The Netslate II - A Handheld Web Browser Continued

by
Craig Newell
CraigN@cheque.uq.edu.au

Department of Electrical and Computer Engineering,
The University of Queensland.

Thesis submitted for the degree of
Bachelor of Engineering (Honours)
in the division of Computer Systems Engineering

October 1997

Lot 3 Nofz Lane,
via Esk, QLD. 4312
Tel. (07) 3365 4175

October 20, 1997

The Dean
School of Engineering
University of Queensland
St Lucia, Q 4072

Dear Professor Simmons,

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Computer Systems Engineering, I present the following thesis entitled “The Netslate II - A Handheld Web Browser Continued”. This work was performed under the supervision of Dr. Mark Schulz.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text, footnotes and citations, and has not been previously submitted for a degree at the University of Queensland nor any other institution.

Yours sincerely,

Craig Newell.

Very Late Breaking News

On the morning of this thesis being submitted, the TCP/IP updates (See Section 3.2.4.5) to Inferno™ 1.0 have been received via email. Many thanks to Erik Van Hensbergen[53] for the sending the patches.

News Flash

The field of network computing, of which the Netslate II is part, has taken off tremendously in the second half of 1997. This can be seen from the following product and news releases from the World Wide Web in the fortnight leading up to the completion of this document on October 20, 1997.

October 17, 1997
TV is future of on-line shopping
A prediction that the set-top box will be the future
`http://www.news.com/News/Item/0,4,15384,00.html`

October 16, 1997
A Pentium in your toaster?
Pentium processors for devices such as the Netslate.
`http://www.zdnet.com/zdnn/content/zdnn/1016/161668.html`

October 15, 1997
Symbol to "ruggedize" the PalmPilot
An attempt announced to make the PalmPilot PDA more sturdy.
`http://www.zdnet.com/zdnn/content/pcwo/1015/161617.html`

October 13, 1997
Java™: Coming Soon to Phones, TV Set-tops, Hand-held Devices . . .
A updated release of Java™ to be used in many network computers.
`http://www.javasoft.com/features/1997/oct/Personal.Embedded.html`

October 13, 1997
Acorn Unveils Consumer NC
A new design for a set-top box that runs Java™ applications.
`http://www.zdnet.com/zdnn/content/inwo/1007/151463.html`

October 10, 1997
Scoop du jour: small, cool handheld computers
Announcements for PDA's with web access from Sharp, and NEC
`http://www.zdnet.com/zdnn/content/pcwo/1010/151800.html`

October 6, 1997
AT&T aims to put smarts in the palm of your hand
Announcing mobile phones with email and web access.
`http://www.zdnet.com/zdnn/content/zdnn/1006/zdnn0006.html`

Thanks

The completion of the Netslate II has been influenced by the help of many people. The most important to this success are acknowledged below:

Dr. Mark Schulz for acting as my thesis supervisor and putting up with the rushed requests to get the various contracts signed as soon as possible.

Ben Williamson of Teknema designed and built the original Netslate I hardware and was an invaluable source of knowledge as to the limitations of the ARM7500 processor and the ADT.

Bernard Larkin of Digital Images for providing me with a copy of the latest IBM Developers Connection CDs containing the OS/2 Development Toolkit allowing for the successful port of Inferno™ to OS/2.

Graham Galligar of Digital Images for insights into the workings of the Inferno™ development team saving me much bashing against the wrong people at Lucent Technologies.

Dr. Ian Cameron has kindly provided me with computing facilities that have allowed the completion of this without the constant worries of the Electrical Department facilities.

Ben Sigalpa for allowing me to use the electronic workshop facilities at the Dept. of Chemical Engineering in which the majority of hardware enhancements to the Netslate were complete.

EDMI for the manufacture and assembly of the FLASH ROM programmer and the second-hand FLASH ROMs used in the Netslate II.

Andrew Mansbridge for general help with design and construction ideas and for assistance in building of the FLASH ROM programmer.

Abstract

The Netslate is a vision of a hand-held web browser, a small portable computer packaged as a digital slate which runs a stand-alone web browser with a wireless link to the network. This will provide a simple, cheap and convenient interface to the World Wide Web as a source of information and entertainment.

In this the second Netslate thesis, the hardware of the original Netslate[48] has been enhanced and the Inferno™ operating system has been ported to allow the Netslate to more completely forfill the vision of a handheld web browser.

The Netslate hardware was extended by the completion of the color LCD panel, the addition of 1 Mbyte of FLASH ROM and a custom IrDA® compatible infrared port for use as a wireless network connection.

The Inferno™ network operating system and demonstration applications including a web browser have been ported to run stand-alone on the Netslate. This was a two part process, with a preliminary port to OS/2 to gain experience with Inferno™ and then a final port to the Netslate. The OS/2 port is completely original and the Netslate port was also done from scratch.

The current state of network computing, the field of computing in which the Netslate is part, is examined in brief with relation to what has been completed with this Netslate II. This is done covering both the current hardware products and the available operating system software.

Finally, the conclusion is reached that the current operational Netslate II has met the vision of the Netslate within the limitations of the University undergraduate thesis environment.

All brands, product names and company names mentioned in this document are trademarks or registered trademarks of their respective owners. An attempt has been made to acknowledge these trademarks where possible but no responsibility is taken for those missed.

Contents

List of Figures

List of Tables

Chapter 1

The Netslate

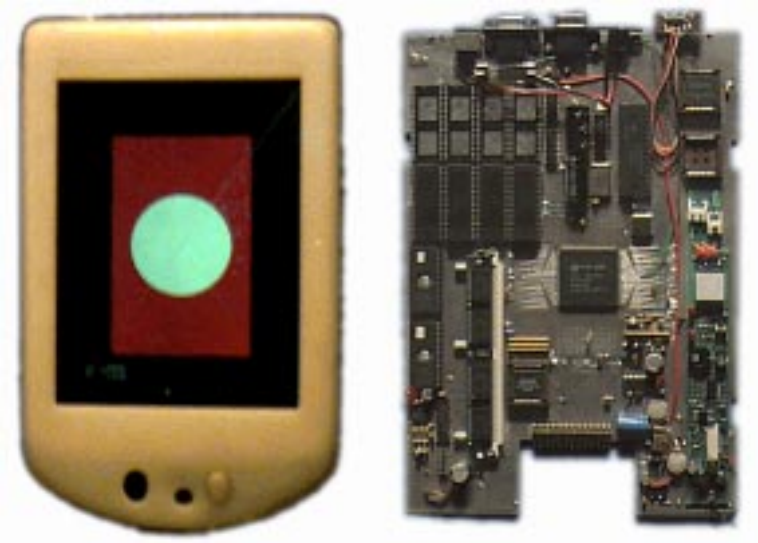

Figure 1.1: The Netslate: Front and Back

The Netslate is a vision of a hand-held web browser, a small portable computer packaged as a digital slate which runs a stand-alone web browser with a wireless link to the network. This will provide a simple, cheap and convenient interface to the World Wide Web as a source of information and entertainment. The general requirements for the design and implementation of this Netslate vision are:

Function
The basic function of the Netslate is to run a stand-alone web browser to surf the web. In addition it should support helper applications and downloadable applets.

User Interface
The Netslate is to be a multimedia device with a color display capable of displaying text, graphics, animations and playing sounds. Surfing the web only requires minimal user input which could be provided by a touch screen.

Physical Characteristics
The Netslate should be small, lightweight, low power and be expandable.

1.1 The Netslate Project

The structure the work done towards the Netslate project is shown below in Figure 1.2. This shows the two completed stages of the Netslate project as well as the future work left in the Netslate project.

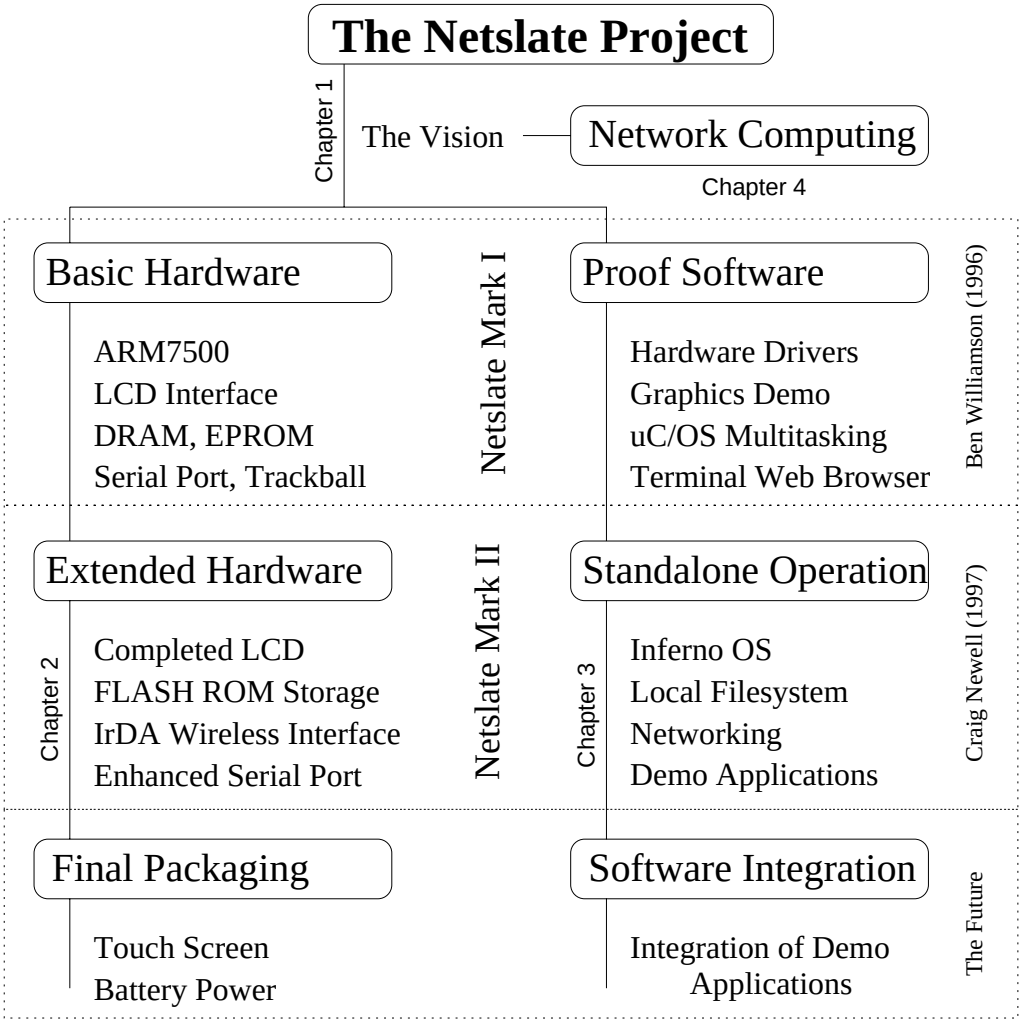

Figure 1.2: Netslate Project Structure

These stages completed show a progression towards meeting the requirements of the Netslate vision listed below:

- Stand-alone Web Browser
- Handheld Computer
- Color Display
- Sound Output
- User Input
- Future Expansion

The follow subsections give an overview of the current status of the Netslate project. The work carried out to reach this point is described in detail in Chapters 2 and 3. In addition, the field of network computing, in which the Netslate project is part of, is examined in Chapter 4. This is done with a brief overview of the current products in both hardware and software.

1.1.1 Netslate Mark I

Ben Williamson started the Netslate project in 1996 for his honours undergraduate thesis[48]. In this thesis, the basic Netslate hardware was built and tested with a number of hardware specific drivers written. The final demonstration included the display of a Tk/TCL[39] Web Browser running on a Linux box, a simple graphics demo and a rough port of the μC/OS.

A brief evaluation of Ben's work towards the Netslate vision can been seen by examining the requirements of the Netslate vision and his progress towards meeting them. The final product of Ben's efforts is referred to for the rest of the document as the Netslate I.

Stand-alone Web Browser With the demonstration of a Tk/TCL based web browser on the Netslate, a web browser was achieved but not stand-alone. The actual web browser was running on a Linux box with just the display and user input occurring on the Netslate.

Handheld Computer The Netslate I was well on the way to becoming a handheld computer. The majority of the components are mounted inside a slate like molded plastic case. However, the Netslate I is still tied down by the need for external power and a serial link.

Color Display The demonstration of the Netslate I was done using an external color video monitor. This was because an interface to the color LCD panel had been designed but the necessary support power supplies were missing. Without a backlight or contrast power supplies nothing can be seen on the LCD panel although the image may be present.

Sound Output The Netslate's processor, the ARM7500 has an internal 8 bit audio interface. This had been connected to but was not operational at demonstration due to a number of problems.

User Input The Netslate I has a working PS/2 style mouse input connecter to a small handheld trackball with two buttons. This provides a useable user interface and provision was made to mount this trackball into the Netslate's case. This is because a touch screen was seen not to be practical for the Netslate.

Future Expansion The Netslate I board was limited in expansion with only a very small amount of prototyping space available on the circuit board and a speed limited serial port.

1.1.2 Netslate Mark II

In 1997, the Netslate I has been extended to more fully meet the Netslate vision to become the Netslate II. These enhancements are listed below according to the requirements of the Netslate vision that they meet.

Stand-alone Web Browser The vision for a stand-alone web browser progressed greatly with the porting of the Inferno™ operating system to run stand-alone on the Netslate II. This required the addition of extra hardware such as the 1 Mbyte of FLASH ROM storage and a faster serial port.

Handheld Computer The Netslate II is now more of a handheld computer with the addition of a IrDA® wireless network link.

Color Display The color LCD panel has been completed with the construction of the necessary power supplies and more complete mountings.

Sound Output An attempt was made at debugging the Netslate's ARM7500 internal 8 bit audio interface but this had to be abandoned due to problems discovered internally in the ARM7500 chip.

User Input When the LCD panel was completed and mounted into the case as designed, it was found that there was insufficient room left to mount the original trackball. A search was made for a more suitable trackball but none could be found within price restrictions.

Future Expansion The majority of the expansion done to the Netslate I board was done at great effort in physical construction leaving very little board space left at all. Although, the serial port has been upgraded and now provides a method of expansion for devices such as a bar code wand, GPS receiver or other serial device.

This work in taking the Netslate I to the Netslate II is described in Chapters 2 and 3 for the hardware and software sections respectively. These chapters have been formatted in a Requirement/Problem-Alternatives- Solution- Evaluation layout to provide a logical documentation as to the development of the Netslate II.

1.1.3 Future Netslate

The Netslate II is currently very close to meeting all of the requirements of the Netslate vision. The remaining sections to complete are listed below with respect to the requirement of the Netslate vision:

Stand-alone Web Browser The Inferno™ operating system and web browser is complete but lacking in support of the latest web features (HTML 3.2). These are available in the latest versions of Inferno™ but getting hold of these updates has been a very slow process not likely to succeed before the demonstration of the Netslate II.

Handheld Computer The Netslate is almost a completely handheld computer but still requires an external power source. This could be eliminated with an improved voltage regulator and rechargeable batteries. However, this would never be very successful due to the large power requirements of the color LCD panel. A more modern and hence much more expensive LCD panel could be used but it is unlikely that the cause would warrant such a large cost.

Audio Output The on-chip audio circuitry of the Netslate's ARM7500 processor is faulty by design and thus an external audio interface is required. This can be implemented but the required components are not easily available.

User Input The current trackball is not mounted in the case due to size problems. The best solution to this would be a touch screen but this is not practical with the current LCD panel and limit budget.

As most of these remaining tasks are not resolvable within the restrictions of the Netslate project, the future must be examined in light of the current progress in the network computing field. This future is examined in Chapter 5.

Chapter 2

Extending Hardware

Contents

This chapter is a documentation of the extensions made to the Netslate I hardware to more fully meet the requirements of the Netslate vision as described in Chapter 1. These requirements and actions taken are listed below:

Stand-alone Web Browser To support the Inferno™ Operating System and a locally run web browser, an amount of non-volatile storage was required (Section 2.1).

Handheld Computer To better support the Netslate as a handheld wireless computer, a wireless network link was designed and built (Section 2.2).

Color Display The required power supplies for the LCD display were completed and the LCD panel mounted properly within the Netslate's case (Section 2.3).

Audio Output An attempt was made to debug the ARM7500's internal 8 bit audio interface, only to discover that it is broken by design (Section 2.4).

User Input The Netslate's external trackball had to be mounted in the case once the LCD had been completed (Section 2.5).

Future Expansion The wired RS232 serial port was upgraded to allow faster baud rates for use in software development and for the attachment of external peripherals (Section 2.6).

2.1 Non-volatile Storage

Problem:

The Netslate I had 128 kbytes of ROM storage as two 27C512 64 kbyte EPROMS. This was more than enough storage for the small concept demonstration applications that were

demonstrated but even the smallest operating system and web browser requires at least 512 kbytes of storage.

Alternatives:

There are three major types of devices that are used for small capacity non-volatile storage. These are:

EPROM

Electrically Programmable Read Only Memory is a common, reasonably cheap source of non-volatile storage. However, it is very slow to develop with due to the long (15 minute) erase procedure that has to be done out of circuit.

EEPROM

Electrically Erasable Programmable Read Only Memory is neat in circuit programmable non-volatile storage. However, it is expensive and not available in large sizes.

FLASH

FLASH ROM is the most recent form of non-volatile storage and it is currently the most popular replacing EPROM in most designs. It is available in large capacities and is in-circuit programmable.

Solution:

Thanks to EDMI, a good supply of second-hand AMD29F010 [4] 128 kbyte FLASH ROMs for the Netslate II was available. Hence, sockets for 8 chips (as many and there is board space on Netslate) was added to the Netslate and an external programmer designed and built[8] to allow the chips to be programmed out of circuit for development.

The 8 chips are interfaced to the Netslate's ARM7500 processor in two banks of 32 bit wide ROM in the ROM0 and ROM1 regions. The read and write signals are decoded directly from the address as the ARM7500 Manual mentions that the ROM chip select is not activated for writes.

Evaluation:

The FLASH ROM storage is great compared to EPROM as the fast erase and programming is very handy for software development. The AMD29F010 has identical pinouts to the 512 kbyte AMD29F040 allowing for plug and play upgradability if necessary.

Unfortunately, the circuit programming of the FLASH ROMs in the Netslate is not possible even with the correct interfacing due to a limitation in the ARM7500 silicon (verified by [51]). The limitation is that external bus cycles are not generated for writes to the ROM address space. The FLASH ROMs could be connect to other portions of the address space, but then it would not be possible to boot or execute code directly from the ROMs. This limitation has been removed in the latest version of the ARM7500 chip, the ARM7500FE.

2.2 Wireless Network Link

Requirement:

To meet the requirement of a portable handheld device, the Netslate must be completely wireless. The two wired connections to the Netslate I, are a serial link (the network) and an external power cable. Hence, for the Netslate II, a wireless network link is required to provide access to the World Wide Web and thus a gateway to the Internet from the wireless link is required.

For the purposes of the Netslate vision and its limitations as an undergraduate thesis project, the wireless link does not need to be useable over any great distance from the network access point.

Alternatives:

There are four major technologies that could be used to meet the requirement for the Netslate II's wireless network link. These are:

Wireless LAN

The current spread spectrum wireless Local Area Network Cards provide a fast (1 Mbit/s+) wireless network over a reasonably large area (up to a few hundred meters). They are available as ISA and PCMCIA cards. The PCMCIA card versions would be great for the Netslate as far as performance, but the PCMCIA interface is complex and little documentation is available. At the beginning of 1997, the price was quite expensive but has become almost within reach by the end of 1997.

Radio/Cellular Modem

Radio modems over dedicated channels or cellular modems over the standard cellular phone frequencies could provide a long range wireless network link for the Netslate. However, these are in general slow (about 9600 bit/s) and are very power hungry.

Diffuse Infrared

Diffuse Infrared can provide medium to very high speed (100 kbit/s to 10+ Mbit/s) wireless networking over the range of a single room. In general, this is new technology that would require a custom solution for both Netslate and a network access point.

Direct Infrared

The IrDA® standard [21] for direct Infrared wireless network links is a simple specification for slow to fast (1200 bit/s to 4 Mbit/s) links over a distance of 1-3 meters. The IrDA® hardware is cheap, power efficient and common in most laptop computers. Hence it is a simple matter to configure a laptop computer with IrDA® port and an Ethernet card as a network access point.

Solution:

An IrDA® interface was chosen for the Netslate II due to the minimal cost and availability of parts and a suitable laptop for use as a network access point. The small range of IrDA® is no great problem as the intended demonstration of the Netslate II will be in a confined space.

The implementation of IrDA® on the Netslate II involved three major components: an Infrared Transceiver that converts between an IR signal and a TTL signal, a IrDA® codec that implements the IrDA® coding and a standard serial UART. The IR transceiver used is a Novalog MiniSIR™ [19] obtained as a free sample as was the National Semiconductors PC16552 UART[46]. It was not possible to obtain an IrDA® codec so a custom design was created.

Appendix B contains a detailed description of the IrDA® specification and its implementation with respect to the Netslate II.

Evaluation:

The IrDA® wireless port works very well over a range of about 2.5 meters at the maximum speed of 115.2 kbit/s. However, due to a software “bug” in the Inferno™ operating system, acceptable performance of the wireless link could not be verified at time of writing. Software updates are hoped to be available to allow operation on the demonstration day.

2.3 LCD Display Completion

For the Netslate I, a SANYO color passive matrix LCD panel had been obtained and a interface from the ARM7500 designed and implemented. Due to the lack of the required power supplies, this interface had only been quickly tested with some simple video drivers and found to be operational.

This left three major things to complete before the color display could be used. These are the LCD ribbon connector, contrast power supply and the backlight power supply. The implementation of each of these for the Netslate II are discussed in the following sections.

2.3.1 Ribbon Connector

Problem:

The SANYO LCD panel has a 30 way ribbon connector with 1 mm spacing pin spacing. Cable with a 1 mm spacing is not obtainable (in less than 1000m quantities!).

Alternatives:

The alternatives are either to solder individual wires to each pin of the LCD connector, as done by Ben's, or to replace the connector on the LCD panel. This can be done relatively easily as the LCD has a small "hack" adapter PCB to reverse the pinouts on the connector.

Solution:

As the individual wire solution was not reliable due to individual wires "falling" off the very small pads of the LCD panel, the adapter PCB was replaced to convert 1mm spacing to .1 inch 34 way IDC connector. A right angle IDC connector was added to the Netslate PCB to connect to the LCD panel with a short length of IDC ribbon cable.

Evaluation:

The final connectors are neat and reliable although slightly bulky. The right angle connectors allow for a logic analyzer to be connected easily when LCD panel is connected for verification of timing waveforms.

2.3.2 Contrast Power Supply

Problem:

The passive matrix LCD panel requires a small amount, about 10 mA, of 32 Vdc and adjustable contrast voltage, 26-30 Vdc, to produce the image.

Alternatives:

A 32 Vdc supply can either be constructed from a "high voltage" supply to the Netslate board and a conventional linear regulator (eg. LM317) or it can be built from a small switch-mode up-converter. This up-converter can be supplied from the current 5V supply on the Netslate board.

Solution:

A small switch-mode power supply has the advantage of portability as there is no need for multiple supply voltages to the Netslate and the components for one built around the MAX641 controller chip that had been obtained by Ben[51]. Thus as the circuit required is very simple, the supply was prototyped on a breadboard to determine correct operation before being constructed using "dead bug" techniques on the Netslate board.

Evaluation:

The contrast supply works well and produces acceptable image on the LCD display. The output is, however, slightly noisy (about 1.4 Vpp noise) so a simple filter was added using a couple of large capacitors and an inductor. This reduced the noise to about 0.3 Vpp but with no discernible improvement in quality of image.

2.3.3 Backlight Power Supply

Problem:

The LCD panel contains two cold cathode tubes to provide the backlight. Each of these tubes are about 6 watts each and require a 1200 Vac strike and 700 Vac operating voltage supply.

Alternatives:

The only practical method of generating these voltages is with a switch mode inverter. This can be built from parts or bought as a module. The parts for a supply and the module cost about the same and are both not easily obtainable.

Solution:

As the necessary components for 6 watt maximum supply were available, this was prototyped on breadboard. However, it would only partially light up one of the tubes. Due to the small cost difference between obtained more components and obtaining the supply as a module, a module was obtained after much chasing of the supplier.

The module arrived with absolutely no documentation, not even as to which of the two loose wires was negative and positive supply. After the circuit had been traced, the module provides the necessary back light voltages for both tubes from a from 11.5 to 14 Vdc supply (voltage determines the brightness). The tracing of the circuit also revealed a -15 Vdc contrast supply which suggests that the module is designed for color active matrix LCD which use -15 Vdc for the contrast supply.

Evaluation:

The module backlight supply works well but it also produces a high voltage on LCD case due to electrostatic coupling (about 700 Vdc). The case of the LCD is electrically isolated but even so if the case is grounded, the backlights go out. No solution to this has been found even after consultation with the various electronics workshop staff. The input voltage requirement of the LCD panel of about 12.5 Vdc is currently the only supply to the Netslate with necessary the 5 Vdc for the ARM7500 being provided by an existing linear regulator.

2.4 ARM7500 Audio

Requirement:

For the full multimedia capabilities of the Netslate vision to be realized, a reasonable quality audio output is required.

Alternatives:

The ARM7500 single chip computer used in the Netslate provides two different sound interfaces. These are a high quality 16 bit serial stereo sound interface which requires external 16 bit DAC (not easily available), and a simple 8 bit stereo sound interface that requires only external amplifier.

Solution:

The Netslate I had implemented the external amplifier necessary for the 8 bit sound interface. However, this was not working and it had been left for further work.

After a detailed reading of the ARM7500 datasheet[1], a simple analog design fault in the interfacing of the external amplifier. This was simply corrected but nothing other than white noise was produced.

This problem was traced down to a timing fault in the was the ARM7500 chip multiplexed the sound output to produce stereo outputs. The exact cause of this problem has not been determined but it is known to be a problem with the particular ARM7500 design used in the Netslate (verified by [51]).

Evaluation:

As an audio interface is part of the Netslate vision, a working audio output would have been nice but it was decided that it was not critical for the Netslate II. It has thus been left for future implementation of 16 bit audio interface which has been verify to be operational on the ARM7500 in the Netslate.

2.5 Mounting a Trackball

Requirement:

To meet the requirement of a handheld computer, the user input interface should be mounted molded plastic case of the Netslate.

Problem:

The user input for the Netslate I was a small trackball. Space had been made for this in the molded plastic case but this did not take into account the final connector and mounting of the LCD panel. The result of this is that the trackball would not fit into the case.

Solution:

A large number of computer mice and trackballs were examined for their suitability to mount into the Netslate case. The conclusion drawn was that mice are not easily modified into trackballs and that the smaller trackballs on the market are very expensive (e.g. AUD$140 for a small laptop trackball). The best solution was a "pad" style mouse but due to its wireless nature, the price tag was too high.

The ultimate solution to this problem is a touch screen covering for the LCD panel but once again price is the limiting point. Hence this has been left for future work.

2.6 Enhanced Serial Port

Problem:

The standard serial port on the Netslate I was limited to 38.4 kbaud due to the limited baud rate divisors of the TI16C550 and 8 MHz clock used. This problem is very common due to the "strange" baud rates defined by the IBM PC compatible standard.

Solution:

There is only one solution to this problem that is to install a 1.8432 MHz clock like that installed in most IBM compatible computers. This was done using a standard crystal oscillator module.

Evaluation:

Using the 1.8432 MHz clock, the serial port can be operated at 115.2 kbaud (fast as a PC can go) reliably assuming that the software drivers at both ends are up to it.

Chapter 3

Stand-alone Software

Contents

This chapter contains documentation of the Netslate II's software that attempts to fully meet the requirements of the Netslate vision as described in Chapter 1. This software is mainly directed towards achieving a stand-alone web browser.

Stand-alone Web Browser To create the stand-alone web browser, an operating system and web browser was ported to the Netslate II.

3.1 Possible Software

Requirements:

The operation of a full featured web browser requires a multitasking operating system with networking and a graphical interface. Given the physical limitations of size and the second requirement for the support for helper applications and downloaded applications.

Alternatives:

At the end of 1996, there was very little in the way of small operating systems with graphical user interfaces and web browsers. However, the beta release source code for a couple suitable operating systems complete with web browsers was advertised as being available for academic use. Both of these operating systems also met the requirements of helper applications and downloadable applications via their respective processor independent execution schemes. These operating systems were Inferno™ [20] and JavaOS™ [27].

Inferno™

The Inferno™ operating system from Lucent Technologies is a small operating system from the lineage of unix and plan9[40]. It is being targeted at small network based computer applications and provides a very integrated method of accessing network resources. To provide this integration across platforms as well as just the network, a virtual machine architecture called DIS and a corresponding modular language Limbo is provided.

There was two sets of demonstration applications shipped with the beta releases that were targeted at the TV set-top box and the desktop network computer markets. These are discussed as they relate to the Netslate vision in Section 3.3.

JavaOS™

The JavaOS™ operating system from Sun Microsystems is almost completely platform independent operating system built on the Java™ virtual machine and object orientated language specifications.

From the very beginning the Java™ vision is tied closely with networking and the Internet. This is shown by the fact the the first major application written in Java™ was the HotJava™ web browser that was being pushed as the first fully extendable web browser by the use of the platform independent Java™ language and small applications known as applets.

Decision:

The decision for which operating system was to be used in the Netslate II was pretty much made by default over the requirement for availability of the source code. The Inferno™ operating system was also chosen for it's smaller size (quoted at the time as 0.5 Mb over 1 Mb minimum ROM space) and the "neater" design of the Inferno™ system.

Another definite advantage of Inferno™ is that as well as being a complete stand-alone operating system, it can also run in a hosted environment on top of an existing operating system. This hosted mode of operation is aimed at application development and server operation with the reliability of a existing operating system.

Evaluation:

The decision to use Inferno™ appears to have been a very good one. From contacts with a number of people who have hands on experience with JavaOS™ and from the general computer media, JavaOS™ is very large (6 Mb) and still unstable even in the first general release.

3.2 Inferno

The process of getting Inferno™ to run on the Netslate II was completed in two major stages.

The first was a port of the hosted kernel to the OS/2 operating system which had not been done before (or since). This port was used to get experience with the Inferno™ source code in an environment where good debugging tools exist.

The second port was then of the stand-alone native kernel to the Netslate hardware. Inferno had previously been ported to the ARM7500 as used in the Netslate but the source code of this port was not obtained.

However, firstly a bit of background to the obtaining of the Inferno™ source code is necessary.

3.2.1 The Source

The source code for Inferno™ 1.0 build 1.0 has a fair story behind it. The next two subsections cover the background of getting the source code and what was actually received.

3.2.1.1 Getting the Code

In December 1996, the first emails asking about the possibility of obtaining a source code licence was sent. A reply saying that we could apply for a source code licence for the beta source to Inferno was received and a response asking for a copy of the licence agreement was returned.

Nothing much happened over the holiday period and after a couple follow up email messages, in late January a fax of the licence conditions was received. This was examined and found to be suitable and a response was emailed.

Once again there was a pause and after a number of further non-answered email messages, a phone call to Lucent Technologies in the US was made at 1am in the morning local time in Brisbane. This resulted in a fax of the license agreement. This took a couple of weeks and was returned in late February.

The characteristic pause with one way email messages occurred again. This was broken once again with a phone call to the US resulted in action with a CDROM arriving in the mail two weeks later. The phone call revealed that the release of Inferno 1.1 was to be made later in the week that the phone call was made and an agreement to send the source for release 1.0 was to be sent immediately and the source to 1.1 as soon as it was finalized. Only ever the 1.0 source has arrived.

And hence the source code to Inferno™ 1.0 build 1 was obtained containing the source for the hosted versions of Inferno™ and for the native versions for the IBM PC and the AMD29K processors (Note that the source for the native kernel on the PC and the ARM7500 was specified on the licence agreement).

This source code has been obtained under a reasonably strict non-disclosure agreement and hence the follow discussions of Inferno™ are more than a bit vague in places.

3.2.1.2 The Versions

There have been (to be) three major releases of Inferno™ since the first Inferno™ beta 4 was decided upon as the software for the Netslate II.

Release 1.0
The first general release of Inferno™ was 1.0. The 1.0 release was complete and reasonably stable with only a small number of major known bugs mainly in the native kernel networking support.

The source that I finally got is Inferno™ 1.0 build 1 including the hosted kernel and native kernel ports to the PC and the AMD29K. Although a port to the ARM7500 had been done, this code was not obtained. The source came on a single CDROM with absolutely no documentation (as none had been written at the time!). The total extent as to the documentation was a short README file from the binary distribution of Inferno™ and the very sparsely commented source code.

Release 1.1
Inferno™ Release 1.1 was an update to the 1.0 release source code with a rewrite of a large portion of the Tk based graphical user interface and ports of the native kernel to more different platforms. This release was also to include some documentation on the internals of Inferno™.

A large number of email messages has been sent since obtaining the release 1.0 source code to a number of people within Lucent Technologies regarding obtaining the 1.1 release and in particular the ARM7500 port and various bug fixes contained within. Although many promises of action were received, it was not until the middle of October that an updated license agreement for the 1.1 release has been received. However, this is too late for the source to be received before this document was written and the demonstration of the Netslate II preformed. A special thanks should go to [52] who after about a month of daily email messages finally managed to fax the contract update to sign.

Release 2.0
Inferno™ release 2.0 is a major upgrade to the 1.x releases with the inclusion of PersonalJava™ compatibility. At the time of writing the official release date is the middle of November 1997.

3.2.2 Getting It to Compile

Once the source code had been obtained, it was necessary to choose the development tools and to compile the large majority of the code which is completely portable.

Which Tools?

Question:
Which tools, assemblers, C compilers and linkers, should be used in the porting of Inferno™ to OS/2 and to the Netslate?

Alternatives:
The available tools consisted of three main classes:

Commercial
For the Inferno™ port to OS/2, both the Borland 2.0 development kit and the Watcom 10.6 compiler set was available. The VLSI JumpStart™ ARM development tool kit (ADT) was also available for the ARM7500 processor in the Netslate II.

Inferno™ Custom
The Inferno™ source code comes complete with a set of assemblers, C compilers and linkers for both the IBM compatible PC and the ARM7500.

GNU
The freely available GNU compilers are available for both OS/2 and the ARM7500. The ARM7500 version was used with the software for the Netslate I. These compilers are reliable and well supported but produce non-optimal code.

Solution:
Due to personal experience with buggy compilers and the requirement for small size due to the limited non-volatile storage on the Netslate, the commercial compilers were selected.

The Watcom 10.6 compilers for OS/2 was chosen due to their better code generation and debugging utilities and the ADT was selected for the Netslate.

A final consideration in this choice was the very good quality of documentation of these commercial compilers especially over the custom Inferno™ compilers which came with absolutely no documentation (not even to the fact that they existed in a very deep sub-directory of the source tree).

Evaluation:

Although the use of compilers other than the custom Inferno™ tools are not supported by Lucent, the good documentation and debugging utilities of the Watcom and ARM ADT tools well and truly made up for the problems encountered in porting the source code.

How to make it?

Problem:

The Inferno™ source code has two sets of script files to control the building process, the `mkfile`'s for use with the "mk" utility from the plan9 operating system (source provided) and `makefile`'s for use with the standard "make" utility available as wmake (Watcom) or armmake (ADT). Which should be used?

Solution:

At the time, there was no documentation for the "mk" utility and the process of porting "mk" to OS/2 appeared a reasonably major task, the option of the "make" utilities was chosen. The Watcom wmake utility was used as versions are available to OS/2 and DOS such that one set of `makefile`'s could be used for both the OS/2 and Netslate ports of Inferno without changes.

Hence, the many `makefiles` used to build Inferno™ were modified and rewritten in some cases to be used with the Watcom and ADT tool sets.

Evaluation:

Overall, the use of the `makefiles` has proved to be the best solution as the `mkfiles` would have also required a reasonable amount of modification to work with the chosen Watcom and ADT tool sets.

Automatically Generated Source

Problem:

The Inferno™ source code contains a number of generated source files: header files from Limbo module definitions and native kernel configuration specific files (netslate.c, devtab.h, error.h). These necessary files were not shipped with the distribution.

Alternatives:

To generate these files, either the generating scripts (which use sed and awk[2]) and programs could be ported to OS/2 (the development environment used) or the files would have to be generated using the supplied plan9/unix scripts.

Solution:

Due to the predicted need to regenerate the native kernel configuration files quite often, it was decided to port the scripts and programs to OS/2. A limited port limbo compiler was done just enough to generate the necessary header files. OS/2 versions of GNU awk and sed where obtained and the plan9 "rc" shell scripts where converted to an OS/2 "tcsh" shell. In doing so, the scripts where enhanced to clearly mark the generated files as automatically generated as the previous versions did not leading to confusion in working out where the files came from.

Evaluation:

Once generated, it proved simpler and quicker just to edit the generated files by hand. Although, if you had to support many different ports from the same source tree then the generated files are a necessity (or you would have to rewrite the code to use `#ifdef`'s like the Linux kernel).

C != C

Problem:

The Inferno™ source code contains a number of extensions to the ANSI C syntax[29] that are not supported by the Watcom and ADT C compilers selected.

Solution:

There was not much option other than to make lots of very small changes to a large amount of of the Inferno™ source code. These changes were made in the following areas:

enums

The definition of enumerations (enums) included a comma after the last element. This caused an error with the ADT compiler and had to be removed.

structures

In quite a few places, structures are assigned to a constant. This is not compatible with both the Watcom and ARM compilers and these assignments had to be expanded to a sequence of element assignments.

There was also the issue of referring to the address of the first (unnamed) element in a structure by just the value of the structure. This required a large number of changes to name these elements and change the references to addresses of the new named element. Once again, this was necessary for both compilers.

64 bit integers

The Inferno source code makes a reasonable use of 64 bit integer arithmetic. As both the Watcom and the ADT compilers do no support emulation of 64 bit integer support on 32 bit processors such as the PC and ARM7500 in the Netslate, all the 64 bit data types where changed to structure definitions and functions written in C to do the operations on these structures.

Wide Characters

Inferno™ has native wide character (unicode) support. Most compilers store these unicode characters as a unsigned shorts (16 bits) but it was found that after much debugging through the ADT compiler output, unsigned ints (32 bits) were being used. This resulted in only very minor changes to the source even though it caused many major bugs.

Assembler Routines

A number of functions in the Inferno libraries were implemented in a "strange" assembler syntax. As speed was not a great concern, these functions where rewritten in C for portability and used both under OS/2 and with the Netslate.

Evaluation:

A lot of the changes are simply making the source code compatible with a strict interpretation of the ANSI C standard. This is an eventual aim of the Inferno source once the code development stabilizes[53]. The issue of 64 bit integer support has yet to be standardized at all even between the different 64 bit unix implementations and the assembler syntax has always been a battle between the unix and PC programmers.

It should be noted that there is still a bug remaining in the 64 bit integer multiply function in that carries are not correctly handled for large values. As this does not often occur, it has been left for future work.

Standard C Libraries

Problem:

The Inferno™ source code is not completely stand-alone and requires some standard C library functions like the string and memory copy functions. Where are these functions to be obtained?

Alternatives:

There are two basic alternatives: each C compiler is supplied with standard libraries that provide the required functions or the source code for these functions can be obtained from the freely available GNU g++ library.

Solution:

The standard libraries shipped with the Watcom™ and ADT compilers were used as they are known working bug free code. However, this caused a number of problems that were worked around.

The Watcom standard libraries caused function name clashes with the Inferno™ source code. As these particular functions are called from within the startup code generated by the linker, a patch was made to the Inferno™ code (malloc/free functions).

The necessary ADT standard C libraries where easy extracted from the unneeded functions but that required that the exception handling code had to be modified to redirect the errors to sensible places.

Evaluation:

Both sets of standard library functions work well but it is thought that the error and exception handling code needs more work before it is complete. As this is not critical for standard operation, it has been left for future work.

3.2.3 OS/2 Port

The port of the Inferno™ hosted kernel to OS/2 involved the writing of an OS/2 specific kernel functions and the necessary "device" drivers to all processes running under Inferno™ to access the OS/2 peripherals.

Threaded Kernel

Requirement:

The Inferno™ emulated kernel uses multi-threading to emulate a the multitasking structure of the native Inferno™ kernel. The required functions include:

- Thread support (start/end/kill/sleep)
- Test-And-Set operator (TAS)
- Rendezvous operators
- Console input
- System time and date
- High speed time (milliseconds)

Solution:

The OS/2 threaded kernel support (os_os2.c) is a complete rewrite of the posix (unix) code. The required functions were implemented as follows:

Thread Support The thread (start/end/kill/sleep) are simply calls to the OS/2 Thread API.

TAS The TAS operator is simply implemented using a global mutex semaphore.

Rendezvous The rendezvous operators are implemented using the private event semaphores provided by OS/2.

Console Input The console input is a simple call to the OS/2 blocking keyboard input API.

Time and Date The date and time is a simple conversion from the OS/2 "DOS" style separate date and time to the single unix style date/time used in Inferno™.

High Speed Time The millisecond time is implemented using the OS/2 free running timer. As this timer is 32 bits wide, there will be a glitch after the OS/2 operating system has been running for about 50 days which is unusual for a IBM compatible PC operating system.

Evaluation:

Once various bugs in the setting and clearing of the mutex semaphores used in the rendezvous functions had been fixed, the OS/2 kernel appears to be completely stable and reasonable efficient. The kernel still has problems with handling of external signals (user hitting Control-C or a kill signal from OS/2) in that these always end up with a complete shutdown of the hosted Inferno™ kernel.

OS/2 Filesystem Driver

Problem:

To allow access of the OS/2 filesystem to Inferno™ processes, a filesystem driver is required implementing all the standard open/read/write/close/dir operators.

Solution:

The OS/2 filesystem driver (devfs_os2.c) is a port of the posix filesystem driver. This involved three main changes: the replacement of the posix API with that of the OS/2 API, the specification of binary file access mode and the generation of *virtual* inode numbers from a hash of the file path and drive letter. An amount of code was removed due to the fact that OS/2 is single user without user and group id's.

Evaluation:

The filesystem driver works fine but depends upon the fast that the handles used in the Watcom C library open/read/write/close functions to be equal to the handles returned from the OS/2 base API (DosOpen/DosRead/DosWrite/DosClose).

IP Network Driver

Problem:

To allow Inferno™ processes to access the OS/2 networking, a TCP/IP socket level driver is required.

Solution:

The OS/2 socket driver (ipif_os2.c) is a simple port of BSD code (as OS/2's TCP/IP is a direct port of BSD). However, some changes were necessary in the handling of socket closing as in OS/2 the socket handles are different than file handles and have to be closed with so_close not just close (which calls DosClose).

Evaluation:

The IP socket driver works fine but it gets confused with what IP number to use when there is multiple IP interfaces defined under OS/2 (eg. PPP link to Netslate and Ethernet to Internet). This causes no problems but is an interesting "feature".

Audio Driver

Problem:

To allow Inferno™ processes to access the OS/2 multimedia sound system, an audio driver is required.

Audio Driver

Solution:

As there is no real requirement for audio in this port as part of the Netslate project, the audio driver (audio_os2.c) is simply a stub.

Graphics Driver

Problem:

To allow the graphic screen of the hosted Inferno™ kernel to be seen, a graphics driver is required to display the image in a window of the host operating system. This is made more complicated under OS/2 by the fact that OS/2 does not allow a process to have both a text-mode command line console and a graphical PM window.

Solution:

The OS/2 graphics driver for Inferno™ (win_os2.c and pmemu.c) uses a technique similar to IBM's implementation of Java where there is two separate processes, one with a command line interface running the VM and another with a graphical PM interface displaying the graphics.

This has been implemented using various OS/2 inter-process communication techniques and the DIVE direct video interface of OS/2 PM. The Inferno™ kernel runs as the command line process (emu[.exe]) and communicates with the graphics process (pmemu[.exe]) via the following methods.

- The mouse position and button status is sent to emu from pmemu via an OS/2 inter-process queue.
- The keyboard input, but not currently the alt/control key status, is also transferred from pmemu to emu via another OS/2 inter-process queue.
- The screen refresh event is passed from pmemu to emu via an OS/2 named event semaphore.
- A portion of named shared memory is used to communicate the size and position of the graphics window and any other covering windows.
- The screen move/window overlap events are passed on to the emu from pmemu using a named event semaphore.
- The actual graphics buffer is written to directly by the Inferno™ graphics libraries inside the emu process.

Hence, the graphics process, pmemu is a very simple front end which just sends and received graphics and input events from it's graphical window and passes them on to the main emu process.

Evaluation:

The is still a few event handling bugs remaining in the pmemu program to be tracked done and fixed in the future. However, it works well with graphical window update rates in the order of 160 times per second (much faster than a normal 72 Hz monitor refresh rate).

Serial Port Driver

Problem:

To get experience in writing Inferno™ kernel drivers and to enable the prototyping of Inferno™ applications that use the serial port, a serial driver for the OS/2 hosted kernel was required.

Solution:

The serial driver (dev_ser.c) was written from scratch using a copy of the device driver template from native kernel (devxxxx.c) as the basic structure. The configuration protocol was taken from the from native kernel serial drivers so as to match the serial interface on the Netslate. As the OS/2 API for accessing the serial port is somewhat complicated due to the huge number of options, the GIO/2 serial library is used.

Evaluation:

The OS/2 serial driver works well at all speeds as the OS/2 serial drivers are very good. It should be noted that this driver currently assumes that you have two serial ports under OS/2.

3.2.4 ARM7500 Port

The second and primary port of Inferno™ was to the Netslate and the ARM7500. Although Inferno™ had previously been ported to the ARM7500 evaluation board, it was not possible to easily obtain this code. Hence, a complete port to a new processor and platform was completed.

With the Inferno™ native kernel code written for portability, there was not a huge amount of code to be written but what did need to be written required a very detailed knowledge of the internals of Inferno and the ARM7500 processor. Also because of the very low-level and hardware dependance of this code, debugging it was a time-consuming process.

The port to the Netslate involved four main areas: A boot loader, initialisation code, the kernel code and device drivers. Each of these sections are described below.

Note that no attempt was made to port the Inferno™ DIS just in time (JIT) compiler as it was known that the source code has bugs. These bugs have been since fixed by Lucent but no updated source could be obtained (See Section 3.2.1.1).

3.2.4.1 Boot Loader

Requirements:

The basic requirement of the Netslate II boot loader is to load the code image of Inferno™ kernel into RAM and execute it. To allow the development of software on the Netslate from a laptop computer, it must be able to accept the code image from the serial port as well as from the FLASH ROM. In addition, it should support the loading of compressed images all more efficient use of the FLASH ROM space and to speed up the serial loading.

Solution:

The implementation of the boot loader (nsloader) and the OS/2 serial client (nsboot) programs have gone through many revisions from the limited initial version (only 582 bytes!) to the final demonstrated code. The serial client is currently only available as an OS/2 executable but as the only OS/2 dependent code is the serial port functions, it should be easily portable to other operating systems.

The overall boot process carried out in the final version is as below:

1. **Processor Initialisation**
   The entry code (load.s) configures the processor and sets up a stack for use by the following C code.
2. **Find the Image**
   The C code (main.c) first finds the source of the image by the CTS signal of the serial UART (which active if connected to another serial device) and a simple timeout. If CTS is present, an image is downloaded as a standard zip file from the serial port using a simple packetizing protocol. Otherwise, the FLASH ROM is searched for a valid zip file.
3. **Copy the Code**
   Once a valid zip archive has been downloaded via the serial port or found in ROM, it is searched for a file named `CODE.IMG`. This file is copied to the beginning of RAM, and uncompressed in the process if necessary.
4. **MMU Configuration**
   Next the Memory Management Unit (MMU) is configured such that the RAM is continuous and located at the start of the virtual address space. The empty regions of address space are marked invalid as to help in tracing software bugs and the IO address space is marked not cache-able.
5. **Go**
   Finally, the MMU and processor cache is enabled and the boot loader jumps to address 0x00000000. This starts the execution of the code image as if the code image itself was located in the ROM upon reset. However, importantly, the code is now in RAM allowing for the interrupt and exception vectors to be modified.

Evaluation:

The serial boot loader has provided to be most worthwhile allowing for software development without the use of a ROM emulator of sufficient size (which was not available). The use of a standard compressed zip archive has allowed for the use of a single zip file for both the Inferno™ filesystem and the kernel code image. The compression of this image, on average 53%, has allowed for much more to be stored in the limited amount of FLASH ROM and for the serial boot process to be about 40% faster.

3.2.4.2 Inferno Initialisation

Requirements:

The initialisation code for Inferno (`main.c`) requires that a simple simple memory check be completed, the zero initialized area of the code to be cleared. Finally, the kernel must be configured and then executed.

Solution:

This code was straight forward with the memory check borrowed from Inferno™ 80386 port, the zero initialisation code was copied in form from ADT Documentation and the configuration code is based on the AMD29K Inferno™ port modified for the amount of available memory, number of processors, etc. in the Netslate.

Evaluation:

It appears that the memory check too quick to be any good in detecting anything other than very very bad memory but does put random "junk" in the non-initialized area which is very useful for tracking down bugs due to uninitialized variables. Overall, the initialisation code is very small and quick with the majority of the size being all the debugging print statements.

3.2.4.3 Kernel Code

The Inferno™ code consists almost entirely of processor independent C code but requires a small amount of assembler code (`l.S`) to interface to handle a few very processor dependent tasks. The following problems document the code written for this purpose in the port to the Netslate.

Kernel Semaphores

Requirement:

The Inferno™ native kernel requires a Test-And-Set (TAS) Operator as the basis of implementing semaphores and other constructs.

Solution:

The ARM7 processor cell used in the ARM7500 has a `SWAP` instruction designed particularly for the implementation of the TAS operator. The solution is thus just a very small piece of assembler code that uses of ARM7
texttttSWAP and returns previous value.

Evaluation:

The assembler implementation is small and reasonably fast and has caused no problems. The returning of previous value in this TAS function allows the use of uninitialized semaphores to be detected as the value returned will not be zero or one.

First Level Interrupt Handler

Requirement:

The ARM7 has only a single entry point for all interrupts and this must be handled by a first level interrupt handler that dispatches the interrupt to the correct handler. This must be done in such a way that a task change can occur within the dispatched interrupt handler.

Solution:

The first level interrupt handler consists of two major sections, the entry and the distribution.

The entry code for the ARM7 interrupt is a "tricky" bit of code that deals with the multiple processor modes of the ARM7. Standard execution is carried out in *supervisor* mode where as entry to the interrupt occurs in *interrupt* mode. As these different modes have partially different register sets, this entry code basically saves the values of all the required registers and returns to *supervisor* mode before calling the interrupt distribution code. A similar reverse procedure restoring the saved registers is preformed on exit of the interrupt.

The interrupt distribution code is based on Ben Williamson's table based priority distribution code with slight improvements for code size.

Evaluation:

The entry code is small but very critical to correct operation. The current implementation is stable even with interrupt nesting but was the source of quite a few very hard to find bugs during development.

Interrupt Control

Requirement:

The Inferno™ native kernel requires functions to enable/disable/test the current status of interrupts.

Solution:

The implementation is basically a port of 80x86 code to the ARM7. This was reasonably straight forward with the good documentation included with the ADT.

Evaluation:

These functions work 100% once the clear/set and enable/disable relationship was correctly figured out and the correct return values from these functions achieved.

"C" Long Jump

Requirement:

The Inferno™ source code makes use of a function similar the "C" long jump function for error handling and task swapping.

Solution:

The implementation of the long jump function is very dependent upon ARM compiler and particular flags used during the compilation of the C code due to which of the ARM7 register need to be saved and which do not. The current implementation works well although is conservative in saving and restoring more registers than is needed with the optimal C compiler flags used.

Evaluation:

The long jump function worked first go (very surprising!) and a small piece of debugging code that preforms a simple check on the place that is being jumped to has helped to catch quite a few bugs.

Exception Handlers

Requirement:

The ARM7 processor and it's memory management unit (MMU) cause a number of exceptions for invalid operations. These are for things such a reset (a call to a null pointer) and MMU aborts (access of invalid memory regions).

Solution:

As when exceptions occur, the processor is not apt to be in a stable state, the exception handlers which dump the values of all the registers to the serial port is written entirely in assembler as to not require the use of a valid stack.

Evaluation:

Once the interpretation of these register dumps had been figured out, it is normally possible to determine exactly where the processor was when it *crashed* and what the values of the local variables were. This almost a necessary feature for finding bugs in complex multitasking code such as Inferno™.

3.2.4.4 Device Drivers

For the Netslate hardware to be useable by the Inferno™ kernel and applications, a number of device drivers were required. There were two classes of device driver written, the debugging serial driver which is completely stand-alone from the Inferno™ kernel and the other drivers which give the kernel access to the peripherals.

Debugging (serial) Driver

Problem:

For use in outputting debugging information from software running on the Netslate, a set of display string, byte, word, etc. functions are required.

Solution:

The two possible output devices that the Netslate contains is the LCD display and the serial port. As printing characters to the display is not a simple task, the serial port was chosen for the debugging output. A very small and simple set of polling serial functions (dbg_ser.c) were written and are used throughout the Netslate port of Inferno™ for displaying debugging information.

Evaluation:

The polling serial drivers have proved very robust even to the point of them working successfully within the interrupt driven serial port drivers. When the FIFO buffers of the serial port are enabled by the interrupt driven serial drivers, the polling routines work fine but their output is apt to be overwritten in the FIFO by the transmit interrupt handler.

Serial/IrDA® Driver

Problem:

Both the Netslate II's serial port and IrDA® port use NS16550 compatible UARTs. For their use under Inferno™, interrupt driven drivers are required.

Solution:

Inferno™ comes with a large driver for the 16552 (a dual 16550 UART) and this was modified to become the serial and IrDA® port driver (devuart.c). This worked successfully for most of the porting process as a serial console and general serial port driver.

However, once the TCP/IP networking was being tested, the driver would suffer overrun errors (not responding to the serial port fast enough). Thus a policy of KISS (Keep It Simple Stupid) was applied and the driver rewritten for speed removing a large number of unused features such as XON/XOFF handshaking and serial breaks.

Evaluation:

The optimized KISS serial driver preforms well handling maximum speed (115.2 kbit/s) input on both the serial and IrDA® ports simultaneously without errors. To date, the missing features eliminated have not been missed.

Local Read-Only Filesystem

Problem:

One of the requirements from the Netslate vision is stand-alone operation of the software. To support this, a reasonable number of files have to be accessible to the software running under Inferno™ as both application programs and data files such as fonts and images. For this purpose some form of simple filesystem was required.

Solution:

Due to the limited size of the FLASH ROM storage on the Netslate II, the idea of using a standard PKZIP format compressed archive file as a read-only filesystem was borrowed from Java™ 1.0 class libraries. The coding was a fairly simple task to complete as source code for the actual decompression is freely available as the zlib 1.04 library from InfoZIP[50]. The creation of this filesystem is very simple as the filesystem is completely compatible to all recent PKZIP 2.04G compatible archivers - just PKZIP the files.

Using the zlib library functions and the PKZIP file format specifications[41], the zip filesystem (devzipfs.c) was written using the standard Inferno™ device driver template and references to the filesystem driver used in the OS/2 hosted kernel.

Evaluation:

Once the required undocumented options for the zlib decompression functions were found that enabled the decompression of the PKZIP deflated format compressed data, the zip filesystem has proved to be most worth while. The use of standard the PKZIP utility to create the filesystem and the average 53% compression ratio achieved has speed up the porting of applications to the Netslate II greatly.

The only current weakness in the zip filesystem is that the logic for finding the zip file in the Netslate II's address space (either FLASH ROM for demonstration or DRAM for development) is not very good and it often does not find the zip file if there is multiple fragments existing from previous reboots. This requires a simple addition to the boot loader to clear the DRAM upon reset and this will be implemented for the demonstration of the Netslate II.

Graphics

Problem:

The color LCD display is the primary output device to the user. The Inferno™ kernel requires a device driver.

Solution:

The development of the graphics device driver (screen.c) for the Netslate was not the simple task it should have been.

The Inferno™ graphics libraries only support 1, 2, 4, 8, 16, and 24 bit color displays but not the 12 bit color used by the LCD panel on the Netslate. To make matters worse, the 12 bit color format used by the LCD panel has a partially out of order format in it's video buffer requiring a different ordering of the red, green and blue sub-pixels depending upon location.

The only solution other than rewriting large chunks of the Inferno™ graphics code, was to use two separate video buffers, one 8 bit and one 12 bit and some code that translated between the two. This was implemented in the graphics driver (screen.c) by using a 256 entry color map and the screen refresh "hook" provided by the Inferno™ kernel. This refresh driver looks up the color of the 8 bit pixel in the color map and then places it into the 12 bit display buffer after doing the necessary sub-pixel shifting.

Evaluation:

The graphics performance of this buffer translating graphics driver is not particularly great and it requires a fair portion of CPU time but is it faster than the response time of the passive matrix color LCD panel on most screen updates.

Trackball

Problem:

The primary source of user input to the Netslate II is the trackball. Hence, a Inferno™ driver was required.

Solution:

The driver (mouse.c), was copied from the Netslate I demonstration software and enhanced to improve the reliability of the initialisation of the trackball. It was then modified to interface into the Inferno™ kernel.

Evaluation:

The trackball driver works fine with the Inferno™ demonstration applications. The only slight glitch is that the driver will pause the Inferno™ boot process for about 10 seconds if the trackball is not connected correctly upon startup.

3.2.4.5 Inferno™ Bugs

Only a very small number of significant bugs were found in the first release of Inferno™ used (Inferno™ 1.0 build 1). Both of these significant bugs are described below.

PPP Bugs

Problem:

the Inferno™ PPP does not connect to the OS/2 PPP and when it does the PPP header compression is not enabled.

Solution:

The problems were due to incomplete code in the PPP negotiation process. After a careful read of the code it was pretty obvious what was missing and when added the majority of the problems were fixed.

Evaluation:

Still does not connect to the IBM OS/2 PPP.EXE but works fine with the InJoy™ OS/2 Dialer.

TCP Bugs

Problem:

TCP packets originating from the Netslate are retransmitted almost immediately causing both incoming and outgoing TCP connections to be very much slower than they should be.

Solution:

As this is a timer problem (very common in TCP implementations) which is not a simple matter to debug, and that the bug has been fixed in the latest Inferno source, attempts have been made to get the latest code. Details of these attempts can be found in Section 3.2.1.1.

3.3 Applications

The application defined for the Netslate II in the Netslate vision is that of running a stand-alone web browser. The Inferno™ operating system now running on the Netslate II, is shipped with two sets of demonstration applications.

The first of these are a simple and very small set of applications designed to run on a TV set-top box style network computer. These applications are known as the “MUX” environment and include amongst other things a very simple and somewhat limited web browser.

The second set of demonstration applications is the “WM” environment which is targeted at the desktop network computer market. The “WM” environment include a much more capable web browser.

As both of these applications have been written in the Inferno™ platform independent form, they have been "copied" into the Netslate's FLASH ROM and successfully executed without modification. The following sections give some details on what these two demonstration environments provide in terms of the Netslate vision for a stand-alone web browser.

3.3.1 General Set-top Box Software

The MUX demo application set provides the basic functionally of TV Set-top box applications. These in the MUX set include:

- Financial News Application
- Movie Database
- On-line Newspapers
- TV program guide
- On-line Pizza Ordering
- A simple Web Browser

Figure 3.1: Sample "MUX" Application Screen

These applications run well on the Netslate as the particularly simple user interface (See Figure 3.1) suits the overall style of the Netslate II. As they have been designed for use with a TV remote control, it has been easy to modify them for use only with the Netslate's trackball.

However, unfortunately, the web browser which is to be the primary application, is very limited and only supports a very small portion of the simplest web pages available. This makes it basically unusable. In addition, the internal politics within the Inferno™ development team as meant that no further work is being done on the "MUX" environment[53].

3.3.2 Simple Desktop Network Computer

The second Inferno™ demonstration environment is "WM". "WM" provides a basic desktop network computer windowing system and applications. These applications include a web browser, email client, file manager, text editor, multimedia viewers, and a text mode terminal emulator. Figure 3.2 shows a typical screen view with the web browser running.

The "WM" web browser *Charon* is much more capable than the "MUX" one as is very useable even though the Inferno™ 1.0 version does not support many of the advanced features of the web now in use such as frames. The latest version of *Charon* to be included in the coming Inferno™ 2.0 release is fully HTML 3.2 compatible complete with Java™ applet support (i.e. Matches the PC Netscape in capability).

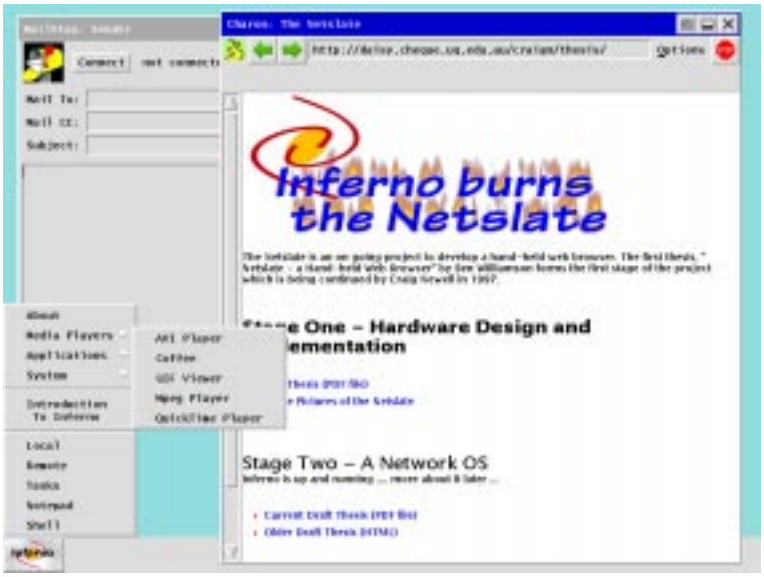

Figure 3.2: Sample “WM” Application Screen

Thus, although the “WM” environment is not quite so suited to “mouse” only input, it provides the Netslate II with the best set of applications, a reasonable web browser and a set of multimedia viewers for content such as AVI and Quicktime™ movies.

Chapter 4

Network Computing

Contents

During 1997, the Netslate project has been continuing with the Netslate II hardware and software achieving what could almost be a prototype commercial product. However, the rest of the world has also been busy with visions similar to the Netslate and in doing so have created a new field of computing products known as network computers.

This chapter attempts to show what the current network computing market contains in both hardware products (Section 4.1) and software (Section 4.2). The number of products currently available in 4th quarter 1997 has expanded tremendously since the start of the Netslate II project at the end of 1996 and the rate is still increasing. Hence, it should be remembered that this snap-shot of network computing is of the end of September 1997.

4.1 Hardware

The Netslate vision is for a small purpose built computer that uses a network connection as the source of information behaving much like a user interface to the network (the world wide web in the case of the Netslate). The following two sections detail the current range of similar purpose built network computers and a brief examination on what sort of thing is inside the case.

4.1.1 Current Products

This section gives an overview of the current network computers on the market. There is a huge number of products available to fill every possible information terminal requirement and the following examples are grouped according to physical packaging.

It should be noted that even though there is a very large number of products available, a good portion are just repackaged (OEM'ed) versions of a single design.

4.1.1.1 Slates

Figure 4.1: Slate Network Computers: The NewsPAD[36](left) and Fujitsu PT510[43](right)

Closest to the Netslate vision are the slate network computers (See Figure 4.1). These are either custom built products like the NewsPAD[36] that is designed as an electronic replacement for a paper newspaper or more general purpose devices such as the Fujitsu PT510[43] which are in effect repackaged IBM compatible computers running standard PC software.

4.1.1.2 PDAs

Figure 4.2: PDA Network Computers: Apple Newton MessagePad 2000[31](top left), Toshiba GNEIO[15](top right), Nokia Communicator 9000[32](bottom left), Psion Workabout Scanner[45] (bottom middle), Brother Geobook[16](bottom right)

The Personal Digital Assistant (PDA) (see Figure 4.2) has been expanded in the latest range of products to embrace the portable cellular phone and the Internet. The Toshiba GENIO[15] and the Nokia Communicator 9000[32] are the best examples of this consisting of a PDA with web browser and email integrated into a mobile phone.

Other PDAs such as the Apple Newton Message Pad 2000[31] and Philips Velo[47] (Windows CE) have pushed the networked PDA ever closer to a full function desktop computer complete with network access.

4.1.1.3 Screen Phones

Figure 4.3: Screen Phone Network Computers: Alcaltel Webphone[3] (top right), CIDCO iPhone[23] (center), Nortel's Webphone[25] (bottom left), Samsung's Webphone[25] (bottom right)

The network computer has also been integrated with the conventional voice telephone (See Figure 4.3) in a large range of screen or web phones. Here, the integrated network computer allows access to the increasing range of directory services available on the web in addition to functioning as a simple email terminal.

4.1.1.4 Set-top Boxes

Figure 4.4: Set-top Box Network Computers: The RCA NC100[33] (left) and Teknema Easyrider[11] (right)

The entertainment value of the World Wide Web is also being exploited with a range of network computers packaged as TV set-top boxes (See Figure 4.4). There is many products available but most appear to be rebadged versions of three or four designs. The base functionality is very similar to the Netslate vision with the LCD replaced by the TV, the wireless network interface by a modem and the trackball by a remote control.

4.1.1.5 Desktop NCs

Figure 4.5: Desktop Network Computers: The Tektronix NC200 Series[34] (top left), the NCD Explora 400[14] (bottom left) and the DEC StrongARM™ Reference NC[10] (bottom right)

The last major division of Network Computers is much more general purpose than that of the Netslate vision. Here the "NC" as it has been coined is a modern graphical network terminal much like an X-terminal. However, by using platform independent software, much more processing than just displaying the information is undertaken within the "NC".

4.1.2 What is inside them?

Although the physical packaging of most network computers is different than that of the Netslate, the actual electronics within are very similar. The Netslate is a single board computer built around a single chip computer, the ARM7500. This is very similar to most network computers as can be seen in Figure 4.6 which is a single board computer built around the StrongARM™ processor and is used within desktop and set-top network computers.

One *unique* feature of the Netslate in modern computing is that it does not use an 80x86 compatible processor. The latest version of the ARM7500 chip used in the Netslate, the ARM7500FE is only one of many single chip computers used in network computing products. Table 4.1 lists some of the more common processors used.

In addition to the processor, there is usually a number of peripherals that create a functioning product like in the Netslate. The typical specifications for these additional components are listed in Table 4.2 and can be seen to be very close to what has been implemented on the current Netslate II.

4.2 Software

The possible section of software suitable for the Netslate vision was very limited when the choice was made (See Section 3.1). With the huge growth in the number of network computers available commercially, this software situation has changed dramatically.

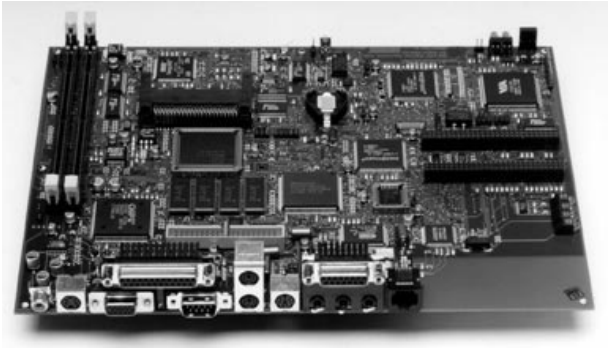

Figure 4.6: Inside a Network Computer: The DEC StrongARM™ Reference Platform[10]

Processor Family	Example Products
ARM	ARM7100, ARM7500FE, StrongARM™
MIPS RISC	R3910, VR4101
Hitashi Super H	SH-3
PowerPC	MPC823, MPC860
Intel Compatible	386EX, 486EX and clones

Table 4.1: Single Chip Computers used in Network Computers

However, the wide range of hardware available has meant that the software used tends to be very product specific unlike with IBM compatible software where it is possible to run most programs on almost every computer. With this and the short development time of network computers (just count the numbers), the only software choice is that of the operating system and this comes bundled with the major applications such as web browsers.

4.2.1 Operating Systems

Unlike the very limited choice of possible operating systems available an the end of 1996, there is currently a reasonably wide range with many more just being announced just before this was written.

All the operating systems on the market can be classified into two categories depending upon the portability of the applications written for them. In the platform dependent class, the application is compiled into native code for the particular processor being used whereas the platform independent operating systems require their applications to be written in a platform independent way.

Component	Specifications
RAM	4Mb+ DRAM, 128kb+ SRAM
ROM	512kb+ ROM, 1Mb+ FLASH ROM
Network Interface	Ethernet, ATM, Cable Modem, Telephone Modem
Display	LCD panel, SVGA, NTSC/PAL Video
User Input	Keyboard, Mouse, Microphone, remote control
User Outputs	Sound, Printer Port, IrDA™

Table 4.2: Typical Network Computer Peripherals

4.2.1.1 Platform Dependent

There are literally hundreds of small operating systems on the market (8 advertised in one issue of C/C++ Users Journal[7] alone). Some of the more recent products are designed specifically for network computing are listed in Table 4.3.

Operating System	Company
GEOS[17]	Geoworks
EPOC[13]	Psion Software PLC
NewtonOS[37]	Apple
Ciao[6]	General Picture
netOS[35]	Neoware
DAVID[9]	Microware Systems Corp.
Windows CE[30]	Microsoft®

Table 4.3: Network Computer Operating Systems

In addition many of the more general purpose small operating systems are also targeting the network computer market with products which are subsets of their full operating systems. Some of these are listed in Table 4.4

Product	Company
QNX4 + Photon GUI[44][1]	QNX
CHORUS/Jazz[5]	Chorus Systems
OS-9[12]	Microware System Corp.
The Internet Package[49]	Wind River Software

Table 4.4: General Purpose Network Computer Operating Systems

4.2.1.2 Platform Independent

The explosion of the number of computing platforms in recent times has lead to an ideal of the application developer – "write once - run anywhere". This is even more important in the network computing field where there is many different platforms available.

There is three basic solutions to this "write once - run anywhere" problem. These are scripting, virtual machines and run-time compilation and these areas are examined below.

Scripting Scripting or run-time interpretation of the text of the source code is the first solution. This has been available ever since the explosion of the home computer and the BASIC language. There are only a few examples of this approach still in use today with the Apple NewtonOS™ [37] probably the best example.

Virtual Machine The current *in* solution is the virtual machine. Here the source code of an application is compiled into an binary "byte" code. This is then interpreted by a "virtual machine" or translated in to the native machine code to run the application. The interpretation of the "byte" code is significantly faster than direct interpretation of source code and the translation of the "byte" code into native machine code (Just In Time compiling or JIT) can speed execution even more.

There are two major standards for this byte code being Java™ [24] which is used by many operating systems and DIS which is used by the Inferno™ [20] operating system. As well as being

[1] See the single floppy internet appliance demo from www.qnx.com

part of the operating system, the virtual machine can also run on top of another operating system such as in the Java™ used in most common PC based web browsers and the hosted Inferno™ as ported to OS/2 in Section 3.2.3.

From a reading of the current computing press, the battle for the dominant VM standard has been won by Java™. However, there are rumors of a UVM (Universal Virtual Machine) from Microsoft and Microsoft's continual attempts at taking over the Java™ standard.

Run-time Compilation The final method used to achieve a "write once - run anywhere" system is run-time compilation. Here, the application source code is half compiled to an intermediate binary form. This form is used in all compilers to allow for compiler optimization of the program and in run-time compilation it is stored as the application. The final stage of compilation, the translation to native machine code, is then done whenever then application is run.

This is implemented in the research operating system Oberon[38] and also as Juice[28] when the scheme is incorporated into a standard web browser as a plug in application.

Chapter 5

The Future

Contents

To end this the second stage, of the Netslate project, the following sections review the progress made towards meeting the Netslate Vision, and the future of the Inferno™ operating system and the current Netslate II's hardware implementation.

5.1 The Netslate Vision

The Netslate II has progressed greatly towards fully meeting the requirements of the Netslate Vision as setout in Chapter 1. This progress is examined according to the individual requirements below:

Stand-alone Web Browser The vision for a stand-alone web browser is complete. The Inferno™ "WM" web browser provides the Netslate II with a reasonably capable, full function web browser. The only issue remaining is one of integration of the limited user input of the Netslate II into such tasks as the entry of URL's into the browser.

Handheld Computer The Netslate II is very close to being a fully portable handheld computer. The addition of the IrDA® wireless link has reduced the number of external connections needed down to just a power cord.

Color Display The color LCD panel has been completed with the construction of the necessary power supplies and more complete mountings and functions absolutely great for its use as a web browser. This is particularly so considering that the display only cost USD$80.

Sound Output The Netslate II's internal 8 bit audio interface had to be abandoned due to problems discovered internally in the ARM7500 chip. However, considering the current limited use of audio on the web this is not much of a short coming.

User Input The Netslate II's trackball provides a very usable interface for the operation of a web browser. The issue of text entry remains but that is not a small problem considering the hardware needed for handwriting recognition.

Future Expansion The Netslate II now has a full function serial port of which to attach external devices such as a bar code wand or GPS receiver for applications other than a strict web browser.

5.2 Inferno™

Inferno™ in the form of Inferno™ 1.0 build 1 as used on the Netslate II has already been superseded by Inferno™ 1.1 and almost Inferno™ 2.0 in the 6 months since the release of Inferno™ 1.0. The latest Inferno™ 2.0 is growing in compatibility with the inclusion of PersonalJava™ and EmbeddedJava™ (a subsets of Java™ 1.1) and thus looks as if it has a future in a market dominated by the media hype over Java™. However, the particular licensing strategy being used might prevent its future wide acceptance in the network computing field.

The ever moving, ever updating Java™ family including JavaOS™ appears to have a future with its more open licensing strategy which is important in these days of very good free operating systems such as Linux and OpenBSD. However, the size and performance problems inherent to Java appear as if they are only going to be solved by specific Java processors[26][42]. Thus Java™ might become a victim of the need for speed and high volume production to keep down production costs.

The concepts and ideas behind Oberon and Juice are very neat and achieve very good performance and small size. However, without a major push in spreading the word, Oberon and Juice will always remain an academic research project.

5.3 The Netslate II's Future

With the current state of progress in single chip computers, the ARM7500 (1995) used in the Netslate II is very very out of date in terms of both speed and features. This is not saying that you can ever be up to date, taking for example the MPC821 from Motorola of which preliminary samples where obtained not less than 9 months ago has been superseded by the MPC823 which contains all the features of MPC821 enhanced plus many more and is more than twice the speed.

The other major outdated feature of the Netslate is the construction. Today most prototype boards are constructed from 4+ layer PCBs with 6 layers almost the norm for devices such as the Netslate. Hence, the point-to-point wiring over a bare two layer board of the Netslate is not worth continuing with. A large number of problems due to the nature of the point-to-point wiring, the PCB without solder mask and the very high noise levels (the current 32 MHz board produces noise up to and beyond 1.3 GHz!) should mean that the current Netslate be "retired".

The future is thus to redesign the Netslate using the StrongARM™ chip (the latest ARM series processor) including PC Card (PCMCIA) slots for expansion hardware. The addition of a modern switch-mode power supply and 16 bit stereo audio interface should also be considered as should a redesign of the LCD interface to allow cleaner software drivers.

Overall, it should be remembered that the Netslate is a project in investigating the design and implementation of a network computer and not the design of a marketable product. As such the final conclusion of the Netslate Project so far is a success.

Bibliography

[1] Advanced RISC Machines Ltd. *ARM7500 Data Sheet*, October 1995.

[2] A. V. Aho, B. W. Kernighan, and P. J. Weinberger. *The AWK Programming Language*. Addison-Wesley, 1988.
A good reference to deciphering the configuration scripts used with the native Inferno kernel to add and remove device drivers at compile time.

[3] Welcome to Alcaltel Internet Phone. World Wide Web (HTML) http://www.alcatel.com/-our_bus/telecom/products/bsd/term/ index2.html, August 1997.
A very good description of a web phone and all it's functions including some screen shots. Runs Java™

[4] AMD Non-Volatile Memory Products Directory. World Wide Web (PDF) http://-www.amd.com/html/products/nvd/nvd.html, August 1997.
The datasheet for the AMD29F010 5 volt only FLASH ROM used on the Netslate II

[5] CHORUS/JaZZ r1 Datasheet. World Wide Web (HTML) http://www.chorus.com/Products/-Datasheets/jazz.html, March 1997.
A Real-time Operating System bundled with the Java™ VM

[6] Ciao Overview. World Wide Web (HTML) http://www.generalpicture.com/gp-cov-p1.htm, February 1997.
An operating system and applications for PDAs.

[7] C/C++ Users Journal, July 1997.
A example magazine that contains lots of advertisements for embedded operating systems.

[8] Andrew Mansbridge Craig Newell and Darius Davis. *FLASH: An AMD FLASH Programmer*, June 1997. Available via World Wide Web (HTML) http://daisy.cheque.uq.edu.au/craign/-e3425/flash.
The E3425 report and reference for the FLASH programmer designed and built for programming the AMD29F010 FLASH ROMs used on the Netslate II

[9] Digital Television - DAVID. World Wide Web (HTML) http://www.microware.com/html/-digital_television.html, 1997.
An OS-9 based operating system for digial TV's.

[10] DIGITAL Semiconductor StrongARM Microprocessors: Network Appliance Reference Design. World Wide Web (PDF) http://www.digital.com/semiconductor/strongarm/dna.htm, August 1997.
A hardware reference design for a generic network computer.

[11] Teknema Inc. - Product. World Wide Web (HTML) http://www.teknema.com/easyrider.htm, August 1997.
The Teknema Easyrider TV set-top box. The web page gives some detailed hardware specifications and also the identities of a couple rebadged versions.

[12] Embedded OS-9. World Wide Web (HTML) http://www.microware.com/html/embedded_os-9.html, 1997.
The operating system being used in a number of set-top boxes and screen phones including Novatel WebPhone and IBM Network Computers

[13] Psion Softare Technology and Architecture. World Wide Web (HTML) http://-www.software.psion.com/tech/techhome.html, September 1997.
The EPOC32 operating system is used on a number of PDAs and includes complete networking and internet applications.

[14] Network Computing Devices Inc. World Wide Web (HTML) http://www.ncd.com, August 1997.
A manufacturer of X-Terminals who has jumped on the Network Computer hype and is selling it's X-Terminals as Network Computers. However, it is only the top of the range models that actually have local execution of applications

[15] Toshiba GENIO PCV. World Wide Web (HTML) http://www.geoworks.com/devices/genio, 1997.
A mobile phone and PDA with PIM and web browser in one. Japan Only

[16] Brother GeoBook. World Wide Web (HTML) http://www.geoworks.com/devices/geobook/, September 1997.
A very small and limited laptop that is really a PDA in a different package. Comes complete with office applications, a web browser and email client.

[17] GEOS: The Flexable Foundation for Handheld Devices. World Wide Web(HTML) http://-www.geoworks.com/os/sso, 1997.
A platform dependant but popular operating system for handheld devices. It is moving towards a network computer operating system with TCP/IP and applications.

[18] Novalog Inc. IrDA® ENDEC Design Hints For ASIC Integration. World Wide Web (PDF) http://www.novalog.com, June 1997.
This application note for the designers of IrDA® encoders and decoders gives some necessary pointers for making IrDA® reliable and power efficient.

[19] Novalog Inc. SIRComm MiniSIR 115.2kbps IrDA Transceiver Module. World Wide Web (PDF) http://www.novalog.com, February 1997.
The MiniSIR is the IrDA compatible infrared transceiver used with the Netslate due to its easy availability in samples and small size and power consumption.

[20] Lucent Technologies' Inferno™. World Wide Web (HTML) http://207.121.184.224, October 1997.
The internet home page for the Inferno™ operating system used on the Netslate II

[21] Infrared Data Association. Serial Infrared Physical Layer Link Specification. World Wide Web (PDF) http://www.irda.org, October 1995.
This is the full technical specifications of the IrDA physical layer and as such is not easy to read. A more understandable description of the infrared encoding can be found in [22].

[22] Texas Instruments. TIR1000 Standalone IrDA Encoder and Decoder. World Wide Web (PDF) http://www.ti.com, September 1996.
A very nice small IrDA Encoder/Decoder but no samples and hard to obtain in Australia. Hence, the IrDA Encoder/Decoder was designed and built using an EPLD as in Appendix B

[23] CIDCO - Internet Solutions. World Wide Web (HTML) http://www.cidco.com/product5.htm, January 1997.
One of the first screen phones available offering a web browser and email client in a telephone.

[24] What is Java™? World Wide Web (HTML) http://www.javasoft.com/nav/whatis/index.html, October 1997.
An attempt to answer the question what is Java™ in non-computer geek terms.

[25] Java™: Comming Soon to Phones, TV Set-tops, Hand-Held Devices. World Wide Web (HTML) http://www.javasoft.com/features/1997/oct/Personal.Embedded.html, October 1997.
Press release with lots of pictures of screen phones that can not be found anywhere else.

[26] JavaChips™. World Wide Web (HTML) http://www.sun.com/sparc/java, October 1997.
Web page listing all the current java processors, but only the picoJava™ CPU core is available for licensing.

[27] JavaOS™: The Standalone Java™ Application Platform. World Wide Web (HTML) http://www.javasoft.com/products/javaos/index.html, October 1997.
JavaOS™ is the attempt to make Java™ a standalone operating system. Not very sucessful from what I have heard on the grape vine but it gets lots of press coverage.

[28] Introducing Juice. World Wide Web (HTML) http://www.ics.uci.edu/~juice/intro.html, October 1996.
The implementation of the platform independant ideas behind Oberon[38] as a plug-in application for PC web browsers.

[29] B. W. Kernighan and D. M. Ritchie. *The C Programming Language, 2nd edition*. Prentice-Hall, 1988.
A useful reference for deciding what is legal to do with unions, structs and enums in C source code.

[30] Microsoft. Welcome to Microsoft® Windows® CE. World Wide Web (HTML) http://-www.microsoft.com/windowsce/default.asp, 1997.
The home page for Microsoft® 's entry into the PDA operating system market.

[31] MessagePad 2000. World Wide Web (HTML) http://www.newton.apple.com/product_info/-devices/MP2000/MP2000.html, March 1997.
A very far looking PDA with good handwriting recognition as well as a detachable keyboard. Comes with internal modem and email/web browser.

[32] NOKIA 9000 Comunicator. World Wide Web (HTML) http://www.nokia.com/com9000/-n9000.html, 1997.
A mobile phone and PDA with PIM, FAX and internet connection.

[33] RCA - Products - NC100. World Wide Web (HTML) http://www.rca-electronics.com/products, August 1997.
A set-top box rebadged and sold by RCA Electronics

[34] Tektronix: Networking Division. World Wide Web (HTML) http://www.tektronix.com/VND/, August 1997.
Another manufacturer of X-Terminals who has jumped on the Network Computer hype.

[35] netOS 3.0 Information. World Wide Web (HTML) http://www.neoware.com/netos3_info.html, September 1997.
The netOS operating system is an operating system targeted and the desktop network computer market.

[36] Datasheet 3 - NewsPAD. World Wide Web (HTML) http://www.ictnet.es/newspad/-newspad.html, June 1997.
An ARM7500 based design of a electronic replacement for a newspaper. Very much of a standalone computer in that data is downloaded to a local harddisk for viewing but the hardware is good example of what the Netslate could have been given the resources.

[37] For New Newton Developers. World Wide Web (HTML) http://www.newton.apple.com/dev/newdevs.html, September 1997.
A set of links to all the various reference manuals on the Apple NewtonOS™

[38] The Oberon Reference Site. World Wide Web (HTML) http://www.math.tau.ac.il/ğuy/Oberon/, August 1997.
The source of information on the Oberno language and operating system.

[39] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
The definitive reference to the Tk Toolkit that is used as the general purpose GUI of the Inferno Operating System.

[40] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from bell labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[41] PKWARE. appnote.txt. World Wide Web (Text) http://www.pkware.com, August 1997.
The definitative and only reference as to the file format of a zip archive file as produced by PKZIP v2.04G and compatible utilities

[42] The PSC1000 Microprocessor. World Wide Web (HTML) http://www.ptsc.com/PSC1000/-intro.html, August 1997.
A small almost stack based processor that runs Java™ very well.

[43] Fujitsu Point 510 Features and Benefits. World Wide Web (HTML) http://-www.fpsi.fujitsu.com/products/pt510.htm, February 1997.
A well done slate like computer that is very similar to the Netslate vision except that it is designed and built to be IBM PC compatible and to run standard PC software.

[44] Take the 1.44 Mb Web Challenge! World Wide Web (HTML) http://www.qnx.com/iat/-index.html, September 1997.
A single 1.44 Floppy disk containing a complete operating system, TCP/IP networking, GUI, Web browser, File Manager, and more. All free!

[45] Psion Industrial - Workabout Scanner. World Wide Web (HTML) http://www.psion.com/-industrial/indworkaboutscanner.html, September 1997.
A PDA with built in scanner that shows the wide range of PDAs available

[46] National Semiconductor. PC16552 Dual UARTS. World Wide Web (PDF) http://www.national.com, 1996.

[47] Philips Mobile Computing Group. World Wide Web (HTML) http://www.velo1.com/, September 1997.
The first of the WindowsCE® [30] based PDAs. A PDA that attempts to have everything of a standard PC is a PDA package.

[48] Ben Williamson. Netslate a Handheld Web Browser. Batchelor's thesis, The University of Queensland, 1996. Also available on World Wide Web (PDF) http://www.elec.uq.edu.au/-m̃arks.

[49] Wind River Introduces Complete Embedded Internet™ Package. World Wide Web (HTML) http://www.wrs.com/embedweb/, June 1997.
A complete package of operating system, GUI, web browser, Java VM and web server designed for the Internet Applicence market (Set-top boxes and Screen Phones).

[50] zlib Home Page. World Wide Web (HTML) http://www.cdrom.com/pub/infozip/zlib, September 1997.
A very good portable compression library that is compatible with PKZIP. This is what was used in devzipfs and in the compressed Netslate II boot loader.

[51] Ben Williamson. Email conversations benw@teknema.com, October 1997.
Discussions on various problems and solutions with the ARM7500 chip as Ben is currently working on a number of commercial products using the ARM7500.

[52] Donald Johnson. Email conversations daj@mh.lucent.com, October 1997.
A member of the Inferno™ technical support team who I got on the telephone once and have continued to bug daily via email about Inferno™ updates.

[53] Erik Van Hensbergen. Email conversations erikvh@lucent.com, October 1997.
Discussions on various problems with the Inferno™ source code and many emails trying to convince him to send me the Inferno™ 1.0 TCP/IP patches.

Appendix A

Netslate II Implementation

This appendix contains some technical details about the current Netslate II platform. The various voltage and current limits are from observation and best guess and hence should not be taken as absolute.

A.1 Specifications

1	ARM7500 running at 32 MHz
1	640x480 color passive LCD display driven by the ARM7500 in 12 bit mode
1	8 Mb DRAM SIMM (2 banks of 10x10 DRAM)
8	128 kb AMD29F010 FLASH ROMs arranged as two 512 kbytes by 32 bit banks
1	PS/2 mouse port (keyboard port is not used)
1	RS-232 Serial Port using the TI16C550 UART
1	IrDA® 1.0 Compatible IR Port using (half of the PC16552 UART)
1	unused UART (half of the PC16552)

Table A.1: Current Netslate Hardware Specifications

Min.	**Typical**	**Max.**	**Units**	**Description**
7.5	13	18	V_{DC}	Netslate
11.5	13	18	V_{DC}	LCD backlight
	200	320	mA	Netslate without display
		600	mA	Netslate with display
		500	mA	LCD Backlight

Table A.2: Current Netslate Electrical Specifications

A.2 Circuit Diagrams

The following circuit diagrams shown what it is thought to be the current Netslate II hardware. However, the circuit diagrams of the Netslate I hardware proved to be incomplete and thus these to are also apt to be incomplete.

A.2.1 ARM7500 Processor

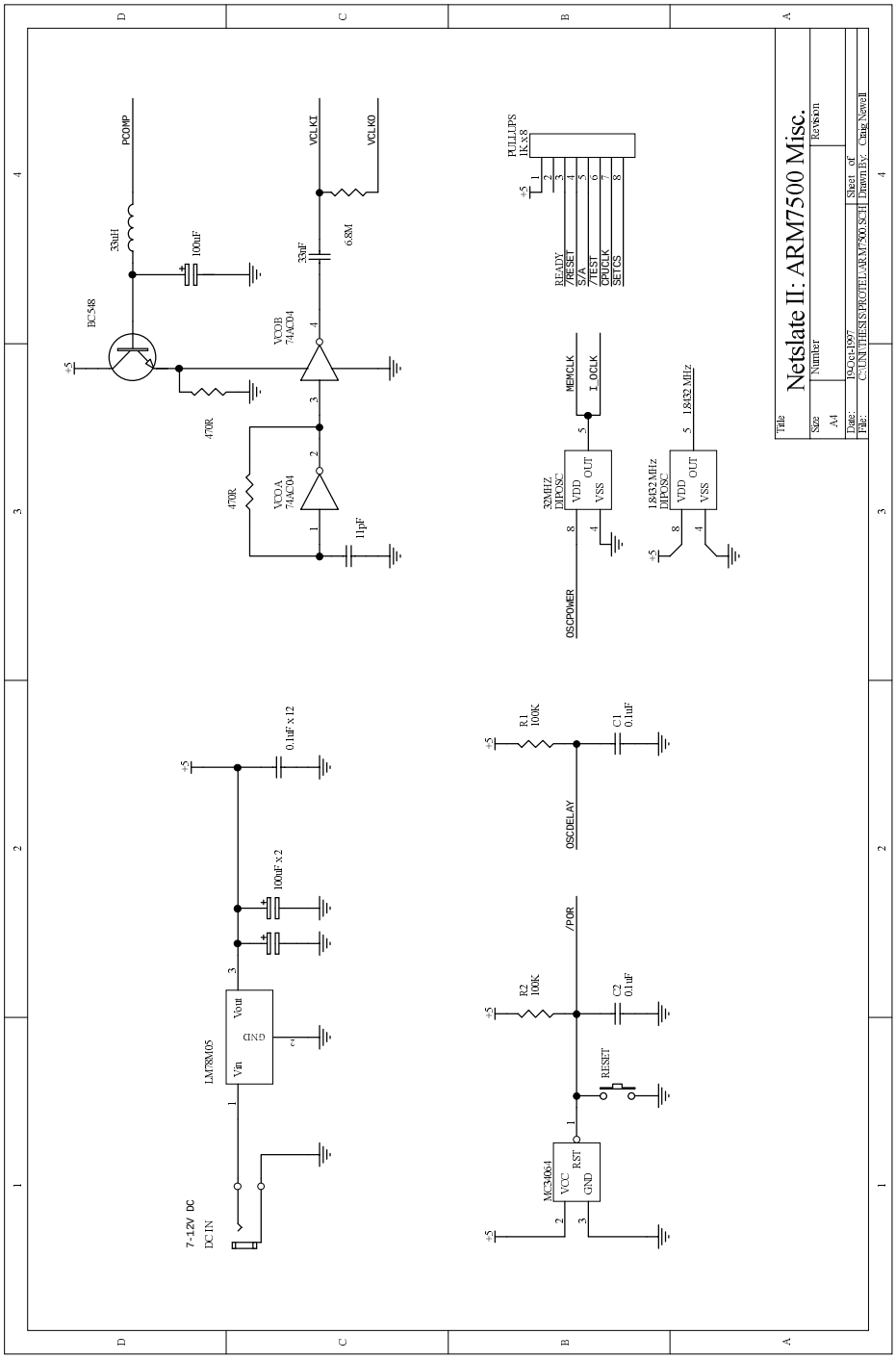

Figure A.1: Netslate Schematic: ARM7500 Processor

A.2.2 LCD Display

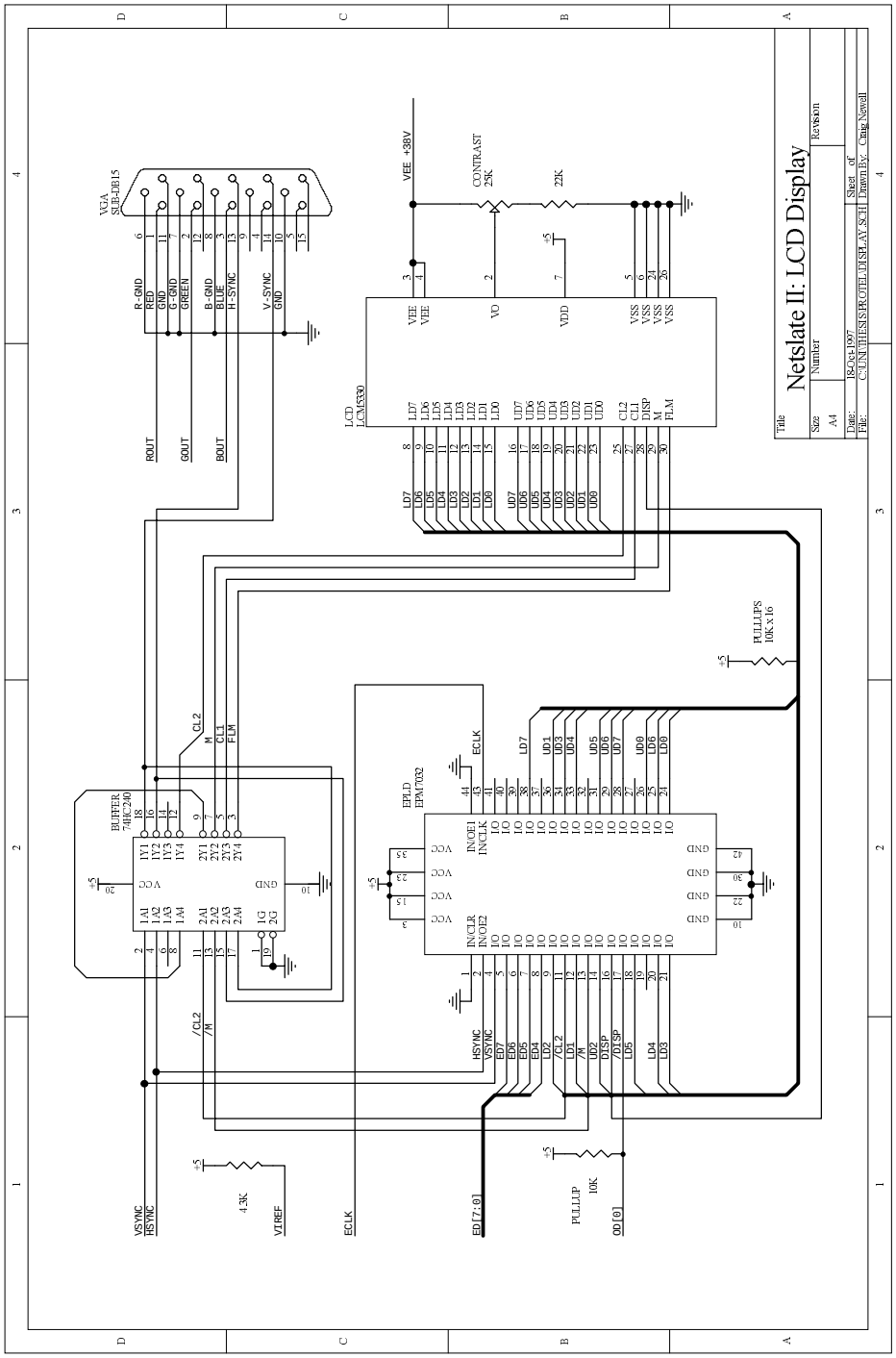

Figure A.2: Netslate Schematic: LCD Display

A.2.3 IrDA® and Serial Interfaces

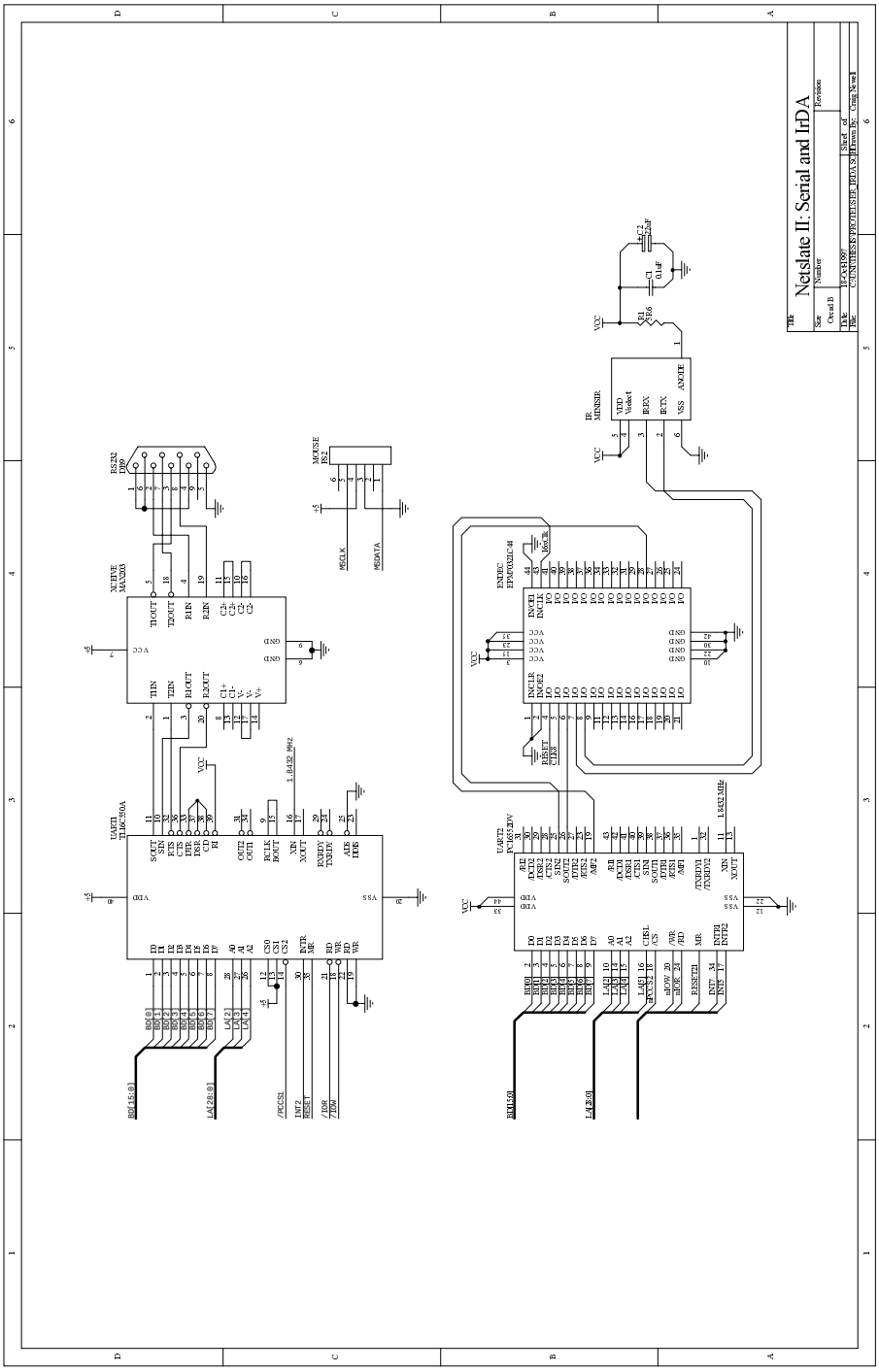

Figure A.3: Netslate Schematic: IrDA and Serial Interfaces

A.2.4 DRAM and ROMs

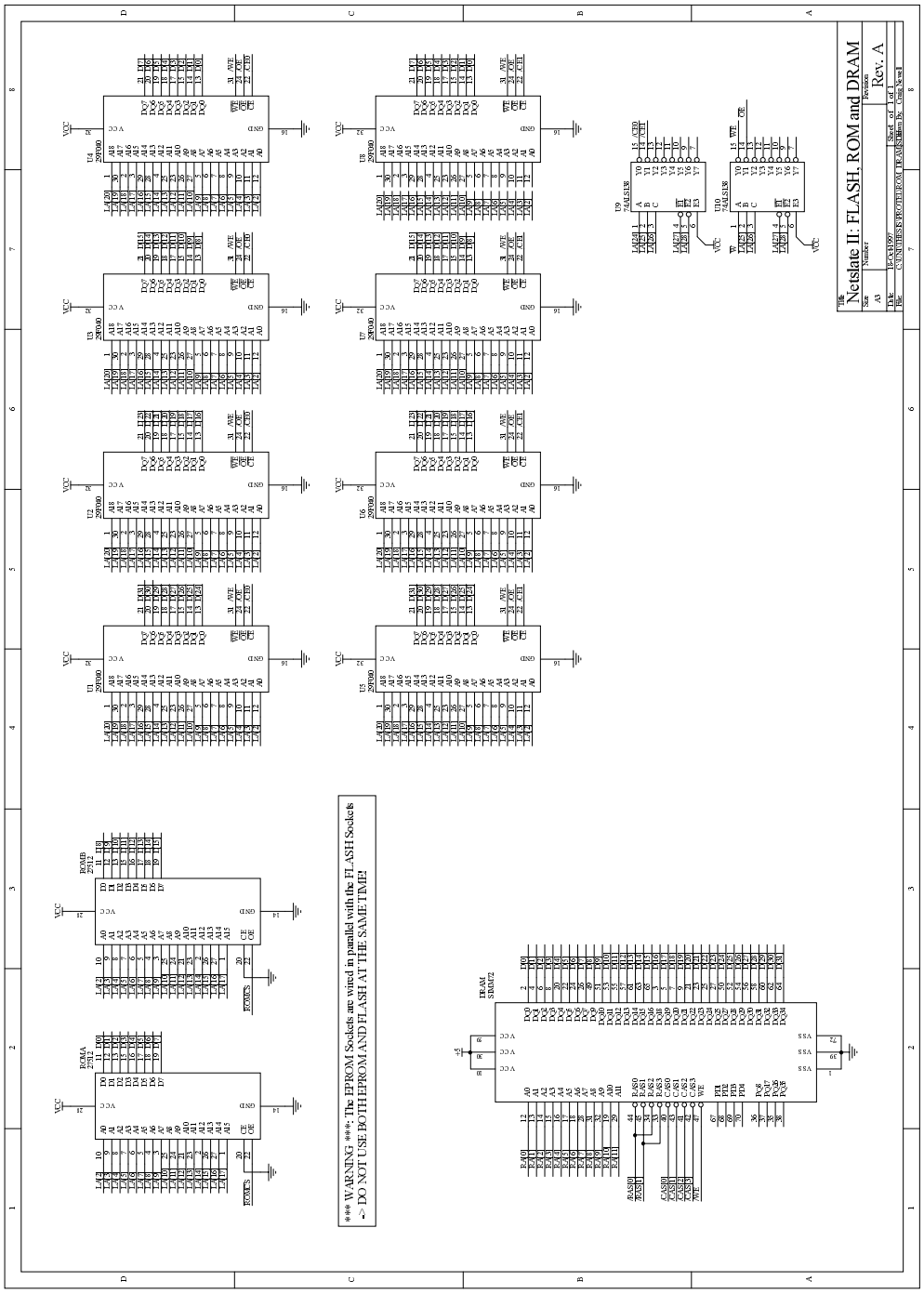

Figure A.4: Netslate Schematic: DRAM and ROMS

Appendix B

An IrDA® Interface

The IrDA® compatible interface on the Netslate is designed to provide the Netslate with a cheap easily available wireless link without a lot of effort in design and implementation. Only the physical layer specification is used as the Point to Point Protocol (PPP) is used to give the Netslate a wireless link to the Internet via router software run on a laptop computer. The laptop runs standard PPP software on it's IrDA® port and routes from the IrDA® to the ethernet and thus Internet.

B.1 What is IrDA® ?

The IrDA® family of specifications defines a method of communicating between computers and peripherals across a wireless infrared link over a short distance of up to 1 meter. The family of specifications define a physical link layer (how the pulses of infrared are to be generated) and a series of communication protocols to be used depending upon the data to be transferred.

The necessary IrDA® hardware to implement physical link layer is built in to most modern UARTs and custom multi-I/O chip sets. However, the IrDA® hardware is simple and easily designed and built with standard logic techniques.

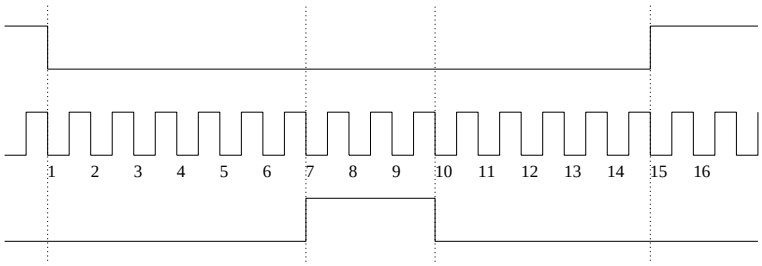

Figure B.1: IrDA® 1.0 IR Encoding

The physical specification[21] defines 3 methods of transmission depending upon the bit rate. At 1200 - 115200 bits/s, a IR pulse of 3/16 th of bit time is generated for each low bit in the serial data stream. From 0.5-1.15 Mbits/s the pulse is 1/4 of the bit time and at 4 Mbits/s the pulses are to be generated according to 4PPM. Figure B.1 shows this encoding at in the 1200 - 115200 bit/s range. This is what is implemented in the Netslate.

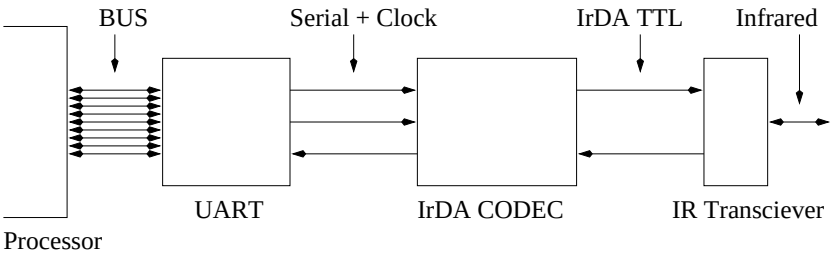

Figure B.2: Basic IrDA® Configuration

B.2 Netslate Implementation

The basic hardware implementation of the IrDA® Specification is shown in Figure B.2 consisting of an IR Transceiver, an IrDA Codec and a standard UART. This was followed for the Netslate II with the details given below.

B.2.1 IR Transceiver

The IR transceiver's purpose is to transform the IR signal to and from a TTL logic signal. This transceiver can be constructed from discrete IR LED's and PIN receiver diodes or obtained as a single integrated module. As there is many integrated modules available (such as Novalog, HP, etc.), this was the chosen alternative for the Netslate II.

The IR Transceiver used was the Novalog SIRComm™ MiniSIR™ 115.2kbps IrDA® Transceiver module[19] which was obtained as free samples courtesy of Novalog Inc.

B.2.2 IrDA® 1.0 Codec

The current trend is to incorporate the IrDA® Codec (Encoder/Decoder) within the UART or microprocessor (eg. TL16PIR552, ST16C650 UARTs with codecs or MPC821 processor with internal codec). However, there is a number of standalone codec's on the market from a number of vendors such as the TIR1000[22], PLX1000, TOIM3000, ST84C01CF8 and others (often relabeled).

Unfortunately, these are not easily available as samples in Australia so I implemented my own design in a Altera 7032 EPLD for 1200 to 115.2 kbps standard. Initially, this was a simple counter based design using edge triggering. This worked fine for the encoding but the decoding would only work at night. During the day, the general background infrared light was causing an overwhelming noise level.

A bit of research on the matter turned up an application note[18] giving design hints for IrDA® codec designers. This suggested a number things including not to edge trigger on the codec inputs. Thus, I redesigned the codec using a couple of simple state machines and oversampling of the inputs. This almost totally removed all noise from the decoder allowing error free use during the daytime (so the constant warnings about edge triggering in the basic digital design subject are right!).

The only other problem was that of feedback inside the IR transceiver. An initial attempt to remove this in software proved unreliable and very slow. Hence, a simple disable signal was added to the decoder to disable decoding when the encoder is producing any output. This works perfectly, with no feedback and no observable turn-around delay.

The following three sections are my final state machine based design incorporating oversampling of the inputs for noise reduction and IR loopback prevention. The performance of this implementation (with the MiniSIR™ transceiver) is very good with the receive error rates lower than that in a IBM 760CD laptop and with no observed transmitted errors.

B.2.2.1 IrDA® Codec Implementation

```
--
-- IrDA -
--
--  This implements an IrDA version 1.0 (max. 115200 baud)
--  Encoder/Decoder.  The encoder and decoder implementations
--  are in encode.tdf and decode.tdf.  This implementation
--  disables the decoder when the encoder is transmitting
--  to remove the IR feedback and uses oversampling to try
--  and reduce the noise in the reciever.
--
--  Craig Newell, CraigN@cheque.uq.edu.au       Sept. 1997
--
TITLE "IrDA 1.0 Encoder/Decoder";

-- Import the encoder and decoder
INCLUDE "encode";
INCLUDE "decode";

SUBDESIGN irda (
        16xclk:         INPUT;  -- 16x buad rate clock
        HSclk:          INPUT;  -- 8MHz high speed clock
        Reset:          INPUT;  -- Active-High Reset
        IR_rxd:         INPUT;  -- Active-Low IR Recieve signal
        Uart_txd:       INPUT;  -- Standard UART TX data signal
        IR_txd:         OUTPUT; -- Active-High IR Transmit signal
        Uart_rxd:       OUTPUT; -- Standard UART RX data signal
)
VARIABLE
        enc: encode;
        dec: decode;
BEGIN
        -- Wire up the encoder
        enc.16xclk = 16xclk;
        enc.HSclk = HSclk;
        enc.Reset = Reset;
        enc.Uart_txd = Uart_txd;
        IR_txd = enc.IR_txd;

        -- Wire up the decoder
        dec.Disable = enc.TX_busy;
        dec.16xclk = 16xclk;
        dec.HSclk = HSclk;
        dec.Reset = Reset;
        dec.IR_rxd = IR_rxd;
        Uart_rxd = dec.Uart_rxd;
END;
```

B.2.2.2 IrDA® Encoder Implementation

```
--
-- encode -
--
--  This is a state machine based IrDA 1.0 encoder that
--  produces a fixed 1.75us IR pulse for power saving
--  at low baud rates.
--
--  Craig Newell, CraigN@cheque.uq.edu.au       Sept, 1997
--
SUBDESIGN encode
(
        16xclk:         INPUT;  -- 16x baud rate clock
        HSclk:          INPUT;  -- 8MHz high-speed clock
        Reset:          INPUT;  -- Active-High reset
        Uart_txd:       INPUT;  -- Transmit data from UART
        IR_txd:         OUTPUT; -- Positive IR pulses
        TX_busy:        OUTPUT; -- Active when transmitting
)
VARIABLE
        sEncode: MACHINE WITH STATES (
```

```
                enc0, enc1, enc2, enc3, enc4, enc5, enc6, enc7, enc8,
                enc9, enc10, enc11, enc12, enc13, enc14, enc15 );
        sBit: MACHINE WITH STATES (
                zero, send0, send1, send2, send3, send4, send5, send6,
                send7, send8, send9, send10, send11, send12, send13 );
        sendbit: NODE;
BEGIN
        --
        -- SBit -
        --     The sBit state machine generates a 14*0.125us =
        --     1.75us pulse when the sendbit node becomes active.
        --
        sBit.clk = HSclk;
        sBit.reset = Reset;

        TABLE
        sBit,   sendbit => sBit,        IR_txd;
        zero,   0       => zero,        0;
        zero,   1       => send0,       0;
        send0,  X       => send1,       1;
        send1,  X       => send2,       1;
        send2,  X       => send3,       1;
        send3,  X       => send4,       1;
        send4,  X       => send5,       1;
        send5,  X       => send6,       1;
        send6,  X       => send7,       1;
        send7,  X       => send8,       1;
        send8,  X       => send9,       1;
        send9,  X       => send10,      1;
        send10, X       => send11,      1;
        send11, X       => send12,      1;
        send12, X       => send13,      1;
        send13, X       => zero,        1;
        END TABLE;

        --
        -- sEncode -
        --  This state machine generates the timing for the
        --  IR pulse from the falling edge of the serial
        --  data and the 16x clock.
        --
        sEncode.clk = 16xclk;
        sEncode.reset = Reset;

        TABLE
        sEncode, Uart_txd => sEncode,   sendbit,       TX_busy;
        enc0,    1        => enc0,      0,             0;
        enc0,    0        => enc1,      0,             0;
        enc1,    1        => enc0,      0,             1;
        enc1,    0        => enc2,      0,             1;
        enc2,    1        => enc0,      0,             1;
        enc2,    0        => enc3,      0,             1;
        enc3,    X        => enc4,      0,             1;
        enc4,    X        => enc5,      1,             1;
        enc5,    X        => enc6,      0,             1;
        enc6,    X        => enc7,      0,             1;
        enc7,    X        => enc8,      0,             1;
        enc8,    X        => enc9,      0,             1;
        enc9,    X        => enc10,     0,             1;
        enc10,   X        => enc11,     0,             1;
        enc11,   X        => enc12,     0,             1;
        enc12,   X        => enc13,     0,             1;
        enc13,   X        => enc14,     0,             1;
        enc14,   X        => enc15,     0,             1;
        enc15,   X        => enc0,      0,             1;
        END TABLE;
END;
```

B.2.2.3 IrDA® Decoder Implementation

```
--
-- decode -
--
--  This is a state machine based IrDA 1.0 decoder with
--  a 1.25us miniumum pulse with detection and a decode
--  disable input to disable the decoder when
--  transmitting.
--
--  Craig Newell, CraigN@cheque.uq.edu.au       Sept, 1997
--
```

```
SUBDESIGN decode
(
        16xclk:         INPUT;  -- 16x buad rate clock
        HSclk:          INPUT;  -- 8 MHz Clock for oversampling
        Reset:          INPUT;  -- Active-High reset
        Disable:        INPUT;  -- Disable the detector
        IR_rxd:         INPUT;  -- Active-Low IR signal
        UART_rxd:       OUTPUT; -- Recieved data to UART
)
VARIABLE
        sDecode: MACHINE WITH STATES (
                dec0, dec1, dec2, dec3, dec4, dec5, dec6, dec7,
                dec8, dec9, dec10, dec11, dec12, dec13, dec14, dec15 );
        sDetect: MACHINE WITH STATES (
                noise, test0, test1, test2, test3, test4, test5,
                test6, test7, test8, gotit);
        bit, out, ack:  NODE;
BEGIN

        --
        -- SDetect -
        --
        --  Detect a pulse
        --
        sDetect.clk = HSclk;
        sDetect.reset = Reset;

        TABLE
        sDetect, IR_rxd, ack,   Disable => sDetect, bit;
        noise,   1,      X,     X       => noise,   0;
        noise,   0,      X,     0       => test0,   0;
        test0,   1,      X,     X       => noise,   0;
        test0,   0,      X,     1       => noise,   0;
        test0,   0,      X,     0       => test1,   0;
        test1,   1,      X,     X       => noise,   0;
        test1,   0,      X,     1       => noise,   0;
        test1,   0,      X,     0       => test2,   0;
        test2,   1,      X,     X       => noise,   0;
        test2,   0,      X,     1       => noise,   0;
        test2,   0,      X,     0       => test3,   0;
        test3,   1,      X,     X       => noise,   0;
        test3,   0,      X,     1       => noise,   0;
        test3,   0,      X,     0       => test4,   0;
        test4,   1,      X,     X       => noise,   0;
        test4,   0,      X,     1       => noise,   0;
        test4,   0,      X,     0       => test5,   0;
        test5,   1,      X,     X       => noise,   0;
        test5,   0,      X,     1       => noise,   0;
        test5,   0,      X,     0       => test6,   0;
        test6,   1,      X,     X       => noise,   0;
        test6,   0,      X,     1       => noise,   0;
        test6,   0,      X,     0       => test7,   0;
        test7,   1,      X,     X       => noise,   0;
        test7,   0,      X,     1       => noise,   0;
        test7,   0,      X,     0       => test8,   0;
        test8,   1,      X,     X       => noise,   0;
        test8,   0,      X,     1       => noise,   0;
        test8,   0,      X,     0       => gotit,   0;
        gotit,   X,      0,     0       => gotit,   1;
        gotit,   X,      0,     1       => noise,   1;
        gotit,   X,      1,     X       => noise,   1;
        END TABLE;

        --
        -- sDecode -
        --
        -- Create the 16 16xclock long decoded pulse for the UART
        --
        sDecode.clk = 16xclk;
        sDecode.reset = Reset;

        TABLE
        sDecode, bit     => sDecode, out, ack;
        dec0,    0       => dec0,    0,   0;
        dec0,    1       => dec1,    0,   0;
        dec1,    X       => dec2,    0,   0;
        dec2,    X       => dec3,    0,   0;
        dec3,    X       => dec4,    0,   0;
        dec4,    X       => dec5,    0,   0;
        dec5,    X       => dec6,    0,   0;
        dec6,    X       => dec7,    0,   0;
```

```
        dec7,    X       => dec8,    0,   0;
        dec8,    X       => dec9,    1,   0;
        dec9,    X       => dec10,   1,   0;
        dec10,   X       => dec11,   1,   1;
        dec11,   X       => dec12,   1,   0;
        dec12,   X       => dec13,   1,   0;
        dec13,   X       => dec14,   1,   0;
        dec14,   X       => dec15,   1,   0;
        dec15,   X       => dec0,    1,   0;
        END TABLE;

        -- Finally generate the output for the UART
        UART_rxd = NOT (out OR bit);
END;
```