STEP BY STEP IMPLEMENTATION OF ETHERNET AND TCP-IP PROTOCOLS ON AN EMBEDDED SYSTEM - Part I

Tariqul Haque, PhD
Electrical Engineering Technology
Southeastern Community College
Whiteville, NC 28472

Michael Urbaniak
Ebtron, Inc
Loris, SC 29569

ABSTRACT

Integration of current internet and networking technology with a microcontroller in an advanced microprocessor class is a useful, and practical learning tool . In the first part of this paper we present hardware interfacing of an ethernet controller chip with a microcontroller. Then, in a tutorial fashion, detailed instructions are presented for the development of a driver for the ethernet system, which includes packet reading and packet writing from and to buffers of the ethernet controller chip. Part II of this paper deals with protocol implementations.

INTRODUCTION

Ethernet technology has become a dominant physical and data link layer connection technique for computer networks and the internet. But gradually this technology is also finding applications for data and information retrieval in industrial, medical, home automation, building automation, and other devices which utilize microcontrollers. Embedded systems, whose applications commonly include sensor data and sensor control, are generally kept in places which are difficult for frequent physical access but readily accessible through ethernet connections. Because of extensive use of ethernet in connecting multiple devices for information transmission, its speed, and ease of implementation, the technology is now being used in embedded systems. Implementation on an embedded system of ethernet hardware and protocols for data transfer is not as straightforward as on a personal computer, which are more prevalent in most publications. This is caused by limited resources, such as low speed, smaller size of data and address bus, permanent storage capacity, temporary storage capacity of microcontroller systems.

In this paper we first describe integration of ethernet hardware in a microcontroller system, then review basic software necessary to drive the hardware, and, finally, using an example of a HTTP server, we implement TCP-IP software protocols for information exchange.

ETHERNET HARDWARE

The heart of ethernet hardware is presently a single chip: the ethernet controller chip. There are several 16-bit ethernet chips now available commercially which basically perform identical tasks and which conforms with IEEE 802.3 requirements. In our design, we have used a 16 bit ISA bus compatible ethernet chip, CS8900A, from Cirrus Logic Inc. The CS8900A is fully compliant with the IEEE ethernet standard; it supports full duplex operation; it has built-in separate transmit and receive buffer, 4K RAM; and it incorporates a 10BASE-T transceiver.[1] Although the chip is capable of handling 16-bit data, for a 8-bit embedded system, only 8 bit of the ethernet chip are utilized by grounding the higher 8 data pins (D8-D15). Only 4 address pins (A0-A3), out of 20, are used for addressing 16 internal registers, where two registers working as a pair accept 2 bytes. In addition to these 4 address pins, 3 control pins are also required for generation of proper timing signals. These control pins are labeled as IO_R (Read timing), IO_W (Write timing) and AD_EN (Address Enable timing). An 8-bit port of any microcontroller can effectively service these 4 address signals and 3 control signals, and a second 8-bit port can serve the 8-bit data bus of the ethernet chip. A microcontroller with two available 8-bit I/O ports can generate all

necessary input and output signals for this ethernet chip. Figure 1 illustrates connections of a CS8900A chip to a 8-bit microcontroller.

The only external part, except for a few capacitors and pull-up resistors, is a 10BASE-T transformer which provides connections with a

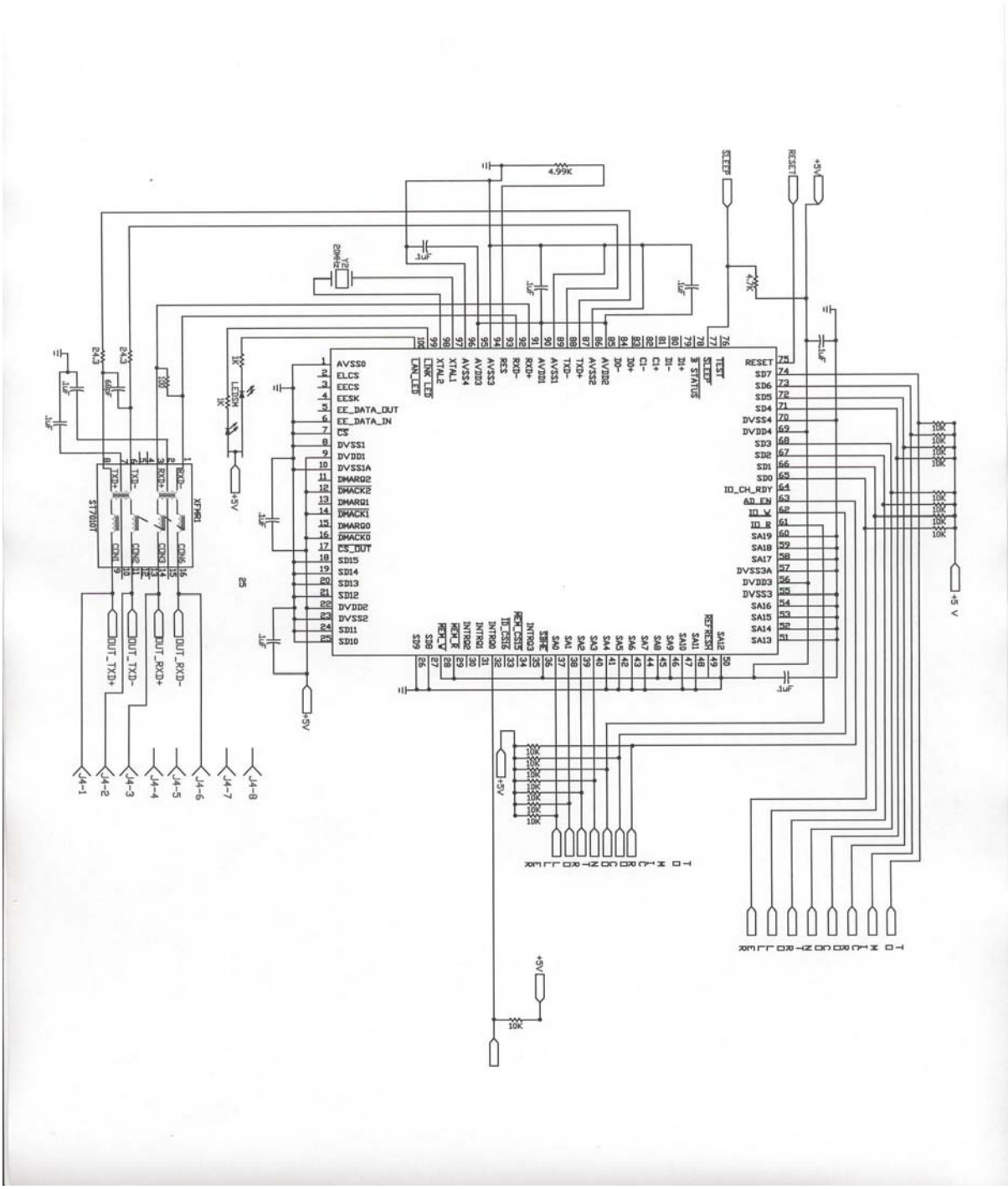

Figure 1: Circuit diagram to interface a CS8900A ethernet controller chip with a microcontroller. Hardware connections with only 8 data lines, 4 address lines and 3 control lines are required for the operation of the chip.

RJ45 connector. When the ethernet chip CS8900A is used with an 8-bit embedded system, a few functions of the chips are sacrificed. For example, in 8-bit mode, external IRQ, DMA, and EEPROM capabilities in the normal configuration of the chip are ignored. Also, I/O mode, instead of memory map mode, becomes the default and only mode with 8-bit.

REGISTERS IN ETHERNET CONTROLLER CHIP CS8900A

The ethernet chip CS8900A contains several 16-bit internal control registers, whose addresses range from 0000 to 0A00, and 16 access registers, with addresses from 00 to 0F, which are needed to write to and read from those internal control registers in 8-bit I/O mode. A few of these registers and their addresses are shown in Table 1 and Table 2. In this paper, all numbers used are in Hexadecimal (Hex) unless indicated otherwise.

More explanations of these registers can be found in reference 1. Each address of Table 2 will accept only one byte. For example, to access the internal register of address 0138, 38 is first written to address 0A and then 01 is written to 0B. Packet Page Data Register 0 (address 0C and 0D) is used for input/output of control data in internal registers; and Receive/Transmit Data Register 0 (address 00 and 01) is used for reading and writing user data from and to the buffers.

BASIC SOFTWARE REQUIRED TO DRIVE THE ETHERNET CHIP

ALGORITHM FOR SINGLE BYTE READ/WRITE

An address or a data input must be written one byte at a time. The timing diagrams of the control signals AD_EN, IO_W, and IO_R can be found in reference 1. The offset addresses of

Register	Address	Register	Address	Register	Address
IntReg	0022	LineCtl	0112	TxEvent	0128
RxCfg	0102	SelfCtl	0114	BusStat	0138
RxCtl	0104	BusCtl	0116	IAReg	0158
TxCfg	0106	TestCtl	0118		
BufCfg	010A	RxEvent	0124		

Table 1: 16-bit internal registers (control registers) and their offset addresses

Address	Description
00-01	Receive/Transmit Data Register 0
02-03	Receive/Transmit Data Register 1
04-05	TxCMD; Transmit Command Register
06-07	TxLength; Transmit Data Length Register
08-09	Interrupt Status Queue Register
0A-0B	Packet Page Pointer: Address Pointer Register
0C-0D	Packet Page Data Register 0
0E-0F	Packet Page Data Register 1

Table 2: In I/O mode these sixteen 8-bit registers(access registers) are used to access the internal registers. For this reason, only 4 address pins of CS8900A have physical connections with the microcontroller. Base address is hardwired to 0300 and here only offset addresses are shown.

the ethernet chip are only 4-bit long which are generally connected to 4 pins of a microcontroller port, they will be referred to as Address Lines. The 8 data pins of the CS8900A, connected to a 8-bit port of the microcontroller , which will be referred to as Data Port, make it possible to place the data byte on the 8-bit port and then send the control signals through another port. The order in which the commands for writing a byte to a 4-bit address are executed are given in the algorithm. All algorithms in this paper are shown in the form of sequential instructions for the advantage of programmers.

- Place 8-bit Data (or half of an address) on the Data Port of the microcontroller.
- Place 4-bit Address of an access register on the Address Lines
- Introduce a delay of minimum 125 nsec
- Pull AD_EN pin LOW
- Pull IO_W pin LOW
- Introduce a delay of minimum 125 nsec
- Pull IO_W pin HIGH
- Pull AD_EN pin HIGH

The order of commands for reading a byte from a register are

- Place the 4-bit address of a access register on the Address Lines
- Pull AD_EN pin LOW
- Pull IO_R pin LOW
- Introduce a delay of minimum 250 nsec
- Read 8-bit data from the Data Port
- Pull IO_R pin HIGH
- Pull AD_EN pin HIGH

The time delay above depends on the speed of microcontrollers. For example, for a 29.49 MHz microcontroller, a delay of 125 nsec and 250 nsec is sufficient for writing and reading respectively.

ALGORITHM FOR 2-BYTE READ/WRITE

It is often necessary, for initialization process, to read and write 2-byte long data into registers whose addresses are also by 2-byte long.. But address pins A8 and A9 are hardwired to 1, creating a permanent address of x3xx. For this reason only the offset addresses (00 to 0F) of the access registers are used in all algorithms. The sequence of commands for writing a 2-byte data into a 2-byte address will have the following steps in the given order.

- Write lower byte of the Address to lower part of the Packet Page pointer (0A)
- Write higher byte of the Address to higher part of the Packet Page pointer (0B)
- Write lower byte of the Data to lower part of the Packet Page Data Register 0 (0C)
- Write higher byte of the Data to higher part of the Packet Page Data Register 0 (0D)

For example, writing a data ABCD to an address 1234 is accomplished in 4 steps:
Write 34 to access register address 0A
Write 12 to access register address 0B
Write CD to access register address 0C
Write AB to access register address 0D

An Algorithm for Reading a 2-byte data from a 2-byte address will have the following steps.

- Write lower byte of the address to 0A
- Write higher byte of the address to 0B
- Read the lower byte of the data from address 0C
- Read the higher byte of the data from address 0D

In all these commands reading and writing of single bytes are required. The procedures for such operations are described in an earlier section.

INITIALIZATION OF THE ETHERNET CHIP

We now describe an algorithm to set up the ethernet chip with various parameters so that an error free data exchange will take place through the ethernet port. Initialization of the chip consists of a 10 msec delay after Reset or power on Reset and writing several 2-byte words in control registers. All transmissions and receptions by the chip takes place as 2-byte (16-

bit) data. Which is accomplished with a 8-bit data bus by sending, with a few exceptions, the lower byte first, followed by the higher byte. One possible initialization sequence of the chip after power on is summarized below with comments within square brackets.

- Write 0x0040 to SelfCtl register. [Software Reset of the chip]
- 10 millisecond delay
- Write 0x0000 to LineCtl register. [Hardware is ethernet]
- Write 0x4000 to TestCtl register. [Full Duplex operation]
- Write 0x0100 to RxCfg register. [Generate interrupt for a good received frame]
- Write 0x0D00 to RxCtl register: [Accept broadcast frame, accept only if MAC numbers match, accept only if CRC is correct, and data length is valid]
- Write 0x8100 to TxCfg register. [Allow 16 collisions, generate interrupt when a frame is transmitted]
- Write 6-byte MAC number to IAReg register. [These 6 bytes are written as three 2-byte words in three attempts. For example, a MAC number 12-34-56-78-9A-BC is written as 9ABC, 5678, and 1234 into addresses IAReg, IAReg+2, IAReg+4 respectively]
- Write 0x0000 to IntReg register. [Use INTRQ0]
- Write 0x8500 to BufCfg register. [Generate interrupt if MAC address is accepted, interrupt if a frame is missed, interrupt if ethernet chip is ready to accept a frame from the embedded system]
- Write 0x00C0 to LineCtl register. [Enable Transmitter and Receiver]
- Write 0x8000 to BusCtl register. [Enable IRQ]

This completes the initialization process of the ethernet chip CS8900A. More explanation of the hex control bytes sent to the registers is found in reference 1. A MAC number is a unique identification number of a node in a network.

ALGORITHM FOR READING A RECEIVED FRAME

So far we have described the procedures to read and write bytes to ethernet access registers, to read and write 2-byte data in control registers, and to show how the latter is used to initialize the ethernet chip. The reception of a ethernet packet is completely handled by the ethernet chip in the background and is stored in the built-in Receive Frame Buffer. We now present an algorithm to read a complete packet from the ethernet chip by the microcontroller. All single byte addresses belong to access registers and all double byte addresses belong to internal control registers.

Repeatedly perform
- Read RxEvent register [A 2-byte read command. Bit 8 indicates reception of a packet]

Until bit 8 of RxEvent is 1.
- Read a byte from address 01. [Higher byte of RxStatus register]
- Read a byte from address 00. [Lower byte of RxStatus register]
- Read a byte from address 01. [Higher byte of Data Length of the received packet]
- Read a byte from address 00. [Lower byte of Data Length]

Now repeatedly perform
- Read a byte from address 00. [Lower byte of a frame data]
- Read a byte from address 01. [Higher byte of the data]

Until all data are read.

The higher byte must be read first for both Status and Data Length registers. The last two Read statements must be performed (Data Length)/2 times because each pair of read statement retrieves 2 bytes of data. In most embedded systems, scarcity of RAM prevents users from reading all bytes indicated by Data Length. In such situations, only the bytes needed to form a response may be read.

ALGORITHM FOR TRANSMITTING A PACKET

Once a packet is written into the transmission buffer of the ethernet chip, the chip handles all necessary transmission procedures internally, completely unattended by the microcontroller. When the last byte of a packet is transferred to the ethernet chip, the microcontroller is set free and the CS8900A chip takes over the remaining task of transmitting the packet over the network media. The algorithm to write a complete packet to the ethernet chip by the microcontroller consists of following sequence of instructions.

- Write C0 to address 04. [Writes lower byte of transmission command 00C0 to TxCMD register.
- Write 00 to address 05. [Writes higher byte of transmission command to TxCMD register]
- Write lower byte of packet length to address 06. [Lower byte written to TxLength register]
- Write higher byte of packet length to address 07. [Higher byte written to TxLength register]

Now repeatedly perform

- Read BusStat Register. [2-byte read command. Bit 8 indicates CS8900 is ready for a packet]

Until bit 8 of BusStat register is 1

Now repeatedly perform

- Write lower byte of a frame data to address 00. [Lower byte is written to Receive/Transmit Data 0]
- Write higher byte of the data to address 01. [Higher byte is written to Receive/Transmit Data 0]

Until all bytes are loaded into buffer.

Now repeatedly perform

- Read TxEvent Register. [2-byte read command. Bit 8 indicates all bytes are transmitted]

Until bit 8 of TxEvent register is 1.

The command byte 00C0 above sets up the ethernet chip to append ethernet CRC, to start transmission after all bytes are loaded into the buffer, and to add padding bytes if length of the packet is under 60. The length of the entire packet must be entered into Transmit Length Register before data are sent to the ethernet chip. To transmit all bytes the repeated operations must be performed (packet length)/2 times.

COMMENT

The methods presented in section 3 are sufficient to initialize the ethernet chip and issue commands necessary in the development of most ethernet based communication protocols. An ethernet controller chip, such as CS8900A, receives the preamble bits, start bits, and CRC bytes of an ethernet packet[2], but discards them when data are read by a microcontroller. Although MAC numbers must be obtained from IEEE, the MAC number of an unused Network Interface Card (NIC) will suffice for experimentation. 8-bit microcontrollers are abundant in educational curricula as well as in industry. Because of hardware and software simplicities of 8-bit designs, and the limitless application potentials, implementation of a 8-bit ethernet system may be preferable in many intermediate-to-advanced microprocessor courses. In Part-II of this paper we will address the problem of developing protocol software for this ethernet system.

REFERENCES

1. Product Data Sheet, CS8900A, Cirrus Logic, Inc. www.cirrus.com

2. Nathan Muller, Networking A to Z, McGraw-Hill, 2003.

3. Jeremy Bentham, TCP/IP Lean, CMP Books, 2000.