



Grantley-EP Server Reference Code User's Guide

June 13th, 2016
Revision 3.6.0

INTEL CONFIDENTIAL - NDA

Copyright 2006 – 2016 Intel Corporation All Rights Reserved. This document contains trade secrets and proprietary information of Intel Corporation or its suppliers and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.



INTEL CONFIDENTIAL - NDA

Copyright 2006 – 2016 Intel Corporation All Rights Reserved.

The source code contained or described herein and all documents related to the source code ("Material") are owned by Intel Corporation or its suppliers. Title to the Material remains with Intel Corporation, its suppliers, or licensors. The Material contains trade secrets and proprietary and confidential information of Intel Corporation, its suppliers, and licensors, and is protected by worldwide copyright and trade secret laws and treaty provisions. No part of the Material may be used, copied, reproduced, modified, published, uploaded, posted, transmitted, distributed, or disclosed in any way without Intel's prior express written permission.

No license under any patent, copyright, trade secret or other intellectual property right is granted to or conferred upon you by disclosure or delivery of the Materials, either expressly, by implication, inducement, estoppel or otherwise. Any license under such intellectual property rights must be express and approved by Intel in writing.

Unless otherwise agreed by Intel in writing, you may not remove or alter this notice or any other notice embedded in Materials by Intel or Intel's suppliers or licensors in any way.



Contents

1	INTRODUCTION.....	7
1.1	REFERENCE CODE PACKAGE CONTENTS.....	7
2	MINIBIOS OVERVIEW	8
2.1	MINIBIOS	8
2.2	BUILD TOOLS.....	8
2.3	BUILD OPTIONS	8
2.4	DIRECTORY STRUCTURE.....	8
2.5	BUILD ENVIRONMENT.....	10
2.6	BUILDING THE MINIBIOS	10
2.7	DEBUGGING THE MINIBIOS USING INTEL SOFTSIMULATION	11
2.7.1	Configuring SoftSimulation to Load the BIOS image file	11
2.7.2	Symbolic Debugging of MiniBIOS	11
2.7.3	Using the DUMPBIN utility	11
3	DETAILS OF MINIBIOS.....	14
3.1	MINIBIOS LAYOUT	14
	FIGURE 1: MINIBIOS LAYOUT	14
3.2	BUILDING THE MINIBIOS	14
3.3	EXECUTION FLOW	15
3.4	MINIBIOS TOOLS	15
3.4.1	FindEntryPoint.....	15
3.4.2	PlacePE	15
4	COMMON REFERENCE CODE COMPONENTS	17
4.1	OVERVIEW.....	17
4.2	BIOS REFERENCE CODE HIGH LEVEL EXECUTION FLOW	17
4.3	ASSUMPTIONS	17
4.4	PLATFORM PORTING AND REFERENCE CODE HOOKS	17
4.4.1	Options.....	17
4.4.2	lowGap.....	18
4.4.3	highGap.....	18
4.4.4	serialDebugMsgLvl.....	18
4.4.5	bsdBreakpoint	18
4.4.6	maxAddrMem.....	18
4.4.7	SocketType	18
4.4.8	debugPort.....	18
4.4.9	nvrampPtr.....	18
4.4.10	sysHostBufferPtr	18
4.4.11	mmCfgBase	18
4.4.12	mmCfgSize.....	18
4.4.13	mmiolBase	19
4.4.14	mmiolSize.....	19
4.4.15	mmiohBase	19
4.4.16	mmiohSize	19
4.4.17	pchUmaEn	19
4.4.18	numaEn.....	19
4.4.19	isocEn	19
4.4.20	DciVcm.....	19
4.4.21	MeSegEn	19
4.4.22	dcaEn.....	19
4.4.23	PSMIEncabledSupport.....	19
4.4.24	BdatEn	19
4.4.25	consoleComPort.....	19
4.4.26	debugComPort.....	19
4.4.27	usbDebugPortSetup.....	19
4.5	SCRATCH PAD REGISTER USAGE	20
4.6	HIGH AND LOW MMIO BAR ALLOCATION	20
5	CPU INITIALIZATION REFERENCE CODE	20



5.1	OVERVIEW.....	20
5.2	CPU REFERENCE CODE EXECUTION FLOW	20
5.3	ASSUMPTIONS	21
5.4	PLATFORM BIOS INTEGRATION GUIDE	21
5.4.1	Build-Time Equates and Defines	21
5.4.2	Input Parameters.....	22
5.5	ROUTINES FOR PLATFORM CUSTOMIZATION	22
5.6	POST CODES AND SERIAL PORT DEBUG MESSAGES	23
5.6.1	POST Code at IO Port 0x80.....	23
5.6.2	Serial Port Debug Messages.....	23
6	QPI REFERENCE CODE.....	25
6.1	OVERVIEW.....	25
6.2	INPUT PARAMETERS	25
6.2.1	PPINrOptIn.....	25
6.2.2	BusRatio/IoRatio/MmioIRatio	25
6.2.3	LegacyVgaSoc	25
6.2.4	MmioP2pDis.....	25
6.2.5	IsocAzaliaVc1En	26
6.2.6	DebugPrintLevel.....	26
6.2.7	AltRtid2S	26
6.2.8	ClusterOnDieEn	26
6.2.9	MctpEn	26
6.2.10	E2EParityEn.....	26
6.2.11	EarlySnoopEn	26
6.2.12	DegradePrecedence	26
6.2.13	QpiLinkSpeedMode	26
6.2.14	QpiLinkSpeed.....	26
6.2.15	QpiLinkL0pEn.....	26
6.2.16	QpiLinkL1En.....	26
6.2.17	QpiLbEn	26
6.2.18	lloUniphyDisable	27
6.2.19	QpiLinkCreditReduce	27
6.2.20	QpiConfigTxWci	27
6.2.21	Qpi AdVNACreditArbThreshold	27
6.2.22	QpiCrcMode	27
6.2.23	QpiCpuSktHotPlugEn.....	27
6.2.24	QpiCpuSktHotPlugTopology	27
6.2.25	QpiSkuMismatchCheck	27
6.2.26	PhyLinkPerPortSetting	27
6.2.27	Input Parameter Default Values	27
6.3	OUTPUT PARAMETERS.....	28
6.3.1	CpuInfo.....	28
6.3.2	SysConfig.....	29
6.3.3	WarningLog.....	29
6.3.4	OutLegacyVgaSoc	29
6.3.5	OutIsocEn	29
6.3.6	OutMesegEn	29
6.3.7	OutIsocAzaliaVc1En.....	29
6.3.8	OutClusterOnDieEn.....	29
6.3.9	OutMctpEn	29
6.3.10	OutE2EParityEn	29
6.3.11	OutEarlySnoopEn.....	29
6.3.12	QpiCurrentLinkSpeedMode	29
6.3.13	OutQpiLinkSpeed.....	29
6.3.14	OutQpiLinkL0pEn.....	29
6.3.15	OutQpiLinkL1En.....	29
6.3.16	OutQpiLinkL0rEn.....	29
6.3.17	OutlloUniphyDisable.....	30
6.3.18	OutQpiCrcMode	30
6.3.19	RasInProgress.....	30
6.3.20	RasEvtType.....	30
6.3.21	RasSocId.....	30



6.3.22	<i>ProgramNonBC</i>	30
6.3.23	<i>ProgramBC</i>	30
6.4	ERROR/STATUS CODES	30
6.5	WARNING LOGS.....	32
6.6	PORTING GUIDELINES.....	32
6.6.1	<i>QpiHooks.c</i>	32
6.6.2	<i>QpiPlatformEparam.c</i>	32
7	MEMORY REFERENCE CODE	33
8	COMPATIBLE DATA STRUCTURE	34
8.1	REQUIREMENTS	34
8.2	PROPOSED SOLUTION	34
8.3	COMPATIBLE ACPI TABLE	34
8.4	SCRATCHPAD	36
8.5	COMPATIBLE BIOS STRUCTURE HEADER	36
8.6	BIOS STRUCTURE DEFINITIONS	37
9	FAULTY PARTS TRACKING	41
10	STORAGE FEATURE MRC USERS GUIDE	42
10.1	MRC CONFIGURATION OPTIONS	42
10.2	OUTPUT PARAMETERS.....	43
10.3	MEMORY MAP LAYOUT	43



Revision History

Rev. No.	Description	Date
0.50	Initial revision	Sept 2012
0.60	Update Power-on settings	Jan 2013
0.70	Update for V0.70 release	March 12th, 2013
0.72	Update for V0.72 release	April, 2013
0.75	Update for V0.75 release	May 16th, 2013
0.76	Update SysSetup and OptionsExt fields, STS code and major error code	June 26th, 2013
0.77	Revised RasModeEx option.	August 9th, 2013
0.78	Added new option: ATTEMPT_FAST_BOOT_COLD	September 20th, 2013
0.79	Added option fields in OptionsExt (bit11 to bit14). Added a new warning error code 0x25 for "COD Enabled but memory not behind each HA". Added MCODT value selection.	October 4th, 2013
0.80	First Beta release. Added new options and warning errors for dram maintenance test.	October 26th, 2013
0.81	Beta release: added a new option for DLL reset test loop	November 2nd, 2013
0.82	Beta release: update MRC entry point and setup options	December 20th, 2013
0.83	Beta release: update section 10: " Storage Feature MRC Users Guide"	Jan 2014
0.84	Beta release: moved RCOMP_TYPE (bit12) in "option" to RCOMP_TYPE (bit15) in "optionExt" field. Removed "pagetablealias" field in memSetup structure.	February 12th, 2014
0.90	Production Candidate release. Added new MRC warning code.	March 9th, 2014
0.96	Production Release. Added one new option: dramraplExtendedRange	March 27th, 2014
1.00	Production Validation release	April 25th, 2014
1.01	Production Validation 1 release	May 15th, 2014
1.02	Production Validation 2 release	June 16th, 2014
1.03	Production Validation 3 release	July 12th, 2014
1.04	Production Validation 4 release. No change from PV3 release.	July 23rd, 2014
1.05	Production Validation 5 release. Description update for SPLIT_BELOW_4GB option	August 15th, 2014
1.06	No change	October 2nd, 2014
1.7.1	Updated for converged core memory reference code	March 20th, 2015
1.7.2	No change	April 1st, 2015
1.7.3	No change	May 5th, 2015
1.8.0	No change	May 29th, 2015
1.8.1	Added QPI status and minor error codes	July 1st, 2015
1.8.5	No change	July 27th, 2015
1.9.0	No change	August 11th, 2015
2.0.0	No change	Sept 4th, 2015
2.4.0	No change	Sept 18th, 2015
2.9.0	No change	Oct 30th, 2015
2.9.1	No change	Nov 12th, 2015
2.9.5	No change	Nov 30th, 2015
3.0.0	No change	Dec 11th, 2015
3.1.0	No change	Feb 5th, 2016
3.2.0	No change	Mar 10th, 2016
3.3.0	No change	April 29th, 2016
3.4.0	No change	May 13th, 2016
3.5.0	No change	May 27th, 2016
3.6.0	No change	June 13th, 2016

1 Introduction

This document accompanies the Reference Code. The primary purpose of this document is to describe the contents of the Reference Code package. In addition this document describes the integration methods for Independent BIOS Vendors (IBVs) and Intel OEM customers. This document caters to EP processors (HSX-EP and BDX-EP)

1.1 Reference Code package contents

The Reference Code package is a collection of code modules and build tools that can be used to construct an executable BIOS image. It is written primarily in C and ASM programming languages, and it uses the processor cache for stack and data storage.

The code is separated into two categories, Reference code and Sample code. The reference code is used directly in Intel validation and CRB BIOS, and it is expected to be compatible with all platforms that follow the Platform Design Guide. The sample code is used only in the MiniBIOS package and is not expected to be compatible with all platforms. Customers should only modify the sample code.

The reference code package contains:

- Sample code to enter 32-bit protected mode and call into the early reference code module.
- Reference code to initialize the processor(s) for basic operation including microcode loading, MMCFG/bus allocation, cache in No-Eviction Mode (NEM) and other early processor/chipset settings.
- Sample code to find the microcode images, locate the NEM code/data ranges, get early setup options, initialize the serial port, and call into the main reference code module.
- Reference code to initialize the QPI links for proper routing and full speed operation among all processors.
- Reference code to detect DIMMs, initialize each memory controller and establish a system address map.
- Reference code for shared functions like MMCFG access, serial debug messages and other common tasks.
- Sample code to initialize default setup options and platform hooks for use within the reference code.
- Build tools for Portable Executable (PE) files and make files for constructing the MiniBIOS image.

2 MiniBIOS Overview

2.1 MiniBIOS

The reference code package primarily consists of processor initialization, QPI and Memory Reference Code (MRC). The package can be compiled into a binary image called the MiniBIOS for stand-alone execution on the Intel Customer Reference Board (CRB). The image will not be able to boot an OS, but it is capable of executing from the CPU reset vector to the completion of system memory initialization. The MiniBIOS may be used as a sample implementation by IBVs and OEM customers that are developing platforms based on Intel processors.

2.2 Build Tools

The MiniBIOS build requires installation of several tools to compile and link the source code. The minimum tool requirements are listed below:

1. Microsoft ® Macro-Assembler version 6.15 - ml.exe
2. Microsoft ® Segmented Executable Linker version 5.31 - \$(MASMPATH)\link.exe
3. Microsoft ® C to MASM Include File Translator 7.10.3077 - h2inc.exe
4. Microsoft ® 32-bit C/C++ Optimizing Compiler Version 13.10.3077 or later - cl.exe
5. Microsoft ® Incremental Linker Version 7.10.3077 or later - link.exe

The latest versions of these tools are included in the Microsoft ® Visual Studio .NET, with exception of the 16-bit linker and the H2INC utility. The 16-bit linker is included in the MASM installation and must be referenced by the MASMPATH environment variable. The H2INC utility was last included with the .NET 2003 installation.

Functional validation of the reference code will use the latest .NET 2008 versions of the build tools. Support for previous tool versions is limited to build errors only, and does not include execution coverage.

Please note that Microsoft ® Macro-Assembler Version 6.11d exhibits structure packing issues that can break the OEM BIOS interface to MRC. This issue is referenced in Part 7 of the Readme.txt file from the ML6.14.exe package under Microsoft Knowledge Base Article [#228454](#).

2.3 Build options

Memory Build Options:

1. SERIAL_DBG_MSG – When defined enables code for Serial debug messages on Serial port. Default defined
2. LRDIMM_SUPPORT - When defined enables code for LRDIMM support. Default defined
3. INPHI_LRBUF_WA – When defined enables code for LRBUF Workarounds. This will only be enabled if LRDIMM_SUPPORT is enabled. Default defined
4. IDT_LRBUF_WA – When defined enables code for LRBUF Workarounds. This will only be enabled if LRDIMM_SUPPORT is enabled. Default defined
5. BDAT_SUPPORT – When defined enables code for Bios Compatible data structure. Default defined
6. QR_DIMM_SUPPORT - When defined enables code for Quad rank DIMM support. Default defined
7. MARGIN_CHECK – When defined enables code for Dimm Margin checking. Default not defined
8. ME_SUPPORT_FLAG – When defined enables code for ME UMA support. Default not defined
9. XMP_SUPPORT – When defined enables code for Extreme memory profile support. Default not defined.
10. HA2_PLATFORM – When defined enables code for 2 Home Agent\Memory Controller support. Default defined.
11. ASM_INC – When defined enables usage of H2INC utility. Default undefined, need to define for legacy BIOS.
12. LT_STRUCT – When defined enables structures for TPM data. Default undefined.
13. QPI_HW_PLATFORM – When defined enables QPI code for Hardware platform. Default defined.
14. IA32 – When defined enables code to compile reference code in IA32 environment. Default defined. If reference code need to be compiled in X64 then undefine this flag.

2.4 Directory Structure

The following sections of this document assume that the MiniBIOS package is located in the root directory Z:\. While this is not a strict requirement to build the MiniBIOS, it does improve portability of the output files among

multiple hosts. Debug information can be loaded from a remote host without rebuilding source code on a local drive provided that both systems map the root directory to Z:\.

There are two separate methods to create virtual drive Z: depending on physical drive location. If the physical drive is located in the local system, then use the command “SUBST”. If the physical drive is located on another system on the network, then use the command “NET USE”. Examples of each method are given below:

To map virtual drive Z: to a local drive containing the MiniBIOS at C:\PlatformRC\MiniBIOS, use the following commands:

subst Z: /D

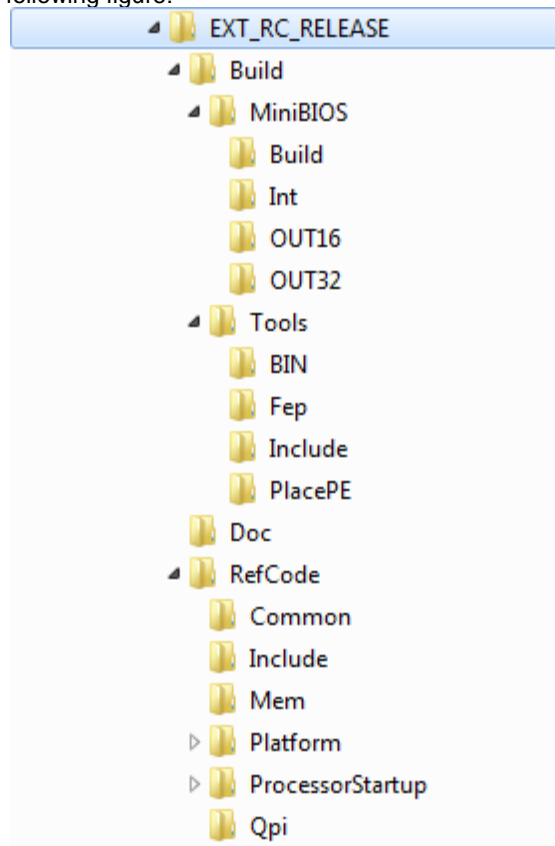
subst Z: C:\PlatformRC\MiniBIOS

To map virtual drive Z: to a network drive containing the MiniBIOS at \\SOXPLO525\c\$\PlatformRC\MiniBIOS, use the following commands:

net use Z: /DELETE

net use Z: \\SOXPLO525\c\$\PlatformRC\MiniBIOS

Given this assumption about the root directory, the subordinate MiniBIOS directory structure is illustrated in the following figure:



A brief description of each directory follows:

1. Build\MiniBIOS: batch and make files to build the MiniBIOS, final binary output files
2. Build\MiniBIOS\OUT16: intermediate 16-bit build output files
3. Build\MiniBIOS\OUT32: intermediate 32-bit build output files
4. Tools\BIN: pre-compiled executables of MiniBIOS tools
5. Tools\Fep: source code for the FindEntryPoint tool
6. Tools\Include: common header files for MiniBIOS tools
7. Tools\PlacePE: source code to the PlacePE tool



8. RefCode: reference code for the high-level flow of QPI and MRC steps, including multi-threaded execution. The following files in this directory are considered “sample code” because they are not used directly in Intel validation BIOS:
 - a. OemMemoryQpiInit.c – platform functions to wrap the reference code
 - b. OemMemoryQpiInit.h – platform header files to wrap the reference code
9. RefCode\Common: reference code to shared functions. This directory should not be modified by customers.
10. RefCode\Include: header files shared by the reference code. This directory should not be modified by customers.
11. RefCode\Mem: reference code to initialize memory. This directory should not be modified by customers.
12. RefCode\Platform\Hooks: platform functions that are called by the reference code. This directory should be modified by the customer as needed.
13. RefCode\Platform\Include: platform header files that are used by reference code. This directory should be modified by the customer as needed.
14. RefCode\Platform\Setup: platform default input parameters to the reference code. This directory should be modified by the customer as needed.
15. RefCode\ProcessorStartup: reference code for early processor initialization and basic chipset settings that are prerequisite for QPI and MRC. The following files in this directory are considered “sample code” because they are not used directly in Intel validation BIOS:
 - a. CpuPm32.asm – Switch to 32-bit protected mode and call ProcessorStartup module
 - b. CpuMain.c – Serial port init, reserve stack frame and call reference code module
 - c. CpuStart.asm – Call CpuMain, check stack usage, tear down NEM and stall
 - d. ProcessorStartupOem.asm – platform hooks for early init
 - e. ProcessorStartupPlatform.inc – platform build options for early init
16. RefCode\Cpu\Ucode: microcode patches loaded by ProcessorStartup module.
17. RefCode\Qpi: reference code for QPI initialization among all processors. This directory should not be modified by customers.
18. Doc*: this file and readme.txt files that track changes to the reference code.

2.5 Build Environment

This section describes the environment variables that are required to build the MiniBIOS.

1. Open a .NET DOS command box.
2. Run the VSVARS32.BAT file that is supplied with Microsoft Visual Studio .NET. This will modify the search path to run the compiler and linker.
3. “SET MASMPATH = <Path to Microsoft Segmented Executable Linker>”. This is required to link the object components into a binary executable.
4. Optionally “SET BIOS86DIR = <Path to the Microsoft Visual Studio .NET tools>\”. This option is useful if the .NET tools are not included on the search path.
5. Optionally “SET MINITOOLS = <Path to the H2INC utility>\”. This option is useful if the H2INC utility is not included on the search path.

2.6 Building the MiniBIOS

Change the working directory to: Z:\Build\MiniBIOS

To do a full build of the MiniBIOS image, execute the commands:



BuildMiniBIOS clean

BuildMiniBIOS

To do a partial build of the MiniBIOS, execute the commands:

BuildMiniBIOS

This generates the 1 MB MiniBIOS binary file Z:\Build\MiniBIOS\BIOS.BIN.

2.7 Debugging the MiniBIOS Using Intel SoftSimulation

This section provides a few quick tips for debugging the MiniBIOS image using the SoftSimulation environment. Please refer to the SoftSimulation User's Guide for more detailed instructions.

2.7.1 Configuring SoftSimulation to Load the BIOS image file

The name of the MiniBIOS binary file to be updated in simics file which is to be ran. Simics files are located at C:\Users\ldSid\simics-workspace\targets\Grantley. This file also will be used to configure system configuration information, including number of DIMMs, processor sockets, cores, threads, etc.

Please refer to simics release note for target script configuration.

In this example, the name of the MiniBIOS binary file is Z:\Build\MiniBIOS\BIOS.BIN.

2.7.2 Symbolic Debugging of MiniBIOS

Start the simulator by running the Simics executable. After the simulator completes loading, an ITP window will be visible to accept command input. Open the source code window to make sure that ITP is halted at the CPU reset vector. In the ITP command window, type the following commands:

```
go til 0xffff00640p; wait
load Z:\Build\MiniBIOS\out32\procstartupinit.bin nocode
load Z:\Build\MiniBIOS\out32\rcinit.bin nocode
```

The command "go til 0xffff00640p" instructs the simulator to execute until the physical address 0xffff00640 is reached. This address is the entry point, ProcessorStartup, of the PROCSTARTUPINIT.BIN module. It can be found by building the MiniBIOS and reading the file Z:\Build\MiniBIOS\out32\procstartupentry.inc. This entry point is also displayed on the screen at the end of the MiniBIOS build. It can change from build to build, so make sure you use the correct address in the command line.

The command "load Z:\Build\MiniBIOS\out32\procstartupinit.bin nocode" loads the symbolic debug information into the ITP source-level debug environment. After this command has executed, C and Assembler source code, as originally written by the BIOS author, may be viewed in the source code window and stepped through.

The command "load Z:\Build\MiniBIOS\out32\rcinit.bin nocode" loads the symbolic debug information into the ITP source-level debug environment. The other key entry point, MemoryQpilnit in the RCINIT.BIN module, can be found by reading the definition for RCINITENTRYPOINT in the file Z:\Build\MiniBIOS\out32\rcentry.h.

2.7.3 Using the DUMPBIN utility

DUMPBIN.EXE is a utility that comes with Microsoft Visual Studio .NET to dump information from PE/COFF binary file formats. A sample of the output is shown below:

```
Z:\Build\MiniBIOS\OUT32>dumpbin rcinit.bin /headers
Microsoft (R) COFF/PE Dumper Version 7.10.3077
Copyright (C) Microsoft Corporation. All rights reserved.
```

Dump of file rcinit.bin

PE signature found

File Type: DLL

FILE HEADER VALUES

```
14C machine (x86)
3 number of sections
0 time date stamp Wed Dec 31 19:00:00 1969
0 file pointer to symbol table
0 number of symbols
E0 size of optional header
2102 characteristics
```



Executable
32 bit word machine
DLL

OPTIONAL HEADER VALUES

10B magic # (PE32)
9.00 linker version
34340 size of code
8F40 size of initialized data
0 size of uninitialized data
336C0 entry point (FFF536C0)
240 base of code
34580 base of data
FFF20000 image base (FFF20000 to FFF5D4BF)
20 section alignment
20 file alignment
5.00 operating system version
0.00 image version
5.00 subsystem version
0 Win32 version
3D4C0 size of image
240 size of headers
0 checksum
3 subsystem (Windows CUI)
400 DLL characteristics
No safe exception handler
100000 size of stack reserve
1000 size of stack commit
100000 size of heap reserve
1000 size of heap commit
0 loader flags
10 number of directories
0 [0] RVA [size] of Export Directory
0 [0] RVA [size] of Import Directory
0 [0] RVA [size] of Resource Directory
0 [0] RVA [size] of Exception Directory
0 [0] RVA [size] of Certificates Directory
3C7A0 [C60] RVA [size] of Base Relocation Directory
36810 [1C] RVA [size] of Debug Directory
0 [0] RVA [size] of Architecture Directory
0 [0] RVA [size] of Global Pointer Directory
0 [0] RVA [size] of Thread Storage Directory
0 [0] RVA [size] of Load Configuration Directory
0 [0] RVA [size] of Bound Import Directory
0 [0] RVA [size] of Import Address Table Directory
0 [0] RVA [size] of Delay Import Directory
0 [0] RVA [size] of COM Descriptor Directory
0 [0] RVA [size] of Reserved Directory

SECTION HEADER #1

.text name
34321 virtual size
240 virtual address (FFF20240 to FFF54560)
34340 size of raw data
240 file pointer to raw data (00000240 to 0003457F)
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
60000020 flags
Code
Execute Read

SECTION HEADER #2

.data name
8217 virtual size
34580 virtual address (FFF54580 to FFF5C796)
8220 size of raw data
34580 file pointer to raw data (00034580 to 0003C79F)
0 file pointer to relocation table



0 file pointer to line numbers
0 number of relocations
0 number of line numbers
C0000040 flags
Initialized Data
Read Write

Debug Directories

Time	Type	Size	RVA	Pointer
4B845ABA	cv	3B	0003C75C	3C75C

Format: RSDS, {75E96270-0E9B-4592-9348-97AA4EDC851D}, 1, Z:\Build\MiniBIOS\OUT32\RCINIT.pdb

SECTION HEADER #3

.reloc name
D18 virtual size
3C7A0 virtual address (FFF5C7A0 to FFF5D4B7)
D20 size of raw data
3C7A0 file pointer to raw data (0003C7A0 to 0003D4BF)
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
42000040 flags
Initialized Data
Discardable
Read Only

Summary

8220 .data
D20 .reloc
34340 .text

The entry point of the PE/COFF executable is indicated in the line:

336C0 entry point (FFF536C0)

This address can be used in itp_ssdrv.cfg to automatically halt and load source level debug information. The above dump is only a sample and the actual address may differ.

3 Details of MiniBIOS

This section describes the MiniBIOS layout in detail. BIOS engineers should read this section carefully, as it explains how to build the reference code modules and integrate them into a customer BIOS.

3.1 MiniBIOS layout

The layout of the MiniBIOS is shown in the figure below. The sample CPUPM32.BIN module occupies the address range based at (4GB – 64KB). This area is what would be the customer's main BIOS. The reference code occupies the address range based at (4GB – 2MB), extending up to the size of the PE executable modules PROCSTARTUPINIT.BIN and RCINIT.BIN. The final output image BIOS.BIN is 2MB in size and top-aligned to (4GB – 2MB).

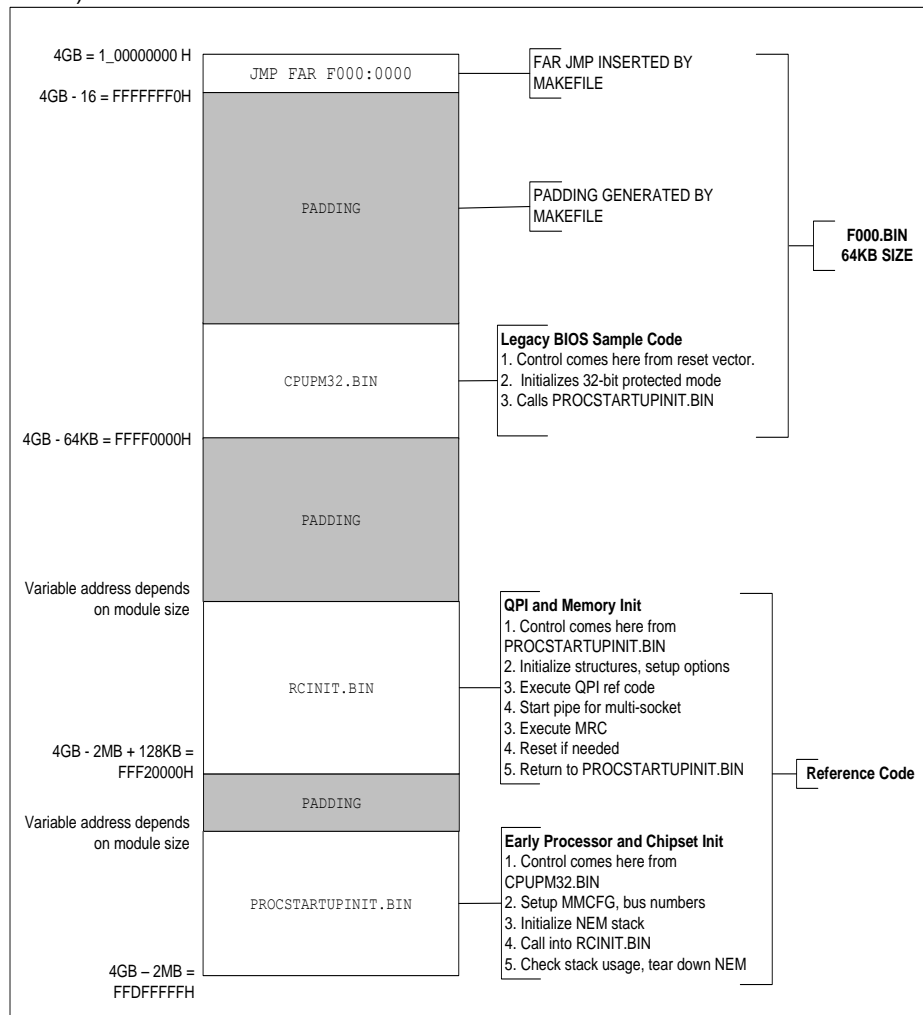


Figure 1: MiniBIOS Layout

3.2 Building the MiniBIOS

The MiniBIOS is built by executing the BuildMiniBIOS.bat file within the Z:\Build\MiniBIOS directory, which in turn launches nmake on the local makefile. The build occurs in the following sequence:

1. Compile and link reference code into the OUT32\RCINIT.DLL module. The entry point is *MemoryQpiInit*.

2. Run **PlacePE** to locate the RCINIT.BIN module at the 32-bit physical address RC_ROM_BASEADDRESS specified in BuildOptions.mak. The address must be aligned to a 16-byte boundary.
3. Run **FindEntryPoint** to get the entry point of the RCINIT.BIN module and save rcentry.h
4. Compile and link processor startup reference code into the OUT32\PROCSTARTUPINIT.DLL module. The entry point is *ProcessorStartup*.
5. Run **PlacePE** to locate the PROCSTARTUPINIT.BIN module at the 32-bit physical address PROCSTARTUP_ROM_BASEADDRESS specified in BuildOptions.mak. The address must be aligned to a 16-byte boundary.
6. Run **FindEntryPoint** to get the entry point of the PROCSTARTUPINIT.BIN module and save procstartupentry.inc.
7. Compile and link the sample code into the CPUPM32.BIN module. This would normally be linked with other OEM BIOS components to form the F000 ROM image.
8. Concatenate the binary files with padding to create the BIOS.BIN image. This file is the complete 1MB image that is shown in *Section 3.1 Mini-BIOS Layout*.

3.3 Execution Flow

This section describes the execution flow of the MiniBIOS as a sample implementation.

1. Control comes from the reset vector at 0xFFFFFFF0 to *_PM32Entry* in RefCode\ProcessorStartup\CpuPm32.asm.
2. Switch to 32-bit protected mode and call into the PROCSTARTUPINIT module. The entry point is *ProcessorStartup* in RefCode\ProcessorStartup\ProcessorStartup.asm.
3. Initialize MMCFG, bus numbers, early chipset settings, NEM stack and call into the RCINIT.BIN module. The entry point is *MemoryQpInit* in RefCode\MemoryQpInit.c
4. Initialize QPI input parameters, execute QPI reference code, and process output parameters.
5. Setup the pipe environment for multi-socket execution, initialize MRC input parameters, execute MRC, and process output parameters. A system reset will be issued if necessary.
6. After MRC completes, copy reference code output parameters to system memory. Disable the pipe handler, tear down NEM stack on the remote sockets and return to the PROCSTARTUPINIT module.
7. Check the stack usage, tear down NEM stack on the SBSP and stall execution with an infinite loop (JMP \$).

3.4 MiniBIOS Tools

This section describes the build tools that come with the reference code.

3.4.1 FindEntryPoint

This utility parses the header of the PE binary file and calculates the 32-bit physical address corresponding to the entry point into the binary. It accepts one optional parameter to specify the output language, /C for C code or /A for ASM code.

Syntax: FindEntryPoint / [C | A] infile

infile: PE file whose entry point is to be found.

Example:

findentrypoint /C out32\rcinit.bin

Output:

```
#define RCINITENTRYPOINT 0xffff536c0
```

3.4.2 PlacePE

This utility locates a PE binary file at a user-supplied 32-bit physical address where it will execute from. It has the ability to locate the executable at any byte-boundary inside 4GB address space, however the executable should normally be aligned to a 16-byte address boundary.

Syntax: placepe [infile] [outfile] [base address]

infile: PE file that is to be fixed up.



outfile: The fixed-up PE file that is generated.

base address: The 32-bit flash address where this code resides.

Example:

```
copy out32\rcinit.dll out32\rcinit.bin
```

```
Placepe out32\rcinit.bin out32\rcinit.bin 0xffff20000
```

Note: This utility requires that the infile be copied to the outfile prior to execution. The outfile will then be patched with appropriate data by this utility.

4 Common Reference Code Components

4.1 Overview

This section details information that is shared or referenced by multiple components.

4.2 BIOS Reference Code High Level Execution Flow

This section shows a high level initial cold-boot flow of the BIOS reference code components. Details are present in the following chapters

- CPU Initialization BIOS Reference Code / Early Initialization section.
- QPI Reference Code ("Phase 1": initializes registers that require a reset to latch, warm reset then runs "phase 2")
- Pipe Init
- Memory Reference Code ("Phase 1": typically exits with a frequency change on reset request).
- CPU Initialization BIOS Reference Code / Feature Early Configuration (sets registers that require a reset to take effect)
- Pipe Exit
- Check for Reset (Typically a reset is required after the first pass through the early BIOS), and issue reset if needed.

Below shows a high level BIOS reference code flow between the BIOS reference code components after software generated reset. Details are present in the following chapters

- CPU Initialization BIOS Reference Code / Early Initialization section.
- QPI Reference Code ("Phase 2")
- Pipe Init
- Memory Reference Code ("Phase 2": system memory initialization).
- CPU Initialization BIOS Reference Code / Feature Early Configuration
- Pipe Exit
- Check for Reset (not typically needed at this point in the flow)

4.3 Assumptions

The various BIOS reference code components assume that the system is initialized at roughly the same level as the miniBIOS and the components are called in the same order. For example, for the Memory Reference code to run multi-socket, it requires the "pipe" infrastructure to be initialized, and for the pipe to work, it requires the QPI links to be initialized at least to the point of Phase 1 of the QPI reference code, and phase 1 of the QPI reference code requires the CPU bus numbers and configuration space be initialized as in the early initialization section of the CPU BIOS reference code.

4.4 Platform Porting and Reference code Hooks

Other files in the Refcode\Platform subdirectories must also be ported for a given platform. There are also files in the RefCode directory and subdirectories should not be modified. By default, these files implement settings compatible with the Intel customer reference board and the Intel Platform Design Guide.

4.4.1 Options

Common Option fields.

4.4.1.1 PROMOTE_WARN_EN (BIT0)

When set, all warnings logged are treated as fatal errors.

4.4.1.2 PROMOTE_MRC_WARN_EN (BIT1)

When set, Memory Reference Code warnings are treated as errors independently of PROMOTE_WARN_EN.



4.4.1.3 HALT_ON_ERROR_EN (BIT2)

When set, all errors stop at a halt loop to allow for easier debug when an error occurs.

4.4.2 lowGap

This is a byte size field that contains the desired size of the low MMIO gap in units of 64MB. Low MMIO gap is defined by TOLM to 4 GB. Minimum size is 512MB assuming PCI configuration space MMCFG is below 4GB. This is a system address region right below 4GB typically used for MMIO configuration registers, memory resources allocated to PCI devices, and so on.

4.4.3 highGap

This is a byte size field that contains the desired size of the high MMIO gap in units of 64MB. Minimum size is 64 MB. HighGap is defined by (LIMIT + 10h - TOLM) to 64 GB. This option is included for systems with special needs and require such memory gap or hole be reserved above 4GB.

4.4.4 serialDebugMsgLvl

Specifies what level of debug messages will be sent to serial port. Available options are a bitfield where bit 0 set = minimal information, bit 1 set = more information (but longer execution time), and bit 3 set = a full trace of the BIOS configuration register accesses.

```
#define SDBG_MIN          BIT0
#define SDBG_MAX          BIT1
#define SDBG_TRACE        BIT2
#define SDBG_MEM_TRAIN    BIT3 + SDBG_MAX
#define SDBG_TST          BIT4
#define SDBG_MINMAX       SDBG_MIN + SDBG_MAX
```

4.4.5 bsdBreakpoint

Specifies the BIOS serial debug breakpoint. It is used to stall BIOS execution at a predefined checkpoint so that EV tools can be run in the proper context (before switching to normal mode and locking registers). Format is similar to port 80 postcode.

4.4.6 maxAddrMem

Maximum addressable memory supported by the platform. Haswell Processor supports up to 46-bit addressing. This input should be the total number of addressable bytes in 256MB units. (0x40000 for 46-bit and 0x1000 for 40-bit). 46-bit addressing is the default.

4.4.7 SocketType

Type of socket used. Socket R3 (= 0) for Platform.

4.4.8 debugPort

User configurable IO port for post code which is traditionally located at 0x80. 0x80 will be used if this field is 0.

4.4.9 nvramPtr

32-bit pointer to an optional OEM NVRAM image to be copied into the host NVRAM structure.

4.4.10 sysHostBufferPtr

32-bit pointer to an optional OEM provided Host structure.

4.4.11 mmCfgBase

Base address of low PCIEX MMIO in 256MB granularity.

4.4.12 mmCfgSize

Size of low PCIEX MMIO memory range. Must be 64MB (0x4000000), 128MB, or 256MB (default).

4.4.13 mmiolBase

Low MMIO base, 64M aligned.

4.4.14 mmiolSize

Low MMIO size, 64M aligned.

4.4.15 mmiohBase

The hex value of the address lines [45:32] where MMIO High resources should be placed.

4.4.16 mmiohSize

Number of 1GB contiguous regions to be assigned for MMIOH space per CPU.

4.4.17 pchUmaEn

UMA support for PCH.

4.4.18 numaEn

Set to 1 to enable NUMA, or 0 to disable (UMA).

4.4.19 isocEn

Set 1 to enable Isoc

4.4.20 DciVcm

Set 1 to enable DciVcm, or 0 to disable.

4.4.21 MeSegEn

Set 1 for MeSeg Support. Exclusive with IsochEn

4.4.22 dcaEn

Set to 1 to enable DCA, or 0 to disable.

4.4.23 PSMIEnabledSupport

Set 1 to enable PSMI.

4.4.24 BdatEn

Enable BDAT information for OS consumption

4.4.25 consoleComPort

Base address of the console ComPort if it's available in the system.

4.4.26 debugComPort

Base address of debug ComPort if it's available in the system.

4.4.27 usbDebugPortSetup

A structure with information about the platform USB Debug compliant port, if one is available in the system.



4.5 Scratch pad register usage

Haswell contains 8 sticky, 7 non-sticky lock bypass and 16 non-sticky scratchpad registers for BIOS usage. The reference code takes advantage of several of these registers which are documented here. Some of these registers have been set aside for future development and have been marked as “Intel Reserved”. All other registers are marked as available customer usage. Please refer to Scratch-Pad consumption documentation in Doc\ScratchPadList.xls

4.6 High and Low MMIO BAR allocation

QPIRC assigns High MMIO (above 4G) and Low MMIO (below 4G) for each socket. QPIRC chooses 0x3800_000_0000 as default address for High MMIO and 0x8000_0000 as default address for low MMIO. This start address can be changed by OEM to meet its platform needs.

The highest address supported by Haswell is 0x4000_0000_0000 (2⁴⁶). Address range 3800_0000_0000 to 4000_0000_0000 is available for OEM to assign for high MMIO space. High MMIO Base address for non-BSP will change based on size selected. To specify the size, OEM could simply change *mmiohSize* in *commonSetup* structure with increment of 1GB to any size limited to the number of sockets within the range between 3800_0000_0000 to 4000_0000_0000. For example, in QPIRC OEM can specify up to 1TB of high MMIO space for each socket in 8 socket glue-less configuration. Similarly OEM can specify up to 2TB of high MMIO space for each socket in 4 socket glue-less configuration.

Code snippet in reference code to change the size in the reference code is as below:

```
Filename: CommonSetup.c
Routine: GetSetupOptions
setup->common.mmiohSize = 2; // 2G per IO
```

Code snippet in QPIRC to change start address of High MMIO base is as below:

```
File: QpiLib.c
Routine: AllocateResources
//
// Initialize the Base value of the resources
//
BusBase = 0;
IoBase = 0;
IoApicBase = IOAPIC_BASE;
MmioLBase = host->setup.common.mmCfgBase + host->setup.common.mmCfgSize;
//100_0000_0000 = 1TB per socket for 8SG
MmiohBase.lo = 0;
MmiohBase.hi = 0x3800; // MMIOH_BASE;
```

After QPIRC allocates and divides MMIO resources across all sockets standard PCI enumeration code in platform codebase will assign these resources based on PCI configuration. Each end point device will request these resources through standard PCI BAR registers. In case platform BIOS uses EDKII core code PCI enumeration and MMIO resource allocation could be found in File “PciEnumatorSupport.c” Function name “PciParseBar”

5 CPU Initialization Reference Code

5.1 Overview

CPU init Reference Code contains source files in Intel x86 assembly and C languages. It exemplifies BIOS boot flow from the reset vector to the end of MRC, when system memory has been initialized and becomes available. It depends on QPI RC for CPU population topology discovery and employs a cold-boot/warm-boot 2-pass approach to complete the basic CPU, QPI and memory initialization.

5.2 CPU Reference Code Execution Flow

This section shows a high level flow of the CPU reference code and its relation to the QPI/Memory reference code as well as what is being done by the CPU reference code where applicable.

Early CPU Init

Run by all Package BSPs

- Switch to 32-bit protected mode
- Load microcode update
- Configure DCU_Mode per user input
- Enable No Eviction Mode (NEM)
- Config TARGET_LIST, CPUBUSNO and MMCFG for CSR access
- Select SBSP (legacy socket)
- Non-SBSPs check-in and loop on Scratchpad CSR

Invoke QPI RC to discover CPU population topology

- On cold-boot path, CPU population and final CPUBUSNO, MMCFG information is determined and stored in sticky SR, and reset request is raised.

PIPE Init

- Set up a mailbox mechanism by which SBSP (PIPE Master) can send command/data to other BSPs. This is intended for multi-threaded MRC execution.

CPU Feature Early Configuration

- Config FlexRatio, DesiredCores, SMT, etc that need a reset to take effect
- Run by SBSP only, via CSR access to all CPUs present
- Set CPU reset global flag if needed

Invoke MRC to initialize memory

PIPE Exit

- SBSP (PIPE Master) sends command via PIPE mailbox to other BSPs to exit PIPE. SBSP sends INIT-IPI to put all BSPs in Wait-For-SIPI state.

Check for Reset

- Consolidate/minimize potential multiple resets
- Check global flag for reset request by all sources
 - QPIRC
 - MRC
 - CPU Feature Early config
- Initiate the type of reset requested in the priority order below:
 - Power Good Reset (IO Port 0xCF9 = 0xE)
 - Warm Reset (IO Port 0xCF9 = 0x6)

5.3 Assumptions

The following assumptions are used by the CPU Ref Code:

- Max of 4 CPU sockets is validated on Intel platforms.
- Reference code will support up to 8 sockets.
- Reference Code has a limitation that MAX_CPU_SOCKETS must be either 4 or 8.
- 8 sockets support is build time decision.
- Legacy socket (with PCH connection via DMI link) is always present and designated as system BSP.
- Reserved usage of mmx0-7, xmm1-7 CPU registers in SEC phase
- Entire reference code needs to be cached into NEM code space. Early reset code before enabling NEM space is exception to this condition.

5.4 Platform BIOS Integration Guide

CPU init Reference Code provides platform/OEM customization provisions such as Build-time defines/equates, input parameters, and OEM code hooks. This chapter contains a summary of platform/OEM customization files and hooks related to CPU init reference code.

5.4.1 Build-Time Equates and Defines

Filename: **ProcessorStartupPlatform.inc**

The following equates define PCIe config base address and No Eviction Mode parameters:

```
PCIEX_BASE_ADDRESS EQU 080000000h
; MMCFG / PCIe Config Base Address, used as cold boot default only.
; Use EDKII_GLUE_PciExpressBaseAddress in PlatformHost.h file to
; define platform choice of PCIe config base address
```



```

CODE_REGION_BASE_ADDRESS      EQU 0ffe00000h
CODE_REGION_SIZE              EQU 0200000h
CODE_REGION_SIZE_MASK         EQU (NOT (CODE_REGION_SIZE - 1))
DATA_STACK_BASE_ADDRESS       EQU 0fe080000h
DATA_STACK_SIZE               EQU 080000h
DATA_STACK_SIZE_MASK          EQU (NOT (DATA_STACK_SIZE - 1))
MAX_CPU_SOCKETS               EQU      x
                               ; where default x = 4 for 4S support
                               ; to build 8S, change to 8

```

5.4.2 Input Parameters

CPU RC related input parameters are defined in file CpuHost.h, struct cpuSetup, as shown below:

```

struct cpuSetup {
    // CPU init related user options
    UINT8  dcuModeSelect;    // 0: 32KB 8-way (hardware default). Non-zero: 16KB 4-way with ECC (CPU MSR 031h)
    UINT8  flexRatioEn;      // FLEX_RATIO Override Enable. If non-zero then flexRatioNext will be used
    UINT8  flexRatioNext;    // Target FLEX_RATIO, common for all CPU sockets.
    UINT32 CoreDisableMask[MAX_SOCKET]; // CoreOffMask value for each CPU socket. If 0 then no CPU cores
                                         // will be disabled by BIOS
    UINT8  smtDisable;       // 0/1 Disable/Enable SMT(HT). common for all CPU sockets
    UINT8  vtEnable;         // 0/1 Disable/Enable VMX. Common for all CPU sockets
};

```

Other common input parameters that affect CPU RC are defined in SysHost.h, struct commonSetup

```

struct commonSetup {
    UINT32 sysHostBufferPtr; // address to copy sysHost structure to after MRC.
    UINT32 mmCfgBase;        // MMCFG Base address, must be 64MB aligned
    UINT32 mmCfgSize;        // MMCFG Size address, must be 64M, 128M or 256M
    UINT8  dcaEn;            // 1 - Enable; 0 - Disable
};

```

5.5 Routines for Platform Customization

Filename: **ProcessorStartupOem.asm**

Routine name: **oem_get_cpu_rom_setup_options PROC**

Returns: Build-time platform options related to CPU early init

- Called early after reset, before loading ucode
- Returns ucode region address/size, etc
- Sample implementation assumes ucode update files are included in the source code as .txt files (similar to Nhm reference code)

Routine name: **OemFindMicrocode PROC**

Returns: 32-bit linear pointer to the CPU ucode patch header

- Called early after reset, before loading ucode
- Sample implementation similar to Nhm reference code.

Routine name: **GetPlatformCpuSettings PROC**

Returns: Run-time platform options related to CPU early init

- Called early after reset, before NEM init
- Returns DCU_MODE select, which is usually determined by the input parameter setup->cpu.dcuModeSelect.
- Sample implementation based on intel ref code package implementation

Filename: **CommonSetup.c**

Function Name: **GetSetupOptions()**

- Provide common default setup options.
- Called after early power-on NEM init, before QPI RC
- mmcfgBase. This is the value for setting the final MMCFG(PCIe) config space base address on the warm boot path.
- maxBusNum can be 0xFF, 0x7F, or 0x3F

Filename: **OemMemoryQpiInit.c**



Function Name: **OemInitializePlatformData()**

- Provide OEM platform specific setup options and/or to override default input options.
- Called after early power-on NEM init, before QPI RC
- Examples include pointer to memory buffer where the sysHost structure will be copied to after memory has been initialized, CPU Flex Ratio, Desired_Cores, SMT enable/disable, DCU_Mode, etc.

Function Name: **OemMemoryInit()**

- OEM hook called after MRC execution
- Sample implementation copies the sysHost structure to a specified buffer in system memory (default address = 0x00100000).

5.6 POST Codes and Serial Port Debug Messages

5.6.1 POST Code at IO Port 0x80

Below is the POST code output to IO port 0x80.

Post Code (Hex)	Status	Error
01	Uncore BUS/MCFG configuration	
02	Local Microcode	
CD		Microcode Not found
03	Enable NEM mode	
D0		NEM Data Stack Test Failed

5.6.2 Serial Port Debug Messages

Below is the capture of a typical example of debug messages to serial port.

Emulation Value is: 1!

Running on SoftSDV

Revision: 32

QPI Init starting...

(QPI RC output messages)

....

....

completed!

Pipe Init starting...completed!

Memory Init starting...completed!

CPU Feature Early Config starting...completed!

Pipe Exit starting...completed!

Emulation Value is: 1!

Running on SoftSDV

Revision: 0

QPI Init starting...

(QPI RC output messages)

....

....

completed!

Pipe Init starting...completed!

Memory Init starting...

(Memory RC output messages)

....

.....

MRC is done!

sysHost structure @ FEFED18 has been copied to memory address 00100000, size = 000102A0

completed!



```
CPU Feature Early Config starting...completed!  
Pipe Exit starting...completed!  
Checking for Reset Requests ... None Continue with system BIOS POST...
```


6 QPI Reference Code

6.1 Overview

This Platform allows systems with variety of QPI topologies. The QPIRC code dynamically determines all aspects of the topology such as number of and type of sockets populated and the link that connects them together. However system aspects such as link electrical parameters are system specific. Also parameters such as Low MMIO region base and size are configurable for these platforms. So QPIRC provides hooks that can be modified to return the data that are system specific. It also provides hooks related to Error/Status code display, issuing reset, changing the input parameters and so on. This section provides the details of input parameters that can be changed by platform BIOS, output parameters produced by QPIRC and the porting needed to integrate QPIRC into platform BIOS code base.

6.2 Input Parameters

The QPIRC uses a set of input parameters to configure how it initializes the QPI fabric of the system. It initializes the input parameters with known good default values. It then calls an OEM hook to override those default values with platform specific values. Typically these values are stored in NVRAM and they are changeable thru BIOS setup menus. The format of the input parameters structure may vary with each QPIRC release. The individual BIOS developer must review this structure to ensure that all fields are properly initialized. The individual field definitions for QPIRC are given below. Some of the input parameters are not applicable to EP platforms but to enable code sharing with EX platforms, they retained in EP QPIRC. Please refer to QpiHost.h file for more info about these fields.

6.2.1 PPINrOptIn

Unused.

6.2.2 BusRatio/IoRatio/MmioIRatio

These parameters are specified by three byte arrays of [MAX_CPU_SOCKETS] to indicate the ratio of Bus/IO/MMIOL resources to be allocated for each socket. Any nonzero value for these parameters indicates the QPIRC to allocate resources for that socket. If resource is requested for a socket that is not currently populated, QPIRC will assume that the ratio is 0 for that socket and won't allocate any resources for it. If resource is not requested for a socket that is populated, QPIRC will force the ratio for that socket to 1. Maximum value allowed is 8 for Bus/Io resources; for MmioI, the maximum value allowed is (4G – MMIOLBase) / 64MB. Note that the size of BUS/MMIOL range is determined based on other input options provided to QPIRC. Size of Legacy IO is fixed at 64KB. For a given socket, multiplying the resource ratio with the chunk size provides the total resource size allocated to that socket. The resource for Bus, IO & MMIOL ranges is 32MB (32 buses), 8KB & 64MB, respectively.

For example, in a 4S system, user can select the IO ratio as 5:1:1:1 in which case, the 64KB IO region will be assigned as follows:

Socket 0: $5 * 8K = 40K$ (0x0000 - 0x9FFF)
Socket 1: $1 * 8K = 8K$ (0xA000 - 0xBFFF)
Socket 2: $1 * 8K = 8K$ (0xC000 - 0xDFFF)
Socket 3: $1 * 8K = 8K$ (0xE000 - 0xFFFF)

If equal distribution is not possible, socket with lowest socket ID will get additional chunks. For example, if the ratio is 4:1:1:1, then the resource allocation will be as follows:

Socket 0: 40K (0x0000 - 9FFF) ((4+1)*8K)
Socket 1: 8K (0xA000 - 0xBFFF)
Socket 2: 8K (0xC000 - 0xDFFF)
Socket 3: 8K (0xE000 - 0xFFFF)

6.2.3 LegacyVgaSoc

This option conveys which socket should be set as the target for legacy video range. Default is to set the legacy socket as the target for this range. This is based on socket ID. If the socket ID selected by this parameter is not actually populated, then legacy socket will be forced as Legacy VGA target.

6.2.4 MmioP2pDis

This option controls the P2P traffic across sockets. This doesn't affect P2P traffic whose source and target end points belong to the same IIO bus hierarchy. Value 0 will enable the P2P; Value 1 will disable the P2P. Other values are invalid.



6.2.5 IsocAzaliaVc1En

This option controls, when Azalia is enabled, if it uses VCp. Value 1 will use VCp for Azalia traffic; Value 0 will use VC1 for Azalia traffic; Value 2 will be treated as Auto in which case VCp will be used for all topologies that support ISOCH transactions.

6.2.6 DebugPrintLevel

Bit map identifying what level of debug message output is needed.

- Bit 0 - Fatal
- Bit 1 - Warning
- Bit 2 - Info Summary
- Bit 3 - Info detailed.

Value of 1 in a bit enables corresponding debug message level; Value of 0 disables it.

6.2.7 AltRtid2S

Unused

6.2.8 ClusterOnDieEn

This option enables the Cluster on Die feature. 1 to enable, 0 to disable. This only is only available in 1/2S EP/4S ring topologies (4S ring topology is restricted to BDX-EP-).

6.2.9 MctpEn

Enable MCTP feature. 1 to enable, 0 to disable. Default is disable.

6.2.10 E2EParityEn

This option enables end to end parity. 1 to enable, 0 to disable. Default is disable.

6.2.11 EarlySnoopEn

This option enables early snoop. 1 to enable, 0 to disable, default is disable.

6.2.12 DegradePrecedence

This option uses DEGRADE_PRECEDENCE definition. TOPOLOGY_PRECEDENCE is default

6.2.13 QpiLinkSpeedMode

This option selects Slow vs Full Speed mode for all QPI links in the system. A value of 0 indicates to remain in slow mode. A value of 1 selects to transition to full speed mode.

6.2.14 QpiLinkSpeed

This option selects the desired speed of all QPI links in the system in Full Speed mode. This option is system wide. A value of 1 selects 6.4GT/s. A value of 3 selects 8.0GT/s, and a value of 5 selects 9.6GT/s. To auto select the fastest mode, input a value of 6. QpiHost.h provides equates that can be used such as SPEED_REC_80GT and MAX_QPI_LINK_SPEED.

6.2.15 QpiLinkL0pEn

To enable L0p feature which cuts the link width in half when bandwidth utilization drops below a configurable low threshold. Default value is 0 (disable).

6.2.16 QpiLinkL1En

To enable L1 feature which is similar to L0s, but also shuts down the clock lane, saves more power and takes longer to restart. Default value is 1 (enable).

6.2.17 QpiLbEn

To enable QPI link loopback mode. Value 0 is disable (default), 1 is enable.

6.2.18 lioUniphyDisable

Byte array of [MAX_CPU_SOCKETS]. This option is valid only for non-legacy socket. It is used to clock gate PCIe ports and IIO logic on platforms that don't have the slots implemented on any of the CPU sockets. If the platform doesn't have PCIe ports implemented but does use the IIO VPP for memory hotplug then it should be set to 2; otherwise it should be set to 1.

6.2.19 QpiLinkCreditReduce

To reduced the number of link layer VNA credits for system and per link. Should be used only for debug.

6.2.20 QpiConfigTxWci

Unused

6.2.21 Qpi AdVNACreditArbThreshold

To program R3QPI credit merging configuration field.

6.2.22 QpiCrcMode

Unused.

6.2.23 QpiCpuSktHotPlugEn

Indicates if Skt is hot plug enabled. 0 is disable (default), 1 is enable.

6.2.24 QpiCpuSktHotPlugTopology

0 is 4S topology, 1 is for 8S topology

6.2.25 QpiSkuMismatchCheck

Option to check Sku mismatch. 0 is No, 1 is yes (default)

6.2.26 PhyLinkPerPortSetting

Data structure of [MAX_CPU_SOCKETS]. Except QpiLinkSpeed, all members of the structure provide input parameters per link for debug purposes. QpiLinkSpeed, is used to configure the per link speed setting. Note that link 0 & 1 must be operating at the same speed while link 2 can have different speed. The following fields are part of the structure:

- **QpiPortDisable:** To disable or enable this port.
- **QpiLinkCreditReduce :** To force reduced link credit operation, set a value of 1 in the QpiLinkCreditReduce field.
- **QpiProbeType :** To specify a midbus type probe, set a value of 1. To specify an interposer type probe, set a value of 2. Zero indicates no probe installed. QpiHost.h provides equates that can be used such as PROBE_TYPE_NO_PROBE.
- **QpiConfigTxWci :** To configure what are the link width modes the link is allowed to operate.

6.2.27 Input Parameter Default Values

The following table is the default value for each QPIRC input option.

Table 1-1. Default Values

Option	Defaults
BusRatio/IoRatio/MmioIRatio	1
LegacyVgaSoc	0
MmioP2pDis	0
IsocAzaliaVc1En	0
DebugPrintLevel	0xF
AltRtid2S	0
QpiLinkSpeedMode	1
QpiLinkSpeed	1
ClusterOnDieEn	0
QpiLinkL0pEn	0



QpiLinkL1En	1
QpiL0rEn	1
QpiLbEn	0
IioUniphyDisable	0
QpiLinkCreditReduce	0
QpiConfigTxWci	0
QpiAdVNACreditArbThreshold	1
QpiCrcMode	0
QpiAdaptationEn	0
QpiAdaptationInParallel	0
QpiAdaptationSpeed	1
QpiCpuSktHotPlugEn	0
QpiCpuSktHotPlugTopology	0
QpiPortDisable	0
QpiLinkCreditReduce	0
QpiLinkSpeed	1
QpiProbeType	0
QpiConfigdTxWci	0

6.3 Output Parameters

Output parameters describe various aspects of the system QPI fabric. Some of the parameters are system wide and some are per socket/link.

6.3.1 CpuInfo

This is an array of MAX_CPU_SOCKETS which is configurable at build time. The array is indexed by socket id. It provides information about the system such as the sockets populated in the system, how they are connected to each other expressed in terms of LEP, the routes between sockets expressed in terms of topology tree and various resources allocated to each socket. The information provided will be sufficient enough to construct ACPI tables (such as SLIT), allocating the MBAR address for IOAPICs and allocating Bus/IO/MMIOL resources for each PCIe device in the system.

The following fields are part of this structure:

- **Valid:** Indicates if this socket is populated or not
- **SocId:** Socket id
- **LepInfo:** This is an array of MAX_QPI_PORTS which is 3 for these platforms. It contains the Link Exchange Parameters of the QPI links. The following fields are part of this structure:
 - **Valid:** Indicates if the link is operational, slow/full speed
 - **PeerSocId:** Socket Id of the peer socket
 - **PeerSocType:** Socket type of the peer socket (only SOCKET_TYPE_CPU is supported)
 - **PeerPort:** Port of the peer socket to which this link is connected
 - **DualLink:** Indicates if this link is part of pair of links connecting two sockets
- **TopologyInfo:** Describes the QPI topology of the system from this socket's perspective. Array element 0 always contains info about this socket (i.e root node) itself.
 - **Valid:** Indicates if this array element has valid information. An invalid array element indicates the termination of the topology tree
 - **SocId:** Socket id of this node
 - **SocType:** Socket type of this node
 - **ParentPort:** Port that connects this node to its parent node
 - **Hop:** Number of nodes between this node and root node including self
 - **ParentIndex:** Array index of this node's parent in the topology tree
- **CpuRes:** It contains the following fields that describe the system resource range assigned for each socket.
 - **BusBase, BusLimit:** Bus range decoded by this socket
 - **IoBase, IoLimit:** IO range decoded by this socket
 - **IoApicBase, IoApicLimit:** IOAPIC range decoded by this socket
 - **MmiolBase, MmiolLimit:** Low MMIO range decoded by this socket
 - **MmiohBase, MmiohLimit:** High MMIO range decoded by this socket

6.3.2 SysConfig

Indicates the QPI topology detected for the system. SYS_CONFIG structure in QpiSi.h defines all the possible values for this field.

6.3.3 WarningLog

Various warnings encountered during QPI initialization. Refer to “Warning Logs” section for more information.

6.3.4 OutLegacyVgaSoc

Indicates the socket ID which is configured to be the target of the legacy VGA region.

6.3.5 OutSocEn

This field indicates if value selected for **IsocEn** in sysHost Input structure by the platform BIOS is overridden or not. QPIRC will override an input option if the QPI topology doesn't allow the value selected for an option.

6.3.6 OutMesegEn

This field indicates if value selected for **MeSegEn** in sysHost Input structure by the platform BIOS is overridden or not. QPIRC will override an input option if the QPI topology doesn't allow the value selected for an option.

6.3.7 OutSocAzaliaVc1En

This field indicates if value selected for **IsocAzaliaVc1En** in QPIRC Input structure by the platform BIOS is overridden or not. QPIRC will override an input option if the QPI topology doesn't allow the value selected for an option.

6.3.8 OutClusterOnDieEn

This field indicates if the Cluster On Die feature is enabled.

6.3.9 OutMctpEn

This field indicates if the MCTP feature is enabled.

6.3.10 OutE2EParityEn

This field indicates if the end to end parity feature is enabled.

6.3.11 OutEarlySnoopEn

This field indicates if early snooping is enabled.

6.3.12 QpiCurrentLinkSpeedMode

This field indicates if the system is in Slow or Full speed operation.

6.3.13 OutQpiLinkSpeed

This field indicates link speed per link.

6.3.14 OutQpiLinkL0pEn

This field indicates if L0p is enabled or not.

6.3.15 OutQpiLinkL1En

This field indicates if L1 is enabled or not.

6.3.16 OutQpiLinkL0rEn

This field indicates if L0r is enabled or not.



6.3.17 OutlioUniphyDisable

This field indicates sockets for which the Uniphy is disabled.

6.3.18 OutQpiCrcMode

This field indicates CRC mode configured for the system.

6.3.19 RasInProgress

Set to TRUE when the QPIRC is called to handle a CPU/IOH event

6.3.20 RasEvtType

Indicates RAS event type. 0 is Online, 1 is Offline

6.3.21 RasSocId

Indicates RAS event on socket ID

6.3.22 ProgramNonBC

Flag to indicate if the Non BC RTA entries to be programmed

6.3.23 ProgramBC

Flag to indicate if the BC RTA entries to be programmed

6.4 Error/Status Codes

Output parameters describe various aspects of the system QPI fabric. Some of the parameters are system wide and some are per socket/link.

- [23:20] - Bitmask of CPU Sockets affected, 0xF - System Wide Error/Status
- [19:18] - Socket Type. 0: CPU Socket, 0x3 System Wide Error/Status
- [17:16] - Bit mask of QPI Links affected, 0 - Link N/A, 0x3 - System Wide Error/Status
- [15:08] - Error/Status Minor Code
- [07:00] - Error/Status Major Code

Each Major error code also has a Minor error code that describes the problem with more level of accuracy. Error/Status Major Code is also passed to OEM code by calling "OemCheckPoint" routine. This is typically display as POST code on IO port 0x80. All the errors are fatal and some of the errors might be transient. When these errors occur, the system will be reset.

Table 1-2. Major/Minor Error Codes

Major Error Code	Minor Error Code	Details
ERR_BOOT_MODE (0xD8)	MINOR_ERR_UNSUPPORTED_BOOT_MODE (0x01) MINOR_ERR_PBSP_CHKIN_FAILURE (0x02)	When system BSP tries to setup path for remote sockets or sends a Boot_Go command to remote socket In SetupSbspPathToAllSockets() or SyncUpPbspForReset(). If the remote socket(s) hasn't checked-in, assert; it is a fatal condition, this error will be logged. No retry. <i>RC Behavior: System Halt</i>
ERR_MINIMUM_PATH_SETUP (0xD9)	MINOR_ERR_ADD_SOCKET_TO_TOPOLOGY_TREE (0x01) MINOR_ERR_TOPOLOGY_TREE (0x02) MINOR_ERR_INTERNAL_DATA_STRUCTURE (0x03) MINOR_ERR_NONCPU_PARENT_NODE (0x04) MINOR_ERR_INVALID_PORT_CONNECTION (0x05)	N/A N/A When SBSP tries to add this remote socket into system topology tree in SetupSbspPathToAllSockets(), there are some errors occur in the data structure. No retry. <i>RC Behavior: The current Socket is not added to the tree.</i>

		N/A When SBSP setups the boot path for the parent which is not directly connected to Legacy CPU in SetupSbspPathToAllSockets(). The Child is not an immediate neighbor of Parent. No retry.
ERR_TOPOLOGY_DISCOVERY (0xDA)	MINOR_ERR_ADD_SOCKET_TO_TOPOLOGY_TREE (0x01) MINOR_ERR_TOPOLOGY_TREE (0x02) MINOR_ERR_INTERNAL_DATA_STRUCTURE (0x03) MINOR_ERR_NONCPU_PARENT_NODE (0x04) MINOR_ERR_INVALID_PORT_CONNECTION (0x05)	N/A
ERR_SAD_SETUP (0xDB)	MINOR_ERR_INSUFFICIENT_RESOURCE_SPACE (0x01)	N/A <i>RC Behavior: System Halt</i>
ERR_UNSUPPORTED_TOPOLOGY (0xDC)	MINOR_ERR_INVALID_SOCKET_TYPE (0x01) MINOR_ERR_INVALID_CPU_SOCKET_ID (0x02) MINOR_ERR_CBO_COUNT_MISMATCH (0x03) MINOR_ERR_HA_COUNT_MISMATCH (0x04) MINOR_ERR_R3QPI_COUNT_MISMATCH (0x05) MINOR_ERR_SKU_MISMATCH (0x06) MINOR_ERR_LEGACY_PCH_MISMATCH (0x07) MINOR_ERR_SBO_COUNT_MISMATCH (0x08) MINOR_ERR_INVALID_TOPOLOGY (0x09)	N/A <i>RC Behavior: System Halt</i>
ERR_FULL_SPEED_TRANSITION (0xDD)	MINOR_ERR_LINK_SPEED_UNSUPPORTED (0x01) MINOR_ERR_MAX_CLK_RATIO_UNSUPPORTED (0x02) MINOR_ERR_QPI_ELEC_PARAM_NOT_FOUND (0x03)	N/A SBSP cannot find QPI TXEQ Parameters for this link in GetSocketLinkEparams(). No retry. <i>RC Behavior: System Halt</i>
ERR_S3_RESUME (0xDE)	MINOR_ERR_S3_RESUME_TOPOLOGY_MISMATCH (0x01)	
ERR_GENERAL_FAILURE (0xDF)	MINOR_ERR_GENERAL (0x01) MINOR_ERR_TIMEOUT (0x02)	

Table 1-3. Major/Minor Status Codes

Major Status Code	Minor Status Code
STS_DATA_STRUCTURE_INIT (0xA0)	N/A
STS_COLLECT_EARLY_SYSTEM_INFO (0xA1)	N/A
STS_OPEN_S BSP_CONFIG_ACCESS (0xA2)	N/A
STS_SETUP_MINIMUM_PATH (0xA3)	MINOR_STS_ADD_SOCKET_TO_MIN_PATH_TREE (0x01) MINOR_STS_COLLECT_LEP (0x02) MINOR_STS_CHK_PBSP_CHKIN (0x03) MINOR_STS_SET_PBSP_BOOT_PATH (0x04) MINOR_STS_SET_S BSP_CONFIG_PATH (0x05)
STS_OPEN_PBSP_CONFIG_ACCESS (0xA4)	N/A
STS_EARLY_RESET_CONFIG (0xA5)	N/A
STS_PBSP_SYNC_UP (0xA6)	MINOR_STS_ISSUE_BOOT_GO (0x01)
STS_TOPOLOGY_DISCOVERY (0xA7)	MINOR_STS_CHK_TOPOLOGY (0x01) MINOR_STS_DETECT_RING (0x02) MINOR_STS_CONSTRUCT_TOPOLOGY_TREE (0x03) MINOR_STS_CALCULATE_ROUTE (0x04)
STS_PROGRAM_FINAL_ROUTE (0xA8)	N/A
STS_PROGRAM_FINAL_IO_SAD (0xA9)	MINOR_STS_ALLOCATE_CPU_RESOURCE (0x01) MINOR_STS_FILL_SAD_TGTLST (0x02) MINOR_STS_PROGRAM_CPU_SAD_ENTRIES (0x03)
STS_PROTOCOL_LAYER_SETTING (0xAA)	MINOR_STS_PROGRAM_RTID (0x01) MINOR_STS_PROGRAM_RING_CRDT (0x02) MINOR_STS_PROGRAM_HA_CRDT (0x03) MINOR_STS_PROGRAM_HTRBT (0x04) MINOR_STS_QPI_MISC_SETUP (0x05)
STS_FULL_SPEED_TRANSITION (0xAB)	N/A



STS_PHY_LAYER_SETTING (0xAC)	N/A
STS_LINK_LAYER_SETTING (0xAD)	N/A
STS_SYSTEM_COHERENCY_SETUP (0xAE)	N/A
STS_QPI_COMPLETE (0xAF)	MINOR_STS_ABOUT_TO_RESET (0x42)

6.5 Warning Logs

QPIRC logs four-byte wide warning data in the output structure. The format of the warning is as follows:

- [15:12] - Bitmask of CPU Sockets affected, 0xF - System wide Warning
- [11:10] - Socket Type. 0 - CPU Socket, 0x3 - System Wide Warning
- [09:08] - Bit mask of QPI Links affected. 0x3 - System Wide Warning, 0 - Link N/A
- [07:00] - Warning Code

The warning log size in the output structure is pre-determined. So whenever the log overflows, the last entry of the warning log will indicate if there is any overflow.

Table 1-4. Warning Codes

Warning Codes	Details
WARN_LINK_SLOW_SPEED_MODE (0x01)	After warm-reset, QPIRC will check the link speed and this warning message will be logged if the system is operating in slow speed mode in QpiPhyLinkAfterWarmReset()
WARN_UNSUPPORTED_LINK_SPEED (0x02)	If requested speed is unsupported, this warning message will be logged. In SelectSupportedLinkSpeed().
WARN_LINK_FAILURE (0x03)	N/A
WARN_PER_LINK_OPTION_MISMATCH (0x04)	Normalize per link options. If the peers don't match their options, they will be normalized and this warning message will be logged. In NormalizeLinkOptions()
WARN_MMIOH_BASE_UNSUPPORTED (0x09)	N/A
WARN_MMIOH_SIZE_UNSUPPORTED (0x0A)	If user sets MMIOH size requested is 0, QPIRC force to 2G and this warning will be logged in AllocatelioResources()
WARN_RESOURCE_NOT_REQUESTED_FOR_CPU (0x0C)	If user selects 0 for IO and MMIOH resource ratio on legacy socket, this warning will be logged in CalculateResourceRatio()
WARN_RESOURCE_REQUEST_NOT_MET (0x0D)	Log a warning if the platform requested resource is more than what is available in CalculateResourceRatio().
WARN_VGA_TARGET_SOC_NOT_PRESENT (0x0E)	If legacy VGA socket is out of the max socket range or not a valid socket, this warning message will be logged in PrimeHostStructure().

6.6 Porting Guidelines

The following files need to be ported to use the QPIRC in an OEM BIOS code base:

6.6.1 QpiHooks.c

These following routines require porting:

- o OemDebugPrintQpi(): Simple printf() style implementation that can print different levels of debug messages.
- o OemWaitTimeForPSBP(): Can be used to specify a max wait time for the PSBP to check in.
- o OemQpiGetEparams(): This function must be ported to return the full-speed electrical parameters of the link to the calling routine. A simple example is provided in the reference code, but can be expanded to detect and support multiple platforms. This enables a single QPIRC binary to support multiple platforms.
- o OemQpiGetHalfSpeedEParams() This function must be ported to return the half-speed electrical parameters of the link to the calling routine. A simple example is provided in the reference code, but can be expanded to detect and support multiple platforms. This enables a single QPIRC binary to support multiple platforms.
- o OemCheckCpuPartsChangeSwap(): This function should detect if the CPU parts have changed or swapped for the platform since last boot. Used to avoid rerunning adaptation if the parts parts have not changed/swapped.
- o OemGetAdaptedEqSettings(): This function should return the adapted EQ values found for the platform.

6.6.2 QpiPlatformEparam.c

This file contains all the electrical parameters for all the sockets, links, and speeds. These values typically come from the board designers.

7 Memory Reference Code

Please see ConvergedCoreMRC.pdf



8 Compatible Data Structure

8.1 Requirements

Intel BIOS reference code implements system validation features that produce significant amounts of data. Customers and suppliers need a compatible method to access and parse this data with generic tools. The access method must support native access from applications running on the target system as well as remote access via ITP (or serial connection) from applications running on a host system. The data should be accessible from the time it is produced throughout the BIOS boot flow and into the OS. To maintain compatibility with future platforms, a pointer to the data structure must be provided via standard mechanisms defined in the Advanced Configuration and Power Interface (ACPI) specification.

The data structure format must be specified and associated with a unique version number such that non-BIOS applications can discover the format and maintain backward compatibility with old revisions. Forward compatibility is not a requirement, but a secondary version number can denote when fields have been appended to the end of the compatible structure. The data structure must support a reliable mechanism to verify data integrity, such as a Cyclic Redundancy Check (CRC) algorithm.

8.2 Proposed Solution

Intel BIOS reference code shall define a compatible data structure using the C programming language and compiler settings for Intel® IA32 Architecture. The BIOS data structure shall consist of three sections: a compatible header, a versioned data range, and an optional OEM data range. The compatible header shall contain the following information: an 8-byte signature string, a total structure size, a 16-bit CRC covering the structure size, primary and secondary versions, and an optional OEM offset.

The versions shall uniquely define the format of the data range such that an application can cast the memory range with the associated C structure and decode the data fields. The secondary version number shall be incremented when data fields are appended to the previous version of the data structure. The versions do not apply to the OEM data range.

BIOS reference code shall reserve a memory range on the System BSP stack while operating from cache in No-Eviction Mode (NEM). The memory range shall be large enough to store the compatible data structure. The BIOS reference code shall initialize fields in the data structure, update the CRC value, and store a pointer to the data structure in a scratchpad register in the processor. To determine BIOS execution context, a remote host application can optionally read the current BIOS checkpoint from a separate scratchpad register. The meaning of checkpoints on a given platform shall be consistent within BIOS reference code, but checkpoints may vary across different System BIOS implementations and different platforms.

After BIOS reference code completes, the System BIOS shall copy the data structure from the NEM stack to an intermediate system memory location and update the pointer in the scratchpad register. The data structure on the NEM stack will be invalidated by tearing down the NEM stack. The System BIOS shall reserve the intermediate memory range using the POST Memory Manager, EFI memory management services, and internal mechanisms so that subsequent steps during BIOS POST do not corrupt the data structure. If the System BIOS needs to update fields within the data structure, it shall be responsible for recalculating the CRC after the updates. The data structure must be relocated to different memory addresses, so pointers to fields within the data structure should be avoided. Instead, offsets can be used relative to the base address of the data structure. External pointers should also be defined as offsets of type UINT32 or UINT64 (relative to base address 0).

When the system BIOS establishes the final system memory map and location of the ACPI memory regions, the data structure shall be copied from the intermediate memory location to its final resting place in AddressRangeReserved, an ACPI Type 2 memory region below 4 GB. The system BIOS must reserve a memory region that is large enough to accommodate the size of the BIOS data structure rounded up to the next 4 KB page size and aligned to a 4 KB address boundary. The system BIOS shall report the physical address range of the Type 2 memory region as required by the ACPI specification. This includes software Interrupt 15h - function AX = E820h, or EFI GetMemoryMap if applicable.

The system BIOS shall initialize a custom ACPI table containing a pointer to the physical base address of the BIOS data structure within the Type 2 memory region. The system BIOS shall then update the scratchpad register with the physical address of the BIOS data structure and free the intermediate memory location. All subsequent accesses to the BIOS data structure should be directed to the final runtime location in the ACPI Type 2 region.

8.3 Compatible ACPI Table

The pointer to the BIOS data structure shall be defined in a custom ACPI table to provide compatibility across multiple platforms. The Root System Description Table (RSDT) shall reference a custom OEM table identified by the unique signature "BDAT". The BDAT table shall conform to the standard ACPI header and contain a Global Address Structure that defines the 64-bit physical base address of the BIOS data structure. An OS driver may be required to access the custom ACPI table and to load pages containing the BIOS data structure.

The following C code is a sample implementation of the BDAT table based on the EFI Developer Kit. The BdatGas field and the table checksum must be updated at boot time based on the address of the BIOS data structure.



```
#pragma pack(1)

typedef unsigned char      UINT8;
typedef unsigned short    UINT16;
typedef unsigned long     UINT32;
typedef unsigned long long UINT64;

#define EFI_SIGNATURE_16(A, B)      ((A) | (B << 8))
#define EFI_SIGNATURE_32(A, B, C, D) (EFI_SIGNATURE_16 (A, B) | (EFI_SIGNATURE_16 (C, D) << 16))

//
// Common ACPI description table header.  This structure prefaces most ACPI tables.
//
typedef struct {
    UINT32  Signature;
    UINT32  Length;
    UINT8   Revision;
    UINT8   Checksum;
    UINT8   OemId[6];
    UINT64  OemTableId;
    UINT32  OemRevision;
    UINT32  CreatorId;
    UINT32  CreatorRevision;
} EFI_ACPI_DESCRIPTION_HEADER;

//
// ACPI 3.0 Generic Address Space definition
//
typedef struct {
    UINT8   AddressSpaceId;
    UINT8   RegisterBitWidth;
    UINT8   RegisterBitOffset;
    UINT8   AccessSize;
    UINT64  Address;
} EFI_ACPI_3_0_GENERIC_ADDRESS_STRUCTURE;

//
// BIOS Data ACPI structure
//
typedef struct {

    EFI_ACPI_DESCRIPTION_HEADER      Header;
    EFI_ACPI_3_0_GENERIC_ADDRESS_STRUCTURE BdatGas;

} EFI_BDAT_ACPI_DESCRIPTION_TABLE;

//
// BIOS Data Parameter Region Generic Address
// Information
//
#define EFI_BDAT_ACPI_POINTER          0x0

//
// BIOS Data Table
//
EFI_BDAT_ACPI_DESCRIPTION_TABLE BiosDataTable = {
    EFI_SIGNATURE_32('B','D','A','T'), // Signature
    sizeof (EFI_BDAT_ACPI_DESCRIPTION_TABLE), // Length
    0x01, // Revision [01]
    //
    // Checksum will be updated during boot
    //
    0, // Checksum
    ' ', // OEM ID
    ' ',
    ' ',
    ' ',
    ' ',
    ' ',
    ' ',
    ' ',
    0, // OEM Table ID
}
```



```

0, // OEM Revision [0x00000000]
0, // Creator ID
0, // Creator Revision
0, // System Memory Address Space ID
0,
0,
0,
0,

// Pointer will be updated during boot
EFI_BDAT_ACPI_POINTER,
};

#pragma pack()

```

8.4 Scratchpad

A scratchpad configuration register shall be reserved to facilitate usage of the BIOS data structure during boot time and by remote host applications. The scratchpad register will contain a 32-bit physical address pointer to the BIOS data structure and must be qualified by the current BIOS checkpoint. The BIOS checkpoint will be located in a different scratchpad register. A NULL pointer of 0 is not valid and must not be de-referenced.

The scratchpad registers will be located in the System BSP Uncore domain as defined in Section 4.5. BDAT feature uses BIOSNonStickyScratchpad5 and BIOSNonStickyScratchpad7.

8.5 Compatible BIOS Structure Header

The BIOS data structure shall begin with a compatible header so that an application can determine the remaining structure format and check the data integrity. The header shall contain the following information: an 8-byte signature string, a total structure size, a 16-bit CRC, primary and secondary versions, and an optional OEM offset.

The signature string shall be initialized to the ASCII sequence "BDATHEAD". The CRC shall be calculated over the specified size of the BIOS data structure, assuming that the CRC field itself contains a value of 0. The 16-bit CRC algorithm shall be compatible with the JEDEC DDR3 Serial Presence Detect (SPD) specification for bytes 126 – 127.

The primary and secondary versions shall uniquely define the format of the data range such that an application can cast the memory range with the associated C structure and decode the data fields. The secondary version number shall be incremented when data fields are appended to the previous version of the data structure. The secondary version number can be recycled when the primary version changes.

The OEM offset provides an optional mechanism for OEMs to customize the BIOS data structure without affecting compatibility of the versioned data range. The version numbers do not apply to the OEM data range, although fields in the versioned data range can be initialized by the OEM system BIOS. The OEM offset is provided mainly as a courtesy for customers that wish to use the BIOS data structure mechanism to transfer information to an OS driver. The format of the OEM data range is outside the scope of this specification.

The header may not be aligned during BIOS use but will be aligned to a 4 KB page boundary when relocated in Type 2 memory for OS use. The header format shall not change across different platform generations. The following data structure defines the compatible header.

```

#pragma pack(1)

typedef struct {
    UINT8    BiosDataSignature[8]; // "BDATHEAD"
    UINT32    BiosDataStructSize; // sizeof BDAT_STRUCTURE
    UINT16    Crc16; // 16-bit CRC of BDAT_STRUCTURE (calculated with 0 in thisfield)
    UINT16    Reserved
    UINT16    PrimaryVersion; // Primary version
    UINT16    SecondaryVersion; // Secondary version
    UINT32    OemOffset; // Optional offset to OEM-defined structure

    UINT32    Reserved1;
    UINT32    Reserved2;
} BDAT_HEADER_STRUCTURE;

#pragma pack()

```



8.6 BIOS Structure Definitions

The definition of the compatible BIOS data structure shall be managed by the BIOS reference code team. Any changes to the data structure identified by the primary and secondary version numbers must be approved by the BIOS reference code team and the external tools team. The version numbers can be combined into a shorthand notation of “primary.secondary” version when referencing the BIOS data structure. The following code defines a preliminary version 2.0 data structure, `BDAT_SYSTEM_STRUCTURE`. Please refer to *BDAT.H* in the Reference Code for latest structure definition.

Version 2.0: The following code defines version 2.0 of the `BDAT_SYSTEM_STRUCTURE`.

```
#pragma pack(1)

#define MC_MAX_NODE          8          // Max memory nodes per system
#define MAX_CH               4          // Max channels per memory node
#define MAX_DIMM             3          // Max DIMM per channel
#define MAX_RANK_DIMM       4          // Max ranks per DIMM
#define MAX_STROBE           18         // Number of strobe groups
#define MAX_SPD_BYTE         256        // Number of bytes in Serial EEPROM

typedef struct {
    UINT8  rxDqLeft;           // Units = 1/64 QCLK
    UINT8  rxDqRight;
    UINT8  txDqLeft;
    UINT8  txDqRight;
    UINT8  cmdLeft;
    UINT8  cmdRight;
    UINT8  rxVrefLow;
    UINT8  rxVrefHigh;
    UINT8  txVrefLow;
    UINT8  txVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;

typedef struct {
    UINT16  recEnDelay[MAX_STROBE];      // Units = 1/64 QCLK
    UINT16  wldDelay[MAX_STROBE];
    UINT8   rxDqDelay[MAX_STROBE];
    UINT8   txDqDelay[MAX_STROBE];
    UINT8   clkDelay;
    UINT8   ctlDelay;
    UINT8   cmdDelay[3];
    UINT8   ioLatency;
    UINT8   roundtrip;
} BDAT_RANK_TRAINING_STRUCTURE;

typedef struct {
    UINT16  mr0;
    UINT16  mr1;
    UINT16  mr2;
    UINT16  mr3;
} BDAT_RANK_MRS_STRUCTURE;

typedef struct {
    UINT8   rankEnabled;
    BDAT_RANK_MARGIN_STRUCTURE  rankMargin;
    BDAT_RANK_TRAINING_STRUCTURE  rankTraining;
    BDAT_RANK_MRS_STRUCTURE      rankMRS;
} BDAT_RANK_STRUCTURE;

typedef struct {
    UINT8   valid[MAX_SPD_BYTE/8];      // Each valid bit maps to SPD byte
    UINT8   spdData[MAX_SPD_BYTE];
} BDAT_SPD_STRUCTURE;

typedef struct {
    UINT8   dimmEnabled;
    BDAT_RANK_STRUCTURE      rankList[MAX_RANK_DIMM];
    BDAT_SPD_STRUCTURE        spdBytes;
} BDAT_DIMM_STRUCTURE;

typedef struct {
```



```

        UINT8                chEnabled;
        UINT8                numDimmSlot;
        BDAT_DIMM_STRUCTURE  dimmList[MAX_DIMM];
    } BDAT_CHANNEL_STRUCTURE;

typedef struct {
    UINT8                imcEnabled;
    UINT16               ddrFreq;           // Units of MT/s
    UINT16               ddrVoltage;       // Units of mV
    UINT16               imcDid;
    UINT8               imcRid;
    BDAT_CHANNEL_STRUCTURE channelList[MAX_CH];
} BDAT_NODE_STRUCTURE;

typedef struct {
    UINT32               refCodeRevision;
    UINT32               marginLoopCount;   // Units of cache line
    BDAT_NODE_STRUCTURE  nodeList[MC_MAX_NODE];
} BDAT_SYSTEM_STRUCTURE;

typedef struct bdatStruct {
    BDAT_HEADER_STRUCTURE bdatHeader;
    BDAT_SYSTEM_STRUCTURE bdatSys;
} BDAT_STRUCTURE;

#pragma pack()

```

Version 2.0: The following pseudo-code defines version 2.0 of the BDAT_SYSTEM_STRUCTURE.

```

#pragma pack(1)

typedef struct {
    UINT8  rxDqLeft;      // Units = piStep
    UINT8  rxDqRight;
    UINT8  txDqLeft;
    UINT8  txDqRight;
    UINT8  rxVrefLow;     // Units = rxVrefStep
    UINT8  rxVrefHigh;
    UINT8  txVrefLow;     // Units = txVrefStep
    UINT8  txVrefHigh;
} BDAT_DQ_MARGIN_STRUCTURE;

typedef struct {
    UINT8  rxDqLeft;      // Units = piStep
    UINT8  rxDqRight;
    UINT8  txDqLeft;
    UINT8  txDqRight;
    UINT8  cmdLeft;
    UINT8  cmdRight;
    UINT8  rxVrefLow;     // Units = rxVrefStep
    UINT8  rxVrefHigh;
    UINT8  txVrefLow;     // Units = txVrefStep
    UINT8  txVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;

typedef struct {
    UINT16  recEnDelay[maxStrobe];          // Array of nibble training results per rank
    UINT16  wlDelay[maxStrobe];
    UINT8   rxDqDelay[maxStrobe];
    UINT8   txDqDelay[maxStrobe];
    UINT8   clkDelay;
    UINT8   ctlDelay;
    UINT8   cmdDelay[3];
    UINT8   ioLatency;
    UINT8   roundtrip;
} BDAT_RANK_TRAINING_STRUCTURE;

typedef struct {

```



```
    UINT16 mr0;           // MR0 settings
    UINT16 mr1;           // MR1 settings
    UINT16 mr2;           // MR2 settings
    UINT16 mr3;           // MR3 settings
} BDAT_RANK_MRS_STRUCTURE;

typedef struct {
    UINT8 rankEnabled;      // 0 = Rank disabled
    UINT8 rankMarginEnabled; // 0 = Rank margin disabled
    UINT8 dqMarginEnabled;  // 0 = Dq margin disabled
    BDAT_RANK_MARGIN_STRUCTURE rankMargin; // Rank margin data
    BDAT_DQ_MARGIN_STRUCTURE dqMargin[maxDq]; // Array of Dq margin data per rank
    BDAT_RANK_TRAINING_STRUCTURE rankTraining; // Rank training settings
    BDAT_RANK_MRS_STRUCTURE rankMRS; // Rank MRS settings
} BDAT_RANK_STRUCTURE;

#define MAX_SPD_BYTE 256 // Number of bytes in Serial EEPROM

typedef struct {
    UINT8 valid[MAX_SPD_BYTE/8]; // Each valid bit maps to SPD byte
    UINT8 spdData[MAX_SPD_BYTE]; // Array of raw SPD data bytes
} BDAT_SPD_STRUCTURE;

typedef struct {
    UINT8 dimmEnabled; // 0 = DIMM disabled
    BDAT_RANK_STRUCTURE rankList[maxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_STRUCTURE spdBytes; // SPD data per DIMM
} BDAT_DIMM_STRUCTURE;

typedef struct {
    UINT8 chEnabled; // 0 = Channel disabled
    UINT8 numDimmSlot; // Number of slots per channel on the board
    BDAT_DIMM_STRUCTURE dimmList[maxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_STRUCTURE;

typedef struct {
    UINT8 imcEnabled; // 0 = MC disabled
    UINT16 imcDid; // MC device Id
    UINT8 imcRid; // MC revision Id
    UINT16 ddrFreq; // DDR frequency in units of MHz / 10
    // e.g. ddrFreq = 1333 for tCK = 1.5 ns
    UINT16 ddrVoltage; // Vdd in units of mV
    // e.g. ddrVoltage = 1350 for Vdd = 1.35 V
    UINT8 piStep; // Step unit = piStep * tCK / 2048
    // e.g. piStep = 16 for step = 11.7 ps (1/128 tCK)
    UINT16 rxVrefStep; // Step unit = rxVrefStep * Vdd / 100
    // e.g. rxVrefStep = 520 for step = 7.02 mV
    UINT16 txVrefStep; // Step unit = txVrefStep * Vdd / 100
    BDAT_CHANNEL_STRUCTURE channelList[maxCh]; // Array of channels per socket
} BDAT_SOCKET_STRUCTURE;

typedef struct {
    UINT32 refCodeRevision; // Matches scratchpad definition
    UINT8 maxNode; // Max processors per system, e.g. 4
    UINT8 maxCh; // Max channels per socket, e.g. 4
    UINT8 maxDimm; // Max DIMM per channel, e.g. 3
    UINT8 maxRankDimm; // Max ranks per DIMM, e.g. 4
    UINT8 maxStrobe; // Number of Dqs used by the rank, e.g. 18
    UINT8 maxDq; // Number of Dq bits used by the rank, e.g. 72
    UINT32 marginLoopCount; // Units of cache line
    BDAT_SOCKET_STRUCTURE socketList[maxNode]; // Array of sockets per system
} BDAT_SYSTEM_STRUCTURE;

typedef struct bdatStruct {
    BDAT_HEADER_STRUCTURE bdatHeader;
    BDAT_SYSTEM_STRUCTURE bdatSys;
} BDAT_STRUCTURE;

#pragma pack()
```



9 Faulty Parts Tracking

FPT feature is part of the MRC training flow for the following training steps:

- Receive Enable
- Read DQ/DQS
- Write Levelling
- Write DQ/DQS

During these training steps, FPT keeps track of errors detected on a per nibble (x4 width) basis. The results are stored in an array of 1 byte information per Node/Channel/DIMM/Rank/Strobe. The array is defined as:

```
UINT8          faultyParts[MAX_STROBE];
```

And is part of structure:

```
struct ddrRank {};
```

Initially all values in the array are set to 0x00. This means no error was found on any nibble. If a failing nibble is found during any of the training steps listed above it would be marked bitwise as described below:

```
#define FPT_REC_ENABLE_FAILED      (0x01)
#define FPT_RD_DQ_DQS_FAILED      (0x02)
#define FPT_WR_LEVELING_FAILED    (0x04)
#define FPT_WR_DQ_DQS_FAILED      (0x10)
```

If a nibble fails in a particular training step all subsequent training steps will be skipped for that particular nibble. After each training step where a faulty nibble has been found, an evaluation function is called to assess if the faulty nibble(s) create an uncorrectable error. If the result of the evaluation indicates that an uncorrectable error has been detected MRC execution will be stopped, otherwise the MRC will continue to execute as usual.

After MRC execution is complete, the above mentioned array can be read to obtain information on any failing nibbles detected.

This information is also part of the serial debug output. An example serial output with FPT information is shown below:

```
N0.C0.D0.R0.S17: Center Point = 39
N0.C0.D0.R0.S17: WrLevel Logic Delay = 0, Pi = 3
STOP_DATA_WR_LVL_BASIC
END - Wr Lvl Training Time      712 ms
Entering no zone 11
START - Tx Dq Training
Checkpoint Code: Socket 0, 0xB7, 0x04, 0x0000
FAULTY_PARTS_TRACKING: EvaluateFaultyParts status: 1
A warning has been logged! Warning Code = 0x16, Minor Warning Code = 0x0, Data = 0x0
Socket = 0 Channel = 0 DIMM = 0 Rank = 0
FAULTY_PARTS_TRACKING: EvaluateFaultyParts status: 2
N0.C0.D0.R0: WrDq/Dqs Failure! (status = 1)
A warning has been logged! Warning Code = 0x16, Minor Warning Code = 0x0, Data = 0x0
Socket = 0 Channel = 0 DIMM = 0 Rank = 0
```



10 Storage Feature MRC Users Guide

The MRC contains support for 2 different ADR usage models that take two very different paths through the MRC. This guide provides a brief overview of each usage model and describes how the MRC should be configured for each usage model.

ADR+BBU: In this usage model the DIMMs are placed in self-refresh upon system power loss by ADR. Following self-refresh entry power is removed from the entire system except for the DIMMs, which are powered via battery. The system remains in this state until system power is restored. Upon entering the MRC, the DIMMs contain data that must be preserved. Therefore the MRC takes a path very similar to S3Resume to insure that data is not lost.

ADR+NVDIMM: In this usage model the DIMMs are placed in self-refresh upon system power loss by ADR. However, following self-refresh entry, an NVDIMM will isolate itself from the host DDR bus. After isolation, on-DIMM controller will copy all data from DRAM to Flash that is also located on the NVDIMM. During this process, the NVDIMM is powered by a Supercap pack attached to NVDIMM via cable, and the rest of the system is free to completely power down. When power is restored the system will power back up, and the MRC will instruct the NVDIMMs to copy data back from Flash to DRAM.

10.1 MRC Configuration Options

Compile time Flags:

#define	Description
MEM_NVDIMM_EN	Controls if NVDIMM code is enabled or not
NVMEM_FEATURE_EN	Controls if the Enhanced interleave code required by both NVDIMM and ADR is enabled

Input Parameters:

Variable	Description
batterybacked	Controls if a given channel is battery backed or not. Only applies to ADR+BBU Must be 0 for all channels for NVDIMM Usage models
ADREn	Controls if the ADR trigger will be enabled. Should be set for both ADR+BBU and ADR+NVDIMM
check_pm_sts	Controls if the PCH_PM_STS status register is read to determine if the system should recover from ADR Only applies to ADR+BBU Must be 0 for all channels for NVDIMM Usage models
check_platform_detect	Controls if the platform specific PlatformDetectADR function is called determine if the system should recover from ADR Only applies to ADR+BBU Must be 0 for all channels for NVDIMM Usage models
eraseArmNVDIMMS	Controls if the MRC will Erase and Arm all NVDIMM in the system or if it will leave those operations to the OS
restoreNVDIMMS	Controls if the MRC will restore NVDIMMs or not
interNVDIMMS	Controls if NVDIMMs are interleaved with other NVDIMMs or not

Recommended Configuration values

Parameter	ADR+BBU	ADR + NVDIMM
MEM_NVDIMM_EN	Don't Care	Must be Defined
NVMEM_FEATURE_EN	Must be Defined	Must be Defined
batterybacked	Set for battery backed channels	Clear for all channels
ADREn	Must be Set	Must be Set
check_pm_sts	Platform Dependent	Must be Clear
check_platform_detect	Platform Dependent	Must be Clear
eraseArmNVDIMMS	Don't Care	Platform Dependent
restoreNVDIMMS	Don't Care	Typically Set

interNVDIMMS	Don't Care	Platform Dependent
--------------	------------	--------------------

10.2 Output Parameters

Parameter	Description		
nvDimmType	A per DIMM field that indicates if a given DIMM is a normal DIMM or an NVDIMM. A value of 0 indicates that it is either not an NVDIMM or not an NVDIMM recognized by the system. Any other value indicates that it is an NVDIMM.		
nvDimmStatus	A per DIMM field that provides status about the operations performed on the NVDIMM by the MRC. The following bits are defined		
	Name	Bit	Description
	STATUS_RESTORE_SUCCESSFUL	2	Indicates that a restore was successfully completed by this NVDIMM.
	STATUS_ARMED	3	Indicates that this NVDIMM was successfully armed, and is capable of starting a SAVE operation
	ERROR_DETECT	8	Indicates that the MRC was unable to determine the status of the NVDIMM, and did not attempt any further operations on it
	ERROR_RESTORE	9	Indicates that the MRC attempted to complete a RESTORE operation, but failed
	ERROR_ARM	10	Indicates that the MRC attempted to complete a ARM operation, but failed
NVmemSize	A variable present at the dimm, channel, home agent, socket, or full system levels to determine the capacity of non-volatile memory(from ADR+BBU or ADR+NVDIMM) present at a given level of the hierarchy. These variables in conjunction with the NUMA_AWARE flag can be used by the BIOS to determine the memory map of the system.		

10.3 Memory map layout

If the NVMEM_FEATURE_EN flag is set the MRC will setup the following memory map for NUMA systems:

- All normal memory from socket 0 interleaved together
- All non-volatile memory(NVDIMM or BBU memory) from socket 0 interleaved together
- All normal memory from socket 1 interleaved together
- All non-volatile memory (NVDIMM or BBU memory) from socket 1 interleaved together

For example, if the system contains 4 Normal 4GB DIMMs on Socket 0, 2 4GB NVDIMMs on Socket 0, 8 Normal 4GB DIMMs on Socket 1 and 6 4GB NVDIMMs on Socket 1, the memory map will be the following assuming TOLM is set to 2GB

- 0-2GB: Socket 0 Normal memory
- 4-18GB: Socket 0 Normal memory
- 18-26GB: Socket 0 non-volatile memory
- 26-58GB: Socket 1 Normal memory
- 58-82GB: Socket 1 non-volatile memory



If NUMA is disabled the memory map will be:

- All non-NVDIMM DIMMs from socket both sockets interleaved together

- All NVDIMM DIMMs from both sockets interleaved together