EtherNet/InterNet Treiber AVRco NetStack *xUDP* vorläufig

Es gibt eine Vielzahl Möglichkeiten, wie zwei Einheiten (Prozessoren, Steuerungen etc) miteinander kommunizieren können. Eine davon ist Ethernet. Der Begriff embedded Ethernet wird zur Zeit sehr viel benutzt, wenn es um schnelle Verbindungen zwischen mehreren Steuerungen geht oder um die PC<->Steuerung Anbindung.
Hier wird sehr oft auch vom embedded TCP/IP gesprochen. Doch hier muss man vorsichtig sein. TCP/IP hat sich zum Sammelbegriff für Ethernet etabliert. In Wirklichkeit steckt aber in den meisten Fällen nicht das TCP-Protokoll selbst dahinter sondern eine UDP/IP Verbindung (z.B. ProfiBUS-NET). Zwar wird hier auch oft das TCP Protokoll mit implementiert, wenn man aber genau hinschaut, werden Echtzeit Aufgaben immer über UDP/IP gefahren. TCP/IP ist durch sein sehr aufwendiges Handshake Verfahren für Steuerungs Aufgaben nicht geeignet.
Der Kenner der Netzwerk Hirarchie wird jetzt einwerfen, dass das UDP-Protokoll zwar extrem schnell ist, aber als „unsicheres“ Protokoll bezeichnet wird. Das ist richtig. UDP kennt kein Handshake. Der Sender schickt ein Packet ab und damit hat es sich für erledigt. Ob das Packet angekommen ist, kann er nicht feststellen, da UDP keine Rückmeldung besitzt.
Dieses Manko des „unsicheren“ Protokoll wird bei E-LAB (und auch bei allen anderen Industrie-UDPs) dadurch umgangen, dass über das Standard UDP ein Handshake Verfahren darüber gesetzt wird. E-LAB nennt das ***extended UDP*** oder abgekürzt ***xUDP***.

xUDP

Beim ***xUDP*** fügt der **Sender** dem Datenpacket noch eine 2Byte Checksumme an, die die Summe aller Bytes des Daten Teils enthält. Der Datenblock (byte count) des Senders erhöht sich dadurch um 2Bytes. Der **Empfänger** rechnet jetzt seine eigene Checksumme aus dem Datenteil ohne die angehängte Checksumme. Als Acknowledge (Handshake) für den Empfang des Packetes schickt er diese gerechnete Checksumme an den Sender (UDPAknPort) zurück.

Der **Sender** weiss damit, dass der Empfänger zumindest das Packet empfangen hat. Er vergleicht nun die empfangene Checksumme mit der beim Senden von ihm gerechneten und angehängten und kann damit feststellen, ob das Datenpacket auch korrekt und vollständig beim Empfänger angekommen ist.

Der **Empfänger** wiederum vergleicht auch die von ihm gerechnete Checksumme mit der der vom Sender an das Datenpacket angehängten und kann auch feststellen, ob das Datenpacket vollständig und in Ordnung ist.
Hiermit wird das normalerweise „unsichere“ UDP zum sicheren ***xUDP.***

Mit ***xUDP*** kann ein sehr sicheres Hochgeschwindigkeits Verfahren installiert werden, das in jeder Beziehung dem TCP überlegen ist. Für diejenigen Applikationen, die TCP, FTP etc. benötigen, wird E-LAB in Kürze ein sogenanntes ***GateWay*** Programm zur Verfügung stellen, das xUDP Packete mit Hilfe eines am Netzwerk angeschlossenen PCs in alle anderen Protokolle umsetzen kann. Dieses Gateway Verfahren wird auch von vielen anderen embedded Ethernet Implementationen angewendet, wo nur beschränkte Resourcen (RAM, ROM) auf dem Chip zur Verfügung stehen (z.B. Motorola HC11/HC16).

Perfomance

Mit einem mega103 mit 6.4MHz können folgende Transferraten erzielt werden :

PING 65 Bytes ca. 3msec PING 1450 Bytes ca. 10msec
Datentransfer ca. 110kBytes/sec entspricht einer Bitrate von ca. 1.0Mbit/sec

Code Grösse ca. 6kBytes Speicherbedarf RAM ca. 400Bytes + RxBuffer + TxBuffer
Speicherbedarf Stack ca. 40 Bytes Frame ca. 60 Bytes EEprom ca. 55 Bytes

Import

Der Treiber muss, wie beim AVRco üblich, importiert werden.

```
Import SysTick, Netstack, ..;
```

Da zur Zeit der generische HardWare Treiber *NetStack_IOS* noch nicht implementiert ist, muss der interne Hardware Treiber *RTL8019* importiert werden. Wenn das StackMapper PC-Programm am Datenaustausch beteiligt werden soll, müssen die zsätzlich benötigten Protokolle importiert werden.

```
From NetStack Import RTL8019, xEmail, xFTP, xHTTP, xTelNet, xChatMode,
                     XFileMode, xPeerMode;
```

Für den Receiver Teil muss das gewünschte InputPort (PINx) und der PortPin definiert werden.

```
Define
        ProcClock    = 6400000;              // 6.4 MHz
        SysTick      = 10;                   // 10msec
        XData        = $8000,$FFFF;          // external data area
        XData1       = $4300,$4FFF, noInit;  // I/O area of RTL8019
        StackSize    = $30, iData;           // at least 50 bytes stacksize
        FrameSize    = $40, iData;           // at least 64 bytes frame size
        NetStack     = 1536, xData;          // total buffer size, location
        ARPcacheSize= 20;                    // arpcache entries
        RTL8019      = $4300, Int5;          // RTL base addr, Interruptpin
        RTLreset     = PortD, 2;             // reset line to RTL8019
```

Exportierte Typen und Konstanten

Der NetStack Treiber exportiert verschiedene Typ Deklarationen, die im Anwendungs Programm verwendet werden müssen:

```
Type tMACaddress        = array[0..5] od byte;

Type tIPaddress         = array[0..3] od byte;

Type tGetPacketInfo     = record
                            TypeOfPacket : TPacketType;
                            SourceIP     : TIPAddress;
                            UserPortID   : byte;
                            SourcePort   : word;
                            DestPort     : Word;
                            DataBeginPtr : Pointer;
                            DataLength   : word;
                          End;

Type tUDPpackInfo       = record
                            TargetIPAddr: TIPAddress;
                            SourcePort  : Word;
                            DestPort    : Word;
                            WaitResp    : boolean;
                            DataPtr     : pointer;
                            DataLen     : word;
                          End;

Type tNetErrorType      = (NetErrNone, NetErrChkSum, NetErrSndPkt,
                           NetErrNoAkn, NetErrPktSize);

Type tPacketType        = (ptARP, ptPINGRequest, ptPINGResponse, ptICMP,
                           ptTCP, ptUDP, ptHTTP, ptFTP, ptTFTP, ptTELNET,
                           ptNONE);

Type tIdentType         = (idxRequMapper, idxFTP, idxEmail, idxTelNet,
                           idxChat, idxHTTP, idxFile);
```

```
type
  TIdentInfo   = record
                   IdentCode    : String[13];
                   BName        : String[20];
                   AknPort      : Word;
                   BufSize      : Word;
                   Stackversion: Word;
                   CompileDate  : String[10];
                   CompileTime  : String[5];
                   CompilerRev  : Word;
                   Firmware     : String[20];
                   xUDP         : boolean;
                 end;

  TMAPerror    = (MapErrNone, MapErrStackUnknown, MapErrFailed);

  TFTPcmd      = (FTPChangeDir, FTPChangeDirUp, FTPDelete, FTPMakeDir,
                  FTPRemoveDir, FTPRename, FTPSendFile, FTPFileAppend,
                  FTPReceiveFile);

  TFTPInfo     = record
                   Username     : string[20];
                   Password     : string[20];
                   FTPServer    : string[15];
                   Port         : word;
                   Filename     : string[32];
                   Filename1    : string[32];
                   TransfBinary : boolean;
                   CmdCode      : tFTPcmd;
                   Result       : tMapError;
                 end;

  tMailInfo    = record
                   MailServer    : string[16];
                   Port          : word;
                   Sender        : string[32];
                   Recipient     : string[32];
                   CC            : string[32];
                   BCC           : string[32];
                   UserID        : string[16];
                   ReplyTo       : string[16];
                   Subject       : string[16];
                   Result        : tMapError;
                 end;

  TFILEInfo    = record
                   Filename     : string[32];
                   Filename1    : string[32];
                   FILEcmd      : tFTPcmd;
                   Result       : tMapError;
                 end;

const
  StackIdentPort    = 61611;
  StackMapperPort   = 61612;
```

Exportierte Variablen

Es wird eine grössere Anzahl Variablen benötigt und exportiert, auf die die Applikation nicht direkt zugreifen sollte. Beim Empfang einer StackMapper Botschaft kann die Art der Botschaft durch die Enumeration

```
Var MapperPackType : tIdentType;
```

identifiziert werden. Je nach dem Inhalt der Variablen „MapperPackType“ kann auf folgende Records zugegriffen werden:

```
Var RxMailInfo    : tMailInfo;
    RxFTPInfo     : tFTPInfo;
    RxTelNetInfo  : tTelNetInfo;
    RxHTTPInfo    : tHTTPInfo;
    RxChatInfo    : tChatInfo;
    RxFileInfo    : tFileInfo;
    RxPeerInfo    : tPeerInfo;
```

Beim Senden einer Botschaft oder Auftrags an den StackMapper muss dafür eine der bereitgestellten Variablen benutzt, werden, abhängig vom Botschafts Typ:

```
Var TxMailInfo    : tMailInfo;
    TxFTPInfo     : tFTPInfo;
    TxTelNetInfo  : tTelNetInfo;
    TxHTTPInfo    : tHTTPInfo;
    TxChatInfo    : tChatInfo;
    TxFileInfo    : tFileInfo;
    TxPeerInfo    : tPeerInfo;
```

Exportierte Funktionen und Prozeduren

Setup

Die folgenden Funktionen und Prozeduren werden normalerweise nur beim Programm Start benötigt.

```
Procedure Net_SetLocalIPAddress(IPOct4, IPOct3, IPOct2, IPOct1 : Byte;
                                MaskOct4, MaskOct3, MaskOct2, MaskOct1 : Byte);
```

Da die lokale IP-Adresse auch im EEprom abgelegt werden muss, ist diese Prozedur nur notwendig, wenn zur Laufzeit die IP-Adresse geändert werden muss.

```
Procedure Net_SetDefaultGateway(Gateway4, Gateway3, Gateway2, Gateway1 :
                                Byte);
```

Da die Gateway Adresse auch im EEprom abgelegt werden muss, ist diese Prozedur nur notwendig, wenn zur Laufzeit die Gateway-Adresse geändert werden muss.

```
Procedure Net_StackSetAktive(YESNO : Boolean);
```

Der NetStack Treiber ist default inaktive. Mit dieser Prozedur kann der Treiber vom Netzwerk abgehängt (false) und wieder angeschaltet werden (true).

```
Procedure Net_ExtendedUDP(YESNO : Boolean);
```

Der Extended UDP-Mode ist default inaktive. Mit dieser Prozedur kann der Extended UDP-Mode abgeschaltet (false) und wieder angeschaltet werden (true).

```
Function  Net_AddUsrPort(Port : Word; ID : byte) : boolean;
```

Mit Net_AddUsrPort wird ein Port freigegeben über das jetzt Daten empfangen werden kann. Port bezeichnet dabei die Port Nummer und ID einen frei wählbaren Wert, der aber insgesamt nur einmal vorkommen darf. Diese ID wird mit der Funktion `Net_GetPacket` im Record `GetPacketInfo` zur Information der Applikation mit zurückgegeben.

```
Function  Net_DelUsrPort(Port : Word) : boolean;
```

Mit Net_DelUsrPort wird ein Port aus der Port LookUp Table gelöscht und ist damit nicht mehr von aussen erreichbar.

Runtime Schalter

Mit diesen Prozeduren kann das Verhalten des NetStacks nach aussen gesteuert werden.

```
Procedure Net_AllowARPResponse(YESNO : Boolean);
```

Mit einem false wird verhindert, dass der Stack gescannt werden kann. D.h. er wird durch ein Broadcast nicht erfasst und ist damit nicht sichtbar. Wichtig für die Sicherheitstechnik. Der Schalter ist default true.

```
Procedure Net_AllowPINGResponse(YESNO : Boolean);
```
Mit einem false wird verhindert, dass der Stack gepingt werden kann. D.h. er wird durch einen Ping nicht erreicht und ist damit nicht sichtbar. Wichtig für die Sicherheitstechnik. Der Schalter ist default true.

```
Procedure Net_AllowRequestBoard(YESNO : Boolean);
```
Mit einem false wird verhindert, dass die internen Daten des NetStack abgefragt werden können. D.h. er wird auf ein BoardRequest (s.u.) nicht reagieren. Wichtig für die Sicherheitstechnik. Der Schalter ist default true.

Operationen

Diese Funktionen/Prozeduren dienen zur Kommunikation des NetStacks über das Netzwerk.

```
Procedure Net_StartNet;
```
Diese Prozedure startet die Kommunikation des NetStacks über das Netzwerk.

```
Function Net_PollPacket : boolean;
```
Diese Funktion gibt den Receive Status des NetStacks zurück. Das Ergebnis ist true, wenn ein Packet vorhanden ist, ansonsten false.

```
Function Net_GetPacket(VAR GetPacketInfo : TGetPacketInfo) : Boolean;
```
Diese Funktion gibt im Record *GetPacketInfo* alle relevanten Informationen des zuletzt empfangenen Packets zurück., wenn das Funktions Ergebnis true war. Ist das Ergebnis false, so wurde ein Packet empfangen, das entweder defekt war, oder für interne Zwecke schon arbgearbeitet wurde, z.B. ARP, PING, NetStackInfo etc. Diese Funktion erwartet also, dass ein Packet empfangen wurde. Wird diese Funktion aufgerufen, ohne dass ein Packet vorhanden ist, kehrt sie erst wieder zurück, wenn dann irgendwann doch ein Packet hereingekommen ist. Deshalb sollte die Funktion **Net_PollPacket** unbedingt vorher aufgerufen werden, und erst wenn diese TRUE ist, sollte Net_GetPacket aufgerufen werden.

```
Function Net_SendUDPPacket(UDPinfo : tUDPpackInfo) : TNetErrorType;
```
Dies ist die Transmit Funktion. Die Applikation übergibt im Record *UDPInfo* alle relevanten Informationen des zu sendenden Packets. Das Resultat der Funktion ist vom Typ tNetErrorType.

```
Function Net_RequestMapper(const IdentType : tIdentType) : TNetErrorType;
```
Diese Funktion dient zum suchen und kommunizieren mit einem StackMapper bzw. xUDPMapper. Das ist ein E-LAB PC Programm, das xUDP Packete in FTP, E-MAIL etc. Packete umsetzt, weiterschickt und empfängt. Der Parameter „*IdentType*“ bestimmt dabei das Verhalten dieser Funktion.

`IdxRequMapper`	der NetStack meldet sich beim Mapper. Anmelden ist Voraussetzung für alle weiteren Operationen mit dem StackMapper. Keine weiteren Daten möglich oder notwendig.
`IdxFTP`	Der NetStack nimmt eine pseudo-FTP Verbindung mit dem Mapper auf. Dazu muss der Record **TxFTPInfo** zuvor mit den entsprechenden Angaben gefüllt werden.
`IdxEmail`	Der NetStack nimmt eine pseudo-E-Mail Verbindung mit dem Mapper auf. Dazu muss der Record **TxMailInfo** zuvor mit den entsprechenden Angaben gefüllt werden.
`idxTelNet`	Der NetStack nimmt eine pseudo-TelNet Verbindung mit dem Mapper auf. Dazu muss der Record **TxTelNetInfo** zuvor mit entsprechenden Angaben gefüllt werden
`idxChat`	Der NetStack nimmt eine Chat Verbindung mit dem Mapper auf. Dazu muss der Record **TxChatInfo** zuvor mit den entsprechenden Angaben gefüllt werden.
`IdxHTTP`	Der NetStack nimmt eine pseudo-HTTP Verbindung mit dem Mapper auf. Dazu muss der Record **TxHTTPInfo** zuvor mit den entsprechenden Angaben gefüllt werden.
`IdxFile`	Der NetStack nimmt Datei-Operationen mit dem Mapper auf dem PC des Mappers vor. Dazu muss der Record **TxFileInfo** zuvor mit den entsprechenden Angaben gefüllt werden.
`IdxPeer`	Der NetStack nimmt Datei-Operationen mit dem Mapper auf dem Peer-To-Peer Netzwerk vor. Dazu muss der Record **TxPeerInfo** zuvor mit den entsprechenden Angaben gefüllt werden.

Bis auf den IdxRequMapper Aufruf kann jedem Aufruf Typ ein Daten Packet angehängt werden, welches durch diese Funktion mitgeschickt wird. Die zugehörigen Funktionen dazu sind „Net_TxInitDataBlock“ und „Net_TxAddDataStr“.

Das Ergebnis der Funktion ist ein „NetErrNone“ wenn der Mapper gefunden wurde bzw. die Kommunkation erfolgreich war, ansonsten ist ein TimeOut Fehler aufgetreten. Ausser bei „IdxReqMapper“ schickt der StackMapper immer ein entsprechendes xUDP Antwort Packet zurück, das ausgewertet werden muss. Dieser Info Record ist vom gleichen Typ wie der gesendete, wobei der letzte Eintrag den Sendestatus enthält.

```
Function Net_TxInitDatablock(const IdentType : tIdentType) : pointer;
```
Der Funktion Net_RequestMapper kann veranlasst werden, zusätzlich zu dem Info-Record auch noch Daten mitzuschicken, z.B. den eigentlichen e-mail Text. Dazu muss vor dem Aufruf des Mappers der Datenblock, der die Daten aufnehmen soll, initialisiert werden. Dies geschieht mit dieser Funktion. Auch wenn der Datenblock leer bleiben soll, muss diese Prozedur aufgerufen werden, um Fehler zu vermeiden.

```
Function Net_TxAddDataStr(st : string[32]) : boolean;
```
Nach der Initialisierung des Datenblocks mit „Net_TxInitDataBlock“ kann mit dieser Prozedur Strings in den Datenblock eingebaut/angehängt werden. Das Resultat „Pointer“ der Init-Funktion ist dafür nicht notwendig. Konnten die Daten angefügt werden, ist das Resultat true.

Es besteht auch die Möglichkeit, mit der Kopierfunktion „CopyBlock“ den Datenbereich zu füllen. Beispiel:

```
destPtr:= Net_TxInitDatablock;
CopyBlock(srcPtr, destPtr, count);
_TX_DATA_COUNT:= count;
err:= Net_RequestMapper(idxFTP);
```

Die System Variable _Tx_Data_Count muss hierbei unbedingt richtig gesetzt werden, wenn CopyBlock oder ähnliches benutzt wird.

Support Funktionen

```
Procedure STRtoIP(IPstr : String[15]; VAR Result : TIPAddress);
```
Die Funktion wandelt einen IP-string im Format ’nn.nn.nn.nn’ in eine IP-Adresse um.

```
Function  IPtoSTR(IP : TIPAddress) : String[15];
```
Die Funktion wandelt eine IP-Adresse in einen IP-string im Format ’nn.nn.nn.nn’ um.

Support Tools

Zur Zeit gibt es drei Support Programme zum Testen und für den Betrieb des NetStacks

xUDPconf ist ein Programm, das aus der IDE PED32 heraus oder auch direkt aufgerufen werden kann. Es Dient zur Inbetriebnahme einer NetStack Hardware und zur absolut notwendigen Konfiguration des Board Namens, der IP-Adresse und der MAC-Adresse

StackCheck ist ein sehr umfangreiches Test Programm mit dem man ein komplettes Netzwerk, die angeschlossenen Computer und NetStacks ausgiebig und intensiv testen kann. Es muss separat von der E-LAB Homepage heruntergeladen, installiert und freigeschaltet werden.

xUDPmapper ist ein Programm mit mehreren Aufgaben. In erster Linie dient es zum Umsetzen von xUDP Packeten nach TCP/IP Packeten (FTP, http etc) und zurück. Es dient zur Verbindung eines oder mehrerer NetStacks mit der TCP/IP Welt. Deshalb sollte es auf einem Rechner im lokalen Netzwerk im WIN-Autostart eingetragen werden. Das Programm *xUDPconf* ist hier auch enthalten. Weiterhin sind ein paar einfache aber effektive Test Möglichkeiten vorhanden. xUDPconf und xUDPmapper werden automatisch mitinstalliert, nicht jedoch StackCheck.

Applikation

Die Applikation muss diverse Konstanten zur korrekten Funktion des NetStacks bereitstellen.

```
const                              // into Flash
  // these 2 constants !must! be provided by the user
  Date = 'dd.mm.yyyy';
  Time = 'hh:mm';

{$EEPROM}
StructConst                       // into EEprom
  // first 2 bytes in AVR EEprom not useable
  eDummy             : word        = 0;
  // these 9 constants !must! be provided by the user
  Net_ARPTimeOut     : Byte        = 100;              // Systicks !!!
  Net_ARPRetry       : Byte        =  3;               // Request retries
  Net_SendTimeOut    : Byte        = 50;               // Systicks !!!
  Net_SendRetry      : Byte        = 5;                // Send retries
  Net_UDPAknPort     : Word        = 1189;             // Handshake port
  Net_LocalMACaddr   : tMACaddress = ($02, $2C, $2C, $CE, $AE, $15);
  Net_LocalIPaddr    : tIPaddress  = (192, 168, 1, 15);
  Net_LocalMask      : tIPaddress  = (255, 255, 255, 0);
  Net_xUDPMapper     : tIPaddress  = (0, 0, 0, 0);
  Net_DefaultGateWay: tIPaddress  = (0, 0, 0, 0);
  Net_BoardName      : String[20]  = 'E-LAB xUDP NET-Stack';
  Net_FirmWare       : String[20]  = 'E-LAB AVRco Test';
```

Optionen

```
procedure NetStack_LEDs(LED : byte; OnOff : boolean);
```

Diese optionale CallBack Prozedur kann von der Applikation bereitgestellt werden. Wenn sie vorhanden ist, ruft sie der NetStack Treiber auf, um bestimmte LEDs an- bzw. auszuschalten.

Details

Zur Zeit noch nicht fertig.

```
Define
  XData
  XData1
  StackSize
  FrameSize
  NetStack
  ARPcacheSize
  RTL8019
  RTLreset

MACaddress
Ipaddress

GetPacketInfo
   TypeOfPacket
   SourceIP
   UserPortID
   SourcePort
   DestPort
   DataBeginPtr
   DataLength

UDPpackInfo
   TargetIPAddr
   SourcePort
```

```
   DestPort
   WaitResp
   DataPtr
   DataLen

NetErrorType
   NetErrNone
   NetErrChkSum
   NetErrSndPkt,
   NetErrNoAkn
   NetErrPktSize

PacketType
   PtARP
   PtPINGRequest
   PtPINGResponse
   ptICMP
   ptTCP
   ptUDP
   ptHTTP
   ptFTP
   ptTFTP
   ptTELNET
   ptNONE

Net_ARPTimeOut
Net_ARPRetry
Net_SendTimeOut
Net_SendRetry
Net_UDPAknPort
Net_LocalMACaddr
Net_LocalIPaddr
Net_LocalMask
Net_ReDirector
Net_DefaultGateWay
Net_BoardName
Net_FirmWare

StackIdentPort

TIdentInfo
   IdentCode
   BName
   AknPort
   BufSize
   Stackversion
   CompileDate
   CompileTime
   CompilerRev
   Firmware
   xUDP
```

Beispiel und Schaltplan
Ein Beispiel Programm ist unter **..\AVRco\Demos\NetWork** zu finden. Der zugehörige Schaltplan **EtherBoardsch.PDF** befindet sich in der Directory **..\AVRco\DOCs**