

# net/mac.h File Reference

## Detailed Description

MAC Module for Modtronix TCP/IP Stack.

### Author:

[Modtronix Engineering](#)

### Dependencies:

**Compiler:** string.h, stacksk.h, helpers.h, mac.h

MPLAB C18 v2.10 or higher  
HITECH PICC-18 V8.35PL3 or higher

***IMPORTANT! For speed, disable the "Procedural abstraction" optimization when using the MPLAB C18 compiler!***

## Description

The MAC module contains the driver software for the RTL8019AS NIC chip. It provides functions for reading and writing data from and to it.

For a detailed description, see the Ethernet section of this document - in [Modules] [TCP/IP Stack] [TCP/IP Base Protocols].

## Configuration

The following defines are used to configure this module, and should be placed in the projdefs.h (or similar) file. For details, see Project Configuration. To configure the module, the required defines should be uncommmented, and the rest commented out.

```

//*****
//----- Mac Configuration -----
//*****
//When STACK_USE_FAST_NIC is defined, a bit longer, but faster code is
used.
#define STACK_USE_FAST_NIC
//When defined, the code will be compiled for optimal speed. If not
defined, code is defined for smallest size.
#define MAC_SPEED_OPTIMIZE
//STACK_DISABLES_INTS can be defined if interrupts are to be disabled
during the MAC access routines
//#define STACK_DISABLES_INTS
//Only valid for the SBC44EC board! If defined, port B6 and B7 will be
available for general purpose I/O.
//The Ethernet port will however be a bit slower.
#define NIC_ENABLE_B6B7
//NIC_DISABLE_INT0 can be defined if the MAC should NOT use INT0
(connected to PIC port RB0) for it's
//interrupt request status line. When defined, INT0 is tri-stated, and the
```

```

PIC port pin connected to
//it can be used as a general purpose user IO pin. The PIC port pin that
becomes available is:
// - For SBC44EC this has no affect - no PIC pins are connected to the
interrupt pins
// - For SBC45EC this has no affect - no PIC pins are connected to the
interrupt pins
// - For SBC65EC and SBC68EC this frees up PIC pin RB0.
#define NIC_DISABLE_INT0
//NIC_DISABLE_IOCHRDY can be defined if the MAC should NOT use the IOCHRDY
signal on the RTL8019AS chip. This
//will mean that an additional PIC pin will be available for general
purpose use. To use this port pin, the
//connection to the IOCHRDY signal on RTL8019AS must be broken. This can
be done via solder jumpers on certian
//SBC boards.
// - For SBC44EC PIC port pin B5 will be available for user IO. Solder
jumper SJ5 must be opened!
// - For SBC45EC PIC port pin A4 will be available for user IO. Solder
jumper SJ5 must be opened!
// - For SBC65EC and SBC68EC this frees up PIC pin RG4. This pin is
currently however not routed to an connectors
#define NIC_DISABLE_IOCHRDY
//Keep a count of CNTR1 - CNTR3 registers. This can be used for debug
purposes, and can be disabled for
//release code.
//#define MAC_CNTR1_3
//Use access RAM variables to optimize speed and code size. There are
only a limited amount of access RAM
//registers in the PIC processor. If they are not used by any other code
modules, this define should be enabled
//seeing that it will speed up the MAC module quite a bit. If they are not
available, an error message will be
//displayed during compilation.
//#define MAC_USE_ACCESSRAM
//This define must be define when using this MAC
#define MAC_RTL8019AS

```

## Data Structures

```

struct _MAC_ADDR

```

## Defines

```

#define INVALID_BUFFER    (0xff)
#define MAC_ARP           (0x6)
#define MAC_GETARR_ALL    (0x00)

```

```
#define MAC_GETARR_RETMASK (0xF0)
#define MAC_GETARR_TRM (0x20)
#define MAC_IP (0)
#define MAC_UNKNOWN (0x0ff)
```

## Typedefs

```
typedef
_MAC_ADDR MAC_ADDR
```

## Functions

```
void MACDiscardRx (void)
void MACDiscardTx (BUFFER buffer)
void MACFlush (void)
BYTE MACGet (void)
WORD MACGetArray (BYTE *val, WORD len)
WORD MACGetArrayChr (BYTE *val, BYTE len, BYTE chr)
WORD MACGetFreeRxSize (void)
BOOL MACGetHeader (MAC_ADDR *remote, BYTE *type)
WORD MACGetNICAddr (void)
BUFFER MACGetTxBuffer (BOOL HighPriority)
void MACInit (void)
BOOL MACIsLinked (void)
BOOL MACIsTxReady (BOOL HighPriority)
void MACPut (BYTE val)
void MACPutArray (BYTE *val, WORD len)
void MACPutHeader (MAC_ADDR *remote, BYTE type, WORD
dataLen)
void MACReserveTxBuffer (BUFFER buffer)
void MACSetRxBuffer (WORD offset)
void MACSetTxBuffer (BUFFER buffer, WORD offset)
void MACTask (void)
void NICReset (void)
```

---

## Define Documentation

```
#define
INVALID_BUFFER (0xff)
```

```
#define
MAC_ARP (0x6)
```

```
#define
MAC_GETARR_ALL (0x00)
TCPGetArrayChr return codes
```

```
#define
MAC_GETARR_RETMASK    (0xF0)
```

```
#define
MAC_GETARR_TRM        (0x20)
```

```
#define
MAC_IP                (0)
```

```
#define
MAC_UNKNOWN           (0x0ff)
```

---

## Typedef Documentation

```
typedef struct _MAC_ADDR MAC_ADDR
Structure for storing Ethernet address
```

---

## Function Documentation

```
void MACDiscardRx (void )
Discard the contents of the current RX buffer.
```

```
void MACDiscardTx (BUFFER buffer )
```

Discard the given transmit buffer

### Parameters:

*buffer* Buffer identifier, is a number from 0-255 identifying the buffer

```
void MACFlush (void )
Flush the MAC
```

```
BYTE MACGet (void )
```

Reads a single byte via Remote DMA from the current MAC Receive buffer. If the last byte of the RX Buffer was read, this function automatically updates the Remote DMA read address to the first page of the RX Buffer. See PreConditions for more info.

**Pre-Condition:**

Remote DMA address has to be set up prior to calling this function. The Remote DMA registers are NOT configured by this function, and simply continue reading from the current "Remote DMA Address". A function like MACGetHeader() can be called prior to this function to configure Remote DMA to read the next packet in RX Buffer (situated in Remote DMA RAM)

**Returns:**

Read Byte

```
WORD MACGetArray ( BYTE    val,
                  *
                  WORD    len
                  )
```

Reads the given amount of bytes via Remote DMA from the current MAC Receive buffer. If the end of the RX Buffer is reached, this function automatically rolls over to the first page of the RX Ring Buffer. See PreConditions below for more info.

**Pre-Condition:**

Remote DMA address has to be set up prior to calling this function. The Remote DMA registers are NOT configured by this function, and simply continue reading from the current "Remote DMA Address". A function like MACGetHeader() can be called prior to this function to configure Remote DMA to read the next packet in RX Buffer (situated in Remote DMA RAM)

**Parameters:**

*len* Length of array to be read  
*val* Buffer to read packet into

**Returns:**

Number of bytes read

```
WORD MACGetArrayChr ( BYTE    val
                    *
                    ,
                    BYTE    len
                    ,
                    BYTE    chr
                    )
```

Reads the given amount of bytes via Remote DMA from the current MAC Receive buffer, and copies them to the given destination buffer until:

- The given buffer for returning the read array is full
- The given Chr terminating character is found (is also copied to destination buffer) If the end of the RX Buffer is reached, this function automatically rolls over to the first page of the RX Ring Buffer. See PreConditions below for more info.

If the termination character is found, it is also copied to the destination

folder. In this case the returned value indicating the number of bytes read will also include this byte.

**Pre-Condition:**

Remote DMA address has to be set up prior to calling this function. The Remote DMA registers are NOT configured by this function, and simply continue reading from the current "Remote DMA Address". A function like MACGetHeader() can be called prior to this function to configure Remote DMA to read the next packet in RX Buffer (situated in Remote DMA RAM)

**Parameters:**

[out] *val* Buffer to read packet into. If NULL, nothing is written to the buffer.  
*len* Length of array to be read, maximum length is 255  
*chr* Character to look for, that will terminate read

**Returns:**

The LSB (Bits 0-7) gives number of bytes loaded into buffer, including the terminating character if found! The MSB (bit 8-15) is a return code. The return code in the MSB has the following meaning:

- MAC\_GETARR\_ALL: All requested bytes were read, buffer was filled. The LSB will contain the number of bytes read = original length we passed this function.
- MAC\_GETARR\_TRM: The given terminating character was found. The LSB will contain the number of bytes read, including the terminating character.

WORD MACGetFreeRxSize ( *void* )

This function returns total receive buffer size available for future data packets.

BOOL MACGetHeader ( MAC\_ADDR \* *remote* ,  
                    BYTE \* *type*  
                    )

Check if the MAC Receive Buffer has any received packets. Reads the following data from the next available packet in the RX buffer:

- The MAC 4 bytes RX packet header (status, next receive packet, length)
- The 14 byte Ethernet header

Once a data packet is fetched by calling MACGetHeader, the complete data packet must be fetched (using MACGet) and discarded (using MACDiscardRx). Users cannot call MACGetHeader multiple times to receive multiple packets and fetch data later on.

**Parameters:**

[out] *remote* Pointer to a MAC\_ADDR structure. This function will write the MAC address of the node who sent this packet to this structure  
[out] *type* Pointer to byte. This function will write the type of

header to this variable. Can be ETHER\_IP, ETHER\_ARP or  
MAC\_UNKNOWN

**Returns:**

True if RX Buffer contained a packet and it's header was read, False  
if not

WORD MACGetNICAddr (void )

Get the current Remote DMA address. This is the address set by the  
NICSetAddr register. It will however be incremented after each Remote DMA  
read. The MACGet() function for example does a remote DMA read.

**Returns:**

The current Remote DMA address.

BUFFER MACGetTxBuffer (BOOL *HighPriority* )

Get handle of current transmit buffer. Is a number from 0-254. The current  
transmit buffer will automatically be set to the next available buffer  
after a MACFlush() function.

**Parameters:**

*HighPriority* Not used for this (RTL8019AS) MAC. High priority  
messages are those that don't need to be acknowledged  
before being discarded (TCP control packets, all ICMP  
packets, all UDP packets, etc.)

**Returns:**

Handle of current transmit buffer. Is a number from 0-254, or  
INVALID\_BUFFER is nothing available.

void MACInit (void )

BOOL MACIsLinked (void )

Check if the MAC is linked

**Returns:**

TRUE if linked, else FALSE

BOOL MACIsTxReady (BOOL *HighPriority* )

Check if ready for next transmission.

**Parameters:**

*HighPriority* Not used for this implementation High priority messages  
are those that don't need to be acknowledged before being  
discarded (TCP control packets, all ICMP packets, all UDP  
packets, etc.)

**Returns:**

TRUE if transmit buffer is empty  
FALSE if transmit buffer is not empty

```
void MACPut ( BYTE  val  )
```

Write a single byte to the Remote DMA. The byte is written to the current Remote DMA address.

**Parameters:**

*val* Byte to write to Remote DMA

```
void  
MACPutArray ( BYTE * val,  
              WORD  len  
              )
```

Write given array of bytes to NIC's RAM

**Parameters:**

*val* Pointer to byte array of bytes that are to be written to the NIC's RAM  
*len* Length of array given in *val*

```
void MACPutHeader (MAC_ADDR * remote,  
                  BYTE      type,  
                  WORD      dataLen  
                  )
```

This function writes the MAC header (Ethernet header) part of a packet that has to be transmitted to the current TX buffer (page in NIC RAM). After this function, the data must still be written to the TX buffer. After both header and data has been written, bits are set to instruct NIC to transmit transmit buffer. This function does the following:

- Reset the NIC Remote DMA write pointer to the first byte of the current TX Buffer
- Write the given header to the current TX Buffer
- Set the NIC Remote DMA byte count to the given *len*. This configures the Remote DMA to receive the given number of bytes.

**Parameters:**

[out] *remote* Pointer to a MAC\_ADDR structure. This function will write the MAC address of the node who sent this packet to this structure  
*type* Type of header. Can be ETHER\_IP, ETHER\_ARP or MAC\_UNKNOWN  
*dataLen* Size of the Data of this Ethernet packet. A Ethernet packet consists of:

- 6 Bytes = Destination MAC address
- 6 Bytes = Source MAC address
- 2 Bytes = Type field, currently only IP or ARP
- n Bytes = Data. Minimum length of 46 bytes

The data will be written to the TX buffer following this command.

**Returns:**

True if header was read, False if not

```
void  
MACReserveTxBuffer ( BUFFER buffer )
```

This function reserves a given transmit buffer and marks it as unavailable. This function is useful for TCP layer where a message would be queued until it is correctly acknowledged by remote host.

**Parameters:**

*buffer* Buffer identifier, is a number from 0-255 identifying the buffer

```
void  
MACSetRxBuffer (WORD offset )
```

This function sets the access location for the active receive buffer, relative to the first byte following the Ethernet header. If the given offset is thus 0, the access location will be set to the first byte of whatever follows the Ethernet header, for example the IP packet or ARP packet. Users must make sure that the supplied offset does not go beyond current receive buffer content. If offset overruns the current receive buffer, all subsequent access to that buffer would yield invalid data.

**Parameters:**

*offs* Location (with respect to first byte following Ethernet header)  
*et* where next access is to occur.

```
void  
MACSetTxBuffer ( BUFFER buffer,  
                WORD offset  
                )
```

This function makes the given transmit buffer active, and sets it's access pointer to be:

- At the given offset after the Ethernet header
- In the given TX Buffer

So, if we pass 0 in as the offset, we will set the pointer to the first byte of after the Ethernet header. Users must make sure that the supplied offset does not go beyond current transmit buffer content. If offset overruns the current transmit buffer, all subsequent access to that buffer would yield invalid data.

**Parameters:**

*buffer* A transmit buffer where this access offset be applied  
*offset* Location (with respect to beginning of buffer) where next access is to occur

```
void MACTask ( void )  
Function for performing MAC related tasks
```

**Pre-Condition:**

: Must be called every couple of ms

```
void NICReset (void )  
Reset the NIC
```

---