Using Ethernet In Embedded APPLICATIONS

Professor Dr Dogan Ibrahim of the Near East University Cyprus describes the use of Ethernet in embedded microcontroller applications and gives the design of an example Ethernet-based home automation system

ETHERNET IS, BY FAR, the leading wired standard for networking as it enables to connect a very large number of computers, microcontrollers and other computer-based equipment to each other. Ethernet allows devices and equipment to be accessed remotely and provides a cost-effective and reliable means of monitoring or controlling such equipment.

Ethernet has traditionally been implemented on PCs and laptops and has been used widely at home, office and industry to access the worldwide Internet and company-wide intranet networks. Internet can nowadays also be accessed using low-cost portable electronic gadgets such as smart mobile phones, PDAs and others.

With recent advances in technology and especially in chip manufacturing, Ethernet can now be fully realized through single-chip devices. Several manufacturers offer either single-chip Ethernet controllers, or general purpose microcontrollers with on-board Ethernet controllers. Such controllers can very easily be configured, programmed and used in Ethernet-based embedded applications. Typical applications of embedded Ethernet include:

- Home automation
- Access control
- Environmental monitoring
- Industrial control
- Safety and security
- POS terminals
- Remote control
- Lighting control.

PIC18F97J60 is an advanced microcontroller chip manufactured by Microchip Inc (www.microchip.com) with built-in 10Base-T Ethernet capability. The device offers 128K flash program memory, 3908 bytes of RAM memory, 70 I/O pins, 16 10-bit A/D channels, timers, counters and many more features. Microchip's ENC28J60 is a popular 28-pin serial Ethernet chip that can easily be used in microcontroller-based applications to provide Ethernet capability to the application.

Several manufacturers offer Ethernet development kits, enabling users to learn and incorporate Ethernet functionalities in their designs. Examples of some development kits are:

- ETHERNETDK (www.silabs.com) is an Ethernet development kit that provides all the necessary hardware and software to create real-time embedded Ethernet-based applications. The kit contains an 8051 microcontroller-based C8051F120 microcontroller and a CP2200 Ethernet chip with an RJ45 connector. A Silicon Laboratories evaluation kit and application examples are provided on a CDROM distributed with the kit.
- PIC-MINI-WEB (www.olimex.com) is an embedded Ethernet development board using a PIC18F25J10 chip as the microcontroller and an ENC28J60 Ethernet chip. 1Mbit on board serial flash memory is provided to store web pages. In addition, a mini ICSP/ICD is available on the board for programming and debugging.
- The Embedded Ethernet Development kit manufactured by CCS (www.ccsinfo.com) is a complete kit based on the PIC18F4620 microcontroller and the ENC28J60 Ethernet chip. The kit provides 30 I/O pins, one potentiometer, LEDs, serial EEPROM, an MMC card reader and RJ45 socket. The development kit is supported by company's IDE, including a compiler and in-circuit real-time debugging tool.
- The Serial Ethernet Board manufactured by mikroElektronika (www.mikroe.com) is based on the ENC28J60 Ethernet chip and it can easily be connected to company's development boards (EasyPIC5 or EasyPIC6). The board is fully supported by mikroElektronika C, Pascal and Basic compilers.

This article describes the details of the ENC28J60 Ethernet chip and shows how the chip can be used in a simple microcontroller-based networked home automation system to turn lights ON and OFF.

DESTINATION ADDRESS	SOURCE ADDRESS	TYPE	DATA	CRC

Figure 1: *Ethernet packet format*

UDP SOURCE PORT	UDP DESTINATION PORT
MESSAGE LENGTH	CHECKSUM
DATA	

Figure 2: *UDP packet format*

VERSION	IHL	TYPE	LENGTH	
IDENIFICATION			FLAGS	OFFSET
Time To Live	PROTOCOL	CHECKSUM		
SOURCE ADDRESS				
DESTINATION ADDRESS				
OPTIONS			PADDING	
DATA				

Figure 3: *IP packet format*

Ethernet Communications

Ethernet is a frame-based computer networking technology for Local Area Networks (LAN), standardized using the IEEE 802.3.

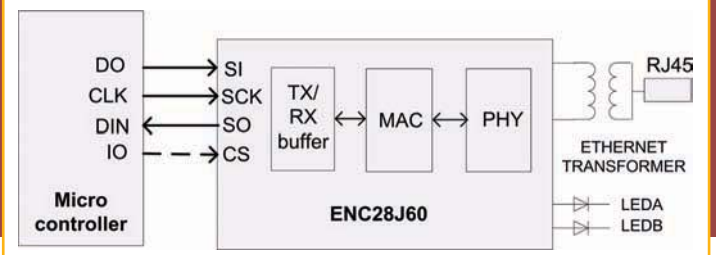

Figure 4: *Connecting the ENC28J60 Ethernet chip to a microcontroller*

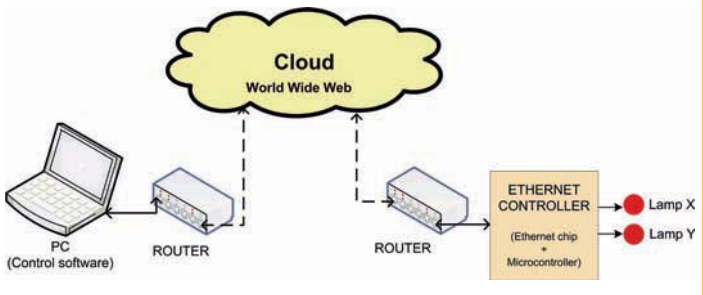

Figure 5: *Block diagram of the project*

Ethernet was originally invented by Xerox in 1972 and then developed by Xerox, DEC and Intel.

An Ethernet LAN typically uses coaxial cable, twisted pair wires, fibre optic, or can be in the form of wireless LANs. The most common form of Ethernet is called 10Base-T and it provides a transmission speed up to 10Mbps. Fast Ethernet or 100Base-T provides transmission speeds up to 100Mbps. Gigabit Ethernet provides even higher level of support at 1000Mbps.

Devices on the Ethernet are all connected together and compete for access using a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. The advantage of CSMA/CD is that, unlike Token Ring and Token Bus, all nodes can see each other with only one transmitting at a time to avoid any collisions. In case of a collision the transmitting nodes wait for a random time and attempt to re-transmit, hoping to avoid the collision. The maximum length of an Ethernet cable depends upon the speed of transmission and the type of cable used. For standard twisted pair type cables operating at 10Mbps, the maximum cable length is specified as 100m. Using fibre-optic cables this length can be extended to 2000m or over.

Ethernet is used to communicate using various protocols, e.g. DECnet, IP, ARP, etc. **Figure 1** shows the Ethernet packet format (en.wikipedia.org/wiki/Ethernet). A packet consists of 6 byte destination MAC address, 6 byte source MAC address, 2 byte data type, 45 to 1500 bytes data and a 4 byte CRC. Additionally, when transmitted on the Ethernet medium, a 7-byte preample field and Start-of-Frame delimiter byte are appended to the beginning of the Ethernet packet. As we shall see later, the data type for IP packets is 0x0800.

In this article we will use the Ethernet with IP packets for data communication over the network. Two protocols are supported under this scheme: TCP and UDP. TCP is an advanced data protocol requiring connection and providing guaranteed packet delivery with re-transmission if an error occurs. TCP acknowledges transmitted packets to confirm their correct delivery. In most remote data control applications the simpler UDP data protocol is used. Since this is the protocol used in the example in this paper, further details about the UDP is given in the next section.

The UDP

The User Datagram Protocol (UDP) is commonly used in remote automation applications. This protocol offers:

- UDP does not establish connection before sending data, it just packages it.
- UDP has only basic error checking using checksums.
- There is no sequencing of data in UDP and thus UDP does not ensure that data is received in the same order that they were sent.
- UDP is efficient for broadcasting/multicast transmission.
- UDP is faster, simpler and more efficient than TCP, however it is less robust than TCP.
- The delivery of data cannot be guaranteed in UDP.
- There is no re-transmission of lost packets in UDP.

UDP is widely used and recommended for data transfer over a network where performance and speed are more important than reliable delivery and where acknowledgement of delivery is not needed. Another important application area of UDP is in multicast or broadcast transmissions, since these are not supported by TCP.

Some common examples where UDP is used are:

- In multicast systems
- Domain Name System (DNS)
- Simple Network Management Protocol (SNMP)
- Dynamic Host Configuration Protocol (DHCP)
- Routing Information Protocol (RIP)
- Streaming video and VOIP.

A UDP packet consists of a Header and Data (see **Figure 2**). The header is always 4 bytes long and consists of the source and destination port numbers, message length and the checksum. This is followed by the actual data bytes.

The checksum is calculated by forming a UDP Pseudo-Header consisting of the source and destination addresses, protocol number and the data length. The UDP packet is encapsulated within an IP packet (see **Figure 3**) consisting of 20 bytes, followed by the UDP packet. The UDP packet protocol number is 17 (0x11). The IP packet is then encapsulated within an Ethernet packet and is transmitted to its destination node over the network.

There are several programs available in the market that can be used to analyze packets on an Ethernet network. Some programs are open source, provide no user support and can be downloaded free of charge from the Internet (e.g. Ethereal, Wireshark, Tcpdump, Dsniff), while some others come with full user support and can be purchased (e.g. EtherDetect, Colasoft Capsa).

The ENC28J60 Ethernet Chip

The ENC28J60 is a standalone 28-pin Ethernet controller chip with SPI interface and meets all of the IEEE 802.3 specifications. The chip has the following basic features:

- Compatible with 10/100/1000Base-T networks.
- Supports automatic polarity detection and correction.
- Supports half-duplex and full duplex operation.

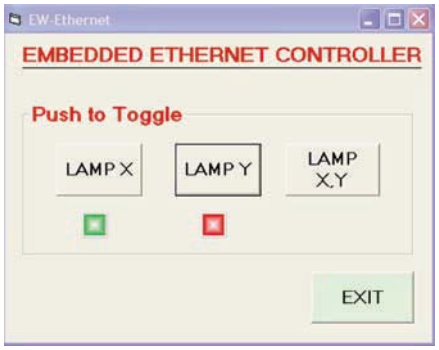

Figure 6: *The PC form*

- Automatic retransmit on collision.
- Automatic rejection of erroneous packets.
- SPI interface with up to 20MHz speed.
- 8K byte transmit/receive buffer.
- Support for unicast, multicast and broadcast addresses.
- Link and Activity LED interface.
- Differential signal input-output for interface to RJ45 connector.

The chip incorporates transmit and receive buffers, MAC (Medium Access Control) module, PHY (Physical Layer) module and associated control and interface logic. **Figure 4** shows how the chip can be used in microcontroller-based applications. Basically, the interface requires the SPI signals SI, SO and SCK to be connected to the microcontroller. In addition, the CS signal can also be driven from the microcontroller. The chip is connected to a network through a RJ45 type socket and a suitable Ethernet transformer.

Example Ethernet-Based Project

This section describes the design of a simple microcontroller-based home automation system using the Ethernet as the communication medium. **Figure 5** shows the block diagram of this example project. The project hardware is in two parts, connected using a network hub or a switch: the Ethernet controller and the PC.

The Ethernet controller consists of a microcontroller and an ENC28J60 Ethernet chip. Two LEDs connected to the microcontroller simulate two lamps X and Y. A GUI-based program on the PC is under the control of the user and sends data packets to the Ethernet controller to turn the lamps ON or OFF.

The PC Program

The PC program is developed using the Visual Basic 6.0 programming language (www.microcoft.com). Microsoft Winsock ActiveX control is used to develop the UDP program on the PC. **Figure 6** shows the GUI form of the PC program. Clicking command buttons Lamp X or Lamp Y toggles the states of lamps X or Y respectively. Similarly, clicking command button Lamp X, Lamp Y toggles the state of both lamps X and Y. Two lights under the command buttons are updated regularly to indicate the status of the lamps, where green indicates ON and red indicates OFF.

The following data is sent to the Ethernet controller when a command button is clicked. Characters "S" and "#" denote the beginning and ending of the command data respectively:

Command	Lamp	Description
SX#	Lamp X	Toggle Lamp X
SY#	Lamp Y	Toggle Lamp Y
SA#	LampX and LampY	Toggle Lamp X and Lamp Y

Similarly, the following data is sent regularly from the Ethernet controller to the PC to indicate the status of the two lamps. Here again "S" and "#" denote the beginning and ending of the data respectively. "x" and "y" can be "0" (OFF), or "1" (ON) and they indicate the status of the two lamps X and Y respectively:

SX=n,Y=m#

Figure 7 shows the program listing of the PC program. When the form is loaded, UDP protocol is selected and remote host IP address is set to "192.168.1.11". Procedures CmdLampX_Click, CmdLampY_Click and CmdLampXY_Click are activated when command buttons Lamp X, Lamp Y and Lamp XY are clicked respectively. Procedure WinSock_DataArrival is activated whenever a UDP packet is received by the program. Here, the status of the two LEDs are updated accordingly on the GUI form.

```
Private Sub CmdExit_Click()
End
End Sub
Private Sub CmdLampX_Click()
WinSock.SendData "SX#"
End Sub
Private Sub CmdLampY_Click()
WinSock.SendData "SY#"
End Sub
Private Sub CmdLampXY_Click()
WinSock.SendData "SA#"
End Sub
Private Sub Form_Load()
LampXLED.Value = False
LampYLED.Value = False
WinSock.Protocol = sckUDPProtocol
WinSock.RemoteHost = "192.168.1.11"
WinSock.RemotePort = 10001
End Sub
Private Sub WinSock_DataArrival(ByVal bytesTotal As Long)
WinSock.GetData stat, vbString

If Left$(stat, 1) = "S" And Mid$(stat, 9, 1) = "#" Then
If Mid$(stat, 2, 2) = "X=" Then
If Mid$(stat, 4, 1) = "1" Then
LampXLED.Value = True
ElseIf Mid$(stat, 4, 1) = "0" Then
LampXLED.Value = False
End If
End If
If Mid$(stat, 6, 2) = "Y=" Then
If Mid$(stat, 8, 1) = "1" Then
LampYLED.Value = True
ElseIf Mid$(stat, 8, 1) = "0" Then
LampYLED.Value = False
End If
End If
End If
End Sub
```

Figure 7: PC program listing

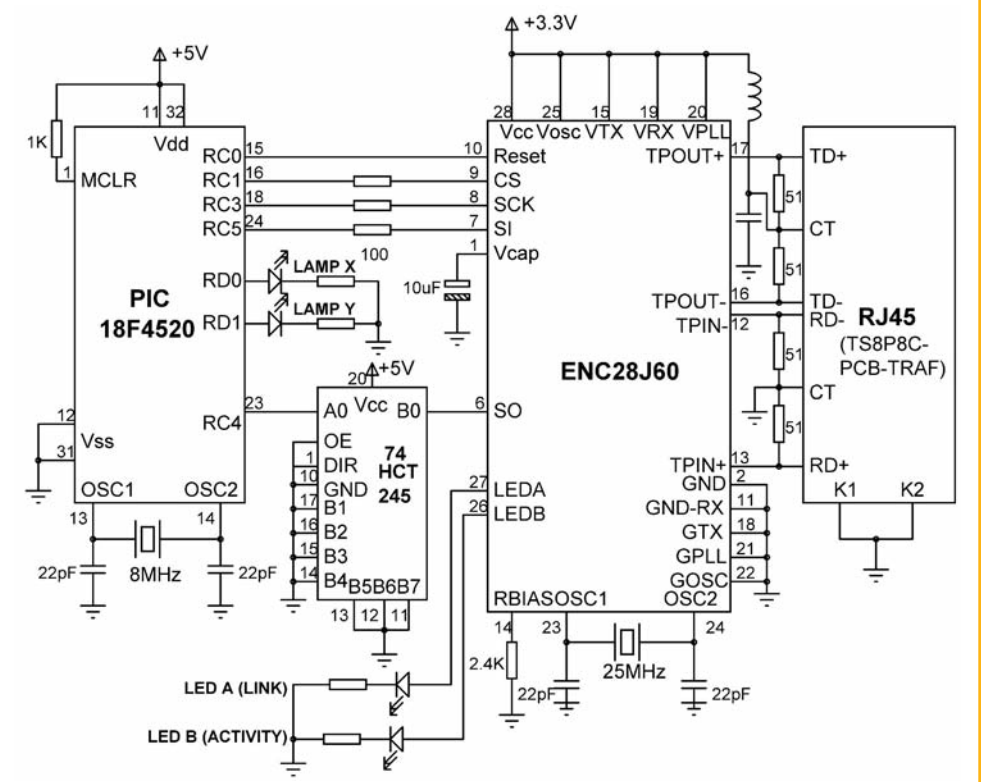

Figure 8: *Circuit diagram of the Ethernet controller*

Figure 8 shows the circuit diagram of the Ethernet controller. The controller is designed around the PIC18F4520 microcontroller, operating at 8MHz and the ENC28J60 Ethernet chip, operating at 25MHz. The interface between the microcontroller and the Ethernet chip is based on the SPI bus protocol (www.zone.ni.com/devzone/cda/tut/p/id/9119) where the SI, SO and SCK pins of the Ethernet chip are connected to SPI pins (PORT C) of the microcontroller. The Ethernet chip operates with 3.3V and its output pins (SO) cannot drive the microcontroller inputs without a unidirectional voltage translator. In Figure 8, a 74HCT245 type buffer is used to boost the output signal level of pin SO. Other low-cost chips such as 74HCT08 (AND gate), 74ACT125 (quad 3-state buffer) or other chips could also have been used instead.

The internal analogue circuitry of the ENC28J60 chip requires that an external resistor is connected from RBIAS to ground. Some of the device's digital logic operates at 2.5V and an external filter capacitor should be connected from Vcap to ground. Lamp X and Lamp Y are connected to RD0 and RD1 pins of the microcontroller respectively. Transmit output pins of the Ethernet chip (TPOUT+ and TPOUT-) and the receive inputs (TPIN+ and TPIN-) are connected to a RJ45 socket with integrated Ethernet transformer (T58P8C-PCB-TRAF). LED A and LED B of the Ethernet chip provide visual indication of the Link and Activity on the line respectively (the RJ45 socket has a pair of built in internal LEDs, but are not used in this project). A 5V to 3.3V power supply regulator chip (e.g. MC33269DT-3.3) is used to provide power to the Ethernet chip. If the PC and the Ethernet controller are on the same network then the two can be connected together using a twisted network cable, otherwise, a hub or a switch may be required. If the PC and the Ethernet controller are located on different networks then routers may be required to establish communication between the two.

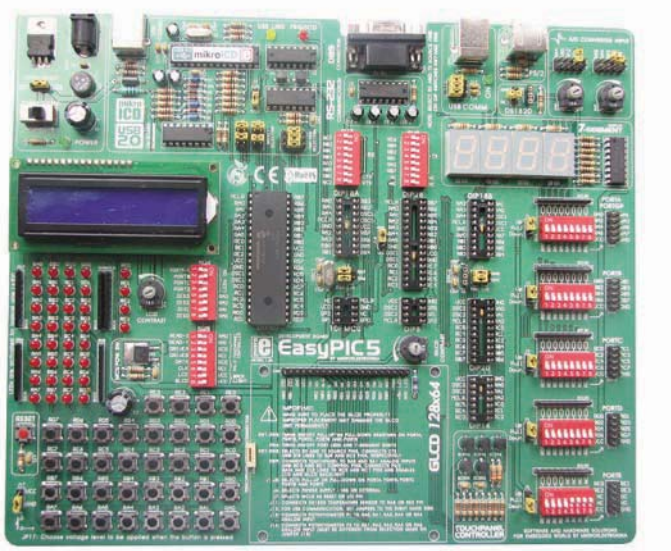

Figure 9: *EasyPIC 5 development board*

The Construction

The project was constructed using the EasyPIC 5 Development Board (see **Figure 9**) and the Serial Ethernet Board (www.mikroe.com). EasyPIC 5 is a fully integrated development board where a program can be developed and compiled on a PC and then downloaded to a PIC microcontroller via the on-board USB programmer. The board also incorporates in-circuit debugger hardware, making it easy to debug software during the development cycle.

The Serial Ethernet Board (see **Figure 10**) simplifies the design of Ethernet-based applications. The board is equipped with an ENC28J60 Ethernet chip, 74HCT245 for voltage level translation, three LEDs, RJ45 socket with integrated transformer and LEDs and a 5V to 3.3V voltage regulator. A 10-way IDC socket is provided at the edge of the serial Ethernet board and this can be directly connected to the 10-way PORT C plug provided at the edge of EasyPIC 5 development board (see **Figure 11**).

The Software

The software of the Ethernet controller is developed using the highly popular mikroC language. It provides built-in libraries for peripheral devices such as RS232, RS485, SD card, Compact Flash card, USB, CAN, Serial Ethernet, LCD, GLCD, touch screen and so on. Development of the controller software using the built in Serial Ethernet Library routines was extremely simple. **Figure 12** shows operation of the controller software as a PDL. The software is based on the UDP protocol. After the basic initialization, the software looks for command data on the line, decodes this data and turns the lamps ON and OFF as requested.

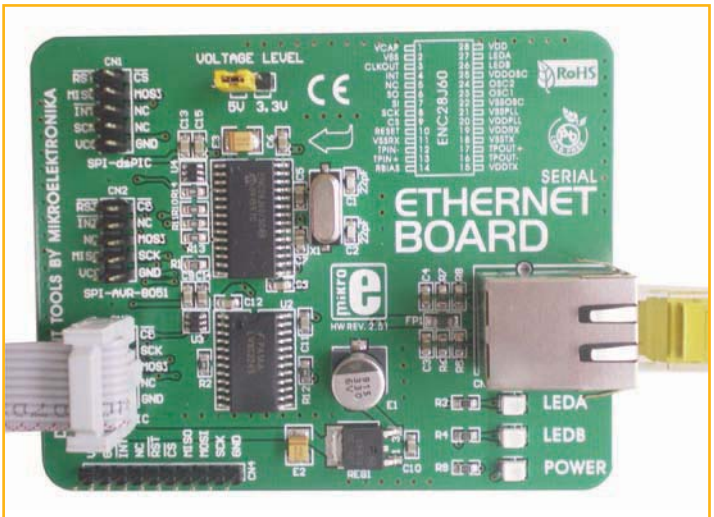

Figure 10: *Serial Ethernet board*

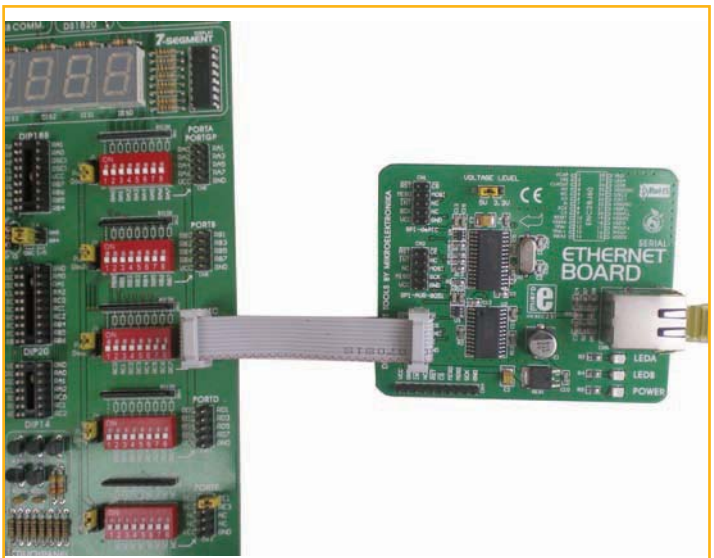

Figure 11: *Serial Ethernet board connection to EasyPIC 5*

MAIN Program
BEGIN
Configure I/O ports
Initialize SPI bus
Initialize serial Ethernet library
DO FOREVER
Check for packets
ENDDO
END

User UDP Code
BEGIN
IF a correct packet received
Decode the packet
Turn appropriate LEDs ON or OFF
ENDIF
Get status of LEDs
Send status of LEDs
END

Figure 12: *Operation of the controller software*

The operation of the Ethernet controller software is very simple and its complete listing is given in **Figure 13**. At the beginning of the main program the I/O ports are configured and the SPI bus is initialized by calling to built-in library function SPI_Init. Then, the serial Ethernet library is initialized by calling to function SPI_Ethernet_Init and specifying the MAC address, the IP address and the mode of operation (Full-Duplex).

The main program then enters an infinite loop where function SPI_Ethernet_doPacket is called to check for the arrival of packets and also to send any outstanding packets. UDP packet arrival and sending outstanding packets are handled in procedure SPI_Ethernet_UserUDP. Here, UDP data is received by calling to function SPI_Ethernet_getByte and the received command data is decoded and the appropriate LEDs are turned ON or OFF. In addition, the state of the two LEDs are read and sent to the PC using function SPI_Ethernet_putBytes.

```
sbit LampX at RD0_bit;
sbit LampY at RD1_bit;
sfr sbit SPI_Ethernet_Rst at RC0_bit;
sfr sbit SPI_Ethernet_CS at RC1_bit;
sfr sbit SPI_Ethernet_Rst_Direction at TRISC0_bit;
sfr sbit SPI_Ethernet_CS_Direction at TRISC1_bit;

unsigned char MACAddr[6] = {0x00,0x14,0xA5,0x76,0x19,0x3F};
unsigned char IPAddr[4]= {192,168,1,11};
unsigned char buffer[3];

typedef struct {
 unsigned canCloseTCP: 1;
 unsigned isBroadcast: 1;
} TEthPktFlags;

unsigned int SPI_Ethernet_UserTCP(unsigned char *remoteHost,
unsigned int remotePort, unsigned int localPort, unsigned int
reqLength, TEthPktFlags *flags)
 {
 return (0);
 }

unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost,
unsigned int remotePort, unsigned int destPort, unsigned int
reqLength, TEthPktFlags *flags)
{
 unsigned char i,x,y;
 unsigned char tx[] = "SX=1,Y=0#";
 for(i=0; i<3; i++)
 {
 buffer[i] = SPI_Ethernet_getByte();
 }
 if(buffer[0] == 'S' && buffer[2] == '#')
 {
 switch (buffer[1])
 {
 case 'X': LampX = ~ LampX;
 break;
 case 'Y': LampY = ~LampY;
 break;
 case 'A': LampX = ~LampX; LampY = ~LampY;
 }
 }
```

```
 x = LampX; y = LampY;
 x = x+'0'; y = y+'0';
 tx[3] = x; tx[7] = y;
 SPI_Ethernet_putBytes(tx, 9);
 return(9);
}

void main()
{
 TRISD = 0;
 CMCON |= 0X07;
 PORTD = 0x00;
 SPI_Init();
 SPI_Ethernet_Init(MACAddr,IPAddr,0x01);

 while(1)
 {
 SPI_Ethernet_doPacket();
 }
}
```

Figure 13: Controller software

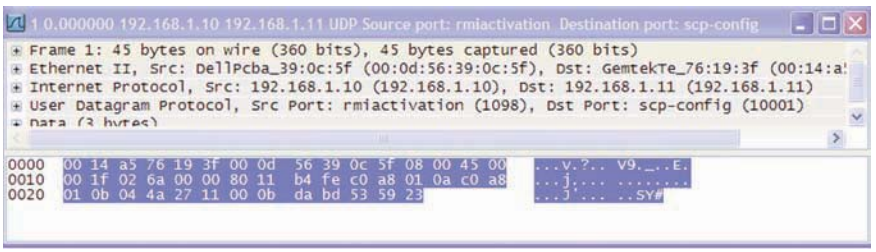

Figure 14: *Packet when command "SY#" is sent from the PC*

Figure 14 shows contents of a packet captured using the Wireshark (www.wireshark.com) data capturing software when command "SY#" was sent from the PC to toggle Lamp Y. The total packet consists of 45 bytes. The first 12 bytes are the destination and source addresses respectively. Two bytes 08 00 denote that this is an IP packet. The UDP data occupies the last 11 bytes of the packet where:

04 4a	**UDP source port**
27 11	**UDP destination port**
00 0b	**Message length (11)**
da bd	**Checksum**
53 59 23	**Data (SY#)**