_getsocket (UINT8 *tos*, INT32(* *listener*)(INT8, UINT8, UINT32, UINT16, UINT16, UINT16), UINT8 *opts*)

Allocate a free socket in UDP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

tos type of service for socket. For now nothing implemented so 0.
listener pointer to callback function that will be invoked by the TCP/IP stack to inform socket application of #UDP_DATA_ARRIVAL event (for now only this, in future maybe others!)
opts Options for checksum generation & inspection. Can be one of the following:

- UDP_OPT_NONE
- UDP_OPT_SEND_CS
- UDP_OPT_CHECK_CS
- UDP_OPT_SEND_CS | UDP_OPT_CHECK_CS

Returns:

- -1 - Error
- >=0 - Handle to reserved socket

Invoke this function to try to obtain a free socket from UDP socket pool. Function returns a handle to the free socket that is later used for accessing the allocated socket.

Definition at line 173 of file udp.c.

INT8 udp_init (void)

Initialize UDP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Returns:

- -1 - Error
- >0 - Number of UDP sockets initialized

Warning:

- This function **must** be invoked before any other UDP-related function is called

This function initializes UDP socket pool to get everything into a known state at startup.

Definition at line 122 of file udp.c.

INT8 udp_open (INT8 *sochandle*, UINT16 *locport*)

Open a given UDP socket for communication.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to socket to be opened
locport local port number

Returns:

- -1 - Error
- >=0 - Handle to opened socket

This function binds local port to given UDP socket and opens the socket (virtually) in order to enable communication.

Definition at line 276 of file udp.c.

INT8 udp_releasesocket (INT8 *sochandle*)

Release a given socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle of UDP socket to be released

Returns:

- -1 - error
- >=0 - OK (returns handle to release socket)

This function releases UDP socket. This means that the socket entry is marked as free and all of the ucb fields are initialized to default values.

Definition at line 235 of file udp.c.

INT16 udp_send (INT8 *sochandle*, UINT32 *remip*, UINT16 *remport*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Send data to remote host using given UDP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to UDP socket to use
remip remote IP address to which data should be sent
remport remote port number to which data should be sent
buf pointer to data buffer (start of user data)
blen buffer length in bytes (without space reserved at the beginning of buffer for headers)
dlen length of user data to be sent (in bytes)

Returns:

- -1 - Error (general error, e.g. parameters)
- -2 - ARP or lower layer not ready, try again later
- -3 - Socket closed or invalid local port
- >0 - OK (number represents number of bytes actually sent)

Warning:

- *buf* parameter is a pointer to data to be sent in user buffer. But note that there **MUST** be sufficient free buffer space before that data for UDP header (of UDP_HLEN size).

Use this function to send data over an already opened UDP socket.

Definition at line 368 of file udp.c.

---

tftps.c File Reference

Detailed Description

OpenTCP TFTP server implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:
1.0

Date:
7.10.2002

Bug:

Warning:

Todo:

- Offer callback functions for TFTP server

OpenTCP implementation of TFTP server application. For interface functions declarations see tftps.h.
Definition in file tftps.c.

```
#include "datatypes.h"
#include "globalvariables.h"
#include "system.h"
#include "timers.h"
#include "tcp_ip.h"
```

```
#include "tftps.h"
```

Defines

- #define TFTPS_STATE_ENABLED 1
- #define TFTPS_STATE_CONNECTED 2
- #define TFTPS_NOTDEFINED 0
- #define TFTPS_ACCESSVIOLATION 2
- #define TFTPS_ILLEGALOPERATION 4
- #define TFTPS_OPCODE_WRQ 2
- #define TFTPS_OPCODE_DATA 3
- #define TFTPS_OPCODE_ACK 4
- #define TFTPS_OPCODE_ERROR 5

Functions

- INT8 tftps_init (void)
  Initializes TFTP server.

- void tftps_run (void)
- INT32 tftps_eventlistener (INT8 cbhandle, UINT8 event, UINT32 remip, UINT16 remport, UINT16 bufindex, UINT16 dlen)
- void tftps_sendack (void)
- void tftps_senderror (UINT8 errno)
- void tftps_deletesocket (void)

Variables

- UINT8 tftpsapp_init = 0
- struct {
- UINT8 **state**
- INT8 **sochandle**
- UINT16 **tmrhandle**
- UINT32 **remip**
- UINT16 **remport**
- UINT16 **blocknumber**
- UINT32 **bytecount**
- UINT8 **retries**
- } tftps
  TFTP server state information.

Define Documentation

#define TFTPS_ACCESSVIOLATION 2

Definition at line 90 of file tftps.c.

#define TFTPS_ILLEGALOPERATION 4

Definition at line 91 of file tftps.c.

#define TFTPS_NOTDEFINED 0

Definition at line 89 of file tftps.c.

#define TFTPS_OPCODE_ACK 4

Definition at line 98 of file tftps.c.

#define TFTPS_OPCODE_DATA 3

Definition at line 97 of file tftps.c.

#define TFTPS_OPCODE_ERROR 5

Definition at line 99 of file tftps.c.

#define TFTPS_OPCODE_WRQ 2

Definition at line 96 of file tftps.c.

#define TFTPS_STATE_CONNECTED 2

Definition at line 83 of file tftps.c.

#define TFTPS_STATE_ENABLED 1

Definition at line 82 of file tftps.c.

Function Documentation

void tftps_deletesocket (void)

Definition at line 419 of file tftps.c.

INT32 tftps_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *remip*, UINT16 *remport*, UINT16 *bufindex*, UINT16 *dlen*)

Definition at line 195 of file tftps.c.

INT8 tftps_init (void)

Initializes TFTP server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.07.2002

Returns:

- -1 - Error
- >=0 - OK, server intialized

This function should be called before the TFTP Server application is used to set the operating parameters of it

Definition at line 135 of file tftps.c.

void tftps_run (void)

Definition at line 181 of file tftps.c.

void tftps_sendack (void)

Definition at line 386 of file tftps.c.

void tftps_senderror (UINT8 *errno*)

Definition at line 403 of file tftps.c.

Variable Documentation

UINT16 blocknumber

Definition at line 116 of file tftps.c.

UINT32 bytecount

Definition at line 117 of file tftps.c.

UINT32 remip

Definition at line 114 of file tftps.c.

UINT16 remport

Definition at line 115 of file tftps.c.

UINT8 retries

Definition at line 118 of file tftps.c.

INT8 sochandle

Definition at line 112 of file tftps.c.

UINT8 state

Definition at line 111 of file tftps.c.

struct { ... } tftps

TFTP server state information.

tftps variable holds various information that the tftp server needs for proper operation. These include server state, socket handle, timer handle, remote IP address and port number of the host we're communicating with, retransmit counter and TFTP block number.

UINT8 tftpsapp_init = 0

Defines whether tftps_init has already been invoked or not

Definition at line 77 of file tftps.c.

UINT16 tmrhandle

Definition at line 113 of file tftps.c.

---

tftps.h File Reference

Detailed Description

OpenTCP TFTP server interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

7.10.2002

OpenTCP TFTP server protocol function declarations, constants, etc.
Definition in file tftps.h.

```
#include "datatypes.h"
```

Defines

- #define TFTPS_SERVERPORT 69
  Default server port for TFTP server.

- #define TFTPS_FILENAME_MAXLEN 20
  Maximum filename allowed by the TFTP server.

- #define TFTPS_DEF_RETRIES 4
  Default number of retries of TFTP server.

- #define TFTPS_TIMEOUT 20
  Timeout (in seconds) after which socket is deleted.

Functions

- INT8 tftps_init (void)
  Initializes TFTP server.

- void tftps_run (void)
- INT32 tftps_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)
- void tftps_sendack (void)
- void tftps_senderror (UINT8)
- void tftps_deletesocket (void)

Define Documentation

#define TFTPS_DEF_RETRIES 4

Default number of retries of TFTP server.

Number of retries of resending the data before aborting.

Definition at line 90 of file tftps.h.

#define TFTPS_FILENAME_MAXLEN 20

Maximum filename allowed by the TFTP server.

Maximum filename-length TFTP server is ready to process.

Definition at line 83 of file tftps.h.

#define TFTPS_SERVERPORT 69

Default server port for TFTP server.

TFTP server will use this UDP port to listen for incoming traffic.

Definition at line 76 of file tftps.h.

#define TFTPS_TIMEOUT 20

Timeout (in seconds) after which socket is deleted.

Definition at line 97 of file tftps.h.

Function Documentation

void tftps_deletesocket (void)

Definition at line 419 of file tftps.c.

INT32 tftps_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)

Definition at line 195 of file tftps.c.

INT8 tftps_init (void)

Initializes TFTP server.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.07.2002

Returns:

- -1 - Error
- >=0 - OK, server intialized

This function should be called before the TFTP Server application is used to set the operating parameters of it

Definition at line 135 of file tftps.c.

void tftps_run (void)

Definition at line 181 of file tftps.c.

void tftps_sendack (void)

Definition at line 386 of file tftps.c.

void tftps_senderror (UINT8)

Definition at line 403 of file tftps.c.

timers.c File Reference

```
#include "debug.h"
#include "datatypes.h"
#include "timers.h"
#include "system.h"
```

Functions

- void timer_pool_init (void)
  Initialize timer pool.

- UINT8 get_timer (void)
  Obtain a timer from timer pool.

- void free_timer (UINT8 nbr)
  Release timer back to free timer pool.

- void init_timer (UINT8 nbr, UINT32 tout)
  Initialize timer to a given time-out value.

- UINT32 check_timer (UINT8 nbr)
  Return the value of a given timer.

- void decrement_timers (void)
  Decrement all timers' values by one.

Variables

- struct {
- UINT32 **value**
- UINT8 **free**
- } timer_pool [NUMTIMERS]
  Timer pool used to keep information about available timers.

Function Documentation

UINT32 check_timer (UINT8 *nbr*)

Return the value of a given timer.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr timer handle who's value is to be returned

Returns:

timer value

Warning:

- Interrupts are not disabled when fetching the value, therefore returned value possibly has an error component +/- TIMERTIC.

Function simply returns timer value of a given timer. No checks are made in order to make the function as fast as possible.

Definition at line 160 of file timers.c.

void decrement_timers (void)

Decrement all timers' values by one.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

18.07.2001

Invoke this function from timer interrupt to decrement timer counter values

Definition at line 175 of file timers.c.

void free_timer (UINT8 *nbr*)

Release timer back to free timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr handle to timer beeing released

This function releases the timer who's handle is supplied as parameter. Use this when timer is not needed any more and other applications might use it.

Definition at line 96 of file timers.c.

UINT8 get_timer (void)

Obtain a timer from timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Returns:

Handle to a free timer

Warning:

- Timers are considered to be critical resources, so if there is no available timer and get_timer is invoked, system will reset.

Invoke this function to obtain a free timer (it's handle that is) from the timer pool.

Definition at line 58 of file timers.c.

void init_timer (UINT8 *nbr*, UINT32 *tout*)

Initialize timer to a given time-out value.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr handle of timer who's value we're setting
tout time-out value to set for this timer

Invoke this function to set timeout value for a timer with a given handle.

TIMERTIC defines how quickly the timers' values are decremented so is it to initialize timers to correct timeouts.

Definition at line 121 of file timers.c.

void timer_pool_init (void)

Initialize timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Warning:

- This function **must** be invoked at startup before any other timer function is used.

This function resets all timer counter to zero and initializes all timers to available (free) state.

Definition at line 32 of file timers.c.

Variable Documentation

UINT8 free

Definition at line 16 of file timers.c.

struct { ... } timer_pool[NUMTIMERS]

Timer pool used to keep information about available timers.

This timer pool is extensively used by most of the modules of the OpenTCP project. All timers that are used are allocated from this pool. Maximum number of timers that can be used at any given time is defined by the NUMTIMERS define.

UINT32 value

Definition at line 15 of file timers.c.

---

timers.h File Reference

Detailed Description

OpenTCP timers interface file.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

18.7.2002

OpenTCP timers function declarations, constants, etc.
Definition in file timers.h.

```
#include "datatypes.h"
```

Defines

- #define NUMTIMERS 55
  Number of timers available in the system.

- #define TIMERTIC 100
  Frequency of timer interrupt.

Functions

- UINT8 get_timer (void)
  Obtain a timer from timer pool.

- void free_timer (UINT8)
  Release timer back to free timer pool.

- void init_timer (UINT8, UINT32)
  Initialize timer to a given time-out value.

- void timer_pool_init (void)
  Initialize timer pool.

- UINT32 check_timer (UINT8)
  Return the value of a given timer.

- void decrement_timers (void)
  Decrement all timers' values by one.

Define Documentation

#define NUMTIMERS 55

Number of timers available in the system.

Change this number to change the size of the timer pool.

Definition at line 77 of file timers.h.

#define TIMERTIC 100

Frequency of timer interrupt.

This value should be changed to reflect the frequency in which timers are decremented. Standard value for this is 100.

Definition at line 86 of file timers.h.

Function Documentation

UINT32 check_timer (UINT8 *nbr*)

Return the value of a given timer.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr timer handle who's value is to be returned

Returns:

timer value

Warning:

- Interrupts are not disabled when fetching the value, therefore returned value possibly has an error component +/- TIMERTIC.

Function simply returns timer value of a given timer. No checks are made in order to make the function as fast as possible.

Definition at line 160 of file timers.c.

void decrement_timers (void)

Decrement all timers' values by one.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Date:

18.07.2001

Invoke this function from timer interrupt to decrement timer counter values

Definition at line 175 of file timers.c.

void free_timer (UINT8 *nbr*)

Release timer back to free timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr handle to timer beeing released

This function releases the timer who's handle is supplied as parameter. Use this when timer is not needed any more and other applications might use it.

Definition at line 96 of file timers.c.

UINT8 get_timer (void)

Obtain a timer from timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Returns:

Handle to a free timer

Warning:

- Timers are considered to be critical resources, so if there is no available timer and get_timer is invoked, system will reset.

Invoke this function to obtain a free timer (it's handle that is) from the timer pool.

Definition at line 58 of file timers.c.

void init_timer (UINT8 *nbr*, UINT32 *tout*)

Initialize timer to a given time-out value.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Parameters:

nbr handle of timer who's value we're setting
tout time-out value to set for this timer

Invoke this function to set timeout value for a timer with a given handle.

TIMERTIC defines how quickly the timers' values are decremented so is it to initialize timers to correct timeouts.

Definition at line 121 of file timers.c.

void timer_pool_init (void)

Initialize timer pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

18.07.2001

Warning:

- This function **must** be invoked at startup before any other timer function is used.

This function resets all timer counter to zero and initializes all timers to available (free) state.

Definition at line 32 of file timers.c.

---

udp.c File Reference

Detailed Description

OpenTCP UDP implementation.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Version:

1.0

Date:

15.7.2002

Bug:

Warning:

Todo:

- Send ICMP Destination Unreachable when receiving UDP packets to non-existent UDP ports.

OpenTCP UDP implementation. All functions necessary for UDP packet processing are present here. Note that only a small subset of these functions must be used for "normal" applications that are using the UDP for communciation. For function declarations and lots of other usefull stuff see tcp_ip.h.
For examples how to use UDP and write applications that communicate using UDP see main.c and udp_demo.c.
Definition in file udp.c.

```
#include "debug.h"
#include "datatypes.h"
#include "ethernet.h"
#include "ip.h"
#include "tcp_ip.h"
#include "system.h"
```

Functions

- INT8 udp_init (void)
  Initialize UDP socket pool.

- INT8 udp_getsocket (UINT8 tos, INT32(*listener)(INT8, UINT8, UINT32, UINT16, UINT16, UINT16), UINT8 opts)
  Allocate a free socket in UDP socket pool.

- INT8 udp_releasesocket (INT8 sochandle)
  Release a given socket.

- INT8 udp_open (INT8 sochandle, UINT16 locport)
  Open a given UDP socket for communication.

- INT8 udp_close (INT8 sochandle)
  Close given socket for communication.

- INT16 udp_send (INT8 sochandle, UINT32 remip, UINT16 remport, UINT8 *buf, UINT16 blen, UINT16 dlen)

Send data to remote host using given UDP socket.

- INT16 process_udp_in (struct ip_frame *frame, UINT16 len)
  Process received UDP frame.

- UINT16 udp_getfreeport (void)
  Returns next free (not used) local port number.

Variables

- ucb udp_socket [NO_OF_UDPSOCKETS]
  UDP table holding socket parameters for every UDP socket.

- udp_frame received_udp_packet
  Used for storing field information about the received UDP packet.

Function Documentation

INT16 process_udp_in (struct ip_frame * *frame*, UINT16 *len*)

Process received UDP frame.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

15.07.2002

Parameters:

frame pointer to received IP frame structure
len length of data in bytes

Returns:

- -1 - Error (packet not UDP, header errror or no socket for it)
- >0 - Packet OK

Invoke this function to process received UDP frames. See main_demo.c for an example on how to accomplish this.

Definition at line 515 of file udp.c.

INT8 udp_close (INT8 *sochandle*)

Close given socket for communication.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to socket to be closed

Returns:

- -1 - Error
- >=0 - handle to closed socket

Closes a given socket in order to disable further communication over it.

Definition at line 319 of file udp.c.

UINT16 udp_getfreeport (void)

Returns next free (not used) local port number.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

19.10.2002

Returns:

- 0 - no free ports!
- >0 - free local TCP port number

Function attempts to find new local port number that can be used to establish a connection.

Definition at line 654 of file udp.c.

INT8 udp_getsocket (UINT8 *tos*, INT32(* *listener*)(INT8, UINT8, UINT32, UINT16, UINT16, UINT16), UINT8 *opts*)

Allocate a free socket in UDP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

tos type of service for socket. For now nothing implemented so 0.
listener pointer to callback function that will be invoked by the TCP/IP stack to inform socket application of #UDP_DATA_ARRIVAL event (for now only this, in future maybe others!)
opts Options for checksum generation & inspection. Can be one of the following:

- UDP_OPT_NONE
- UDP_OPT_SEND_CS
- UDP_OPT_CHECK_CS
- UDP_OPT_SEND_CS | UDP_OPT_CHECK_CS

Returns:

- -1 - Error
- >=0 - Handle to reserved socket

Invoke this function to try to obtain a free socket from UDP socket pool. Function returns a handle to the free socket that is later used for accessing the allocated socket.

Definition at line 173 of file udp.c.

INT8 udp_init (void)

Initialize UDP socket pool.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Returns:

- -1 - Error

- >0 - Number of UDP sockets initialized

Warning:

- This function **must** be invoked before any other UDP-related function is called

This function initializes UDP socket pool to get everything into a known state at startup.

Definition at line 122 of file udp.c.

INT8 udp_open (INT8 *sochandle*, UINT16 *locport*)

Open a given UDP socket for communication.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to socket to be opened
locport local port number

Returns:

- -1 - Error
- >=0 - Handle to opened socket

This function binds local port to given UDP socket and opens the socket (virtually) in order to enable communication.

Definition at line 276 of file udp.c.

INT8 udp_releasesocket (INT8 *sochandle*)

Release a given socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle of UDP socket to be released

Returns:

- -1 - error
- >=0 - OK (returns handle to release socket)

This function releases UDP socket. This means that the socket entry is marked as free and all of the ucb fields are initialized to default values.

Definition at line 235 of file udp.c.

INT16 udp_send (INT8 *sochandle*, UINT32 *remip*, UINT16 *remport*, UINT8 * *buf*, UINT16 *blen*, UINT16 *dlen*)

Send data to remote host using given UDP socket.

Author:

- Jari Lahti (jari.lahti@violasystems.com)

Date:

26.07.2002

Parameters:

sochandle handle to UDP socket to use
remip remote IP address to which data should be sent
remport remote port number to which data should be sent
buf pointer to data buffer (start of user data)
blen buffer length in bytes (without space reserved at the beginning of buffer for headers)
dlen length of user data to be sent (in bytes)

Returns:

- -1 - Error (general error, e.g. parameters)
- -2 - ARP or lower layer not ready, try again later
- -3 - Socket closed or invalid local port
- >0 - OK (number represents number of bytes actually sent)

Warning:

- *buf* parameter is a pointer to data to be sent in user buffer. But note that there **MUST** be sufficient free buffer space before that data for UDP header (of UDP_HLEN size).

Use this function to send data over an already opened UDP socket.

Definition at line 368 of file udp.c.

Variable Documentation

struct udp_frame received_udp_packet

Used for storing field information about the received UDP packet.

Various fields from the received UDP packet are stored in this variable. See udp_frame definition for struct information.

Definition at line 101 of file udp.c.

struct ucb udp_socket[NO_OF_UDPSOCKETS]

UDP table holding socket parameters for every UDP socket.

UDP table is an array of ucb structures holding all of the necessary information about the state, listener, port numbers and other info about the UDP sockets opened. Number of UDP sockets that can be opened at any given time is defined by the NO_OF_UDPSOCKETS and may be changed in order to change the amount of used RAM memory. See ucb definition for more information about the structure itself.

Definition at line 94 of file udp.c.

---

udp_demo.c File Reference

Detailed Description

Demonstration of a possible scenario of writing UDP application.

Author:

- Vladan Jovanovic (vladan.jovanovic@violasystems.com)

Version:

1.0

Date:

10.10.2002

Bug:

Warning:

- This example is given for demonstration purposes only. It was not tested for correct operation.

Todo:

Blank UDP demo application showing UDP functions and how applications might use them.
Definition in file udp_demo.c.

```
#include "debug.h"
#include "datatypes.h"
#include "globalvariables.h"
#include "system.h"
#include "tcp_ip.h"
#include "udp_demo.h"
#include <string.h>
```

Defines

- #define UDP_DEMO_PORT 5000
- #define UDP_DEMO_RMTHOST_IP 0xC0A80169
- #define UDP_DEMO_RMTHOST_PRT 5001
- #define MSG_SIZE 20

Functions

- INT16 udp_demo_send (void)
- void udp_demo_init (void)
- void udp_demo_run (void)
- INT32 udp_demo_eventlistener (INT8 cbhandle, UINT8 event, UINT32 ipaddr, UINT16 port, UINT16 buffindex, UINT16 datalen)

Variables

- unsigned char data_buffer [64]
- INT8 udp_demo_soch
  Socket handle holder for this application.

- UINT8 udp_demo_senddata

Define Documentation

#define MSG_SIZE 20

Definition at line 225 of file udp_demo.c.

#define UDP_DEMO_PORT 5000

Port number on which we'll work

Definition at line 99 of file udp_demo.c.

#define UDP_DEMO_RMTHOST_IP 0xC0A80169

Remote IP address this application will send data to

Definition at line 101 of file udp_demo.c.

#define UDP_DEMO_RMTHOST_PRT 5001

Port number on remote server we'll send data to

Definition at line 103 of file udp_demo.c.

Function Documentation

INT32 udp_demo_eventlistener (INT8 *cbhandle*, UINT8 *event*, UINT32 *ipaddr*, UINT16 *port*, UINT16 *buffindex*, UINT16 *datalen*)

Definition at line 176 of file udp_demo.c.

void udp_demo_init (void)

Definition at line 109 of file udp_demo.c.

void udp_demo_run (void)

Definition at line 137 of file udp_demo.c.

INT16 udp_demo_send (void)

Definition at line 226 of file udp_demo.c.

Variable Documentation

unsigned char data_buffer[64] [static]

Definition at line 80 of file udp_demo.c.

UINT8 udp_demo_senddata

Used to trigger data sending

Definition at line 97 of file udp_demo.c.

INT8 udp_demo_soch

Socket handle holder for this application.

This variable holds the assigned socket handle. Note that this application will reserve one UDP socket immediately and will not release it. For saving resources, UDP sockets can also be allocated/deallocated dynamically.

Definition at line 95 of file udp_demo.c.

---

udp_demo.h File Reference

Functions

- void udp_demo_init (void)
- void udp_demo_run (void)
- INT32 udp_demo_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)

Function Documentation

INT32 udp_demo_eventlistener (INT8, UINT8, UINT32, UINT16, UINT16, UINT16)

Definition at line 176 of file udp_demo.c.

void udp_demo_init (void)

Definition at line 109 of file udp_demo.c.

void udp_demo_run (void)

Definition at line 137 of file udp_demo.c.

Vectors.c File Reference

Typedefs

- typedef void(* near)(void)

Functions

- void near _Startup (void)
- void near RealTimeInterrupt (void)
- void near emac_ec_isr (void)
  excess collisions ISR - ECIF — Excessive Collision Interrupt Flag
- void near emac_lc_isr (void)
  late collisions ISR - LCIF — Late Collision Interrupt Flag
- void near emac_b_rx_error_isr (void)
  Babbling Receive Error ISR.
- void near emac_rx_b_b_o_isr (void)
  RXAOIF — Receive Buffer B Overrun ISR.
- void near emac_rx_b_a_o_isr (void)
  RXAOIF — Receive Buffer A Overrun ISR.
- void near emac_rx_error_isr (void)
  Receive Error ISR.
- void near emac_mii_mtc_isr (void)
  Management Transfer Complete ISR - MMCIF — MII Interrupt Flag.
- void near emac_rx_fc_isr (void)

RX flow control ISR.

- void near emac_f_tx_c_isr (void)
  transmit complete ISR - TXCIF — Frame Transmission Complete Interrupt Flag

- void near emac_rx_b_b_c_isr (void)
  Valid Frame Reception to Receive Buffer B Complete ISR.

- void near emac_rx_b_a_c_isr (void)
  Valid Frame Reception to Receive Buffer A Complete ISR.

- void near ephy_isr (void)
  EPHY ISR - Type of EPHY interrupt detemined by MII read of PHY_REG_IR register.

- void near PortHInterrupt (void)
- __interrupt void software_trap (void)

Variables

- const tIsrFunc _vect[] xFF80

Typedef Documentation

typedef void(* near)(void)

Definition at line 43 of file Vectors.c.

Function Documentation

void near _Startup (void)

void near emac_b_rx_error_isr (void)

Babbling Receive Error ISR.

Definition at line 1171 of file ne64driver.c.

void near emac_ec_isr (void)

excess collisions ISR - ECIF — Excessive Collision Interrupt Flag

Definition at line 1292 of file ne64driver.c.

void near emac_f_tx_c_isr (void)

transmit complete ISR - TXCIF — Frame Transmission Complete Interrupt Flag

Definition at line 1306 of file ne64driver.c.

void near emac_lc_isr (void)

late collisions ISR - LCIF — Late Collision Interrupt Flag

Definition at line 1278 of file ne64driver.c.

void near emac_mii_mtc_isr (void)

Management Transfer Complete ISR - MMCIF — MII Interrupt Flag.

Definition at line 1268 of file ne64driver.c.

void near emac_rx_b_a_c_isr (void)

Valid Frame Reception to Receive Buffer A Complete ISR.

Definition at line 1226 of file ne64driver.c.

void near emac_rx_b_a_o_isr (void)

RXAOIF — Receive Buffer A Overrun ISR.

Definition at line 1204 of file ne64driver.c.

void near emac_rx_b_b_c_isr (void)

Valid Frame Reception to Receive Buffer B Complete ISR.

Definition at line 1246 of file ne64driver.c.

void near emac_rx_b_b_o_isr (void)

RXAOIF — Receive Buffer B Overrun ISR.

Definition at line 1215 of file ne64driver.c.

void near emac_rx_error_isr (void)

Receive Error ISR.

Definition at line 1193 of file ne64driver.c.

void near emac_rx_fc_isr (void)

RX flow control ISR.

Definition at line 1160 of file ne64driver.c.

void near ephy_isr (void)

EPHY ISR - Type of EPHY interrupt detemined by MII read of PHY_REG_IR register.

Definition at line 909 of file ne64driver.c.

void near PortHInterrupt (void)

Definition at line 192 of file main.c.

void near RealTimeInterrupt (void)

Definition at line 54 of file RTI.c.

__interrupt void software_trap (void)

Definition at line 37 of file Vectors.c.

Variable Documentation

const tIsrFunc _vect [] xFF80

Definition at line 44 of file Vectors.c.