Netslate
a
Handheld Web Browser

Ben Williamson

Supervisor: Dr Mark F. Schulz

The Dean
Faculty of Engineering
University of Queensland
St Lucia 4072

Ben Williamson
Mayfield Road
Samford 4520
(07) 3289-4367

1 November 1996

Dear Sir,

In partial fulfilment of the requirements of the Bachelor of Engineering (Computer Systems) Honours Degree, I hereby submit for your consideration this thesis entitled

“Netslate - A Handheld Web Browser”

Yours faithfully,

Ben Williamson.

Abstract

Demand for cheap, easy access to global networks of information is growing rapidly. A handheld Web browser is a portable computing device which allows users to explore the Internet without the need for a PC. The Netslate is a prototype of such a device, integrating processing, communications and a graphical user interface into a portable unit. A library of support software demonstrates the capabilities of the device. Further work is needed to develop operating system and applications software for the Netslate.

Acknowledgements

Thanks go to Chris Stott formerly of ART for supplying the sample ARM7500s which made this project possible, Art Sobel and Jim Jacot of VLSI for supplying the JumpStart development tools and their invaluable documentation, Richard Earnshaw of ARM for his assistance with the GNU tools, and Marco Graziano of Teknema for his assistance with the uC/OS kernel.

Special thanks go to Ian Field for his assistance in preparing this document, Len Payne for sharing his down-to-earth engineering experience, and Mark Schulz for pointing me in the most difficult direction at every opportunity.

Table of Contents

Chapter 1 - Handheld Computing and the Internet

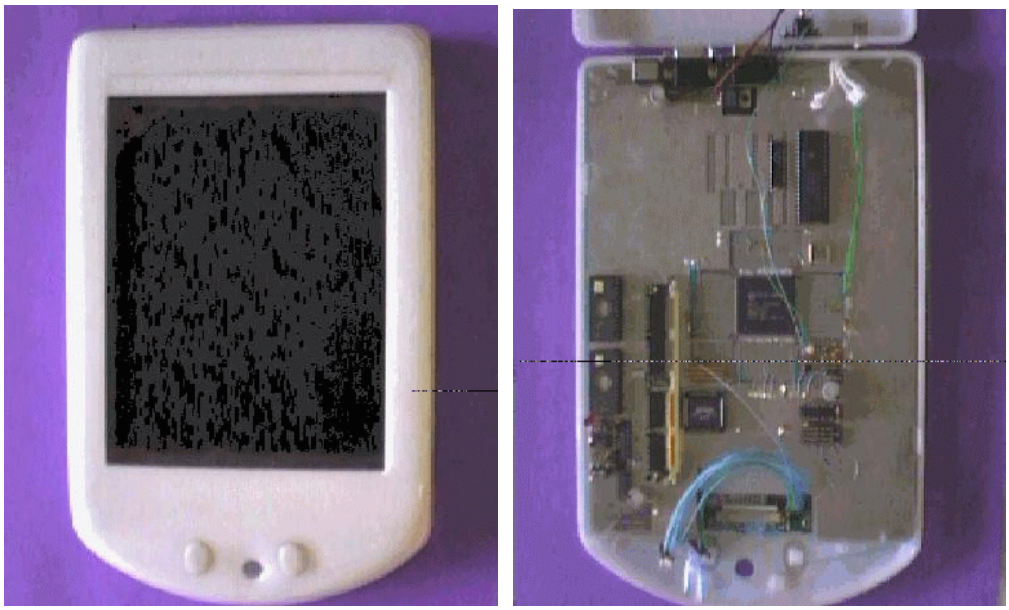
Introduction

Radio, television, magazines, newspapers and telephony are all migrating to the Internet, and for very good reasons. Printed media are by nature portable, and handheld radios, televisions and especially telephones are now part of our lives. People are finding more reasons to need access to more information in more places. It is inevitable, then, that handheld access to the Internet and all of the precious information it offers will soon be in high demand.

The most prominent feature in any explanation of why the Internet has experienced such explosive growth in recent times has to be the advent of the World Wide Web. Without warning, the Internet transformed from a notoriously complicated hangout for computer nerds and scientists, into a point-and-click source of information on absolutely everything.

Emerging technologies in multimedia, portable computing and internetworking point to an exciting future for computer systems engineering. There is great competition to build systems that allow non-expert users to access Internet content conveniently and cheaply. This project aims to develop a working prototype of a handheld Web browser, a portable device that connects people to the Internet.

The Netslate is an entry in the race to build a portable computing and communications tool for the masses. It can be compared to the Network Computer concept, which promises to bring the Web into our living rooms and offices for $500 a piece. It also has links with currently available Personal Digital Assistant products, which primarily function as digital diaries but are now clambering to provide all the attractions of handheld Internet connectivity.



(a) (b)

Fig. 1 - The Netslate prototype hardware: (a) external; (b) internal.

Project Goals and Results

There are two goals discussed in this thesis document. The first is a long term goal of a handheld Web browser product that is fully supported with operating system and application software, ready for production. The more immediate goal of the Netslate project is to build prototype hardware and develop low level software that can support this greater goal, to provide a starting point and direction for further development. Hence the motivations for the design decisions in this project come from a goal more distant than the demonstration day at the end of the project.

The tangible results of this project are:

- the Netslate prototype hardware packaged in a moulded case with display, communications interface etc, pictured in figure 1;
- the `nslib` development library of low level functions to drive the Netslate hardware, with a number of demonstration programs;
- this report, providing direction for further development of the Netslate.

Overview of this Report

The remainder of this chapter examines the nature of interactive computing and arrives at the requirements of the Netslate prototype. Chapter 2 examines current products and projects in the field of handheld computing devices, and looks at software and hardware technologies which may form the basis of new generations of these systems.

Chapter 3 details the development of the Netslate prototype hardware, examining the design decisions and features of the prototype at each stage. Chapter 4 describes the software development and testing of the Netslate prototype, including an overview of the `nslib` library and the GNU development environment.

Chapter 5 discusses future developments of software and hardware towards the long term goal of the Netslate, and beyond to examine the directions in which this field of technology may evolve.

Computing and its Costs

There are three central activities involved in computing: the processing, storage and communication of information. All computer systems exhibit each of these to some degree. The evolution of computing revolves around reducing every kind of cost associated with these activities.

The three fundamental costs of computing are time, space and energy. We want to make our computer systems as fast, as small and as power efficient as possible. This results in cheap, convenient, useful devices with value to people.

It is often possible to make tradeoffs between the different costs, such as choosing to build a faster computer which consumes more power. But real advancements in technology allow us to reduce all of the costs without a tradeoff. Using transistors instead of valves, reducing the minimum feature

size of a silicon manufacturing process, advancing from CISC to RISC architectures, these are all ways that we have managed to make faster computers in smaller packages which consume less power and cost less.

Personal vs Embedded vs Ubiquitous Computing

Personal computers are currently the most obvious computer systems in our day-to-day experience. A PC occupies a space on a desk which you sit at to use the computer. General purpose user input and display devices perform the same functions regardless of whether you are writing a thesis report or battling aliens from Mars.

The PC market is wildly performance driven as new and more bloated versions of programs and operating systems demand more grunt from the hardware to do the same jobs. While the time cost of personal computing has improved by orders of magnitude (if somewhat offset by inefficient software), the space and energy costs have barely changed at all. A PC still comes in the same size box with the same size power supply it did fifteen years ago, and the price tag has not changed significantly in that time.

Embedded computer systems, while less conspicuous than desktop PCs, are certainly recognisable as individual devices. CD players, VCRs, modern microwave ovens and air conditioners, mobile phones, these are all examples of appliances incorporating embedded computers. The computing power embedded in the system allows the user more flexible and interactive control over the appliance.

There are still more examples which fall into the category of embedded control systems, such as vehicle engine management computers. These systems differ in that they are not interactive, that is the user does not have a dialogue with them. They are buried deep within some piece of equipment to make it behave more intelligently. These non-interactive embedded

devices are less relevant to our discussion because they lack this direct interaction with humans.

The market for embedded systems is often driven by physical size. Measures of integer performance are fairly irrelevant when purchasing a mobile phone, assuming that the response time for searching the stored phone numbers is unnoticeable. Embedded systems have well defined tasks, eliminating the need to upgrade the processor in your mobile to the fastest one on the market. While total power consumption of many battery operated products in which embedded systems appear is certainly important, the power drawn by the CPU is often a negligible component of that total. The lion's share of current consumed by a mobile phone is that used to power the transceiver.

Ubiquitous computing is most conspicuous by its absence. If, as some advocates suggest, ubiquitous computing devices will be so small that you won't even notice them, it is tempting to wonder how we will ever know when they have arrived. Taken more literally, ubiquitous computing means devices which are not noticed simply because they are absolutely everywhere.

While a world where every surface in a room is smeared in computing power may seem a little hard to imagine right now, the potential benefits of this availability of computing are enormous. When computers come by the billion, floating in tins of paint to be sprayed onto any object you care to make more intelligent or interactive, the information revolution will surely be over. The purpose of presenting this idea here is to convince the reader that there is a point to making computers smaller.

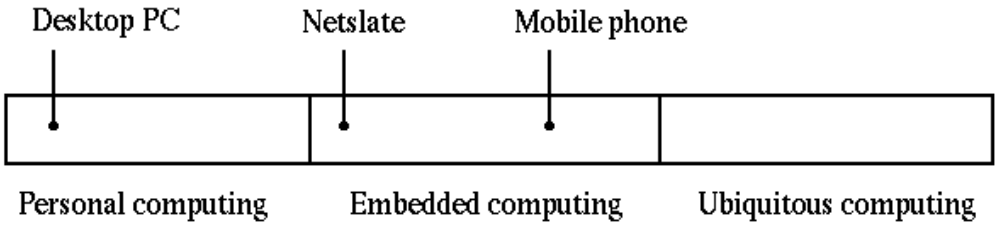


Fig. 2 - The spectrum of computing assimilation.

Figure 2 depicts the spectrum of the assimilation of computing into our lives. At the left is the typical desktop PC, which demands the user to adapt to it. On the right is ubiquitous computing, with its promise of an omnipresent intelligent environment. In the middle is embedded computing, which adds interactivity to special purpose devices.

The Netslate is aimed somewhere left of centre on this spectrum, within the range of embedded systems but leaning towards more general purpose computing. Its single purpose is to connect people to the Internet, but this still allows a certain latitude in the nature of the applications software that it will run. It is a handheld device, not a wearable object that might place the project further towards the ubiquitous end of the scale.

User Interaction

The Netslate is not intended to be perfectly suited to every possible kind of interaction with the Internet. Rather it is designed to fulfil the basic needs of Internet access for the masses. An average user spends much more time looking for information on the Web than adding to it.

This leads to the first major difference between the Netslate and the PCs on which we currently run Web browsers - the lack of a keyboard. The requirement for text entry on the Netslate is limited to pages which require search keywords or form fill-ins. We examine some ways of performing these tasks without a keyboard in the final chapter.

User input on the Netslate is by some pointing mechanism. For the prototype built in this project, the input device is a trackball. A more natural method of input might be a touch screen or pen input.

The display device on the Netslate prototype is a colour LCD panel similar to those in laptop computers. Colour flat panel displays are continually improving in cost and performance, and present a much more attractive graphical rendering of Web pages than do cheaper monochrome displays. LCD panels have slower response times than CRT type monitors thus blurring animations on laptop Pcs, but this is less of a problem for a Web browser where viewing Web pages is the primary task.

Hardware Requirements

As we have discussed, the basic functional requirement of the Netslate is to support a Web browser application comparable to those running on PCs today, with minimum cost. Because the Netslate does not need to be capable of running all of the various kinds of applications that run on a PC, we can eliminate a number of expensive components.

Hard disk storage is not required by the Netslate design. Because the Netslate is only used while connected to the network, any extra software that is not stored in ROM can be downloaded as it is needed. The user does not need to use the Netslate to store documents or save copies of Web pages, or install various applications. All permanent storage is elsewhere on the network. Floppy disk drives are unnecessary for similar reasons.

In the spirit of all embedded devices, performance upgrades are unnecessary. Once the system is capable of performing its particular tasks satisfactorily, there is no point in upgrading the CPU or memory to make the system faster. As the Web takes on more intensive multimedia content requiring faster processing, old systems will become out of date and be

replaced by newer more capable ones. The cost of replacing a device is far less significant than that of a PC. Eliminating general purpose expansion buses, CPU and memory sockets and the like from the design, we further lower the cost and size of the Netslate.

Because the Netslate is a handheld device, it must be powered by batteries within the unit. The simpler we can make the electronics, the longer the batteries will last on a single charge. Battery technology improves at a far slower rate than computing, so low power consumption is important. Keeping power dissipation from the board as low as possible also removes the need for a cooling fan. This further reduces the size of the product and eliminates the annoying noise that emanates from laptop computers and PCs.

Networking and Communications

Obviously a portable device that draws on information from the Internet will need some wireless means of connection to the global network. Building a fast reliable wireless digital communications network is a sufficiently involved topic to make up a whole separate thesis, and this is indeed another rapidly advancing field of research. Accordingly, for the purposes of the Netslate project the question of how the device connects to the Internet is considered someone else's problem.

The prototype Netslate hardware implements a high speed serial port which is sufficient for development and demonstration purposes. In the final chapter some practical wireless solutions and the directions in which they are evolving are discussed.

Regardless of the exact nature of the communications medium used, a product which connects to the Web will need to speak the language of the Internet. Of course this implies implementing a TCP/IP protocol stack, a matter which is also discussed in the final chapter. The need for this

protocol support does not have any direct impact on the hardware design of the system.

Summary

We have looked at the motivation behind building a handheld Web browser, and discussed the basic design requirements of such a device. By narrowing the range of applications run by personal computers down to those which are central to benefiting from the Internet, we have reached an outline of a simpler, lower cost product which will appeal to the masses.

The aim of the Netslate is to integrate a point-and-click graphical interface with sufficient processing and communications capability into a portable unit. The Netslate concept takes on the role of Web browser programs running on PCs today, in a device that is closer to a special purpose embedded system than a PC.

The next chapter compares the Netslate concept to a number of other products and projects in handheld computing, and examines some currently available technologies that are competing for application in this field.

Chapter 2 - Handheld Devices and Technologies

The Netslate concept described in the first chapter shares some common ground with a number of existing products and projects. In this chapter we survey some prominent handheld computing developments, and the technologies which drive them.

Apple Newton PDA

Fig. 3 - The Apple Newton MessagePad PDA.

The Apple Newton [1] is almost certainly the most prominent of Personal Digital Assistant (PDA) products in recent times. The Newton, shown in figure 3, is roughly 200mm x 100mm x 30mm in size with a monochrome touch sensitive LCD screen. In contrast to the Netslate, its central purpose is to store information.

PDAs evolved out of the huge range of electronic organisers and diaries that have flooded the market for years. These devices all use battery backed static memories to retain personal information belonging to the user. This information is typically divided into phone and address details, calendar, schedules, memos and reminders etc. Like a paper diary, the only information you get out is the information you put in.

The designers of the Newton wanted to do something fundamentally new with the way we organise our personal information. They conducted experiments in which subjects were given featureless rectangular objects and asked what they would like those objects to be able to do for them if they were computing devices. A large number of the subjects began

scribbling on these blank objects, writing information like notes and reminders.

The fact that the subjects in these experiments had a tendency to add information to a featureless "computing" object is hardly surprising. Word processing and diary management are two of the familiar functions that we have come to expect from our computers and electronic organisers. The author is willing to bet that not too many of the subjects saw these blank rectangles as windows onto diverse sources of information from around the globe, which they could browse at leisure.

With this research data in hand, the designers went on to build a device that could accomplish old tasks with new technology. The Newton was successful in pushing advances in handwriting recognition software and the development of the ARM610 CPU (which is discussed later in this chapter). While successful as a product, the Newton did not bring about a global revolution in the way people store reminders and memos.

To its credit, the original Newton did recognise that electronic communications would become an important component of portable computing. Unfortunately even todays Newtons without expansion hardware have a maximum communications radius of one metre. This is fine for swapping digital business cards, if you can find someone else with a Newton to put yours next to.

Of course the Newton range has evolved somewhat since its first release. The new Newton operating system (Newton 2.0) supports networking applications with a TCP/IP protocol stack, allowing it to run a basic Web browser and email interface. The display limitations of the Newton platform limit the Web browser to static pages with four-greyscale graphics at low resolution.

The Apple Newton is still marketed with organiser functions first and communications second on the feature list. It will be interesting to see if the Newton product experiences an inversion of these priorities in future versions.

Motorola Marco and Envoy Communicators

Fig. 4 - The Motorola Marco.

Motorola manufactures two interesting products that have a lot in common with the Apple Newton. In fact one of them, the Marco [2] shown in figure 4, is based on the Newton platform. The other, the Envoy [3] in figure 5, is Motorola's own architecture running the Magic Cap operating system. Both have touch sensitive screens, handwriting recognition software and a limited amount of static memory.

What sets these two apart from the Newton is that they each have integrated cellular radio modems. This means that they can connect to an Internet Service Provider without extra hardware. As would be expected, the communications features of these two PDAs receive top billing over diary functions on the feature list.

Fig. 5 - The Motorola Envoy.

The Marco runs all of the software available for the Newton, and the Envoy runs a similar range of software for the Magic Cap platform. Both run versions of the Web browser and email applications for the Newton mentioned above. While the bandwidth and availability of current cellular services are less than ideal for regular Web browsing sessions, these products seem to be ahead of the Newton in that Motorola has made communications the priority.

Xerox ParcTab

The ParcTab [4] is part of an ongoing project at Xerox's Palo Alto Research Centre (PARC). The ParcTab mobile hardware shown in figure 6, or Tab for short, is a palm-sized device with a touch sensitive monochrome display and an InfraRed transceiver, run by an 8051-family microcontroller. The project is an experiment exploring concepts in ubiquitous computing.

A number of Tabs have been manufactured and distributed to employees in the research centre. The workers can use the devices anywhere within the IR communications cells around the centre to organise collaborations and communicate with electronic mail messages.

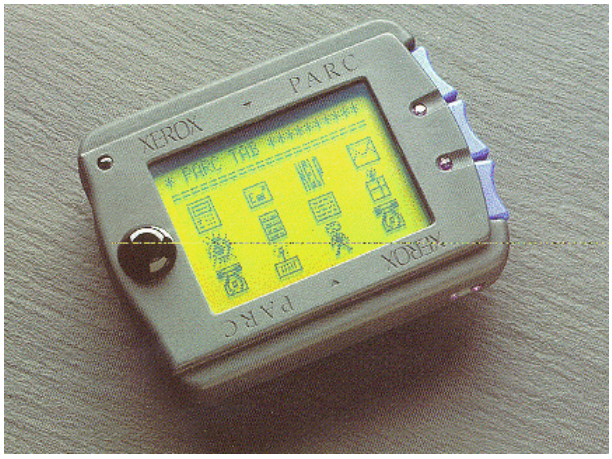


Fig. 6 - The Xerox ParcTab.

Of course the Tab is far too small to function as a useful Web browser. Users need to scroll the display back and forth just to read their email. But the project has shown that even a device of this simplicity and small size can be useful if equipped with networked communications capability.

Berkeley InfoPad

The InfoPad project [5] at the University of California at Berkeley is another project aimed at providing ubiquitous information access. While the InfoPad hardware bears some similarity to the handheld devices we have already discussed, the most interesting feature of the project is the way the computing for the terminal is distributed.

The InfoPad terminal hardware, shown in figure 7, is based on an ARM610, ASIC parts, a monochrome touch screen and a radio modem. It also

contains sound generation and sampling hardware. The design basically implements a general purpose multimedia terminal which is capable of displaying text, graphics and compressed video, playing and sampling audio, and capturing pen input.

The processing capability of the portable terminal is used exclusively to control the communications link and coordinate the display and audio output, and the audio and pen input to and from the terminal. The terminal does not participate in executing any of the applications it displays, as these are all running on other machines on the network.

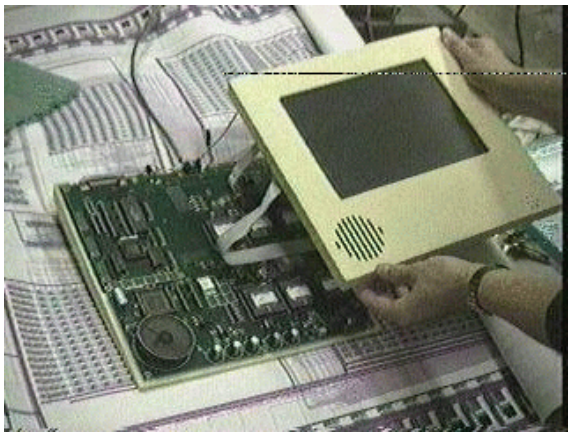


Fig. 7 - The Berkeley InfoPad.

In fact the applications are all standard or modified X windows programs for Unix workstations. Apart from the servers running the applications, intermediaries between these hosts and the terminal are needed to make the mobile multimedia terminal appear as a fixed X windows terminal to the applications.

The result of this approach is that the only data which passes over the radio link is graphics, video, audio and pen data. These kinds of data are error resistant, so errors in the data streams only cause glitches in the display or sound. In effect the InfoPad system is a portable wireless X windows terminal which depends entirely on fast server machines on the network.

While the InfoPad introduces some novel techniques which have been successful in making the terminal useable over an error prone link, this approach does have some disadvantages. The first and most obvious is the total reliance on expensive servers for every application and every moment of interaction with the terminal.

The justification for moving all of the computing onto the network is to reduce power consumption on the terminal and to free the applications from the terminal's performance limitations. However as the (time, space and energy) costs of computing continue to fall, this tradeoff may turn out to be inappropriate.

One of the problems with the InfoPad design is the latency of response from pen input to digital ink appearing on the display. This is due to the large number of layers between the input device up to the application on the network and then back down to the display device.

Large amounts of computing power and fast network connections have been thrown at this problem to make the response latency less noticeable. While this works between rooms at Berkeley, it will not work across cities or countries when the server is more distant on a larger network. End users of cheap terminals can not afford the expensive server for the next room to keep latency low.

The lesson here is to keep the mechanisms for user feedback in interactive systems close to the user. Making the handheld device a terminal for applications running elsewhere on the network violates the principle. It also greatly increases the amount of computing required to achieve the same task, by placing too many layers of computation between the user and the application.

A final doubtful point is whether the X windows system will ever really be suitable for the general population to use for running Web browsers and the like. The cost of running networks of X workstations (wireless or otherwise) and the complexity of administrating Unix hosts to run applications for them is certainly prohibitive for now.

The Network Computer

It is clear from examining each of the above systems that good software design is at least as important to the success of a product or project as good hardware. We now diverge temporarily from our discussion of handheld hardware issues to examine a new reference standard for consumer computing called the Network Computer or NC [6].

The NC is not a product from a particular manufacturer. Rather, it is a hardware and software reference design that will be licensed and implemented by various manufacturers. The reference design is owned and licensed by Oracle, the world's second largest software company. At the time of writing, no commercially available NC implementations have yet been released.

The central theme of the NC concept is to simplify computing to make it available to everyone. The complexity of the PC is the barrier making it too daunting and too expensive for many. The NC, while not specifically for handheld devices, is a design that lends itself to providing a suitably simple and accessible operating environment for a handheld Web browser.

As implied by the name, the network is a fundamental part of the NC architecture. The NC loads its system and application software from the network as it boots up, eliminating the need for users to periodically upgrade or maintain the software on their system. This also requires no local hard disk storage.

The NC operating environment is centred around the Web browser. Working from the assumption that browsing Web pages is a task simple enough for anyone, this becomes the mechanism by which the user navigates the system to find applications, tools, network files, help documents etc.

The hardware required to support the reference NC design is very similar to the prototype hardware of the Netslate project. The NC is not specifically a handheld product, but rather a range of products working within the same operating environment. A commercial implementation of a handheld NC is almost inevitable.

Java

Applications for the NC are written in Java [7], a platform independent language from Sun Labs which is designed for distributing programs across the Internet. Java is already used to build interactive Web pages. The Java programs embedded in these pages are interpreted or compiled by the Web browser software on the client's machine.

Maintaining applications and tools as links to Java programs on the Web further simplifies the user's task, as upgrading and maintaining these add-on applications also becomes unnecessary. This will mean that writers of Java applications will have to carefully consider the effects of changing their programs. Upgrades to applications will have to be made in smooth and logical steps. The provision of software may transform from a product market to a service market.

There is a certain amount of performance overhead incurred by running interpreted Java programs. This slowdown can be reduced by compiling modules of applications to the machine's native executable code as required, using a technique called Just In Time (JIT) compilation.

The ARM Processor Architecture

The processor used in the first implementations of the NC design was the ARM7500 [8], the same part as used in the Netslate prototype. The ARM architecture [9] is a novel range of low cost CPUs designed by Advanced RISC Machines in the UK. The specifics of the ARM7500 are discussed in the following chapter.

The very first ARM parts, the ARM2 and ARM3, were the basis of the first Acorn computers. Advanced RISC Machines was spun off from Acorn with the help of Apple Computer, to develop the ARM610 for use in the Newton. The ARM7100 and ARM7500 are aimed at PDA and multimedia devices. Digital Semiconductor licensed the rights to customise the ARM core architecture, and have recently gone into production of the StrongARM.

The StrongARM range is fabricated on the same process as Digital's Alpha series of very high performance 64-bit processors. While the core of the ARM7500 is capable of 40MHz clock rates, the StrongARM extends up to clock rates of 235MHz. This is in addition to other architectural enhancements. Even while delivering around 260 MIPS (Dhrystone 2.1) of performance, the StrongARM typically dissipates less than one Watt.

For their high performance, low cost and low power requirements, the StrongARM parts are likely contenders for future portable Internet products and NC implementations.

Summary

Computing has been evolving from standalone computation to networked interaction. Products originally designed to function as diaries are working towards accessing the Internet. Even small simple devices can be useful if they are a means of communication across a network.

Keeping the user feedback tightly coupled to the user input improves interaction, and this implies some amount of processing on the terminal. Simplicity of software is the key to networked computing for the general population, and simplicity of hardware is the key to making this computing portable. Successful portable devices integrate a necessary level of computing performance with communications and user interfaces appropriate to the application.

In the next chapters we delve into the hardware and software development of the Netslate prototype, a project which aims to apply some of these principles.

Chapter 3 - Hardware Development

We have discussed our motivation for building the Netslate, and examined some relevant products and technologies in the field. Now we turn to the actual implementation of the prototype Netslate. This chapter deals with the hardware design and construction, and the following chapter examines the software development and testing.

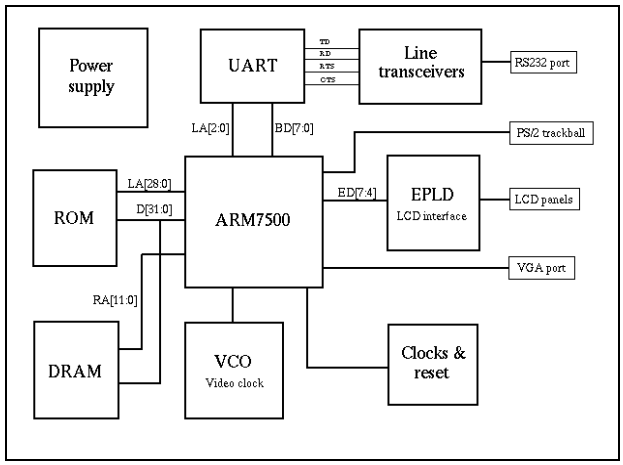


Fig. 8 - The Netslate hardware block diagram.

Figure 8 shows the block diagram of the completed Netslate prototype hardware. Each block is described in the following sections.

The ARM7500 CPU

The ARM processor architecture has already been mentioned in the preceding chapter as a prominent component in several handheld computing products. The distinguishing features of ARM processors are good performance, low power dissipation, small size, high integration and low cost. All of these features are attractive to designers of consumer and embedded products.

The ARM7500 is the CPU chosen for the Netslate. It consists of a cached 32-bit RISC core, a video and sound macrocell, and extensive memory and I/O support. It is this combination of many of the major components needed by the Netslate that makes it ideally suited for this project.

Integer performance of the ARM7500 is roughly equivalent to a 66MHz Intel 486, while dissipating less than one Watt. The efficient design of the CPU core keeps the transistor count low, making for a small, low cost part. The low power consumption also means that a cheap surface mount package is sufficient, without the need for a heatsink or cooling fan.

Perhaps the most novel feature of the ARM7500 is the video and sound macrocell. This provides analog and digital video and sound outputs which may directly drive a CRT display and audio amplifier respectively. Dedicated DMA channels are used to refresh the display and play sound samples directly from main memory. External logic can be added to drive LCD panels and CD quality audio DACs.

Other very useful I/O modules within the ARM7500 include ports for PS/2 style mouse and keyboard, four analog inputs for resistive devices such as joysticks, PCMCIA bus support, flexible I/O cycle timings and a number of internally decoded address spaces with chip select lines. Also included is a multiplexed row/column address bus for direct DRAM interfacing.

The ARM7500 also includes sophisticated clocking and reset control. It is possible to power down the crystal oscillator module and halt the CPU with only static currents being drawn. A number of event sources can then bring the CPU back up cleanly after a delay while the clock stabilises. An intermediate suspend mode allows the clock to the CPU core to be gated off until and interrupt event occurs. This is performed as a single write to a memory-mapped register, which makes it possible to suspend clocking until the next context switch from within an idle task. DMA transfers continue while in suspend mode, so display refresh and sound playback are unaffected.

The Netslate Development Board

From the beginning of the project, it was clear that a flexible way of prototyping the Netslate hardware would be needed. The ARM7500 comes in a 240 pin surface mount quad flat pack and runs at a clock frequency of 32MHz. Obviously conventional socketing and breadboarding techniques were out of the question.

A two layer PCB with etched tracks for all buses and connections was ruled out as being too inflexible and prone to layout mistakes. It was also impossible for a two layer board to support the density of routing required and still provide sufficient power and ground distribution traces. Ideally in a production environment multilayer boards with layers dedicated to power and ground planes would be used to provide a low resistance noise immune power supply. However for the Netslate prototype multilayer PCBs were ruled out as too inflexible and too expensive.

It would have been possible to build the board using wirewrap techniques by printing a PCB which brought all 240 surface mount pins out to 100 mil spaced wirewrap pins. Wirewrap boards are capable of operating at the frequencies required, however a significant amount of board space would be required just for the wirewrap pins, and the protrusion of these pins from the underside of the board would have significantly increased the bulkiness of the Netslate prototype when packaged with the display.

The prototyping technique developed for the Netslate was a hybrid of the above techniques. A two layer PCB was designed with pads for the ARM7500 brought out on short fine traces to small vias, allowing access to all pins on both sides of the board without consuming much board space. Pads for all the anticipated components and connectors as well as some grids of extra pads were laid out on the board. All unused area was filled with power and ground planes on top and bottom respectively.

Insulated single strand wire of the same kind used for wirewrapping was soldered point-to-point directly to the pads and vias to build up the buses and connections required. This allowed a degree of flexibility when connections needed to be altered, and allowed the design to start simple and build up in an incremental fashion. With the wiring running flat against the ground plane the board created an electrically quiet environment without the space overhead of wirewrap pins.

As well as the ARM7500 CPU, provisions were made on the board for ROMs, a 72-pin SIMM, an EPLD, a serial port with transceiver, and connectors for RS232, VGA and PS/2 ports.

Memory

The ARM7500 includes interfacing logic for 16-bit and 32-bit ROMs. A 16-bit ROM configuration using two 27512 8-bit 512kbit ROMs was chosen, as this was readily emulated using the ROM emulators available. As will be described in the next chapter, all Netslate programs copy themselves into DRAM before executing, so the bandwidth limitation of using half-word fetches is not a problem.

This configuration gives a total program space of 128Kbytes, which was ample for all of the code developed throughout the project. If later projects using the Netslate hardware run out of program space, the author recommends either compressing the ROM images and running a self-extracting utility on start-up, or booting over the serial link. The latter is the technique used by commercially available development boards with debug monitors in ROM.

The ROM emulator modules used have access times of around 100ns, and typical EPROMs of up to 200ns. With a 32MHz memory clock this equates to at least six cycles for each ROM access to be safe. A stupid coding error early in the project set the ARM7500's ROM timing register for two cycles

per access, causing all sorts of problems. The ARM7500 resets into 14 cycle ROM timing, and the `nslib` code (described in the next chapter) leaves it that way.

DRAM interfacing is often a problematic area in microprocessor projects, due to the complexity introduced by row/column address multiplexing and cyclic memory refresh requirements. However interfacing the ARM7500 to a 32-bit DRAM module is simply a matter of connecting address, data and RAS/CAS strobe lines, as all of the necessary DRAM interface logic is internal. DRAM refresh continues during and after reset, so no internal registers need be altered on boot up in order to use the DRAM.

A low profile 72-pin SIMM socket is included on the Netslate board, which has been tested with 4 and 8 Mbyte SIMMs. As some SIMM modules are fairly current thirsty and noisy on the supply rails, sufficient decoupling capacitors are required around the SIMM socket. A number of electrolytics and multilayer ceramics have been used around the board for decoupling across the power and ground planes.

Video Display

As mentioned earlier, the ARM7500 provides analog and digital video outputs, fed from dedicated DMA channels. One channel fetches data for the display, while the other fetches data for a hardware cursor mask. (On the Netslate the latter is used to produce a mouse pointer without altering the display buffer.) Both streams of data are serialised and passed through palette lookup tables. The display gets a 256 entry palette while the cursor has a three entry palette. A fourth value in the cursor masks is transparent to the display data beneath it, allowing a shaped cursor.

The Netslate board includes a variable frequency clock source for flexible driving of a number of different types of CRT and LCD displays. This clock source determines the frequency at which pixels are driven out to the

display device. Registers within the video subsystem are programmed with values representing the dimensions of the display and border in pixel units. The clock source is a Voltage Controlled Oscillator (VCO) which is coupled to the ARM7500's internal phase comparator, completing a phase locked loop or PLL. This circuit is able to generate many frequencies from DC to over 90MHz by changing the divisors in the phase comparator. The design of the VCO is similar to that described in Appendix E of the ARM7500 Data Sheets [8].

VGA Monitor Interface

No extra external circuitry is needed to drive a VGA monitor. The VGA connector on the Netslate board was first fed directly from the ROUT (red), GOUT (green), BOUT (blue), HSYNC and VSYNC outputs of the ARM7500. Later the HSYNC and VSYNC lines were buffered to prevent reflected pulses from the VGA lead from disturbing the LCD driving logic. The VGA interface has been tested at resolutions of 320x240 and 640x480 at 8-bits per pixel (256 colours).

The VGA display port is useful for debugging purposes and would also be handy in a finished product for driving external displays for presentations etc, but in order to be a self-contained computing device it requires some form of compact integrated graphical display. Two different Liquid Crystal Display (LCD) panels were interfaced to the ARM7500 on the Netslate board.

Monochrome LCD Interface

The first LCD panel to be driven successfully by the Netslate board was the Hitachi LM215XB. This panel is a monochrome type with 480x128 pixel resolution and no onboard controller. Its physical dimensions and long aspect ratio make it unsuitable for permanent use in the Netslate prototype, but it uses a somewhat similar interface to the colour LCD to be described next, and provided a good starting point.

The LM215XB has a 12-pin connector carrying: power and ground; four data bits; pixel, raster and frame clocks; an AC driving signal; and contrast and bias supplies. The bias supply for this panel is nominally -13.5V at around 3mA. The four data lines each correspond to one quarter of the display panel, which is divided into four subpanels each 240x64 pixels.

Each pixel of an LCD panel may only be driven on or off for each frame, so greyscaling consists of modulating each pixel value over many frames to produce an averaged duty cycle. Due to the slow response time of the human eye this is seen as a grey level, although some flicker may be visible. The ARM7500 employs patented LCD greyscaling logic which modulates each pixel value with its grey level, position and frame number so that neighbouring pixels do not flash on and off together, minimising the visible flicker.

The ARM7500 provides 16 grey levels from four bits per pixel. Four pixels (i.e. 16 bits of data) are greyscaled in parallel and driven out of the four bit LCD data port, along with the ECLK clock signal. To drive the LM215XB each data bit drives one quarter of the panel, requiring each sequence of 16 bits in the display buffer in memory to consist of four bits for each subpanel. The ECLK drives the LCD pixel clock CL2, HSYNC pulses the LCD raster clock CL1 at the start of each line, and VSYNC pulses the LCD frame clock FLM at the start of each frame.

LCD pixels consist of crystals which twist when a voltage is applied. The magnitude of twist determines the brightness of the pixel, regardless of the direction of the twist. If DC is applied, the crystal continues to twist until it is damaged.

A requirement of all LCD panels is that no DC voltage is applied to any pixel. To ensure this, both panels described here require an AC driving signal called M, which alternates with each frame. The M signal is used to

alternate the polarity of the applied voltage and hence the direction of twist with each frame, so that on average no DC bias is applied. To generate the M signal, a T (toggle) flip-flop in the EPLD is clocked by the VSYNC pulses. This maintains the duty cycle of the M signal at 50% as long as the VSYNC pulse occurs regularly.

Colour LCD Interface

While the monochrome LCD panel was useful in testing the LCD driving capabilities of the ARM and verifying the clocking circuitry, the Hitachi display was never intended to become part of the Netslate prototype. A passive colour LCD panel from Sanyo, the LCM-5330-22NSK, was the display finally packaged with the Netslate. This panel is of the type used in colour laptop computers. While it does not provide the same quality of colour reproduction as more recent active matrix displays, it is a lower cost part and is sufficient for prototyping.

The Sanyo display requires a +38V bias voltage for the panel. It also contains two Cold Cathode Fluorescent Lamp (CCFL) backlight tubes, each of which requires 1200VAC to strike the lamp and 400VAC under normal loads. Each tube consumes around 3W. These high voltages can be produced using switch mode inverters [10], supplied by ordinary batteries.

At the time of writing, the Netslate lacks the power supply circuitry needed to drive the fluorescent backlights in the display. While under bright light it is possible to make out lines and shapes on the display, it has been impossible to characterise the quality of the colour reproduction. Unfortunately a black against dark black display does not make for a good demonstration of the Netslate, so the prototype was demonstrated running an external VGA monitor.

The digital video output of the ARM7500 may be configured either for 4-bit greyscaled LCD data as used above, or 8-bit raw data. The ARM7500

datasheets suggest that the 8-bit raw mode is suitable for driving colour LCD panels, however this would require external greyscaling logic to achieve more than 3-bit colour.

A design was developed and implemented in the Netslate's EPLD which adapts the 4-bit monochrome LCD output port to the colour Sanyo panel's input port, which is 16 data bits wide. By running the ARM7500 in 4-bit greyscale mode and converting groups of three pixels to red-green-blue pixel triads for the panel, 12-bit colour is produced.

ARM Ltd of the UK has requested an application note detailing the interfacing of the ARM7500 to the Sanyo LCM-5330-22NSK. Appendix A gives details of the logic contained in the EPLD which performs this task. The application note will not be finalised until backlight supplies are ready and the colour reproduction is tested.

Mouse Interface

The ARM7500 provides two PS/2 style serial interfaces, one for mouse and one for keyboard. Each generates its own interrupts on receive and transmit of complete bytes. The Netslate board brings out the mouse data and clock lines to a PS/2 style connector which is currently used for an external trackball. When the trackball is eventually integrated in the Netslate casing, this connection will become internal and the socket will be free for an external PS/2 or AT style keyboard.

Serial Port Interface

The Netslate board incorporates a fairly standard RS232 transceiver and UART combination for serial communications. The UART is a Texas Instruments 16C550 and the transceiver is a Maxim MAX203 level converter with internal charge pumps. The 16C550 is pin and function compatible with the National Semiconductor 16550 employed in PCs, but is fabricated in CMOS reducing the power consumption considerably.

The UART is clocked from the 8MHz reference clock output from the ARM7500, which allows rates up to 38,400 baud to be produced accurately using the UART's internal frequency divider. If higher baud rates are required, the UART could be clocked from an external 1.8432MHz crystal oscillator.

The serial port implements the RTS and CTS lines for hardware flow control. The DTR, DSR and CD lines are looped back in similar fashion to a null modem connection, as the MAX203 does not provide sufficient level converters to implement these signals. The `nslib` library described in the next chapter includes interrupt driven RTS/CTS handshaking which has been tested up to 38,400 baud.

The ARM7500 provides two different levels of interrupt service, IRQ and FIQ. IRQ interrupts are used for most interrupt sources, and FIQ interrupts are used to service "fast" interrupts. When servicing FIQs, the CPU switches in a banked set of registers. This can reduce or eliminate the need to save registers on a stack before servicing the interrupt.

The UART's interrupt output is connected to both the INT2 and INT5 interrupt inputs on the ARM7500. This means it is able to cause both IRQ and FIQ interrupts, so it will not be necessary to change wiring if UART interrupts need to be upgraded from IRQ to FIQ level.

Power Supply

Power consumption of the Netslate board was measured under various operating conditions, and was found to be around 300 to 400mA at 5V, ie 1.5 to 2.0 Watts. Consumption is affected most by the amount of DRAM activity and whether or not a display is being driven.

The board is supplied by a 7805 5V regulator, which also dissipates some amount of power depending on its input voltage. More sophisticated

techniques are available which employ switch mode DC-DC converters [10] to boost or drop the supply to the required voltage at high efficiencies. These are able to cope with changes in supply voltage as the battery pack discharges to give as much operating time as possible.

Other techniques to save battery power usually involve powering down various parts of the system when they are not being used. It is an advantage to be able to power down the backlight of the display panel, as this is one of the more power hungry components of many systems.

Moulded Casing

The Netslate board and the colour LCD panel are housed in a moulded PVC plastic case, which was produced especially for the project. The case is 300mm x 190mm x 40mm in size, with a curved lower edge surrounding the space for the internal trackball. Thanks go to Colin Redmond for designing and moulding the case.

Summary

The Netslate prototype hardware contains a powerful CPU, memory, display and user interface hardware and a high speed communications port. The board has sufficiently low power consumption to run from a battery pack, and is packaged in a self-contained unit.

While the Netslate is hardly ready for mass production, it is a useful and functional prototype platform on which to build software. In the next chapter, we describe the software that runs on the Netslate board and demonstrates its capabilities.

Chapter 4 - Software Development

The previous chapter covered the hardware design and development of the Netslate prototype board. Of course, this hardware is useless without software. This chapter traces the stages of software development and testing, which progressed alongside the hardware development. We first look at the compiler and software development tools used, and then detail the library of functions for the Netslate board (`nslib`) and the demonstration programs that were developed.

Software Development Environment

Throughout this project, the GNU [11] software development tools have been used. These consist of a set of binary utilities including an assembler, linker, librarian and disassembler which make up the "binutils" package, and a C and C++ compiler in the "gcc" package. Both packages may be configured as cross development tools with the ARM as the target machine, and like all GNU software are freely available.

The development platform used was a Linux PC with a ROM emulator attached. The author ported the ROM emulator download program to Linux for this purpose. Linux provides a stable Unix environment in which to develop code and run the GNU cross development tools.

A third package, the "semilib" library [12] from ARM Ltd, is needed to build gcc with an ARM target. This library provides a very limited libc implementation with routines for such operations as division and modulus which are needed by the compiler but are not part of the ARM instruction set. The semilib package is also freely available.

The semilib library is intended for use with semi-hosted ARM development boards such as the PID board from VLSI, which connect to a host with

filesystem and I/O support via a debug monitor. The Netslate board does not have a debug monitor or hosted debugging facilities, so these I/O functions were not used. The startfile `crt0.o` in semilib is designed for loading programs using the debug monitor, so this was also unusable.

First Test Programs

When building a computer for the first time, it is very important to work by increments and test thoroughly at every stage. The first sign of life from the Netslate board came when only the CPU and ROM were installed. A single instruction program consisting of:

```
here:    b here
```

which branches to itself in an infinite loop was loaded into the ROM emulator and power was applied. On coming out of reset the ARM7500 commences executing code from address zero, which is the start of the ROM address space. Logic probes on the address lines confirmed that all fetches were occurring within the first four words of memory. This is because the ARM core employs a three stage pipeline with two prefetches after a branch, so the first three words of memory were being fetched continually.

From this point, a number of test programs of increasing complexity were written in ARM assembly language to test the memory and the general purpose I/O port IOP[7:0]. This port is eight open drain pins which may be individually pulled low by programming the IOLINES register with a control byte. When floating, the input levels on the port may be read back from the same register. This is a very useful way to get initial feedback from a board with otherwise no I/O implemented, as it requires no extra hardware except a logic probe on the relevant pins.

When DRAM was added to the board, it was tested by a number of programs which would write various patterns of words into the memory and read them back, signalling via the IOP[7:0] lines if a mismatch was found.

This allowed ROM and DRAM to be fully tested and debugged before any other I/O capabilities were added.

The `nslib` Library

As hardware was added to the Netslate board, functions to test and utilise the new modules were written and added to the body of Netslate code. The end result of all of the software modules that were produced is a library of functions called `nslib`. The `nslib` package has been made freely available for other developers using the ARM7500 in the hope that it will be found useful.

In this section, the modules which make up `nslib` are described, followed by a number of test programs which demonstrate the library and the Netslate board's capabilities. Appendix B lists `nslib.h`, the header file which describes all of the externally visible features of the library.

The Exception Vector Table

Exceptions are events which require special handling by the processor, and cause execution to jump to an exception handler address for this purpose. Exceptions which may arise in the ARM7500 are caused by reset, hardware interrupts (IRQ and FIQ), software interrupts (SWI) and when the processor traps an undefined instruction or an illegal memory access.

The first eight words in memory make up the exception vector table. The first of these vectors is for reset, which is why execution starts at address zero upon being reset. Each vector consists of one instruction which should branch to the appropriate exception handler.

All Netslate programs are linked to start at address zero, so that the exception vector table may be part of the program. The table is written in eight lines of assembly code in `crt0.S`, which produces the start file `crt0.o`. This start file is automatically linked to the start of every program, so it is guaranteed to start at address zero in the ROM. At the time

of writing, all exceptions except resets and IRQ interrupts cause execution to halt.

Initialisation and Memory Map

A number of initialisations of the Netslate hardware and specifically the ARM7500 need to be performed immediately after coming out of reset. The reset exception handler is written in assembly so that it can set up the memory map and stacks required before any C functions can be called.

The CPU is configured for 32-bit program and data address spaces and set to supervisor mode. Netslate programs run in supervisor mode as opposed to user mode so that they may freely enable and disable interrupts. Stacks are set up for supervisor and interrupt mode programs at this point.

The other two main actions that are performed on reset are to copy the program from the ROM into the DRAM, and map the DRAM into the address space previously occupied by the ROM at address zero. It is always necessary to copy the initialised data section (data) into DRAM, so that its contents may be altered by the program. In the case of the Netslate board, copying the program code itself speeds execution, as fetches from DRAM require less cycles than ROM. Remapping the DRAM to commence at address zero means that the exception vector table on the front of the program points to the exception handlers in DRAM.

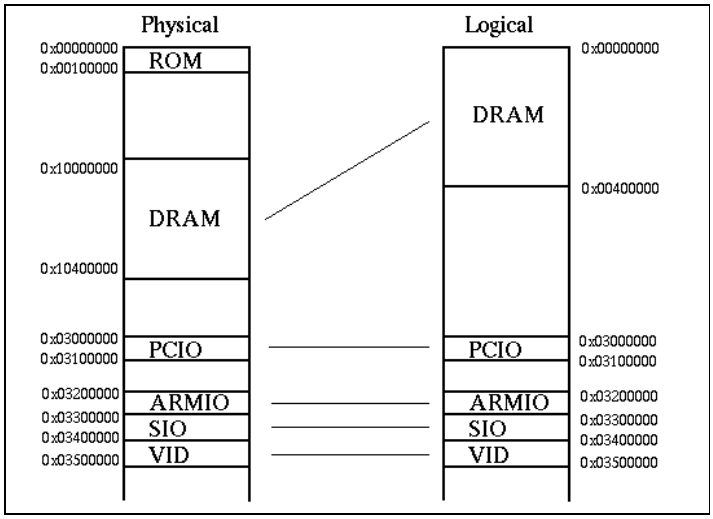


Fig. 9 - The Netslate logical-to-physical memory mappings

The ARM7500 contains a Memory Management Unit (MMU) which handles address translation and permission checking, as well as controlling the cache and write buffer. A 16Kbyte block of memory called the level one translation table describes mappings of one megabyte sections of logical address space to sections of physical memory or I/O address space. Figure 9 shows the logical and physical memory maps used by the Netslate board.

The MMU is also capable of much finer division of memory mappings using pages which range in size from 4kB to 64kB. In this case, the entry for an address range in the level one table points to a level two table or page table which gives the information on individual pages. Currently the Netslate software does not make use of pages as these are most useful in a protected memory environment.

Once the level one translation table and associated registers have been set up, the MMU can be enabled bringing the memory mappings into effect. At this point the cache and write buffer are also enabled, the program is running from DRAM, and control passes to the `main()` function.

Interrupt Handling

Apart from reset, the only exception which is implemented on the Netslate is the IRQ interrupt exception. The ARM7500 has almost thirty different IRQ interrupt sources, some internal and some external. Internal interrupt

sources are generated by events within the chip, such as when one of the internal timers reaches its terminal count or when the PS/2 mouse interface receives a complete byte. External interrupts are generated by either levels or transitions on external interrupt pins, such as INT2 used by the 16C550 serial port.

Several sets of 8-bit memory mapped registers in the ARM7500 provide masks and status information on all of the interrupt sources. If a particular interrupt event occurs when the corresponding mask bit for that interrupt source is set, an IRQ exception will occur and the corresponding status bit will read as one.

When an interrupt occurs, the IRQ exception trap saves the register set onto the IRQ stack and passes control to an interrupt dispatch routine (`IrqDispatch`). The interrupt dispatcher examines the interrupt request registers (which are the status registers bitwise-ANDed with the mask registers) from highest to lowest priority, until a set bit is found. A table of interrupt vectors supplies the address of an Interrupt Service Routine (ISR) for that particular source, which is called to handle the interrupt. Upon return of the ISR, the register set from the interrupted program is restored, interrupts are reenabled and control returns to the interrupted address.

In order to keep the interrupt response time as short as possible, a priority resolution table is used to determine the highest priority bit that is set in a non-zero request register. When indexed with a byte mask which has at least one bit set, the table gives the bit position of the least significant bit that is set. This lookup process is a constant time operation which is faster than looping and shifting through the bits in each byte until a non-zero bit is found.

The current interrupt handling scheme does not allow nested interrupts. It is possible to implement a prioritised nested interrupt scheme on the

ARM7500 by masking off all lower priority interrupt sources and enabling interrupts before passing control to the ISR. Of course, these interrupts must be masked back on again on return. Nested interrupt schemes are useful when ISRs have significant execution times and high priority interrupts require low latency times.

Mouse Module

The mouse module is a good example of how interrupts are handled with `nslib`. This module provides an interrupt driven software interface to the PS/2 style serial mouse interface.

This module contains two functions, one to initialise the mouse and the module (`MouseInit`), and another which is the ISR (`MouseIsr`). A point to note is that before installing and enabling the ISR any pending interrupt from that source is cleared.

Each time the PS/2 mouse interface receives a complete byte from the attached peripheral, a mouse receive interrupt occurs and the interrupt dispatcher calls the `MouseIsr` routine. This routine maintains a static variable which counts the bytes received in each message from the mouse. When a complete message is received, the mouse state variables are updated.

The position and button state of the mouse is maintained in global variables which may be polled by the main program. In a multitasking environment this could be replaced by a queue or some other event structure which is posted to by the service routine, as will be discussed later in the section on multitasking kernels.

Serial Communications Module

The serial communications module provides a number of functions which allow bytes to be transmitted and received over the RS232 serial link. This module is similar to the mouse module in that it contains an initialisation

function and an ISR function. The module buffers all data in both directions and provides functions to send bytes, receive bytes and check if any bytes are ready in the receive buffer.

The `SerialSend` and `SerialRecv` functions which are called by the main program to send and receive bytes will block until space becomes available in the transmit buffer or a byte is available in the receive buffer respectively. In a multitasking environment the transmit and receive buffers would have associated event control blocks which would allow other tasks to continue while the caller is blocked waiting for space or data.

There are established and reliable ways of handling serial port interrupts. The serial port ISR performs the following sequence of actions:

1. Mask off and acknowledge the interrupt source
2. Repeatedly read the Interrupt Identification Register (IIR) and service the indicated interrupt cause until bit zero is set, signifying no more interrupt causes
3. Unmask the interrupt source

It is important to mask off transmit interrupts at the UART immediately the transmit buffer becomes empty. Otherwise a spurious interrupt will arise after the last byte has been transmitted. Correspondingly, transmit interrupts must be reenabled when data is added to a previously empty transmit buffer.

The serial module also implements RTS/CTS hardware flow control on receive. This is used to signal to the remote host whether the Netslate is able to accept data. When the receive buffer is almost full, the RTS line is lowered to signal that the remote host should stop transmitting. When the main program accepts enough data to make space in the receive buffer, RTS is raised again.

Video Display Module

The Netslate board supports graphical output to a number of display devices. The video module in `nslib` provides routines for controlling and displaying a variety of graphics primitives on SVGA monitors, and the monochrome Hitachi and colour Sanyo LCD panels.

These primitives include pixels, lines, filled and outline triangles rectangles and circles, characters, text and shaped mouse pointers. The video modes currently supported are: CRT monitor at 640x480x8bit colour with palette, Hitachi LCD panel at 480x128x4bit greyscale and Sanyo LCD panel at 640x480x12bit colour.

All of the video functions are accessed through a global record corresponding to the current video mode. This allows modes to be changed by changing the pointer to the configuration record. This is most useful in a system which allows video modes to be changed on the fly, which is envisaged for the Netslate. The library header `nslib.h` provides a set of macros with which to call all of the video functions through the configuration record.

New video modes can be added to the library by modifying the drawing and initialisation routines and creating a new configuration record. Each mode also has an associated assembly language file which allocates space for the display buffer. Assembly is used so that the buffer is guaranteed to be quad-word aligned, a requirement of the video DMA channel.

One of the ARM7500's internal interrupt sources is a video flyback interrupt which is triggered on every vertical retrace of the display. The video module contains an ISR which is hooked to this interrupt. It maintains a count of the number of retraces since the main program last synchronised to the display.

Programs that need to perform page flipping for smooth animated graphics can call the synchronise function (`VidSync`) which waits for the next retrace and then returns the number of frames that have elapsed, allowing the rate of motion to be independent of the frame rate. This technique is used in the Space demo program described later in this chapter.

Heap Management

The `nslib` package includes very simple implementations of the `malloc` and `free` routines used to dynamically allocate and deallocate blocks of memory on the heap. These routines maintain a doubly linked list of word-aligned blocks. The `nslib malloc` and `free` are non-reentrant and so are not suitable for use in a multitasking environment. The heap is used by the Server demo program described below.

Server Demonstration Program

A number of demonstration programs have been developed during the Netslate project, each using the routines in the `nslib` library. The first and most important of these is the Server program.

The Server program is a display server for a host program running on the Linux host. The host program sends drawing requests over the RS232 serial link to the Netslate board, which the Server program interprets and displays. This allows a limited subset of the X-windows drawing capabilities to be mirrored on the Netslate display.

The host program is a modified Tcl/Tk interpreter from Sun Labs [13]. Tcl/Tk is a scripting language with a graphics extension, which runs on Unix workstations with X-windows terminals. The Tcl/Tk interpreter is freely available in source form. The modifications made involved redirecting all of the calls to Xlib display functions through "hook" functions. These hook functions then call the real Xlib function as well as making up a request record and sending it over the serial link.

Surfit! [14] is a freely available Web browser written entirely in Tcl/Tk. It handles standard HTML as well as a number of extensions now in common use. When Surfit! is run using the modified Tcl/Tk interpreter, the Web browser appears both on the X-windows terminal and on the Netslate display.

Some visible differences between the two are due to the Netslate server only having one fixed width font and incompletely implementing clipping of drawing requests. However the colour map is true to the original and Surfit!'s GIF image handling is supported, loading images in raw form over the serial link.

Once an image is loaded it may be redrawn from the copy stored on the heap by the Netslate server, making scrolling and redrawing much faster. The Netslate server also dynamically allocates space for X-windows "pixmaps" which are used to draw complete widgets before copying them to the visible screen.

While the Server program and its associated serial protocol implements only a very limited subset of the X-windows protocol, it is sufficient to demonstrate the display capabilities of the Netslate hardware. This program was the key demonstration of the Netslate as a Web browser at the end of the project.

Space Demonstration Program

The Server program demonstrates that the Netslate has sufficient display and communications capability to run as a terminal for a Web browser on a host machine. Unfortunately the performance of the Server demonstration is limited by the speed of the host machine and the serial link.

The ultimate goal for the Netslate is to run all of the Web browser software locally without relying on a remote host. The Space demonstration program

is aimed at showing that the Netslate hardware has sufficient processing power to perform this task standalone.

Three dimensional graphics is often used as a demonstration of system performance for its crowd appeal and high "wow" factor. The Space program draws a smoothly rotating scene in 3D space made up of points, lines and filled triangles and circles. The user can move the viewpoint around the scene with the trackball and zoom in and out with the buttons.

The Space program implements matrix transformations, vector operations and trigonometry functions using fixed point arithmetic. The sine and cosine functions use linear interpolation on a compact lookup table, which provides more than enough accuracy for graphics purposes. No floating point maths is used as the ARM7500 does not implement a floating point coprocessor and the Netslate environment does not provide software emulation. In any case, integer arithmetic is sufficient and probably faster than hardware floating point.

The program uses page flipping to draw to a back page and then switch the display to that page to eliminate flickering. The video module's synchronisation function is used to ensure page flipping occurs during retrace, and also locks the rate of movement and rotation. If three frames have elapsed since the last page flip, the movement of the scene is updated three times.

Depending on the proximity of the point of view to objects in the scene, the frame rate achieved varies around 20 to 30 frames per second. This could be improved by further optimisation of the video display routines. This frame rate is comparable to if not better than that achieved by many commercial games for the PC.

The main motivation for demonstrating the performance of the Netslate hardware is to show that it is capable of dealing with the increasing complexity of graphics and multimedia content which is appearing on the Web.

Tasks Demonstration Program

Multitasking and multithreading are a basic necessity of operating systems with graphics and networking capabilities. Separate tasks for dealing with the network, redrawing areas of the display, accepting user feedback etc make programming for interactive systems far more intuitive and straight forward.

uC/OS [15][16] is a real time multitasking kernel that is freely available in source form. It has been ported to a number of platforms including the ARM600 PID development board from VLSI. This port is written with the Norcroft ARM C compiler and binary utilities in mind.

The author modified the ARM600 version of uC/OS for the ARM7500 and the GNU development tools. There are significant differences in the assembler syntaxes of the ARM and GNU assemblers. The interrupt handling also needed modification to work with `nslib` and the ARM7500's interrupt mask and status registers, as the PID board uses an external interrupt controller.

While the modified kernel runs reliably with a single interrupt source, it usually crashes when two interrupts arise simultaneously. It is unclear whether the cause of the problem is a bug carried over from the PID version or whether it has been introduced by the Netslate code. The problem has been isolated to context switching on returning from a nested interrupt.

The Tasks demonstration program runs three independent tasks which each periodically display filled rectangles, triangles or circles. A mouse cursor is

also displayed. Moving the mouse rapidly increases the number of interrupts and the chance of two occurring simultaneously, encouraging the program to crash. Upon crashing the program halts with an aborted data access exception, suggesting that code or data is corrupted.

Menu Program

The Menu program is a small front-end to the other demonstration programs which allows them all to be linked into the one ROM image. On resetting the Netslate, the user is presented with a mouse driven menu of the programs available. This program was used for the demonstration session of the project to eliminate the ROM emulator pod. A pair of EPROMs were programmed with all three demonstration programs which could be individually selected upon resetting.

Summary

The `nslib` library provides a starting point for further development of the Netslate platform. All of the modules of the Netslate hardware have been tested during the development of the library, and a number of demonstration programs apply the library to show the modules working together.

The Server program provides proof-of-concept of the handheld Web browser and demonstrates the Netslate communications capabilities. The Space program demonstrates the performance of the Netslate hardware, and the Tasks program provides a rudimentary demonstration of multitasking.

The GNU compiler and binary utilities provide a stable environment for further software development with the Netslate. The final chapter of this report discusses directions for building operating system and applications software on this platform.

Chapter 5 - Future Directions

Chapters 1 and 2 introduced the concept and importance of a handheld Web browser and examined a number of related products and developing technologies in the field. Chapters 3 and 4 detailed the hardware design and software development of the Netslate prototype. In this final chapter we look to the future of this fast growing field and explore directions for further development of the Netslate project.

As this project has mainly been concerned with the hardware and low level support software of the Netslate prototype, the most immediate concern in taking the project further must be to develop an operating system and substantial applications software.

The author hopes that this chapter will provide direction to students continuing with Netslate-related projects in coming years.

Operating System

An operating system for the Netslate should consist of a multitasking kernel with sufficient Inter-Process Communication (IPC) and a set of modules for networking, filesystem support and display management.

The Applications Programming Interface (API) between the operating system and the application program(s) needs to be clearly defined. If the operating system and applications are loaded separately, it is likely that all system calls will be via the software interrupt (SWI) instruction, with specific SWI numbers corresponding to particular system calls.

The simpler option is to compile the operating system and the application into one program and make function calls as normal. This affords no protection mechanisms for the operating system code, but fully protected

operating systems are only needed when running untrusted applications programs. When the two are compiled into one, the API is defined by a single header file describing a layer of function calls.

Multitasking and IPC

As mentioned in the previous chapter, multitasking is almost a prerequisite of an operating system in interactive and realtime systems. It allows separate tasks to handle various parts of an interactive system concurrently, while providing communication between processes. IPC is made up of a number of communication primitives within the kernel. The uC/OS kernel implements counting semaphores, mailboxes and message queues.

One example of the use of IPC primitives is in the mouse interface. Currently a program using `nslib` needs to poll global variables to check for a button press. In a multitasking system this could be replaced by a semaphore posted to by the mouse ISR. The mouse handler task would pend or wait on this semaphore, removing it from the list of tasks that are ready to run, and the kernel would schedule some other task. When a mouse event is received, the task would again be ready to run and would be scheduled.

In a multitasking system it is often the case that all tasks are waiting for some event and nothing is ready to run. As the kernel always needs to have a task to switch to in this case, an "idle" task of the lowest possible priority is used to occupy the CPU until the next event. The idle task in the Tasks program is an infinite loop which places the ARM7500 in SUSPEND mode, gating off the clock to the core until the next interrupt. This practice saves power consumption when the program is idle while still allowing DMA to continue.

TCP/IP Networking

The ultimate goal of the Netslate is to connect directly to the Internet through some service provider using the networking protocols already in

common use. This means that it will need to implement a basic TCP/IP protocol stack with support for either the SLIP or PPP serial protocols.

The absolute bare minimum set of protocols needed to connect to the Internet, contact a remote host and download a Web page is SLIP [17], IP [18], TCP [19] and HTTP. SLIP (Serial Line Internet Protocol) encapsulates packets for transmission over the serial link. IP (Internet Protocol) is responsible for the routing of individual packets, which is fairly trivial to implement on a SLIP-connected client.

TCP (Transmission Control Protocol) is the most involved of these protocols and provides the abstraction of a continuous stream of data to the application layer. It is responsible for retransmission of lost packets, end to end flow control, packet ordering and so on. HTTP (HyperText Transfer Protocol) is the protocol used to talk to Web document servers and is trivial to implement.

PPP (Point-to-Point Protocol) [20] is similar in function to SLIP but is by far the preferred protocol for performance, robustness and flexibility. It is, however, far more complex to implement than SLIP. A complete implementation of SLIP is given in C code in the RFC document that describes the protocol.

All of these protocols (and in fact all open protocols used on the Internet) are documented in freely available documents called RFCs (Request For Comments). A list of relevant RFCs and sites where they can be obtained is given in the reference section. Consult Stevens [21] for the Bible of network programming.

The TCP protocol makes extensive use of timers to determine when packets that have not been acknowledged should be retransmitted, when the

connection to the remote host has been lost etc. The uC/OS kernel provides a system timer which can be used for this purpose.

All pends on event control structures such as semaphores can be made to timeout after a certain period of time, which lends itself to the construction of a retransmission queue. Message queues would also make a suitable means for applications programs to send and receive data to and from the TCP manager task.

Filesystem Support

Some degree of filesystem support will be necessary for a standalone Web browser, to supply temporary space for cached documents, image files etc. A RAM-disk style filesystem should be straight forward to implement as no complex directory structures or protection mechanisms are needed.

While space for files could be dynamically allocated on the heap, this could lead to problems with fragmentation of memory. This occurs when the heap grows as blocks are allocated and then some blocks are deallocated. This results in the free heap space being distributed in fragments of memory, limiting the largest contiguous block that can be allocated.

A better implementation would use a single large block of memory divided into sectors analogous to those on magnetic disks. Files then become distributed over chains of sectors without fragmenting the heap. Example implementations of simple filesystems using sectors (RAM or otherwise) can often be found in embedded software archives [22].

The TCP networking protocol discussed earlier requires some amount of buffer storage. Performance of network transfers can be improved to a point by offering larger "window" sizes to the remote host, requiring larger buffers. One technique to simplify code and make better use of memory

might be to implement the network buffers as sectors allocated in the filesystem.

A simple RAM filesystem is sufficient for temporary storage purposes, but bookmark files and user preferences require some non-volatile storage. While these could be stored on the network, it could be useful for the Netslate to have a small amount of battery backed or flash memory in which to store information such as a dialup or login script with which to contact the service provider. This may or may not be needed depending on the means of connection. The Netslate prototype board has surface mount pads brought out to vias for a 32Kbyte SRAM chip if this is later required.

Windowing System

The last prerequisite module for our operating system is a windowing system. This provides the foundation for the Graphical User Interface (GUI) provided by the applications software.

The X windows protocol [23][24][25] is very big and very ugly, qualities that are shared by the code that implements it. While familiarity with X is useful to understand the concepts and techniques involved, it should not be taken as an indication of the complexity required of a basic windowing system.

Start with a clear idea of just what constructs are required of the windowing system at the application level. The system should be made up of three fundamental constructs: windows, pixmaps and drawing primitives.

A window is a rectangular area with a size and a position within a parent window. The root window is the special case window which occupies all of the display and has no parent. The window hierarchy is stored in a tree structure. A window is only ever visible within the bounds of its parent window. Each widget in an application's main window will have its own

window, and many widgets will have sub-windows. Each widget window is a child of the main window which is a child of the root window.

A pixmap is similar to a window but is stored off screen and is not part of the window hierarchy. Pixmaps are used as temporary scratch space for drawing widgets before copying them to visible windows. Pixmaps and windows are sometimes generalised and termed drawables.

Drawing primitives make up the rectangles, lines, strings, images, copy operations etc that can be drawn to a window or a pixmap. All drawing to a pixmap must be clipped to the edges of the pixmap. All drawing to a window must be clipped to whatever portions of the window are actually visible and unobscured by other windows. A window can only be obscured by the immediate children of that window's parent or by anything obscuring the parent. A generalised set of operations on regions made up of rectangles may be useful for clipping.

Whenever a previously exposed area becomes visible, that area is said to be damaged as it needs to be redrawn with the contents of the exposed windows. Dragging a window across the screen accumulates damage to the windows below it. Redrawing is delayed until the CPU is not busy drawing the moving window as this task is more important to responsive user interaction.

The exposed area is accumulated in a damage region structure. When the window system task is scheduled (presumably because the application is not busy moving windows etc) it calls callback functions on each damaged window. The application's callback functions in turn call drawing primitives to redraw themselves. The area of the each window that receives an expose callback is then removed from the damaged region until no damage remains.

Floating Point Arithmetic Support

It is inevitable that some applications will require floating point arithmetic. Both the Tcl/Tk and Java interpreters use some amount of floating point. While floating point arithmetic instructions are part of the ARM instruction set, on the ARM7500 they trigger an undefined instruction exception. It is possible to emulate the instructions in an exception handler by tracing back to the unrecognised instruction. This also requires the operating system to maintain a set of coprocessor registers. What is worse, these registers must be swapped on every context switch, which changes the shape of the saved stack frame of the kernel.

The simpler approach to this problem seems to be to avoid floating point instructions by letting the C compiler generate integer code to emulate them. Recent versions of gcc include code generation to emulate floating point instructions, but some amount of work is required to build this into the compiler. The author has not yet ventured down either of the floating point emulation paths.

Application Programs

Unless the developer plans to write a Tcl/Tk or Java interpreter or even a Web browser from scratch, it is likely that building the application will be a matter of porting code from some other platform such as Unix. This is the case with the Tcl/Tk interpreter which was modified for this project. The author investigated porting the entire interpreter to the Netslate platform before deciding that much more operating system support was needed than could be implemented in the time available.

Once the basics of a useable operating system are in place, porting larger applications such as Tcl/Tk become possible. Any given application will require a certain amount of library support which may not have already been implemented for the kernel. In the case of Tcl/Tk, a fair number of string operations such as searching and formatting are required. All of the

library functions used for evaluating floating point expressions will also need implementations, but most of these can probably remain as stubs to get the interpreter running initially.

Obviously it will save unnecessary effort if the developer is familiar with just what library and operating system support are required by the application in question. Try linking the application on a Unix platform without any libraries (not even libc), the linker errors will give an indication of what functions are required. Remember that this list is not comprehensive as some library functions call more library functions. In any case do not feel that you need to implement a complete POSIX environment before you can get the basics of your application working.

Customisations and Extensions

Once the application has been ported and is running reliably, it is time to consider changes to the application to tailor it to the Netslate hardware. The most obvious lacking of the Netslate compared to a workstation is that of a keyboard. One solution to this is to implement a virtual keyboard which appears on the display when a text entry field has the keyboard focus. Because browsing the Web is mostly point-and-click, slow text entry of this type is sufficient for the occasional form or keyword search entry.

Further extensions requiring complex algorithms, intensive processing and major changes to the applications software include handwriting and speech recognition. Of course, extensions to the Netslate hardware would also be needed to use these input methods. These are both techniques that we will certainly see more of in future devices and the Netslate prototype is certainly capable of supporting extensions of this kind.

Further Hardware Development

While the Netslate prototype hardware is reasonably complete for the purposes of developing software, a certain amount of work would be needed to bring the Netslate from a prototype to a product ready for

production. Mainly this would involve complimenting the terminal hardware with the necessary wireless communications circuitry and generally finishing the product. A number of other improvements and developments are indications of how this type of device will take shape in the coming years.

Wireless Networking

This project has considered the matter of how the Netslate makes a wireless connection to the Internet to be somebody else's problem. Of course this attitude does not make for a complete product and this is one of the major barriers to be overcome before a Netslate product is marketable.

The two major technologies in wireless communications are InfraRed (IR) and cellular radio. Each has its pros and cons.

Cellular radio is the same technology that is used currently by mobile telephones. Its immediate advantage is that many areas are already covered by cellular service, so devices with inbuilt radio modems can be used in these areas. This is the case with the Motorola Marco and Envoy products discussed in chapter 2.

The disadvantages of cellular radio are limited bandwidth and governmental regulation concerns. The radio frequency spectrum is under heavy demand from a many different groups of users, and so regulatory bodies are unlikely to give up great swathes of the band for personal wireless digital communications any time soon. Range and power requirements of cellular radio devices is also of concern.

InfraRed technology has until recently been the domain of line-of-sight remote controls for consumer electronics products. A number of research efforts are underway to develop very high bandwidth diffuse InfraRed

communication links. Diffuse links do not require line-of-sight, instead using reflections from the walls and ceiling.

IR has the distinct advantage that its use is not controlled by regulatory bodies. Since IR does not travel through walls, it cannot interfere with other IR systems in neighbouring rooms or buildings. However this does imply that the cell size for an IR network is one room. The cost of IR technology and the cabling that runs to transceivers in each room will need to drop substantially before it comes into common usage.

One way of deferring the choice of communications technology is to equip the Netslate with an interface such as a PCMCIA slot [26], which can accept a variety of network interfaces, expansion cards etc. Of course this requires appropriate card and socket driver software to be built into the operating system, a not insignificant task. The ARM7500 is compatible with a number of "single chip" solutions to PCMCIA socket interfacing.

Input Devices

Natural methods of interaction go a long way to making a user feel in control of and comfortable with a system. As mentioned earlier in this chapter, handwriting and speech recognition are currently the two big pushes in this direction for handheld devices. Both of these would be a major undertaking as an extension to the Netslate prototype, but both are within the realms of possibility.

Even if handwriting recognition was not implemented, a touch screen for the Netslate would be a great improvement over the trackball. The current Netslate prototype would ideally have had a touch screen if this was possible within the budget of the project. It was decided that a large colour display and trackball was better than a small monochrome touch screen.

Touch screens generally produce a number of resistive variables to the system which provide the position and pressure of the touch. The resistive inputs on the ARM7500 are sufficient for joystick inputs but do not have the sample rate and accuracy required by a touch screen interface. This implies that an external Analog-to-Digital Conversion (ADC) interface would be required.

The extra hardware needed to support speech input is also an ADC, however a higher sample rate is required to preserve enough information for speech recognition. A DMA channel is ideal, but the ARM7500 does not provide any spare general purpose channels. An alternative is to make use of the fast FIQ interrupt inputs to grab each sample from a port. Interrupt traffic could be reduced by buffering a number of samples outside the CPU, perhaps implemented in an EPLD similar to the colour LCD interface.

Future VLSI Technologies

One promise of current trends in Very Large Scale Integration (VLSI) manufacturing technology is to integrate greater numbers of functions into single packages. One inevitability is that we will soon see a package with a processor and sufficient DRAM for a useful system on a single die.

Every year transistors get pushed closer together on the silicon. Clock speeds rise, power consumption falls. The amount of functionality on one die increases, and package counts and prices drop.

Levels of integration are reaching the point where it should be possible to include an ARM CPU on a 32Mbit (4Mbyte) DRAM die. DRAM manufacturing processes require special steps not necessary for CPUs, but this does not preclude fabricating a CPU on a DRAM process.

The number of pins on processor packages has been steadily increasing for years, as data and address buses get wider. One significant result of placing

processor and memory on the same silicon is that the pin count will actually drop. Sixteen data lines is sufficient I/O bus width for most applications, and the address bus can be reduced to a few low order register select bits and a number of internally decoded chip select lines.

As more and more transistors can fit on one die, we can expect to see package counts drop even further as the CPU absorbs DRAM, even more I/O functionality, even FPGA style programmable logic. At the same time performance and efficiency will continue to grow, pointing to an exciting future for true single-chip handheld devices.

Lessons Learned

Above all, the Netslate project has been an incredible learning experience for the author. The following are a few lessons learned during the project that have very broad application.

1. Never assume anything

Almost all of the problems encountered during the project were due to incorrect assumptions. Thoroughness in design is everything. One example of a mistake that could have been avoided occurred in the interrupt handling code for the ARM7500. The IRQ request registers are each 8 bits wide but occupy word addresses. After spending many hours searching for the source of a spurious interrupt, it was found that the higher order bits in the word do not always read back zero, as had previously been assumed.

2. Test, don't guess

The above problem was not found by the hours of fiddling with code to see what the results would be. It was found by inserting a debug statement which displayed the word value of the IRQ request register. Good debugging tools and techniques are invaluable.

3. Document debugging

A problem with the Netslate startup code which was fixed early in the project reared its head once again much later. By this time it was forgotten what the original source of the problem was and how it had been fixed the first time. Good revision control practices could have prevented the problem from reappearing, and proper documentation of problems and fixes throughout the project could have saved the time spent debugging it again.

Summary

As demonstrated by the programs described in the previous chapter, the Netslate platform is capable of high speed communications, graphical user interfaces, gaming-quality animations and multitasking. This sets the scene for a compact operating system with communicating processes managing TCP/IP networking, a volatile filesystem, a windowing GUI and all the applications support needed to run a standalone Web browser.

The author challenges the reader to take up the Netslate where this project leaves off, and develop the operating system and applications software that will turn the prototype into a complete handheld Web browser system.

Conclusions

The Netslate project has achieved the short term goals stated in the first chapter of this report. These were to design and construct hardware for a prototype handheld Web browser and to develop a low level support library for testing, demonstration and further development of the prototype.

By combining architectural influences from personal computers, embedded systems and ubiquitous computing projects, the Netslate design integrates processing, communications and user interface hardware into a self-contained, low power, low cost system.

The ARM7500 CPU integrates a large part of the functionality required by the
prototype into a single device. By adding the necessary support hardware to a flexible development board, the Netslate represents a platform that is ready to support high level software development.

The GNU software development tools and the `nslib` support library written specifically for the Netslate make up the basic for further software development on this platform. A number of demonstration programs have proven that the Netslate prototype hardware and the `nslib` library is a functional combination.

Much work is still to be undertaken before the Netslate is a fully-functional standalone handheld Web browser. The author hopes that future thesis students will take on these challenges and learn as much as was gained from this project in the process.

The last fifteen years have seen incredible change in the computing industry. Now that the era of the PC is coming to an end and the Network Computer is coming to the fore, technologies like the handheld Web

browser will no doubt play their part in bringing global information access to the masses.

References

[1] Apple Newton MessagePad PDA
http://www.newton.apple.com/

[2] Motorola Marco Communicator
http://www.mot.com/MIMS/WDG/products/marco/

[3] Motorola Envoy Communicator
http://www.mot.com/MIMS/WDG/products/envoy/

[4] Xerox ParcTab project
http://www.ubiq.com/parctab/

[5] Berkeley InfoPad project
http://infopad.eecs.berkeley.edu/

[6] Oracle Network Computer
http://www.oracle.com/products/nc/

[7] The Java language at Sun
http://www.javasoft.com/

[8] ARM7500 datasheets
http://www.arm.com/Documentation/UserMans/

[9] ARM processor architecture
http://www.arm.com/

[10] Maxim Application Datasheets (including switch mode supplies)
http://www.maxim-ic.com/

[11] The GNU project of the Free Software Foundation
http://www.gnu.org/

[12] ARM semilib library and versions of GCC for the ARM
ftp://ftp.cl.cam.ac.uk/arm/gnu/

[13] The Tcl/Tk language at Sun
http://www.smli.com/research/tcl/

[14] The Surfit! web browser from ANU
http://surfit.anu.edu.au/

[15] Jean Labrosse's uC/OS (book)
http://www.rdbooks.com/microcos.htm

[16] Jean Labrosse's uC/OS (source)
ftp://ftp.cygnus.com/pub/embedded/ucos/

[17] RFC1055 - Serial Line Internet Protocol
http://www.ua.ac.be/RFC/r1055.html

[18] RFC791 - Internet Protocol
http://www.ua.ac.be/RFC/r791.html

[19] RFC793 - Transmission Control Protocol
http://www.ua.ac.be/RFC/r793.html

[20] RFC1661 - Point-to-Point Protocol
http://www.ua.ac.be/RFC/r1661.html

[21] Stevens, Unix Network Programming , Prentice Hall 1990

[22] Embedded software archives at Cygnus
ftp://ftp.cygnus.com/pub/embedded/

[23] X Windows Reference Vol. 0 - Protoco l Reference

[24] X Windows Reference Vol. 1 - Xlib Reference

[25] X Windows Reference Vol. 2 - Programmers Manual

[26] Personal Computer Memory Card International Association (PCMCIA)
http://www.pc-card.com/

Appendix A - `lcd.tdf` colour LCD interface

The following is a listing of `lcd.tdf`, a Altera text design file written in AHDL, which describes the function of the EPLD device making up the colour LCD interface.

```
-- lcd.tdf
--
-- Copyright (c) 1996 Ben Williamson.
-- All rights reserved.
--
-- This file is an Altera Text Design File, written in AHDL.  It
-- describes an EPLD device for latching LCD display data from the
-- ARM7500 and presenting it to a colour LCD panel.  The ARM gives
-- 4 bits of greyscaled LCD data on every ECLK.  The colour LCD
-- requires 16 bits of data on every CL2 clock period.  Three levels
-- of 4 bit DFFs store four ECLK's worth of data, while a 2 bit
-- counter produces CL2 and latches the 16 bit word every four
ECLKs.

SUBDESIGN lcd
(
        eclk            : INPUT;        -- clock from the ARM7500
        ed[7..4]        : INPUT;        -- data from the ARM7500
        hsync           : INPUT;        -- pulses high after each line
        vsync           : INPUT;        -- pulses high after each frame
        ndisp           : INPUT;        -- an active low enable signal

        ud[7..0]        : OUTPUT;       -- drives even-numbered pixels
        ld[7..0]        : OUTPUT;       -- drives odd-numbered pixels
        ncl2            : OUTPUT;       -- drives CL2 (buffered externally)
        nm              : OUTPUT;       -- drives M (buffered externally)
        disp            : OUTPUT;       -- drives DISP to enable display
)
VARIABLE
        shift[15..4]    : DFF;          -- three levels of 4-bit registers
        data[15..0]     : DFFE;         -- stores the output word
        count[1..0]     : TFF;          -- counts through four ECLKs
        toggle          : TFF;          -- produces the M signal
        EN              : SOFT;         -- a name for the data latch enable
        NHSYNC          : SOFT;         -- a name for NOT hsync
BEGIN
        -- ndisp is driven from an active low ARM output.  This could
        -- have been buffered elsewhere, but what the hell
        disp = !ndisp;

        -- Aliasing !hsync to NHSYNC makes an Altera error go away
        NHSYNC = !hsync;

        -- Everything is synchronous with the falling edge of eclk
        -- The counter gets reset after each raster.  It drives
        -- the fast LCD clock, CL2.
        count[1..0].clk = !eclk;
        count[1..0].clrn = NHSYNC;
        count[0].t = VCC;
        count[1].t = count[0].q;
        ncl2 = count[1].q;

        -- The LCD requires an M input which is high for a frame, low
        -- for a frame.  This flip-flop toggles on every vsync pulse.
        -- If this signal stops toggling while DISP is enabled, the
        -- display will die very quickly.
        toggle.clk = NHSYNC;
        toggle.t = vsync;
        nm = toggle.q;
```

```
        -- Shift in the data from ED[7  :4] - this is the main function
        -- of the EPLD.
        shift[].clk = !eclk;
        shift[ 7.. 4].d = ed[7.. 4];
        shift[11.. 8].d = shift[ 7.. 4].q;
        shift[15..12].d = shift[11.. 8].q;

        -- When the counter reaches its terminal count, latch the data
        --  onto  the  output  pins.    This  extra  level  of  latching
wouldn't
        -- be neccessary if the LCD had faster setup/hold times.
        EN = count[0].q AND count[1].q;
        data[15..0].ena = EN;
        data[15..0].clk = !eclk;
        data[15..4].d = shift[15..4].q;
        data[3..0].d = ed[7..4];

        -- Renam e the output pins to match the naming conventions of
        -- the LCD
        ud[7] = data[15].q;
        ud[6] = data[13].q;
        ud[5] = data[11].q;
        ud[4] = data[9].q;
        ud[3] = data[7].q;
        ud[2] = data[5].q;
        ud[1] = data[3].q;
        ud[0] = data[1].q;

        ld[7] = data[14].q;
        ld[6] = data[12].q;
        ld[5] = data[10].q;
        ld[4] = data[8].q;
        ld[3] = data[6].q;
        ld[2] = data[4].q;
        ld[1] = data[2].q;
        ld[0] = data[0].q;
END;
```

Appendix B - `nslib.h` library header

The following is a listing of `nslib.h`, the header file declaring all externally visible features of the `nslib` library.

```
/*
 * nslib.h --
 *
 *     Declarations of public and private features of nslib,
 *     a library for programs running on the ARM7500-based
 *     Netslate Development Board.
 *
 * Copyright (c) 1996 Ben Williamson.
 * All rights reserved.
 *
 * This file is part of nslib, a library used by programs
 * running on the Netslate Development Board.
 *
 * This software is released under the GNU Public License.
 * See the file COPYING for more information.
 */

#ifndef _NSLIB_H
#define _NSLIB_H

#include <ioregs.h>
#include <vidregs.h>


/* constants */

#ifndef EXTERN
# ifdef __cplusplus
#  define EXTERN     extern "C"
# else
#  define EXTERN     extern
# endif
#endif

#define NULL         0

#define IRQ_MAX_NUM  37

#define IRQ_INT2     0            /* INT2 rising edge */
#define IRQ_NINT1    2            /* nINT1 falling edge */
#define IRQ_FLYBACK  3            /* video frame flyback */
#define IRQ_POR      4            /* Power On Reset */
#define IRQ_TIMER0   5            /* 2MHz timer 0 */
#define IRQ_TIMER1   6            /* 2MHz timer 1 */
#define IRQ_ALWAYS   7            /* always active */

#define IRQ_NINT8    8            /* nINT8 active low */
#define IRQ_INT7     9            /* INT7 active high */
#define IRQ_NINT6    10           /* nINT6 active low */
#define IRQ_INT5     11           /* INT5 active high */
#define IRQ_NINT4    12           /* nINT4 active low */
#define IRQ_NINT3    13           /* nINT3 active low */
#define IRQ_KEYBT    14           /* keyboard transmit */
#define IRQ_KEYBR    15           /* keyboard receive */

#define IRQ_IOP0     16           /* IOP[0] active low */
#define IRQ_IOP1     17           /* IOP[1] active low */
#define IRQ_IOP2     18           /* IOP[2] active low  */
#define IRQ_IOP3     19           /* IOP[3] active low */
#define IRQ_IOP4     20           /* IOP[4] active low */
#define IRQ_IOP5     21           /* IOP[5] active low */
#define IRQ_IOP6     22           /* IOP[6] active low */
```

```
#define IRQ_IOP7     23            /* IOP[7] active low */

#define IRQ_MSER     24            /* mouse receive */
#define IRQ_MSET     25            /* mouse transmit */
#define IRQ_ATOD     26            /* A-to-D comparators */
#define IRQ_NEVENT1  27            /* nEVENT1 active low */
#define IRQ_NEVENT2  28            /* nEVENT2 active low */

#define IRQ_SOUND    36            /* sound DMA */

#define HIBYTE(x)    ((unsigned char)((x) >> 8))
#define LOBYTE(x)    ((unsigned char)(x))

#define Random(n)    (random() % (n))
#define Suspend()    {SUSMODE = 0x01;}

#define _bottomOfHeapLogical _end

/*
 * Macros to access video dimensions and call video functions
 * through the configuration struct:
 */

#define vidMaxX                    (vidConfigPtr->maxX)
#define vidMaxY                    (vidConfigPtr->maxY)
#define vidMaxCol                  (vidConfigPtr->maxCol)
#define vidBufSize                 (vidConfigPtr->bufSize)
#define VidStart()                 (vidConfigPtr->startProc)()
#define VidClear()                 (vidConfigPtr->clearProc)()
#define VidDrawPage(p)             (vidConfigPtr->drawPageProc)(p)
#define VidDisplayPage(p)          (vidConfigPtr->displayPageProc)(p)
#define VidFrameIsr                (vidConfigPtr->frameIsrProc)
#define VidSync()                  (vidConfigPtr->syncProc)()
#define VidShapeCursor(p)          (vidConfigPtr->shapeCursorProc) \
                                          ((p))
#define VidMoveCursor(x, y)        (vidConfigPtr->moveCursorProc) \
                                          ((x), (y))
#define VidGetCol(r, g, b)         (vidConfigPtr->getColProc) \
                                          ((r), (g), (b))
#define VidSetRGB(c, r, g, b)      (vidConfigPtr->setRGBProc) \
                                          ((c), (r), (g), (b))
#define VidPixel(x, y, c)          (vidConfigPtr->pixelProc) \
                                          ((x), (y), (c))
#define VidHLine(x1, x2, y, c)     (vidConfigPtr->hlineProc) \
                                          ((x1), (x2), (y), (c))
#define VidVLine(x, y1, y2, c)     (vidConfigPtr->vlineProc) \
                                          ((x), (y1), (y2), (c))
#define VidLine(x1, y1, x2, y2, c) (vidConfigPtr->lineProc) \
                                          ((x1), (y1), (x2), (y2), (c))
#define VidRect(x1, y1, x2, y2, c) (vidConfigPtr->rectProc) \
                                          ((x1), (y1), (x2), (y2), (c))
#define VidTriangle(x1,y1,x2,y2,x3,y3,c) \
                                   (vidConfigPtr->triangleProc) \
                                          ((x1), (y1), (x2), (y2), \
                                          (x3), (y3), (c))
#define VidCircle(x, y, r, c)      (vidConfigPtr->circleProc) \
                                          ((x), (y), (r), (c))
#define VidFillRect(x1,y1,x2,y2,c) (vidConfigPtr->fillRectProc) \
                                          ((x1), (y1), (x2), (y2), (c))
#define VidFillTriangle(x1,y1,x2,y2,x3,y3,c) \
                                   (vidConfigPtr->fillTriangleProc) \
                                          ((x1), (y1), (x2), (y2), \
                                          (x3), (y3), (c))
#define VidFillCircle(x, y, r, c)         (vidConfigPtr-
>fillCircleProc) \
                                          ((x), (y), (r), (c))

#define VidChar(c, x, y, f, b)     (vidConfigPtr->charProc) \
                                          ((c), (x), (y), (f), (b))
#define VidText(s, x, y, f, b)     (vidConfigPtr->textProc) \
                                          ((s), (x), (y), (f), (b))
```

```
typedef void (VidStartProc)(void);
typedef void (VidClearProc)(void);
typedef void (VidDrawPageProc)(int page);
typedef void (VidDisplayPageProc)(int page);
typedef void (VidFrameIsrProc)(void);
typedef int  (VidSyncProc)(void);
typedef void (VidShapeCursorProc)(char *xpmPtr[]);
typedef void (VidMoveCursorProc)(int x, int y);
typedef int  (VidGetColProc)(int red, int green, int blue);
typedef void (VidSetRGBProc)(int col, int red, int green, int blue);
typedef void (VidPixelProc)(int x, int y, int col);
typedef void (VidHLineProc)(int x1, int x2, int y, int col);
typedef void (VidVLineProc)(int x, int y1, int y2, int col);
typedef void (VidLineProc)(int x1, int y1, int x2, int y2, int col);
typedef void (VidRectProc)(int x1, int y1, int x2, int y2, int col);
typedef void (VidTriangleProc)(int x1, int y1, int x2, int y2,
                               int x3, int y3, int col);
typedef void (VidCircleProc)(int cx, int cy, int r, int col);
typedef void (VidFillRectProc)(int x1, int y1, int x2, int y2,
                               int col);
typedef void (VidFillTriangleProc)(int x1, int y1, int x2, int y2,
                               int x3, int y3, int col);
typedef void (VidFillCircleProc)(int cx, int cy, int r, int col);
typedef void (VidCharProc)(unsigned char ch, int x, int y,
                               int fgcol, int bgcol);
typedef void (VidTextProc)(char *s, int x, int y,
                               int fgcol, int bgcol);


typedef struct {
       int                 maxX;
       int                 maxY;
       int                 maxCol;
       int                 bufSize;
       VidStartProc        *startProc;
       VidClearProc        *clearProc;
       VidDrawPageProc     *drawPageProc;
       VidDisplayPageProc  *displayPageProc;
       VidFrameIsrProc     *frameIsrProc;
       VidSyncProc         *syncProc;
       VidShapeCursorProc  *shapeCursorProc;
       VidMoveCursorProc   *moveCursorProc;
       VidGetColProc              *getColProc;
       VidSetRGBProc              *setRGBProc;
       VidPixelProc        *pixelProc;
       VidHLineProc        *hlineProc;
       VidVLineProc        *vlineProc;
       VidLineProc         *lineProc;
       VidRectProc         *rectProc;
       VidTriangleProc     *triangleProc;
       VidCircleProc              *circleProc;
       VidFillRectProc     *fillRectProc;
       VidFillTriangleProc *fillTriangleProc;
       VidFillCircleProc   *fillCircleProc;
       VidCharProc         *charProc;
       VidTextProc         *textProc;
} VidConfig;

/* data structures */

typedef unsigned int uint;
typedef unsigned long int size_t;
typedef volatile unsigned int ureg;
typedef unsigned char byte;

/*
 * Pointer to function with no arguments returning int
 */
typedef int (*PFI)(void);
```

```
/*
 * Pointer to function with no arguments returning void
 */
typedef void (*PFV)(void);

/*
 * Pointer to function with a pointer argument returning void
 */
typedef void (*PTV)(void *);


/*
 * A block on the heap
 */

typedef struct _HeapBlock {
       uint length;                /* bytes including header */
       struct _HeapBlock *next;    /* next block in heap */
       struct _HeapBlock *prev;    /* previous block in heap */
} _HeapBlock;


/*
 * Declarations of memory map constants from memmap.S:
 */

EXTERN int    _topOfStackPhysical;
EXTERN int    _l1TableStartPhysical;
EXTERN int    _bottomOfHeapLogical;
EXTERN int    _topOfHeapLogical;
EXTERN int    _bottomOfStackLogical;
EXTERN int    _topOfStackLogical;
EXTERN int    _l1TableStartLogical;
EXTERN int    _topOfMemoryLogical;
EXTERN int    _l1TableStart;
EXTERN int    _romStartPhysical;
EXTERN int    _memStartPhysical;
EXTERN int    _ram0StartPhysical;
EXTERN int    _ram1StartPhysical;
EXTERN int    _ram2StartPhysical;
EXTERN int    _ram3StartPhysical;
EXTERN int    _pcioStartPhysical;
EXTERN int    _armioStartPhysical;
EXTERN int    _sioStar tPhysical;
EXTERN int    _vidStartPhysical;
EXTERN int    _memStartLogical;
EXTERN int    _ram0StartLogical;
EXTERN int    _ram1StartLogical;
EXTERN int    _ram2StartLogical;
EXTERN int    _ram3StartLogical;
EXTERN int    _pcioStartLogical;
EXTERN int    _armioStartLogical;
EXTERN int    _sioStartLogical;
EXTERN int    _vidStartLogical;
EXTERN int    _romStartLogical;

EXTERN _HeapBlock *_heapStart;
EXTERN _HeapBlock *_heapLimit;

/*
 * Video buffers, cursors and fonts:
 */
EXTERN char _vid1Display0Logical;
EXTERN char _vid1Display1Logical;
EXTERN char _vid1Display0Physical;
EXTERN char _vid1Display1Physical;

EXTERN char _vid2Display0Logical;
EXTERN char _vid2Display1Logical;
EXTERN char _vid2Display0Physical;
EXTERN char _vid2Display1Physical;
```

```
EXTERN char _vidCursorLogical;
EXTERN char _vidCursorPhysical;

EXTERN char *cursorPixmap[];
EXTERN unsigned char fixedFont[];

/*
 * Other global variables and constants
 */
EXTERN VidConfig *vidConfigPtr;

EXTERN VidConfig vid1Config;
EXTERN VidConfig vid2Config;

EXTERN volatile int mouseX, mouseY, mouseB;

EXTERN uint const mapTbl[];
EXTERN uint const unmapTbl[];


/* procedures */

/*
 * From debug/
 */

EXTERN uint   _getcpsr(void);
EXTERN void   _setcpsr(uint cpsr);
EXTERN uint   _getsp(void);
EXTERN void   halt(void);
EXTERN char *        number(unsigned int n);
EXTERN void   print(const char *s);
EXTERN void   putchar(char c);

/*
 * From intrpt/
 */

EXTERN void   IntEnable(void);
EXTERN void   IntDisable(void);
EXTERN void   IrqInit(void);
EXTERN void   IrqEnable(int irqNum);
EXTERN void   IrqDisable(int irqNum);
EXTERN PFV    IrqInstall(int irqNum, PFV isrProc);
EXTERN void   IrqDispatch(void);

/*
 * From mem/
 */

EXTERN void   _MmuInitTable(void);
EXTERN void   HeapInit(void);
EXTERN void *        malloc(size_t size);
EXTERN void   free(void *ptr);

/*
 * From misc/
 */

EXTERN void   _main(void);
EXTERN uint   random(void);

/*
 * From mouse/
 */

EXTERN void   MouseInit(void);
EXTERN void   MouseIsr(void);

/*
 * From serial/
 */
```

```
EXTERN void   SerialInit(int baud);
EXTERN void   SerialSend(char ch);
EXTERN char           SerialRecv(void);
EXTERN int    SerialReady(void);
EXTERN void   SerialIsr(void);

/*
 * From string/
 */

EXTERN void   _clear(void *start, int len);
EXTERN void   bzero(void *s, int n);

/*
 * From video/
 */

EXTERN void   VidInit(VidConfig *configPtr);


#endif /* _NSLIB_H */
```