FEATURE ARTICLE

by Alexander Pozhitkov (USA)

The NakedCPU (Part 2)

Experimentation and Communication

The first part of this two-part series provided an overview of the NakedCPU platform, which is designed to provide full access to hardware and a CPU without any operating system restrictions. This article explores the BIOS power-on code, noise production, and reading and writing from the parallel port. Experiments with a LAN adapter and network packets are also covered.

Part 1 of this two-part series covered the hardware of a computer based on an Intel CPU. I detailed the NakedCPU, which is an operating system-free experimental platform. The platform consists of two computers connected via a serial port. One computer, the NakedCPU itself, is booted up with a tiny 32-bit protected-mode start-up code. After the start-up, this computer waits for a stream of executable and data bytes coming from the master computer, which is running Microsoft Windows.

Now it's time to consider actual experiments that open up the possibilities for further research! Using the NakedCPU Explorer I presented in Part 1, I will explore the BIOS power-on code, describe how to make the speaker produce some noise, and explain how to read and write the parallel port. In the end, I will detail how I experimented with the LAN adapter and received and dissected network packets.

Listing 1—This is the protocol created as a result of reading ICH documentation and conducting a few experiments.

```
>write              //Speaker enabled. NakedCPU begins producing a continuous sound.
Port address: 0x61
Value: 0x3
>write              //Change the default frequency by writing a timer configuration word. Sound stops,
Port address: 0x43  //because timer is expecting to receive a 2-B divisor. Write a new value 0x0110 in two
Value: 0xbe         //transactions, after which the NakedCPU will begin producing a high-pitch noise.
>write
Port address: 0x42
Value: 0x10
>write
Port address: 0x42
Value: 0x1

>write              //Write a larger divisor 0x2000 to lower the frequency. Writing the configuration
Port address: 0x42  //word is not necessary.
Value: 0x0
>write
Port address: 0x42
Value: 0x20
```

Listing 2—This protocol demonstrates writing to and reading from the parallel port.

```
>write                  //Parallel port turns into Extended Capabilities Port mode via modification of
                        //the Extended Control Register (0x77A). Line D2, goes HIGH and LED glows.
Port address: 0x77A
Value: 0x34
>write
Port address: 0x378
Value: 0x4

>write                  //Reading from LPT is also possible. By writing 0x2c into the Port Control
                        //register (0x37A), we make parallel port read. The LED slightly glows
                        //indicating that pull up resistors are enabled in the reading mode.
Port address: 0x77A
Value: 0x34
>write
Port address: 0x37A
Value: 0x2c
```

EXPERIMENTS: MAKING NOISE

Although it may sound trivial, making a PC speaker produce sound involves an understanding of timers and some low-level work. Ironically, there isn't a way to make a speaker beep using the Windows API on Vista or XP 64-bit version because Microsoft decided that the speaker hardware is obsolete.[1] Certainly, in the past, DOS programmers must have known how to do it, but now it seems to be forgotten. Reading input/output controller hub (ICH) documentation and conducting a few experiments resulted in the protocol for the NakedCPU Explorer shown in Listing 1.[2]

LIGHTING AN LED

The parallel port is becoming more and more obsolete. Nevertheless, it offers a possibility to read and send more than eight lines of data. Strangely, ICH documentation does not say anything about programming a parallel port. Browsing the Internet reveals that there is still some interest regarding the parallel port and programming information is available. Connect an LED to the D2 port line via a 470-Ω resistor and follow the protocol in Listing 2, which demonstrates writing to and reading from the parallel port.

THE "FIRST CRY"

Which instruction does the processor execute first after powering on? Intel documentation says that the processor reads its first instruction from the address 0xFFFFFFF0 (i.e., 16 bytes below 4 GB).[2] Attempting to examine this address with a debugger is fruitless. (I tested it, and it did not work.) In order to reach this high address, which is in the range of high BIOS, a small executable for the NakedCPU was prepared. The executable defined a segment of memory addressing high BIOS and sent the content of the 16 bytes below 4 GB back to the master computer. As expected, there was a short jump to approximately 30 KB below. The executable was modified to download the entire chunk of memory 30 KB below 4 GB up to the top. Curious inquirers are welcome to investigate the saved content and the code of the executable, which are available on *Circuit Cellar*'s FTP site. As a general impression, one can see many accesses to PCI bus and calls for CPUID instruction. It certainly makes sense. Various devices have to be set up and BIOS is attempting to determine which processor is being used.

NETWORK

Communication via the network is accomplished using the media access controller (MAC). Documentation is available from Intel. It is also helpful to read the first three chapters of the IEEE 802.3-2008 standard to get an idea of the low-level network lingo as well as the packet format sent over the wires.[3]

The NakedCPU Explorer enabled me to investigate the internals of the MAC and conduct some experiments. The MAC requires data structures in memory and configuration transactions via the I/O address space. The first step is to determine the I/O address of the MAC, which is called Base Address 2 (BAR2). The address is stored in the PCI configuration space at bus 0, device 25, function 0 (B0:D25:F0) register 0x18. By the way, there is confusion in the documentation referring to the same register. ICH calls this particular register MBARC, while

Listing 3—Instructions for conducting PCI transactions

```
>pci                                                    //PCI transaction consists of two steps: define the
                                                        //location and read the content.
Enter Bus Device Function 0xRegister: 0 25 0 0x18
>read32
Port address: 0xcfc
ecc1
```

Field	BitEnd	BitStart	Initial Value	Description	4100240
FD	0	0	1b	Full-Duplex controls the MAC duplex setting when explicitly set by software. Loaded from the NVM word 13h 0b = Half duplex. 1b = Full duplex.	0
Reserved	1	1	0b	Reserved, write as 0b for future compatibility.	0
Master Disable	2	2	0b	When set, the MAC blocks new master requests on the PCI device. If no master requests are pending by this function, the Master Enable Status bit is set.	0
Reserved	5	3	000b	Reserved, write as 0b for future compatibility.	000
Reserved	6	6	1b	Reserved.	1
Reserved	7	7	0b	Reserved, always set this bit to 0b.	0
SPEED	9	8	10b	Speed selection. These bits determine the speed configuration and are written by software after reading the PHY configuration through the MDIO interface. These signals are ignored when autospeed detection is enabled. 00b = 10 Mbps 01b = 100 Mb/s 10b = 1,000 Mbps 11b = not used	10
Reserved	10	10	0b	Reserved, write as 0b for future compatibility.	0

Table 1—A breakdown of the value 0x4100240 (top right) into bits is shown as a fragment of the worksheet. Bits appear after textual descriptions of the individual fields.

the MAC documentation calls it BAR2. Conduct PCI transactions with the NakedCPU Explorer (see Listing 3).[2, 4]

The number 0xECC1 means the I/O address is actually 0xECC0 with the 0th bit hardcoded to 1 to indicate that the address is indeed in the I/O space as opposed to being memory-mapped.[4] The latter indication is important because all configuration and communication with the MAC can also be done using memory-mapped registers, which is faster. However, for our experiments, it is sufficient to use the I/O space, because it is simpler and accomplishes the same results as memory-mapped operations.

In order to interact with the MAC, the inquirer writes an address of a register within the MAC into the BAR2 I/O address (0xECC0). After that, BAR2+0x4 (0xECC4) becomes a window to the value of that MAC register. It is important to mention that BAR2 and BAR2+0x4 accept only 32-bit double-word read/write operations. The MAC registers have plenty of bits to deal with and some bits are dependent on one another. It is difficult to understand the settings just by looking at the hexadecimal value of a register. An Excel worksheet, InterpretRegister.xls (also available on *Circuit Cellar*'s FTP site), features a macro that helps in this situation. Specifically, a table containing bit descriptions should be copied and pasted from the hardware documentation, and a hexadecimal value of a register will be converted into binary 1s and 0s in appropriate cells right next to the description text (see Table 1).

Let us examine the CTRL(0x0) and STATUS(0x8) registers. The address of each register is provided in parentheses. After power-on with the network cable unplugged, you will see what is shown in Listing 4.

After connecting the master computer with the NakedCPU via a network cable, the CTRL register stays the same as expected, while the STATUS register changes to 0x80080683. The new value means full duplex communication, established link, and 1-Gbps speed. The master computer running Windows XP reported the same communication parameters, which indicates that the NakedCPU network interface was able to negotiate with the master computer's interface on the hardware level.

Listing 4—The code shown after power-on with the network cable unplugged

```
>write32                    //This particular bit constellation
                            //determines among other things
                            //enabled automatic configuration for
                            //speed and full/half duplex.
Port address: 0xecc0
Value: 0
>read32
Port address: 0xecc4
100240

>write32                    //These bits tell that there is no
                            //link established but initialization
                            //is completed.
Port address: 0xecc0
Value: 0x8
>read32
Port address: 0xecc4
80080600
```

RECEIVING & INTERPRETING NETWORK PACKETS

In this section, I'll cover how to experiment with reading network packets originated from the master computer. I found that when the master computer detected the live NakedCPU via the network cable, Windows began generating DHCP requests. These requests are attempts to obtain an IP address and other high-level network settings because Windows assumes (erroneously) that the NakedCPU is a router or a network server. Although Windows is mistaken, this is perfectly fine for our experiments, because we can catch these packets and examine them.

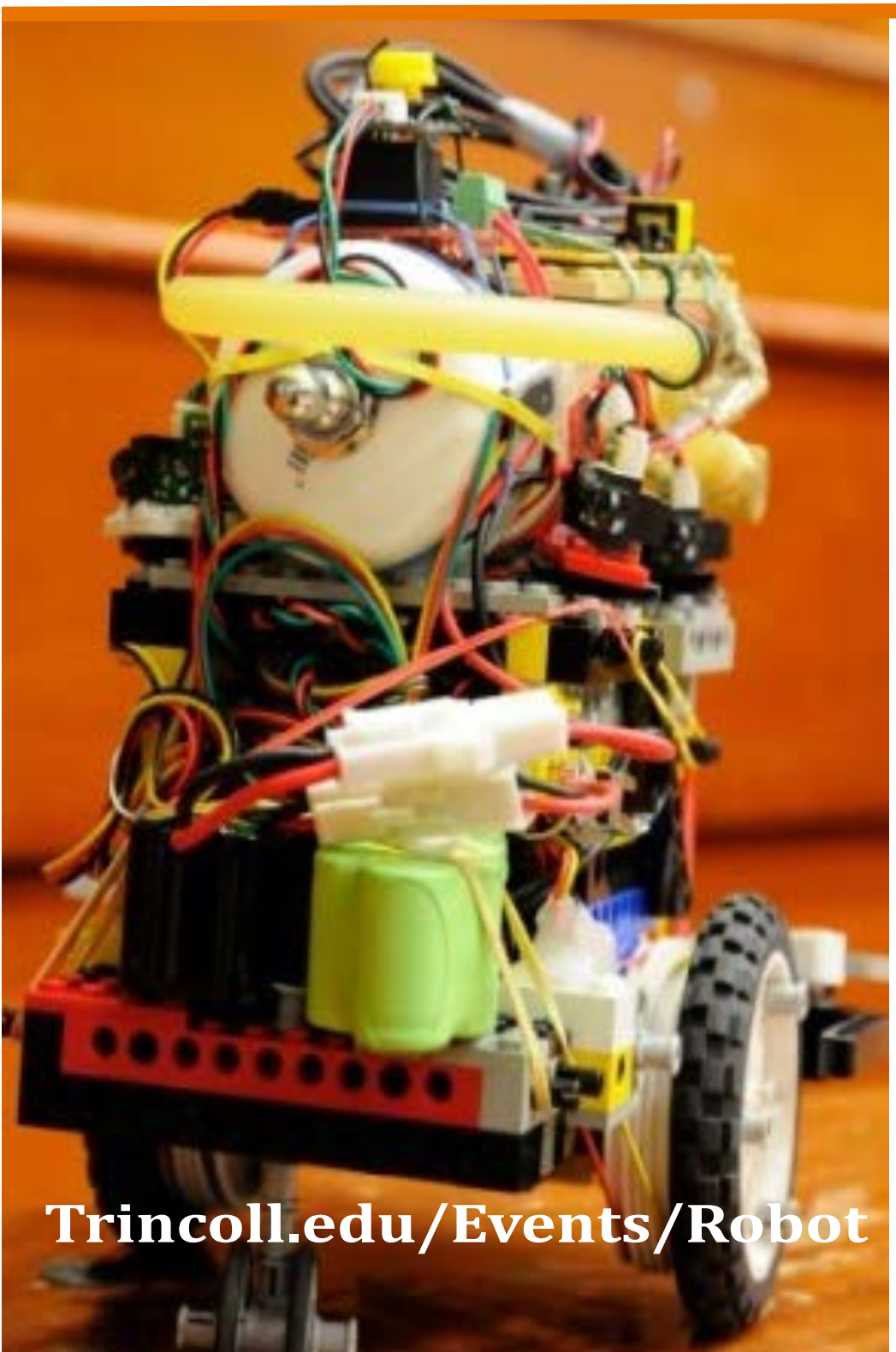
The MAC uses direct memory access to store the received data. We

Trinity College Robotics Competitions

Hartford, Connecticut, USA

ROBOWAITER

Promoting Assistive Robotics for Persons with Disabilities



Trincoll.edu/Events/Robot

Saturday March 31

8:30 a.m. - 7:00 p.m.
Robot Practice

10:00 a.m. - 11:30 a.m.
Robotics Workshops

12:30 p.m. - 1:45 p.m.
Robotics Keynote Speakers

2:00 p.m. - 4:00 p.m.
RoboWaiter Competition

Sunday April 1

11:00 a.m. - 4:00 p.m.
Versa Valves, Inc. presents
Firefighting Home Robot Contest

Proudly Sponsored by:

VERSA Valves

Shown left: 2011 Winner, *MonsterBot* from Massachussetts, USA

Listing 5—After the MAC finishes storing packets, it will update the descriptors to indicate received packet size, errors, and several other parameters such as these.

```
>memwrite                          //The command "memwrite" asks for the address and the number
                                   //of double words to be written. Note, writing below 0x100000
                                   //will cause general protection fault and reboot of the NakedCPU.
0xAddress above 0x100000: 0x100500
Num dwords: 8
0x101000
0
0
0
0x101200
0
0
0
```

have to create several descriptors to tell the MAC where to write the data. Thus, two memory ranges are required: one for the descriptors and the other for packets. Refer back to Table 3 in Part 1 of this series. You can see there is an area of memory above the address 0x100000 available for the inquirer's data. Bearing in mind that the NakedCPU Explorer uses a tiny bit of that memory to store incoming commands, we can safely use addresses above 0x100500. It is sufficient to create two descriptors for the initial experiments. A descriptor is a data structure of four double words (16 bytes). The first two double words are a 64-bit physical address of the location where the packet is to be stored. With the NakedCPU Explorer's ability to write into memory locations, I can create the descriptors at the address 0x100500, pointing to two 512-byte long buffers located at the addresses 0x101000 and 0x101200. A collection of descriptors is called a "queue." After the MAC finishes storing packets, it updates the descriptors to indicate the received packet size, errors, and several other parameters (see Listing 5).

The MAC has to know the location of the descriptors, the size of the receive buffers, and the type of packets to receive. This information has to be stored in several MAC registers: RDBAL0(0x2800) and RDBAH0(0x2804)—low and high portions, respectively, of a 64-bit physical address of the base of the queue; RDLEN0(0x2808)—length of the memory buffer allocated for the queue; RDH0(0x2810) and RDT0(0x2818)—head and tail pointers, respectively; RFCTL(0x5008)—receive filter control register; and RXCSUM(0x5000)—receive checksum control register. Before setting up these registers, bit 26 of the CTRL(0) register has to be set to 1, which causes the MAC to reset. After setting up all registers, data reception is initiated by writing into RCTL(0x100) to set up the "enable" bit, the size or the receive buffers reception mode, and the type of descriptors.

For the CTRL, RCTL, RFCTL, and RXCSUM registers, the InterpretRegister.xls worksheet shows values (and bit states) that will be used in our experiments. The meaning of value for the RDLEN is somewhat confusing. According to the documentation, the length of the queue buffer must be a multiple of 128, which means at least eight descriptors (128/16) must be in the queue. However, we have only two descriptors. I determined experimentally that it is not a problem to tell the MAC that the queue buffer is larger than it needs to be, as long as the RDT0 register is pointing to the end of the actual queue. Hence, for my particular experiment, I set RDLEN = 0x80, RDT0 = 0x2.

In order to set the MAC registers, I had to copy and paste the columns from Table 2 into the NakedCPU Explorer in the same order, column I through V. Note that at the end of the fourth step, the MAC is ready to enable reception. That step ends up in reading from the

I	II	III	IV	V
write32	write32	write32	write32	write32
0xecc0	0xecc0	0xecc0	0xecc0	0xecc0
0	0x2804	0x2810	0x5008	0x100
write32	write32	write32	write32	write32
0xecc4	0xecc4	0xecc4	0xecc4	0xecc4
0x4100240	0	0	0x8000	0x402800A
write32	write32	write32	write32	
0xecc0	0xecc0	0xecc0	0xecc0	
0x2800	0x2808	0x2818	0x8	
write32	write32	write32	read32	
0xecc4	0xecc4	0xecc4	0xecc4	
0x100500	0x80	0x2		

Table 2—Commands and values to be pasted into the NakedCPU Explorer

Descriptor 1	Descriptor 2
0	0
f8270fe8	f8270fe9
20073	20073
156	156

Figure 1—The descriptors updated by the media access controller (MAC)

status register, which should result in a value of 0x80683. This value is similar to the one previously described (0x80080683) with the difference that bit 31 is cleared, which indicates that the DMA clock cannot be lowered to one quarter of its value. The reason why the MAC changed its "mind" concerning the DMA clock is not clear, but this is not relevant for our experiments. I initiated the sending of DHCP packets by typing ipconfig/renew in the Master's command-line tool.

After reading from the network, the MAC updates the two descriptors (see Figure 1). With the `memread` command, observe the new values by reading eight double words beginning from the address 0x100500. You will see that the descriptor's address field is changed and two additional values appeared. An inset shows the new values my computer produced. Detailed information about the fields is provided in the MAC documentation.[4] Briefly, the lengths of the packets are 342 bytes (0x156). No errors occurred, the descriptors are indicated as "done," and the entire packet fits with the buffer (0x20073).

The MAC stored the two actual packets at the addresses 0x101000 and 0x101200, respectively. Their contents are present in the InterpretRegister.xls "Packets" worksheet. Figure 2 shows the beginning of the stored data array and the order of transmission. The first 6 bytes marked in blue are the destination (broadcast) address, which are followed by the source addresses and the 2-byte length/type field. The latter field is transmitted with most-significant byte first, which makes it 0x0800. If the value of the length/type field is less than or equal to 0x05DC, then it indicates the length of the packet, otherwise its type (Ethertype). Information found at the Institute of Electrical and Electronics Engineers

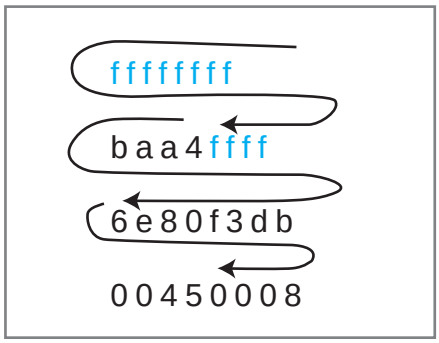


Figure 2—The beginning of the stored data array and the order of transmission

"The NakedCPU platform enables developers to create task-specific applications using only necessary components. For instance, for a large-scale database, there is no need to support a GUI, USB plug-and-play, audio cards, .NET, and many other things. An additional bonus is that the NakedCPU platform is immune to viruses. As in biology, where flexible viruses attack well-evolved organisms, computer viruses attack well-developed operating systems. With the NakedCPU, on the other hand, a particular task-specific solution can be very unique. Therefore, virus creators simply won't have enough information to explore potential security holes."

Bit:	0 – 3	4 – 7	8 – 15	16 – 31
… 00450008 E80F4801 …	Version [0x4]	Internet header length (IHL) [0x5]	Type of service [00]	Total length [0x0148]

Table 3—The first 32-bit word of the IP header. The top line shows the bit numbers. The left column is a fragment of the received packet. The underlined values of the packet are broken down into the fields.

(IEEE) website in theory provides specific Ethertype values; however, it is virtually impossible to find the actual list of values. Luckily, Wikipedia points to an exact URL in the innards of the IEEE site. According to the standard, Internet protocol (IP) is designated with the Ethertype 0x0800. Apparently, the next step is to investigate the format of the IP, described in RFC894, which points to RFC791.[5]

The fields of the IP header are transmitted in a similar way as the previously described length/type field (i.e., the most-significant bit and the most-significant byte first). In contrast to the intuitive notation, where bit 0 is the least-significant bit, RFC791 provides the opposite, with bit 0 as the most-significant bit. Table 3 shows the continuation of the received packet and the assignment of its specific values to the fields of the first 32 bits of the IP header. The important fields are IHL and Total Length. IHL indicates the number of 32-bit words in the header. In my case, it is five, which means according to RFC791, the Options and Padding fields are omitted. It is interesting to note that the MAC reported receiving 342 bytes and the IP header indicated 328 (0x0148) bytes. The difference of 14 bytes makes perfect sense. They make up to 2 bytes of the length/type field plus 2 × 6 bytes of destination and source hardware addresses.

All other fields of the IP header are easy to map following this example. Specifically, the field values are: Identification—0x0fe8 (packet 1), 0x0fe9 (packet 2); flags and fragment offset—0; time to live—0x80; protocol—0x11; header checksum—0x29be (packet 1), 0x29bd (packet 2); source IP address—0.0.0.0; and destination IP address—255.255.255.255. It is understandable that the master PC is asking for an IP address while doing the dynamic host configuration process. Therefore, the source IP address is all zeroes. For the destination, the address is all 255.255.255.255, which is a broadcast address, similar to the hardware broadcast address (6 bytes, all 255).

The value for the protocol field is 0x11 (17). According to RFC790, it refers to the user datagram protocol (UDP)—the next level of data encapsulation—which is described in RFC768. According to that document, a UDP header contains four 16-bit words. In the order of transmission, these words are source port, destination port, length, and checksum. The values received by the NakedCPU in my experiment were: source port—0x0044, destination port—0x0043, length—0x0134, and checksum—0x5c1a (packet 1) and 0x581a (packet 2). The length of 308 (0x134) bytes makes sense, since the IP header reported the total datagram length of 328 bytes minus 20 bytes occupied by the IP header. The source and destination port numbers should have been explained in RFC790; however, it turned out that a long chain of other documents makes this one obsolete. At the end of the chain, it is suggested to look at the website of the Internet Assigned Numbers Authority (IANA), where, unfortunately, the information is not well organized. It was possible, however, to find among the obsolete RFCs that port numbers 67 (0x43) and 68 (0x44) correspond to the Bootstrap protocol, server and client ports, respectively. The Bootstrap protocol described in RFC1542 points to the DHCP protocol (see RFC2131). Figure 3 depicts all the fields covered in this section.

FUTURE OUTLOOK

It is possible to directly experiment with the Intel CPU and other PC hardware without any layers of unknown intermediate code intended to make our lives "easier." As of yet, the most comprehensive documentation exists for the processor itself.[6]

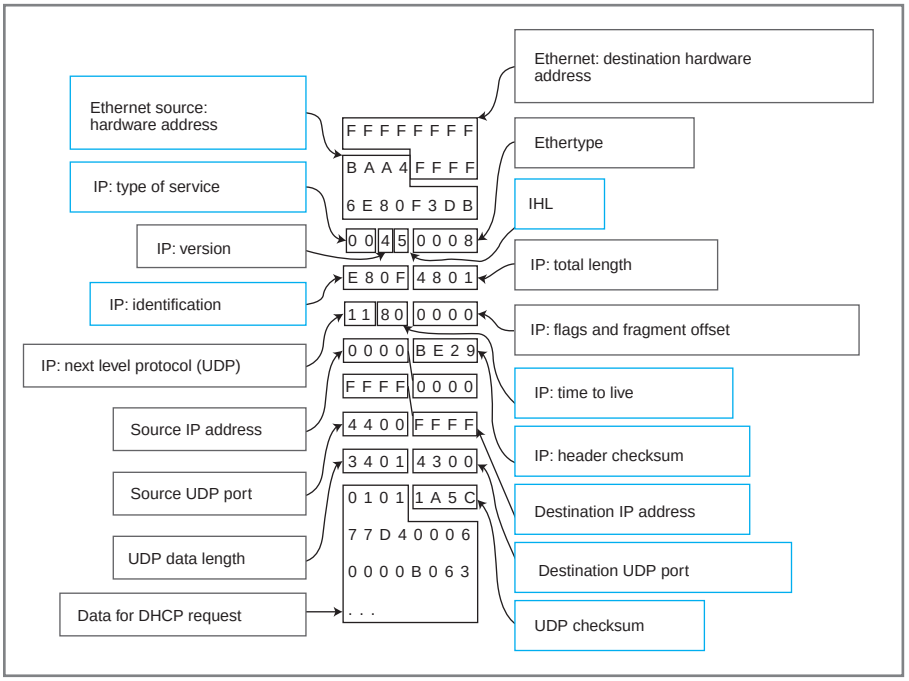


Figure 3—This is a summary of the data fields for Ethernet, IP, and UDP protocols in the received packets.

The other hardware is not well documented, which is why it takes a significant amount of effort to gather pieces of information from the Internet and conduct experiments. The old books that dealt with hardware are DOS-oriented and seriously outdated. The new hardware is hidden behind layers of unknown code.

In theory, the NakedCPU platform enables developers to create task-specific applications using only necessary components. For instance, for a large-scale database, there is no need to support a GUI, USB plug-and-play, audio cards, .NET, and many other things. An additional bonus is that the NakedCPU platform is immune to viruses. As in biology, where flexible viruses attack well-evolved organisms, computer viruses attack well-developed operating systems. With the NakedCPU, on the other hand, a particular task-specific solution can be very unique. Therefore, virus creators simply won't have enough information to explore potential security holes.

Dr. Alexander Pozhitkov (pozhit@uw.edu) has an MS in Chemistry and a PhD in Genetics from Albertus Magnus University in Cologne, Germany. For 12 years he has been involved with interdisciplinary research relating to molecular biology, physical chemistry, software, and electrical engineering. Currently, Dr. Pozhitkov is a researcher at the University of Washington, Seattle. His technical interests include hardware programming, vacuum tubes, and high-voltage electronics.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2012/260.

REFERENCES

[1] Microsoft Corp., MSDN, Beep function. http://msdn.microsoft.com.

[2] Intel Corp., "I/O Controller Hub 10 (ICH10) Family Datasheet," 2008, www.intel.com/content/dam/doc/datasheet/io-controller-hub-10-family-datasheet.pdf.

[3] Institute of Electrical and Electronics Engineers (IEEE) Standards Association, "IEEE Get Program," http://standards.ieee.org/about/get/.

[4] Intel Corp., "I/O Controller Hub 8/9/10 and 82566/82567/82562V Software Developer's Manual," 2009, www.intel.com/content/dam/doc/manual/i-o-controller-hub-8-9-10-82566-82567-82562v-software-dev-manual.pdf.

[5] The Internet Engineering Task Force (IETF), "Index of /rfc," www.ietf.org/rfc.

[6] Intel Corp., "Intel Core2 Duo Processor E8000 and E7000 Series Specification Update," 2010, http://download.intel.com/design/processor/specupdt/318733.pdf.

RESOURCE

Internet Assigned Numbers Authority (IANA), www. iana.org.

SOURCE

Windows XP
Microsoft Corp. | www.microsoft.com