WIZnet iMCU Design Contest 2009
Budget Network Attached Storage Unit
By Lindsay Meek

ABSTRACT

This project uses the iMCU W7100 capabilities to connect an IDE Hard Drive to a 100MB/s LAN. The project is intended to be a low-cost solution, only using the core W7100 circuitry and two level-shifting TTL latches to implement the design. The lightweight ATA-Over-Ethernet (ATAOE) protocol is used to move the data over the LAN at high speed with the minimum of software overheads.

TABLE OF CONTENTS

BACKGROUND

NAS units traditionally tend to be rather expensive solutions to adding additional storage to a network. They usually contain an Embedded PC running a Linux distribution, which shares the internal hard-drives using a series of open-source protocols like NFS, SAMBA or FTP. This gives the NAS developers the advantage of using off-the-shelf components at the expense of the hardware cost. The proposed Budget NAS does away with the complexity of the LAN protocols by making use of the increasingly popular ATAOE protocol [1].

The ATAOE protocol allows simple and lightweight network file systems to be constructed, by passing the raw ATA requests and responses over a LAN using a special Ethernet packet identifier. The ATAOE protocol functions at the same protocol level as the wildly popular Internet Protocol, however it does away with a lot of unnecessary complexity and as a result can be implemented easily and gives surprisingly good performance on modest hardware.

The ATAOE drives themselves are identified by two fixed addresses; their “shelf” and “slot”, which is somewhat similar to the SCSI addressing. An ARP-style broadcast packet is sent over the LAN when a client is attempting to find a drive, and the drive will respond if it has a matching shelf and slot number. The drive can then mounted on the client and subsequent ATA requests are directed at the drive’s Ethernet MAC address.

The typical ATA commands that need to be supported as a minimum are IDENTIFY DEVICE, READ SECTOR and WRITE SECTOR. ATAOE clients are freely available on the internet for Linux and Windows distributions [3],[4].

Another touted advantage of ATAOE is its security, as routers don’t currently recognise the protocol type and will not route packets between broadcast isolated subnets. This could however be a disadvantage particularly with Wi-Fi routers, and future routers should have a facility to reroute these packets perhaps using an approach similar to RIP.

PROJECT DESCRIPTION

The ATAOE adaptor operates by connecting a legacy parallel IDE hard drive to the spare I/O of the W7100. The bare minimum I/O for PIO mode is the three address lines, the sixteen data lines, as well as the read strobe, write strobe and chip select lines. The IDE specification supports TTL levels so the 3.3V logic outputs of the W7100 is suitable for directly connecting to the hard drive. However the data lines can be driven at 5V from the hard drive, so a pair of level translating latches were needed to convert between 3.3V and 5V to avoid possible damage to the IC. These latches as well as the connector adaptors form the majority of the additional hardware required.

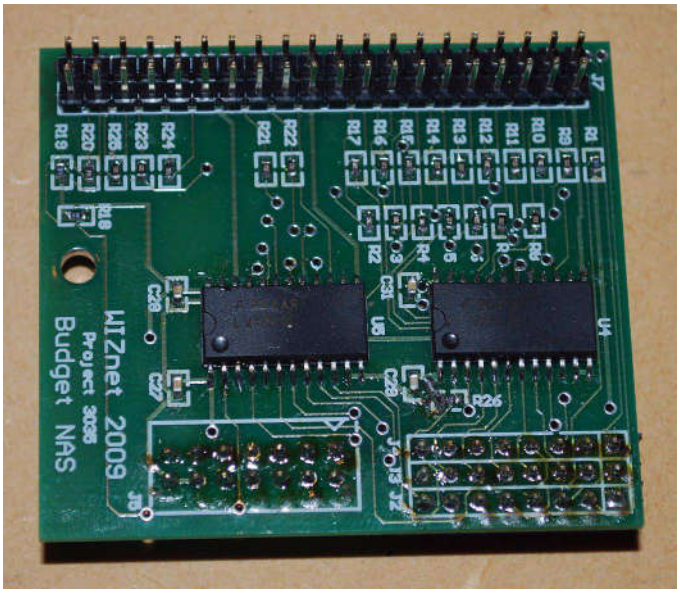


Figure 1. Level Translating Latch PCB with IDE 40-way connector

The software for the ATAOE driver running on the W7100 makes use of the MACRAW mode of socket operation. This mode sees the onboard Ethernet hardware pass unprocessed incoming Ethernet frames to the ATAOE driver. The ATAOE driver decodes the Ethernet frames and if required, converts them into ATA commands and sends them down to the Hard Drive. Data transfers using ATAOE usually sends two 512-byte sectors at a time, so as to keep the Ethernet packet length below 1500 bytes [2]. Any data coming back from the Hard Drive is packaged into an Ethernet response and sent to the remote end.

RESULTS AND FURTHER WORK

The software attached to the Appendix was able to operate with the Starport ATAOE client driver [4]. It detected the hard drive, mounted it and then displayed the directories under windows as expected. During the course of testing, it was found that the I/O ports on the W7100 used weak internal pull-ups with open-collector drivers to implement the bidirectional I/O ports. This had the unwanted side effect of slowing the slew rate when driving from a low to a high output, meaning the driver had to be artificially slowed down in order for all of the PIO accesses to the hard drive to complete in a reliable manner. In a future revision of this board, external pull-ups with lower resistance values should be fitted to the I/O lines to increase their slew rate and thus the speed of the PIO accesses.

Wiznet may also consider extending their silicon to accelerate the transfer of data to a hard drive, for example by implementing a DMA-style 16-bit burst mode for moving the sector data to and from the internal memory. Alternatively, hardware support for SATA or USB OTG interfaces would greatly accelerate the transfer rate.

PROJECT PHOTOS

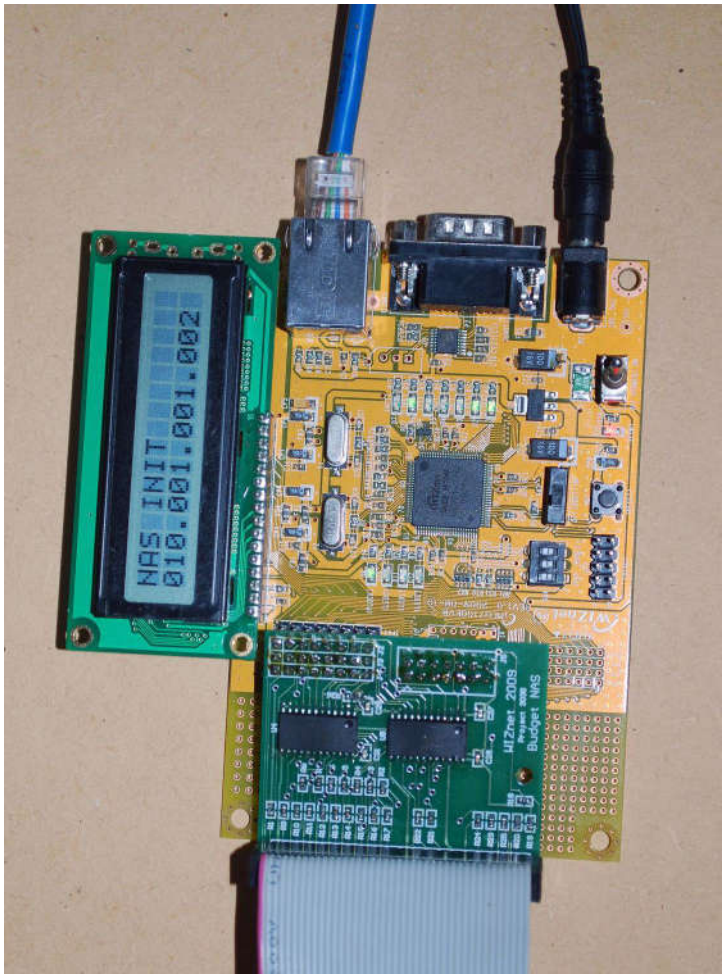
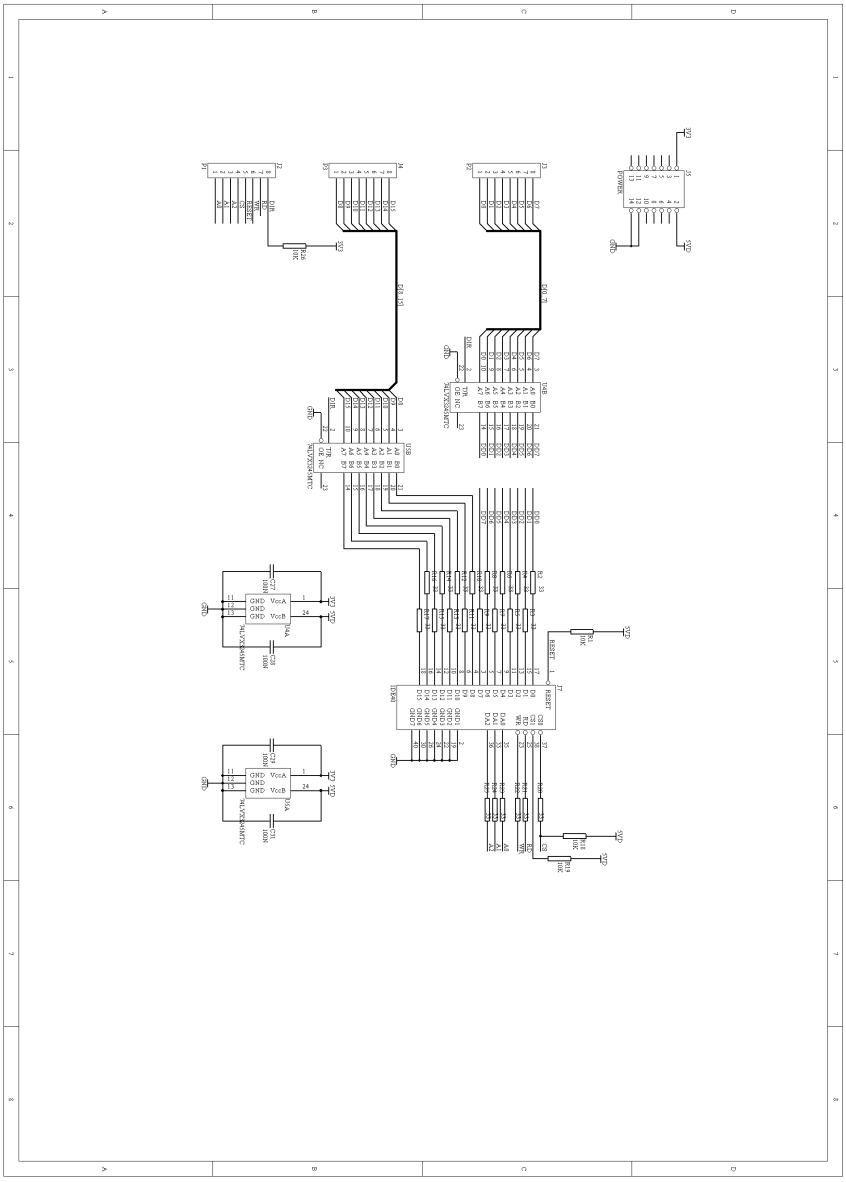


Figure 2. Unit Operating

Figure 3. Unit Operating with IDE HDD

ADAPTOR SCHEMATIC



SOURCE CODE

MAIN PROGRAM LOOP

```
/*

Circuit Cellar WizNet Contest 2010
Project 3035
Budget Network Attached Storage Unit

*/


#include <stdio.h>
#include "lcd.h"
#include "serial.h"           // serial related functions
#include "socket.h"           // W7100A driver file
#include "types.h"
#include "w7100.h"
#include "Wizmemcpy.h"
#include "TCPIPcore.h"
#include "delay.h"
#include "ataoe.h"

void Init_iMCU();
void Init_Network();

uint8 xdata ip[4] = {10,1,1,2};                   // for setting SIP register
uint8 xdata gw[4] = {10,1,1,1};                     // for setting GAR register
uint8 xdata sn[4] = {255,255,255,0};                     // for setting SUBR register
uint8 xdata mac[6] = {0x00,0x08,0xDC,0x00,0x00,0x00};      // for setting SHAR
register

void main()
{
        init_ata_hardware();

        Init_iMCU();          // Initialize   iMCUW7100
        lcd_init();
        evb_set_lcd_text(0,(uint8 *) "NAS INIT");
        Init_Network();       // Initialize   Network Configuration

        //start message
        printf("\r\n-------ATAOE DRIVER START--------\r\n");

        ataoe(0, &mac);

        printf("\r\n-------ATAOE DRIVER END--------\r\n");

        while(1);

}

void Init_iMCU(void)
{
        uint8 xdata i;

        EA = 0;                       // Disable all interrupts
        CKCON = 0x02;         // External Memory Access Time
        WTST = 0x03;
#ifdef __DEF_IINCHIP_INT__
        IINCHIP_WRITE(IMR,0xff);
        for(i=0;i<MAX_SOCK_NUM;i++) IINCHIP_WRITE(Sn_IMR(i),0x1f);
        IINCHIP_ISR_ENABLE();
#endif
        InitSerial();  // Initialize serial port (Refer to serial.c)
        EA = 1;
}

void Init_Network(void)
{
```

```
        uint8 xdata str[17];

        uint8 xdata txsize[MAX_SOCK_NUM] = {8,1,1,1,1,1,1,1};
        uint8 xdata rxsize[MAX_SOCK_NUM] = {8,1,1,1,1,1,1,1};
        // uint8 xdata txsize[MAX_SOCK_NUM] = {2,2,2,2,2,2,2,2};
        // uint8 xdata rxsize[MAX_SOCK_NUM] = {2,2,2,2,2,2,2,2};

        /* Write MAC address of W7100 to SHAR register */
        IINCHIP_WRITE (SIPR0+0,ip[0]);
        IINCHIP_WRITE (SIPR0+1,ip[1]);
        IINCHIP_WRITE (SIPR0+2,ip[2]);
        IINCHIP_WRITE (SIPR0+3,ip[3]);

        /* Write GATEWAY of W7100 to GAR register */
        IINCHIP_WRITE (GAR0+0, gw[0]);
        IINCHIP_WRITE (GAR0+1, gw[1]);
        IINCHIP_WRITE (GAR0+2, gw[2]);
        IINCHIP_WRITE (GAR0+3, gw[3]);
        /* Write SUBNETMASK of W7100 to SURR register */
        IINCHIP_WRITE (SUBR0+0, sn[0]);
        IINCHIP_WRITE (SUBR0+1, sn[1]);
        IINCHIP_WRITE (SUBR0+2, sn[2]);
        IINCHIP_WRITE (SUBR0+3, sn[3]);
        /* Write MAC address of W7100 to SHAR register */
        IINCHIP_WRITE (SHAR0+0, mac[0]);
        IINCHIP_WRITE (SHAR0+1, mac[1]);
        IINCHIP_WRITE (SHAR0+2, mac[2]);
        IINCHIP_WRITE (SHAR0+3, mac[3]);
        IINCHIP_WRITE (SHAR0+4, mac[4]);
        IINCHIP_WRITE (SHAR0+5, mac[5]);

        set_MEMsize(txsize,rxsize);

      while(!(EIF & 0x02));
        EIF &= ~0x02;

        printf( "=============================================\r\n");
        printf( "       W7100   Net Config Information         \r\n");
        printf( "=============================================\r\n");

        printf( "MAC ADDRESS IP : %.2x.%.2x.%.2x.%.2x.%.2x.%.2x\r\n",
                (int)IINCHIP_READ (SHAR0+0), (int)IINCHIP_READ (SHAR0+1),
                (int)IINCHIP_READ (SHAR0+2), (int)IINCHIP_READ (SHAR0+3),
                (int)IINCHIP_READ (SHAR0+4), (int)IINCHIP_READ (SHAR0+5)
                ) ;
        sprintf(str,"%.3d.%.3d.%.3d.%.3d",
                        (int)IINCHIP_READ (SUBR0+0), (int)IINCHIP_READ (SUBR0+1),
                        (int)IINCHIP_READ (SUBR0+2), (int)IINCHIP_READ (SUBR0+3));
        printf( "SUBNET MASK  : %s\r\n",str);

        sprintf(str,"%.3d.%.3d.%.3d.%.3d",
                        (int)IINCHIP_READ (GAR0+0), (int)IINCHIP_READ (GAR0+1),
                        (int)IINCHIP_READ (GAR0+2), (int)IINCHIP_READ (GAR0+3));

        printf( "G/W IP ADDRESS : %s\r\n",str);

        sprintf(str,"%.3d.%.3d.%.3d.%.3d",
                        (int)IINCHIP_READ (SIPR0+0), (int)IINCHIP_READ (SIPR0+1),
                        (int)IINCHIP_READ (SIPR0+2), (int)IINCHIP_READ (SIPR0+3));

        printf( "LOCAL IP ADDRESS : %s\r\n", str);

      //display IP
      evb_set_lcd_text(1,str);

}
```

ATAOE DRIVER

```
/*

Circuit Cellar WizNet Contest 2010
Project 3035
Budget Network Attached Storage Unit
ATAOE driver

 */

#include "ataoe.h"
#include "socket.h"
#include "w7100.h"
#include "Wizmemcpy.h"
#include "TCPIPcore.h"
#include "delay.h"
#include "sockutil.h"
#include <stdio.h>

#define MY_MAJOR 1
#define MY_MINOR 0

#define AOE_TYPE 0x88a2

#define AOE_CMD_ATA 0
#define AOE_CMD_CONFIG 1
#define AOE_CMD_MAC_MASK 2
#define AOE_CMD_RESERVE_RELEASE 3

#define ATA_CMD_PIO_EREAD            0x24
#define ATA_CMD_PIO_READ             0x20
#define ATA_CMD_PIO_WRITE            0x30
#define ATA_CMD_ID_ATA               0xEC

///
///IDE status register bits
///
#define IDE_BSY               (1<<7)
#define IDE_DRDY              (1<<6)
#define IDE_DF                (1<<5)
#define IDE_DRQ               (1<<3)
#define IDE_ERR               (1<<0)

#define LBA_SIZE              4
#define AOE_TAG_SIZE          4

static uint8 xdata ide_status=0;

static AtaIssue xdata aATA _at_ 0x007000;


static void DumpAtaHdr(AtaHdr *p)
{
  printf("DEST %02X:%02X:%02X:%02X:%02X:%02X\r\n",
                              (int)p->dst_mac[0],(int)p->dst_mac[1],
                              (int)p->dst_mac[2],(int)p->dst_mac[3],
                              (int)p->dst_mac[4],(int)p->dst_mac[5]);

  printf("SRC %02X:%02X:%02X:%02X:%02X:%02X\r\n",
                              (int)p->src_mac[0],(int)p->src_mac[1],
                              (int)p->src_mac[2],(int)p->src_mac[3],
                              (int)p->src_mac[4],(int)p->src_mac[5]);

  printf("FLAGS %x\r\nMAJOR %u\r\nMINOR %u\r\nCMD %x\r\n",
                       (int)p->flags,htons(p->major),(int)p->minor,(int)p->cmd);

}

static void DumpATAIssue(AtaIssue *p)
{
        DumpAtaHdr(&p->header);

        printf("ATA CMD %X\r\n", (int)p->acmd);
```

```
        printf("SECTORS %u\r\n", (int)p->sectors);
        printf("LBA %02X:%02X:%02X:%02X:%02X:%02X\r\n",
                        (int)p->lba0,(int)p->lba1,(int)p->lba2,
                        (int)p->lba3,(int)p->lba4,(int)p->lba5);


}

static void DumpATAConfig(AtaConfig *p)
{
           DumpAtaHdr(&p->header);

           printf("Max Queue Length %u\r\n", htons(p->buffer_count));
           printf("Firmware Version %u\r\n", htons(p->firmware_version));
           printf("Max Sectors %u\r\n",(int)p->sectors);
           printf("AOE Protocol %u\n\r",(int)((p->aoe_ccmd>>4)&15));
           printf("CCMD %u\r\n",(int)(p->aoe_ccmd&15));
           printf("String length %u\r\n",htons(p->length));


}

static void DumpATAMacMask( AtaMACMask *p)
{
           DumpAtaHdr(&p->header);

}

static void DumpATAReserveRelease( AtaReserveRelease *p)
{
           DumpAtaHdr(&p->header);

}

AtaConfig xdata ConfigResp;


static uint8 SendConfigAck(SOCKET s,uint8 *src_mac,AtaConfig *req)
{
        uint8 i;
        uint32 xdata tip = 0xFFFFFFFF;

        DumpATAConfig(req);

        ConfigResp.header.msg_type = AOE_TYPE;
        ConfigResp.header.flags = 0x18;
        ConfigResp.header.error = 0;
        ConfigResp.header.major = MY_MAJOR;
        ConfigResp.header.minor = MY_MINOR;
        ConfigResp.header.cmd = AOE_CMD_CONFIG;

        ConfigResp.header.dst_mac[0] = aATA.header.src_mac[0];
        ConfigResp.header.dst_mac[1] = aATA.header.src_mac[1];
        ConfigResp.header.dst_mac[2] = aATA.header.src_mac[2];
        ConfigResp.header.dst_mac[3] = aATA.header.src_mac[3];
        ConfigResp.header.dst_mac[4] = aATA.header.src_mac[4];
        ConfigResp.header.dst_mac[5] = aATA.header.src_mac[5];

        ConfigResp.header.src_mac[0] = src_mac[0];
        ConfigResp.header.src_mac[1] = src_mac[1];
        ConfigResp.header.src_mac[2] = src_mac[2];
        ConfigResp.header.src_mac[3] = src_mac[3];
        ConfigResp.header.src_mac[4] = src_mac[4];
        ConfigResp.header.src_mac[5] = src_mac[5];

        ConfigResp.header.tag = req->header.tag;

        // Maximum # of queued commands
        ConfigResp.buffer_count = htons(1);
        ConfigResp.firmware_version= 0;

        ConfigResp.sectors=0;  // actually means 2 sectors max
        ConfigResp.aoe_ccmd=0x10;
        ConfigResp.length=0;   // no config string response

        if( sendto(s,(uint8*)&ConfigResp,sizeof(ConfigResp),(uint8 *)&tip,3000) ==0)
               return 0;
```

```
        else
                return 1;
}


static uint8 ReadReg8(uint8 reg)
{
        static uint8 t;

        reg &= 7;

        // Load address
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK) ;

        wait_1us(1);
        // CS=0
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_DIR_MASK) ;


        wait_1us(1);

        // Set the I/O pin to input (open collector with pullup)
        IDE_DATA_MSB = 0xFF;
        IDE_DATA_LSB = 0xFF;

        // DIR=0 RD=0 CS=0
        P1 = reg | (IDE_RESET_MASK|IDE_WR_MASK) ;

        // 1.25us

        wait_1us(2);

        t = IDE_DATA_LSB;

        wait_1us(1);

        // DIR=0 RD=1 CS=0
        P1 = reg | IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK ;

        wait_1us(1);

        // DIR=1 RD=1 CS=1
        P1 = reg | IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK ;

        wait_1us(1);

        return  t;
}

static void ResetBus(void)
{

        // DIR=1 RD=1 CS=1 WR=1 RESET=0
        P1 = (IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK) ;

        wait_1us(100);

        // DIR=1 RD=1 CS=1 WR=1 RESET=1
        P1 = (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK) ;

        wait_1ms(4);

}

static uint8 WaitBusy(void)
{
        uint16 i=0;
        uint8 j=0;

        do
        {

                do
                {
                        ide_status = ReadReg8(7);

                        if(!(ide_status & IDE_BSY))
```

```
                                return 1;

                } while(++i!=0);

        } while(++j != 10);

        return 0;
}

static void WriteReg8(uint8 reg,uint8 x)
{
        reg &= 7;

        // Load addr
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK) ;

        wait_1us(1);

        // CS=0 DIR=1
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_DIR_MASK) ;

        wait_1us(1);

        IDE_DATA_LSB = x;
        IDE_DATA_MSB = 0xFF;

        wait_1us(2);

#pragma ASM
                NOP
                NOP
                NOP
                NOP
                NOP
#pragma ENDASM

        // CS=0 WR=0 DIR=1
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_DIR_MASK) ;

        wait_1us(1);

#pragma ASM
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
#pragma ENDASM

        // WR=1 CS=0 DIR=1
        P1 = reg | (IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_DIR_MASK) ;

        wait_1us(1);

        IDE_DATA_LSB = 0xFF;
        IDE_DATA_MSB = 0xFF;

        wait_1us(1);

        // WR=1 CS=1 DIR=1
        P1 = reg | IDE_RESET_MASK|IDE_RD_MASK|IDE_WR_MASK|IDE_CS_MASK|IDE_DIR_MASK ;

        wait_1us(1);
}

static void DumpRegs(void)
{
        uint8 i;

        printf("Regs: ");

        for(i=1;i<8;i++)
```

```
        {
                printf("%02x ", (int)ReadReg8(i));
        }

        printf("\r\n");

}

// pointer dereferencing

static uint8 ATAHandler(SOCKET s,uint8 *src_mac)
{
        uint16 byte_cnt;
        uint8  blk_cnt;
        uint8 blocks;
        unsigned long m;
        uint32 xdata tip = 0xFFFFFFF;
        uint8 i,j,*p;

//      DumpATAIssue(&aATA);

        // Wait for interface to be free
        WaitBusy();

//      DumpRegs();

        // extended or normal access?
        if(aATA.aflag & 0x40)
        {
        //      printf("Extended\r\n");

                WriteReg8(6,(aATA.aflag & 0x50) | 0xa0);

                // Wait for device to respond
                WaitBusy();

                WriteReg8(3, aATA.lba3);        //      LBA low

                WriteReg8(3, aATA.lba0);

                WriteReg8(4, aATA.lba4);        // LBA Mid

                WriteReg8(4, aATA.lba1);

                WriteReg8(5,aATA.lba5);         // LBA High

                WriteReg8(5,aATA.lba2);

                WriteReg8(2, 0);                        // Sector count
        }
        else  // normal access
        {
        //      printf("Normal\r\n");

                WriteReg8(6, aATA.lba3);        // head

                // Wait for device to respond
                WaitBusy();

                WriteReg8(3,aATA.lba0);         //      LBA low

                WriteReg8(4,aATA.lba1);         // LBA Mid

                WriteReg8(5,aATA.lba2);         // LBA High
        }

        WriteReg8(2,aATA.sectors);              // Sector count

        WriteReg8(1,aATA.err);                  // Err/Features

//      printf("Write values %02X %02X %02X %02X %02X %02X\r\n",(int)aATA.err,
(int)aATA.sectors,(int)aATA.lba0,
//                                                      (int)aATA.lba1,(int)aATA.lba2,
(int)aATA.lba3);

//      DumpRegs();
```

```
// Set up block copy registers
m=(unsigned long)aATA.dat;
DPX1 = (uint8)(m>>16);
DPH1 = (uint8)(m>>8);
DPL1 = (uint8)(m);

p=aATA.dat;

WaitBusy();

// Writing to ATA?
if(aATA.aflag & 1)
{
        blocks=0;

    if(ide_status & (IDE_DRDY|IDE_DRQ))
    {

                // Address zero

                P1 = IDE_CS_MASK | IDE_RD_MASK | IDE_WR_MASK | IDE_RESET_MASK |
                         IDE_DIR_MASK;

                wait_1us(1);
                // CS=0
                P1 = IDE_RD_MASK | IDE_WR_MASK | IDE_RESET_MASK | IDE_DIR_MASK;

                wait_1us(1);
                blk_cnt=aATA.sectors;

                // Send the data to the device
                while(blk_cnt!=0)
                {
                        --blk_cnt;

                        byte_cnt=0;
                        do
                        {

                                IDE_DATA_LSB = *p++;
                                IDE_DATA_MSB = *p++;

                                wait_1us(1);

                                // WR=0 RD=1 RESET=1 DIR=1 CS=0
                                P1 = IDE_RD_MASK | IDE_RESET_MASK | IDE_DIR_MASK;

                                wait_1us(2);

                                // WR=1 RD=1 RESET=1 CS=0 DIR=1
                                P1 = IDE_WR_MASK | IDE_RD_MASK | IDE_RESET_MASK |
                                        IDE_DIR_MASK;

                                wait_1us(1);

                        } while(++byte_cnt != 256);
                }

                IDE_DATA_MSB=0xFF;
                IDE_DATA_LSB=0xFF;

                wait_1us(1);
                P1 = IDE_CS_MASK | IDE_WR_MASK | IDE_RD_MASK | IDE_RESET_MASK |
                         IDE_DIR_MASK;

                wait_1us(1);

                WaitBusy();
        }

}
else
{
        blocks=0;

        if(ide_status & (IDE_DRDY|IDE_DRQ))
    {
```

```
                        blocks=aATA.sectors;
                        blk_cnt=aATA.sectors;

                        // Fetch data from the device
                        while(blk_cnt!=0)
                        {
                                --blk_cnt;

                                IDE_DATA_MSB = 0xFF;
                                IDE_DATA_LSB = 0xFF;

                                // Address zero
                                P1 = IDE_DIR_MASK | IDE_CS_MASK | IDE_RD_MASK |
                                        IDE_WR_MASK | IDE_RESET_MASK;

                                wait_1us(1);

                                for(byte_cnt=256;byte_cnt!=0;--byte_cnt)
                                {

                                        // CS=0 DIR=0 RD=1 WR=1 RESET=1
                                        P1 = IDE_RD_MASK | IDE_WR_MASK | IDE_RESET_MASK;

                                        wait_1us(1);
                                        // CS=0 RD=0 DIR=0 RESET=1 WR=1
                                        P1 = IDE_WR_MASK | IDE_RESET_MASK;

                                        wait_1us(2);

                                        *p++ = IDE_DATA_LSB;
                                        *p++ = IDE_DATA_MSB;

                                        wait_1us(1);
                                        // CS=0 RD=1 DIR=0 WR=1 RESET=1
                                        P1 = IDE_WR_MASK | IDE_RD_MASK | IDE_RESET_MASK;

                                        wait_1us(1);

                                        // CS=1 RD=1 DIR=1 WR=1 RESET=1
                                        P1 = IDE_CS_MASK | IDE_WR_MASK | IDE_RD_MASK |
                                        IDE_RESET_MASK ;

                                        wait_1us(1);

                                }
                        }

        // CS=1 RD=1 DIR=1 WR=1 RESET=1
        P1 = IDE_CS_MASK | IDE_WR_MASK | IDE_RD_MASK | IDE_RESET_MASK | IDE_DIR_MASK ;

        wait_1us(1);

                                }

#if 0
        // dump contents of packet

         for(i=0;i<32;i++)
         {
                for(j=0;j<16;j++)
                {
                        printf("%02X ",(int)aATA.dat[j+(i<<4)]);
                }
                printf("\r\n");
         }
#endif

        }

        // Copy registers from device to response packet
        aATA.err = ReadReg8(1);       // Err/Features
        aATA.sectors = ReadReg8(2); // Sector count
        aATA.lba0 = ReadReg8(3);      //      LBA low
        aATA.lba1 = ReadReg8(4);      // LBA Mid
        aATA.lba2 = ReadReg8(5);      // LBA High
        aATA.acmd =ReadReg8(7);               // Status
```

```
        aATA.header.flags = 0x18;

        aATA.header.major = MY_MAJOR;
        aATA.header.minor = MY_MINOR;
        aATA.header.msg_type = AOE_TYPE;

        aATA.header.dst_mac[0] = aATA.header.src_mac[0];
        aATA.header.dst_mac[1] = aATA.header.src_mac[1];
        aATA.header.dst_mac[2] = aATA.header.src_mac[2];
        aATA.header.dst_mac[3] = aATA.header.src_mac[3];
        aATA.header.dst_mac[4] = aATA.header.src_mac[4];
        aATA.header.dst_mac[5] = aATA.header.src_mac[5];

        aATA.header.src_mac[0] = src_mac[0];
        aATA.header.src_mac[1] = src_mac[1];
        aATA.header.src_mac[2] = src_mac[2];
        aATA.header.src_mac[3] = src_mac[3];
        aATA.header.src_mac[4] = src_mac[4];
        aATA.header.src_mac[5] = src_mac[5];

        byte_cnt=sizeof(AtaIssue)-(SECTOR_SIZE*2)+(blocks<<SECTOR_SHIFT);

        return sendto(s,(uint8*)&aATA,byte_cnt,(uint8 *)&tip,3000);

}

void init_ata_hardware(void)
{

        // Set DDRs to input

        IDE_DATA_LSB=0xFF;
        IDE_DATA_MSB=0xFF;

        // CS=1 RD=1 WR=1 RESET=1 DIR=1 ADDR=0
        P1 = IDE_CS_MASK | IDE_RD_MASK | IDE_WR_MASK | IDE_RESET_MASK | IDE_DIR_MASK;

}


uint8 init_ata(void)
{

        init_ata_hardware();

        ResetBus();

        return WaitBusy();

}


void ataoe(SOCKET s,uint8 *mac)
{
        uint16 xdata rlen =0;
        uint16 xdata mac_destport;
        uint16 xdata len =0;
        uint8 xdata mac_destip[4];

        init_ata();

        while(1)
        {
                switch(getSn_SR(s))
                {
                 case SOCK_CLOSED:
                  close(s);                          // close the SOCKET
                  socket(s,Sn_MR_MACRAW,3000,0);  // open the SOCKET with MACRAW mode
                  break;

                 case SOCK_MACRAW:

                  if ( (rlen = getSn_RX_RSR(s) ) > 0)
                  {
                        /* receive data from a destination */
                        len = recvfrom(s,(uint8 *)&aATA,rlen,mac_destip,&mac_destport);
```

```
                        if(aATA.header.msg_type == AOE_TYPE)
                        {
                                 switch(aATA.header.cmd)
                                {
                                        default: break;

                                        case AOE_CMD_ATA:

                                                ATAHandler(s,mac);
                                                break;
                                        case AOE_CMD_CONFIG:

                                                SendConfigAck(s,mac,(AtaConfig *)&aATA);
                                                break;

                                        case AOE_CMD_MAC_MASK:

                                                DumpATAMacMask((AtaMACMask *)&aATA);
                                                break;

                                        case AOE_CMD_RESERVE_RELEASE:

                                        DumpATAReserveRelease((AtaReserveRelease
                                        *)&aATA);
                                                break;
                                }
                        }

                }

                break;

                default:

                        break;
        }
   }
}
```

REFERENCES

[1] ATA over Ethernet http://en.wikipedia.org/wiki/ATA_over_Ethernet

[2] ATAOE Protocol Specification
http://support.coraid.com/documents/AoEr11.pdf

[3] WINAOE Open-Source ATAOE Driver for Windows http://winaoe.org/

[4] Free Starport ATAOE Driver for Windows
http://www.starwindsoftware.com/starport-download-free