VM LABS

The Hitchhiker's Guide to

An Overview of the NUON Open Platform Development Tools

Revision 0.90
29-Oct-01

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

Table of Contents

1. Introduction

Welcome to the NUON Open Platform development system. This document will serve as a basic introduction to the system. It will attempt to answer the basic questions you may have when you first get started.

> ***October 16, 2001 — Please note that since the NUON Open Platform is brand new, there are still a lot of things that are subject to change and improvement.***
>
> ***In particular, the first several chapters of this document will probably be revised fairly regularily for at least the first few months of the NUON Open Platform.***
>
> ***If you find anything that is especially confusing or just plain wrong, please refer to the NUON Open Platform website and discussion forums for updates and corrections.***

1.1 NUON "Open Platform" -vs- Commercial Development

The first series of questions that need to be addressed are:

- What's the difference between a NUON Open Platform developer and a commercial developer?
- What special advantages are there to being a commercial developer?

The rest of this section will attempt to answer these questions.

1.1.1 What is a "Commercial Developer" Anyway?

A commercial developer, quite simply, is someone who intends to create a product and bring it to market in order to make a profit. For a commercial developer, money really is the ultimate consideration in any decision. If they don't think there is a profit to be made, they won't bother trying. There is no consideration of "cool", or "this system is better than that one", or anything else, unless and until it's been decided that there is a profit to be made.

A commercial developer is also willing and theoretically able to spend the money required to create a product that can compete in the marketplace. For a video game, this translates to hiring programmers, producers, artists, musicians, and so forth. A production budget of $500,000 is fairly modest these days, and some games are many times more expensive to create.

When you're spending that kind of money to create a game, it's important to make sure that everybody involved has the equipment necessary to work efficiently. Otherwise, it's probably going to cost you money in man-hours in the long run when you skimp on equipment. For this reason, commercial developers usually have fairly recent, up to date computers and other equipment.

When it comes to developing software for a system such as NUON, or game consoles such as PlayStation, GameCube, or XBox, the category of "other equipment" includes whatever customized development system hardware is available. Such hardware is customized for development purposes and it is considerably more expensive than the consumer models found in stores.

By "considerably more expensive", we mean anywhere from 30 to 100 times more expensive than the corresponding consumer hardware. Such hardware is not mass-produced, and is generally somewhere between $5000 and $30000 per unit. Plus, it often requires a dedicated computer system that is not included in the base price.

For NUON development, VM Labs created a special version of the NUON Reference Design DVD player that is often referred to as the "NUON Development System" or "devkit". This machine has additional memory and special debugging interfaces that aren't available on a standard production NUON DVD player.

The bottom line is, if you're the sort that would spend $5000 to $30000 or more on custom hardware in order to develop software for a single proprietary platform, then you're either a commercial developer or your allowance is way, way too big.

1.1.2 What's The NUON Open Platform?

The NUON Open Platform system is based around the idea of allowing hobbyist programmers to develop for the NUON system, without requiring them to spend large amounts of money on special hardware as described earlier. The NUON Development System is on the low-end of the cost scale for such hardware, but it's still a lot of money for a hobbyist.

For the most part, the programming tools and libraries available to the NUON Open Platform developer are the same exact ones that are made available to commercial developers. The main differences are:

- Software that requires the custom NUON Development System hardware is not included in the NUON Open Platform distribution.
- Software or programming libraries that are not properly licensed for open distribution are not included in the NUON Open Platform distribution.[1]

That's it. Everything else is pretty much the same, as regards tools and libraries.

The last main difference between the NUON Open Platform and commercial development is the support offered by VM Labs. This will be discussed in a later chapter.

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

Chapter 1 will introduce you to some details about support.

Chapters 3 through 5 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 7, we demonstrate how to build and execute a sample program, and help you with trouble-shooting.

Chapter 8 has a number of programming guidelines which should be observed.

Chapter 9 contains Frequently Asked Questions (FAQ) compiled from developer inquiries on a broad range of subjects ranging from setup issues to programming.

Chapter 10 provides a basic glossary of terms that you'll see used often throughout the NUON developer documentation.

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, or places where a change wasn't really practical. 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".

[1] At the time of this writing, I can't actually think of anything that fits into this category, but I'm hedging my bets in case we do run across something before the NUON Open Platform goes public.

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 NUON 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. This chip revision was never used in any consumer hardware.

Aries is the name used to refer to the MMP-L3B revision of the NUON chip. This version of the chip was used in the development system as well as in production consumer 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 will begin appearing in consumer machines in late 2001 or early 2002. This revision features additional cost reduction changes. This revision will also offer performance increases for new applications. (Most older applications will run with Aries 2-level performance.)

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

Note that it is not possible to upgrade an existing system to a newer revision of the chip.

2. NUON Open Platform Support

2.1 Support For Open Platform Developers

VM Labs does not offer direct support via telephone or EMAIL to users of the NUON Open Platform development tools. All support is through the website and discussion forum. Engineers from VM Labs may participate in the discussion forum as time permits.

The primary reason for not providing direct support is simply that it's not economically practical. For commercial developers, the cost of support is factored into the cost of the customized development hardware and the royalties paid for each unit of software that is sold. However, VM Labs does not derive any income directly from the NUON Open Platform developers. This means it's simply not practical to provide dedicated support personnel.

3. 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.

3.1 System Requirements

The system requirements for the development tools are as follows:

- 133Mhz or faster Pentium-based PC
- 32mb RAM minimum… but frankly we haven't really gone out of our way to test everything on such a system. Considering how cheap RAM is these days, you should probably just max out your motherboard and be done with it.
- 90mb 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)
- Windows 95/98/ME, Windows NT 4.0, Windows 2000, Windows XP (Home or Professional)
- CD-R Burner

These requirements are subject to change as new revisions of the SDK are made available. Please note that the operating system being used may have additional system requirements beyond those listed here.

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

3.1.1 Which Version Of Windows?

The "officially supported" operating system for the NUON SDK tools is Windows 2000. Most of the tools should work fine on any 32-bit version of Windows since Windows 95, but Windows 2000 is the only version that we formally test against.

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.

In particular, there are differences with the command shell (also known as the MSDOS prompt) from one version to another.

3.1.1.1 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.

3.1.1.2 Windows 98

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

3.1.1.3 Windows ME

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

3.1.1.4 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.

3.1.1.5 Windows 2000

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

3.1.1.6 Windows XP

The NUON SDK tools have been tested with the preview edition of Windows XP and there are no known problems. Note that it doesn't make any difference if you're using Windows XP Home Version or Windows XP Professional.

3.1.2 What about Linux?

At this time, many, but not all, of the NUON SDK tools are available for Linux (Intel only). Linux is not an officially supported platform.

3.1.3 What about Macintosh?

At this time, none of the NUON SDK tools are available for Macintosh.

3.2 Tools Installation

Information on obtaining the NUON SDK is available on the website.

3.3 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.

3.3.1 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.

3.3.2 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.

3.3.2.1 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.

3.3.3 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.

3.4 MS Windows & Environment Variables

3.4.1 Setting Environment Variables Under Windows 2000 & Windows XP

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.

3.4.2 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 VMLABS=C:\VMLABS
```

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

3.4.3 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 line shell, environment variables can be changed by using the "set" command. However, such changes only affect that particular instance of the command shell and those programs that are executed from it.

- The command line shell's "start" command (which is used to start a program and then return control to the user without waiting for it to finish) is really a message to Windows asking it to start the specified program. Therefore, the program will only inherit the global environment. Any changes to the environment made within the command line shell will not be recognized.

3.5 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.

4. 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.

4.1.1 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
COLLECT2.EXE	Prelinker used for C++ Template processing.
CPP.EXE *or* VMCPP.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.

4.1.2 Assembler

The assembler is known as Llama.

Program Name	Description
LLAMA.EXE	NUON assembler

4.1.3 Linker

The linker is VMLD.

Program Name	Description
VMLD.EXE	NUON Linker

4.1.4 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.

Program Name	Description
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.

4.1.5 Miscellaneous Tools

The following tools are used for a variety of purposes.

Program Name	Description
GMAKE.EXE	Program builder utility

4.1.6 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 **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.

4.1.7 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.

4.1.8 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.

4.1.9 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.

4.1.10 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

4.1.11 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.

4.1.12 Photoshop Plug-Ins

The "**VMLABS\Photoshop Plug-Ins"** directory (if present) 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.

5. 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.

5.1 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.
-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.
-frepo	This option should be used when compiling C++ code that uses templates.
-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.
-mnopatch	This option tells the compiler to link against the startup code module named **crt0p.o** instead of the usual **crt0.o**. This alternate file does not include system patches, and is intended for use with code overlays and other secondary executables called from your main executable (which would normally include the patches).
-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
-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.

5.1.1 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.

5.1.2 NUON C/C++ Compiler Notes

This section was last updated on September 11, 2001. Note that the compiler is now based on *gcc 2.95.3* **rather than the** *egcs* **variation.**

- The C++ interfaces have changed. C++ code and libraries should be recompiled.
- The C++ compiler is now more ANSI compliant and also stricter about enforcement. Some code which generated warnings with the previous versions will now generate errors.
- Code generation for exception handling has changed. Any C++ code using exceptions should be recompiled. Code using exceptions should not use the **–fomit-frame-pointer** option.
- "MERLIN" is no longer pre-defined by the preprocessor. Use "NUON" instead.
- The compiler now uses **collect2** for the link stage. This program is required when using exceptions or when using templates compiled with the –frepo option. The **collect2** program will call **vmld** in turn as required.
- 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.
- The new **–malignfuncs** option forces functions to be aligned on 64-byte boundaries, which is the default instruction cache line size.
- The compiler now combines shifts, additions, and logical operations (AND, OR, XOR, etc.) into one instruction where possible.
- 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.

5.1.2.1 Command Line Options

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

- 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.

5.1.2.2 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*.

5.1.2.3 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.

5.1.2.4 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.

5.1.3 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.

5.2 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.

5.2.1 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.

5.2.2 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.

5.3 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.
-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.

5.3.1 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.

5.4 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* --directory = *directory*	**or**	Change to the specified *directory* before doing anything
-d -debug	**or**	Debug mode. Print out MAKEFILE debugging information. Prints commands being executed, and lots of other stuff.
-e --environment-overrides	**or**	System environment variables override variables with same name which are defined by GMAKE and within MAKEFILE. (Default is vice-versa.)
-f *file* --file=*file* --makefile=*file*	**or** **or**	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 -help	**or**	Print out command line options
-I *DIRECTORY* --include-dir=*DIRECTORY*	**or**	Search *DIRECTORY* for included makefiles.
-I --ignore-errors	**or**	Ignore errors. Continue with MAKE process even when commands return error.
-j *num* -jobs *num*	**or**	Allow a maximum of *num* jobs at once. Default is unlimited.
-k --keep-going	**or**	Continue with MAKE process even if some targets cannot be made.
-l *load* --load-average *load*	**or**	Specify maximum load allowed.
-n --just-print --dry-run --recon	**or** **or** **or**	Print command lines, but do not execute commands.
--no-print-directory		Turn off "-w" family of options, even if turned on implicitly.
-o *file* --old-file=*file* --assume-old=*file*	**or** **or**	Assume that *file* is very, very old, and do not remake it.
-p --print-data-base	**or**	Print GMAKE's internal predefined rules database which specify how to build target files from source files.
-q --question	**or**	Do not build target, simply return exit code that specifies if target is up to date or not.
-r --no-builtin-rules	**or**	Disable GMAKE's internal predefined rules which specify how to build target files from source files.
-s --silent --quiet	**or** **or**	Do not echo commands as they are executed.
-S --no-keep-going --stop	**or** **or**	Cancel "-k" family of options. Stop when target cannot be built.
-t --touch	**or**	Touch targets (update time/date stamp) instead of building them.
-v --version	**or**	Display GMAKE's internal version number
-w --print-directory	**or**	Print the current directory
-W file --what-if=*file* --new-file=*file* --assume-new=*file*	**or** **or** **or**	Consider file to always be new (always build target)
--warn-undefined-variables		Warn when undefined variables are referenced.

5.4.1 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.

5.5 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`
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`

Option	Description
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.
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 tv 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.
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.

5.6 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.

5.6.1 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  |            |      |     |
```

5.6.2 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.

5.6.3 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.

5.6.4 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.

5.6.5 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.

5.6.6 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.

5.7 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.

5.7.1 COFFDUMP Output

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

5.7.1.1 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.

5.7.1.1.1 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.

5.7.1.2 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.

5.7.1.3 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.

5.8 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.
-r symbol	Removes the specified *symbol* even if it would be removed otherwise.

5.9 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
```

5.10 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.
-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.

5.10.1 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

5.11 COFFPACK Utility

The COFFPACK utility is designed to consolidate program sections in an executable COFF file. Adjacent program sections are combined together, with padding inserted as required. The result is written to a new COFF file.

The primary goal of the COFFPACK tool is to reduce the amount of time it takes to load a program.

The amount of time it takes to load a program from DVD is largely based on how many sections are contained within the COFF file. A program that uses a lot of code overlays may easily have 150-200 program sections.

When a COFF file is loaded, the BIOS must load and authenticate each section individually. Each section may target any arbitrary portion of memory, so it's not really possible to read the COFF file in big chunks. Reading the file in smaller chunks reduces the efficiency of reading data from the DVD, making for increased load times.

The COFFPACK tool will reduce the number of program sections down to the bare minimum. Some programs may go from 150-200 sections down to 1 or 2. The actual file size might actually increase in some cases, but combining sections allows the BIOS to read data in large chunks, which provides a dramatic improvement in loading time.

The format of the COFFPACK command line is:

```
coffpack [options] coff-file
```

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

Option	Description
-o filename	Specify the output filename. By default, if none is specified, the output file name is B.OUT in the current directory.
-p <num>	Set the maximum size for padding that can be added between adjacent sections in the source file. The default is 32, meaning that if two program sections are within 32 bytes of each other, they will be merged together in the output file.
-v	Verbose mode. This causes COFFPACK to print a detailed report of what it's doing as it works. This is mainly for troubleshooting if there is a problem with a particular COFF file.

5.12 RGB2YCRCB Utility

The RGB2YCRCB utility is designed to convert graphics image files using the RGB color space into a NUON-compatible image file that uses the YCrCb color space. The RGB2YCRCB tool supports a variety of input and output formats. All input files must use the RGB color space, but there are options for both RGB and YCrCb output.

The format of the command line is:

```
rgb2ycrcb [options] imagefile [[options] imagefile]…
```

The *imagefile* parameter specifies the source graphics image to be converted. Note that wildcards may be used, but may not provide the desired result if more than one file matches. Also be aware that the file format ***must*** be explicitly specified on the command line. The filename extension is not used as a clue to the file format.

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

Option	Description
-?	Print out command line option usage information.
-a0	Specify that there is no alpha channel in the source image. This option implies a 24-bit per pixel source image. This option is only allowed when the source format is RAW. This is the default option if nothing else is specified.
-a1	Specify that the alpha channel value for each pixel comes before the color value. This option implies a 32-bit per pixel source image. This option is only allowed when the source format is RAW.
-a2	Specify that the alpha channel value for each pixel comes after the color value. This option implies a 32-bit per pixel source image. This option is only allowed when the source format is RAW.
-bgr	Specify that the RGB color information for each pixel in the image file uses a byte order of Blue-Green-Red. This does not imply anything about the presence or absence of an alpha channel value. This option is only allowed when the source format is RAW.
-bmp	Specify that the source image is in BMP format.
-f_bmp	This specifies that the output format is a BMP file. The pixel depth will depend on the source file. For palette-based source images, a palette-based BMP file will be created. If the source image is not palette-based, a 24-bit BMP will be created.
-f_bmpa	This specifies that the output format is a BMP file with 32-bits per pixel, each containing an RGB color value and an alpha channel. Please note that the Windows BMP format does not officially support images with an alpha channel and more than 24-bits per pixel. The resulting files may not be recognized by all Windows-based programs.
-f_e655	This specifies a RAW output format with 16-bits per pixel in YCrCb format, corresponding to NUON pixel format 2.
-f_e655z	This specifies a RAW output format of 32-bits per pixel, each with a 16-bit YCrCb color value and a 16-bit Z value. This corresponds to NUON pixel format 5. Note that there is usually no need to store the Z value in a graphics file.
f_e888alpha	This specifies an output format of 32-bits per pixel in YCrCb format, corresponding to NUON pixel format 4.
-f_e888alphaz	This specifies a RAW output format with 64 bits per pixel, each with a 32-bit YCrCb pixel and a 32-bit Z value This corresponds to NUON pixel format 6. Note that there is usually no need to store the Z value in a graphics file.
-f_egrb655	This specifies a RAW output format with 16-bits per pixel, each containing a 16-bit RGB color value.
-f_ergbalpha1555	This specifies a RAW output format with 16-bits per pixel, each with 15-bits used for a RGB color value and 1 bit for an alpha-channel value.
-f_grb	This specifies a RAW output format with 24-bits per pixel, each containing an RGB color value. The byte ordering is Green-Red-Blue.
-f_grba	This specifies a RAW output format with 32-bits per pixel, each containing a 24-bit RGB color value and 8-bit alpha channel value. The byte ordering is Green-Red-Blue-Alpha.
-f_mbm	This specifies that the output format is an MBM file to be used with the M3DL graphics library. The output will use the RGB colorspace. The pixel depth will depend on the source file. For palette-based source images, a palette based MBM file will be created. If the source image is not palette-based, a 16-bit MBM file will be created.
-f_mbm_ycc	This specifies that the output format is an MBM file to be used with the M3DL graphics library. The output will use the YCrCb colorspace. The pixel depth will depend on the source file. For palette-based source images, a palette based MBM file will be created. If the source image is not palette-based, a 16-bit MBM file will be created.

Option	Description
-f_rgb	This specifies a RAW output format with 24-bits per pixel, each containing an RGB color value. The byte ordering is Red-Green-Blue.
-f_rgba	This specifies a RAW output format with 32-bits per pixel, each containing a 24-bit RGB color value and 8-bit alpha channel value. The byte ordering is Red-Green-Blue-Alpha.
-grb	Specify that the RAW image file uses a byte order of Green-Red-Blue. This does not imply anything about the presence or absence of an alpha channel value. This option is only allowed when the source format is RAW.
-o <filename>	Specify the output filename for the next file to be converted. Note that a single command line may include several source files. An output filename must be provided for each, or the last one provided will be used more than once. For example: rgb2ycrcb –bmp –o background.ycrcb background.bmp –tga –o sprite.ycrcb sprite.tga This specifies that the first file to be converted is ***background.bmp*** and that the new file ***background.ycrcb*** should be created. Then the source format is changed to TGA and ***sprite.tga*** is converted into ***sprite.ycrcb***
-raw	Specify that the source image is in RAW format. By default, the source image is assumed to be in RGB format. Palette-based images are not supported. This is the default source file format if none is specified.
-rgb	Specify that the RAW image file uses a byte order of Red-Green-Blue. This does not imply anything about the presence or absence of an alpha channel value. This option is only allowed when the source format is RAW. This is the default option if nothing else is specified.
-tga	Specify that the source image is in Targa (TGA) format. Only 16, 24, or 32-bit files are supported.
-tim	Specify that the source image is in TIM format. (TIM files may contain multiple images, but only single-image TIM files are supported.)
-v	Print out the RGB2YCRCB version number.
-vflip	This option will cause the destination image to be vertically inverted compared to the source image. This option is only valid when the source format is BMP or Targa (TGA), which are normally stored from bottom to top instead of top to bottom. The RGB2YCRCB tool is capable of recognizing an inverted image when the file header is set properly, but this may not be done correctly in all files. Therefore, this option allows the user to override the flag in the file header when necessary.

For more information on NUON pixel formats, see the ***NUON Multi-Media Architecture Programmer's Guide***.

Some of the output pixel formats correspond to formats used by the MML2D graphics library. See the MML2D documentation for more information.

5.13 CREATENUONCD

The CREATENUONCD utility is designed to package a NUON executable COFF program file into a NUON.CD file that can be used to create a bootable NUON CD-R disc.

The format of the command line is:

```
createnuoncd filename
```

There are no command line options. The name of the input file may be provided on the command line. If none is given, the default file it looks for is **CD_APP.COF.** The output file will be always named **NUON.CD**.

At this point, you may create a CD-R disc with the **NUON.CD** file in the root directory. If your CD-R mastering software offers a choice of disk format, make sure to select ISO9660. Other formats will not work. In particular, if you use a CD-RW disc, make sure that the disc does not use the UDF file system.

Once the disc has been created, it should boot in a Samsung N-501 or later NUON DVD player.[2]

5.14 NUON Data Tool

The ***NUON Data Tool*** is a software application that runs under Microsoft Windows. It may be used to combine a list of smaller data files together into a single large data file. It is included in the NUON SDK as **datatool.exe** in the **BIN** directory.

The NUON Data Tool accepts no command line options.

The program maintains a list of all the files that will be combined to create the target NUON data file. You may add files to this list, delete files from the list, change the position of items in the list, or change the options for each file.

Once you have all of the desired files added to your list, then you can tell the program to create the NUON data file.

5.14.1 How Does It Work

Upon launch, the program normally starts with an empty list window. Please note that you can also launch the program by double-clicking a list file, in the same way you might launch Microsoft Word by double-clicking on a .DOC file.

You may also select ***New*** from the *File* menu to open a new, empty list window.

5.14.1.1 Adding Files To The List

Once you have an empty list window, the easiest way to add files is to simply drag them from Windows Explorer into the ***NUON Data Tool*** window and release them. This adds the files to the list.

You can also add files by right-clicking the mouse over the list window. When the context popup menu appears, the first item should be ***Add New File***. This will bring up a file selector dialog so that you may select a file to be added to the list.

Once files have been added to the list, you can add more items, delete items, alter the order, or change the alignment options for individual items.

5.14.1.2 Changing The List Order

You can change the order of items in the list by right-clicking the mouse over an item. This will bring up the context popup menu, which includes choices for moving files up or down in the list by an item at a time, or by all the way to the top or bottom.

5.14.1.3 Refreshing The List

If any of the files in the list are altered after the list has been created, you should refresh the list to correct it. This will cause the program to read the current file size and time/date stamp information.

[2] Please be aware that some NUON DVD players, especially older models such as the Samsung N-2000 or Toshiba SD-2300, may not support the use of CD-R media.

You may refresh an individual file entry by right-clicking the mouse over an item. This will bring up the context popup menu, which includes a choice for ***Refresh Size & Time/Date Stamp***. If the file cannot be found, you will be asked if the entry should be removed from the list.

You may also refresh all the files in the list by selecting the menu item titled ***Refresh Size & Time/Date Stamp*** in the *Options* menu. If any items cannot be found, the file size will be reset to zero.

5.14.1.4 Options

For each file you add to the list, the ***NUON Data Tool*** gives you the option of:

- Output C/C++ Header Files — This option will cause the ***NUON Data Tool*** to create include files for C & C++ that contain preprocessor definitions describing each file in your list.

5.14.1.5 Creating Your Data File

Also see ***Include File Generation*** below.

5.14.1.6 Include File Generation

When the ***NUON Data Tool*** creates your data file, your application needs a method to determine where each piece of data is located and how big it is.

To accommodate this, the ***Create NUON Data File*** dialog includes an option for creating a special C/C++ include file with definitions for the location and size of each piece of data. There is also an option for creating the equivalent Llama assembly language file.

For each file in your list, the program creates six definitions. Each definition begins with a modified version of the original source file's complete pathname with all of the special filename characters like ":" or "\" converted to underscore ("_") characters. After the pathname comes a special suffix that indicates the type of information represented by the definition. For example, if you had a source file named:

```
C:\NUONSpaceCannon\Data\Level1\background.jpg
```

Then for the C/C++ include file option, you would see a set of six definitions that look like this.

```
#define C__NUONSpaceCannon_Level1_background_jpg_FILEREF     (2)
#define C__NUONSpaceCannon_Level1_background_jpg_BLOCK       (12)
#define C__NUONSpaceCannon_Level1_background_jpg_NUMBLOCKS   (210)
#define C__NUONSpaceCannon_Level1_background_jpg_NUMBYTES    (429602)
#define C__NUONSpaceCannon_Level1_background_jpg_BLOCKOFFSET (0)
#define C__NUONSpaceCannon_Level1_background_jpg_BYTEOFFSET  (24576)
```

The meaning of each of the suffixes is defined in the table below:

Suffix	Meaning
_FILEREF	Indicates which entry this was in the list. Useful mainly for reference purposes.
_BLOCK	Indicates the starting sector number for the data, relative to the beginning of the file. This the value that would be passed to the ***_MediaRead*** function.
_NUMBLOCKS	Indicates how many blocks you must read in order to obtain all of the data for the item. Note that an item may not be aligned to the beginning of a sector, so this value may be larger than you would expect from the file size
_NUMBYTES	Indicates the size in bytes of the original data file
_BLOCKOFFSET	Indicates the offset in bytes where the data begins, relative to the first data block of the item. For example, if the _BLOCK definition says 12, and _BLOCKOFFSET says 356, then the data item begins at byte offset 356 of sector 12 of the data file generated by the ***NUON Data Tool***. This will always be zero for files where you have specified sector boundary alignment.
_BYTEOFFSET	Indicates the offset in bytes where the data begins, relative to the overall data file generated by

Suffix	Meaning
	the ***NUON Data Tool***. Useful mainly for reference purposes.

5.14.1.7 The Home Directory

The *Home Directory* is simply the directory that contains all of the source files used to generate your data file. For example, suppose that your project has the following three source files:

```
C:\NUONSpaceCannon\Level1\background.jpg
C:\NUONSpaceCannon\Level2\background.jpg
C:\NUONSpaceCannon\Level3\background.jpg
```

Each of the files is located in its own folder, but all of those folders are contained in:

```
C:\NUONSpaceCannon\
```

So this would be your *Home Directory*. Note that in order to really take advantage of the *Home Directory* feature, all of the source data files must be located in the same folder of the same drive.

As we saw earlier, the complete pathname of each item is used to create the definitions that describe the data file being generated. However, as our example in section 5.14.1.6 indicates, this can result in some very long definitions.

When you specify the *Home Directory*, the ***NUON Data Tool*** will automatically remove that part of the pathname from the beginning before generating definitions. So if our home directory was specified as shown above, then our definitions for the first file would be:

```
#define Level1_background_jpg_FILEREF      (2)
#define Level1_background_jpg_BLOCK        (12)
#define Level1_background_jpg_NUMBLOCKS    (210)
#define Level1_background_jpg_NUMBYTES     (429602)
#define Level1_background_jpg_BLOCKOFFSET  (0)
#define Level1_background_jpg_BYTEOFFSET   (24576)
```

instead of:

```
#define C__NUONSpaceCannon_Level1_background_jpg_FILEREF     (2)
#define C__NUONSpaceCannon_Level1_background_jpg_BLOCK       (12)
#define C__NUONSpaceCannon_Level1_background_jpg_NUMBLOCKS   (210)
#define C__NUONSpaceCannon_Level1_background_jpg_NUMBYTES    (429602)
#define C__NUONSpaceCannon_Level1_background_jpg_BLOCKOFFSET (0)
#define C__NUONSpaceCannon_Level1_background_jpg_BYTEOFFSET  (24576)
```

As you can see, this will eliminate a lot of typing in the long run.

5.14.1.8 Global Options

The ***Global Options*** menu item in the *Options* menu allows you to specify several default options that will apply to your file list.

- Default sector alignment for each file added to the list
- Sector size of the target media. This is almost always 2048 bytes for DVD or CDROM, but could also be different values for other types of media. This value is only used to determine sector alignment.
- Home Directory
- Filename for Data File Generation

5.14.1.9 Using The .H Include File Definitions

Using the definitions provided in the generated include file is quite simple. If you are trying to access the data for **background.jpg** and you have definitions that look like this:

```
#define Level1_background_jpg_FILEREF      (2)
#define Level1_background_jpg_BLOCK        (12)
#define Level1_background_jpg_NUMBLOCKS    (210)
#define Level1_background_jpg_NUMBYTES     (429602)
#define Level1_background_jpg_BLOCKOFFSET  (0)
#define Level1_background_jpg_BYTEOFFSET   (24576)
```

Then your call to read the data will look something like this:

```
error = MediaRead( media_handle, MCB_END,
                   Level1_background_jpg_BLOCK,
                   Level1_background_jpg_NUMBLOCKS,
                   buffer, callbackfunction );
```

The *media_handle* parameter is the successful return value from a previous call to *MediaOpen(). The MCB_END* parameter indicates that the callback will occur after the last block is read.

The next two parameters are taken from the definitions generated by the ***NUON Data Tool*** and they represent the starting sector of the data, relative to the beginning of the open file, followed by the number of blocks that must be read 9in order to get all of the data.

Next we have the memory address of the buffer that will receive the data.

Last, we have a function pointer to our callback routine. Because we're using ***MCB_END***, this function will be called when the data has been completely read into memory, or if an error occurs.

See the NUON ***BIOS Documentation*** for more information on the details of the media access functions.

5.15 MERGEMF Utility

The MERGEMF utility is designed to convert a MIDI type 1 sequence file into a MIDI type 0 sequence file so that it may be used with the MIDI parser that is built into the NUON's LIBSYNTH library. The MERGEMF tool can also take a MIDI file that is already a type 0 file and filter out MIDI events that are not applicable for playback with LIBSYNTH.

The format of the command line is:

```
mergemf [options] inputfile outputfile
```

The *inputfile* parameter specifies the source MIDI sequence file to be converted. The *outputfile* parameter specifies the name to be used for the new MIDI file that will be created. This may be the same as the source file, in which case the original file will be overwritten. The command *options* are described in the table below.

Option	Description
-StripText	Filter out all text meta events (such as would be used for song lyrics)
-Karaoke	Similar to the ***–striptext*** option, except will not filter out those events required for MIDI Karaoke files.

6. 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 5. For some tools, no further documentation is required. For others, the information in chapter 5 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.

6.1 Tools

Chapter 5 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
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

6.2 System & Hardware

Title	Description	Format
NUON BIOS Documentation	Describes the NUON BIOS API functions	PDF
NUON Multi-Media Architecture Programmer's Guide	The basic documentation for the NUON processor. Includes an assembly language reference, register documentation, and tons of other important details.	PDF
De Re NUON	An introduction and overview of programming for NUON	PDF

6.3 Libraries

Title	Description	Format
M3DL Graphics Library	Details about the M3DL 2D & 3D Graphics Library	PDF
MML2D 2d Library	API for 2d graphics (lines, circles, text, etc.)	PDF
MML3D 3D Library	Details about the original NUON 3D Graphics library.	PDF
NUON Audio Library Functions	Details about the LIBSYNTH and LIBNISE audio libraries	PDF
mGL 3D Graphics Library	Details about the mGL 3D Graphics Library	PDF

Title	Description	Format
Merlin Utility Functions Programmer's Manual	Details about the Merlin Utilities library	PDF
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

6.3.1 Sample Program Source Code

At the current time, the various sample programs in the NUON SDK come with an HTML document that provides a basic overview of the source code.

7. 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.

7.1 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.[3]

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.

[3] 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.

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.

7.1.1 Running The Sample Program

The process above should have resulted in a file named **NUON.CD**. Create a CD-R disc with this file in the root directory. If your CD-R mastering software offers a choice of disk format, make sure to select ISO9660. Other formats will not work.

Once the disc has been created, it should boot in a Samsung N-501 or later NUON DVD player.

7.2 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.

8. 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.

8.1 Memory Usage

8.1.1 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.

8.1.2 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.

8.1.3 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.

8.2 Runtime Memory Allocation

8.2.1 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.[4]

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.

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

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 8.2.2, it is possible to do your own heap management.

8.2.2 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.

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.

8.2.3 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 8.2.1 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 8.2.2 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.

8.2.3.1 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.

8.3 DMA Operations

8.3.1 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.

8.3.2 Issuing DMA Commands

If you're going to perform DMA operations from the primary processor while the cache is turned on, 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.

8.4 Timers & Interrupts

8.4.1 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 8.4.2.

8.4.2 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.

9. 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***.

9.1 Tools

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

A: The "official" operating system that VM Labs supports and tests against is Windows 2000. Testing is not done on other versions of Windows. However, there are no known problems with Windows NT 4.0, Windows 98, Windows Millennium, or Windows XP.

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.

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. The more, the better. If all else

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.

9.2 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.

9.3 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.

10. 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. Generally not used except for small pieces of test code.

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.