

# Ethernet library version V3.5 with TCP/IP Stack MCU family PIC18F97J60 and ENC28J60 MikroPascal

## Available types:

- **IpHeader** : array[4] of byte;
- **MacHeader** : array[6] of byte;

## Available functions and procedures :

(AddrPacket is address of received packet, TXSTART is start address of transmitted packet)

### - Procedure Eth\_SetParameters;

```
// Set your IP, Mac, Mask, Gateway ... etc... here.  
// Str2Ip('192.168.1.14',eth_ip_addr);  
// Str2Ip('85.120.221.251',eth_ntp_addr);  
// Str2Ip('192.168.1.1',eth_gateway);  
// Str2Ip('255.255.255.0',eth_mask);  
// Str2Mac('0004A300809A',eth_mac);
```

### - Procedure Eth\_Init;

```
// Init Ethernet part from MCU.
```

### - Procedure Eth\_DoPacket;

```
// Process incoming packets without TCP/IP Stack
```

### - Procedure Do\_EthernetProc;

```
// Process all Ethernet processes including TCP/IP Stack
```

### - Procedure Eth\_UserProcess;

```
// List of user processes, put you desired Ethernet processes here.  
// this procedure is called by library, must be placed in unit eth_lib_user  
// see examples for more detail
```

### - Procedure Eth\_UserTCP(var dest\_ip\_addr\_T : IpHeader; var source\_port\_T, dest\_port\_T, len\_T : word);

```
// This procedure is called by library. Put your TCP response for TCP processes here.
```

### - Function Eth\_UserUDP(var dest\_ip\_addr\_U : IpHeader; var dest\_port\_U, source\_port\_U, len\_U : word) : word;

```
// This function is called by library. Put your UDP response here. See example. ( ECHO example )
```

- **Procedure Eth\_PutByte(value : byte);**  
// Put one byte in ETH memory.
  
- **Function Eth\_GetByte : byte;**  
// Get one byte from ETH memory.
  
- **Function CopyFlashToEthMem\_CP(start\_Flash\_address : longint): word;**  
// Copy Const from flash to Eth Memory and return length of Const data.  
// Const data must be defined as STRING. (and must be zero terminated)  
// Ex. len := CopyFlashToEthMem\_CP(@httpHeader);
  
- **Function CopyRamToEthMem\_CP(var s : string[255]): word;**  
// Copy var data to Eth Memory and return length of var data.  
// Var data must be defined as ARRAY. (and must be zero terminated)  
// Ex. len := CopyRamToEthMem\_CP('data data data');
  
- **Function Eth\_SendARP(var ip\_dest : IpHeader) : Boolean;**  
// Get ARP request. Return True if Mac exist.  
// Return dest. mac address in var dest\_mac : MacHeader;.  
// Return dest. ip address in var dest\_ip\_addr : IpHeader;.  
// Ex. bol := Eth\_SendARP(user\_ip\_addr);  
// First function search in ArpCache (size of 17) ...
  
- **Procedure SendUDP(dest\_port\_S, source\_port\_S, len\_data : word;**  
**var data\_udp : array[1472] of byte);**  
// Send UDP message. Max 1472 bytes.  
// Ex. If Eth\_SendARP(user\_ip\_addr) then SendUDP(dest\_port, eth\_port, 10, 'Test Test');
  
- **Procedure ErasePingCache;**  
// Erase Ping cache table.
  
- **Procedure Eth\_SendPing(SlotState : boolean; PingSlot : Byte);**  
// Have 8 Slots available 0 .. 7 and Ping Cache size of 8.  
// Ex. Eth\_SendPing(Eth\_SendARP(ip\_addr),0); Send Ping at ip\_addr and put result in Slot 0.  
// PingCache format :  
// PingCache[Slot].IpP : IpHeader;  
// PingCache[Slot].MacP : MacHeader;  
// PingCache[Slot].Time : word;  
// PingCache[Slot].TTL : byte.
  
- **Procedure CopyEthMemToRam(start\_eth\_address, dest\_ram\_address, length\_w : word);**  
// Ex. CopyEthMemToRam(AddrPacket+6,@dest\_mac\_addr,6);
  
- **Procedure CopyRamToEthMem(start\_ram\_address, dest\_eth\_address, length\_w : word);**  
// Ex. CopyRamToEthMem(@eth\_mac,TXSTART+22,6);
  
- **Procedure CopyFlashToEthMem(start\_Flash\_address : longint; dest\_eth\_address,**  
**length\_w : word);**  
// Ex. CopyFlashToEthMem(@httpHeader,TXSTART+54,30);

- **Procedure CopyEthMemToEthMem(start\_eth\_address, dest\_eth\_address, length\_w : word; where : byte);**  
// where = 0 copy from Eth RxBuf to Eth TxBuf;  
// where = 1 copy from Eth TxBuf to Eth TxBuf  
// Ex. CopyEthMemToEthMem(AddrPacket+38, TXSTART+28, 4, 0);
- **Procedure WriteToEthMem(dest\_eth\_address : word; value : byte);**  
// Ex. WriteToEthMem(TXSTART+12, \$08);
- **Function ReadFromEthMem(start\_eth\_address : word) : byte;**  
// Ex. data := ReadFromEthMem(AddrPacket+38);
- **Function EthMemCompareWithRam(start\_eth\_address, start\_ram\_address, length\_w : word) : boolean;**  
// Ex. bol := EthMemCompareWithRam(AddrPacket+30, @eth\_ip\_addr, 4);
- **Function EthMemCompareWithFlash(start\_eth\_address : word; start\_Flash\_address : longint; length\_w : word) : boolean;**  
// Ex. bol := EthMemCompareWithFlash(AddrPacket+54, @httpHeader, 30);
- **Function Eth\_Cksum(start\_eth\_address, length\_w : word) : word;**  
// Ex. cksum\_ip := Eth\_Cksum(TXSTART+14, 20);
- **Procedure Eth\_WritePHYReg(register\_address : byte; data : word);**  
// Write to PHY registers;
- **Procedure Eth\_SetLedConfig(NewConfig: word);**  
// Set Eth Led configuration. See datasheet for more detail.
- **Function Eth\_ReadPacket : word;**  
// Read packet and return TYPE OF SERVICE.
- **Procedure Eth\_Send(length\_w : word);**  
// Send packet from Tx buffer.
- **Function Send\_Ping(var ip\_address : IpHeader) : word;**  
// Send ping at specified ip\_address and return response time.  
// Ex. PingTimeResponse := Send\_Ping(user\_ip\_address);
- **Function Send\_UDP(var ip\_address : IpHeader; dest\_port\_S, source\_port\_S, len\_data : word; var data\_udp : array[1472] of byte) : boolean;**  
// Send UDP message, max 1472 bytes, at specified ip\_address and return true if success.  
// Ex. Success := Send\_UDP(user\_ip\_addr, 10001, 9999, 10, 'data data ');
- **Function Send\_ARP(var ip\_address : IpHeader; var mac\_address : MacHeader) : boolean;**  
// Get MAC for specified ip\_address, put result in mac\_address variable and return true if success.  
// Ex. Success := Send\_ARP(user\_ip\_addr, reply\_mac\_addr);

- **Procedure EraseARPCache;**  
// Erase ARP cache table.
  
- **Procedure CounterTask;**  
// Increments all counters used by library in ARP, ICMP, TCP, UDP,NTP routine.  
// This procedure must be called at 1ms in interrupt routine.
  
- **Procedure CopyEthMemToRam\_Inv(start\_eth\_address, dest\_ram\_address, length\_w : word);**  
// Ex. CopyEthMemToRam(AddrPacket+6,@data\_dWord,4);
  
- **Procedure CopyRamToEthMem\_Inv(start\_ram\_address, dest\_eth\_address, length\_w : word);**  
// Ex. CopyRamToEthMem(@data\_dWord,TXSTART+22,4);
  
- **Function Ntp\_query : Boolean;**  
// Synchronize time.  
// Ntp address must me stored in eth\_ntp\_addr (see Eth\_SetParameters)  
// If synchronization was successfully Ntp\_sync flag will be true.
  
- **Procedure Get\_Time;**  
// Transform NTP time in day/month/year ...  
// Data will be stored in :  
// TTime.Rfc = Time in RFC format  
// TTime.Unix = Time in UNIX format  
// TTime.Year = Year (ex. 2008)  
// TTime.Month = Month (ex. 07)  
// TTime.Day = Day (ex. 24)  
// TTime.Hour = Hour(ex. 12)  
// TTime.Min = Min (ex. 59)  
// TTime.Sec = Sec (ex. 32)  
// TTime.Str = String format max. 32 ('2008-07-24 12:59:32')
  
- **Procedure Firewall (ICMP, TCP, UDP : boolean);**  
// Default all false, allow all type of packets, depending of **OPENED PORTS**.  
// If ICMP = true, ignore ICMP request. Ex. Firewall(true, false, false);  
// If TCP = true, ignore TCP packets. Ex. Firewall(false, true, false);  
// If UDP = true, ignore UDP packets. Ex. Firewall(false, false, true);

**20-Aug-2008**

Added **hw\_cksum** flag, default state **false** (Ethernet checksum in software)

User can modify this flag on the fly, (can chose between **software checksum** and **hardware checksum**)

Regarding ERRATA, hardware checksum is not recommended!

10-Nov-2008

Added new **Firewall** rules.

Default **ALL Ports** are blocked, both **TCP** and **UDP**.

Have **10** sockets for opened **TCP** Ports and **10** for **UDP**.

### Implemented:

- **Procedure UDP\_Open\_Port(port\_u : word);**

// Open UDP Port number, must be <> 0.

- **Procedure UDP\_Close\_Port(port\_u : word);**

// Close UDP Port number, must be <> 0.

- **Procedure TCP\_Open\_Port(port\_u : word);**

// Open TCP Port number, must be <> 0.

- **Procedure TCP\_Close\_Port(port\_u : word);**

// Close TCP Port number, must be <> 0.

- **Procedure Eth\_Policy(poli : byte);**

// **poli** can be **Drop** or **Reject** .

// if is **Drop**, library will ignore incoming packets addressed to closed ports.

// if is **Reject**, library will reply to Host with ICMP packet **Destination Host Unreachable**

// default value is **Drop**.

### TCP/IP Stack

The **Structure** for one socket, **10** are available:

```
type  TSocket = Record
      State_S : byte;           // State of Socket, 0–disconnect, 1-waiting for connection
                                   // 2-connected
      source_port_S : word;     // Ethernet module Source Port
      Dest_Ip : IpHeader;       // Destination IP Address
      Dest_Mac : MacHeader;     // Destination MAC Address
      dest_port_S : word;       // Destination Port Address
      Seq_No_S : dword;         // Sequence Number
      Ack_No_S : dword;         // Acknowledgment Number
      Wait_ACK : byte;          // Wait_ACK flag
      Exp_Time : byte;          // Expiration Time (sec)
      Start_addr : dword;       // Pointer Start Address for transmitted data
      Stop_addr : dword;        // Pointer Stop Address for transmitted data
      RAM_ROM : byte;           // Pointer location, 0-RAM, 1-ROM
      Keep_Alive : boolean;     // Keep Alive connections, max 60 sec without activity if true
end;
```

**- Function Open\_TCP\_Connection(var dest\_ip\_addr: IpHeader; dest\_port\_T, source\_port\_T : word) : boolean;**  
// Open one TCP connection , see exaples  
// Return True if done successfully.

**- Procedure Close\_TCP\_Connection(var dest\_ip\_addr: IpHeader; dest\_port\_T : word);**  
// Close TCP connection

**- Function Send\_TCP(var dest\_ip\_addr: IpHeader; dest\_port\_T, len\_data : word; var data\_tcp : array[1400] of byte) : boolean;**  
// Send data over TCP, from RAM, max 1400 bytes, return true if successfully

**Author: Florin Andrei Medrea, Software Version V3.5**  
**Copyrights (c) 2008 - YO2LIO, All Rights Reserved**