Internet Connected FPL

Hamish Fallside[1] and Michael J. S. Smith[2]

[1]Xilinx Inc, 2100 Logic Drive, San Jose, CA 95124, USA
hamish@xilinx.com

[2]University of Hawaii at Manoa, 2540 Dole Street, Honolulu, HI 96822, USA
msmith@eng.hawaii.edu

Abstract. In this paper we explore the design of internet-based systems using field-programmable logic (FPL). We describe results from designing a hardware platform that connects FPL directly to an internet. This hardware platform comprises an FPGA; an Ethernet interface; storage for static and dynamic configuration; and nonvolatile configuration logic. An important feature of our hardware platform design is the implementation of network protocols that allow transfer of both application and configuration data to and from an FPGA across an internet. We provide quantitative comparisons between the implementation of network protocols in programmable logic and implementations using general-purpose processors.

Introduction

There is a growing demand for small, low cost, low power, flexible computing devices or appliances that attach directly to internets. For example, there are marketing forecasts that non-PC devices will account for almost 50 percent of internet connected devices by 2002 and that sales of such internet appliances will likely exceed the number of PCs sold by 2005 [1]. These internet connected devices or appliances are a potential application for field-programmable logic (FPL). The ability to use FPL and the Internet to reprogram hardware remotely would enable re-programmable switches, routers, and firewalls; multipurpose hardware devices; remote debugging and instruction; field upgrades and fixes; support for new and changing standards; and many other applications [2–6]. We set out to explore the role of FPL in such internet applications.

Currently PCs dedicate special hardware to control peripherals (memory, disks, video, even for mouse and keyboard) yet there is no special hardware to control what may become the most important peripheral of all: the Internet. Instead, we typically use a large and inefficient operating system running on a processor that is highly optimized for arithmetic and data processing. The complex pipelines and large arithmetic and floating-point units of modern microprocessors remain largely unused during communication. We wondered what was the most appropriate design for internet applications: processors (general or special purpose), dedicated hardware, programmable hardware, or a combination of any or all of these with software.

R.W. Hartenstein and H. Grünbacher (Eds.): FPL 2000, LNCS 1896, pp. 48–57, 2000.
© Springer-Verlag Berlin Heidelberg 2000

There are three types of questions we set out to answer:

- Engineering. Can we replace conventional microprocessor and software by dedicated internet hardware? What performance can we achieve?
- Economic. What is the cost? Does it make economic sense to dedicate hardware to the Internet as a peripheral?
- Technology. If the economics make sense, what is the appropriate technology?

We built a hardware platform to explore the answers to these questions; particularly the trade-offs between hardware and software implementations of an increasingly important function: internet connectivity. In this paper we answer the first group of engineering questions by explaining how we designed one solution. We do not have complete answers to the remaining questions (yet), but in this paper we begin to address these issues in a quantitative manner.

There are two ways to communicate over a channel: in a dedicated or shared manner. In a shared network such as the Internet, data is divided up into packets and each packet travels separately from source to destination. We focus on the Internet in this paper, though our work applies to any packet-shared network: wired or wireless.

The Internet is built around a set of protocols, which are methods and algorithms that handle the data packets. The most commonly used protocols established over the last 20 years are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). TCP and IP are usually thought of as separate layers, but normally implemented together in software, called the TCP/IP stack, that runs on a general-purpose processor. However, the Internet runs on many layers of software and hardware, not just the TCP/IP stack. Comer [7] and Stevens [8] explain the software fabric of the Internet.

If the Internet is a road and a connected device is a car, you could think of the TCP/IP stack as the car engine. The hardware layer below the TCP/IP stack is the media access controller (MAC), the gearbox and brakes of the car. The hardware layer below the MAC that connects to the cables of the Internet is the physical layer (PHY), the wheels of the car. There will be different PHY layers depending on whether we use cable, wireless, or fibre, for example. However, the connection between the MAC and different PHY layers is very similar (and often identical), and is called the media-independent interface (MII). The digital MII is the interface between special (often mostly analog) PHY circuits and the digital systems that implement the MAC and TCP/IP stack. We have been unable to find good reference materials that explain the (mostly hardware) MAC and PHY layers, other than the datasheets from companies that make these components (see [9], for example). The definitive reference is the IEEE standard 802.3.

If we wish to add internet connectivity to a device, we must decide how to implement the MAC layer and protocol processing, such as the TCP/IP stack, either in programmable logic, dedicated hardware, processor plus software, or some combination of these. In order to make these decisions we must be able to measure the advantages and disadvantages of each approach. Kumar et al. have examined benchmark suites for configurable computing systems [10], but we need measurement techniques for network processing. There is little published work in this area and we decided to establish some metrics ourselves before designing out hardware platform.

In the following section we establish some complexity measures for a MAC as well as a software plus processor implementation of a TCP/IP stack. TCP and IP may

not be the most important protocols (and certainly not the only protocols) for future internet applications, but in this paper we use the TCP/IP stack to make our complexity estimates, because few devices use other protocols.

We will examine three example designs using a PIC, an 8051 microcontroller, and a Sparc microprocessor to estimate average as well as lower and upper bounds for the amount of hardware and software required to implement a MAC and TCP/IP stack. At the end of this paper, we will compare the figures for these example designs with our FPGA hardware implementation.

Bounds on Protocol Processing

In June 1999 a small web server was implemented in 512 12-bit instructions (256 instructions for the TCP/IP functions) using a programmable microcontroller (a Microchip 12C509A, often called a PIC [11]) and a 24LC256 EEPROM (256 kbit). This web server used the Serial Line Internet Protocol (SLIP) in order to communicate with a host computer (bypassing the need for an Ethernet MAC). This PIC-based design supported a web site with approximately 32 small files (located in the external EEPROM) [12].

Though few details of this particular design are available, it is reasonable to conclude that it used „stateless TCP/IP“ instead of a conventional TCP/IP stack. Stateless TCP/IP uses precomputed packets to drastically simplify the stack design. Nevertheless, we will use this PIC-based design to establish a lower bound on the complexity of an implementation of a very simple form of TCP/IP stack using software plus processor.

We can make an estimate of gate size (in four-transistor NAND gate equivalents) if we were to implement the PIC-based design using an ASIC or FPGA. We have used metrics from [13, Chapter 15] to estimate the number of gates used by the components, the ALU and datapath, for which we have area, but not gate size estimates. We used a conversion factor of $5 \times 10^{-4} / \lambda^2$ for a standard-cell ASIC implementation. The parameter λ measures the feature size and permits gate size measurements to be made from area measures, independently of the process generation.

The 12C509A PIC contains a processor, 1024 by 12 bits of EPROM memory, and 41 bytes of data memory [11]. Table 1 shows the estimates of the gate counts for each of the components in the 12C509A PIC. The total gate count estimate is 16.4 kgate (including onboard program and data memory) or 4.5 kgate (excluding program memory). This estimate is definitely a lower bound because some of the TCP/IP tasks (constructing packet headers, for example) have been precomputed and stored in the external EEPROM.

Table 2 shows the PIC-based design gate estimates translated to the Xilinx Virtex FPGA family. The Virtex parts contain static RAM (called Block RAM) that are separate from the CLBs. (The CLB, or Configurable Logic Block, is the basic unit of logic in a Xilinx static RAM-based FPGA. One Virtex CLB contains four logic cells plus four flip-flops organized as two slices.) Each Block RAM is a synchronous dual-port 4 kbit RAM. The 12 kbit of EPROM onboard the 12C509A PIC thus translates to four blocks of Virtex RAM. We will explain the figures used to estimate the size of the MAC presently.

Table 1. Microchip 12C509A PIC Gate Size Estimates.

PIC component	Area/kλ^2	kgates	Notes
ROM program memory [11]		12	1024 x 12 bits, 1 gate per bit
EEPROM data and control memory [11]		0.5	41 bytes, 1 gate per bit
8-bit ALU [13]	330	0.2	Estimate for a simple 8-bit processor, 2901 equivalent
Instruction decode		2	Estimate for a simple 8-bit processor
8-bit datapath	2400	1.2	Registers and Muxes
Timers (estimate)		0.5	Registers and Muxes
Total		16.4	

Our average bound design uses a typical implementation of a MAC and a TCP/IP stack in an embedded system with an 8051 microcontroller. A commercial 8051 core (an 8-bit machine) occupies approximately 500 CLBs in a Xilinx Virtex FPGA (this figure is for the processor core and excludes program memory) [14]. The synthesizable 8051 soft core from Synopsys requires 10–13 kgate [15]. These two measurements of the size of an 8051 give us a conversion factor (biased towards circuits such as microprocessors) of approximately 20 gates/CLB that we will use in this paper. (Any attempt to compare ASIC and FPGA capacities in gates is fraught with problems, but we are using these figures here for no more than back-of-the-envelope calculations.) Additional space for code is required to implement the TCP/IP stack and an Ethernet MAC in our 8051 based example. We can estimate the code requirements as follows:

- A typical embedded TCP/IP stack requires approximately 25 kbyte of code (see [16], for example).
- A full-featured commercial Ethernet MAC core occupies about 800 Virtex CLBs [17], or about 16 kgate.
- A basic Ethernet MAC requires about 4 kgate [13], or about 200 Virtex CLBs.
- Driver code for the Ethernet and a real-time operating system (RTOS) to handle the 8051 interrupts is also required. A typical RTOS requires a minimum of 4 kbyte of code to perform task management and handle interrupts (see [18], for example, for figures for an ARM7 processor).

Table 2 shows that the 8051 based design totals 10 kgate (excluding program and data memory). Table 2 also includes a lower estimate of 25 kbyte of ROM and 4 kbyte of RAM for program and data memory, which translates to 58 Virtex RAM blocks.

Our upper bound design uses a Sparc processor. The Leon embedded Sparc compatible processor requires 1800 Virtex CLBs (36 kgate) [19]. A TCP/IP stack for a Sparc compatible processor requires approximately 25 kbyte [16]. Again, this is in addition to the approximately 10 kbyte memory requirement of a Sparc compatible RTOS (see [18] for figures for a Sparc-lite processor). Table 2 shows the Sparc-based design totals 36 kgate (excluding program and data memory) and would require about 70 blocks of RAM if implemented in a Virtex FPGA.

Table 2. Processor implementation summaries.

Resource	PIC	8051	Sparc	Notes
Gates (kgate)	4.5	10	36	
ROM (kbyte)	1.5	25	25	1024 x 12 bits
RAM (kbyte)	0	4	10	
Virtex BRAM	3	58	70	4 kbit per block RAM
Virtex CLBs	425	700	2000	Include 200 CLB for simple Ethernet MAC

Our estimates for implementing the lower bound (PIC), average (8051), and upper bound (Sparc) solutions for internet protocol processing in FPGA logic are summarized in Table 2. The smallest member of the Virtex family, the XCV50, contains 384 CLBs and eight blocks of RAM; the XCV1000 contains 6144 CLBs and 32 blocks of RAM. The lower bound PIC-based design fits in a single Virtex device, as the memory requirements are three RAM blocks. The other two designs would require external RAM.

Details of Our Hardware Platform Design

Figure 1 is a block diagram of our hardware platform, the Demonstation. The Ethernet interface to the Demonstation is a PHY chip [9]. The analog portion of the PHY chip is the only thing preventing us from connecting a Xilinx FPGA physically to the Ethernet cable.

We use the Xilinx byte-wide SelectMap interface to configure the FPGA [20]. Data from a 1 Mbyte parallel ROM is written to the SelectMap interface of the FPGA at power-up by a configuration controller, implemented in a nonvolatile CPLD. The FPGA then takes control of the configuration controller.

Two banks of asynchronous byte-wide SRAM, each 1 MB, are connected to the FPGA. One of these SRAM banks connects to both the FPGA and the configuration controller. This allows the FPGA to load configuration data from the Internet and then write that data to SRAM. When a configuration file has been assembled in SRAM, the FPGA instructs the configuration controller to reconfigure the FPGA using the data in SRAM. Using this scheme, FPGA reconfiguration may be either full or partial.

All the hardware implementations of embedded web servers or TCP/IP stacks (that we are aware of) either use a serial protocol or require a computer or router to connect to a shared network. These are important differences, both of which are often ignored, from a direct internet connection. The Demonstation interfaces directly with the packet-switched multiple-access Ethernet physical layer. In addition, our design permits dynamic reconfiguration over the Internet, without the need for an attached computer, and this (as far as we know) is also unique.

Constructing the Internet in layers (software or hardware) allows each layer to be changed. Applications use the Internet protocols to transfer data. These protocols are defined by the Requests for Comments (RFC) [21]. The FPGA must implement an application that is capable of file transfer using the network protocols to move data to and from the Demonstation.

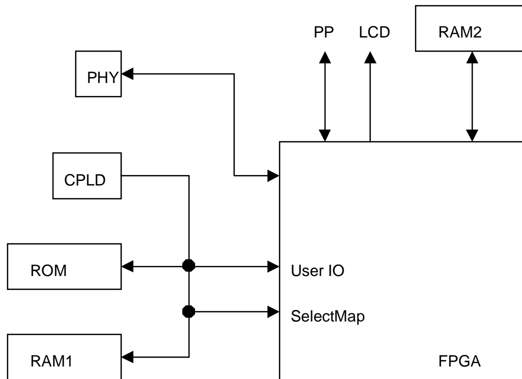

Fig. 1. Block diagram of the Demonstation.

One application that uses TCP is the File Transfer Protocol (FTP). We could use FTP to transmit configuration and data to the FPGA, but to simplify things, we started with the Trivial File Transfer Protocol (TFTP), which is less complex than the FTP. TFTP uses the Universal Datagram Protocol (UDP) rather than TCP. Taken together, UDP and TFTP function in a similar fashion to TCP, and as a subset of FTP.

The MAC currently supports 10BaseT, but the PHY device will also support 100BaseT, and provides full Carrier Sense, Multiple Access, Collision Detect (CSMA/CD), and Ethernet II and IEEE 802.3.

The IP layer provides IP address checking and the IP header checksum. In our TFTP implementation no defragmentation is performed on received IP datagrams (fragmentation occurs when a datagram is too large for a network between the source and destination). Defragmentation is generally regarded as being undesirable, because loss of a single fragment means that the entire datagram will be discarded [8]. TFTP uses a maximum data block size of 512 bytes, and thus datagrams will not be fragmented on an Ethernet. We did implement a simple defragmentation design in approximately 100 Virtex CLBs as an experiment.

The UDP layer performs a checksum on the message contained within the IP datagram, and provides source and destination port numbers for the applications that use it. TFTP on the Demonstation implements a write-request server to a client. TFTP sends a response message back to the client for each message received. This return message is usually an acknowledgement of data received or an error message. Data received over the Internet from the client is written to the SRAM (RAM1 in Fig. 2) on the Demonstation. Once the last data message is acknowledged the TFTP layer signals the configuration controller to initiate reconfiguration of the FPGA.

Results

Our Demonstation hardware platform successfully performs reconfiguration of Virtex devices across 10 Mbit/s Ethernet using TFTP in a few seconds. The actual

reconfiguration time depends heavily upon network loading. TFTP transmits data in 512 byte messages and each message has to be acknowledged before the next can be transmitted. This gives a theoretical maximum throughput of 8 Mbits/s. Our measured TFTP results give reconfiguration times of up to a minute for the largest Virtex devices (which require up to 8 Mbit of configuration data). TCP has a theoretical maximum throughput of 1.2 Mbyte/s [22] with measured sustained rates at 90% of this value. We would thus expect to reduce reconfiguration times slightly using TCP over 10 MB/s Ethernet.

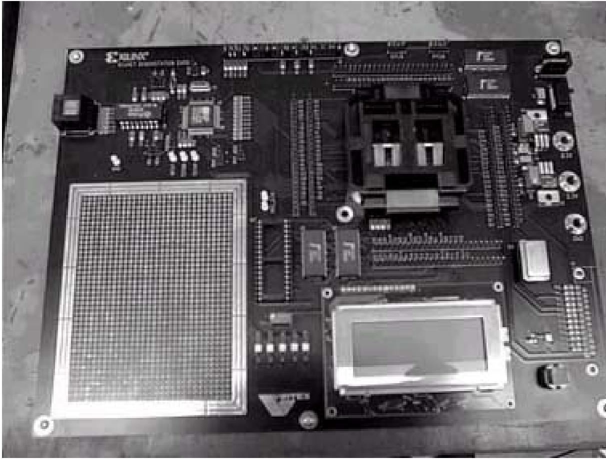

Fig. 2. The Demonstation platform. Top left is the Ethernet interface: cable connector, transformer and PHY chip. Bottom left is a prototyping area. Bottom right is an LCD panel, and parallel port (PP) for debugging. Above the LCD is a socket for a Virtex FPGA. Next to the FPGA socket (above right and below left) are two banks of SRAM (RAM1 and RAM2). To the right of the FPGA socket are the power regulators. The configuration controller, a nonvolatile CPLD, is on the underside of the board.

The worst case timing for our design occurs when a full-length frame (containing 1500 bytes of data) is followed by a minimum length frame (with 46 bytes of data). Frames are separated by an inter-frame gap (IFG) of 9.6 μs. In the worst case we have to move 1500 bytes from the MAC buffer to SRAM before we can process the next frame. So we have 67.2 μs to process and store 1500 bytes, or 44.8 ns per byte. Our design meets this timing constraint using a system clock of 50 MHz, giving us two clock cycles per byte to process and store each byte.

For 100BaseT Ethernet there would be 15.4 μs to process a full-length frame, which requires a clock frequency of 146 MHz. The increased data rate could be implemented by adding another receive buffer and/or by increasing the data path sizes within the stack.

Table 3. FPGA implementation summary. The design results in this table were obtained using Synplify synthesis software with the Xilinx Alliance tools version 2.0. The CLB and RAM block counts are for the Virtex FPGA family.

Protocol/layer	VHDL lines	CLBs	RAM blocks
MAC	3000	200	6
IP	2000	150	0
UDP	4000	175	3
TFTP	2400	133	0
Total	11400	658	9

The complete FPGA protocol stack contains a total of nine block RAMs and 658 CLBs (or about 13,000 gates by the simple conversion metrics we have used here). Table 3 summarizes the details of the various protocol and application layers implemented in the FPGA.

We can compare the breakdown of VHDL and hardware for the Demonstation protocol stack shown in Table 3 to an analysis of network protocol software for Unix [22]. About 40% of the Unix code is dedicated to TCP; 20% to IP; 20% to network utilities (error checking and so on); and 10% to UDP and ARP (a protocol for address resolution) combined. These estimates hold for both the number of lines of C code and number of procedures in each protocol layer. One thing these figures tell us is that we should expect to write another 5000 lines of VHDL code and use at least 200 Virtex CLBs (or about 4 kgate) to implement a TCP layer.

We have tried to compare the results shown in Table 3 with other implementations, but we believe we are the first to implement a protocol stack in an FPGA and the first to design a platform that implements reconfiguration over the Internet. Probably the closest work to ours is a product announced by iReady aimed at internet FAX machines [23]. The custom iReady chip is built by Seiko and uses 67 kgate for a network protocol stack (for the point to point protocol, PPP, using a serial port together with support for IP, TCP, UDP, and two sockets) and 20 kbyte for network buffers. The iReady chip is thus not able to connect directly to the Internet, but does implement a TCP/IP stack. From brief discussions with iReady, we believe one reason for the large difference between their implementation (67 kgate) and ours (13 kgate) is due to the fact that their chip has been designed and tested to work with many different routers and switches, incurring additional overhead.

In the introduction we explained that we set out to answer three sets of questions. In this paper we have presented answers to the first set of engineering questions. We can replace conventional microprocessor and software by FPL in internet applications. We can implement internet reconfiguration. We can reconfigure an FPGA containing several hundred thousand gates in a reasonable time across the Internet.

In the introduction we also posed economic and technology questions for which we do not yet have answers. We have included very brief remarks on the issues of costs and power because we find these are questions that we are asked repeatedly when discussing our project.

We made no cost estimates before building Demonstation and cost did not influence our design. The current platform is dominated by the cost of the Virtex part, but low-cost alternatives, such as the Xilinx Spartan FPGAs, are equally suitable.

Perhaps more telling is that there is no other way to implement the ability to reconfigure hardware using an internet. In situations where remote reconfiguration is valuable (such as satellites, for example) cost may be less of a factor.

The 8051 design gives us the ability to estimate power dissipation of an average embedded TCP/IP stack implementation. The Dallas Semiconductor 8051 consumes about 5 mW/MHz [24]. In the Xilinx Virtex family, the 8051 core runs at about 30 MHz. At 30 MHz the Dallas 8051 consumes 150 mW. We can quickly estimate that an 8051 core running in a Virtex part at 30 MHz would consume more than this. Our current Virtex implementation draws nearly 1 A total while using three different supply voltages (5V, 3.3V, and 2.5V). The Demonstation includes FPGA, CPLD, static RAM, LCD, parallel port, and PHY chip, and we made absolutely no effort to minimize the power consumption of the present platform design.

We presented an FPGA implementation of the network protocols required to connect hardware to an internet. We compared our design with other possible implementations using standard processors. We have demonstrated that FPL can be used to provide network protocols and applications that are currently implemented using software and a general-purpose microprocessor. Complete details of the XCoNet project are documented on the Web [25].

References

1. IDC http://www.idc.com:8080/Press/Archive/jun15c.htm
2. Brebner, G. and N. Bergmann. 1999. Reconfigurable Computing in Remote and Harsh Environments. Ninth International Workshop on Field Programmable Logic and Applications (FPL99).
3. Lockwood, J. W., J. S. Turner, and D. E. Taylor. 2000. Field Programmable Port Extender (FPX) for Distributed Routing and Queuing. Eighth ACM International Symposium on Field-Programmable Gate Arrays (FPGA00).
4. Maly, K., C. Wild, C. M. Overstreet, H. Abdel-Wahab, A. Gupta, A. Youssef, E. Stoica, R. Talla, and A. Prabhu. Interactive Remote Instruction: Initial Experiences. 1996. Proceedings of the Conference on Integrating Technology into Computer Science Education.
5. McHenry, J., P. Dowd, T. Carrozzi, F. Pellegrino, and W. Cocks. 1997. An FPGA-Based Coprocessor for ATM Firewalls. The Fifth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM97).
6. Miyazaki, T., K. Shirakawa, M. Katayama, T. Murooka, A. Takahara. 1998. A Transmutable Telecom System. International Workshop on Field Programmable Logic and Applications (FPL98).
7. Comer, D. E. 1995. Internetworking with TCP/IP Vol. I: Principles, Protocols, and Architecture. 3rd edition. Prentice Hall. ISBN: 0132169878.
8. Stevens, W. R. 1994. TCP/IP Illustrated, Vol. 1. The Protocols. Addison-Wesley. ISBN: 0201633469.
9. Level One Ethernet Transceiver. See http://www.level1.com/product/pdf/lxt970ad.pdf
10. Kumar, S., L. Pires, S. Ponnuswamy, C. Nanavati, J. Golusky, M. Vojta, S. Wadi, D. Pandalai, and H. Spaanenburg. 2000. A Benchmark Suite for Evaluating Configurable Computing Systems—Status, Reflections, and Future Directions. Eighth ACM International Symposium on Field-Programmable Gate Arrays (FPGA00).
11. Microchip. See http://www.microchip.com/Download/Lit/PICmicro/12C5XX/40139e.pdf
12. IPIC. See http://www-ccs.cs.umass.edu/~shri/iPic.html
13. Smith, M. J. S. 1997. Application-Specific Integrated Circuits. Reading, MA: Addison-Wesley. ISBN 0201500221. TK7874.6.S63.

14. Dolphin Flop805X Core. See http://www.support.xilinx.com/products/logicore/alliance/dolphin/flip805x-pr.pdf
15. Synopsys 8051 Core. See http://www.synopsys.com/products/designware/8051_ds.html
16. SuperTask RTOS. See http://www.ussw.com/products/supertask.
17. CoreEl Fast MAC Cores. See http://www.xilinx.com/products/logicore/alliance/coreel/cs1100.pdf
18. Express Logic ThreadX RTOS. See http://www.expresslogic.com/threadx.html. ThreadX on an ARM7 is about 4 kbyte. ThreadX on an ARC processor is 4-25 kbyte.
19. ESA, European Space Agency Leon SPARC V8 Development. See http://www.estec.esa.nl/wsmwww/leon/
20. Xilinx Virtex family datasheet. See http://www.xilinx.com/partinfo/ds003.pdf and application note on parallel configuration, http://www.xilinx.com/xapp/xapp137.pdf
21. RFC 691 FTP, RFC 768 UDP, RFC 783 TFTP, RFC 791 IP, RFC 793 TCP. See http://www.cis.ohio-state.edu/htbin/rfc/rfcXXX.html
22. Comer, Douglas E. and. David L. Stevens. 1998. Internetworking With TCP/IP: Design, Implementation, and Internals. 3rd edition. Vol 2. Englewood Cliffs, NJ: Prentice Hall. ISBN: 0139738436
23. iReady Internet Tuner. See http://www.iready.com/products/internet_tuner.html
24. Dallas Semiconductor. See http://www.dalsemi.com/DocControl/PDFs/87c520.pdf
25. XCoNET. See http://www-ee.eng.hawaii.edu/~msmith/XCoNET/XCoNET.htm