A SOHO ROUTER IMPLEMENTATION ON MOTOROLA MCF5272 PROCESSOR AND UCLINUX OPERATING SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET NAZİR KAÇAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Mübeccel Demirekler
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Cüneyt F. Bazlamaçcı
Supervisor

Examining Committee Members

Prof. Dr. Hasan Güran

Prof. Dr. Semih Bilgen

Prof. Dr. Uğur Halıcı

Asst. Prof. Dr. Cüneyt F. Bazlamaçcı

Kürşad T. Tüzer (MSc)

ABSTRACT

A SOHO ROUTER IMPLEMENTATION ON MOTOROLA MCF5272
PROCESSOR AND UCLINUX OPERATING SYSTEM

Kaçar, Mehmet Nazir
M.S., Department of Electrical and Electronics Engineering
Supervisor: Asst. Prof. Dr. Cüneyt F. Bazlamaçcı

September 2003, 68 pages

Recently, various special purpose processors have been developed and are frequently being used for different specialized tasks. Prominent among these are the communication processors, which are generally used within an embedded system environment. Such processors can run relatively advanced and general purpose operating systems such as uCLinux, which is a freely available embedded Linux distribution. In this work, a prototype SoHo (Small office / Home office) router is designed and implemented using Motorola MCF5272 as the core communication processor and uCLinux as the operating system. The implementation relies purely on the existing hardware resources of an available development board and the publicly available open source utilities of uCLinux. The overall development process provides an embedded system implementation and configuration example.

Keywords: Communications Processor, Embedded System Design, Linux, SoHo Router.

ÖZ

MOTOROLA MCF5272 İŞLEMCİ VE UCLINUX İŞLETİM SİSTEMİ ÜZERİNDE BİR SOHO YÖNLENDİRİCİ UYGULAMASI

Kaçar, Mehmet Nazir
Yüksek Lisans Tezi, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Yrd. Doç. Dr. Cüneyt F. Bazlamaçcı

Eylül 2003, 68 Sayfa

Yakın geçmişte, çeşitli özel amaçlı işlemciler geliştirilmiştir ve değişik özel işler için sıklıkla kullanılmaktadırlar. Bunların arasında en çok öne çıkan tür, genellikle gömülü sistemlerde kullanılan iletişim işlemcileridir. Bu tür işlemciler, serbestçe kullanılabilen bir gömülü Linux dağıtımı olan uClinux gibi, gelişmiş ve genel amaçlı işletim sistemlerini çalıştırabilmektedirler. Bu çalışmada, çekirdek iletişim işlemcisi olarak Motorola MCF5272 ve işletim sistemi olarak da uCLinux kullanılarak bir SoHo (küçük ofis / ev ofisi) yönlendiricisi tasarlanmış ve gerçekleştirilmiştir. Bu uygulama, tamamen, varolan bir geliştirme ortamının kullanılabilir donanım kaynaklarına ve uCLinux'un herkesçe kullanılabilir açık kaynak kodlu yardımcı programlarına dayanmaktadır. Bütünüyle geliştirme süreci bir gömülü sistem gerçeklemesi ve düzenleşim örneği sunmaktadır.

Anahtar Sözcükler: İletişim İşlemcisi, Gömülü Sistem Tasarımı, Linux, SoHo Yönlendirici.

ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervisor Asst. Prof. Dr. Cüneyt F. Bazlamaçcı for his guidance and constructive criticism throughout this study.

My special thanks and appreciation goes to my family, their support and encouragement, not only in this work, but during all my life, deserves a lifetime for repayment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADSL	:	Asymmetric Digital Subscriber Line
API	:	Application Programming Interface
ARP	:	Address Resolution Protocol
BDM	:	Background Debug Module
BOOTP	:	Bootstrap Protocol
BPS	:	Bits Per Second
CD	:	Compact Disc
CMOS	:	Complementary Metal Oxide Semiconductor
COFF	:	Common Object File Format
COM	:	Communications
COTS	:	Commercial Off-The-Shelf
CPU	:	Central Processing Unit
CS	:	Chip Select
CTS	:	Clear-To-Send
DC	:	Direct Current
DCD	:	Data Carrier Detect
DCE	:	Data Communications Equipment
DHCP	:	Dynamic Host Configuration Protocol

DMA	:	Direct Memory Access
DNS	:	Domain Name System
DSP	:	Digital Signal Processing
DTR	:	Data Terminal Ready
ELF	:	Executable and Linking Format
FIFO	:	First In First Out
FTP	:	File Transfer Protocol
GNU	:	GNU's Not Unix
GPIO	:	General Purpose Input Output
GPL	:	General Public License
HTTP	:	HyperText Transfer Protocol
IDE	:	Integrated Drive Electronics
IETF	:	Internet Engineering Task Force
IP	:	Internet Protocol
IPCP	:	Internet Protocol Control Protocol
ISDN	:	Integrated Services Digital Network
ISP	:	Internet Service Provider
JTAG	:	Joint Test Action Group
LAN	:	Local Area Network
LCP	:	Link Control Protocol
MAC	:	Medium Access Control
MII	:	Media Independent Interface
MIPS	:	Million Instructions Per Second
MHz	:	Mega Hertz (10^6 Hertz)

MMU	:	Memory Management Unit
NAK	:	Negative AcKnowledgement
NAPT	:	Network Address Portmap Translation
NAT	:	Network Address Translation
NCP	:	Network Control Protocol
NFS	:	Network File System
OEM	:	Original Equipment Manifacturer
OS	:	Operating System
PBGA	:	Plastic Ball Grid Array
PC	:	Personal Computer
PDA	:	Personal Digital Assistant
PIC	:	Position Independent Code
PLIC	:	Physical Layer Interface Connector
POS	:	Point-Of-Sale
POSIX	:	Portable Operating System Interface
PPP	:	Point-to-Point Protocol
PPPD	:	Point-to-Point Protocol Daemon
PWM	:	Pulse Width Modulation
QSPI	:	Queued Serial Peripheral Interface
RAM	:	Random Access Memory
RAMFS	:	Random Access Memory File System
RISC	:	Reduced Instruction Set Computer
ROM	:	Read Only Memory
ROMFS	:	Read Only Memory File System

RPM	:	RedHat Package Manager
RTS	:	Request-To-Send
SCSI	:	Small Computer System Interface
SDK	:	Software Development Kit
SDRAM	:	Synchronous Dynamic Random Access Memory
SIM	:	System Integration Module
SLIP	:	Serial Line Internet Protocol
SOHO	:	Small Office Home Office
SPARC	:	Scalable Processor Architecture
SSH	:	Secure Shell
TCP	:	Transport Control Protocol
TDM	:	Time Division Multiplexing
TFTP	:	Trivial File Transfer Protocol
UART	:	Universal Asynchronous Receiver and Transmitter
UDP	:	User Datagram Protocol
URL	:	Uniform Resource Locator
USART	:	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	:	Universal Serial Bus
VOIP	:	Voice Over Internet Protocol
VPN	:	Virtual Private Network
WAN	:	Wide Area Network
XIP	:	Execute In Place

CHAPTER 1

INTRODUCTION

Embedded systems are becoming more and more complex and powerful with every generation and many new features are constantly being put into these systems day by day. As applications get much bigger and more complex, it is now essential to have an actual operating system (OS) in such devices to keep the development time reasonable. In today's embedded systems, OS requirements are also getting higher and higher and traditional embedded operating systems like DOS are becoming obsolete. Commercial kernels provide very limited or minimal multitasking environments and non-standard application programming interfaces (API's). Therefore, general purpose operating systems are preferred to a greater extent to satisfy the increasing OS requirements of modern embedded systems. Windows, Linux and Solaris have all developed embedded versions.

Complex embedded systems also require more enhanced processor tasks. Motorola Coldfire family of processors with many on-chip peripherals like SDRAM controllers, timers, interrupts, GPIO, DMAs, etc. provides a suitable platform for various embedded systems such as Internet access devices, industrial equipment, test and measurement devices, printers, disk drives, digital set top boxes and audio electronics.

The present study is about implementing a SoHo router on uCLinux, one of the embedded versions of the Linux operating system, using a powerful and cheap embedded processor, namely the Motorola Coldfire MCF5272.

SoHo routers are used to connect the local area network (LAN) devices in a small office or home office environment to wide area network (WAN) or to the Internet and manage the interconnections between these LAN devices in an economical and user-friendly manner. These devices provide basic router functionality in addition to the flexibility and reliability of the Linux system. They often integrate a broadband modem with a router in the same box but models without an onboard modem are also present. ADSL, Cable, ISDN or simple modem models are available.

A high-end SoHo router often provides voice capability, multiple channel control, NAT (network address translation), DHCP (dynamic host configuration protocol), compression algorithms, security, encryption and a privacy firewall to support secure e-commerce transactions and access from authorized service providers. It also boasts remote management capabilities. A basic SoHo router, on the other hand, often includes the following features. It makes it possible for various LAN devices to share the internet connection; supports in-home file/print/device sharing; hosts a secure web server; provides VPN connectivity to/from workplace; provides firewall security (IP packet filter) and URL filtering (parental protection) in addition to web based gateway configuration and management, print server functionality and automatic network address translation (NAT).

The present implementation relies purely on the existing hardware resources of an available development board for MCF5272 and the publicly available open source utilities of uCLinux. By using a bootloader program for uCLinux, the prototype device is made to function as a standalone system. For this standalone device, no other additional special purpose hardware or software components other than some configuration and make files and a modem have been used. This way, if the prototype is to be mass produced, it should not be very difficult to design a minimal cost hardware/software base around the core processor.

While building the SoHo router prototype, a desktop Linux system is also required for compilation and experimentation. The system set up for this purpose uses RedHat Linux 8.0 and in addition to being a general purpose server, it also provides a download server for code image downloading to MCF5272 evaluation board. uCLinux operating system is configured and compiled on this RedHat Linux machine and the code downloading to MCF5272 is achieved using the ethernet download facility of MCF5272 evaluation board.

The overall development process provides an embedded system implementation and configuration example using uCLinux.

The thesis study is organized as follows:

Chapter 2 presents the embedded systems market and gives a brief description of embedded system design challenges. Features of Motorola's Coldfire family MCF5272 processor and its associated evaluation board are also summarized.

Chapter 3 explains in detail what a SoHo router is and lists all its features. Background information such as IP sharing, DHCP, VPN and dial-up networking are also included.

Chapter 4 first explains Linux and embedded Linux and then presents the uCLinux operating system including its benefits, features and applications.

In Chapter 5, the work environment necessary for the system development and implementation is explained. Tasks in preparing this environment such as setting up the host computer, the evaluation board and other interconnections are described in detail.

Configuration and compilation of the uCLinux OS are presented in Chapter 6. Technical details in the kernel configuration, application configuration, the application config files and compilation of the OS are all explained in sequence.

Chapter 7 is composed of the final prototype system diagram and its configuration, which is used for prototype testing, product demonstration and the overall system evaluation.

Chapter 8 concludes the study, presents the opinions that are gathered about uCLinux operating system throughout the study and states some possible future work.

CHAPTER 2

EMBEDDED SYSTEM DESIGN

2.1 Embedded Systems Market

Embedded systems constitute the computing power existent in every part of our daily lives: in offices, department stores, factories, hospitals, homes and automobiles. Supporting all these diverse environments means meeting equally diverse hardware and software requirements, ensuring ease-of-integration despite the flexibility. System reliability is also critical in many embedded systems, which for example handle confidential or critical data.

With advances in low cost/high performance 32 and 64-bit processors, wireless technologies and sophisticated software applications, embedded systems are now among the fastest growing and the most widely discussed technologies. The following market data represents the incredible market growth in embedded systems. [8]

- Personal Digital Assistants (PDA’s) production has doubled in years 2000 and 2001.

- Digital TV boxes and game consoles have grown at a rate of 44% in 2001.
- The number of wireless handsets are projected to reach 1.5 billion by the year 2004.

While the embedded systems market is expected to continue to grow with increasing use of devices such as Internet appliances, wireless handsets, network processors, set-top boxes and automotive navigation and communication systems, the PC market is expected to achieve modest growth rates. Within the next few years, double or triple digit growth rates are estimated, resulting in multi-billion dollar markets. In total, PC microprocessors form only less than %1 of all the processors sold worldwide. The remaining 99% are embedded processors sold for various different purposes, which are generally programmed by using C (80%), assembly (75%) and C++ (49%). [7,9]

2.2 Embedded Operating Systems

Every new generation of embedded systems is getting more complex and powerful and many new features are continuously being put into such systems day by day. Since applications get much bigger and more complex it is now essential to have a very capable operating system in these devices.

With the need for an embedded operating system, developers began to write homebrew OS-like non-standard software at the beginning and nearly every system had its own OS, none of them being compatible with each other. Then came the COTS (Commercial Off-The-Shelf Software) revolution and companies started to shift away from home-grown operating systems. Instead of investing money into OS R&D, they bought software components to build similar operating systems.

There are three types of embedded systems classification according to their OS capability requirements. Consumer type systems need low-cost, low-performance operating systemss while industrial embedded systems need better testing and operability. Military or aerospace system OS needs are much different such as reliability, stability, often cost not being a problem.

As the embedded systems market grew, different operating systems with different features have been developed. Systems like set-top boxes, kiosks or thin-clients are more PC-like and the operating system for them have features similar to desktop OSes. Windows NT Embedded, Windows XP Embedded and Linux are already being used in such systems [9].

Systems like cellular phones, PDAs or broadband routers require much smaller OS footprint and some real-time capabilities. Since they have limited RAM and no hard drive at all, optimum usage of resources is needed. PocketPC, PalmOS, Symbian and DOS are already used on these devices. Recently, WindowsCE and Linux are being used widely for their programmer availability and application repository [9].

Hardened real-time systems used in missiles, satellites, vehicles, robots and industrial machinery require even smaller footprint, critical reliability and a fully preemptive kernel. Operating systems like VxWorks, QNX, WindowsCE, Integrity and Phar Lap satisfy these requirements. Recently, Linux is also preferred and used as a real-time operating system by adding some extensions and modification to its kernel [9].

In embedded systems, hard drives are often replaced with some kind of flash, typically ranging from 512KB to 512MB and have limited lifespan on write access. Also, RAM is very precious, programs must be executed from the ROM or Flash, which is called Execute in Place (XIP). While choosing an operating system, hardware needs, real-time requirements, fault tolerance, user interface needs and time to market should all be considered simultaneously.

2.3 Embedded Processors

2.3.1 Motorola Coldfire Processors

The Motorola Coldfire processor family is Motorola’s 32-bit processors since 1996. Electronic device manufacturers have preferred and used Coldfire processors in a variety of devices ranging from inkjet printers to digital set-top boxes to market their products quickly and cost effectively. Integrated standard peripherals and development tools with small price/performance ratio provide a creative and free environment for the system designer who wants to find the distinctive solutions to meet their customers’ needs.

The ColdFire instruction set was developed as a subset of the 68K family instruction set since Motorola had a large customer base with 68K programming experience. This compatibility provides an easy way to upgrade system performance without much software development cost [10].

The ColdFire architecture offers a wide range of functionality for low-end to high-end applications. With broad range of performance levels from 10 to over 300 MIPS, ColdFire devices provide a range of integration levels with some on-chip peripherals. These peripherals include SDRAM controllers, timers, interrupts, GPIO, DMAs, JTAG, CSs (chip selects), and the BDM (background debug module). With these powerful peripherals, ColdFire processors seems ideal for a broad range of applications, like Internet access devices, industrial equipment, test and measurement devices, printers, disk drives, digital set top boxes and audio electronics.

The architecture is supported by a complete package of tools, including compilers, assemblers, linkers, debuggers, code converters, simulators, and evaluation kits, which all together accelerate the design process and usually help products get to market faster.

2.3.2 MCF5272 Processor and M5272C3 Board

The Motorola MCF5272 processor is a Static Version 2 ColdFire variable-length RISC processor with 32-bit address and data paths and 66 MHz core and bus frequency. The core has sixteen general-purpose 32-bit data and address registers, and a multiply-accumulate unit (MAC) for DSP and fast multiply operations. There is 4 KB SRAM and 16 KB ROM on CPU internal bus, in addition to 1 KB instruction cache in the core [11].

MCF5272 operates at 3.3V, from DC to 66 MHz using an external CMOS oscillator, packaged in a compact ultra low-profile 196 ball-molded plastic ball-grid array package (PBGA) and can operate from 0 to 70 degrees Celsius.

The chip has strong power management features with processor sleep and whole-chip stop modes. When in low-power sleep mode, the chip can give very rapid response to an interrupt given, i.e. can wake up instantly. With the use of clock enable/disable signals each on-chip peripheral can be shut off when not used. Even the external clock can be disabled via software for virtually zero power consumption.

The chip has two UARTs based on MC68681 dual-UART programming model, with full-duplex operation, flexible baud rate generator, processor interrupt and wake-up capability. Modem control signals (CTS and RTS) are available. There are 24-byte enhanced Tx and Rx FIFOs.

The on-chip ethernet module has half or full duplex 10baseT and half or limited full duplex 100baseT capabilities. The transmit and receive FIFOs are also on-chip. The module has a Media-independent interface (MII). The on-chip USB module is fully compatible with USB 1.1 specifications with endpoint FIFO and with selectable on-chip analog interface and 12 Mbps full-speeed.

External memory interface is glueless, supporting 8, 16 and 32-bit SRAM and ROM interface bus. SDRAM controller supports 16 to 256 Mbit devices with external bus configurable for 16 or 32 bits. There exists a queued serial peripheral interface (QSPI), 4x16 bit general purpose multi-mode timer with processor interrupts, software watchdog timer, three channel pulse width modulation (PWM) unit, system integration module (SIM), physical layer interface controller (PLIC), and IEEE 1149.1 boundary scan test access port (JTAG) for board-level testing.

Motorola's M5272C3 Board is a single board computer based on the MCF5272 ColdFire processor. It may be used in a variety of applications. With the addition of an RS-232 compatible terminal, it serves as a complete microcomputer system, for reference design, development, evaluation, training and educational use [4].

System memory consists of one on-board 16-Mbits (2 MB) AM29PL160C Flash ROM and 16 MB on-board SDRAM in addition to the on-chip 4 KB SRAM of MCF5272. Board also has a footprint for a 512 KB FSRAM which can be populated according to the developer's needs.

The board includes RS232 drives/receivers and DB9 connectors connected to the MCF5272's to built-in UARTs. UART0 functions as the "TERMINAL" channel used by the debugger to communicate with an external terminal or personal computer.

MCF5272 has 48-bits of general purpose parallel I/O lines. Each pin can be individually programmed as input or output. These GPIO signals are configured as three 16-bit wide ports, which at the same time, are shared by USART, USB, TDM, data bus and Ethernet controller signals. All the signals that exist on the board are available through the 120-pin expansion connectors.

The M5272C3 board supplies a USB transceiver and a JR1 connector which is directly connected to MCF5272 processor. An on-board level-one ethernet controller LXT971L is also supplied which is again connected to the processor. The on-board dBUG ROM monitor allows the user to download files from a network to memory in S-Record, COFF, ELF or raw binary formats. The board requires +5V to 14V DC with a minimum of 1A to operate properly. No item other than an RS232 terminal is required.

CHAPTER 3

SOHO ROUTERS

3.1 SoHo Router

SoHo routers are used to connect the local area network (LAN) devices in a small office or home office environment to wide area network (WAN) or to the Internet and manage the interconnections between these LAN devices in an economical and user-friendly manner. SoHo routers often integrate a broadband modem with a router in the same box but models without an onboard modem are also present. ADSL, Cable, ISDN or simple modem models are available.

A high-end SoHo router often provides voice capability, multiple channel control, NAT (network address translation), DHCP (dynamic host configuration protocol), compression algorithms, security, encryption, and a privacy firewall to support secure e-commerce transactions and access from authorized service providers. It also boasts remote management capabilities. These routers are usually not dependent on a PC. A basic SoHo router, on the other hand, often include the following features. It makes it possible for various LAN devices to share the internet connection; supports in-home file/print/device sharing; hosts a secure web server; provides VPN connectivity to/from workplace; provides firewall security (IP packet filter) and URL filtering (parental protection) in addition to web based gateway configuration and management, print server functionality and automatic network address translation (NAT).

3.2 SoHo Market Overview

SoHo is an acronym that stands for “small office/home office”. The term encompasses a range of entrepreneurial activities and business structures, from individually working employees to companies up to 20 employees. These companies include home-based businesses, free agents, independent contractors, telecommuters, e-lancers, small dot-com companies and other independent professionals.

The people working in SoHo companies vary from 25 to 40 million people [13]. The need for home servers comes from the telecommuters and day extenders working from home or remote offices who connect to corporate LAN through the internet. They use applications such as e-mail, file, directory services, web or internet transactions, and voice communications.

SoHo businesses are demanding more bandwidth for internet services like voice, video, and data, for applications such as streaming video, web browsing, e-mail, VoIP (voice over IP), video-on-Demand, MP3 files, online gaming and shopping. In the year 2003, even though the service provider markets have slowed down, the low-end router segment (including SoHo and branch office routers) continued to increase [14]. SoHo routers are used to connect the LAN (local area network) devices to the Internet or WAN (wide area network) and manage the interconnections between LAN devices. They comprise the largest percentage of total router unit shipments and new firms entering to the market are competing with Cisco's routers.

According to the statistics, the residential gateway market is estimated to exceed $8.9 billion by the end of 2003. Average Selling Prices (ASP's) of low-end routers decreased by 42% in the first half of 2003. More than 60% of SoHo routers sold are shipped to North America. The top three firms in the market are Linksys, Cisco and Zyxel [14].

3.3 SoHo Router Features

3.3.1 IP Sharing

IP Sharing, also known as IP Masquerading, is a particular case of NAT (Network Address Translation) and is a common technique to make multiple computers transparently share an Internet connection. The computer that performs the masquerading function is often known as a "NAT box".

NAT relies on rewriting IP addresses of packets passing through a router or firewall. This became necessary because the number of IP addresses are too few to cover all computers to be connected to the Internet after the number of computers connected to the Internet increased exponentially. NAT is vital in countries other than USA, where the assigned IP addresses are relatively too few. Personal and SoHo routers usually provide NAT as their core function. In addition to necessity, some arguments proposed in favor of NAT are simplicity and security. Since NAT changes the IP addresses in the packets, all computers connected to the NAT box are seen as a single device.

NAT translates the addresses in both outgoing and incoming datagrams by replacing the source address in outgoing datagrams with the NAT box's global IP address and replacing the destination address in each incoming datagram with the private address of the correct computer. To perform this mapping, NAT maintains a translation table. Each entry in the table is composed of two items; IP of a host on the internet and IP of a host on the subnet of the NAT box (Intranet). There are a number of ways to create this table:

- Manual initialization: The network administrator configures the table manually before communication occurs.
- Outgoing datagrams: When the NAT box receives a datagram to be sent to the Internet, NAT creates an entry in the translation table to record the address of the host and the address of the destination.
- Incoming name lookups: When a host on the Internet searches for the IP address of an internal host by sending domain name information, the DNS software creates an entry in the NAT table and answers the request by sending the NAT box's IP address.

The NAT method described so far allows at most one computer to access a site at any time and called 1-to-1 NAT, basic NAT or static NAT. To have many computers access to Internet at the same time, new methods should be devised. The number of IP addresses that a NAT box has can be increased. If it has K valid IP's, it allows up to K internal hosts to access to a given destination. But this increases the cost significantly and makes no sense at all.

A better approach is the port-mapped NAT, also called NAPT. In this approach, the NAT box translates TCP or UDP protocol port numbers as well as addresses. NAT gives a unique port number to each connection. When a connection is made by two internal hosts to an internet computer, two entries in the NAT table are created. The following packets that the internal hosts generate

(192.168.1.10, 23023, 144.122.166.76, 80)
(192.168.1.11, 386, 144.122.166.76, 80)

become as follows after translation is performed by the NAT box

(144.122.116.93, 14003, 144.122.166.76, 80)
(144.122.116.93, 14010, 144.122.166.76,80)

where 144.122.116.93 is the NAT box, 144.122.166.76 is the remote host and 192.168.1.10 and 11 are internal host IP addresses. An application protocol that passes IP addresses or protocol port numbers as data will not operate correctly accross NAT.

The benefits of NAT are great. It allows many computers to access the Internet utilizing only a single IP address – only one ISP account – on the internet. This means saved money for the user. Another benefit of NAT is the ability to conceal the internal configuration of user's network from external observers such as hackers. NAT can also be used to redirect connections pointed at some server to randomly chosen servers to do load balancing. NAT can be used to redirect HTTP connections in the Internet, targeted to a special HTTP proxy, which will be able to cache content and filter requests.

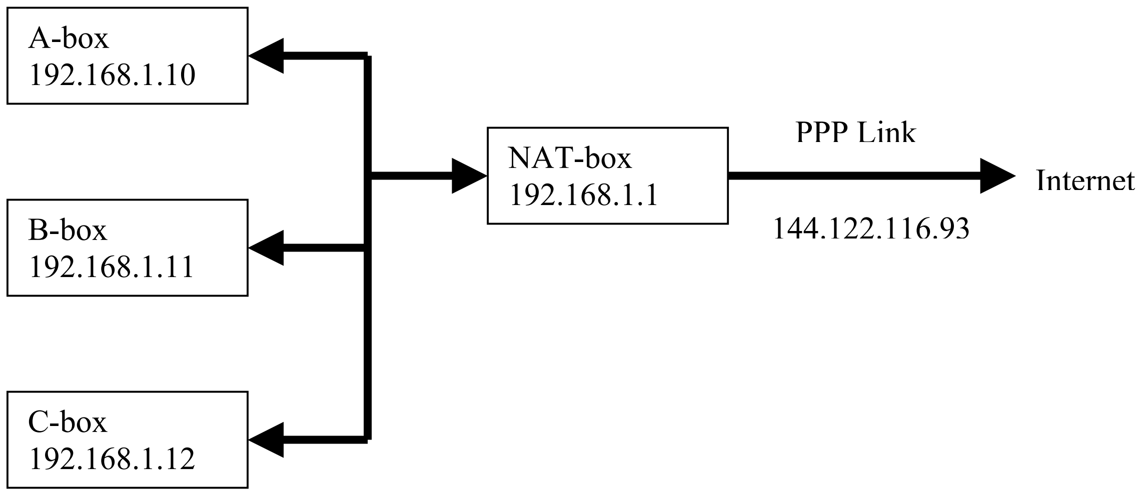


Figure 1. NAT box used as an Internet Gateway

In figure 1, there are three boxes that are connected to the NAT-box. The NAT-box is connected to Internet via a PPP link and does IP sharing. A valid IP is obtained from PPP link dynamically. Intranet IP’s are distributed by the NAT-box, and are not valid Internet IP’s. A switch or hub can be used to connect all the boxes together.

3.3.2 DHCP Configuration

In an IP-based network, each device on the network, such as workstations, routers, firewalls and printers, requires a unique IP address. As the network expands, available addresses get used very quickly. Subnets should be added and IP address tables should be maintained in order to manage the network. But any change in the infrastructure will result in reassigning a significant number of addressses. Furthermore, non-working clients waste some of the available IP addresses and working clients cannot get any IP address because of scarce resources.

An approach has been developed to get all the processes obtain IP addresses automatically. The Dynamic Host Configuration Protocol (DHCP) allows flexible, self-configuring networks to be created. When a user logs on the network, the DHCP server assigns an IP address to that device. This address remains valid for a period of time. If the client is not active when the time expires, the server releases the IP address so that it can be assigned to another device. This period of time is called a “lease” and the server “leases” addresses.

DHCP supports three modes of operation to allocate IP addresses:

- **Manual:** Administrator assigns IP addresses for each device or each group of device on a network. When a device requests an IP address, a specific address is given according to its MAC address, if its MAC address is defined at the server. By this way, specific addresses for each device can be addressed, or, an IP address can be shared between devices that do not access to the system at the same time.
- **Automatic:** Each device gets an IP address from the server when it first contacts the server, and holds that IP forever. This is often useful for configuring a static network.
- **Dynamic:** When the server gets a request, it sends an IP address that remains valid for a preset time period. When the time expires, another IP address should be requested by the client. This is the most flexible way of using DHCP.

Mostly, a mixed version of manual and dynamic modes are used on intranets. Devices such as network printers, routers and gateways are given static IP's, often manually, and workstations are given dynamic IP adresses.

As a device or workstation connects to the network, it broadcasts a "DHCPDiscover" message, repeated with random delays to avoid simultaneous submissions. Then the server receives the DHCPDiscover message and responds with a DHCPOffer message containing a valid IP address information. Multiple offers can be generated if more than one server is present.

The requesting device or workstation receives the offer and generates a DHCPRequest message for the IP address it selects. Meanwhile, DHCP server can verify that the IP address is not in use. When the server receives the request, it responds with a DHCPAck message containing the length of time (lease time) that the given IP address is valid.

3.3.3 Dial-up Networking with PPP and ISP’s

Millions of individuals who own a PC at home and want to use internet connect to the internet using modems and dial-up telephone lines. Usually, the user’s home PC calls an Internet Service Provider (ISP), whose router acts as an internet host. With this approach, all internet services are available to the home PC.

For the dial-up host-router connection, a point-to-point data link protocol is used for framing, error control, and other necessary data link layer functions. In the Internet, SLIP and PPP are widely used for this purpose and in this work, PPP is the protocol used throughout this study.

PPP was devised by IETF to be an official Internet standard for point-to-point protocols. It handles error detection, supports multiple protocols, permits authentication, introduces a framing method and carries out IP negotiation. PPP provides a link control protocol (LCP) for bringing lines up, testing and negotiation of options, and bringing lines down. Also, a network control protocol (NCP) is provided to negotiate network layer options independent of the network layer protocol used [12].

After the PC calls the ISP, the ISP's modem answers the phone and a physical connection is established, PC sends a series of LCP packets encapsulated in PPP frames. This way, PPP parameters to be used are negotiated. Then, a series of NCP packets are sent to negotiate network layer options. Since the PC should run TCP/IP stack to connect to the Internet, it requires an IP address. The ISP assigns an IP address to the PC dynamically.

Completing all the negotiations and getting an IP address, the home PC is now an Internet host that can send and receive IP packets. When the connection is to be broken, NCP is used to break the network layer connection and free the IP address. Then, LCP is used to break the datalink layer connection, followed by hanging up the phone, releasing the physical layer connection.

CHAPTER 4

UCLINUX OPERATING SYSTEM

4.1 Linux Operating System

Linux is a new operating system that resembles the Unix operating system, developed originally for home PCs. Linux was written by Linus Torvalds and volunteer programmers across the world using Internet. The Linux operating system kernel and most of Linux system components are released under the GNU General Public License (GPL). Linux source code, including modifications, are available under the GPL or a compatible license.

Operating systems are intermadiate tools between the computer user and the computer hardware. They make the system convenient and efficient to use. They also decide which programs to run, manage memory, provide a file system interface, schedule hard disk access, manage network operations, allow programs to communicate, and provide security.

The name “Linux” means the operating system itself, the Linux kernel. A “Linux system” includes the kernel, system components, and essential applications. It is reasonably hard to hand-pick all the necessary components and applications since most of them are constantly under development and scattered across the Internet. For this reason, various collections of packages have been developed to provide easy installation. These collections are called “distributions” and they include the basic Linux system, system management utilities and a wide range of applications.

Linux is a full-featured, modern operating system. It supports all the newest computer technologies like Bluetooth and USB. It includes features like protected memory, multiprocessor support, paging, etc. Linux can be scaled from 386 processors upto massive servers. The open development environment favors optimized code and greater modularity, allowing users to pick the precise set of features they need for their system.

The open source environment allows broad oversight and flexible development efforts. Kernel improvements are fed back to the community. In just a few years, Linux has taken a significant percentage of the server market. Today, Linux runs on a variety of CPUs including PowerPC, Mips, Alpha, Sparc, M68K and ARM. Linux is POSIX compliant in order to maintain compatibility with other UNIX-like systems. With millions of users wordwide, Linux is the most popular UNIX-like OS in the world.

Linux systems are long been known for their stability, they are known for running months or even years without crashing, freezing or having to be rebooted [6]. The most common causes of reboots are power and hardware failures.

4.2 Embedded Linux

Embedding Linux means porting the Linux kernel code on a particular CPU and board. There are many companies that sell embedded Linux solutions. Mostly, the APIs and kernel codebase are the same as desktop Linux.

4.3 Embedded Linux Market

More than 30 companies have announced their own distributions, comprising the Embedded Linux Market. It is estimated that at least as many companies are developing homegrown Linux offerings, rejecting to partner with a distributor. By the nature of embedded systems, the market is further fragmented. Until Linux, the embedded markets have never had a unified software platform. Developers mostly built their software from scratch. More than 50 real-time and other operating systems compete, but home-brew operating systems are still more popular. Because of special and different needs of every embedded system, legacy operating systems do not fit.

The embedded linux kernels in the market are Linus Torvald’s kernel, MontaVista Linux, BlueCat Linux, uCLinux, Embedix, ELKS, REDICE-Linux, TimeSys Linux, RTLinux, RTAI and QLinux.

Commercial embedded linux distributions include MontaVista Linux from MontaVista, BlueCat Linux from LynuxWorks, Embedix from Lineo, RedHat Embedded Linux Developer Suite, uCLinux from Altera/Microtronix, Tynux from PalmPalm, XOE from Transvirtual, TimeSys Linux from TimeSys, Linu@ from Mizi Research, Coollinux from Coollogic, RedBlue Linux from Esfia, ControLinux from Red Flag, mLinux from MobileSoft, and REDICE-Linux from REDSonic. Also, there are a few non-commercial distributions like Embedded Debian Project, PeeWeeLinux, OpenZaurus, and Familiar from Handhelds.org.

The four biggest companies in the market are MontaVista, LynuxWorks, Lineo and to a lesser degree, Red Hat. The key advantage of commercial distributions are support meaning SDKs, i.e. ported distributions, board support packages, documentation, tools and professional services. The primary costs and challenges in business are for porting, driver and tool development. Most of the vendors stated also maintain real time Linux variants.

4.4 Why Embedded Linux?

Using Linux in embedded systems has many benefits. Firstly, Linux is a mature and robust OS in the PC OS market. It support a large number of devices, filesystems, and networking protocols. It has a large pool of users, upgrades to the kernel and applications are constantly being added, tested and refined. Also, there's complete visibility of the source code. Linux has very low cost, and in addition, a large number of applications exist which require little or no porting effort. It has the ability to run on generic hardware and this decreases the costs associated with purchasing development systems.

Aside from being an embedded OS, Linux is also used as a cross-development platform in the embedded market. The early approach for a typical development system consisted of a computer system dedicated to run an operating system supplied by the embedded chip manufacturer. These systems did also include debugging and emulation capabilities.

As low-cost general purpose personal computers become available, cross-assemblers and cross-compilers which ran on Unix or MS-DOS have emerged. These programs supported a whole host of microprocessor lines. Today, Linux is offering a fully featured multi-user, multi-tasking operating system environment, and vendors are porting cross-development tools. With the help of rich development tools available for Linux, the porting process has become easy.

Linux can also be used as a Real-Time OS with some extensions to the kernel. Although the non-deterministic nature of task scheduling makes it unsuitable for some hard real-time systems, Linux is ideal for tasks as report generation as well as networking and addressing interoperability issues, thanks to the wealth of utility programs included. Also there are implementations that run the Linux kernel as a task under a real-time OS, getting all the advantages of hard real-time along with the user interface, stability and huge application support of Linux.

Linux is being widely used in the embedded OSes market. It has been used in palmtop computers, cameras, set-top boxes, cellular and VoIP telephones, home or business routers, test equipment, scientific equipment, airborne imaging, clustered supercomputers, factory automation and even in gas pumps and POSes. In OEM arena, there exist single chip Linux processors, single board computers, custom Linux boxes, rackmount servers and multiprocessor Linux boxes [15].

4.5 uCLinux Operating System

The uCLinux OS was first run on a SCADA controller in February 1998, after the discussion of the possibility of implementing Linux on MMU-less processors to act as a low cost network controller driving data communications between ethernet and microwave systems [16] The Arcturus Networks company has maintained and led the open-source project until now. Partnering with the open source community SnapGear, the company developed a product line of internet appliances based on the Motorola ColdFire family. The uCLinux kernel has been deployed on various CPU architectures and platforms including ARM, MIPS, SPARC, SH and ETRAX.

Until uCLinux, developers of embedded systems lacking an MMU (Memory Management Unit) could not embed Linux into their products. Since component costs are of primary concern in embedded systems, which are required to be small and inexpensive, the complex and expensive microprocessors with on-chip MMU hardware are usually not being selected. uClinux is a variant of Linux running on MMUless processor architectures.

4.5.1 Features

uCLinux kernel has drivers for IDE and SCSI disks, CD drives, and floppy support. It makes a block device out of the memory range where the root filesystem resides, and mounts this as root. The root filesystem is in a read-only unix-like format called ROMFS. The kernel and user programs are run in-place, regardless of the device that they are put on.

The kernel can be much more network-based, the root file system can be served via NFS (Network File System). DHCP and BOOTP are also supported, and even the kernel can be downloaded and run from a server. The main console is mostly on the serial port. The root filesystem may include a telnet daemon, a web server, NFS client support and a collection of well-known Linux tools.

The kernel and user programs may be configured for the user needs. There may not be a main console, a network interface, or even an input device. The size of the OS image will vary according to the configuration and applications selected.

Typical sizes vary from 300-400 kilobytes to a few megabytes. The image is mostly downloaded on to a Flash RAM and booted with the use of a bootloader program. Developing software for uCLinux systems is mostly done by using a cross-compiler toolchain built from the GNU compiler tools. Software that builds under the famous gcc (GNU C Compiler) without giving any errors for x86 architectures on a host computer, often builds without modification for any uCLinux target.

A fundamental difference between uCLinux and mainstream Linux is the absence of MMU support. Luckily, this does not affect people developing kernel and user space programs significantly. But the lack of both memory protection and a virtual memory model are important for the developer, since certain system calls are affected. Operating without memory protection can be a major problem source, since an invalid pointer reference or an unprivileged process may cause address errors and corrupt or even shutdown the system. The code must be written carefully and tested to ensure robustness and security.

Lacking virtual memory, uCLinux must run the processes independently from their position in memory. Also, since very dynamic memory allocation can starve the system due to fragmentation, a preallocated buffer pool must be created, replacing *malloc* calls with buffer requests.

uCLinux has no *fork* system call. The *fork* call clones a process to create a child, but without an MMU, uClinux cannot reliably clone any process. To compensate the lack of *fork*, uCLinux implements *vfork*. The *vfork* call creates a child, but both the parent and the child process share the same memory space including the stack. Then, the parent's execution is suspended until the child process *exit*s. Programs making extensive use of *fork*'s abilities must be modified according to the new structure.

Since basic memory management functions are still needed, the reliance on MMU hardware must be eliminated by the kernel itself, providing the basic functions by software. Kernel and user memory allocation and deallocation routines are reimplemented. Transparent swapping and paging support are also removed. Program loaders are changed to support PIC (Position Independent Code) and a new binary object code format was created. Program loaders other than PIC are modified to use absolute references fixed up by the kernel during run time.

4.5.2 Applications

The application level API of uCLinux is just the same as the standard Linux system call set, which is fully compatible with any UNIX system. The exceptions to this are stated above. Thus, the system appears to be any other Linux system, which makes it simple to use uCLinux on embedded systems.

uCLinux has a custom libc library, which is light-weight, and includes all the usual functions included in a standard C library. Some functions seen on a standard UNIX/Linux systems are not implemented, instead, the missing functions will be filled as the need arises.

A large number of applications have been ported into the uCLinux environment. Some of them are listed in Table 1.

Command Shells	sash, bash, nwsh, minix-shell
Shell tools	busybox, shutils, fileutils, sysutils, stty
Network tools	ping, traceroute, ifconfig, route, telnet, dhcpcd, iptables
Network services	inetd, telnetd, dhcpd, pppd, slattach, discard
Web servers	boa, simple httpd, thttpd
System services	init, gettyd, agetty, login, tinylogin
VPN	PoPToP, Ipsec
Modem	diald, chat, tip
Filesystem	Mount/unmount, smbmount/smbunmount, e2fsprogs, samba
FLASH tools	flashw, netflash
Editors	levee (vi clone)
Audio	wave player, mp3 decoder/player
Debug	gdbserver

Table 1. uCLinux Applications

Most of these applications are standard Linux packages, just ported to uCLinux. Most of the time, cross-compiling the source is enough to port the application.

4.6 The Colilo Bootloader

The ColdFire Linux Loader (Colilo) is primarily used to boot uClinux on embedded systems based on the Motorola ColdFire processor. However, it is not Linux-specific, it may be used to boot any other ColdFire code or operating system.

The uCLinux operating system is designed to run on dBUG, Motorola's on board user interface. However, downloading the operating system every time the system is rebooted can be a burden. Since uClinux can not boot without dBUG, some sort of bootloader should be employed. Colilo comes in handy at this point, and does the trick.

After the ColdFire processor is reset, it begins to execute the code at 0x400 from the memory on Chip Select 0 (CS0). On the Motorola evaluation boards, the entry point of dBUG resides at this address. Colilo sets its entry point to this address, making dBUG disappear, and takes control. It configures the CS registers which are used to map memory chips to ColdFire's address range, and initializes SDRAM control registers [5].

Next, Colilo copies initialized data from flash to SDRAM, and clears the variable data section of SDRAM. This is done for setting up a proper C execution environment. Once the environment is set up, compressed operating system image is expanded into SDRAM. Lastly, Colilo gives the control to the operating system, executing the code in SDRAM.

In Table 2, configuration options for Colilo are given. Colilo must be configured for processor number, evaluation board vendor, evaluation board name, debugging and user interface. All the configuration parameters are written into the top-level makefile, as shown in Table 2.

After the makefile is configured, the `make` command is used to compile the bootloader. Resulting files are `colilo.bin`, `colilo.elf`, and `colilo.s19`.

Option	**Value**	**Line in Makefile**
ARCH	5272	ARCH = 5272
VENDOR	Motorola	VENDOR = Motorola
BOARD	M5272C3-16MB	BOARD = M5272C3-16MB
RAM_SIZE	16MB	RAM_SIZE = 16MB
CONFIG_FLASH	1	CONFIG_FLASH = 1
BOOTDEBUG	1	BOOTDEBUG = 1
CONFIG_UI	(Comment)	#CONFIG_UI = 1
MCF_FAST_CLK	(Comment)	#MCF_FAST_CLK = 0

Table 2. Colilo Bootloader Configuration Options

CHAPTER 5

SYSTEM DEVELOPMENT: THE WORK ENVIRONMENT

5.1 Preparing the Host Computer

To compile uCLinux, a Linux-based x86 or Pentium machine is needed with necessary libraries and compilers installed. In this thesis, Redhat Linux 8 on a Pentium 4 computer is used. RedHat is chosen due to its embedded distributions and wide support on the internet. RedHat 8 is configured and set-up as a server, with compiling options enabled.

5.1.1 Installing the uCLinux Sources

uCLinux sources can be downloaded from the uCLinux official web site [1] or using the following link: http://www.uclinux.org/pub/uClinux/dist/uClinux-dist-20030305.tar.gz. Updates can be found at uCLinux official web site or cvs.uclinux.org.

The first step in the installation process is the extraction of the file uClinux-dist-20030305.tar.gz to the workplace using the command

```
tar xzvf uClinux-dist-20030305.tar.gz
```

Then a sub-directory called uClinux-dist appears, containing the whole distribution.

5.1.2 Installing the m68k-elf-toolchain

The m68k-elf toolchain can be downloaded from the following link: http://www.uclinux.org/pub/uClinux/m68k-elf-tools/m68k-elf-tools-20030314.sh. Installation is done by logging in as the superuser (necessary for the installation) and then issuing the command:

```
sh m68k-elf-tools-20030314.sh
```

5.1.3 Installing the Tftp Server

The tftp server is required for program downloading to the M5272C3 board. By default, RedHat 8 does not install the tftp server. The RPM file for the installation can be found on Redhat 8 CD#3 in the RPM directory, or on the RedHat official web site [2]. The installation can be carried out by running the CD in X-window environment, or booting with CD#1 and selecting upgrade installation. However, the simplest approach for setting up the tftp server is using the following command in the directory where RPM file exists:

```
rpm –i tftpd-server-0.29-3.i386.rpm
```

If tftpd is required to run at startup, it should be enabled in system services. The machine should then be rebooted or xinetd, the internet services daemon, should be restarted instead [3]. However in this work, the service is started by first creating a directory named `/tftpboot` (a default name contained in the top level makefile) that will contain the files to be served and then by issuing the command:

```
        /usr/sbin/in.tftpd –l –a 144.122.16.76:69 –u root –s /tftpboot
```

Also, if a firewall exists, it should be configured to let tftp connections through.

5.2 Preparing the M5272C3 Card Setup

M5272C3 card is connected to the PC via the card’s RS232 port named “terminal” and one of the host Linux PC’s COM ports. A terminal emulator is required for this communication such as the Linux application minicom. Invoking of this application is done using the command `minicom –s` and the necessary minicom configuration is as follows: The serial device is /dev/ttyS0 (com 1), Bps/Par/Bits setting is 19200 bps, 8 data bits and no parity bits (19200 8N1), hardware flow control off, software flow control on. Then minicom can be started only by typing `minicom` after this setup is successfully completed and saved as default.

After the connection is established, the board is reset and the following prompt appears. To set up the tftp download configuration, the successive command set is applied.

```
Hard Reset
SDRAM Size: 16M

Copyright 1995-2001 Motorola, Inc.  All Rights Reserved.
```

```
ColdFire MCF5272 EVS Firmware v3a.1a.1a (Build 18 on May 14 2002
10:03:39)

Enter 'help' for help.

dBUG> set client 144.122.167.244

dBUG> set server 144.122.166.76

dBUG> set filename image.bin
```

Configuration is completed at this point. The command

```
dBUG> dn –i
```

downloads the uCLinux image into SDRAM and uCLinux starts with

```
dBUG> go 20000
```

5.3 Preparing the Modem and Connection Cable

In this thesis, a Datron external 56K modem is used. A modem is a DCE type RS232 device. The M5272C3 card's auxiliary RS232 connector is also a DCE type RS232 device, therefore a modified DCE-DCE cable as shown in Figure 2 is used since the UART of MCF5272 has no DTR/DCD signals. This way, modem is made to think that M5272C3 is always ready to transmit/receive data.

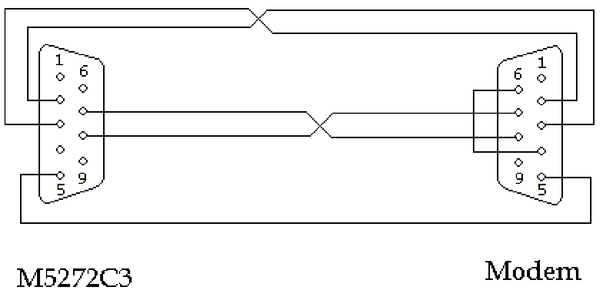


Figure 2. Modem Serial Connection Cable

CHAPTER 6

SYSTEM DEVELOPMENT: UCLINUX OS CONFIGURATION AND COMPILATION

Compilation operation consists of the following major steps:

- make menuconfig
    - Target platform configuration
    - Kernel configuration
    - Application configuration
- make dep (use the config files and make dependencies)
- make (the main compilation and image generation)

6.1 OS Configuration

Configuration of the uCLinux OS is performed while carrying out the above compilation steps. Issuing the `make menuconfig` command, `uClinux v1.3.4 Configuration` window appears, which allows the user to

- make the target platform selection
- load a saved configuration

- save the current configuration

The necessary parameter adjustments concerning the target platform are tabulated in Table 3. These changes configure the board, processor and kernel options that uCLinux will use in the intended system.

Vendor/Product	Motorola M5272C3
Kernel Version	linux-2.4.x
libc version	UClibc
Customize Kernel Settings	(Select)
Customize Vendor/User Settings	(Select)

Table 3. Main Configuration Options

Making the above changes, the exit option should be selected twice, saving the new configuration. This completes the first-level configuration.

6.1.1 Kernel Configuration

The next step is the Linux Kernel Configuration, which is performed through the `Linux Kernel v2.4.20-uc0 Configuration` windows that opens automatically after the first step. Table 4 tabulates the necessary changes.

Tab	**Option**	**Value**
Code Maturity Level Options	Prompt for development and/or incomplete code/drivers	(Select)
Processor type and features	CPU	MCF5272
	CPU Clock Frequency	66 MHz
	Motorola M5272C3 board support	(Select)
	RAM size	16 MB
	RAM bit width	32 bit

	Kernel executes from	RAM
General Setup	Networking Support	(Select)
	Sysctl Support	(Select)
	Kernel core (/proc/kcore) format	A.OUT
	Kernel support for flat binaries	(Select)
Block devices	RAM disk support	(Select)
	Default RAM disk size	4096
	ROM disk memory block device (blkmem)	(Select)
	FLASH type	AMD
	FLASH size	2 MB
	FLASH bit width	(16 bit)
Networking Options	Packet Socket	(Select)
	Network Packet Filtering (replaces ipchains)	(Select)
	Socket Filtering	(Select)
	Unix Domain Sockets	(Select)
	TCP/IP Networking	(Select)
	IP:TCP syncookie support (disabled per default)	(Select)
Networking Options IP:Netfilter Configuration	Connection Tracking (Required for masq/NAT)	(Select)
	FTP protocol support	(Select)
	IP tables support (required for filtering/masq/NAT)	(Select)
	Connection state match support	(Select)
	Unclean match support (experimental)	(Select)
	Packet filtering	(Select)
	REJECT target support	(Select)
	Full NAT	(Select)
	MASQUERADE target support	(Select)
	REDIRECT target support	(Select)
	Packet mangling	(Select)
	LOG target support	(Select)
Network device support	Network device support	(Select)

	Dummy net driver support	(Select)
	PPP support	(Select)
	PPP support for async serial ports	(Select)
	PPP support for sync tty ports	(Select)
	PPP deflate compression	(Select)
	PPP BSD-compress compression	(Select)
	SLIP support	(Select)
	CSLIP compressed headers	(Select)
Network device support Ethernet (10 or 100 Mbit)	Ethernet (10 or 100 Mbit)	(Select)
	FEC Ethernet controller (of Coldfire 5272)	(Select)
Character devices	ColdFire serial support	(Select)
Filesystems	/proc filesystem support	(Select)
	ROM filesystem support	(Select)
	Second extended fs support	(Select)
Kernel hacking	Full symbolic/source debugging support	(Select)
	Suppress Kernel bug messages	(Select)

Table 4. Kernel Configuration Options

The choices that does not exist in Table 4 but that appears during the configuration should not be selected.

6.1.2 Application Configuration

uClinux has a broad range of applications coming in bound. Some of these applications are needed for the router operation. Since the M5272C3 board has limited Flash memory, the remaining set should be avoided unless there is a definite need for them. The necessary applications for proper SoHo router operation are selected to be as listed in Table 5.

Tab	**Option**	**Value**
Core Applications	init	(Select)
	enable console shell	(Select)
	Shell program (Sash)	(Select)
	expand	(Select)
Network Applications	boa	(Select)
	dhcpd	(Select)
	inetd	(Select)
	ifconfig	(Select)
	iptables	(Select)
	pppd	(Select)
	route	(Select)
Miscellaneous Applications	chat	(Select)
Miscellaneous Configuration	RAMFS Image	512K

Table 5. Application Configuration Options

6.1.3 Addition of Necessary Config files

Both uCLinux and the selected applications need various configuration files to function properly. The directory for these files is generally the `/etc` directory with some files being in subdirectories.

"host.conf": Includes a single line `order hosts,bind`. This line is necessary for the DNS resolving process. It tells uCLinux to look to `/etc/hosts` file first and then ask to the DNS servers stated in `resolv.conf` file if a DNS lookup is to be performed.

“resolv.conf”: Includes IP addresses of the name servers that can be used to perform DNS lookups.

```
      /etc/resolv.conf:
            nameserver 144.122.199.20
            nameserver 144.122.199.90
```

“config/dhcpd.conf”: Contains three lines that gives network-specific information to the DHCP daemon. The subnet and router information are necessary since DHCP daemon passes them to its clients.

```
      /etc/dhcpd.conf:
            subnet 255.255.255.0
            router 192.168.1.1
            dns 144.122.199.20
```

“config/dhcpd.iplist”: This file contains the IP addresses that can be given to DHCP clients. The information should be coded in binary. The `makeiplist` tool given in appendix is used to create this file.

“config/dhcpd.leases”: Contains nothing except runtime. At runtime, DHCP daemon writes the client information into this file. Unfortunately, dhcpd can not create this file and should be created before compilation with the command below:

```
      touch romfs/etc/config/dhcpd.leases
```

This command creates the `dhcpd.leases` file with zero file size.

“ppp/options”: The options file contains pppd specific information and configuration. The lines and their usage are explained in Table 6.

Line	**Meaning**
asyncmap 0	Set character map to allow telnet/rlogin operation
crtscts	Use hardware flow control
lock	Use the UUCP-style lock on serial device being used
modem	Use modem control (i.e. connect using a modem)
nodetach	Do not use fork() to go background
proxyarp	Adds an entry to the ARP table containing the IP address of the client and the IP address of the NIC.
lcp-max-configure 30	Set the maximum number of LCP configure request transmissions
lcp-max-failure 30	Set the maximum number of LCP configure NAKs returned
defaultroute	Add a default route to system routing tables, when IPCP negotiation is fully completed.
ms-dns 144.122.199.20	Set the primary DNS server IP address
ms-dns 144.122.199.90	Set the secondary DNS server IP address
/dev/ttyS1	Set the serial port to be used.
57600	Set the serial port speed to be used
connect /etc/ppp/dial.modem	Use the given script to establish serial connection.

Table 6. PPP Options

“ppp/dial.modem”: Includes the script to establish the serial connection. The `chat` program is used to establish the connection. First line specifies the shell to be invoked while running the second line. The second line specifies the chat program to be used with the ‘chat script file name’ given as a parameter.

```
/etc/ppp/dial.modem:
#!/bin/sh
/bin/chat –v –f /etc/ppp/number
```

“ppp/number”: Includes the chat script to be used by the chat program while establishing the serial connection. The script used is given below and includes a single line that does all the trick. In table 7, each command given in the script is explained.

```
/etc/ppp/number:
“” “AT” “OK” “ATH0” “OK” “ATDT4300” “username:” “mkacar”
“password:” “thesis” “METU:” ppp
```

Command	**Response**	**Meaning**
“”	“AT”	Do not wait anything to happen and send the string “AT” i.e. check the line for connection.
“OK”	“ATH0”	Wait for an “OK” response and then send “ATH0” command i.e. hang up the modem in case any connection is present before.
“OK”	“ATDT4300”	Wait for an “OK” response and then send “ATDT4300” command i.e. dial the number 4300 using touch-tone dialing.
“username:”	“mkacar”	Wait for the string “username:” and then send the string “mkacar”, i.e. a valid username for the ISP
“password:”	“thesis”	Wait for the string “password:” and then send the string “thesis”, i.e. a valid password for the ISP
“METU:”	ppp	Wait for the string “METU:” and then send the string ppp to make the ISP switch to point-to-point protocol.

Table 7. Chat Script for serial connection

“rc”: This file is the first file that is executed during the system startup procedure. All the necessary commands to be executed at the startup are written into this file. Table 8 presents the commands in this file and their explanations.

Command	**Explanation**
hostname uClinux	set hostname to uClinux
/bin/expand /etc/ramfs.img /dev/ram1	uncompress the RAMdisk image to the RAM block device ram1
/bin/expand /etc/ramfs.img /dev/ram2	uncompress the RAMdisk image to the RAM block device ram2
mount –t proc proc /proc	mount the proc filesystem to directory /proc
mount –t ext2 /dev/ram1 /var	mount the device ram1 as ext2 to directory /var
mkdir /var/config	create a temporary directory to save configuration data
cp /etc/config/* /var/config	save configuration
mount –t ext2 /dev/ram2 /etc/config	mount the device ram2 as ext2 to directory /etc/config
cp /var/config/* /etc/config	restore configuration
mkdir /var/tmp	Create necessary directories under /var directory
mkdir /var/log	
mkdir /var/run	
mkdir /var/lock	
ifconfig lo 127.0.0.1	configure local loopback address as 127.0.0.1
ifconfig eth0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255	configure ethernet address, netmask and broadcast address
route add –net 127.0.0.0 netmask 255.0.0.0 lo	add the local loopback route to the kernel routing table
pppd &	start point-to-point protocol daemon
dhcpd –d eth0 &	start dynamic host configuration protocol daemon
iptables –F	flush (delete) all the rules present
iptables –t nat –F	flush (delete) all the rules present in the nat (connection) table
iptables –t mangle –F	flush (delete) all the rules present in the mangle (alteration) table
iptables –t nat –A POSTROUTING –o ppp0 –j MASQUERADE	append rule into nat (connection) table, route every packet to ppp0 after routing to masquerade
iptables –A INPUT –m state --state ESTABLISHED,RELATED –j ACCEPT	allow any existing connections or anything related
iptables –A INPUT –m state --state NEW –i ! ppp0 –j ACCEPT	allow new connections from the intranet

iptables –P INPUT DROP	set default rule that every packet should be dropped if it does not comply any other rules
iptables –A FORWARD –i ppp0 –o ppp0 –j REJECT	reject (give response with host unreachable) for any packets trying to loop in device ppp0
cp /etc/bir /proc/sys/net/ipv4/ip_forward	enable IP forwarding by writing a “1” into the ip_forward kernel switch. the /etc/bir file contains a “1” only.
cat /etc/motd	show the start-up banner

Table 8. The /etc/rc file

6.2 Compilation

Having set up all the necessary configuration parameters and putting all the configuration files into necessary places at this point, main compilation process may start. If the command `make clean` is employed first, a clean compilation is guranteed, i.e. no partial but full compilation from scratch. This is not necessary if uCLinux is being compiled for the first time or the changes made are minor. However if some major or minor change is performed in the Kernel or Application Configuration, this operation ensures a correct compilation.

Compiling the dependencies is the first step in the main compilation process, which is achieved by `make dep` command that reads every source file and writes the names of the files that the source file includes, directly or indirectly, to a file. Completing the dependencies, a `make` command is issued to obtain the uCLinux operating system image, the `image.bin` file. Then this file is stored into the folder `/tftpboot` for downloading using `tftp`.

6.3 OS Image and Bootloader Download Process

The dBUG monitor may be used for experimental purposes to run the uCLinux image without any need for a bootloader. Care should be taken when downloading the bootloader to the Flash because by default the Colilo bootloder overrides and destroys the dBUG monitor. Yet, the dBUG monitor can still be downloaded from Motorola's web site and be restored in a case it is again required. However in this thesis an alternative approach, explained in the following paragraphs, has been developed and adopted to keep both the dBUG monitor and the Colilo bootloader in FLASH simultaneously.

The 2MB Flash of M5272C3 is mapped between addresses 0xFFE00000 and 0xFFFFFFFF. The dBUG ROM monitor of M5272C3 resides at address 0xFFE00000, using the first 256Kbytes of Flash. M5272C3 board provides a jumper, J13, to execute user code from Flash. The default position of this jumper makes the board boot from address 0xFFE00000, i.e. the dBUG ROM monitor. When jumper's position is changed, board boots from address 0xFFF00000, the second half of Flash RAM where the user code can be placed.

In this approach, the uClinux image file goes under a series of operations. The gzip utility is used to compress the operating system kernel plus the ROM file system as follows:

```
    gzip -9 image.bin
    m68k-elf-objcopy -I binary -O srec --adjust-vma 0xfff40000
image.bin.gz imagegz.s19
```

These two commands generate an S19 downloadable file with download address 0xFFF40000.

The Colilo source code is slightly modified in order to keep the dBUG ROM monitor working and to make Colilo get the operating system from address 0xFFF40000. This modification is done by updating two files found in directory `vendors/Motorola/M5272C3-16MB/` under Colilo main directory as follows:

In file `board.c`:

```
source_addr = (unsigned char *)0xfff40000;
down_addr = (unsigned char *)0xfff40000;
```

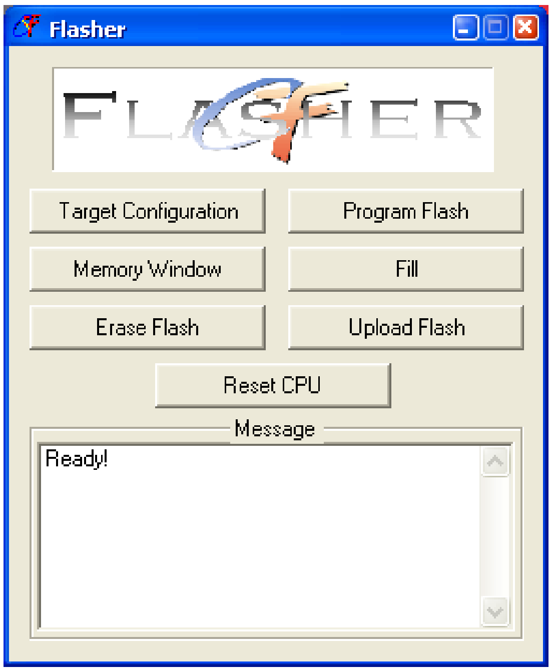
In file `Makefile`:

```
$(OBJCOPY) --input-target=binary --output-target=srec -
-adjust-vma 0xfff00000 \
```

After compiling with these options, the `colilo.s19` file is made to contain the download address 0xFFF00000.

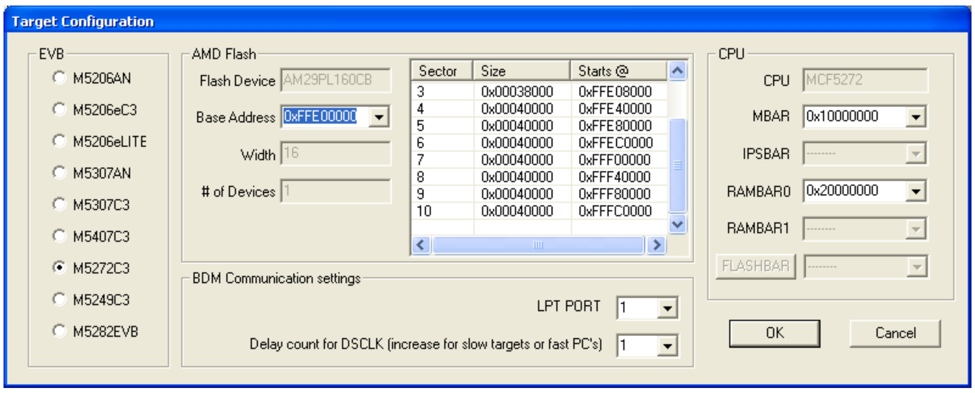
To download uCLinux image and the bootloader image to M5272C3's Flash RAM, the CFFLASHER program supplied by Motorola is used (Figure 3). Configuration of CFFLASHER is already done when M5272C3 option is selected from "Target Configuration" Window (Figure 4).

Flash memory should be erased before reprogramming. Using the "Erase Flash" window of CFFLASHER Flash regions 7, 8, 9 and 10 are erased. After completing the erase operation, Flash is programmed using the "Program Flash" window. Colilo and uClinux images are programmed seperately. Colilo image resides at Flash region 7, i.e. between addresses 0xFFF00000 and 0xFFF3FFFF. uClinux image resides at Flash regions 8, 9 and 10, i.e. between 0xFFF40000 and 0xFFFFFFFF. When programming finishes, the BDM cable is unplugged and the board is reset with jumper J13 position changed. Then, uClinux boots successfully.



Ready!

Figure 3. The CFFLASHER Program



EVB
- ( ) M5206AN
- ( ) M5206eC3
- ( ) M5206eLITE
- ( ) M5307AN
- ( ) M5307C3
- ( ) M5407C3
- [✓ M5272C3]
- ( ) M5249C3
- ( ) M5282EVB

AMD Flash

Flash Device	AM29PL160CB
Base Address	0xFFE00000
Width	16
# of Devices	1

Sector	Size	Starts @
3	0x00038000	0xFFE08000
4	0x00040000	0xFFE40000
5	0x00040000	0xFFE80000
6	0x00040000	0xFFEC0000
7	0x00040000	0xFFF00000
8	0x00040000	0xFFF40000
9	0x00040000	0xFFF80000
10	0x00040000	0xFFFC0000

BDM Communication settings

LPT PORT	1
Delay count for DSCLK (increase for slow targets or fast PC's)	1

CPU

CPU	MCF5272
MBAR	0x10000000
IPSBAR	--------
RAMBAR0	0x20000000
RAMBAR1	--------
FLASHBAR	--------

Figure 4. CFFLASHER Target Configuration

CHAPTER 7

PROTOTYPE TEST SETUP

For initial testing, a small local area network (LAN) composed of two computers, an 8-port 10/100 switch and the SoHo router is built. SoHo router is powered with a homebrew adaptor, which can be controlled from the host Linux PC via its parallel port and is connected to a Windows PC through the PC's parallel port and the M5272C3 board's BDM port for OS downloading purposes.

The homebrew adaptor can be opened or shut down by issuing a command on the host Linux PC. This property is used for experimenting with the board remotely, since host Linux PC is connected to the Internet and accessible worldwide via the SSH protocol.

The client computers are chosen to be Windows XP machines having P4 1800 MHz processors. They are configured to use DHCP to obtain dynamic IP addresses, gateway address and DNS server addresses. No other configuration for the client PC's other than the above is required. Actually, any type of computer and operating system can be used as the intranet client as long as it can employ a TCP/IP stack and can use DHCP.

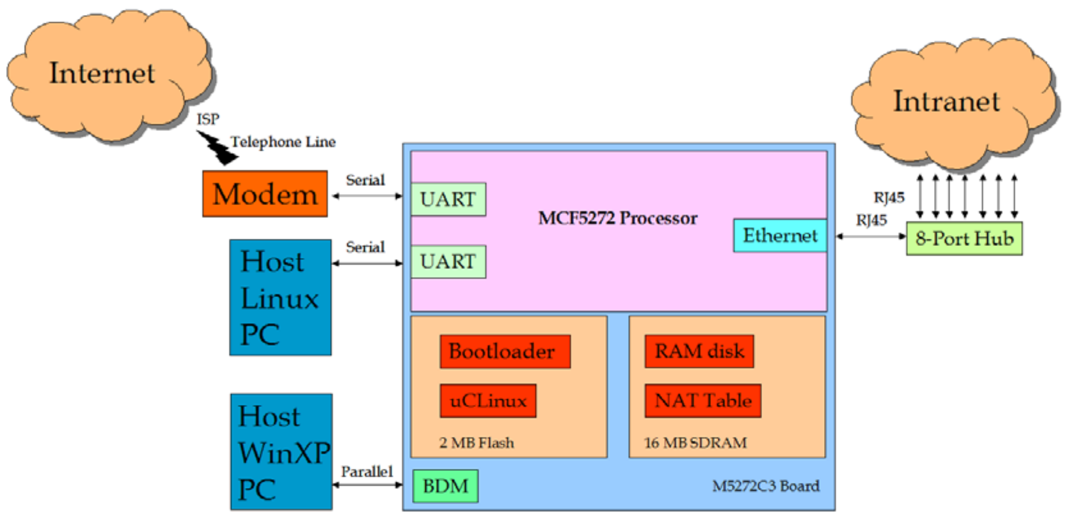


Figure 5. SoHo Router Test Setup

Figure 5 illustrates the initial prototype test set up. In the early experiments, operating system is not written into Flash but instead, it is downloaded directly into RAM and dBUG is used as the loader. UCLinux is compiled on the Linux PC and the WinXP PC is employed to program the Flash.

The figure also illustrates the basic internal architecture and the ports that have been used. Later, the system is modified to work in the standalone mode without the need for the Linux PC and the WinXP PC.

Instead of a serial modem, it is also possible to attach to the present system an ADSL or cable modem via the 8-port hub. In this configuration, two IP addresses correspond to the same Ethernet interface and the router can speak both to the intranet and the ADSL or cable modem. In such a system, it is possible to keep the serial modem as a backup.

Figure 6 illustrates this possible type of connection in a standalone operation. However, the current system does not include an ADSL or cable modem. The final test of the SoHo router is carried out using this configuration and its intended operation and functionality is validated. Its performance is tested subjectively and a satisfactory level of connectivity performance in proportion to a 56K serial modem is observed.

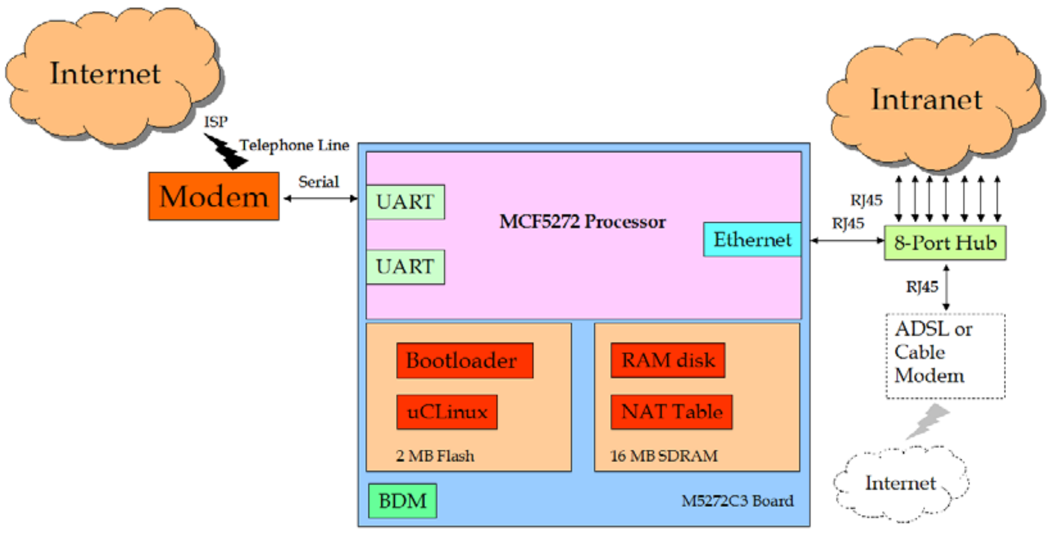
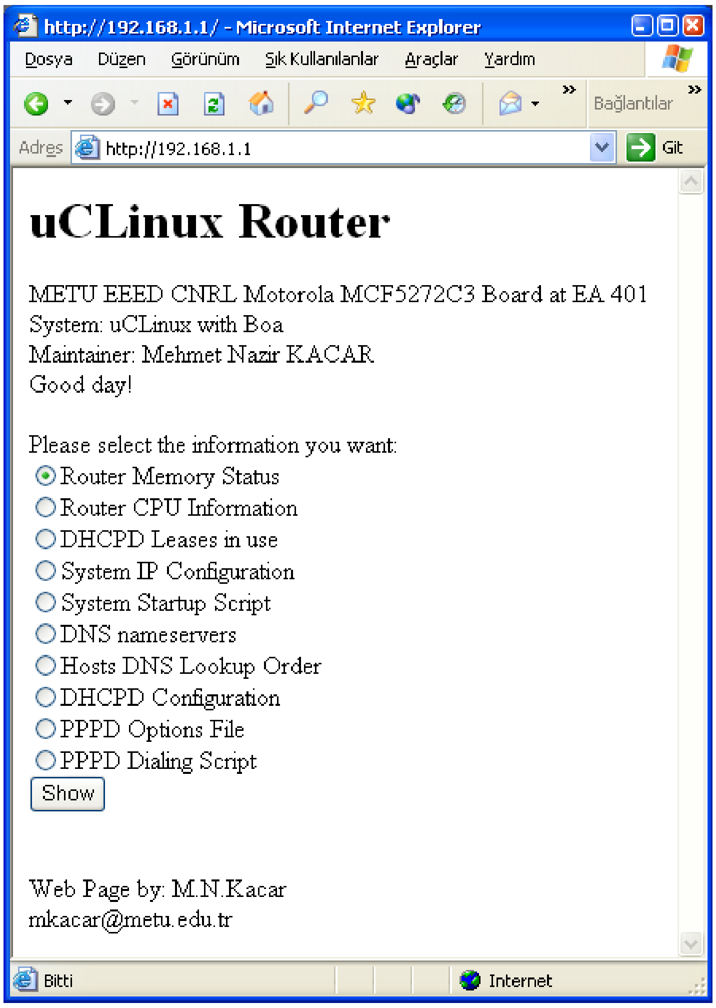


Figure 6. Standalone SoHo Router with Cable Modem

Within the present system, a web interface to view the basic configuration parameters of the router is also included. This interface, as can be seen in Figure 7, provides information such as Router Memory Status, Router CPU Information, DHCPD leases in use, System IP Configuration and some vital system configuration files. The interface is written using Dynamic HTML and CGI functionality is used to get the router parameters.

Kernel parameters such as memory, CPU or Ethernet status or important configuration files are obtained by a code written in C, which is given in the appendix. This program, according to the CGI parameters passed to it as environment variables, generates HTML code on the fly as a response and the client's web browser shows this response as if it was an ordinary web page.


http://192.168.1.1/ - Microsoft Internet Explorer
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
Bağlantılar
Adres http://192.168.1.1
Git
Bitti
Internet

uCLinux Router

METU EEED CNRL Motorola MCF5272C3 Board at EA 401
System: uCLinux with Boa
Maintainer: Mehmet Nazir KACAR
Good day!

Please select the information you want:
- [✓] Router Memory Status
- Router CPU Information
- DHCPD Leases in use
- System IP Configuration
- System Startup Script
- DNS nameservers
- Hosts DNS Lookup Order
- DHCPD Configuration
- PPPD Options File
- PPPD Dialing Script

Show

Web Page by: M.N.Kacar
mkacar@metu.edu.tr

Figure 7. Web Configuration Interface

This feature of the router is useful for system administrators to track network functionality and to see the router status.

CHAPTER 8

CONCLUSION

In recent years, general purpose operating systems are preferred to a greater extent to satisfy the increasing operating system requirements of modern embedded systems. Linux is one example of a highly popular general purpose open source operating system and it also various embedded versions.

In modern embedded systems, the applications are getting more and more complex and therefore more enhanced processor tasks are continuously being employed in these systems. Motorola Coldfire family of processors with many on-chip peripherals is among the flexible choices and heavily preffered by many embedded system designers and developers.

The present thesis work presents an embedded system implementation on uCLinux, one of the embedded versions of the Linux operating system, using a powerful and cheap Motorola Coldfire family processor MCF5272. The application choice is a SoHo router. SoHo routers are used to connect the local area network (LAN) devices in a small office or home office environment to wide area network

(WAN) or to the Internet and manage the interconnections between these LAN devices in an economical and user-friendly manner. The developed system in this thesis work provides basic router functionality in addition to the flexibility and reliability of the Linux system. It does not integrate the modem with the router in the same device but rather uses a simple and external one.

With its current capabilities, it is a basic SoHo router and includes the following features: It makes it possible for various LAN devices to share the internet connection; hosts a secure web server; provides firewall security (IP packet filter) in addition to web based gateway configuration and management and automatic network address translation (NAT).

The present implementation relies purely on the existing hardware resources of an available development board for MCF5272, namely the M5272C3, and the publicly available open source utilities of uCLinux. By using a bootloader program for uCLinux, the prototype device is made to function as a standalone system. For this standalone device, no other additional special purpose hardware or software components other than some configuration and make files and the modem have been used. This way, if the prototype is to be mass produced, it should not be very difficult to design a minimal cost hardware/software based around the core processor.

The tasks that were completed by the author within the scope of this thesis work include the following:

- While building the SoHo router prototype, a desktop Linux system is set up for compilation and experimentation which uses RedHat Linux 8.0. It also provides a download server for image downloading to MCF5272 evaluation board. This step includes installing the uClinux sources, the m68k-elf-toolchain, and the tftp server.

- uClinux operating system is configured by target platform, kernel and application configurations. In kernel configuration, the necessary drivers, protocols and kernel modules for the intended applications are chosen among hundreds of options. In application configuration, the applications that are vital for the SoHo router are selected. Since overall image size cannot exceed a certain level, optimization in the overall configuration process is essential.

- Configuration files necessary for the proper operation of kernel and applications are written. Standard compilation process can then be employed, following the above customization.

- A bootloader for uClinux operating system is configured and compiled in accordance with evaluation board's properties. The source code is slightly modified to keep the board firmware unchanged.

- A C program is written to serve as the web configuration interface for the SoHo router.

- The system operation is validated on a test setup.

The overall development process provides an embedded system implementation and configuration example using uCLinux.

uCLinux is currently a developing operating system and as a result it has many inadequacies. Originating from the commercially available Lineo's Embedix and SnapGear's distributions, we believe that the maintainers tend to support the commercial versions rather than the freely distributed one. Research and development efforts on uCLinux seem to be much less than the Linux kernel itself. This opinion may be supported by the fact that there seems to exist a handful of people currently working actively on porting and distribution.

In fact, uClinux should use the advantage of being a Linux clone. However, compiling and porting uCLinux is rather an advanced task and without being an experienced C++ programmer and especially being skilled in cross compiling, modification of the source code or compile arguments is challenging.

Being an embedded operating system, uCLinux mostly suffers from Rom File System (ROMFS) and Flash. Most programs do not like the idea of running on a ROMFS or being executed from Flash (XIP). Discussions on running the kernel from the ColdFire Board's Flash are still continuing. Porting the upgrades is not as simple as stated.

uCLinux lacks a fully featured documentation. Most of the time, Linux documentation does not apply and it may be difficult to even configure the bundled tools. It is stated in various related sources that programs can be run on uCLinux with slight changes but most of the time this is observed not to be true. The code must be revised from the very beginning up to the end, editing all the system calls, etc.

Advantages of using uCLinux are also present. One can have a basic working operating system in a matter of hours. Since developing a homebrew operating system can last for months, this is a still a great advantage.

There still exists a challenging future work for the developed system. The system is yet not fully equipped as a high end SoHo Router. Flash file system support is required to write the configuration to flash memory and keep it there. Completion of the web-based configuration depends on the existence of this support. Whenever this support is available, updating the router configuration from the web can be added easily. Dial-on-demand function is also necessary due to the fact that dial-up connections are billed according to connection time. VPN connectivity support and local print server support are among the other tasks that should be implemented for enhancements.

REFERENCES

1. Embedded Linux/Microcontroller Project Official Web Site, www.uclinux.org
2. RedHat Inc. Official Web Site, www.redhat.com
3. The Linux Documentation Project and HowTo's, www.tldp.org
4. M5272C3 User's Manual, Motorola
5. The Colilo Howto www.reasonability.net/uclinux/colilo/colilo-HOWTO.html
6. UNGERER, G., "Embedded Linux On The Motorola Coldfire Processor", Embedded Systems Conference Fall 2000, Class #561.
7. Embedded Systems Sourcing Web Site, www.microcontroller.com
8. CITRON, C., "Challenges in Application Availability for The Embedded Systems Market", Transitive Technologies White Paper, 2001 www.transitive.com/documents/Challenges_in_Application_Availability1.pdf
9. SEMACK, MYRON A., "Embedded Operating Systems: Linux and Others", www.lug.psu.edu/presentations/embedded_operating_systems.pdf
10. Motorola ColdFire Family Overview & Technology Roadmap, Motorola, 2002
11. Motorola MCF5272 Product Brief, Motorola, 2002
12. TANENBAUM, A., Computer Networks, Third Edition, Prentice-Hall International Inc., 1996.
13. FAQ on Marketing to SoHo, www.workingsolo.com/boco/faqboco.html
14. In-Stat/MDR Market Alert, "SoHo Routers Anything But Small in 2003", December 3, 2002, http://www.instat.com/newmk.asp?ID=447
15. HUGHES, PHIL, "Why Linux in the Embedded Market?", http://www.lynuxworks.com/products/whitepapers/whylinux.php3
16. Arcturus Networks Inc., "uClinux – Linux for Microcontrollers", Arcturus Networks Whitepaper, www.ucdimm.com/Docs/UCLINUXWP.pdf

APPENDIX

Makeiplist.c

```
/* makeiplist.c */

#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define DHCPD_IPLIST_FILE
        "/home/nazir/uClinux-
dist/vendors/Generic/romfs/etc/config/dhcpd.iplist"

/* local prototype */
int addAddress(char *ip);

int main() {
      printf("Make IP List Utility (for the Lineo DHCP server)\n");

      addAddress("192.168.1.10");
      addAddress("192.168.1.11");
      addAddress("192.168.1.12");
      addAddress("192.168.1.13");
      addAddress("192.168.1.14");
      addAddress("192.168.1.15");
      addAddress("192.168.1.16");

      return 0;
}

int addAddress(char *ip) {
      FILE *in;
      struct in_addr inp;

      printf("adding: %s\n", ip);
      inet_aton(ip, &inp);
      in = fopen(DHCPD_IPLIST_FILE, "a");
      if(in == NULL) return -1;
      fwrite(&inp.s_addr, sizeof(u_int32_t), 1, in);
      fclose(in);

      return 0;
}
```

ShowConfig.c

```
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <syslog.h>
#include <signal.h>
#include <errno.h>

/** Convert a two-char hex string into the char it represents. **/
char x2c(char *what) {
   register char digit;

   digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0]
- '0'));
   digit *= 16;
   digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 :
(what[1] - '0'));
   return(digit);
}

/** Reduce any %xx escape sequences to the characters they
represent. **/
void unescape_url(char *url) {
    register int i,j;

    for(i=0,j=0; url[j]; ++i,++j) {
        if((url[i] = url[j]) == '%') {
            url[i] = x2c(&url[j+1]) ;
            j+= 2 ;
        }
    }
    url[i] = '\0' ;
}


/** Read the CGI input and place all name/val pairs into list.
**/
/** Returns list containing name1, value1, name2, value2, ... ,
NULL  **/
char **getcgivars() {
    register int i ;
    char *request_method ;
    int content_length;
    char *cgiinput ;
    char **cgivars ;
    char **pairlist ;
    int paircount ;
    char *nvpair ;
    char *eqpos ;

    /** Depending on the request method, read all CGI input into
cgiinput. **/
    request_method= getenv("REQUEST_METHOD") ;

    if (!strcmp(request_method, "GET") || !strcmp(request_method,
"HEAD") ) {
        /* Some servers apparently don't provide QUERY_STRING if
it's empty, */
        /*   so avoid strdup()'ing a NULL pointer here.
*/
        char *qs ;
        qs= getenv("QUERY_STRING") ;
        cgiinput= strdup(qs  ? qs  : "") ;
    }
    else if (!strcmp(request_method, "POST")) {
        /* strcasecmp() is not supported in Windows-- use strcmpi()
instead */
        if ( strcasecmp(getenv("CONTENT_TYPE"), "application/x-www-
form-urlencoded")) {
          printf("Content-Type: text/plain\n\n") ;
            printf("getcgivars(): Unsupported Content-Type.\n") ;
            exit(1) ;
        }
        if ( !(content_length = atoi(getenv("CONTENT_LENGTH"))) ) {
          printf("Content-Type: text/plain\n\n") ;
            printf("getcgivars(): No Content-Length was sent with
the POST request.\n") ;
            exit(1) ;
        }
        if ( !(cgiinput= (char *) malloc(content_length+1)) ) {
          printf("Content-Type: text/plain\n\n") ;
            printf("getcgivars(): Couldn't malloc for cgiinput.\n")
;
            exit(1) ;
        }
        if (!fread(cgiinput, content_length, 1, stdin)) {
          printf("Content-Type: text/plain\n\n") ;
            printf("getcgivars(): Couldn't read CGI input from
STDIN.\n") ;
            exit(1) ;
        }
        cgiinput[content_length]='\0' ;
    }
    else {
      printf("Content-Type: text/plain\n\n") ;
        printf("getcgivars(): Unsupported REQUEST_METHOD.\n") ;
        exit(1) ;
    }

    /** Change all plusses back to spaces. **/
    for (i=0; cgiinput[i]; i++) if (cgiinput[i] == '+') cgiinput[i]
= ' ' ;
```

```
    /** First, split on "&" and ";" to extract the name-value pairs
into **/
    /**   pairlist.
**/
    pairlist= (char **) malloc(256*sizeof(char **)) ;
    paircount= 0 ;
    nvpair= strtok(cgiinput, "&;") ;
    while (nvpair) {
        pairlist[paircount++]= strdup(nvpair) ;
        if (!(paircount%256))
            pairlist= (char **)
realloc(pairlist,(paircount+256)*sizeof(char **)) ;
        nvpair= strtok(NULL, "&;") ;
    }
    pairlist[paircount]= 0 ;    /* terminate the list with NULL */

    /** Then, from the list of pairs, extract the names and values.
**/
    cgivars= (char **) malloc((paircount*2+1)*sizeof(char **)) ;
    for (i= 0; i<paircount; i++) {
        if (eqpos=strchr(pairlist[i], '=')) {
            *eqpos= '\0' ;
            unescape_url(cgivars[i*2+1]= strdup(eqpos+1)) ;
        } else {
            unescape_url(cgivars[i*2+1]= strdup("")) ;
        }
        unescape_url(cgivars[i*2]= strdup(pairlist[i])) ;
    }
    cgivars[paircount*2]= 0 ;   /* terminate the list with NULL */

    /** Free anything that needs to be freed. **/
    free(cgiinput) ;
    for (i=0; pairlist[i]; i++) free(pairlist[i]) ;
    free(pairlist) ;

    /** Return the list of name-value strings. **/
    return cgivars ;

}

/***************** end of the getcgivars() module
*******************/


int main() {
char **cgivars;
char *datafile;
FILE *f;
int i,ch;
cgivars = getcgivars();
datafile = cgivars[1];
if(*datafile == 'l') {
        FILE *in;
        u_int8_t mac_addr[16];
        u_int8_t ip_addr[4];
        size_t items; /* return value for fread */

        if ((in = fopen("/etc/dhcpd.leases", "r")) == NULL) {
                printf("Content-type: text/plain\n\n");
                printf("dhcpd.leases not found -- can't offer an IP
without a table to draw from!");
                return -1;
        }
        printf("Content-type: text/html\n");
        printf("\n");
        printf("\n");
        printf("<html><body><h1>IP Addresses in Use\n</h1><h3>");
        printf("Mac Address       IP-Address\n");
        printf("</h3>\n");
        while(1) {
        items = fread(&mac_addr, sizeof(mac_addr), 1, in);
        if(items < 1)
                break;
        items = fread(&ip_addr, sizeof(ip_addr), 1, in);
                if(items < 1)
                break;
        printf("%02X:%02X:%02X:%02X:%02X:%02X  ",
                mac_addr[0],mac_addr[1],mac_addr[2],mac_addr[3],
                mac_addr[4],mac_addr[5]);
        printf("  %u.%u.%u.%u",
ip_addr[0],ip_addr[1],ip_addr[2],ip_addr[3]);
        printf("<br>");
        }
        printf("</body></html>\n");
        fclose(in);
}
else if(*datafile == 's') {
                printf("Content-type: text/plain\n\n");
                fflush(stdout);
                system("/bin/ifconfig");
                fflush(stdout);
}
else {
f = fopen(datafile,"r");
if(f == NULL) {
  printf("%s%c%c\n",
  "Content-type:text/html;charset=iso-8859-1",13,10);
  printf("\n");
  printf("<TITLE>Failure</TITLE>\n");
  printf(datafile);
  printf("<P>Unable to open data file!"); }
else {
  printf("%s%c%c\n",
  "Content-type:text/plain;charset=iso-8859-1",13,10);
  printf("\n\n");
  while((ch=getc(f)) != EOF)
    putchar(ch);
  fclose(f); }
}

    /** Free anything that needs to be freed **/
    for (i=0; cgivars[i]; i++) free(cgivars[i]) ;
    free(cgivars) ;

    exit(0) ;
}
```