VM LABS

The Hitchhiker's Guide to NUON™

An Overview of the VM Labs Development System

Revision 0.87
12-Feb-01

Note: This document is continually updated to reflect the current state of the NUON development system hardware and software. If you have a version that is more than five or six months old, it is likely out of date.

Please address comments or report errors to Mike Fulton at VM Labs (EMAIL: mfulton@vmlabs.com).

VM Labs, Inc.
520 San Antonio Road
Mountain View, CA 94040

Tel: (650) 917-8050
Fax: (650) 917-8052

Table of Contents

1. Introduction

1.1 Unpacking The Development System

Congratulations on becoming a NUON developer. You've just received your development system and have opened up the boxes. What's next?

The first thing you should do is to make sure that nothing is missing. The tables below allow you to check off the items that should be included in the package:

Item	Check
NUON Development Unit in ATX-style PC case	
Documentation Package (This may not be included if you have received other NUON development systems from VM Labs in the past.)	
AC Power Cable	
DB-9 to DB-9 Serial Cable	

Figure 1-1 — Items Included With NUON Development System

Please note that the controller packaged with your development system may be either a standard production NUON controller or a prototype controller. Each controller comes with specific other items as shown in the table below.

Item	Check
Prototype Controller	
Controller Adapter Box	
Cable: Controller Adapter Box to NUON (DB-9 to DB-9 cable with gender adapter)	
OR	
NUON Game Controller	
NUON Controller to USB Connector[1] Adapter Cable (This item may not be included, depending on what connector style is used on the NUON development system.	

Figure 1-2 — Game Controller Items Included With NUON Development System

[1] Some revisions of the NUON development system use a USB style connector for game controllers, so an adapter cable may be provided. However, it is not an actual USB port, so do not attach USB peripherals to your NUON development system via this connector or damage may result.

1.2 What's in the *Hitchhiker's Guide*?

Chapter 2 will introduce you to some details about VM Labs Developer Support, such as how to contact support personnel, how to log onto the support FTP site, and how to download the latest versions of the SDK and other files.

Chapter 3 describes the ports found on your NUON development system and talks about the differences between different revisions of the system.

In chapters 4 through 6, we'll walk you through the steps of getting your system put together and connecting it to your host PC and TV.

Chapters 7 & 8 discuss the system firmware and debugging/communications interface.

Chapters 9 through 11 will introduce the various tools of the SDK, and provide a basic reference guide for the tools you'll most often. Chapter 1 will discuss where to find more in-depth documentation for the various tools and libraries.

In chapter 13, we demonstrate how to build and execute a sample program, and help you with trouble-shooting.

Chapter 14 discusses the NUON file server, which can be used for communication and data transfer between the client program running on the NUON system and the host PC.

Chapter 15 has a number of programming guidelines which should be observed.

Chapter 16 contains Frequently Asked Questions (FAQ) compiled from developer inquiries on a broad range of subjects ranging from setup issues to programming.

Chapter 17 provides a basic glossary of terms that you'll see used often throughout the NUON developer documentation.

1.2.1 Deletions In This Revision

The **Hardware Bug List** and **Software Bug List** sections that came at the very end of the previous revisions have been removed. It became clear that those lists should be moved to another place where they could be maintained more efficiently. That information is now maintained in a separate document.

1.2.2 Online Versions of the *Hitchhiker's Guide*

If you are reading this from a printed version of the *Hitchhiker's Guide*, then make sure to check the online documentation section of the VM Labs Developer Website. The *Hitchhiker's Guide* is updated often and a newer version may be available in Adobe Acrobat PDF format.

1.3 “Merlin” –vs– “NUON” –vs– “Project X”

Prior to the announcement of the ***NUON*** name in October 1998, VM Labs used the name “Merlin” to refer to both the development system and the underlying technology. The term “Project X” was also used to refer to the platform in early press notices and news items.

These terms refer to the same thing as “NUON”. In our documentation, we have changed most references to “Merlin” but you may still find places we’ve missed. Therefore, we ask that you please always keep in mind that “NUON”, “Merlin”, and “Project X” all refer to the same thing.

1.3.1 Common Acronyms

You may find the following acronyms used interchangeably throughout our documentation:

NDK — This stands for “NUON Developer Kit” and generally refers to anything in the entire development system, hardware and software.

MDK — Same thing as “NDK” except this is the older usage with “Merlin” instead of “NUON”.

NDS — This stands for “NUON Developer System” and usually refers specifically to the hardware portion of the development system.

MDS — Same thing as “NDS” except this is the older usage with “Merlin” instead of “NUON”.

1.4 “Aries” –vs– “Oz”

Oz is the name used to refer to the MMP-L3A revision of the NUON chip. This is the original version of the NUON chip that was used in the first generation of development systems. As would be expected, the ***Oz*** revision of the chip had a number of bugs where something did not work correctly. For the most part, these bugs were relatively small and could be worked around in software without too much difficulty.

Aries is the name used to refer to the MMP-L3B revision of the NUON chip. This is the latest version of the chip used in current versions of the development system as well as in production hardware. The ***Aries*** version fixes most of the bugs from the ***Oz*** revision.

Aries 2 is the name used to the MMP-L3C revision of the NUON chip. It fixes a few more bugs and has some changes in the external memory interface (for cost reduction).

Aries 3 is the name used to the MMP-L3D revision of the NUON chip, which is not yet available.

For a time, the tools and libraries in the SDK were designed to create software that would run on either Aries or Oz systems. In many cases, this required that a library sacrifice some performance in order to provide code that would work on both systems. In the case of the audio libraries, separate versions for each chip revision were required.

As of April 1999, new releases of the SDK will no longer provide libraries or tools that are designed to be backwards compatible with Oz-based systems. Programs created with these SDK releases will probably not work right on Oz-based systems.

The Aries 2 and Aries 3 chip revisions does not require any SDK changes.

We have attempted to update development systems using the older Oz chip revision. If you still have an Oz-based system, please contact VM Labs immediately to arrange for an updated system.

2. VM Labs Developer Support

2.1 Contacting Developer Support

You may contact the VM Labs Developer support staff via telephone, fax, E-Mail, or regular mail as follows:

E-Mail:	devsupport@vmlabs.com
Phone:	+1 (650) 917-8050 (see note below)
Fax:	+1 (650) 917-6610 (see note below)
Mailing Address:	VM Labs Attention: Third Party Developer Support 520 San Antonio Rd. Mountain View, CA 94040-1217

Additional contact information may be found on the VM Labs Developer Support Website. See section 2.2 below for more information.

2.1.1 Sending E-Mail to Developer Support

One of the best ways to contact developer support is via E-Mail. In order to help ensure a swift and accurate response, please always try to provide the following information in your message:

- Your name
- Your company
- The name of your project
- Your return E-Mail address
- Your telephone number (In some cases, our support engineers may decide that calling you would be more efficient and effective than an E-Mail reply.)
- As much detail as possible about the problem you are having or the question you want answered.

Please do not assume that you do not have to include this information because "they already have it". You may be correct that **someone** at VM Labs has this information, but that will not necessarily be the person that will be responding to your message. If you leave out this information, it may delay your response.

2.1.2 Response Time

When you send an E-Mail message or fax, our support staff will usually be able to respond within one business day, at least to acknowledge receipt. Please keep in mind that holidays and certain special events may sometimes affect response time.

Finally, while we do our best to avoid it, things do occasionally slip through the cracks. If you do not receive at least an acknowledgement of your inquiry within a few business days, please try again.

2.1.3 Contacting Non-Support Personnel at VM Labs

Occasionally, to help resolve a problem, our support engineers may put you into direct contact with another engineer at VM Labs who is not part of the regular support team.

In such cases, please keep in mind that non-support personnel are normally quite busy with their normal everyday assignments. Unlike the support staff, they are not accustomed to dealing with outside developers on a regular basis and cannot always devote their complete attention to your problem. They will do their best, but may not be able to respond as quickly as you would like.

When you contact non-support staff regarding an issue, please do your best to keep the original support staff member in the communications loop. Please make sure the support staff receives copies of any correspondence. This will help to avoid any problems and ensure that confusion is kept to a minimum.

2.2 Developer Support Online

Aside from EMAIL, VM Labs offers a variety of online support services including a private developer-only website, FTP site, and newsgroups. Each of these services is discussed later in this section.

2.2.1 Developer Web Site

The VM Labs Developer Support website is located at:

```
http://developer.vmlabs.com
```

The developer website is password protected. Except for a few pages at the top level, you must have an account to access it. The main page of the website includes a link to a page where you must fill out and submit a form to request a new account. Alternately, you may send an EMAIL message containing the same information requested by the form.

Please note that no accounts may be activated without this process.

Once you have submitted your account request, the information must be validated before the account can be activated. Assuming there is no difficulty in validating your information, it typically takes about one business day to activate your account. However, this depends on individual circumstances and some accounts may take longer to validate.

All information obtained from the VM Labs Developer Support website is considered confidential and proprietary to VM Labs, Inc.

2.2.1.1 Other VM Labs Websites

Other websites maintained by VM Labs are open to the public. Information found on these sites is not considered confidential and may be freely discussed.

The VM Labs corporate website is located at:

```
http://www.vmlabs.com
```

The official NUON website for consumers is located at:

```
http://www.nuon.tv
```

2.2.2 Developer Support FTP Site

There is an FTP site that goes along with the developer website. This may used to download files which are available through the website, or it may be used for developers to send files to VM Labs. The URL is:

```
ftp://developer.vmlabs.com
```

To login with the FTP site, please use the same account information as for the developer web site.

Most developers will only have access to the ***develop*** and ***develop/upload*** folders of the FTP site. However, some developers may have a private folder in the ***develop/3rdparty*** folder where they can receive special downloads or send uploads to VM Labs. This is ordinarily done only when there is a significant amount of traffic back and forth.

Developers may be given access to other folders on the FTP site as required by individual circumstance.

2.2.2.1 Legacy FTP Site

Prior to the opening of the developer website, VM Labs maintained a different FTP site for developers. This URL for this FTP site is:

```
        ftp://204.31.130.4
```

This FTP site is separate from the one at *developer.vmlabs.com*, including account information.

As of this writing, the older site is still active, but access is no longer granted to 3rd party software developers. Currently, access is granted only to OEMs and 2nd party companies.

The older FTP site will eventually be phased out. There is no guarantee that all items made available on the new website will be made available through the older FTP site as well.

2.2.2.2 Downloading Files From the FTP Site

Always configure your FTP program for a BINARY transfer when downloading files from the FTP site. With many FTP programs, the default transfer mode is ASCII, and this may cause changes to the file during transfer.

Please note that while many web browsers have basic FTP support built-in, they generally do not support the whole range of options that a dedicated FTP program would have. In particular, they may not allow you to login using your account information.

If you don't have a separate FTP program, don't forget that Windows comes with a simple console-based FTP client program called "FTP". It's not pretty, but it will serve the purpose.

Simply open a command shell window and enter the command FTP, followed by the name of the FTP site. For example:

```
        ftp developer.vmlabs.com
```

. The table below lists the main commands you'll need.

Windows FTP Program Basic Commands	
ASCII	Switch to ASCII download mode
BINARY	Switch to BINARY download mode
CD <directory>	Changes to the specified <directory>

Windows FTP Program Basic Commands	
DIR <specification>	Lists files matching the given <specification>
GET <file>	Downloads the specified <file> to the current local directory
HELP [command]	By itself, lists available commands. If a particular command is specified (i.e. "help get") then more detailed help on that command is shown.
OPEN <address>	Opens the FTP site specified by <address>, which may be either an IP address or an URL.
PUT <file>	Uploads the specified <file> (by default in the current local directory) to the current directory on the FTP site.

2.2.2.3 Uploading To The FTP Site

To upload a file to the FTP site, first change to the ***/develop/upload*** folder (unless you have been granted access to a different location). In your FTP program, select "binary" transfer mode (rather than ASCII). Finally, start the upload.

Please note that by default, you will only have write access to the ***/develop/upload*** directory. That means you won't be able to do anything except upload, not even get a directory of existing files.

If you get an error message saying "access denied" when you try to upload, there is a good chance that the cause is simply that a file already exists with the specified name. The system does not allow you to overwrite an existing file. This may happen if a previous attempt to upload the file does not succeed. The solution is to select a new filename and try again.[2]

After uploading your file, please always remember to send an EMAIL message to the appropriate parties at VM Labs, including ***webmaster@vmlabs.com***. Make sure your message includes the filename of the uploaded file. Also include the filenames for any failed upload attempts so that the Webmaster can remove those files.

2.2.3 Developer Newsgroups

There is also a newsgroup server that features developer-only message bases. The URL is:

```
news://developer.vmlabs.com
```

Please note that the newsgroup server only maintains private developer-only areas. There is no danger of the general public reading and responding to messages. You may feel free to discuss technical issues with VM Labs or with other developers.

[2] Unfortunately, there is no facility at this time for resuming an upload.

In general, we would prefer that technical questions be submitted through the newsgroups rather than EMAIL whenever possible. For one thing, your message will be seen by all of the developer support staff instead of just one individual. Your message will probably also be seen by other engineers at VM Labs who may offer a response. And you might also get a response from another developer.

Of course, if your question includes technical details you cannot discuss with other developers, you still have the option of sending private EMAIL to the developer support staff at VM Labs.

2.2.3.1 Configure Your Newsgroup Reader Software

To login with the newsgroup server, please use the same account information as for the developer website and FTP site.

It is very important that you configure your newsgroup reader software to login using your account information. If you do not, your reader software may give you a message such as "access denied" or perhaps another error message that is not very specific about the cause of the problem.

2.3 Obtaining The NUON SDK

The NUON SDK is available to authorized 3rd party developers for downloading through either the developer website or the developer FTP Site.

On the website, simply follow the links to the download page, along with any other instructions shown. This is the preferred method.

On the developer FTP site, or on the legacy FTP site, the SDK files are located in the DEVELOP/SDK directory. The filename should be something like:

```
SDK_0326.DES
```

The exact filename will depend on the SDK revision date and encryption method.

In this example, the "0326" portion of the filename means March 26th. Of course, this refers to a specific release and is subject to change as newer revisions will have filenames that indicate other dates. However, the SDK folder will always contain only the current release.

If you have any doubts about which file is the most recent, you should use the website rather than the FTP site, as the website will include a text description that should help you figure everything out.

2.3.1 File Encryption

Most of the files stored on the developer website and FTP site are encrypted in order to prevent unauthorized access. This includes all of the SDK and related demo files.

Please note that the encryption method used may change at any time. At any given time, you can find detailed information about the current setup on the developer website.

2.3.2 DES Encryption

A "DES" filename extension on a file indicates that the file is DES-encrypted. You may decrypt such a file using the DES utility, which is also available on the FTP site at:

```
TOOLS\DES.EXE
```

2.3.2.1 Decrypting the SDK File

For a DES-encrypted SDK archive, the command line used for decryption is:

```
des -3Dk SecretPassword <inputfile> <outputfile>
```

where *SecretPassword* must be replaced by the real password, which is provided to you separately. The *inputfile* parameter indicates the filename of the encrypted file. The *outputfile* parameter indicates the name of the decrypted file that will be created. For example:

```
des -3Dk NUON4Ever vm990510.des vm990510.zip
```

Please note that the command line options and password for the DES utility are case-sensitive. The command line above specifies that "**NUON4Ever**" is the password, opens the **VM990510.DES** file, and creates the **VM990510.ZIP** file, which can then be used with any standard Windows 95 ZIP file tool.

Please note that if you do not specify the output file on the command line, the output of the DES tool is sent to the standard console output device. The most likely result is a long stream of garbage characters.

2.3.3 Installing the SDK

We're not going to provide details about installing the SDK just yet, because there are other important issues that we should discuss first. However, if you want to peek ahead, please see section 9.2, ***Tools Installation***.

2.4 Obtaining Other Files

Please note that there may be other files containing sample code and/or demos available from the website or FTP site. For example, the website features a number of demo programs (mostly without source code) which are non-essential, but which may be interesting.

2.4.1 Decrypting Other Files

Please note that all files on the FTP site or website posted on the same date will use the same decryption password. For example, SDK0209.DES and DEMO0209.DES share the same password. Therefore, simply apply the instructions given in section 2.3.2.1 above, using different filenames on the command-line as appropriate.

If you download a file from the website and do not have the proper decryption password, please contact developer support via telephone or EMAIL.

2.4.2 Personalized Downloads

Please note that from time to time, the FTP site may have files intended for specific developers. For example, a developer may send some source code to a developer support engineer who will make changes and post a revised version for the developer to download.

Such personalized downloads will always be encrypted with a unique password to ensure confidentiality, so you should never download a file unless you know the file is intended for you. If you aren't the intended recipient, then you will not be provided with the password, and you will not be able to decrypt these files.

2.4.3 Obtaining Technical Notes

The VM Labs Support FTP Site contains a variety of technical notes that may be useful and interesting. Please look at the website on a regular basis for updates and new materials.

2.5 Things to Keep in Mind About Developer Support

There are a number of things about the whole developer support process that are important to know and remember. When contacting developer support, please keep them in mind. This will make the entire developer support process much smoother for everybody.

2.5.1.1 “We are both under non-disclosure”

Sometimes developers are reluctant to share information or code fragments that are required to find the solution to a particular problem. This might be because your project hasn’t been publicly announced, or for many other reasons.

Please always keep in mind that the relationship between VM Labs and a developer is covered under a mutual non-disclosure agreement. Our support staff is not going to discuss your confidential information with the press, the public, or even other people within VM Labs who do not have reason to know about it.

If there’s any doubt about which information is “confidential” and which might be considered “public”, then just let us know which is which. For example, details about the inner workings of your code are always going to be “confidential” and there’s no doubt about that.

On the other hand, just the fact that you’re working on a particular project could be either “public” or “confidential” depending on a number of factors.

When in doubt, let us know in advance so that we can take appropriate care. That way, there should be no reason for you to be reluctant to share information about your project with our developer support staff.

2.5.1.2 “I know it sounds stupid, but please try it anyway?”

Frequently a developer support engineer will make a suggestion that sounds very simplistic and unlikely to solve the problem. If the suggestion is more than minimally time consuming, the developer may be reluctant to try it.

It’s amazing how often those “stupid” suggestions lead directly or indirectly to the solution.

Experience shows that a very high percentage of developer support issues are based on very simple problems or mistakes that are easy to overlook. Finding the cause of a problem is often a matter of collecting as much data as possible. Even “stupid” suggestions will help eliminate possibilities and the results often provide data that will help determine the real problem.

2.5.1.3 RTFM: Read the fine manual

We’ve all heard “R-T-F-M” in response to a problem. It’s a cliché, but it got to be one because it’s true so often. A surprising number of times, the answer to your question is right in front of you, somewhere within the documentation at hand.

We acknowledge that finding a specific piece of information in a sea of documentation can sometimes be difficult, so please don’t hesitate to contact developer support whenever you feel it’s necessary. But always keep in mind that if the answers are in the documentation, you'll find them faster yourself than if you contact developer support.

It's difficult to commit large amounts of technical documentation entirely to memory and probably impractical to even try. However, it's really helpful if you make an effort to at least scan through all of the documentation so you have a basic idea of what subjects are covered, and in which section of the docs they are located. This will help you more quickly locate the information you need when you need it.

2.5.1.4 "The support guy's asking me too many questions, instead of answering mine!"

Sometimes when a developer asks some questions, instead of giving the answer they wanted, the support engineer asks questions: What they are doing? And why are they doing it that way? The developer then gets annoyed at being grilled for details, and often suspects one or more of the following:

- They just don't want to give me that information. Maybe it's "undocumented".
- They think I'm doing something "illegal" and they want to catch me at it.
- They don't think I know what I'm doing.

The result is a situation full of distrust and frustration on both sides, which doesn't do anybody any good.

In most cases, providing the best answer to the developer's original question depends greatly on the context of the situation. Maybe the original question is not very precise, and more detail is needed in order to distinguish one problem from another. Perhaps the description is unusual in some fashion, and more detail is needed in order for things to make sense.

The main goal of a developer support engineer is always to help developers past their problems. There are two components involved in accomplishing this goal: first, figuring out the solution to problems that pop up for the first time; second, giving out solutions to problems that have previously hit other developers.

Whenever a problem comes up for the first time, the developer support engineer has to try to help solve the problem, but must also prepare for the possibility that the same problem will eventually happen to someone else. Different people might describe the same problem in drastically different ways. Without details, it can be difficult to determine that they are talking about the same basic problem.

That's why we ask so many questions.

There may be rare occasions when we may suspect the developer is doing something “illegal” or is trying to access something that is undocumented. Such occasions are very much in the minority, but they do occur occasionally.

Please keep in mind that we’re not trying to catch a developer doing something wrong in order to slap them on the wrist or to berate them for not following the rules. Our goal is to make sure the developer’s software will operate properly and efficiently, and to avoid compatibility problems with different pieces of hardware.

This page intentionally left blank.

3. NUON Development System Revisions

Aside from the issue of which revision of the NUON chip is used, there are currently two revisions of the NUON Development System in common use. The unique attributes of each revision are described in separate sections below.

3.1 Revision 4 System

Revision 4 systems are no longer being shipped, but in case you have one, we still include the information here. These systems are shipped in AT-style PC computer cases. All of the system ports are accessed through a hole in the back of the case.

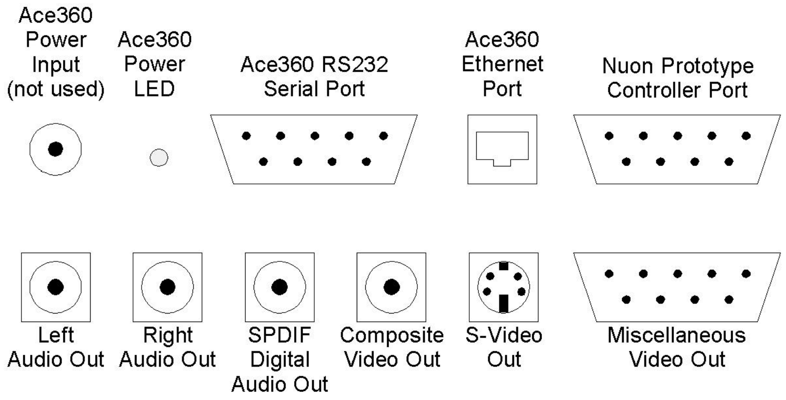

Figure 3-1 — NUON Development System Ports (Revision 4)

The “ACE360” ports along the top row are used to connect your NUON to your host computer system. These ports are part of a daughtercard that is mounted on the motherboard. This daughtercard contains the Ethernet interface and serial port, as well as the ACE360 processor that controls these devices.

The controller port at the right end of the top row allows you to connect a prototype NUON controller.

The ports along the bottom row are used to connect your NUON system to a television, monitor, and/or stereo system.

The revision 4 system has three backplane expansion slots, but they are normally unused on a standard system.

3.2 Revision 5 System

The revision 5 system is shipped in a ATX-style PC computer case. Most of the system ports are accessed through the hole on the back of the case.

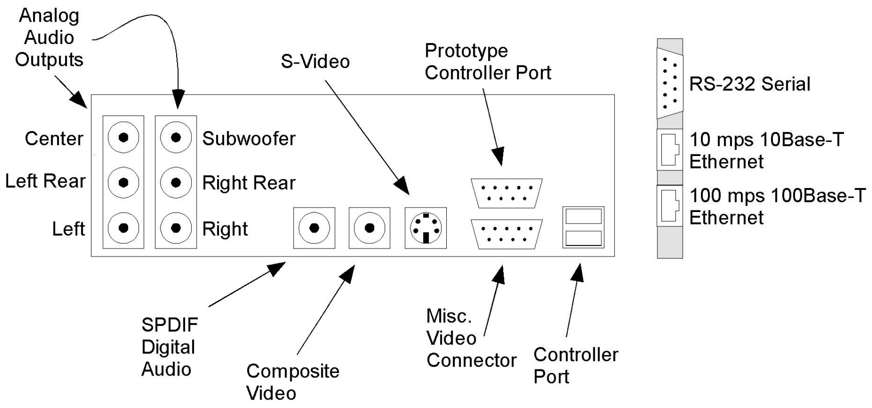

Figure 2 — NUON Development System Ports (Revision 5)

Revision 5 systems use a backplane expansion board for the interface card that contains the Ethernet interface and serial port. This interface card uses the PowerPC 860 processor instead of the ACE360 processor used on revision 4 systems.

Please note that the 100Base-T Ethernet port is not yet functional and may not even be present on all systems. Use the 10Base-T port until you hear otherwise. Also, some interface cards may not have the second connector for 100Base-T. If there is only one connector, it will be for 10Base-T.

The ports on the left side are used to connect your NUON system to a television, monitor, and/or stereo system. The ports along the right side are used to connect game controllers to your NUON system.

Depending on the specific revision of your system, your controller ports will either be genuine NUON connectors as found on a regular consumer machine, or they will be USB connectors. However, in the latter case, it's just a USB connector, not a USB port. Do not connect USB peripherals to these connectors.

3.3 Development System Expansion Cards

Please note that while it uses a similar connector and form factor, the expansion cards used by NUON development systems are not PCI cards and will not work in any other type of device, nor will a PCI card intended for another system work in the NUON system.

4. NUON A/V Connections

The back of your NUON development system contains a variety of connectors. In this section, we will discuss the connection of the A/V connectors to your TV, monitor, and/or stereo system.

We'll only show illustrations for revision 5 development systems. Please refer to section 1 for details on port locations.

4.1 Connecting NUON to your TV or monitor

The NUON system uses standard RCA cables for audio and composite video outputs. If your TV or monitor supports it, you may use the S-Video output for improved video quality. These cables may be obtained at most audio/video electronics dealers.

The NUON development system does not include RF modulated output for connection to a television's antenna leads. Your television or monitor must have composite video or S-Video inputs, or you must use a separate RF modulator obtained from your local electronics retailer.

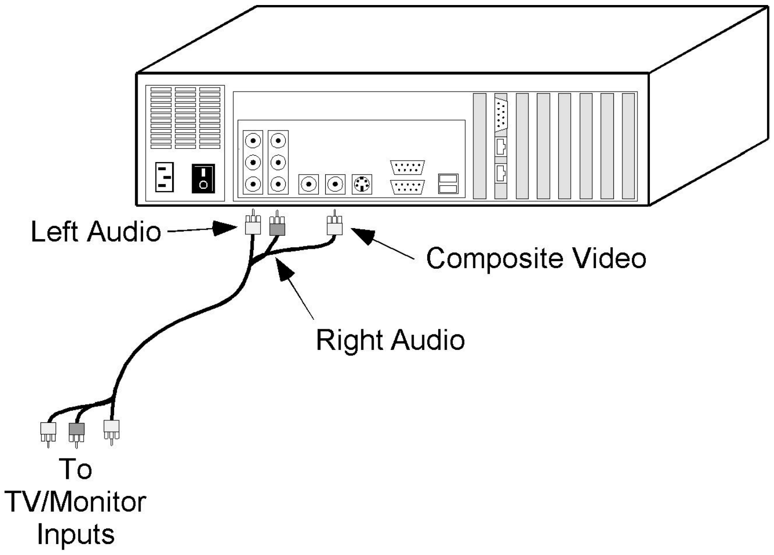

Figure 4-1 — Standard Video & Audio Connections

Warning: Do not connect the SPDIF Digital Audio output on the NUON to an analog audio input on your television, stereo, or other device. Doing so may result *in speaker damage. Connect this output signal only to a coaxial SPDIF digital audio input.*

4.1.1 NTSC & PAL

NUON's video display system is configured by the system BIOS whenever the system is powered-up or reset. Currently, the display is always set for standard NTSC video output with a 60 Hz refresh rate.

The ability to configure the video display for PAL mode with a 50 Hz refresh rate is not available at the time of this writing, but this may have changed by the time you read this. Please be alert for updates and if necessary, contact VM Labs Developer Support for more information.

4.1.2 Building your own video cable

You may wish to build your own cable to connect the NUON Development System to a monitor without standard composite or S-Video inputs (for example, a SCART monitor). Here are the pin-outs of the 9-pin ***Miscellaneous Video Out*** connector:

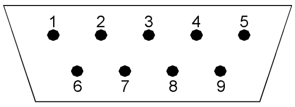

Figure 4-2, Miscellaneous Video Out Connector

Pin:	Signal:
1	Composite video (used as sync for RGB SCART)
2	Composite sync (TTL levels, for Amiga-type monitors)
3	Blue
4	Green
5	Red
6	Ground
7	+12V
8	+3.3V
9	Ground

4.1.3 Connecting NUON to Your Stereo

If you are connecting your NUON development system to a stereo system, please note that the SPDIF digital output is used only by DVD movie playback at this time. In order to hear the sound produced by games, the SDK sample programs, and so forth, it will be necessary to use the analog audio outputs.

Warning: Do not connect the SPDIF Digital Audio output on the NUON to an analog audio input on your television, stereo, or other device. Doing so may result *in speaker damage. Connect this output signal only to a coaxial SPDIF digital audio input.*

4.1.4 DVD Movie Playback on the Development System

At this time, the NUON development system now includes the firmware required to play DVD movie discs. However, because the system is first a development system, the process is not the same as if it were a standard DVD player.

To play a DVD-Video disc, insert the disc into your NUON dev system and cycle the power. So long as you don't have a very old firmware revision installed in your system, you'll see a screen containing a red NUON logo and some text.

At this point, if you press "A" on controller 1, the system will begin playing the disc.

Please note that because a development system does not have a front panel or remote control like a standard production DVD player, you will have to control playback using the game controller. We regret that it is not practical for us to include a reference of which buttons perform which functions because this may depend on the specific firmware revision involved. Please experiment with different button combinations and you'll figure it out.

This page intentionally left blank.

5. Connecting the NUON Controller

There are two different methods of connecting a controller to a NUON development system, depending on the controller itself and the development system revision. This chapter will cover both methods.

5.1 Connecting A Production NUON Controller

Please note that on some revisions of the NUON development system, USB-style connectors are used for game controllers. However, please be aware that these are not USB ports.

Never connect USB peripherals to a NUON development system. Doing so may damage the peripheral and the development system.

If your development system has USB-style connectors, connect the controller to the supplied adapter cable, and then plug the other end of the adapter cable into the USB-style connector. Port 0 is the one on top. Port 1 is the one on the bottom.

If your development system has genuine NUON connectors, simply plug in the controller directly. Port 0 is the one on the left. Port 1 is on the right.

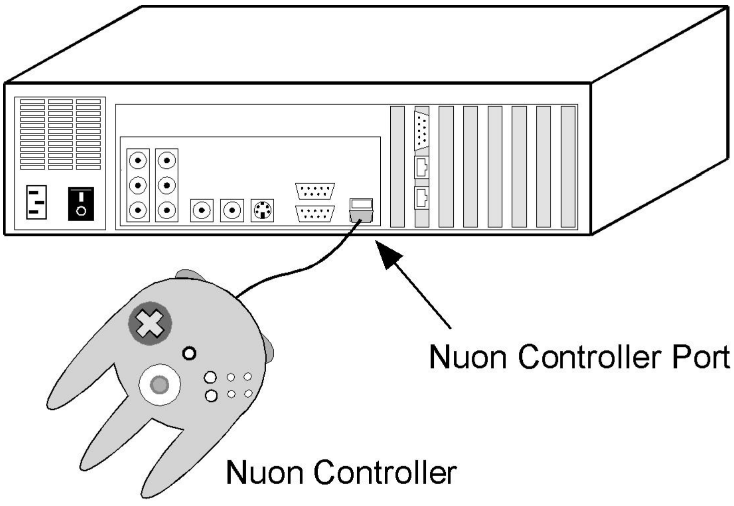

Figure 5-1 — Connecting A Controller

5.2 Controller Prototypes & The Interface Box

Controller prototypes do not have the electronics required to encode the data from the controller inputs and transmit it to the NUON system. These controllers simply contain switches and analog joysticks that must be connected to a separate interface box that will take the data and finish the job.

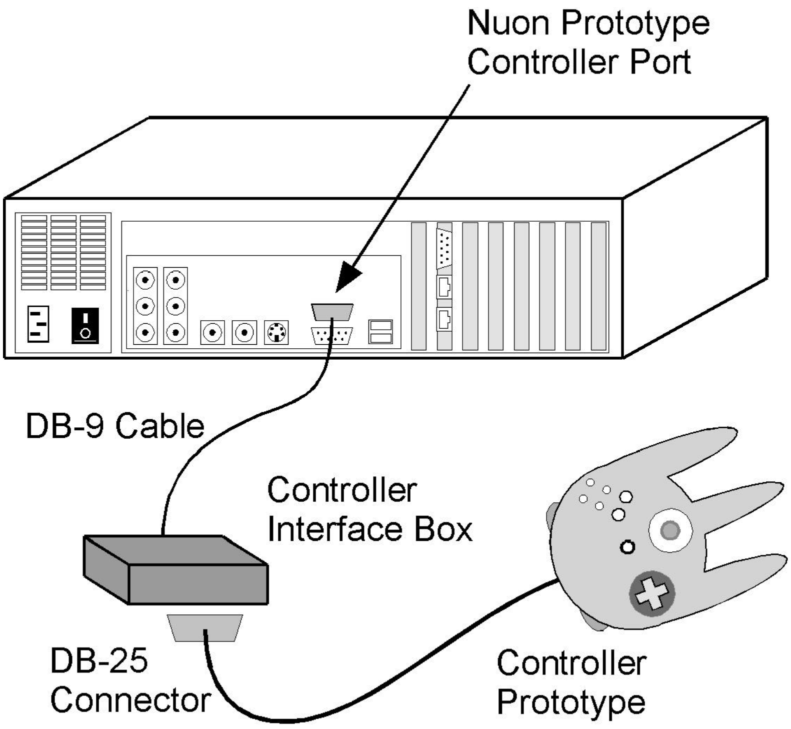

Figure 5-2 — Connecting A Controller Prototype

The controller interface is a small black plastic box with a DB-25 connector on one side and a DB-9 connector on the other. It goes between the controller and the development system.

Both revision 4 systems and revision 5 systems can use controller prototypes connected to an interface box. Simply connect the DB-25 connector coming from the controller into the DB-25 port on the interface box. Next, take the DB-9 cable that was supplied with your development system and connect it between the NUON Controller Port and the small interface box. That's all that is needed.

Please note that current revisions of the NUON BIOS no longer include support for the controller prototypes mentioned here. If you don't have any production controllers, please contact your VM Labs account executive immediately.

5.2.1 Interface Box Notes

Please note that a different interface box is required depending on the chip revision being used. The development system revision itself doesn’t matter, only the chip revision. An interface box designed for an Aries system will not function when connected to an OZ system, and vice versa.

This page intentionally left blank.

6. Connecting NUON & Your Host PC

The NUON development system is designed to connect to your host computer via standard 10Base-T Ethernet running on a standard twisted pair RJ45 cable. Category 5 or better cable is recommended.

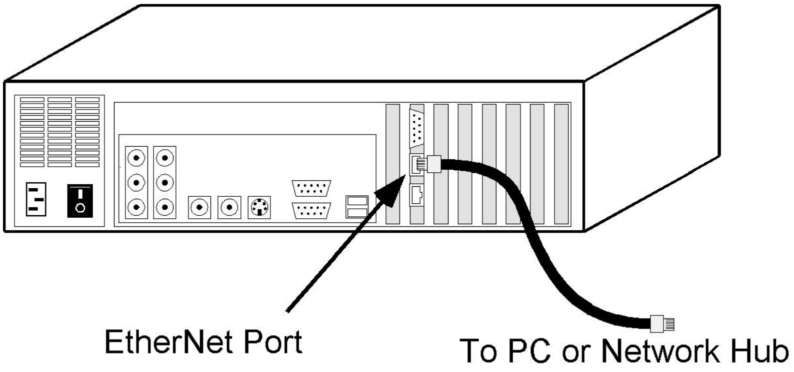

Figure 6-1 — NUON Ethernet Connection For Revision 5 Systems

6.1.1 Connecting as part of an existing network

If your host computer is already part of a network, simply connect a standard twisted pair RJ45 network cable between the NUON's Ethernet port and your network hub.

You may need to obtain a different network hub if your current one does not support 10Base-T on twisted pair RJ45 cable. If your network uses BNC coaxial cables, the best solution is to find a small hub that supports both twisted pair RJ45 and BNC. The computer can be connected using the existing cables, but the NUON must be connected using a twisted pair RJ45 cable.

If your network is running 100Base-T, the best solution is to use a dual-speed hub which allows both 100Base-T and 10Base-T connections.

Please note that although revision 5 development systems have separate ports for 10Base-T and 100Base-T, the 100Base-T port is not yet functional.

6.1.2 Connecting Directly

If your host computer is not connected to a network, then it may instead be connected directly to the NUON development system using a special *crossover* network cable. This cable should run directly from the NUON's Ethernet port to a network adapter card installed in your host computer.

If your host computer does not have a 10Base-T Ethernet adapter using twisted pair RJ45 cable, you will need to install one.

Note: Most crossover Ethernet cables are colored bright orange to distinguish them from standard cables. Making your own cable is not recommended.

6.1.3 NUON Serial Port

The NUON Development System also has a serial port. However, because the transfer rate of an RS-232 serial port is relatively very slow compared to Ethernet, it is used only for configuration of the NUON system and other special functions of the systems debug stub, and not for general communication.

6.2 Communicating With NUON

The TCP/IP network communications protocol is used to communicate with the NUON development system. Before you can run sample programs or do anything else with NUON, you must make sure that the TCP/IP configurations for NUON and your host computer are correct.

6.2.1 Do I have TCP/IP?

If your computer is already connected to a network, and you have the ability to access the Internet via that network, then you already have TCP/IP installed. You will probably not need to change anything about your host computer's network configuration.

If you access the Internet only via modem, or do not have Internet access at all, then you may need to enable and/or configure the TCP/IP protocol so that it may be used with your network adapter to communicate with the NUON development system.

Please contact your network administrator if you require assistance with TCP/IP setup.

6.2.2 Network Configuration

NUON requires no special network configuration for your host PC beyond enabling the TCP/IP protocol.

When configuring network hardware and software, if you do not have a strong working knowledge of the TCP/IP protocol and network configuration, it is recommended that you contact your in-house network administrator for assistance.

6.2.3 NUON's IP Address

The IP address of the NUON development system is set by default to 192.1.1.*xxx*, where *xxx* is indicated by a sticker on the back of the system.

Please note that the NUON development system currently requires that the host PC and NUON to be on the same *subnet* in order to communicate. The subnet is defined by the *subnet mask.*

The subnet mask is used to determine which IP addresses can be addressed on the local node of your network, and which are on other nodes or even on completely different systems.

With most small networks, the subnet mask value is 255.255.255.0. This means that an IP address that has the same first three values should be treated as "local". If the subnet mask is 255.255.0.0, then only the first two values must be the same.[3]

For example, if your subnet mask is 255.255.255.0 and your host computer's IP address is 203.34.123.54, then your NUON system must be set to 203.34.123.*xxx*, where *xxx* must be a value from 0-255 which is not used any other machine on the local network.

Accessing non-local IP addresses is typically done through what is known as a *gateway*. This is a network server or dedicated hardware device that is responsible for acting as a dispatcher, receiving packets of data from one subnet and passing them through to other subnets or the outside world. It may perform special filtering tasks to limit what sorts of packets may be passed through.[4] It may also perform special translation operations to change IP addresses as needed before sending them through to the outside world.[5]

Assigning IP addresses is usually the responsibility of your in-house network administrator. In order to avoid conflicts with other network devices, always ask them to assign an IP address for NUON to use.

You probably want to know how you can change the IP address. However, you must first know how to access the debug stub and setup menus so we'll discuss that information first in section 7. Then we'll come back to the topic of changing the IP address in section 8.4.

[3] Please note that this is an extremely simplified explanation. If you want more detailed information, you'll have to look at books on networking or TCP/IP.

[4] This is what's known as a "firewall".

[5] This is known as "Network Address Translation" and it allows an internal network to use a range of IP addresses without requiring that they be unique with respect to all of the other networks in the rest of the world.

This page intentionally left blank.

7. NUON Boot/BIOS ROM

This section will discuss the NUON boot/BIOS ROM. This is the flash ROM on your development system's motherboard that contains the NUON BIOS and related firmware required for the system to operate.

The information in this section applies to revisions released since October 2000.

7.1 Built-In System Configuration Program

Current revisions of the boot ROM include a configuration program that allows you to control a variety of important details about the machine's operation.

To access the configuration program, simply cycle the power or press the RESET button on your development system. After a few seconds, you should see a screen with a black background, a big red NUON logo, and white text.

If you press the **"Up"** direction on your controller's D-PAD while viewing this screen, the configuration program will be activated. If nothing happens, check your controller connection.

If the text on screen indicates that you should press a different button, then you may be running a new version of the software that was not available at the time of this writing. In that case, follow the prompts on screen.

Once the configuration program appears on screen, you can use the controller to change your system settings and perform certain tests. Simply follow the prompts on screen.

If you do not see the NUON logo screen described above and cannot access the configuration program, the most likely cause is that you have an older version of the system firmware installed.

7.2 Boot ROM Versions & Demo Programs

Please note that certain older demo programs for NUON may not function properly with newer versions of the boot ROM. In most cases, this is because the system video setup may be done differently than on earlier versions of the boot ROM.

This mainly affects older demos that are provided without source code, not sample programs which are provided along with source code as part of the SDK.

7.3 Updating Your Firmware

In the event that it is necessary for you to update your system's firmware, please follow the steps described below.[6]

Try the "EZ Update" steps first. They will work fine in most cases and they will save you time and effort since it's not necessary to open your machine.

If you experience problems with the "EZ Update" instructions, then move to the "Step-By-Step Trouble-Shooting Update" instructions, which include extra steps designed to avoid or recover from such problems.

7.3.1 EZ Update

1) Open an MSDOS command prompt window.
2) Change to the "**VMLABS\BIOS Update**" folder of your SDK installation.
3) Execute the "update.bat" batch file located in that folder.
4) Wait for the "Update Completed" message.
5) Power-cycle your NUON development system to complete the process.

These steps will download the program which reprograms the NUON development system's flash ROM, and then the update files themselves. There may be several files to be downloaded and the process may last a few minutes.

7.3.2 Step-By-Step Trouble-Shooting Update

If you see any error messages during the "EZ Update" instructions, try the expanded instructions below.

1) Open your NUON development system's case to get access to the motherboard.
2) Verify that your system has an EPROM chip in the socket on the corner near the DVD drive.[7] If you cannot locate the socket, or if no chip is installed, then stop now and contact VM Labs Developer Support.
3) Turn off your development system.

[6] This refers to SDK releases since February 15, 2001.

[7] This refers to Revision 5.x systems.

4) Look at your motherboard and locate the jumper that is labeled “FLASH” on one end and “ROM” on the other end. Change it to “ROM”

5) Turn your development system back on. After a few seconds, it should show color bars. If not, then stop now and contact VM Labs Developer Support.

6) Open an MSDOS command prompt window.

7) Change to the “**VMLABS\BIOS Update**” folder of your SDK installation.

8) Execute the “**update.bat**” batch file located in that folder.

9) When you see the prompt to “Press any key to continue . . .” change the jumper from “ROM” back to “FLASH” ***with the power still turned on.***

10) Press a key on your keyboard. The update process should begin.

11) Wait for the “Update Completed” message.

12) Power-cycle your NUON development system to complete the process.

If you see error messages again, please contact VM Labs Developer Support.

This page intentionally left blank.

8. The Debug Stub

The debug stub built into each NUON development system is separate from the system boot/BIOS ROM. It resides on the debug interface card and has the following basic functions:

- Allow configuration of Ethernet & TCP/IP settings.
- Manage the transfer of program code, data, and other information through the Ethernet TCP/IP connection and the serial port connection.
- Provide debugging functions through the serial port and through the Ethernet TCP/IP connection.

8.1 Debug Interface Card

Each NUON Development System has an interface card that contains the Ethernet network adapter and a serial port. This interface card also has a microprocessor that manages all communications using those ports.

- Revision 3 or 4 development systems use an ACE 360 card.
- Revision 5 development systems use a card based on the PowerPC 860.

The interface card also contains a Flash ROM chip that stores the code that runs on the local processor as well as NUON code used for data transfer and debugging. The code stored on this Flash ROM is known as the debug stub.

8.2 Updating The Debug Stub

The debug stub is updated from time to time to provide bug fixes or additional functionality. Because the code is maintained in an EEPROM on the interface card, it may be downloaded from the host PC without the need to open the machine and swap a chip.

In the SDK, the VMLABS\STUBS directory contains subdirectories for both the ACE360 stub and the PPC860 stub. Select the folder that is appropriate for your system, and make sure you view the README file within for the proper instructions.

8.3 Communicating With The Debug Stub

The steps below describe how you can use a simple terminal program to communicate with the NUON system's debug stub through the serial port.

These instructions assume you are using the HyperTerminal program that is included with Windows 95/98/ME, but they should be easily adapted to any other terminal emulation software.

1) Connect a DB-9 (female) null-modem serial communications cable from an available COM port on your host PC to the serial port on the back of your NUON box.

2) Run "HyperTerminal"

3) Create a new HyperTerminal session using the following parameters:

   Port = COM*x* (where *x* indicates the port you're using)
   Baud = 9600
   Data bits = 8
   Parity = none
   Stop bits = 1
   Flow control = none

4) If it is powered on, turn off your NUON system for a few seconds, then turn it back on, or hit the front-panel ***Reset*** switch. If you've successfully established a connection, then in HyperTerminal you'll see a startup message from NUON that looks something like this:

```
IP Address 192.1.1.222
Aries Debug Stub 3.95d - Jul 14 1999, 23:13:30
Lightweight debugging on 3
Listener initialized
Enter Ctrl-C to go interactive
```

The exact text you see will depend on the version of the debug stub that is installed on your NUON system. However, if you see no text at all, or if the text is garbled, then you do not have a valid connection. You should check your null modem cable connections and your HyperTerminal settings.

Once you've established a connection, hit <Control-C> on your host PC's keyboard to bring up the debug stub main menu. If you do not get this menu, check that your HyperTerminal settings match those specified in step 3 and then try again. If nothing works, contact VM Labs for assistance.

8.3.1 PPC860 Stub Menu

There really isn't a menu shown on the PPC860 stub serial connection. Instead, there is a simple command prompt. However, you can enter the command "help" to display a list of available commands, as follows:

```
> help

dump                        dump the packet log
help                        display help text
quit                        leave interactive mode
set ipaddr <addr>           set the ip address
set ipmask <mask>           set the subnet mask
set log info y              set info logging on
set log info n              set info logging off
set log requests y          set request logging on
set log requests n          set request logging off
set log rpackets y          set recv packet logging on
set log rpackets n          set recv packet logging off
set log xpackets y          set xmit packet logging on
set log xpackets n          set xmit packet logging off
set macaddr <addr>          set the mac address
set serve <n> y             enable service on mpe <n>
set serve <n> n             disable service on mpe <n>
show                        show parameter values
```

You will never need to use most of these commands. The commands that you do need to access are those for network configuration. These will be described in context in section 8.4.

8.4 Changing The IP Address Via The Stub

In most cases, it will be preferable to configure your system's IP address using the boot ROM's built-in configuration program as described in section 7.1. However, there may be situations where you want to change the IP address via the stub. For example, if you needed to change the IP address without affecting a program that's already executing on the NUON chip.

8.4.1 Using The Serial Port Menu

Following the steps below performs the IP address configuration on all development system revisions.

Establish a connection to the debug stub via the serial port connection. This process is described in section 8.1. Once you've established a connection, hit <Control-C> on your keyboard to enter interactive mode:

```
Type 'quit' to exit interactive mode

> _
```

The exact text of the prompt depends on the stub version and machine revision. If you do not get any prompt, check that your HyperTerminal settings match those specified in step 3 and then try again. If nothing works, contact VM Labs for assistance.

From this point, please follow the appropriate steps from the table below, depending on which type of machine you have.

Revision 5 Systems	Revision 4 Systems
You can get a list of available functions by entering “help” and pressing <Enter>.	Enter “E” followed by <Enter> to access the EEProm functions menu.
To see the current settings, type “show” and press <Enter>. This will result in a display similar to that shown below.	On revision 4 systems, enter “S” followed by <Enter> to display the current settings.

At this point, the stub will display several lines of information as shown below (it may not look exactly like this). The important settings are the *IP Address*, the *IP Mask*, and the *MAC Address*. It’s a good idea to write down the existing settings before making changes.

```
IP address 192.1.1.222
IP mask 255.255.255.0
MAC address 00:a0:c9:69:c9:de
Don't log info
Don't log requests
Don't log recv packets
Don't log xmit packets
Lightweight Debugging
  MPE 0 no
  MPE 1 no
  MPE 2 no
  MPE 3 yes
```

Revision 5 Systems	Revision 4 Systems
Change the IP address by typing “set ipaddr” followed by the new IP address in dotted notation. For example: `set ipaddr 209.46.69.224` would change the IP address to 209.46.69.224	Change the IP address by typing “I” followed by the new IP address in dotted notation. For example: `I 209.46.69.224` would change the IP address to 209.46.69.224

Revision 5 Systems	Revision 4 Systems
To change the IP subnet mask address, type “set ipmask” followed by the new IP mask address. For example: `set ipmask 255.255.255.0` would change the IP subnet mask address to 255.255.255.0.	To change the IP subnet mask address, type “M” followed by the new IP mask address. For example: `M 255.255.255.0` would change the IP subnet mask address to 255.255.255.0.
After making your changes, type “show” again to verify the new settings.	After making your changes, type “S” and hit <Enter> to verify the new settings.

Remember to write the new settings down. You should always use a label on the back of the machine to indicate the current settings.

Cycle the power to NUON so that the updated settings will be recognized. Alternately, you can hit the RESET button at the end of the PPC860 interface card.

That’s it! You’ve changed your NUON system’s IP address.

8.5 Changing NUON’s MAC Address

The MAC address is another type of network address. The “MAC” part of the name stands for “Media Access Control” and it is the low-level address used by Ethernet to uniquely identify each piece of hardware on a local area network. The MAC address value consists of a series of six hexadecimal numbers. For example:

```
00 A0 C9 69 C9 DE
```

In rare cases, it may be necessary to change the MAC address of your NUON system to avoid conflicts with other systems on the same network.

In most cases, the default factory setting of the MAC address will work fine. However, if you experience odd network conflicts between different NUON systems on the same network, or between a NUON system and another network device, then changing the MAC address may fix the situation.

Please follow the appropriate steps from the table below, depending on which type of machine you have.

Revision 5 Systems	Revision 4 Systems
Change the MAC address by typing “set macaddr” followed by the new MAC address in hexadecimal notation. For example: `set macaddr 00 a0 c9 69 c9 de`	Change the MAC address by typing “W” followed by the new MAC address in hexadecimal notation. For example: `W 00 a0 c9 69 c9 de`
After making your changes, type “show” again to verify the new settings.	After making your changes, type “S” and hit <Enter> to verify the new settings.

Remember to write the new settings down. You should always use a label on the back of the machine to indicate the current settings.

When you've finished, cycle the power to your NUON system so that the updated settings will be properly recognized.

8.5.1 Selecting A New MAC Address

If you decide to change the MAC address, the question is what to change it to? Unfortunately, there is no easy way to determine what MAC addresses are already used by other devices on your network. However, the fact that there are six separate values means that random selections are likely to work without conflict. Therefore, it’s probably best to simply change one of the six existing values and then see if the conflict goes away.

Please note that the usual convention for NUON is that the last digit of your MAC address should be the same as the last number of your TCP/IP address.

8.6 Your Host PC’s IP Address

Depending on your network configuration, the IP address used by your host PC may be obtained in different ways.

8.6.1 Using A Static IP Address

The first method is to assign a static IP address. This address is the same each time the computer connects to the network. This requires that the network administrator keep track of which IP addresses are in use at all times, in order to ensure that no single IP address is assigned to more than one machine.

8.6.2 Using DHCP

The other method is known as DHCP (Dynamic Host Configuration Protocol), which allows a machine to request an IP address and other network configuration settings from a server when it starts up. This method requires that a particular machine on the network act as a DHCP server.

The DHCP server listens to the network for login requests, and responds by assigning an IP address. It also broadcasts the address of the network gateway, DNS server, and other network resource information.

NUON does not know how to use DHCP (as of this writing), but there is no restriction on using it for your host PC.

8.6.2.1 When No DHCP Server Is Available

If a DHCP server is not available to assign an IP address dynamically to your host PC when it connects to the network, then you must specify a fixed IP address for your host PC. Otherwise, the TCP/IP services on your PC will not know what address to use and they will not initialize properly. Then you will not be able to communicate with your NUON development system.

8.6.3 Using Multiple Network Cards

Please note that if you have multiple network interface cards installed in your host PC, the TCP/IP protocol is handled separately for each one. You can have a fixed IP address for one card and a dynamically assigned IP address for the other. They can even be on completely different subnets.

This page intentionally left blank.

9. NUON Development Tools

This section will give a basic overview of the NUON development tools: how to get them working, what files are involved, etc.

9.1 System Requirements

The system requirements for the development tools are as follows:

- 133Mhz or faster Pentium-based PC
- 10Base-T (RJ45) Ethernet adapter
- Windows 95/98/ME, Windows NT 4.0, or Windows 2000
- 16mb RAM (minimum, 32mb or more recommended)
- 70mb free disk space (the basic amount required for the SDK tools, libraries, sample code, and demos, not counting your own project's code and data)

These requirements are subject to change as new revisions of the SDK are made available.

Please note that the SDK tools may work with lesser configurations, but they are not recommended.

9.1.1 Which Version Of Windows?

The NUON SDK tools should work on any 32-bit version of Windows since Windows 95. However, please be aware that there may be subtle differences from one version to another. Also note that individual software projects for NUON may be configured to use tools or methods that are specific to a particular version of Windows.

9.1.1.1 Windows 95

If you're still using Windows 95, we strongly recommend that you upgrade to Windows ME or Windows 2000. However, at this time there are no known problems using any of the NUON SDK tools with Windows 95.

9.1.1.2 Windows 98

At this time, there are no known problems using any of the NUON SDK tools with Windows 98.

9.1.1.3 Windows ME

At this time, there are no known problems using any of the NUON SDK tools with Windows Millennium Edition.

9.1.1.4 Windows NT

At this time, there are no known problems using any of the NUON SDK tools with Windows NT v3.5 or 4.0.

9.1.1.5 Windows 2000

At this time, the preferred choice of operating system for the NUON SDK is Windows 2000. There are no known problems using any of the NUON SDK tools with Windows 2000.

9.2 Tools Installation

Information on obtaining the NUON SDK is provided in section 2.3. We'll presume you've followed the instructions there and have decrypted the downloaded file to obtain an archive file named SDK.ZIP.

You should extract the contents of the SDK.ZIP archive to the root directory of your selected drive. When expanding the SDK archive, make sure you have selected the option to preserve the archive's directory hierarchy. This will result in a folder named **VMLABS** that will contain everything else.

Note that using the standard DOS-based PKZIP or PKUNZIP tools from a DOS command shell will not preserve the long filenames used by many of the files within the archive. A Windows 95/NT compatible ZIP tool that properly understands long filenames is required to properly extract the contents of the archive.

If you do not have such a tool already, please visit the DOWNLOAD.COM web site at **http://www.download.com** and search for "WINZIP".

After you have extracted the contents of the SDK archive to your hard disk, most of the NUON development tools require no additional setup except for setting a few environment variables, as detailed in section 9.3.

9.3 NUON SDK Environment Variables

The environment variables listed in this section must be set as indicated in order for the NUON SDK tools to function properly.

9.3.1 MD_PORT

This must be set to the IP address assigned to the NUON development system. For example:

```
set MD_PORT=192.1.1.222
```

If you have multiple NUON development systems and wish to communicate with more than one, you may change the contents of the MD_PORT variable to switch back and forth using the “set” command of the DOS command prompt.

Note that the PUFFIN debugger relies on the presence of the MD_PORT variable to determine if it should communicate with a development system or use the built-in NUON processor emulation mode. If the MD_PORT variable is not set, then PUFFIN will use emulation. ***(Note that not all versions of PUFFIN support the emulation mode.)***

9.3.2 VMLABS

The **VMLABS** variable should point at the folder containing the SDK. For example:

```
set VMLABS=C:\VMLABS
```

This variable should specify a single path as shown above.

Throughout this document, we may occasionally refer to this variable as **$VMLABS**.

If you experience difficulty compiling or linking when your tools and source code are on different drives, make sure this variable is set to point at the SDK folder.

9.3.3 PATH

This variable is used to contain a list of directories to search when looking for executable programs.

The **PATH** list must include the VMLABS\BIN directory on the drive where you have installed the SDK. You can easily add this to your existing path using the following command:

```
PATH=%PATH%;%VMLABS%\bin
```

This presumes that the **VMLABS** variable is already defined before you set the **PATH** variable.

9.3.3.1 Conflicts With Other Tools

If you have development tools for other platforms installed on your system, check to see if anything has a filename that matches that of any of the NUON tools. If so, this is likely to cause problems.

To fix this, you may need to change the order of the directories specified in the **PATH** variable, or you may need to remove the directory containing the other variables from the **PATH**.

You may wish to create batch files that reset the **PATH** variable to include or exclude the VMLABS\BIN directory as needed in order to switch back and forth to other tools.

9.3.4 DJGPP

The DJGPP variable and the DJGPP.ENV configuration file are commonly used by MSDOS/Windows versions of the GCC & GNU tools, including versions of the NUON SDK tools released prior to October 1998. However, they are no longer required by the current NUON SDK tools/

9.4 MS Windows & Environment Variables

9.4.1 Setting Environment Variables Under Windows 2000

Under Windows 2000, environment variables are configured via the ***System Properties*** control panel. Either right-click on the *My Computer* icon and select “Properties”, or select *System* from the Control Panel.

Once the ***System Properties*** control panel is shown, select the *Advanced* tab at the top. Then select the *Environment Variables* button. This will bring up a new dialog box where you can edit your environment variable settings.

Alternately, you can also configure environment variables directly on the command shell’s command line, or in a batch file. However, such configuration is specific to that particular instance of the command shell.

9.4.2 Setting Environment Variables Under Windows 95/98/ME

With Windows 95/98/ME, environment variables are normally configured by your AUTOEXEC.BAT file. For example, a line such as:

```
      set MD_PORT=192.1.1.222
```

will set the variable named "**MD_PORT**" to the value shown.

9.4.3 Things To Remember

Please note that under Windows 95/98/ME, the rules below apply to how environment variables are used. These may affect your configuration.

- Global environment variables used by Windows are defined in your AUTOEXEC.BAT file. If you do not have an AUTOEXEC.BAT file, then Windows will create a minimal set of global environment variables.
- Applications executed from the Windows desktop inherit the global environment variables. This includes the DOS command shell.
- Programs executed from another application may inherit environment variables from that application or from Windows, depending on how the application launches it.
- Within the DOS command shell, environment variables can be changed by using the "set" command either on the command line or in a batch file. However, such changes only affect that particular instance of the command shell and those programs that are executed from it.
- The command shell's "start" command to run a program is really a message asking Windows to start the program. Therefore, only the global environment variables are inherited, not changes made within the shell.

9.5 Windows Cross-Platform Issues

If you are working in an environment where multiple versions of MS Windows are used, please be aware that there are some subtle differences from one version to another. These differences may impact your build environment and you need to be are of them.

The main thing to watch for is that there are some differences in the way the command shell works under 95/98/ME compared to the Windows NT/2000 command shell.

Specifically, the options for some commands may be different, or certain commands on one system may have no direct equivalent on the other system.

We’re not aware of any place in the NUON SDK where this is an issue, but it’s good to know if a problem does pop up.

9.6 Making the tools work

The main things required to make the development tools work are:

1) Make sure your NUON’s IP address is configured properly. See section 6.2.3 for further information.

2) Download the most recent version of the SDK and decrypt it using the DES tool as described in section 2.3. This will give you a ZIP archive file containing the SDK files.

3) Extract the contents of the SDK ZIP file as described in section 2.3.3.

4) Set the environment variables used by the tools as described in section 9.3.

10. NUON SDK Overview, By Category

This section will break down the programs included in the SDK into separate categories according to function. This is primarily a reference so that you know what a particular file is for.

Please note that some files may be listed in multiple places. Some tools may not be listed because they have been removed from the SDK. This is usually because the tool in question has been replaced by improved functionality in another tool, or because it was never really intended to be part of the SDK in the first place.

This section is intended to give you a general idea of what's in the SDK. Because the SDK is being updated on a regular basis, the information in this chapter is very likely to be at least a little different from what is contained in the current SDK release.

10.1.1 C/C++ Compiler

There are several files that are part of, or related to, the C/C++ compiler:

Program Name	Description
MGCC.EXE	C/C++ compiler driver. NUON-specific version.
AS.EXE *or* LLAMA.EXE	NUON LLAMA assembler. Assembles the intermediate assembly language output created by the C/C++ compiler. The normal convention of the GCC compiler is to call the assembler "AS". However, the NUON version of GCC has been altered to first look for "LLAMA".
CC1.EXE	C Compiler executable
CC1PLUS.EXE	C++ compiler executable
CPP.EXE	C/C++ Preprocessor
LD.EXE *or* VMLD.EXE	Linker used to combine compiled object modules and libraries into an executable program file. The normal convention of the GCC compiler is to call the linker "LD". However, the NUON version of GCC has been altered to first look for "VMLD".
MG++.EXE or G++.EXE	C++ compiler driver. Basically similar to MGCC, but is more C++ specific. Using MGCC.EXE is recommended for NUON development.

10.1.2 Assembler

The assembler is known as Llama.

Program Name	Description
LLAMA.EXE	NUON assembler

10.1.3 Linker

The linker is VMLD.

Program Name	Description
VMLD.EXE	NUON Linker

10.1.4 Debugger

The debugger is known as “Puffin” and there are two incarnations:

Program Name	Description
PUFFIN2K.EXE	Windows version of debugger based on the Tcl/TK GUI scripting language This is the primary incarnation of the debugger that you will be using. It supports C and assembly source level debugging as well as the XLisp scripting capabilities
PUFFIN.EXE	Console version of debugger. Commands are issued using the XLisp script language.

10.1.5 Object Module Utilities

The following tools are designed to perform various operations on compiled object module files or executable program files.

Program Name	Description
COFFDUMP.EXE	Dumps a list of all symbols within a COFF object module or executable program file.
VMAR.EXE	Library archive utility
VMNM.EXE	Library & object module symbol name utility
VMOCOPY.EXE	Object module manipulation & conversion utility
VMSTRIP.EXE	Symbol strip utility.

10.1.6 Miscellaneous Tools

The following tools are used for a variety of purposes.

Program Name	Description
GMAKE.EXE	Program builder utility

Program Name	Description
MLOAD.EXE	Aside from the debugger, this program is the programmer's main interface to NUON. It allows you to download code and data, reset the machine, update the flash ROM, and more.
MTRACE.EXE	Walks the program stack to determine the current call stack (the list of functions that were called to arrive at the current program counter address).
REDIR.EXE	This tool allows the user to individually redirect the standard character devices ***stdout*** and ***stderr*** so that error message, status information, and other output from the tools may be captured as needed.

10.1.7 C & C++ Include Files

The **VMLABS\INCLUDE** directory contains C/C++ include files for standard ANSI C and C++. These files are not NUON-specific.

The **JPEG** subdirectory contains the C & C++ include files needed by the standard JPEG library.

The **MACHINE** subdirectory contains the C & C++ include files which define basic data types and machine characteristics.

The **M3DL** subdirectory contains the C & C++ include files needed by the M3DL graphics library.

The **NUON** subdirectory contains C/C++ include files which are specific to NUON.

The **OBJC** subdirectory may be present in some SDK releases. It contains include files which are used for Objective C, an object-oriented variation on the basic C language. The GCC compiler understands Objective C, but this mode of operation is not supported by VM Labs.

The **SYS** subdirectory contains a number of C & C++ include files needed by the C & C++ runtime library.

Other subdirectories may also be present, and will usually contain include files required by specific libraries.

10.1.8 Library Files

The **VMLABS\LIB** directory contains linkable library archives containing object modules from the various programming libraries. It also includes the program startup code for C and C++ programs.

The **SRC** subdirectory contains a variety of other subdirectories that contain much of the source code for the NUON-specific programming libraries.

10.1.9 Sample Code

The **VMLABS\SAMPLE** directory contains a number of subdirectories with source code to a wide variety of sample programs that demonstrate the use of the NUON system and programming libraries.

10.1.10 Tutorials

The **VMLABS\DOCS\TUTORIAL** directory contains a number of subdirectories that contain NUON programming tutorials. Please note that the program code contained in these directories is intended for teaching purposes and may not actually compile, assemble, or link properly without additional editing or other files.

10.1.11 Interface Card & Debug Stubs

The **VMLABS\STUBS** directory contains the files needed to update the flash ROM of a development system's Ethernet interface card.

Please see section 8.1 for more information.

10.1.12 BIOS ROM Images

The **VMLABS\BIOS Update** folder contains the files required to update your development system's FLASH ROM, which contains the system BIOS and bootstrap code.

Please note that the NUON SDK samples and all current development presumes that a ROM-based BIOS is available. Older ROM images (before January 2000) may not include the BIOS.

Aside from the ROM image binary files, this folder also contains the programs required to perform the update.

10.1.13 Documentation

The **VMLABS\DOC** directory contains a variety of documentation about NUON and the various programming libraries. There may be additional subdirectories dividing the documentation into different categories.

Documentation is provided in one of the following formats:

- Adobe Acrobat Portable Document Format (PDF)
- Hyper Text Markup Language (HTML)
- ASCII Text

10.1.14 3D Studio MAX Plug-Ins

The “**VMLABS\3D Studio MAX Plug-Ins”** directory contains plug-in modules for the 3D Studio MAX program. These plug-ins allow you to save 3D graphics information which can be used with the NUON 3D graphics libraries.

The *3D Studio MAX* program itself is not included with the NUON SDK. It is a popular 3D modeling and animation program commonly used for 3D graphics development.

10.1.15 Photoshop Plug-Ins

The “**VMLABS\Photoshop Plug-Ins”** directory contains plug-in modules for the Adobe Photoshop program. These plug-ins allow you to save bitmapped graphics information using NUON-specific file formats.

The *Adobe Photoshop* program itself is not included with the NUON SDK. It is a popular graphics & photo editing program commonly used for bitmapped graphics editing.

This page intentionally left blank.

11. NUON SDK Tools Mini Command Reference

This section will provide a brief description of the main command line options for the primary SDK tools, and provide other basic information that may be useful.

This is intended to serve primarily as an introduction. Some tools will feature more command line options and features not mentioned here. For more detailed information on any given tool, please also refer to whatever additional documentation is provided separately.

This documentation refers to SDK tools released on or after September 10, 1999. Previous versions of the tools may function differently in some respects.

11.1 C/C++ Compiler

The program normally used to execute the C/C++ compiler is MGCC. It uses a command line formatted as follows:

```
mgcc [options] [source files]
```

The table below shows some of the more useful command line options for MGCC. Please note that the commands are case-sensitive.

This information applies to compiler releases of February 15, 2001 or later.

Option	Description
-ansi	Force strict ANSI-C syntax. Disables GCC features which are not ANSI-compliant, including the `asm` and `inline` keywords, certain predefined macros, and recognition of C++ style comments in C code.
-c	Perform a compile operation only, do not call the linker
-C	Tells the preprocessor not to discard comments. Used in conjunction with the "-E" option.

Option	Description
-D<*macro*>[=*value*]	Defines a preprocessor *macro* on the command line. This is equivalent to having “#define macro” statements embedded at the top of the C source code. For example, “-DSCRNWIDTH=360” is equivalent to having “#define SCRNWIDTH 360” at the top of your source code file. The value field is optional. If no value is specified, the macro is assigned a value of “1”. For example, having “-DNOJOYSTICK would be equivalent to having either “#define NOJOYSTICK” or “#define NOJOYSTICK 1”.
-E	Stop compiling after the preprocessor stage is finished. If no output filename is specified using the “-o” option, the output from the preprocessor is sent to standard output.
-fomit-frame-pointer	This will reduce code size by omitting the stack frame pointer when possible. However, this does make the resulting program more difficult to debug.
-g	Add source-level debugging information to the output file.
-I *directory*	Add the specified *directory* to the paths searched for included files during the preprocessor stage.
-include *file*	Process *file* as input immediately before processing the source file.
-L *directory*	Add the specified *directory* to the paths searched for library archive files during the link stage.
-l<libraryname>	Specify that a particular library archive should be included in the link process. For example, “-lmath” would be used to include “math.a”. The library specified is expected to be in the current directory or in the $VMLABS\LIB directory. Libraries and object modules (including those resulting from source files) are always searched in the order specified.
-malignfuncs	Specify that all functions should be aligned to begin on a 64-byte boundary, which is the default cache line size. This will often result in better cache performance.

Option	Description
-mbl	This option specifies that the program should use a library-based BIOS (LIBBIOS.A) rather than the system's built-in ROM BIOS. This also causes a different startup code module to be used during the link stage. This is essentially the opposite of the **–mrom** option, and was formerly the default.
-mpe0	Specify that the program will use MPE 0 as the primary processor. This affects the compiler's target memory map. (This was the default with older compiler versions.)
-mpe3	Specify that the program will use MPE 3 as the primary processor. This affects the compiler's target memory map. (This is the default mode of operation since February 2001.)
-mreopt	Invoke the assembler with -b -O1
-mreopt-more	Invoke the assembler with -b -O2
-mrom	Specify that the program uses the ROM-based BIOS. This tells the compiler to use the appropriate startup code module during the link stage. (This is the default mode of operation.)
-no-builtin	Don't recognize built-in functions whose names do not begin with two leading underscores. This includes *abort*(), *abs*(), *alloca*(), *cos*(), *exit*(), *fabs*(), *ffs*(), *labs*(), *memcmp*(), *memcpy*(), *sin*(), *sqrt*(), *strcmp*(), *strcpy*(), and *strlen*(). GCC normally generates special code to handle certain built-in functions more efficiently. However, this can affect debugging, and you cannot change the behavior of those functions by writing customized versions.
-o <file>	Specify the filename used for output
-O0 -O -O1 -O2 -O3	Specify that optimization should be used, in varying degrees. "–O0" is no optimization. "-O" is basic optimization. "-O3" is extreme optimization. Note that this does not specify optimization for the assembler stage. To specify assembler optimization, see the **mreopt** and **mreopt-more** commands.
-pedantic	Issue all warnings required by strict ANSI standard C. Reject forbidden extensions.
-S	Compile only, do not call the assembler, do not delete the assembly language source file that gets generated

Option	Description
-s	Remove all symbol table and relocation information from output file.
-traditional	Support some aspects of older non ANSI-C compilers. May not work with header files written to the ANSI-C specification.
-U *macro*	Undefine the specified *macro* symbol. Equivalent to "#undef *macro".* This option is always processed after the "-D" option and before the "-include" option.
-u *symbol*	Pretend that *symbol* is undefined, forcing linking of library modules in order to define it.
-v	Verbose mode. Output information about command lines to programs called by MGCC, such as the C preprocessor, assembler, or linker.
-Wall	Turn all warnings on.
-Xlinker *option*	Pass *option* through to linker stage. Note: if linker option requires arguments, the "-Xlinker" command must be given for each. For example, to pass through "assert definitions" you must write "-Xlinker assert –Xlinker definitions"

For example:

```
mgcc -O2 -mreopt-more -o hello.cof hello.c
```

This would specify extreme optimization for the C compiler, as well as optimization for the assembler. It specifies that the output filename should be HELLO.COF. Finally, the input file HELLO.C is specified. This will compile the HELLO.C file, link it with the standard libraries, and produce the HELLO.COF output file.

11.1.1 Paths

The compiler expects to find executables within a directory specified by the **PATH** environment variable. Other files are located relative to the **VMLABS** environment variable.

The compiler expects include files to be located in the $VMLABS\INCLUDE directory, or in the current directory, or the directory specified using the "**-I**" option.

The compiler expects runtime startup code and linkable libraries to be located within the $VMLABS\LIB directory, or in the current directory.

11.1.2 NUON C/C++ Compiler Notes

This section was last updated on February 5, 2001.

- C++ code compiled with compiler releases dated before February 2001 should be recompiled.
- Since the Feb. 2001 release, the compiler is now more ANSI compliant and also stricter about enforcement. Some things which generated warnings with the previous versions will now generate errors.
- As of the Feb. 2001 compiler release, the compiler now combines shifts, additions, and logical operations (AND, OR, XOR, etc.) into one instruction where possible.
- The compiler no longer always puts function addresses into a register. It now sometimes generates a direct reference (i.e. "**jsr foo**") rather than an indirect reference (i.e. " **jsr (r1)** ").
- In many cases, the best code generation (as regards execution speed) may be obtained by using compiler options of:

  ```
  -O3 –mreopt-more –fomit-frame-pointer
  ```

  rather than simply specifying "**-**Os". You may wish to experiment.
- The **–mrom** and **–mpe3** options are now the default mode of operation and no longer need be used on the command line. They are still recognized, but this may change in future releases of the compiler.
- The **–mbl** and **–mpe0** options have been added. These cause the compiler to behave as it did before the **–mrom** and **–mpe3** options were made the default.

11.1.2.1 Command Line Options

Please note the following information regarding compiler options that may have changed from one version of the compiler to another.

- As of the February 2001 compiler release, the options **–mrom** and **–mpe3** are now the default mode of operation. Please note that these options will most likely disappear from a future compiler version.
- Since **–mpe3** is now the default, a new **–mpe0** option has been added. This tells the compiler that the code is intended to run on MPE 0 rather than MPE 3.

11.1.2.2 Function Calling Conventions

Please Note: *The compiler conventions are subject to change as the compiler is optimized for the NUON system. When in doubt about the behavior, you should examine the compiler output, or contact VM Labs Developer Support.*

Generally, the first ten words of parameters (counting from the left) are passed in registers r0 through r9.

Any excess parameters are passed on the stack, with scalar alignment.

Parameters with short or char type will be promoted to *int* (or *unsigned int*) before the call.

A parameter will be placed on the stack:

- If it is an unnamed parameter to a <stdarg.h> function, or
- If it is the last named parameter to a <stdarg.h> function, or
- If the type has variable size, or
- If the type is marked as addressable (it is required to be constructed into the stack), or
- If the padding and mode of the type is such that a copy into a register would put it into the wrong part of the register, or
- If it is too large to fit in the remaining parameter registers. In this case, subsequent parameters will still be candidates to be passed in registers.

A value in a register is implicitly padded at the most significant end. On a big-endian machine, that is the lower end in memory. So a value padded in memory at the upper end can't go in a register.

Once a parameter has been forced onto the stack, all the remaining parameters will go there too. (Except as noted)

Return values are in r0. Five to eight byte return values are in *r0* and *r1*. Bigger return values are in memory, with the address in *r0*.

The called function is responsible for preserving *r12 – r28, r30, r31, sp,* and *acshift*.

The *rc0* & *rc1* registers are not preserved by a function.

Functions are not responsible for preserving *r0* - *r11*.

11.1.2.3 Obsolete NUON-Specific Options

- The **-minterrupt** option is no longer supported.
- The **-mno-prologue** option is no longer supported.
- The **-mpack** option is no longer supported
- The **–moz** and **-maries** options are no longer supported. Aries-only code generation is now the only mode of operation.
- The **–mrom** and **–mpe3** options are no longer required to specify code that will execute on MPE 3 on a ROM-based system, as this is now the default. These options will be removed from a future compiler release. To restore the old behavior, use the **–mbl** and **mpe0** options.
- The **-mfastcalls** option may be recognized, but currently has no effect.

11.1.2.4 Other Conventions

- The compiler generates code that uses *r31* as its stack pointer. It expects *acshift* to be zero. These get set by the code in the C runtime startup file.
- The *r31* register must be vector (16 byte) aligned at all times.

By default, type *char* is signed.

11.1.3 Additional Documentation

The C/C++ compiler in the NUON SDK is based on the GNU GCC compiler from the Free Software Foundation.

Additional documentation on the GCC compiler from the Free Software Foundation is available in a separate document. This document discusses only the aspects of the compiler that are not NUON-specific.

11.2 Llama Assembler

The Llama assembler was created by VM Labs to meet the specific requirements of the NUON processor. It uses a command line formatted as follows:

```
llama [options] [source file]
```

The table below shows some of the more useful command line options for the Llama assembler. Please note that the commands are case-sensitive.

Option	Description
-?	Help. Display command line options.
-b	Assume condition codes need not be preserved across branches
-B *linkbase*	Set base address for linking (Valid for MPO output only)
-c#	Add padding for executing from cache; # is the length of cache lines in bytes. Using -c alone is the same as -c32.
-Dsymbol[=val]	Define a symbol and optionally assign it a value. Equivalent to: Symbol = value at the top of your assembly source code file.
-e *errfile*	Output XLisp compatible error records to *errfile*
-fasm [,bin] [,expand-syms] [,expand-includes] [,expand-all]:	Create assembly language output; options available: *bin*: annotate output with hex representation of instructions *expand-syms*: expand symbols to their ultimate definitions *expand-includes*: expand contents of .include directives *expand-all*: expand symbols and interpret module definitions
-fbinary:	Output raw binary data
-fcoff:	Output COFF object file
-f*FMT*	Select format of output file
-flist:	Same as **-fasm,bin,expand-includes**
-fm68k:	Same as **-fveri,width=32,segheader,prefix=' .dc.l 0x'**
-fmpo:	Output .mpo file for debugger / NUON emulator
-fsrec:	Output Motorola S-Records

Option	Description
-fveri [,width=nn] [,segheader] [,absaddr] [,prefix =*'string'*] [,rom]:	Create Verilog load file; options available are: width=nn: set width of output file in bits (default 128) segheader: output 2 long word header for each segment: the segment origin and segment length in bytes absaddr: output absolute addresses in the Verilog file; otherwise the top bits of the address will be masked off and it will be divided by the memory width in bytes prefix='string': causes the given *string* to be printed at the beginning of each line, instead of a tab. Must be the last option given. rom: same as `width=8,segheader,absaddr'
-g	Include GDB debugging information
-g-old	Include old-style (obsolete) debugging information
-i *incfile*	Process contents of file, but do not include it in assembly output
-I *incpath*	Add incpath to the search path for include files
-jextern	Default is –jlocal, unless -c was given
-jlocal	Assume jumps/jsrs are in local RAM
-M	Generate MAKEFILE dependency list (only, causes assembly to be disabled.)
-nolines	Remove all line number info methods
-nolisp	Assume jumps/jsrs are in external (non-local) RAM remove all before and after methods
-o outfile	Set output file name
-O#	Optimization level: 0 = no optimization 1 = fast (but not very good) optimization n>1 = good optimization with n-1 levels of lookahead (potentially unbelievably slow) Default is -O0
-Osize	Optimize for space rather than time
-r#	Assembly language revision number. Default is: 20
-v	Verbose flag: print interesting statistics about the program and the file being assembled.

There are a few additional command line options in Llama, but they are used for debugging the assembler itself and are not required for application development.

11.2.1 Paths

Llama expects include files to be located in the $VMLABS\INCLUDE directory, or in the current directory, or the directory specified using the “-I” option.

11.2.2 Additional Documentation

Additional documentation regarding the Llama assembler is available in two documents:

- ***Llama User’s Manual*** — This document is available in Adobe Acrobat format.

- ***Optimizing Your Llama*** — This is an HTML-based tutorial that demonstrates how to optimize NUON assembly language.

11.3 Linker

The linker from the NUON SDK was created by VM Labs to meet the specific requirements of the NUON processor.   It uses a command line formatted as follows:

```
vmld [options] [source files] [library files]
```

The table below shows some of the more useful command line options for the linker. Please note that the commands are case-sensitive.

Option	Description
-?	Help. Display command line options
-b *file*	Link in the specified *file* as raw binary information.
-B *value*	Use alternate base memory location. Default base is 0x80000000.
-e *symbol*	Define an entry point. Required in order to create an executable program file.
-i *or* -r	Incremental link. Makes the linker produce a relocatable output file that can be used as input to another link.
-L*directory*	Add the specified *directory* to the path list searched for library files. Normally, the path list is the current directory and $VMLABS\LIB.

Option	Description
-l*name*	Include the library archive specified by *name* in the link. The prefix of “lib” and a filename extension of “.a” is automatically added to *name*. For example, “-lmath” specifies that the library archive LIBMATH.A should be included in the link.
-n	Generate an output file even if non-fatal errors occur.
-o *name*	Specify that *name* should be used for the output file. The default output filename is LD.OUT.
-T *name[, name2…]* =*value[+offset][:limit]*	Specify the load address of *value* for the section with the specified *name*. Multiple segment names may be provided, separated by commas. An additional *offset* may be provided. This value will be added to *value*. The *limit* value can optionally specify a maximum size for the section(s) involved. If the combined size is larger than the specified value, the linker will show a warning. By default, the runtime address of the specified section(s) will be changed as well as the load address. See the **–R** option for more specific control over the runtime address.
-V	Print version of linker
-R *name[, name2…]* =*value[+offset][:limit]*	Assign a specific runtime address of *value* for the section with the specified *name*. Multiple segment names may be provided, separated by commas. An additional *offset* may be provided. This value will be added to *value*. The *limit* value can optionally specify a maximum size for the section(s) involved. If the combined size is larger than the specified value, the linker will show a warning.

11.3.1 Paths

The linker expects object modules and library files to be located in the current directory, or in the $VMLABS\INCLUDE directory, or the directory specified using the “-L” option.

11.4 GMAKE Utility

The GMAKE utility uses a command line formatted as follows:

```
    gmake [options] [target(s)]
```

The table below shows some of the more useful command line options for the GMAKE utility. Please note that the commands are case-sensitive.

Option	Description
-C *directory* **or** --directory = *directory*	Change to the specified *directory* before doing anything
-d **or** -debug	Debug mode. Print out MAKEFILE debugging information. Prints commands being executed, and lots of other stuff.
-e **or** --environment-overrides	System environment variables override variables with same name which are defined by GMAKE and within MAKEFILE. (Default is vice-versa.)
-f *file* **or** --file=*file* **or** --makefile=*file*	Specify that *file* is the makefile to be processed. If this option is not used, then GMAKE looks for a file named MAKEFILE in the current directory.
-h **or** -help	Print out command line options
-I *DIRECTORY* **or** --include-dir=*DIRECTORY*	Search *DIRECTORY* for included makefiles.
-I **or** --ignore-errors	Ignore errors. Continue with MAKE process even when commands return error.
-j *num* **or** -jobs *num*	Allow a maximum of *num* jobs at once. Default is unlimited.
-k **or** --keep-going	Continue with MAKE process even if some targets cannot be made.
-l *load* **or** --load-average *load*	Specify maximum load allowed.
-n **or** --just-print **or** --dry-run **or** --recon	Print command lines, but do not execute commands.
--no-print-directory	Turn off “-w” family of options, even if turned on implicitly.
-o *file* **or** --old-file=*file* **or** --assume-old=*file*	Assume that *file* is very, very old, and do not remake it.
-p **or** --print-data-base	Print GMAKE’s internal predefined rules database which specify how to build target files from source files.

Option	Description
-q **or** --question	Do not build target, simply return exit code that specifies if target is up to date or not.
-r **or** --no-builtin-rules	Disable GMAKE's internal predefined rules which specify how to build target files from source files.
-s **or** --silent **or** --quiet	Do not echo commands as they are executed.
-S **or** --no-keep-going **or** --stop	Cancel "-k" family of options. Stop when target cannot be built.
-t **or** --touch	Touch targets (update time/date stamp) instead of building them.
-v **or** --version	Display GMAKE's internal version number
-w **or** --print-directory	Print the current directory
-W file **or** --what-if=*file* **or** --new-file=*file* **or** --assume-new=*file*	Consider file to always be new (always build target)
--warn-undefined-variables	Warn when undefined variables are referenced.

11.4.1 Additional Documentation

The GMAKE utility in the NUON SDK is the GNU MAKE utility from the Free Software Foundation.

Additional documentation on the GNU MAKE utility, not specific to NUON, is available separately. There is an MS Windows HELP file and also a Adobe Acrobat PDF file contained in the SDK's VMLABS\DOC folder.

11.5 Puffin Debugger

There are currently two flavors of the Puffin debugger for NUON.

The first is a console-based application named PUFFIN. This version is a 32-bit application, but is strictly console-based.

The PUFFIN2K version uses a graphic user interface and includes such enhancements such as assembly and C/C++ source level debugging. Please note that this was originally named PUFFINTK.

At one time, there was a third incarnation which used a simple MS Windows interface, PUFFINW, but this version is no longer supported.

11.5.1 Puffin & Tcl/TK

The PUFFIN2K version of the debugger uses the Tcl/TK scripting language and graphics tool kit, which must be installed separately from the NUON SDK.

The Tcl/TK package is created and distributed by Scriptics. The installation files can be downloaded from the VM Labs FTP site, or from the Scriptics web site:

http://www.scriptics.com

11.5.2 NUON Processor Emulation

Some versions of Puffin include a built-in MPE emulator that may be used for limited debugging if hardware is not available.

Puffin relies on the presence of the MD_PORT variable to determine if it should communicate with a development system or use the built-in machine emulation mode. If the MD_PORT variable is not set, then Puffin will use attempt to use emulation.

Please note that the built-in emulation only handles the MPE itself, not the entire machine. Any code that requires the presence of external hardware will not function properly.

11.5.3 Additional Documentation

The existing documentation on the PUFFIN debugger is available in the form of three separate documents:

- ***X-Lisp: An Object Oriented Lisp***
- ***X-Lisp Tutorial: An Introduction For C Programmers***.
- ***Puffin API***

Additional documentation regarding the updated version of the PUFFIN debugger will be available separately.

11.6 MLOAD Utility

The MLOAD tool is used for general purpose communication and data transfer between your host PC and your NUON development system. It includes commands for downloading program files or binary files, as well as performing register dumps and more.

MLOAD takes a command line of the form:

```
      mload [options] [file] [more options] [file]
```

MLOAD essentially processes the command line step by step from left to right. Each option either issues a command or sets the state for commands to follow. For example:

```
      mload –p3 –n game.cof -r -m
```

This is a common combination of options for MLOAD. It processes the commands "-p3" and "-n", then downloads the COFF file "game.cof". Then it continues to process the remainder of the commands on the command line.

The MLOAD program assumes that any filename specified by itself on the command line will be an appropriate NUON executable program file. If "**game.cof**" had been "**image.jpg**" instead, MLOAD would not have known what to do with the file and an error message would be shown.

11.6.1 MLOAD Command Options

The table below shows some of the more useful command line options for the MLOAD utility.

Option	Description
-?	Help. Display command line options.
-!	Reset the NUON chip
-~ *from : to : count*	Update host flash memory f(rom), t(o), c(ount) Note that this command affects' the host machine's flash memory, not the ACE360 board.
-a *file : address*	Load the specified *file* into host memory at the specified *address*. Hexadecimal addresses should be specified by a prefix of "0x"
-b *file : address*	Load the specified *file* into host memory at the specified *address* as a straight binary transfer. The file may contain anything. Hexadecimal addresses should be specified by a prefix of "0x"
-c	Compare contents of memory against file

Option	Description
-dd *address : num [options]*	Dumps memory starting from the specified address, via the data cache. Hexadecimal addresses should be specified by a prefix of "0x" You may also specify several options: m = Monitor progress when redirecting output to a file. i = Dump in binary image mode a = Dump in ASCII text mode w# = Specify number of scalars to write per line.
-df *address : num : file*	Dumps memory starting from the specified address and saves it to the specified *file*. Hexadecimal addresses should be specified by a prefix of "0x"
-di *address : num [options]*	Dumps memory starting from the specified address, via the instruction cache. Hexadecimal addresses should be specified by a prefix of "0x" You may also specify several options: m = Monitor progress when redirecting output to a file. i = Dump in binary image mode a = Dump in ASCII text mode w# = Specify number of scalars to write per line.
-dm *address : num [options]*	Dumps memory starting from the specified address. May not match contents of cache. Hexadecimal addresses should be specified by a prefix of "0x" You may also specify several options: m = Monitor progress when redirecting output to a file. i = Dump in binary image mode a = Dump in ASCII text mode w# = Specify number of scalars to write per line.
-dr	Dump registers for the currently selected MPE
-ds *address : num*	Disassembles *num* scalars of memory starting at *address*. Hexadecimal addresses should be specified by a prefix of "0x"
-f	Use the fast loader for the next file (default)
-fs	After everything else has been done, start up the file system server mode of the file server. See chapter 14 for more information on the file server.

Option	Description
-fsv	Same as the “-fs” option, except displays verbose tracking information about each file server packet.
-h	Stop the processor
-help *or* -?	Display list of available command line options
-ip *ipaddr*	Specify that a particular TCP/IP address should be used for subsequent commands.
-m	Monitor mpe exceptions after loading
-me *file*	After everything else has been done, start up the media server mode of the file server, using *file* as the specified data file. See chapter 14 for more information on the file server.
-mev *file*	Same as the “-me” option, except displays verbose tracking information about each file server packet.
-ms *blocksize*	Specify the block size for transfers from the media server
-msv *blocksize*	Same as the “-ms” option, except displays verbose information.
-n	Specify to MLOAD that a ROM-resident BIOS is already running so that it can take the appropriate actions when downloading a program.
-p *mpe*	Select the processor specified by *mpe* (0-3).
-r	Start processor running
-s	Use the slow loader for the next file
-t *seconds*	Request that MLOAD should profile program execution for the specified number of seconds. After the time has elapsed, a report will be printed. Should be used with the **–y** option to load program symbols.
-u *file*	Transfer an ACE360 or PPC860 Flash ROM memory update file. Note: This command is supported only by stub revisions dated April 6, 1998 or later.
-v	print the debug stub version number
-w	Wait one second
-wm *address : values*	Write the scalar values represented by *values* into memory at the specified address. Hexadecimal addresses should be specified by a prefix of “0x”.
-wd *address : values*	Same as the **–wm** option, except goes through the data cache.
-y *programfile*	Load program symbols from the specified COFF file. Used in conjunction with the –p option for profiling.

For example:

```
mload -p0 foo.mpo -r -p1 bar.mpo -r -m
```

This will load FOO.MPO into the NUON's mpe 0 processor and start it running. It will also load BAR.MPO into the NUON's mpe 1 processor and start it running, then it will monitor processor exceptions until both of those processors halt.

11.6.2 Using MLOAD For Debugging

The MLOAD tool has many features which can be used for debugging certain types of problems.

11.6.2.1 Monitor Processor Exceptions Option

The "-m" option tells MLOAD to monitor the NUON system for processor exceptions, and if one is detected, to print out a detailed register dump that describes the processor state at the time the error occurred. This gives you output that looks like this:

```
Serving files and monitoring exceptions (type 'q' to exit)...
MPE 3 halted: excepsrc=00000100
  mpectl: 02008000, rz: 80010812, sp: 20100ff0
  pcexec: 8001080a, pcroute: 00000000, pcfetch: 00000002
  excepsrc: 00000100, excephalten: ffffffff, cc: 00000021
  intsrc: a20a03f0, intctl: 00000000
  inten1: a0000008, intvec1: 807690a0, rzi1: 80010000
  inten2sel: 00000004, intvec2: 807671a0, rzi2: 8076f13a
  mdmactl: 00000020, mdmaptr: 20500f20
  odmactl: 00000020, odmaptr: 00500f30
  commctl: 80452048, comminfo: 00000000
  commxmit: 80000200 03110880 00000000 00000000
  commrecv: 02000001 00000000 4a6f686e 204d6174
  r0: 00452048 r1: 20500510 r2: 80028078 r3: 00000000
  r4: 00000048 r5: 00000000 r6: ffffffff r7: 80000874
  r8: 800013c0 r9: 800013c0 r10: 00000000 r11: ffffffff
  r12: 40426c50 r13: f9ced8a2 r14: 40398f7f r15: 0a1f1efc
  r16: 00452048 r17: 00000000 r18: 00000000 r19: 00000000
  r20: 4005a14d r21: 8931a901 r22: 80000298 r23: 00000000
  r24: 408591d3 r25: 0000007e r26: 00000ba4 r27: 0000007e
  r28: 00000003 r29: 80000178 r30: 8075fff0 r31: 8075ff80
```

Figure 11-1 — MLOAD Register Dump

This is very raw information, but in many cases, it contains most of what you need to figure out what went wrong.

11.6.2.2 Register Dump Option

Another important MLOAD option is the register dump command. There are two variations.

The "-dr" variation will provide a complete dump of all of the processor's registers as shown above, but may not be completely transparent to any program that is currently running. It’s normally best to use “-dr” only when the program has crashed or locked-up.

The "-drq" variation provides only the registers shown in **boldface** in Figure 11-1, but is more transparent to any program that is currently running.

Remember that in order to address an MPE other than 0, you must first specify which one you want using the "-p*num*" option.

11.6.2.3 Deciphering The Register Dump

When you're attempting to find out the cause of a crash bug, the important things to look at first are the value of the **excepsrc**, **pcexec**, **pcroute**, and **pcfetch** registers.

The **excepsrc** value indicates what processor exception(s) occurred. It's important to note that more than one may have occurred simultaneously. For example, a value of 0x180 means that there were two different errors. Bit 7 indicates a data read/write address error. That means that a bad address was used for a load or store instruction. Bit 8 indicates that there was an address error on an instruction fetch.

For other bits, see the description of the **excepsrc** register in your ***Programmer's Guide*** in the section labeled **MPE Register Set Reference**.

The **pcexec** register indicates the address of the instruction that was being processed when the error occurred. This should contain a valid address located somewhere in SYSRAM, SDRAM, or internal MPE memory. In most cases, if **pcexec** contains an invalid address, the reason is that the MPE had been idle and was told to start executing at an invalid address.

Sometimes if **pcexec** is pointing to a position immediately following an **rts** instruction, that is an indication that the stack or the stack pointer was corrupted in such a way that the return address pulled off the stack was invalid.

The **pcfetch** and **pcroute** registers indicate the addresses of the next instruction packets that are going to be read and executed. These should contain valid addresses in SYSRAM, SDRAM, or internal MPE memory. Normally, they should either contain address that immediately follow the value in the **pcexec** register, or else they should indicate the address following a branch operation, subroutine call, jump instruction, return from subroutine, or return from interrupt. If either of these values is not a valid address, that would normally indicate that either a return address on the stack has been corrupted, the stack pointer has been corrupted and no longer points to the return address, or that a bad pointer address in a register is being used as the target for a **jump** or **jsr** call.

11.6.2.4 Disassembly From Memory

The MLOAD tool can also disassemble blocks of memory. The syntax of this command is:

```
      MLOAD -p3 -ds <startaddress>:<size>
```

The *startaddress* parameter is always provided in hexadecimal, but no leading "$" or "0x" is required. The *size* parameter is provided in decimal. So for example, the following command would disassemble 20 bytes starting from address 0x80010000:

```
      MLOAD -p3 -ds 80010000:20
```

The main disadvantage of this disassembly is that it does not provide symbols. However, you can manually match addresses against a list of symbols obtained using another tool such as **vmnm** or **coffdump**.

You can tell MLOAD to load symbols from the COFF file for the current program, provided you actually have the file and that it contains symbols. This is done by adding "-y *program.cof*" to your command line, where *program.cof* is the name of the file. When you do this, the disassembly will use those symbols where applicable.

11.6.2.5 Memory Dump

Another useful feature is the ability to dump a block of memory using MLOAD. There are several variations on this command. They are:

Basic Memory Dump: MLOAD -p3 -dm <address>:<size>

I-cached Memory Dump: MLOAD -p3 -di <address>:<size>

D-cached Memory Dump: MLOAD -p3 -dd <address>:<size>

Memory Dump To File: MLOAD -p3 -df <address>:<size>:<file>

The *address* parameter specifies the starting address in hexadecimal. No leading "$" or "0x" is required. The *size* parameter indicates the number of scalars (4 byte blocks) to be displayed. For the last variation, the *file* parameter indicates the filename that will be created.

11.6.2.6 Other Useful Programs

Other programs which are useful in combination with MLOAD for debugging are VMDISASM, MTRACE, VMNM, and COFFDUMP. Please see the individual sections in this chapter on these programs for further information.

11.7 MTRACE Utility

The MTRACE utility is used to execute a call stack trace. This is normally used when a program has frozen or crashed to determine the sequence of program functions that preceded the problem.

Option	Description
-b	Specify that MTRACE should display a stack trace of the background task of the Presentation Engine.
-f	Specify that MTRACE should display a stack trace of the foreground task of the Presentation Engine.
-p *mpe*	Specify the processor that owns the stack that you want to trace. In most cases, this will be MPE 3.
-a *address*	Specify the starting address for the trace.

For example:

```
      mload -p0 foo.mpo -r -p1 bar.mpo -r -m
```

11.7.1 Presentation Engine Tasks

The Presentation Engine is the kernel that controls the system when the NUON's DVD player firmware is activated. It provides a limited multi-tasking system that has a foreground task and a background task. The "**-b**" and "**-f**" options are used to specify which task should be traced. These options are only applicable when an application is using the Presentation Engine.

11.7.2 Output

Each entry in the stack trace is of the form:

```
Label + offset "fileName", line n
```

The *label* shown will be the closest label that precedes the actual address. The offset from that address will be shown as a hexadecimal number. For example:

```
_main+0000002c "main.c", line 12
```

This says that the program counter value that was found on the stack is pointing to an instruction that is 0x0000002C bytes past the address specified by the *_main* label, and that this was at line 12 of the “**main.c**” source code file.

The symbols and line number information are taken from the COFF file specified on the command line. It is often useful to try several different COFF files if the program makes use of several separately loaded files.

11.8 VMAR Utility

The VMAR utility allows you to create and modify library archive files. An archive is a single file that contains a group of smaller files, usually object modules containing compiled code and data that will be accessed by the linker.

A library archive created by VMAR is intended to make life easier for the programmer and the linker. The linker is designed to search an archive file, determine what pieces of code and data are contained in each individual module, and then extract only those modules that are required to successfully link a program.

In order to allow more efficient searching by the linker, the VMAR program can create an index of all the symbols contained in the individual object modules within the archive. Once created, this index is automatically updated when object modules are added or deleted.

Other types of files may also be stored in VMAR archives, but unless they are intended to be used by the linker, this may not be the best choice, as no compression is done by VMAR.

The format of the VMAR command line is:

```
      vmar [options][modifiers] ARCHIVE [member…]
```

The command *options* and *modifiers* are described in the tables below. The *archive* parameter indicates the filename of the archive to be used or created. The *member* parameter contains a single module name, or a list of module names.

Please note that all command line options and modifiers are case-sensitive, and only one command option may be specified per command line. Note that the leading “-” before a command option is accepted, but not required.

Option	Description
d	Delete the object module *member* from the archive. Multiple members may be specified. Examples: `vmar –d libmath.a cosine.o` `vmar –dv libmath.a cosine.o sine.o`

Option	Description
m	Moves the module(s) specified by *member* to the end of the archive. If certain symbols are defined in more than one module within a library, the order of those modules within the archive can make a difference in how programs are linked. Using the ‘a’, ‘b’, or ‘i’ modifiers with the “m” option will allow you to move the module to a specific location rather than the end of the archive. The example below would operate on the LIBMATH.A library, and move the module named fabs.o to the position following the sine.o module: `Example:     vmar -ma sine.o libmath.a fabs.o`
p	Prints the specified modules to standard output. This would be used for text files such as source code which may be contained in an archive. This command does not produce readable output for a compiled object module. `Example:     vmar p sine.c libmath.a`
q	Quick Append. Add the specified module to the end of the archive, without checking if there is another module of the same name already. The ‘a’, ‘b’, or ‘i’ modifiers do not affect this operation. `Example:     vmar q libmath.a sine.o`
R	Insert the specified module into the archive, deleting any existing module with the same name. By default, modules are added to the end of the archive, but using the ‘a’, ‘b’, or ‘i’ modifiers will allow you to move the module to a specific location. Both examples below would operate on the LIBMATH.A library and add a module named sine.o. The first example would add it to the end of the library. The second example would add it to the library immediately before the arctan.o module: Example: `vmar r libmath.a sine.o` `vmar ri arctan.o libmath.a sine.o`
S	Create or update the library’s object module index. Also available as a modifier which can be specified along with another command.

Option	Description
T	Display a table listing of the modules within the archive. The ‘v’ modifier will cause the permissions flag, timestamp, owner, group, and size information to be printed as well. Example: `vmar q libmath.a sine.o`
X	Extract the specified module from the library to an external file. If no module is specified, all modules within the library are extracted. If an external file with that name already exists, it is overwritten. The archive contents are not changed. Example: `vmar x libmath.a sine.o`

Modifier	Description
a *relpos*	Specify that the operation should happen at the position within the archive immediately after the specified *relpos* module. This modifier can be used with the “r” and “m’ command options. Example: `vmar ra arctan.o libmath.a sine.o`
b *relpos* **or** i *relpos*	Specify that the operation should happen at the position within the archive immediately before the specified *relpos* module. This modifier can be used with the “r” and “m’ command options. Example: `vmar rb arctan.o libmath.a sine.o` `vmar ri arctan.o libmath.a sine.o`
c	Disable warning when it’s necessary to create the archive in order to make an update. Normally, if the archive does not exist, it is created when you attempt to add a module using the “r” command option, but a warning is issued. Using this modifier will disable the warning.
i	Specify that the operation should happen at the position within the archive immediately before the specified *RELPOS* module. This modifier can be used with the “r” and “m’ command options. Example: `ar rb arctan.o libmath.a sine.o`
o	Preserve the original dates of modules when extracting them. Used with the “x” command option.
s	Create or update the library’s object module index.

Modifier	Description
u	Conditionally add the specified files to the archive only if the timestamp is newer than the version of the file already in the archive.
v	Turns verbose mode on. More information gets printed about whatever command is being executed.
V	When used with any command option, causes the version of VMAR to be printed, instead of executing the command.

VMAR also has a mode that can be driven by either interactive commands or a script file. More details about this mode are listed below. The command line format for this mode is:

```
vmar –M [<script]
```

Note that the script file is specified using standard input redirection. During interactive use, VMAR prompts for input (the prompt is "AR >"), and continues executing even after errors. If you redirect standard input to a script file, no prompts are issued, and VMAR abandons execution (with a non-zero exit code) on any error.

The command language is not designed to be equivalent to the command-line options; in fact, it provides somewhat less control over archives. The purpose of the command language is to ease the transition for developers who already have scripts written for the MRI "librarian" program.

The command language syntax works according to the following rules:

- Commands are recognized in upper or lower case. For example, *LIST* is the same as *list*.
- Only one command per line, located at the beginning of the line.
- Empty lines are ignored.
- Comments begin with "*" or ";", everything afterwards on that line is ignored.
- A list of file names or module names may be separated by either spaces or commas.
- The "+" character is used to continue a command on the following line.

A basic reference to the available command is provided below. Note that most commands operate on the *current* library, which is the one last specified using either the OPEN or the CREATE command.

Command	Description
addlib *library*	Reads the archive specified by *library* and copies all modules into the current archive
addlib *module*	Adds the specified *modules* to the current archive.
Clear	Deletes all modules from the current archive.
Create *archivefile*	Creates a new archive with a filename of *archivefile*.
Delete *module*	Deletes the specified module from the current archive
Directory *archive* [(*modules)*] [*outputfile*]	Lists the modules contained in the specified *archive* file. If a module list is specified within parenthesis following the archive name, then only modules matching the list will be shown. The output may optionally be directed to a file by specifying the *outputfile* parameter.
End	Exit from VMAR with an exit code of zero. Does not automatically save changes to the current archive. If you do not use SAVE before END, then your changes are lost.
Extract *modulelist*	Extract the specified module(s) from the current archive and save them into the current disk directory.
List	Display the contents of the current archive.
Open *archive*	Opens a new archive.
Replace *module*	Replaces the specified module within the archive with a file of the same name in the current disk directory.
Save	Saves changes to the current archive. Any changes to the archive will not be permanent until this command is used.
Verbose	Turns on the verbose mode of each command.

11.9 VMNM Utility

The VMNM utility displays information about symbols contained in executable program files, object modules or library archives. This can be one of the most useful tools for debugging.

The format of the VMNM command line is:

```
vmnm [options] [objfiles]
```

The *objfiles* parameter is a list of executable program files created by the linker, object modules created by the assembler or compiler, or library archives created with VMAR. The command *options* are described in the table below. Please note that all commands and modifiers are case-sensitive:

Option	Description
-A **or** -o **or** --print-file-name	Print the module name next to each symbol, rather than just once at the top of each group.
-a **or** --debug-syms	Display special debugger-only symbols normally not listed.
-B	Specify BSD-format output. Same as “–f bsd”.
-C **or** --demangle	Demangle encoded C++ names into user-level names.
-D **or** --dynamic	Display dynamic symbols
-f *format*	Specify output format. The format parameter should be “bsd”, “sysv”, or “posix”. The default format is “bsd”.
-g **or** --extern-only	Display only externally defined symbols
-n **or** --numeric-sort	Sort symbols according to address, rather than name
-p **or** --no-sort	Do not sort symbols. Output them in order found.
-P **or** -portability	Use Posix.2 format output. Same as “-f posix”.
-s **or** --print-armap	When input file is a library, include the archive object module index.
-r **or** --reverse-sort	Reverse the order of the symbol sort.
--size-sort	Sort symbols by size. Size is determined by the next highest symbol.
-t *radix* **or** --radix=*radix*	Use radix for printing symbol values. Valid values are “d” for decimal, “o” for octal, or “x” for hexadecimal.
--target=*bfdname*	Specify a non-standard object file format
-u **or** --undefined-only	Display only undefined symbols (those external to object module or library).
--version	Display the internal version number of VMNM.
--help	Display available command line options for VMNM.

11.9.1 VMNM Output Format

The output format of VMNM changes depending on the command line options. The default format is “bsd”. Examples of the available output formats are shown below.

-B or **–fbsd** (BSD Format) = <value> <flag> <symbolname>

```
8001b382 T _CopyFromMPE
00000000 a _FALSE
00000a10 A _FXCode_size
8009ffb0 ? _FXCode_start
8009b124 D _OutputPeriod
8009b104 d _PCMCallDataReq0
```

-P or **–fposix** (Posix format) = <symbolname> <flag> <value>

```
_CopyFromMPE T 8001b382
_FALSE a 00000000
_FXCode_size A 00000a10
_FXCode_start? 8009ffb0
_OutputPeriod D 8009b124
_PCMCallDataReq d 08009b104
```

-fsysv (SYSV Format) = <symbolname> <value> <flag> <additional info>

```
Symbols from vmballs.cof:

Name                  Value   Class     Type      Size   Line
Section

_CopyFromMPE        |8001b382|   T  |            |      |     |
_FALSE              |00000a10|   a  |            |      |     |
_FXCode_size        |00000a10|   A  |            |      |     |
_FXCode_start       |8009ffb0|   ?  |            |      |     |
_OutputPeriod       |8009b124|   D  |            |      |     |
_PCMCallDataReq0    |8009b104|   d  |            |      |     |
```

11.9.2 Symbol Flags

Regardless of the output format, each symbol is given a flag consisting of a single character that indicates what program segment it belongs to. A lowercase flag indicates a symbol that is local to the module in which it is defined. An uppercase flag indicates a symbol that is defined as global.

Symbol Flag	Meaning
"?"	Symbol defined as external, but the segment is not defined within the current module.
"a" or "A"	Symbol value is absolute.
"b" or "B"	Symbol is located in the "BSS" segment.
"c" or "C"	Symbol is located in the "common" segment.
"d" or "D"	Symbol is located in the "data" segment
"g" or "G"	Symbol is located in the "sdata" segment
"r" or "R"	Symbol is located in the "Rdata" or "rodata" segment
"s" or "S"	Symbol is located in the "sbss" segment.
"t" or "T"	Symbol is located in the "text" segment
"u" or "U"	Symbol type is not defined within the current module.
"w" or "W"	Symbol is located in the "dtram" segment.

“x” or “X”	Symbol is located in the “dtrom” segment.
“y” or “Y”	Symbol is located in the “iram” segment.

11.9.3 Segment Descriptions

Segment Name	Description
“bss” or “sbss”	BSS stands for “Block Storage Segment” and SBSS stands for “Small BSS”. These segments normally contain storage space for variables which are not pre-initialized.
“common”	Reserved storage space for items that are not pre-initialized. Basically the same as the BSS segment.
“rdata” or “rodata”	ROM-Data or “Read-Only Data”. This segment normally contains pre-initialized data intended to be read-only (such as would be placed into ROM).
“text”	Contains executable program code.
“data” or “sdata”	Contains pre-initialized data. The “sdata” segment is the “Small data” segment.
“dtram”	This segment defines the data area of an MPE’s local data memory space.
“dtrom”	This segment defines the data area of an MPE’s ROM address space.
“iram”	This segment is located at the beginning of an MPE’s local instruction memory space.

11.9.4 Getting a Symbol Map

Perhaps the most common use of VMNM is to extract a list of symbols from an executable COFF file. This would be done via the command:

```
vmnm –fsysv –n program.cof >map.txt
```

This reads symbols from **PROGRAM.**COF, sorts the output according to the symbol’s numerical value, and formats the entire list into a table. Then the results are redirected from the console to the **MAP.TXT** file.

11.9.5 Symbol Names

There are a few things to keep in mind about the symbol names displayed by VMNM:

- Symbols created by the C/C++ compiler will have a leading underscore character. So if you’re looking for the symbol matching the “`draw_water`” function, you really need to look for “`_draw_water`”.

- Function names in C++ may be mangled by the compiler. Use the “**-**C” option to demangle the function names.

- Non-global symbols may be used and therefore will appear in the output of VMNM over and over again, making debugging difficult. To avoid this problem, simply use more unique symbol names.

11.9.6 Symbol Values, Object Modules, & The Linker

When reading the output from VMNM, it is important to know how something about how symbol values work and how they are manipulated by the linker.

Within an object module, either as a separate file or a module within a library archive, symbols which are defined within the object module itself are considered to have values that are relative to the base address of the segment in which the symbol is located, within that particular module.

In other words, if the “`program.o`” object module contained a function named “`flowing_water`” that is located 120 bytes from the beginning of the “text” segment, then the “`flowing_water`” symbol will be given a value of 120.

On the other hand, if the “`graphics.o`” object module contained a function named “`draw_water`” that is also located 120 bytes from the beginning of the “text” segment within that object module, then the “`draw_water`” symbol will also be given a value of 120.

The symbol values overlap because each object module was created individually without any knowledge of the other. It’s OK, because object modules are not designed for direct usage. They are supposed to be processed by the linker to create an executable program file.

The linker is responsible for taking a number of separate object modules and combining them together to create a program file, all while making sure that the “`flowing_water`” function and the “`draw_water`” function are given their own space that doesn’t overlap.

(This is a very simple explanation of what the linker does. We’re simplifying quite a bit here, so don’t take this as an exact description of the linker’s behavior.)

Once the linker knows what object modules must be combined to create the program file, it copies the “text” segment from each one into the output file. This process is repeated for the “data” segment, the “bss” segment, and any other segments found within the object modules.

In the process, the linker changes symbol values and references to reflect how things have been combined together in the output file. Symbol values and references are defined as relative positions from the start of their respective segments.

If the linker is creating a executable program file that is meant to be loaded to an absolute fixed address, as is the case with NUON, there is one last step. The linker must now step through the list of symbols and change each symbol value or reference into an absolute address relative to the load address that has been specified.

11.10 COFFDUMP Utility

The COFFDUMP utility displays information about the contents of a COFF object module or executable program file.

The format of the COFFDUMP command line is:

```
coffdump [options] [objfiles]
```

The *objfiles* parameter is a list of executable program files created by the linker, object modules created by the assembler or compiler, or library archives created with VMAR.

The command *options* are described in the table below. Please note that all commands and modifiers are case-sensitive:

Option	Description
-h	Show headers for each program segment defined in file. See the description of the output below for more information.
-r	Dump information about any objects in the file which require relocation (linking)
-s	Dump all symbols described within the file

These options may be provided in any order. The order does not affect the program output.

11.10.1 COFFDUMP Output

Each of the command line options for COFFDUMP produces a different type of output.

11.10.1.1 Program Section Information

Here's the information generated by using the "-h" option with an executable program file:

```
C:\vmlabs\sample\Miscellaneous\showpic>coffdump -h showpic.cof
     showpic.cof
16 sections
41185 symbols
Entry at        0x80010000
Target: mpe3
#    Name     Addr        Rta         Size    Nrel    Nlnn   Flags
00   text    :80010000   80010000    107112  2082    20209  00200060
01   data    :8002a270   8002a270    1386976 30      0      00100040
02   bss     :8017cc50   8017cc50    176     0       0      00100080
03   comm    :8017cd00   8017cd00    352     0       0      00040080
04   ctors   :8017ce60   8017ce60    12      1       0      00040040
05   dtors   :8017ce6c   8017ce6c    8       0       0      00040040
06   PATCH   :8017ce80   8017ce80    752     50      29     00100060
07   rodata  :8017d170   8017d170    2600    208     0      00040040
08   bicC    :8017dba0   8017dba0    3576    21      143    00200020
09   bicI    :8017e9a0   8017e9a0    6736    32      473    00200020
10   bic1    :80180400   80180400    2688    9       117    00200020
11   bic0    :80180e80   80180e80    3032    7       124    00200020
12   biostab :80181a60   80000000    6592    0       0      00100080
13   cookie  :80183420   8000f000    336     0       0      00040080
14   biostb2 :80183570   20100c80    528     0       0      00100080
15   heap    :80183780   80183780    16      0       0      00200080
```

First it shows the name of the file, in case you've specified more than one. Then it shows how many program sections there are, how many symbol entries, what the runtime start address should be, and finally the NUON MPE which is the target.

The remainder of the output is a list of information for each of the program sections in the file. For each section, the information from the table below is provided:

#	Section number
Name	Section name (up to 8 characters)
Addr	Load address for section
Rta	Runtime address for section. This is usually the same as the load address, but may be different in the case of a code overlay which will be copied to the runtime address when needed.
Size	Size in bytes of the section
Nrel	Number of relocated items
Nlnn	Number of line numbers (source-level debugging information)
Flags	The low word indicates the type of program section(s) that are represented: 0x20 = text (program code) 0x40 = data (initialized data) 0x80 = bss (block storage segment, otherwise known as uninitialized data space)

	The high word indicates the alignment requirements for the section.

11.10.1.1.1 A Very Special Note About The “heap” Section

Please note that the “heap” section is normally the last one in a COFF file. This represents the memory heap that is used to satisfy memory allocation requests for functions like ***malloc()*** and the C++ **new** operator. Don’t be alarmed because the size is just 16 bytes, because this is designed to trigger the C runtime startup code into expanding the heap at runtime. It will expand the heap to use all available memory from the end of the program’s load area to the bottom of the stack.

This means you can also control the size of the heap by creating your own custom heap segment in your program file. If the heap isn’t 16 bytes long then the C runtime startup code will simply use whatever size is provided.

11.10.1.2 Relocation Information

Here’s the information generated by using the “**-**r” option:

```
C:\vmlabs\sample\Miscellaneous\showpic>coffdump -r graphics.o

      graphics.o
1 sections
885 symbols
Relocations for text
0000003a    __MemLocalScratch 0802 00000000
0000006a    _gl_screenbuffers 0802 00000000
00000070          __DMALinear 0802 00000000
0000007a       _gl_drawbuffer 0802 00000000
00000080        __DMABiLinear 0802 00000000
00000138       _gl_drawbuffer 0802 00000000
00000142    _gl_displaybuffer 0802 00000000
0000014e    _gl_screenbuffers 0202 00000000
0000015a _mmlSimpleVideoSetup 0802 00000000
00000164           _gl_sysRes 0802 00000000
0000019a    _gl_screenbuffers 0802 00000000
000001a0           _gl_sysRes 0802 00000000
000001a6    _gl_displaybuffer 0802 00000000
000001ac       _gl_drawbuffer 0802 00000000
000001b2 _mmlInitDisplayPixma 0802 00000000
000001fa _mmlSimpleVideoSetup 0802 00000000
```

The first value for each item is the offset within the section where the relocation is to be applied. This is followed by the first 20 characters of the symbol name.

Next is a bitmapped flag value that defines the relocation style and method. The last value is an adjustment value that is added to the symbol value to obtain a relocation argument.

11.10.1.3 Symbol Dump

Here's the information generated by using the "-s" option:

```
C:\vmlabs\sample\Miscellaneous\showpic>coffdump -s graphics.o
      graphics.o
1 sections
885 symbols
.file                    00000000        [       D]      103    0 +1
gcc2_compiled.           00000000        [    text]      3      4 +0
___gnu_compiled_c        00000000        [    text]      3      4 +0
___int32_t               00000000        [       D]      13     4 +0
___uint32_t              00000000        [       D]      13     e +0
.
.
.
```

The first item shown for each symbol is the first 20 characters of the symbol name. This is followed by the symbol value. The value in brackets indicates the section where the symbol is defined (or "D" for a symbol that's not part of a section).

The next value indicates the "storage class" of the item. This is followed by a value representing the symbol type. The last value is an auxiliary record.

11.11 VMSTRIP Utility

The VMSTRIP utility is designed to remove symbols and debugging information from an executable COFF file. There are main two reasons for doing this. First, this frequently reduces the size of the COFF file considerably. Second, it makes it more difficult for someone to disassemble and decipher your code.

The format of the VMSTRIP command line is:

```
       vmstrip [options] coff-file
```

The command *options* are described in the table below. Please note that all commands are case-sensitive:

Option	Description
-D	Converts symbol references to section references and removes all symbols.
-E	Keep external symbol references
-F	Force removal of everything. By default, only undefined symbols are kept in the output file
-k symbol	Keep the specified *symbol* even if it is not referenced.
-o filename	Specify the output filename. By default, if none is specified, the output file name is STRIP.OUT in the current directory.

Option	Description
-r symbol	Removes the specified *symbol* even if it would be removed otherwise.

11.12 VMDISASM Utility

The VMDISASM utility reads an executable COFF program file and disassembles a specified section.

The format of the VMDISASM command line is:

```
vmdisasm [options] inputfile range
```

The command line options are described in the table below. Please note that all commands are case-sensitive:

Option	Description
-a	Show real addresses
-n	Display labels as addresses

Please note that some versions of the VMDISASM tool display the wrong command line format when showing a list of the command line options. However, the format above is correct.

The *range* parameter defines the starting address and size of the range you wish to disassemble. Please note that the address is treated as decimal unless you add "0x" as a prefix. (Unlike many other tools which assume addresses are always in hexadecimal regardless of prefix.)

Please note how the output changes when different options are used. For example:

```
C:\vmlabs\sample\showpic>vmdisasm -a -n showpic.cof 0x80010000:20
80010000        mv_s #$8017cc0c,r31
80010006        ld_s (r31),r31
80010008        nop
8001000a        mv_s #$00000000,r30
8001000c        st_s #$00000000,acshift
80010010        jsr #$80000378

C:\vmlabs\sample\showpic>vmdisasm -a showpic.cof 0x80010000:20
80010000        mv_s #$8017cc0c,r31
80010006        ld_s (r31),r31
80010008        nop
8001000a        mv_s #$00000000,r30
8001000c        st_s #$00000000,acshift
80010010        jsr pixgo+fffe2338

C:\vmlabs\sample\showpic>vmdisasm showpic.cof 0x80010000:20
+ffff1fc0       mv_s #$8017cc0c,r31
```

```
+ffff1fc6       ld_s (r31),r31
+ffff1fc8       nop
+ffff1fca       mv_s #$00000000,r30
+ffff1fcc       st_s #$00000000,acshift
+ffff1fd0       jsr pixgo+fffe2338

C:\vmlabs\sample\showpic>vmdisasm -n showpic.cof 0x80010000:20
80010000        mv_s #$8017cc0c,r31
80010006        ld_s (r31),r31
80010008        nop
8001000a        mv_s #$00000000,r30
8001000c        st_s #$00000000,acshift
80010010        jsr #$80000378
```

11.13 VMOCOPY Utility

The VMOCOPY utility copies the contents of one object module to another, optionally performing a format conversion in the process.

The format of the VMOCOPY command line is:

```
      vmocopy [options] infile [outfile]
```

The *infile* parameter specifies the source file. The *outfile* parameter is optional. If present, it defines the name of the output file. If absent, then a temporary file is created, and after processing is finished, the input file is overwritten.

The command *options* are described in the table below. Please note that all commands are case-sensitive:

Option	Description
--adjust-start=*increment*	Adjust the starting address of the file by adding *increment*. Note that some object file formats may not support this.
--adjust-vma=*increment*	Adjust the address of all sections, as well as the start address, by adding *increment*. Note that some object file formats may not support this.
--adjust-warnings	Issue warnings is a section specified via the "—adjust-section-vma" option does not exist. (default)
-b *byte* **or** --byte=*byte*	Discard file contents, except for every *byte*th byte. The *byte* parameter should be in the range of 0 to *interleavefactor*-1, where *interleavefactor* is the value specified using the "-i" command option, or the default value of 4. This option is used to create source files for multiple ROM or EPROM chips.

Option	Description
--debugging	Convert debugging information, if possible. Default for this option is FALSE.
-F fmt **or** --target=*fmt*	Specify that the original file uses the object code format specified by *fmt*, and rewrite it in the same format. If this option is not used, VMOCOPY will attempt to deduce the source format. See the *File Format Types* section below for more information.
-g **or** --strip-debug	Remove debugging symbols only
--gap-fill=*value*	Fill gaps between sections with the specified *value*. This is done by increasing the size of the section with the lower address, then filling in the gap with bytes containing *value*.
--help	Display help about command line options
-I *fmt* **or** --input-target=*fmt*	Specify that the input file uses the object code format specified by *fmt*. If this option is not used, VMOCOPY will attempt to deduce the source format. See the *File Format Types* section below for more information.
-i *interleavefactor* **or** --interleave=*interleavefactor*	Specify the interleave factor to use in conjunction with the “-b” command option..
-K *symbolname* **or** --keep-symbol=*symbolname*	Keep only *symbolname* from the source file. May be used multiple times to specify multiple symbols.
-N *symbolname* **or** --strip-symbol=*symbolname*	Remove *symbolname* from the source file. This option may be used multiple times for multiple symbols. May be combined with –K option.
--no-adjust-warnings	Disable warnings in the event that a section specified via the “—adjust-section-vma” option does not exist.
-O *fmt* **or** --output-target=*fmt*	Specify that the output file uses the object code format specified by *fmt*. If this option is not used, VMOCOPY will use the source format. See the *File Format Types* section below for more information.
--pad-to=*address*	Pad the output file until *address* is reached. The size of the last section is increased to match. The extra space is filled in with the value specified for the “—gap-fill” option, or the default of zero.

Option	Description
-R *sectname* **or** --remove-section=*sectname*	Remove the section specified by *secname* from the output file. This option may be used multiple times for different sections. Note that using this option incorrectly may make the output file unusable.
--remove-leading-char	Some object file formats use a special symbol at the start of every symbol, such as an underscore. This option will remove the first character from all global symbols.
-S **or** --strip-all	Remove all symbols.
--set-section-flags= *section=flags*	Set the flags for the specified *section*. The *flags* argument should be a comma-separated string of flag names: alloc, load, readonly, code, data, rom. Not all flags are supported by all object formats.
--set-start=*value*	Set the starting address of the file to *value*. Note that some object file formats may not support this.
-v **or** -verbose	Verbose output. List everything that is modified. For archives, this lists all members.
-V **or** --version	Display VMOCOPY's internal version number
-X **or** --discard-locals	Remove compiler-generated local symbols (usually starting with "L" or ".")
-x **or** --discard-all	Remove all non-global symbols.

11.13.1 File Format Types

Certain command options require that a file format be specified. This should be one of the formats shown in the table below:

Format Name	Description
oz-local-coff	NUON-specific flavor of the COFF format (default file format)
srec	S Records

12. NUON Development System Documentation

This chapter will provide an overview of other pieces of documentation that are available for the tools and libraries that make up the NUON SDK.

Basic introductory documentation for the main tools is provided in chapter 11. For some tools, no further documentation is required. For others, the information in chapter 11 just scratches the surface.

Almost all documents are available in an online readable form, either as Adobe Acrobat files which can be viewed with the Adobe Acrobat reader, or else as HTML files which can be viewed with a web browser such as Microsoft Internet Explorer or Netscape Navigator. Contact VM Labs Developer Support if you are unable to view these files.

Some of the documents mentioned in this chapter are also available in hardcopy.

For each document, we'll list the title, the format(s) available, and give a brief description of the contents.

Documents in online format are always located in the VMLABS\DOC directory or a subdirectory, with the exception of the main SDK README file, which is located in the VMLABS directory.

The documents are grouped into separate sections for ***Tools***, ***System & Hardware***, and ***Libraries***. Documents regarding tools that are specific to a particular library are listed along with the library.

12.1 Tools

Chapter 11 includes a basic introduction and reference for several tools in the NUON SDK. Additional documentation may also be found in the following documents.

Please note that the titles of some documents may be changed to specify "NUON" rather than "Merlin".

Title	Description	Format
Optimizing Your LLAMA	A step-by-step tutorial to hand optimizing assembly code for NUON using the Llama assembler	HTML
LLAMA User's Manual	This tells you everything there is to know about the Llama assembler.	PDF

Title	Description	Format
NUON GCC README	NUON-specific details about the C/C++ compiler	HTML
Using GNU CC	The basic non-NUON specific user's manual for the C/C++ compiler.	PDF
GNU Make Manual	Details about the GNU Make utility.	PDF
Xlisp Manual	An introduction and manual for the X-Lisp programming language used by the Puffin debugger.	PDF
Xlisp Tutorial	A tutorial showing how to get the most out of the Xlisp programming language built into the Puffin debugger.	PDF
Puffin API	Basic documentation for the Xlisp API commands provided with the Puffin debugger.	PDF

12.2 System & Hardware

Title	Description	Format
BIOS Overview		PDF
NUON Multi-Media Architecture MMP–L3C Specifications	The basic documentation for the NUON processor. Includes an assembly language reference, register documentation, and tons of other important details.	Printed

12.3 Libraries

Title	Description	Format
Merlin Troubleshooting	This document discusses a number of programming related problems and potential solutions	PDF
Jeff Minter's Object List API	Document describing the C/C++ level API for Jeff Minter's Object List sprite library	PDF
Merlin 2d Library	API for 2d graphics (lines, circles, text, etc.)	PDF
Merlin 3d Library	Details about the original Merlin 3d library.	PDF
mGL 3D Graphics Library	Details about the mGL 3D Graphics Library	PDF
M3DL Graphics Library	Details about the M3DL 2D & 3D Graphics Library	PDF

Title	Description	Format
Merlin Utility Functions Programmer's Manual	Details about the Merlin Utilities library	PDF
Merlin Synth API	Details about the Merlin Synth, a General MIDI compatible synthesizer.	PDF

12.3.1 Sample Program Source Code

At the current time, there is no individualized documentation for the various sample programs in the NUON SDK. We do hope to address this situation in the future. The most likely scenario will be that each sample program will come with an HTML document that provides a basic overview of the source code.

This page intentionally left blank.

13. Running Your First Program

This section will walk you through the steps involved in compiling and executing your first NUON sample program.

At this stage, you should have all of your hardware connected and configured as described in the ***Hitchhiker's Guide To NUON*** document. You must also have downloaded and installed the NUON SDK.

13.1 Check your configuration

If everything is installed and connected correctly, you should be able to reset the NUON development system using the command line:

```
        mload -!
```

The screen should blink, change to a color static-like display for a moment, then show either a vertical color bar test pattern or the NUON logo. If this fails, continue through the troubleshooting information below. Otherwise, you can skip ahead to section 13.2.

If your machine does not respond, cycle the power and try again. If it still doesn't work, then try this:

```
        ping <ip address>
```

where <ip address> is the TCP/IP address that has been assigned to your NUON development system. If the network connection is being located correctly, you'll get a message like this:

```
        C:\vmlabs>ping 192.1.6.222

        Pinging 192.1.6.222 with 32 bytes of data:

        Reply from 192.1.6.222: bytes=32 time=6ms TTL=60
        Reply from 192.1.6.222: bytes=32 time=2ms TTL=60
        Reply from 192.1.6.222: bytes=32 time=5ms TTL=60
        Reply from 192.1.6.222: bytes=32 time=4ms TTL=60
```

Note that the numbers returned may be different from machine to machine.

If you're able to "ping" the machine, then you should double check that the MD_PORT environment variable is set correctly as specified in section 9.3.

If your machine does not respond, you'll get a message like this:

```
C:\vmlabs>ping 192.1.6.226

Pinging 192.1.6.226 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

This most likely indicates that the network configuration is not correct. In rare cases, it may indicate a hardware problem.

Make sure that everything is connected properly and configured correctly, as shown in the ***Hitchhiker's Guide To NUON*** document. If all else fails, contact VM Labs Developer Support.

13.2 Your First Sample

This section will walk you through the process of compiling and running a simple NUON program written in C.

13.2.1 Compiling A Sample Program

1) Change to the \VMLABS\SAMPLE\HELLO-WORLD directory. This folder contains a very simple "hello world" sort of sample program.

2) Execute the GMAKE utility. This utility will execute the C compiler to compile the source code according to the rules specified in the MAKEFILE. This should result in output that looks similar to this:

```
C:\vmlabs\sample\hello-world>gmake

mgcc -O3 -g -mreopt  -Wall -mrom -mpe3 -c hello.c -o hello.o

mgcc -mrom -mpe3 -o hello.cof hello.o -lmutil
```

   The line breaks may be formatted differently on screen.

Let's go over the output from step 2 and get into the details of what's being done.

The tool being called by GMAKE is the MGCC program, which is a driver program for the GCC C/C++ compiler. It provides a single interface to the C compiler, assembler, and linker. It's essentially equivalent to the basic "cc" program included with most command-line-oriented C/C++ compilers.

Here MGCC is being called twice in a row by GMAKE. Let's look at the first call and describe each of the options specified.

The command-line argument “-O3” specifies that optimization level 3 should be used.

The “-g” option specifies that source-level debugging information should be included in the final output file.

The “-mreopt” option specifies that the assembler should use a special optimization process.

The “-Wall” option tells the compiler should display all warnings.

The “-mrom” and “-mpe3” options tell the compiler that the program is intended to run on a ROM based system with MPE 3 as the main processor.[8]

The part that reads “-c hello.c” specifies that the compiler should compile the source file hello.c and then exit, without calling the linker.

The last part of the line, “-o hello.o” specifies that the name of the output file being created should be HELLO.O.

Now let’s look at the second call to the MGCC program again. This time, a different set of arguments is used.

Once again, the “-mrom” and “-mpe3” options tell the compiler and linker that the program is intended to run on a ROM based system with MPE 3 as the main processor.

The “-o hello.cof” argument specifies that the name of the output file should be HELLO.COF. This is followed by the name of the HELLO.O file, which will be passed to the linker to build an executable program file.

The “-lmutil” at the end of the line specifies that the linker should include the library file LIBMUTIL.A in the link. This file is assumed to be located in the “LIB” directory that is part of the NUON SDK.

13.2.2 Running The Sample Program

1) Before you execute the sample, make sure the machine has been reset. You may cycle the power, or enter the command:

```
mload -!
```

This command will reset the machine and ensure that it is ready to download new code.

[8] These compiler options are now the default mode of operation. Therefore, it’s possible that a revised version of the sample program code may remove them.

2) Now you can download the sample we just compiled by using the command:

```
mload –p3 –n hello.cof -r
```

The "-p3" portion of the command-line specifies that MPE 3 is the target for the options that follow. The "-n" option tells MLOAD that the ROM BIOS is already running, so that it takes appropriate care during the program download. The "hello.cof" file is the name of the executable that was created earlier. Finally, the "-r" option specifies that it should start running the code when the download is complete.

3) You should now see the output of the sample program on the TV/monitor screen connected to your NUON development system.

If the MLOAD command displays any error message in regards to downloading the program, try turning the NUON system off for a moment, and then try again starting at step 2.

You'll note that the MLOAD program is used quite a bit. In fact, it is your main interface to the NUON system in most cases.

All of the NUON samples are also configured so that the MAKEFILE contains the commands needed to build and execute the program in one step. Simply change to the desired sample directory and issue the command:

```
gmake load
```

This will invoke the GMAKE tool to build the program and then load it.

13.3 Additional Samples

Now you're ready to run more sample programs from the NUON SDK. Please note that there are a wide variety of samples covering different topics.

Many samples are intended to show a single specific idea, while others show off a variety of techniques.

14. NUON File Server

The NUON File Server allows programs running on NUON to access the console device and file system of the host PC.

The file server is built into the MLOAD tool.

The file server is a development tool only and is not available on consumer machines. Therefore, the standard C & C++ library functions for file I/O may not be used in your final product.

14.1 File Server Modes

There are two basic modes of operation for the file server. The first mode has the file server pretending to be a native file system as referenced through the C/C++ standard library functions. The second mode has the file server pretending to be a DVD disc or similar media, responding to NUON's media access BIOS functions.

We'll discuss each mode separately below.

14.1.1 File System Server

Adding the "-fs" option to the end of the MLOAD command line will cause it to go into file system server mode and begin serving files after everything else has been done.

In file system server mode, MLOAD listens for commands coming from the client (the application running on your NUON development system), and then echoes those commands to the PC file system, acting as a conduit for the data that is being transferred between the PC file system and the NUON application.

14.1.2 Media Server

Adding the "-me" option to the end of the MLOAD command line will cause it go into media server mode and begin responding to media access requests coming from the NUON system.

In media server mode, MLOAD listens for data requests coming from the client, and transfers the appropriate portions of the specified data file.

14.2 Debugging & The File Server

We have considered the possibility of ultimately incorporating the file server functions into the PUFFIN debugger. In the meantime, you can run PUFFIN concurrently with MLOAD. You should use MLOAD only as the file server, and use the debugger for loading and executing your program.

14.3 Client Side — File System Server

The client software needed to access the file server is built into the NUON version of the standard C/C++ library. When your NUON program calls any function that performs file I/O, the library's device driver sends a message to the ACE360 or PPC860 interface board. This message is then sent from the interface board to the host PC, where it is received by the file server. Then the required data transfer operation takes place between the NUON system and the host PC.

14.3.1 Examples

When you call the ***printf()*** function, the resulting output is sent to the **stdout** stream, which is normally opened to the character file attached to the console device. The text to be printed is sent to the host PC, where the file server sends it to the console.

When you call ***scanf()***, the NUON application sends commands to the host PC asking for data from the **stdin** console device, which the file server sends back to the NUON as it is typed.

Aside from the console device, your NUON application can also read, write, and create files on the host PC. For example, the program below would appear to function identically when run on NUON with the file server and when as a native program on the host PC:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
FILE *pcfile;
char linebuf[300];

        pcfile = fopen( "fstest.c", "r" );

        if( pcfile )
        {
                // read a string from PC
                fgets( linebuf, 299, pcfile );
                // Make sure we're null-terminated
                linebuf[299] = 0;

                // print string back to PC
                printf( "%s", linebuf );
        }
        fclose(pcfile);
}
```

This example opens the file named “FSTEST.C”, attempts to read 299 characters from the file, prints them to the console device, and then closes the file.

14.3.2 Current Directory

From the NUON’s perspective, all file I/O functions are performed relative to the directory from which MLOAD was executed. From the user’s perspective, it’s as though a native PC program were running.

14.3.3 Available Functions

As of this writing, the following low-level file I/O functions are available through the file server.

open()	close()	read()
write()	isatty()	lseek()
stat()	fstat()	ioctl()
link()	unlink()	

Many of the standard C library’s I/O functions are not listed here, such as ***printf()*** or ***fopen()***, because they do not actually perform any I/O tasks directly. Instead, they call the low-level functions listed above, and therefore take full advantage of the file server.

Additional functions may be added to a later version of the file server if a strong need arises.

14.4 Client Side — Media Server

For media access, using the file server is a bit different than using the standard C/C++ library functions. The first important difference is that the NUON application has the choice of accessing either the file server or real media.

The NUON’s media access functions support the notion of a “BOOT device” which refers to the device and media from which the application was executed. In most cases, an application will simply read data through this device and not worry about what type of device it is. In fact, this is the recommended mode of operation as it makes it easier for a program to be delivered using a variety of media.

When your application is started via the MLOAD program from your host PC, then the “boot device” is the network connection to the file server running on your host PC.   When your application is started from DVD, then the boot device is the DVD.

During the development process, it can be useful to access a device directly, rather than using the “BOOT device”. An application in the development stages may wish to access some data from the host PC through the file server, and other data from a disc in the DVD drive. For example, movie sequences or streaming audio data could be taken from the DVD, while level data or code overlays are taken from the file server on your host PC.

14.4.1 More Information To Come

More information on the media server mode of the file server will be added once this mode of operation is fully functional.

14.5 Using The File Server Intelligently

The file server is one of the most powerful and useful tools that a NUON developer can use in creating their application.

It’s also one of the easiest ways to mess up, if you’re not careful.

Right now, the file system server mode is very useful as debugging tool, and also as a temporary means of data access . However, this is where the danger lies, so please read all of this chapter carefully in order to avoid some of the common pitfalls.

14.5.1 Using The File Server For Debugging

The file server is ideally used as a debugging tool, using a wide variety of techniques. Here are just a few suggestions:

- The ***printf()*** function can be used to display warnings or ongoing status information from the NUON application.

- Your program can use ***scanf()*** to read input from the host PC’s console and then use that input to control the program’s actions.

- Your NUON application can create log files that can be used to save important details regarding the program’s execution. For example, you might dump a list of which objects in your 3D world were rendered for a given frame.

- Profile performance by writing out the system clock values before and after a function call. (Make sure you don't include the ***printf***() function or file I/O in the part that gets timed.)
- Your program can perform hex data dumps from your memory buffers to the PC console or to a file.
- Your program can use ANSI terminal escape codes to format custom debugging menus or interfaces on the host PC.

14.5.2 Using The File Server For Media Access

The file server can be used for media access, and because it is called via standard C library functions, it would be very easy and very convenient to use those functions wherever needed throughout your application to load all of your data files at runtime.

Please don't do it that way!

First of all, the file server is strictly for development purposes only. On a consumer machine, there will be no file server and therefore the standard C library functions will not be available.

This doesn't come as a big surprise to most developers. But it would be easy to use those standard C library functions anyway, expecting that there will be a fairly close correlation between them and the system-specific file I/O functions, and therefore that it will not be difficult to change everything as needed later.

That would be a mistake. The media access functions in the NUON BIOS are actually quite a bit different from the standard C library functions. Your program's data reading should consider this from the very beginning of your project. Otherwise, if your program uses the standard C functions for all of its data reading, it will require a significant code rewrite to switch over to the NUON BIOS media access functions.

14.5.2.1 NUON BIOS Media Access –vs– Standard C Library I/O

The NUON BIOS includes a number of functions for media access which have been specifically designed for optimal performance on an embedded system where most media access is reading from read-only media and where the end user will never have to deal with manually saving or retrieving files. These functions are optimized for performance and simplicity.

The file I/O functions in the standard C library, on the other hand, are designed primarily for accessing read/write media on general-purpose computer systems.

The library has been designed around the idea that the user may arbitrarily access any particular file at any time, create or delete files, and otherwise manipulate the contents of the media. The library must allow access to a relatively much more complex file system. Furthermore, because of the way the standard C library evolved into a standard, there is quite a bit of overlapping functionality between different functions.

The bottom line is that the standard C library file I/O functions are overcomplicated and inefficient with regards to the needs of a NUON application.

If you haven't read the NUON BIOS documentation yet, specifically the section covering the media access functions, you should do so now. These warnings regarding the usage of the standard C library file I/O functions will then make a lot more sense.

14.5.2.2 Doing It Right

The ideal means of using the file server for media access is to create functions that are as close as possible to the NUON BIOS media access functions. In particular, these functions would accept the same arguments but have different names.

For example, instead of calling ***MediaRead()***, your application would call ***My_MediaRead()***. This function would be implemented as part of your application and would call the standard C library functions to access the appropriate data file on the host PC.

VM Labs intends to offer some form of media emulation that can be accessed using the BIOS media access functions via an emulation device type.

15. NUON Programming Guidelines

Even though it comes towards the end of this document, this chapter is one of the most important. This is where we'll give you all of the important little rules that must be followed in order to ensure that your program runs smoothly.

This chapter is divided into a number of subsections that cover different areas such as memory usage, DMA operations, media access, and so forth.

15.1 Memory Usage

15.1.1 Low BIOS Memory Area

Unless otherwise specifically instructed by a BIOS or system library function, your program should not access system memory locations from 0x80000000 to 0x8000FFFF. This area is reserved for use by the BIOS and system firmware.

Accessing this range may corrupt data or code necessary for proper operation of the hardware.

15.1.2 High BIOS Memory Area

An application should not access system memory locations ranging from 0x80760000 to 0x807FFFFF. This high memory area is reserved for use by the BIOS.

Accessing this range may corrupt data or code necessary for proper operation of the hardware.

15.1.3 Presentation Engine Memory Area

If a program uses the DVD Video Presentation Engine (aka "PE") for playing full-screen video, then the range of 0x804A0000 to 0x8075FFFF must be avoided whenever the PE is active.

When the PE is inactive, this memory may be used for other purposes.

15.2 Runtime Memory Allocation

15.2.1 C & C++ Runtime Heap Initialization

The standard C/C++ runtime libraries recognize the concept of a “system heap” of memory that is used for runtime memory allocation using functions like ***malloc()*** and the C++ **new** operator. The heap is usually considered to be whatever portion of memory is not used by the operating system or for loading your application.

On a NUON system, there are two types of memory (excluding the built-in memory): SYSRAM and SDRAM. SYSRAM memory lives on the “Other Bus” and is where your program code normally resides. SDRAM memory lives on the “Main Bus” and is used for your video frame buffer(s) and data storage.[9]

SYSRAM may be accessed via explicit DMA or through the cache mechanism. SDRAM is normally accessed only via explicit DMA. For this reason, the heap is normally positioned in SYSRAM.

Since the “heap” is actually managed by the runtime library that is linked into your program’s executable file, there must be a mechanism to allow it to determine the size and location of the heap at runtime. This is done by having the runtime library define a special program segment named “heap” where it reserves 16 bytes of space.

At runtime, when the memory allocator is initialized, it uses the starting address of the “heap” segment as the start of the system heap. Then it looks for the size of the “heap” segment. If it’s 16 bytes, it expands the heap size to extend all the way to the bottom of the stack area.

In order to use this automatic heap initialization process, it’s required that the “heap” segment will be the very last thing in memory before the program stack. However, as we’ll see in section 15.2.2, it is possible to do your own heap management.

15.2.2 Managing Your Own Heap

Sometimes it’s useful to manage your own heap initialization, rather than allowing the runtime library to do it automatically. You might want to force a certain maximum size, or you might want to avoid conflicts with other memory usage.

[9] For more information about the different types of memory and memory busses, please refer to the NUON processor’s ***Programmer’s Guide*** document.

Regardless of your motivation, initializing your own heap is pretty simple. All you really need to do is reserve space within a special program segment named “heap”. The short assembly language file shown below would do just that:

```
        .segment “heap”

        .ds.b   0x00100000
```

This would reserve 1 million bytes of storage in the “heap” section. The linker would add this to the 16 bytes that it already reserves.

At runtime, the memory allocator initialization code would see that the “heap” segment is greater than 16 bytes, and then it would use the existing location and size without any changes.

15.2.3 Avoiding Conflicts Between the Heap And Other Memory Usage

If you use the linker’s ability to position a particular section of your program at a specific memory address in SYSRAM, then you run into the possibility that there will be a conflict between the memory load map of your COFF file and the C/C++ runtime library’s notion of where the system heap should be located.

Please make sure that you have read section 15.2.1 for an overview of how the heap is allocated.

The problem occurs because when you tell the linker to explicitly position a segment, it doesn’t change the position of any other segments. Each segment is normally positioned immediately after the end of the previous one, except for any alignment padding. So what usually happens is that the section you positioned is right in the middle of memory between the start of the heap and the bottom of the stack. When the heap gets resized, the runtime library has no way to know that this other program segment is already taking up some of the memory in question.

As your program does memory allocation, the memory blocks being handed out will eventually overlap whatever was in the other program segment. When that happens, one item or the other will be corrupted and the program most likely crashes.

Fixing the problem is quite simple, and there’s a choice of at least two methods you can use. First, you can set up your own heap so that the runtime library won’t resize it. With this method, you’ll at least get a warning from the linker if the program sections are going to overlap. See section 15.2.2 for more information.

Alternately, simply tell the linker to explicitly position the location of the “heap” section so that it comes after any other sections that you are explicitly positioning. Keep in mind that this may reduce the total available heap space.

15.2.3.1 Using the COFFDUMP tool to help avoid problems

The COFFDUMP tool can be very helpful in determining if there are any conflicts. If you do:

```
coffdump –h program.cof
```

You will get a list of all the program sections. If the “heap” section is 16-bytes, you need to make sure that no other section starts at a higher address in SYSRAM. If one does, then you’ll have to change something and try again.

15.3 DMA Operations

15.3.1 DMA Transfer Size

Even though the hardware allows larger transfers, it is considered very bad practice to transfer more than 64 long words (256 bytes) of data in a single Main Bus DMA operation. This includes the total of a series of DMA operations where the chain transfer mode is used. It does not apply to batch mode transfers.

Under no circumstances should more than 64 long words be transferred in a single Other Bus DMA operation. When the bus is especially busy, Other Bus DMA transfers should be no more than 32 long words (128 bytes).

The reason for the limitation is that longer transfers will monopolize the bus long enough that real time interrupt-driven processes may not function properly due to timing and latency issues. This includes operations such as media access, sound playback and synthesizer operation.

15.3.2 Issuing DMA Commands

If you’re going to perform DMA operations from the primary processor while the cache is turned on, it is very strongly recommended that you use the BIOS functions ***_DMALinear()*** and ***_DMABiLinear()*** rather than directly accessing the DMA control registers.

The reason for this recommendation is that there are a variety of potential conflicts between the cache mechanism and other DMA usage. The BIOS DMA functions are aware of the various issues involved and know how to avoid problems.

However, please be aware that when running code on a non-cached processor, you will not be able to use the BIOS DMA functions. In this case, you will have to write your own code that accesses the DMA control registers as needed.

15.4 Timers & Interrupts

15.4.1 Vertical Blank Interrupt

The NUON has the ability to tie an interrupt process to the vertical blank routine. This is generally very useful for game programs which must synchronize certain operations with the display.

However, we *very strongly* recommend that any code which directly affects the speed of your game should not use a vertical blank interrupt or counter as a timer.

Different systems may not always synchronize at the same rate. For example, NTSC systems have 60 vertical blank periods per second while PAL systems have 50. Systems with progressive scan output or HDTV output may handle the vertical blank differently. For this reason, if you use the vertical blank counter as a timer, your game may run at drastically different speeds on different systems, or may not work properly at all on some systems.

It's perfectly OK to use vertical blank synchronization to change frame buffers, or do other operations which are tied to the screen display. However, the vertical blank counter should not be used as a timer for things like moving a sprite a certain distance, or deciding which frame of sprite animation to show. Use the system timer instead, as described in section 15.4.2.

15.4.2 Using the System Timer

The NUON system has a built-in timer which should be used to time operations whenever possible.

For example, if your game wants to have a certain sprite move across the screen at a certain speed, then it should use the timer to determine how much time has elapsed since the previous frame was processed. That way, it can determine what the new position of the sprite should be.

If the game assumed that it should move the sprite a fixed amount for each new frame, then the movement would be jerky and inconsistent if the game did not run at the same frame rate all of the time. It would also run at different speeds on PAL systems versus NTSC systems. However, using the timer to determine how far to move will work properly under all circumstances.

To use the timer, simply call the BIOS function ***_InitTimer()*** at the beginning of your program to initialize it. Then just call ***_TimeElapsed()*** at any time to retrieve a millisecond timer count:

```
    ms = _TimeElapsed(0,0);
```

It’s possible to get finer resolution, as detailed in the BIOS documentation. However, for most purposes in a game, the millisecond counter is more than sufficient. At 60 frames per second, it’s about 33 milliseconds between frames. Furthermore, it’s easier to use the millisecond counter in most cases since it’s not broken into two different values.

15.5 MLOAD

15.5.1.1 Machine Reset via MLOAD

When you use the “-!” command line option to reset your NUON development system, always make sure that there is a reasonable delay period before you attempt to download and execute a program. The delay required may vary depending on the firmware currently installed on your system, but generally about 3-4 seconds is sufficient.

A four second delay may be added after the reset by adding “-w –w –w –w” to your mload command line immediately following the “-!” option. For example:

```
      mload -! –w –w –w –w –p3 –n program.cof -r
```

15.6 Media Access

This section under construction.

15.7 MPE Usage

This section under construction.

15.8 More

This section under construction.

16. FAQ For New Developers

This chapter contains a variety of frequently asked questions on various topics, along with the appropriate answers.

Each question is only listed once, so if you don't find what you're looking for under one topic, make sure to check under related topics.

This chapter will be updated regularly. For more information relating to programming issues, please also see the separate document titled ***NUON Troubleshooting Guide***.

16.1 Communicating With NUON

Q: I have connected the NUON development system to my network, but I cannot communicate with it from my host PC.

A: Most likely, there is a problem with the IP address setting of your NUON development system. See chapter 6.2 for details. Otherwise, it's possible that the MD_PORT environment variable is not set properly. See section 9.4 for details.

Q: I have two separate NUON development systems connected to my network, but I can only communicate with one machine at a time. If both machines are connected and turned on together, I get communications errors from MLOAD. If I power down or disconnect one machine, it works OK.

A: Both NUON systems may be set to the same IP address. Otherwise, both systems may have the same MAC address setting. See chapter 6.2 for details.

Q: I made a crossover network cable to connect my PC directly to the NUON development kit, but they're not talking.

A: Making your own network cables, especially a crossover cable, is not recommended. The connections for a crossover cable are easy to get wrong, since multiple connections must be switched. Also, a homemade cable is more likely to provide significant amounts of electrical noise that will cause network packet errors, slowing down transfers and communication.

16.2 Connections

Q: What's the miscellaneous video connector for? I don't seem to have a cable that will fit into it!

A: This connector allows you to create your own cable for monitors with non-standard connections. It uses a standard DB-9 connector because they are readily available.

Please note that although this connector may appear to be the same type, it is NOT directly plug-in compatible with the old-style NEC Multisync computer monitor. An adapter cable can be created for this monitor, however, provided it is one of those which can display NTSC video.

16.3 Tools

Q: Do the NUON SDK tools work under Windows NT?

A: Yes.

Q: Does the NUON SDK include C++?

A: Most of the libraries provided by VM Labs are C-oriented, but C++ is fully supported by the compiler for writing your own code.

There was a bug with older releases of the C++ compiler that prevented the “virtual” keyword from working properly. This was been fixed in subsequent revisions.

Q: The C compiler is giving me a message saying “Virtual memory exhausted”. What do I do?

A: The compiler likes lots and lots of memory. Your host computer should have at least 32mb of RAM, and 64mb won’t hurt.

Secondly, if you’re running an MSDOS Command prompt under Windows 95/98, then locate the command prompt icon, right-click it, and select “Properties.”

Under “memory”, change the “DPMI” setting from “auto” to “65535”. This will configure things so that the maximum possible amount of memory may be used by tools running under the command prompt.

16.4 Libraries

Q: Do the runtime libraries use multiple processors?

A: Yes they do. Most of the supplied libraries have the ability to use multiple processors for various types of 2D or 3D graphics rendering. However, processor usage is always controllable by the programmer.

16.5 Video

Q: Can the NUON development system be switched into PAL video mode?

A: Not currently. A future revision of the boot ROM will allow you to configure the video display to choose between NTSC and PAL video output.

Q: Can I use video input?

A: Video input is possible with the addition of a video encoder board to your NUON development system. These boards are not yet available to developers. An announcement will be made when something becomes available.

16.6 DVD Reading

Q: Are development units with DVD drives available yet?

A: Yes. DVD drives are included in all development systems currently shipping, and this has been the case for quite awhile now. If you have an older system without a DVD drive, contact your account manager at VM Labs regarding a replacement.

Q: Can I connect a DVD-ROM drive to my NUON development kit?

A: No. The electronic interface for a DVD-ROM drive is designed to connect to a computer. The NUON development system expects a bare mechanism, and customized firmware is required for each different type.

16.7 Inter-Processor Communication

Q: Can a process running C code in one MPE start a process in another MPE?

A: Absolutely. This can be done using the ***StartMPE()*** function from the LIBMUTIL library.

Q: Can a process running C code in one MPE read or write data to the memory space of another MPE?

A: For regular internal memory, this can be done using the ***_mpedma()*** function. For the register space of another processor, use the ***_mpedmaregister()*** function. Both functions are from the LIBMUTIL library.

16.8 Memory Cards

Q: How much data can be stored on a memory card?

A: At this time, memory cards are expected to have a minimum of 2 megabytes of storage space. However, this is subject to change.

17. Glossary

Aries — Code name for the current version (MMP-L3B) of the NUON processor, the version that will be used in end-user consumer systems.

GCC — The Gnu C Compiler created by the GNU project of the Free Software Foundation. This compiler is available for most microprocessors and was adapted by VM Labs for the NUON processor.

Llama — The NUON assembler.

Merlin — This is the original internal name of the VM Labs custom processor, and in a broader sense the name of the development system hardware. The final public name is ***NUON***.

MPE — An acronym for “multi-processing element”. It refers to either one of the four separate modules of the *NUON* processor, or to a software program intended to run in a single module.

MPO — NUON Processor Object module. An object module output by the LLAMA assembler.

NUON — This is the name of the VM Labs custom processor, and in a broader sense the name of the development system hardware.

Oz — Code name for the original hardware release (MMP-L3A) of the NUON processor used in development systems only, not end-user consumer systems.

Puffin — This is the name of the debugger/emulator from the NUON development system.

Puffin TK — This is the name of the version of the Puffin debugger designed to run under the Tcl/TK graphics toolkit & scripting language.

Puffinw — Version of Puffin designed to run under Microsoft Windows.

This page intentionally left blank.