Intel® I210 and I211 Design-In for ARM

eepromARMtool Usage Guide

Networking Division (ND)

July 2013

Intel Confidential

Revision 1.1

LEGAL

Revision History

Revision	Date	Comments
1.1	July 2013	Updates to Section 2.0, “Introduction” and Section 4.5, “Programming a Device”.
1.0	April 2013	Initial release (Intel Confidential)

NOTE: ***This page intentionally left blank.***

Contents

NOTE: ***This page intentionally left blank.***

1.0 About This Document

This document contains an overview of Intel's eepromARMtool for Linux®, a key tool for designing solutions on Intel Ethernet Controllers I210 and I211. It provides a discussion of the tool's features, and examples of it's use.

1.1 Target Audience

The intended audience of this document are customers who wish to use certain Intel LAN Controllers on ARM-based systems. For a list of supported devices, refer to Section 2.1.

Thorough understanding of this document, and the tool in its current state, can assist you in implementing your own solutions.

1.2 Acronyms and Abbreviations

Table 1 lists and defines the acronyms and abbreviations used in this document.

Table 1. Acronyms and Abbreviations

Acronym or Abbreviation	Description
EEPROM	Electrically Erasable Programmable Read Only Memory
Flash	Another term for external Non-Volatile Memory (NVM).
iNVM	Internal Non Volatile Memory - iNVM is similar to OTP memory except iNVM allows a limited number of modifications.
NIC	Network Interface Card
NVM	Non-Volatile Memory (external)
OS	Operating System
OTP	One Time Programmable

2.0 Introduction

eepromARMtool is a basic, standalone tool that can be run from the Linux command line. It provides an avenue for basic NVM (both iNVM and flash) operations when other EEPROM tools (such as EEUpdate or LANConf) cannot be used. However, eepromARMtool is not designed to be a replacement - use EEUpdate and LANConf whenever possible.

For IEEE testing, use ieeeARMtool and the help parameter for usage associated with ieeeARMtool. Obtain the source from your Intel representative.

Important: This tool is a work in progress and may contain unknown bugs. To reduce or eliminate undesired results it is recommended that you become familiar with the information provided in this document, and strongly advised that you read and understand all information provided about eepromARMtool.

Important: While Intel provides eepromARMtool as a convenience, changes to the tool may adversely affect its performance. The user assumes all risk, and changes to the tool are not supported by Intel.

2.1 Supported Devices

The eepromARMtool is supported on the following Intel devices:

- Intel® Ethernet Controller I211 with internal NVM (iNVM)
- Intel® Ethernet Controller I210 with flash and iNVM
- Intel® 82574 Gigabit Ethernet Controller

2.2 Tool Features

Table 2 lists the features of the eepromARMtool.

It is possible for both NVM and iNVM to be present on a device. This tool always defaults to use the flash for read and write operations, such as write and dump.

Table 2. eepromARMtool Features

Feature	Description
Display Supported LAN Controllers	Shows a list of supported LAN Controllers in the system. LAN Controllers are enumerated in the order they are found on the PCI bus. Information is displayed showing the bus, device, and function for the LAN Controller. It also displays the type of LAN Controller and whether the LAN controller has NVM, iNVM, or both. For more information, refer to Section 4.3, "Displaying Devices" on page 4
Write NVM	Programs the NVM with a specified image file to a specified LAN Controller. The NIC number where the controller resides and image filename are provided by the user in the command line invocation. When programming the iNVM, the user should fully understand the details listed in Section 6.0, "Considerations for iNVM Programming" on page 7.
Dump NVM	Reads the contents of the NVM specified by the user, and writes that data to a file. The resultant file is placed in the directory that contains the tool application. For more information, refer to Section 4.4, "Dumping a Device" on page 5
Test NVM	Not implemented.

3.0 Runtime Environment

This section contains the system and development requirements for eepromARMtool. Select information about the systems used during development is also included in this section for reference.

3.1 Considerations for an ARM-Based Solution

A major difference between x86/64 platforms and ARM platforms is that EEUpdate and LANConf do not currently support ARM. Therefore, you should use eepromARMtool for ARM based platforms.

The major design consideration for ARM vs. x86/64 platforms is that all feature implementation is done using a programming language library compatible with the OS. You may need to explore the library offered by the programming language. eepromARMtool was developed using the GNU C library. Some features assume understanding of memory structure and layout in the OS.

Another thing to consider is that ARM systems can be Bi-Endian. Therefore, you should verify the Endianness being used, especially when programming values at the bit level.

This document does not cover verifying Endianness. However, you should be aware that eepromARMtool reflects a Little Endian implementation to match the hardware datasheets.

3.2 System Requirements

Following is the list of system requirements:

- Supported Intel Ethernet LAN controller (as described in Section 2.1, “Supported Devices” on page 2)
- Linux OS
- GNU C Library (glibc)
- GNU Compiler (gcc)

3.3 Development Environments

The tool has been shown to produce correct results on the following systems during development:

- Red Hat Enterprise Linux 6.2 Kernel release 2.6.32, 0x86_64 Architecture
  - This system was used for application development purposes only. Use EEUpdate or LANConf for x86/x64 architectures.
- Ubuntu Linux for Tegra from nVidia release 15

4.0 Using the Tool

This section provides getting started information as well as detailed usage instructions. You should understand this section completely before attempting to program using the tool.

4.1 Getting Started

Before getting started, address the following points:

- Obtain all of the necessary source files for the tool. For relevant source code, see your Intel representative.
- Ensure the System Requirements are met; a system identical to those listed in Section 3.3 would also be sufficient. However, this tool was developed on a single system. Since there is a high degree of variability between systems, your experience may differ.
- Familiarize yourself with this documentation.
- To run the tool, ensure root privilege access.

4.2 Running the Tool

Follow these steps:

1. Navigate to the source directory. This directory should contain a Makefile, an include directory, and the c source files.
2. To begin compiling using the provided Makefile, enter the **make** command:

   ```
   make
   ```

   This command may produce a warning similar to the following:

   ```
   make: warning: Clock skew detected. Your build may be incomplete
   ```

   To eliminate this warning, run a sequence of commands similar to the following:

   ```
   make clean
   touch -r *
   make
   ```

3. To run the tool, enter the command, replacing *arglist* with the specific argument list that tells the tool what functionality should occur.

   ```
   sudo ./eepromARMtool [arglist]
   ```

For additional information, refer to Section 4.3, Section 4.4 and Section 4.5.

4.3 Displaying Devices

To display the supported devices in the system, run the tool with no arguments, as shown in the example below.

```
sudo ./eepromARMtool
```

An example of the command output is shown in Figure 1.

```
Intel(R) Eeprom ARM Tool ARM OTP Programming Tool
Provided under the terms of a CNDA.  Do Not Distribute.
Copyright(C) 2012 by Intel(R) Corporation
NIC     BUS     DEV     FUN     Silicon Memory Type Present
===     ===     ===     ===     =====   =====================
 1      2        0       0      I211         INVM
```

Figure 1. Displaying Devices Example Output

4.4 Dumping a Device

To dump the contents of a device into a file, follow these steps:

1. Display the devices using the command in Section 4.3.

   sudo ./eepromARMtool

2. Obtain the NIC number that corresponds to the desired device.
3. Run a command like the following, replacing *X* with the NIC number.

   sudo ./eepromARMtool -dump -NIC=*X*

This command creates an output file in the same directory. The file is of the format:

<*silicon*>NIC<*X*>.OTP

where <*silicon*> is the silicon type (I210, I211, 82574) and <*X*> is the NIC number specified in the invocation.

For example, if you dump an I210 that is NIC 1 in the system, the file is named I210NIC1.OTP.

eepromARMtool defaults to dumping flash memory when both flash and iNVM are present. However, you can change this default by modifying the code to create a custom solution.

4.5 Programming a Device

If all of the information that needs to be written to the device is in a single file, the command below shows how to use the file to program the device.

sudo ./eepromARMtool -write -NIC=*X* **-f=**<*filename*>

In this example, <*filename*> is a file contained in the same directory, and *X* is the NIC number discovered using the display command in Section 4.4.

eepromARMtool defaults to dumping flash memory when both flash and iNVM are present. However, you can change this default by modifying the code to create a custom solution.

5.0 Input File Format

There are two different input files, depending whether the device has standard flash or iNVM. Ensure the correct input file format is used to prevent undesired results.

This section is not intended to assist in creating exact values with which to program the flash or iNVM. For that information, refer to the relevant product datasheet.

Note: To avoid unpredictable behavior, do not use programming data files larger than the size of the flash.

5.1 iNVM File

An input file for an iNVM is a table of 8 columns and X rows of 32 bit words, presented in hexadecimal format. There should always be exactly 8 non-empty columns. However, the number of rows (X) varies depending on how much data needs to be programmed to the device.

The tool reads 8 words at a time, so there should always be 8 words per row. Those words can be zero or non-zero. Typical iNVMs are 256 bytes in size, translating to 8 columns and a maximum of 8 rows.

An example snippet is shown below in Section 2.

```
16E80002 00001541 038D0002 0B403C21 402F1411 34003611 15391A11 05844211
73431E19 0000001A 08100241 16D10002 00FF00A8 16D00002 5E000090 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Figure 2. Snippet of File for Programming iNVM

5.2 Flash File

An input file for a flash is a table of 8 columns and X rows of 16 bit words, presented in hexadecimal format. There should always be exactly 8 non-empty columns of data (the values can be zero or non-zero). However, the number of rows (X) varies depending on how much data needs to be programmed to the device.

An example snippet is shown below in Figure 3.

```
1B00 B121 C250 0520 F746 2010 FFFF FFFF
FAFA 02C2 026B 0001 8086 10D3 FFFF 9C58
0000 2001 7E94 FFFF 1000 0048 0000 2704
```

Figure 3. Snippet of File for Programming Flash

6.0 Considerations for iNVM Programming

This tool has been designed for programming blank iNVMs, but it can be extended to program a non-blank iNVM using the information below.

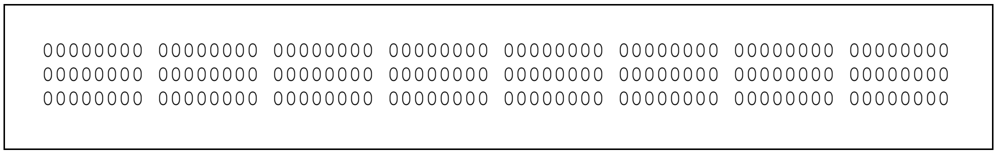
6.1 Blank iNVM

To determine if the iNVM is blank, execute the dump command on the command line, as shown in the following example:

sudo ./eepromARMtool -dump -NIC=*X*

where *X* is the NIC number.

If the device is blank, a message displays and prints to the file. Furthermore, the first few words and most of the others are zero, as shown in Figure 4.



```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Figure 4. Snippet of File Containing Dumped Contents of Blank iNVM

Execute the write command to write an image to a blank iNVM.

6.2 Non-Blank iNVM

To verify if the iNVM has been previously programmed and is not blank, execute the dump command on the command line, as shown in the following example:

sudo ./eepromARMtool -dump -NIC=*X*

where *X* is the NIC number.

If there is non-zero data in any of the first words of the dumped file, the iNVM is not blank and has been previously programmed. Although the exact values may differ, the output file looks similar to that shown in Figure 1 on page 5.

When programming the non-blank iNVM, manually merge the current iNVM contents with the new data to be programmed to create a merged data file that is used with eepromARMtool. Use the tool to dump the current iNVM contents, then modify the resultant file to reflect the merge with the new data.

Note: iNVM bits can only be changed from 0 to 1 once.

6.2.1 Merging Data

Since iNVM bits can only be changed from 0 to 1 once, practice dumping the iNVM, modifying the file as desired, and reprogramming it. The first word in the iNVM may not be able to represent the first word in the unmerged data file, and therefore won't be correctly represented in the end result.

Table 3. Programming the iNVM - Manually Merging Data Snippet

Current iNVM Dump Contents	User File Contents (Desired Active iNVM Contents)	Merged Data

`7431041F 15391A19`	`36A00019 029F0219`
	`74310419`
		`7431041F 15391A1F`
		`36A00019 029F0219`
		`7431041  74310419`

`FFFFFFFF 029F0FFF`
`7431041F 15391A19`
	`15391A19 36A00019`
	`029F0219 74310419`
		`FFFFFFFF 029F0FFF`
		`7431041F 15391A19`
		`36A00019 029F0219`
		`74310419`

The merge examples shown above in Table 3 are not meant to reflect valid iNVM or merged data for an actual device. Rather, they are meant to convey the method used to manually merge data.

7.0 Troubleshooting and Help

- Do not attempt to program a file that is larger than the flash size.
- Use proper word sizes when programming different types of flash.

§ §