BIOS Data ACPI Table (BDAT)

Interface Specification v4.0.9

February 2022
The material Contained herein is not a license, either expressly or impliedly, to any intellectual property owned or Controlled By any of the authors or developers of this material or to any Contribution thereto. The material Contained herein is provided on an "AS IS" Basis and, to the maximum extent permitted By applicable law, this information is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material hereby disclaim all other warranties and Conditions, either express, implied or statutory, including, But not limited to, any (if any) implied warranties, duties or Conditions of merchantability, of fitness for a particular purpose, of accuracy or Completeness of responses, of results, of workmanlike effort, of lack of viruses and of lack of negligence, all with regard to this material and any Contribution thereto. Designers must not rely on the absence or Characteristics of any features or instructions marked "reserved" or "undefined." The Unified EFI Forum, Inc. reserves any features or instructions so marked for future definition and shall have no responsibility whatsoever for Conflicts or incompatibilities arising from future Changes to them. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SPECIFICATION AND ANY CONTRIBUTION THERETO.

IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR ANY CONTRIBUTION THERETO BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS DOCUMENT, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright © 2020, Unified Extensible Firmware Interface (UEFI) Forum, Inc. All Rights Reserved. The UEFI Forum is the owner of all rights and title in and to this work, including all copyright rights that may exist, and all rights to use and reproduce this work. Further to such rights, permission is hereby granted to any person implementing this specification to maintain an electronic version of this work accessible by its internal personnel, and to print a copy of this specification in hard copy form, in whole or in part, in each case solely for use by that person in connection with the implementation of this Specification, provided no modification is made to the Specification.
# Contents

1 Introduction .......................................................................................................................... 6  
1.1 Purpose ............................................................................................................................ 6  
1.2 Intended Audience ............................................................................................................ 6  
1.3 Related Documents .......................................................................................................... 6  
2 Requirements & Overview ................................................................................................... 7  
2.1 Requirements .................................................................................................................... 7  
2.2 Overview .......................................................................................................................... 7  
3 ACPI Table Interface .......................................................................................................... 8  
4 BIOS Data Structure Header ............................................................................................... 10  
4.1 Version 4.0 ....................................................................................................................... 11  
5 Memory Schemas ................................................................................................................ 13  
5.1 Memory Data Schema 2 .................................................................................................. 13  
5.2 Memory Data Schema 2B ............................................................................................... 16  
5.3 Memory Data Schema 4 .................................................................................................. 19  
5.4 Memory Data Schema 4B .............................................................................................. 21  
5.5 RMT Schema 4 .............................................................................................................. 23  
5.6 RMT Schema 5 .............................................................................................................. 25  
5.7 Columnar Style Memory Schema 6 ................................................................................ 27  
5.7.1 RMT Schema 6 ........................................................................................................... 30  
5.7.1.1 RMT Schema 6 metadata ..................................................................................... 30  
5.7.1.2 RMT Schema 6 columns ..................................................................................... 31  
5.7.2 RMT Schema 6B ....................................................................................................... 35  
5.7.2.1 RMT Schema 6B metadata .................................................................................. 35  
5.7.2.2 RMT Schema 6B columns .................................................................................. 35  
5.8 DIMM SPD RAW Data Schema 7 ............................................................................... 39  
5.9 Memory Training Data Schema 8 .................................................................................. 40  
6 PCI Express (PCIE) Schemas ............................................................................................ 49  
6.1 PCIe Topology Schema ................................................................................................. 49  
6.2 PCIe Software Equalization Phase 2/3 Schema ............................................................. 51  
6.3 PCIe Software Equalization Score Schema ................................................................... 53  
6.3.1 Fixed Decimal Point Parsing Sample Code ............................................................... 53  
6.4 PCIe Port Margin Schema .............................................................................................. 55  
6.5 PCIe Lane Margin Schema ............................................................................................. 57  
7 eMMC Schemas ................................................................................................................ 58  
7.1 eMMC Bus Margin Schema .......................................................................................... 58  
8 EWL Schema .................................................................................................................... 60  
9 Appendix A – Acronyms ..................................................................................................... 62
Introduction

Figures

No table of figures entries found.

Tables

No table of figures entries found.
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2015</td>
<td>4.0</td>
<td>Initial version based on Compatible BIOS Data Structure v1.0</td>
</tr>
<tr>
<td>January 2016</td>
<td>4.0.1</td>
<td>Added columnar style memory result schema</td>
</tr>
<tr>
<td>June 2017</td>
<td>4.0.2</td>
<td>Added RMT result schema 6B. Added comments to RMT schema 6.</td>
</tr>
<tr>
<td>February 2020</td>
<td>4.0.3</td>
<td>Added product specific schema 6 columns for Purley and Whitley/Jacobsville</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to section 5.7.3.2.3</td>
</tr>
<tr>
<td>July 2020</td>
<td>4.0.4</td>
<td>Added Dimm SPD schema 7 section 5.8, Memory training data schema 8 section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.9 and EWL schema section 8.</td>
</tr>
<tr>
<td>November 2020</td>
<td>4.0.5</td>
<td>Updated license disclaimer for contribution to UEFI Forum</td>
</tr>
<tr>
<td>February 2021</td>
<td>4.0.6</td>
<td>Added PPIN and UUID to Memory training data schema 8</td>
</tr>
<tr>
<td>April 2021</td>
<td>4.0.7</td>
<td>Added turnaround timing data to Memory training data schema 8</td>
</tr>
<tr>
<td>June 2021</td>
<td>4.0.8</td>
<td>Added DIMM(DRAM/RCD/DB) register entry, updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION to support pseudo channel and subrank,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>updated MEM_TRAINING_DATA_SCOPE to make type1 entry support signal data in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>memory training data schema 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed EWL entry type definition which was defined in the EWL spec.</td>
</tr>
<tr>
<td>February 2022</td>
<td>4.0.9</td>
<td>Adding the BDAT_PRODUCT_SPECIFIC_DATA_STRUCT and BDAT_MARGIN_CONFIG_STRUCT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>schemas</td>
</tr>
</tbody>
</table>

§
1 Introduction

1.1 Purpose
The purpose of this document is to describe the interfaces for the BIOS Data ACPI Table.

1.2 Intended Audience
This document is targeted at all platform and system developers who need to consume BDAT interface in their solutions. This includes, but is not limited to: system IA firmware or BIOS developers, bootloader developers, system integrators, as well as end users.

1.3 Related Documents
- JEDEC Specifications located at http://www.jedec.org/standards-documents
Requirements & Overview

2 Requirements & Overview

2.1 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. Optionally, the platform could provide a mechanism for remote access using an ITP/JTAG connection. The exact mechanism for ITP based access is beyond the scope of this document. 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.

2.2 Overview

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.

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 destination 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 UEFI 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. All accesses to the BIOS data structure should be directed to the final runtime location in the ACPI Type 2 region.
ACPI Table Interface

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.

```c
#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 6.0 Generic Address Space definition
typedef struct {
    UINT8   AddressSpaceId;
    UINT8   RegisterBitWidth;
    UINT8   RegisterBitOffset;
    UINT8   AccessSize;
    UINT64  Address;
} EFI_ACPI_6_0_GENERIC_ADDRESS_STRUCTURE;
```
ACPI Table Interface

// BIOS Data ACPI structure
typedef struct {
    EFI_ACPI_DESCRIPTION_HEADER          Header;
    EFI_ACPI_6_0_GENERIC_ADDRESS_STRUCTURE BdatGas;
} BDAT_ACPI_DESCRIPTION_TABLE;

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

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

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

#pragma pack()
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.

The BiosDataStructSize field should indicate the total size of all the BDAT data, including the header, the table of offsets, and all of the schemas.

```c
#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 this field)
    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()
```
4.1 Version 4.0

Version 4.0 of the BDAT structure enables the BIOS reference code to publish any possible data structure desired to OS applications.

To accomplish this, the version 4.0 structure defines an array of offsets. Each offset marks the start of a new data structure relative to the beginning of the BDAT structure. The data structures identify themselves by a GUID, which indicates the schema of the data they contain. At a higher conceptual level, this creates a list of key-value pairs with each key being a GUID that identifies the schema of the data located at the offset. The `BDAT_SCHEMA_LIST_STRUCTURE` also contains date-time information indicating when the structure was generated.

```
#pragma pack(push, 1)
typedef struct BdatSchemaList {
    UINT16                SchemaListLength;
    UINT16                Reserved;
    UINT16                Year;
    UINT8                 Month;
    UINT8                 Day;
    UINT8                 Hour;
    UINT8                 Minute;
    UINT8                 Second;
    UINT8                 Reserved;
    UINT32                Schemas[SchemaListLength];
} BDAT_SCHEMA_LIST_STRUCTURE;
```

```
typedef struct BdatStruct {
    BDAT_HEADER_STRUCTURE        BdatHeader;
    BDAT_SCHEMA_LIST_STRUCTURE   BdatSchemas;
} BDAT_STRUCTURE;
```

```
#pragma pack(pop)
```

Every structure pointed to by the schemas array will have the following header:

```
#pragma pack(push, 1)
typedef struct BdatSchemaHeader {
    EFI_GUID              SchemaId;
    UINT32                DataSize;
    UINT16                Crc16;
} BDAT_SCHEMA_HEADER_STRUCTURE;
```

```
#pragma pack(pop)
```

The SchemaId GUID uniquely identifies the format of the data contained within the structure. If a change is required to the schema, then one simply assigns a new GUID to the schema. DataSize indicates the total size of the memory block, including both the header as well as the schema specific data. Crc16 is computed in the same manner as the field on `BDAT_HEADER_STRUCTURE`. 
Data following the `BDAT_SCHEMA_HEADER_STRUCTURE` is dependent on the schema. The schema itself is responsible for defining the top level data structure for the schema. When doing so, the top level structure must have a `BDAT_SCHEMA_HEADER_STRUCTURE` as the first element. Since the schema header is the first entry in the structure, the UINT32 to this structure in the Schemas array can be added to a pointer to the BDAT structure and then casted to either `(BDAT_SCHEMA_HEADER_STRUCTURE*)` or to the top level schema structure itself. The flow on the data extraction side would first cast to the header to read the GUID. Then if it’s the data the extractor tool is looking for, it would cast again to the appropriate top level schema data structure.

The following sections define the schemas that are currently defined by this specification.
5 Memory Schemas

5.1 Memory Data Schema 2

This memory schema stores data produced by the Rank Margin Tool included with the Memory Reference Code for several Intel platforms.

```c
#pragma pack(push, 1)
///
/// Memory Schema 2 GUID
///
/// {CE3F6794-4883-492C-8DBA-2FC09844710}
///
#define BDAT_MEMORY_DATA_2_GUID \
   { \
   0xCE3F6794, 0x4883, 0x492C, 0x8D, 0xBA, 0x2F, 0xC0, 0x98, 0x44, 0x77, 0x10\n   }
#define MAX_MODE_REGISTER 7  // Number of mode registers
#define MAX_DRAM_DEVICE 9    // Maximum number of memory devices

typedef struct {
   UINT16 modeRegister[MAX_MODE_REGISTER]; // Mode register settings
} BDAT_DRAM_MRS_STRUCTURE;

typedef struct {
   UINT8 RxDqLeft;  // Units = PiStep
   UINT8 RxDqRight;
   UINT8 TxDqLeft;
   UINT8 TxDqRight;
   UINT8 RxVrefLow;  // Units = RxVrefStep
   UINT8 RxVrefHigh;
   UINT8 TwVrefLow;  // Units = TwVrefStep
   UINT8 TwVrefHigh;
} BDAT_DQ_MARGIN_STRUCTURE;
```
typedef struct {
    UINT8 RxDqLeft;       // Units = PiStep
    UINT8 RxDqRight;
    UINT8 TxDqLeft;
    UINT8 TxDqRight;
    UINT8 CmdLeft;
    UINT8 CmdRight;
    UINT8 RecvEnLeft;     // Units = RecvEnStep
    UINT8 RecvEnRight;
    UINT8 WrLevLeft;      // Units = WrLevStep
    UINT8 WrLevRight;
    UINT8 RxVrefLow;      // Units = RxVrefStep
    UINT8 RxVrefHigh;
    UINT8 TxVrefLow;      // Units = TxVrefStep
    UINT8 TxVrefHigh;
    UINT8 CmdVrefLow;     // Units = caVrefStep
    UINT8 CmdVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;

typedef struct {
    UINT16 RecsDelay[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 {
    UINT8 RankEnabled;       // 0 = Rank disabled
    UINT8 RankMarginEnabled;  // 0 = Rank margin disabled
    UINT8 DqMarginEnabled;    // 0 = Dq margin disabled
    BDAT_DQ_MARGIN_STRUCTURE DqMargin[MaxDq];  // Array of Dq margin data per rank
    BDAT_RANK_MARGIN_STRUCTURE RankMargin;       // Rank margin data
    BDAT_RANK_TRAINING_STRUCTURE RankTraining;   // Rank training settings
    BDAT_DRAM_MRS_STRUCTURE RankMrs[Max_DRAM_DEVICE]; // Rank MRS settings
} BDAT_RANK_2_STRUCTURE;

#define MAX_SPD_BYTE_512 512 // Number of bytes in Serial EEPROM

typedef struct {
    UINT8 Valid[MAX_SPD_BYTE_512/8]; // Each valid bit maps to SPD byte
    UINT8 SpdData[MAX_SPD_BYTE_512];  // Array of raw SPD data bytes
} BDAT_SPD_2_STRUCTURE;

typedef struct {
    UINT8 DimmEnabled;     // 0 = DIMM disabled
    BDAT_RANK_2_STRUCTURE RankList[MaxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_2_STRUCTURE SpdBytes;  // SPD data per DIMM
} BDAT_DIMM_2_STRUCTURE;
typedef struct {
    UINT8 ChEnabled; // 0 = Channel disabled
    UINT8 NumDimmSlot; // Number of slots per channel on the board
    BDAT_DIMM_2_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_2_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 = 13333 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
    UINT8 CaVrefStep; // Step unit = caVrefStep * Vdd / 100
    UINT8 RecvEnStep; // Step unit = RecvEnStep * tCK / 2048
    UINT8 WrLevStep; // Step unit = WrLevStep * tCK / 2048
    BDAT_CHANNEL_2_STRUCTURE ChannelList[MaxCh]; // Array of channels per socket
} BDAT_SOCKET_2_STRUCTURE;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    UINT32 RefCodeRevision;
    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_2_STRUCTURE SocketList[MaxNode]; // Array of sockets per system
} BDAT_MEMORY_DATA_2_STRUCTURE;

#pragma pack(pop)
5.2 Memory Data Schema 2B

Same as the memory data schema 2 except that dqLaneCnt and rankMarginValidSignals were added to BDAT_RANK_2B_STRUCTURE.

```c
#pragma pack(push, 1)
```

```c
/// Memory Schema 2B GUID
///
/// {DE18DF61-E783-4E1D-87AA-4D8083D17C25}
///
#define BDAT_MEMORY_DATA_2B_GUID \
{ 0xde18df61, 0xe783, 0x4e1d, 0x87, 0xaa, 0x4d, 0x80, 0x83, 0x93, 0xd1, 0x7c, 0x25 }
```

```c
#define MAX_MODE_REGISTER 7 // Number of mode registers
#define MAX_DRAM_DEVICE 9 // Maximum number of memory devices
```

```c
typedef struct {
    UINT16 ModeRegister[MAX_MODE_REGISTER]; // Mode register settings
} BDAT_DRAM_MRS_STRUCTURE;
```

```c
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;
```

```c
typedef struct {
    UINT8 RxDqLeft; // Units = PiStep
    UINT8 RxDqRight;
    UINT8 TxDqLeft;
    UINT8 TxDqRight;
    UINT8 CmdLeft;
    UINT8 CmdRight;
    UINT8RecvEnLeft; // Units = RecvEnStep
    UINT8 RecvEnRight;
    UINT8 WrLevLeft; // Units = WrLevStep
    UINT8 WrLevRight;
    UINT8 RxVrefLow; // Units = RxVrefStep
    UINT8 RxVrefHigh;
    UINT8 TxVrefLow; // Units = TxVrefStep
    UINT8 TxVrefHigh;
    UINT8 CmdVrefLow; // Units = CaVrefStep
    UINT8 CmdVrefHigh;
} 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 {
    UINT8 RankEnabled; // 0 = Rank disabled
    UINT8 RankMarginEnabled; // 0 = Rank margin disabled
    UINT8 RankMarginValidSignals; // Each valid bit maps to a RMT signal
        // bit 0 - RxDq, bit 1 - TxDq,
        // bit 2 - Cmd, bit 3 - RecvEn
        // bit 4 - Wrlevel, bit 5 - RxVref
        // bit 6 - TxVref, bit 7 - CmdVref
    UINT8 DqMarginEnabled; // 0 = Dq margin disabled
    UINT8 DqLaneCnt; // Actual DQ lane cnt
} BDAT_RANK_MARGIN_STRUCTURE RankMargin; // Rank margin data

typedef struct {
    UINT16 RankMrs[MaxRankDimm]; // Array of ranks per DIMM
} BDAT_RANK_2B_STRUCTURE RankMrs[MaxRankDimm]; // Rank MRS settings

typedef struct {
    UINT8 ChEnabled; // 0 = Channel disabled
    UINT8 NumDimmSlot; // Number of slots per channel on the board
} BDAT_CHANNEL_2_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel

typedef struct {
    UINT8 Valid[MAX_SPD_BYTE_512/8]; // Each valid bit maps to SPD byte
    UINT8 SpdData[MAX_SPD_BYTE_512]; // Array of raw SPD data bytes
} BDAT_SPD_2_STRUCTURE;

typedef struct {
   (UINT8 DimmEnabled; // 0 = DIMM disabled
    BDAT_RANK_2B_STRUCTURE RankList[MaxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_2_STRUCTURE SpdBytes; // SPD data per DIMM
} BDAT_DIMM_2_STRUCTURE;

typedef struct {
    UINT8 ChEnabled; // 0 = Channel disabled
    UINT8 NumDimmSlot; // Number of slots per channel on the board
    BDAT_DIMM_2_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_2_STRUCTURE;

#define MAX_SPD_BYTE_512 512 // Number of bytes in Serial EEPROM
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 = 13333 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 TwVrefStep; // Step unit = TwVrefStep * Vdd / 100
    UINT16 CaVrefStep; // Step unit = CaVrefStep * Vdd / 100
    UINT8RecvEnStep; // Step unit =RecvEnStep * tCK / 2048
    UINT8 WrLevStep; // Step unit = WrLevStep * tCK / 2048
} BDAT_CHANNEL_2_STRUCTURE

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    UINT32 RefCodeRevision;
    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_MEMORY_DATA_2B_STRUCTURE;

#pragma pack(pop)
5.3 Memory Data Schema 4

This memory schema stores the same data as the memory data schema 2. Cached txvref training values for each rank where added. More significantly, the RMT data is separated from the other memory data. This allows for alternate mechanisms of generating the RMT data. If you do not intend to include a BDAT_DRAM_MRS_STRUCTURE please use Memory Schema 4b instead.

```c
#pragma pack(push, 1)

/// Memory Schema 4 GUID
/// {715C6C51-7774-42E7-AB06-51BDB5A24615}
///
#define BDAT_MEMORY_DATA_4_GUID
  {
    0x715C6C51, 0x7774, 0x42E7, 0xAB, 0x06, 0x51, 0xB5, 0xA2, 0x46, 0x15
  }

typedef struct {
  UINT16  Mr0;       // MR0 settings
  UINT16  Mr1;       // MR1 settings
  UINT16  Mr2;       // MR2 settings
  UINT16  Mr3;       // MR3 settings
  UINT16  Mr4;       // MR4 settings
  UINT16  Mr5;       // MR5 settings
  UINT16  Mr6[MaxMrDevice]; // MR6 settings
} BDAT_DRAM_MRS_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;
  UINT8   Txvref[MaxStrobe]; // TxVref training values per rank & strobe
} BDAT_RANK_TRAINING_4_STRUCTURE;

typedef struct {
  UINT8                          RankEnabled;    // 0 = Rank disabled
  BDAT_RANK_TRAINING_4_STRUCTURE RankTraining; // Rank training settings
  BDAT_DRAM_MRS_STRUCTURE        RankMrs[MaxMr]; // Rank MRS settings
} BDAT_RANK_4_STRUCTURE;

#define MAX_SPD_BYTE_512         512 // Number of bytes in Serial EEPROM

typedef struct {
  UINT8                          Valid[MAX_SPD_BYTE_512/8]; // Each valid bit maps to SPD byte
  UINT8                          SpdData[MAX_SPD_BYTE_512]; // Array of raw SPD data bytes
} BDAT_SPD_4_STRUCTURE;
```
typedef struct {
    UINT8 DimmEnabled;  // 0 = DIMM disabled
    BDAT_RANK_4_STRUCTURE RankList[MaxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_4_STRUCTURE SpdBytes;  // SPD data per DIMM
} BDAT_DIMM_4_STRUCTURE;

typedef struct {
    UINT8 ChEnabled;  // 0 = Channel disabled
    UINT8 NumDimmSlot; // Number of slots per channel on the board
    BDAT_DIMM_4_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_4_STRUCTURE;

typedef struct {
    UINT8 ImcEnabled;  // 0 = MC disabled
    UINT8 ImcDid;    // MC device Id
    UINT8 ImcRid;   // MC revision Id
    UINT16 DdrFreq; // DDR frequency in units of MHz / 10
    // e.g. DdrFreq = 13333 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
    UINT16 CaVrefStep; // Step unit = CaVrefStep * Vdd / 100
    UINT8RecvEnStep; // Step unit = RecvEnStep * tCK / 2048
    UINT8 WrLevStep; // Step unit = WrLevStep * tCK / 2048
    BDAT_CHANNEL_4_STRUCTURE ChannelList[MaxCh]; // Array of channels per socket
} BDAT_SOCKET_4_STRUCTURE;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    UINT32 RefCodeRevision;
    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; // Num of Dqs used by the rank, e.g.18
    UINT8 MaxMr; // Number of DRAM MRS structures
    UINT8 MaxMrDevice; // Number of DRAM MRS registers
    BDAT_SOCKET_4_STRUCTURE SocketList[MaxNode]; // Array of sockets per system
} BDAT_MEMORY_DATA_4_STRUCTURE;

#pragma pack(pop)
5.4 Memory Data Schema 4B

Same as memory data schema 4 but without the BDAT_DRAM_MRS_STRUCTURE, MaxMr and MaxMrDevice variables.

```c
#pragma pack(push, 1)

/// Memory Schema 4B GUID
/// {5B274DC7-4222-4033-BAC8-5F13A111A215}

#define BDAT_MEMORY_DATA_4B_GUID \
{ 0x5b274dc7, 0x4222, 0x4033, 0xba, 0xc8, 0x5f, 0x13, 0xa1, 0x11, 0xa2, 0x15 }

typedef struct {
    UINT16 RecEnDelay[MaxStrobe]; // Array of nibble training results per rank
    UINT16 WlDelay[MaxStrobe];
    UINT8  RxQDelay[MaxStrobe];
    UINT8  TxQDelay[MaxStrobe];
    UINT16 ClkDelay;
    UINT16 CtlDelay;
    UINT16 CmdDelay[3];
    UINT8 IoLatency;
    UINT8 Roundtrip;
    UINT8 TxVref[MaxStrobe]; // TxVref training values per rank & strobe
} BDAT_RANK_TRAINING_4_STRUCTURE;

typedef struct {
    UINT8 RankEnabled; // 0 = Rank disabled
    BDAT_RANK_TRAINING_4_STRUCTURE RankTraining; // Rank training settings
} BDAT_RANK_4_STRUCTURE;

#define MAX_SPD_BYTE_512 512 // Number of bytes in Serial EEPROM

typedef struct {
    UINT8 Valid[MAX_SPD_BYTE_512/8]; // Each valid bit maps to SPD byte
    UINT8 SpdData[MAX_SPD_BYTE_512]; // Array of raw SPD data bytes
} BDAT_SPD_4_STRUCTURE;

typedef struct {
    UINT8 DimmEnabled; // 0 = DIMM disabled
    BDAT_RANK_4_STRUCTURE RankList[MaxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_4_STRUCTURE SpdBytes; // SPD data per DIMM
} BDAT_DIMM_4_STRUCTURE;

typedef struct {
    UINT8 ChEnabled; // 0 = Channel disabled
    UINT8 NumDimmSlot; // Number of slots per channel on the board
    BDAT_DIMM_4_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_4_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 = 13333 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
    UINT16 CaVrefStep;  // Step unit = CaVrefStep * Vdd / 100
    UINT8 RecvEnStep;  // Step unit = RecvEnStep * tCK / 2048
    UINT8  WrLevStep;  // Step unit = WrLevStep * tCK / 2048
} BDAT_CHANNEL_4_STRUCTURE;  // Array of channels per socket

BDAT_MEMORY_DATA_4B_STRUCTURE {
    BDAT_SCHEMA_HEADER_STRUCTURE  SchemaHeader;
    UINT32 RefCodeRevision;  // Matches JKT 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
} BDAT_SOCKET_4_STRUCTURE;  // Array of sockets per system

#pragma pack(pop)
5.5 RMT Schema 4

This memory schema stores the same data as the memory data schema 2. The changes are that the RMT data is separated from the other memory data. This allows for alternate mechanisms of generating the RMT data. This schema defines the RMT results.

```c
#pragma pack(push, 1)

/// RMT Schema 4 GUID
/// {E2E0270A-6F87-4759-A239-CA867170AE83}

#define BDAT_RMT_4_GUID
{ 0xE2E0270A, 0x6F87, 0x4759, 0xA2, 0x39, 0xCA, 0x86, 0x71, 0x70, 0xAE, 0x83 }

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   RecvEnLeft;     // Units = RecvEnStep
    UINT8   RecvEnRight;
    UINT8   WrLevLeft;      // Units = WrLevStep
    UINT8   WrLevRight;
    UINT8   RxVrefLow;      // Units = RxVrefStep
    UINT8   RxVrefHigh;
    UINT8   TxVrefLow;      // Units = TxVrefStep
    UINT8   TxVrefHigh;
    UINT8   CmdVrefLow;     // Units = CaVrefStep
    UINT8   CmdVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;

typedef struct {
    UINT8                      DimmEnabled;    // 0 = DIMM disabled
    BDAT_RMT_RANK_4_STRUCTURE  RankList[MaxRankDimm]; // Array of ranks per DIMM
} BDAT_RANK_RANK_4_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_RMT_RANK_4_STRUCTURE;

typedef struct {
    UINT8                      DimmEnabled;    // 0 = DIMM disabled
    BDAT_RMT_RANK_4_STRUCTURE  RankList[MaxRankDimm]; // Array of ranks per DIMM
} BDAT_RANK_RANK_4_STRUCTURE;
```
typedef struct {
    UINT8   ChEnabled;   // 0 = Channel disabled
    UINT8   NumDimmSlot; // Number of slots per channel on the board
    BDAT_RMT_DIMM_4_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_RMT_CHANNEL_4_STRUCTURE;

typedef struct {
    UINT8                         ImcEnabled;   // 0 = MC disabled
    BDAT_RMT_CHANNEL_4_STRUCTURE  ChannelList[MaxCh]; // Array of channels per socket
} BDAT_RMT_SOCKET_4_STRUCTURE;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE  SchemaHeader;
    UINT32  RefCodeRevision;
    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   MaxDq;         // Number of Dq bits used by the rank, e.g. 72
    UINT32  MarginLoopCount; // Units of cache line
    BDAT_RMT_SOCKET_4_STRUCTURE  SocketList[MaxNode]; // Array of sockets per system
} BDAT_RMT_4_STRUCTURE;

#pragma pack(pop)
5.6 RMT Schema 5

This memory schema removes the write leveling and receive enable entries from BDAT_RANK_MARGIN_STRUCTURE and adds entries for CTL timing margins. This schema defines the RMT results.

```c
#pragma pack(push, 1)
///<
/// RMT Schema 5 GUID
///<
/// \{1838678E-ED14-4e70-A90D-BF053D2\}
///<
#define BDAT_RMT_5_GUID \{
\ 0x1838678E, 0xED14, 0x4E70, 0xA9, 0xD, 0x48, 0x57, 0x2B, 0xF0, 0x53, 0xD2 \}
```

```c
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;
```

```c
typedef struct {
    UINT8   RxDqLeft; // Units = PiStep
    UINT8   RxDqRight;
    UINT8   TxDqLeft;
    UINT8   TxDqRight;
    UINT8   CmdLeft;
    UINT8   CmdRight;
    UINT8   CtlLeft;
    UINT8   CtlRight;
    UINT8   RxVrefLow; // Units = RxVrefStep
    UINT8   RxVrefHigh;
    UINT8   TxVrefLow; // Units = TxVrefStep
    UINT8   TxVrefHigh;
    UINT8   CmdVrefLow; // Units = CaVrefStep
    UINT8   CmdVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;
```

```c
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_RMT_RANK_5_STRUCTURE;
```
typedef struct {
    UINT8                     DimmEnabled;    // 0 = DIMM disabled
    BDAT_RMT_RANK_5_STRUCTURE RankList[MaxRankDimm]; // Array of ranks per DIMM
} BDAT_RMT_DIMM_5_STRUCTURE;

typedef struct {
    UINT8   ChEnabled;    // 0 = Channel disabled
    UINT8   NumDimmSlot; // Number of slots per channel on the board
    BDAT_RMT_DIMM_5_STRUCTURE DimmList[MaxDimm]; // Array of DIMMs per channel
} BDAT_RMT_CHANNEL_5_STRUCTURE;

typedef struct {
    UINT8   ImcEnabled;    // 0 = MC disabled
    BDAT_RMT_CHANNEL_5_STRUCTURE ChannelList[MaxCh]; // Array of channels per socket
} BDAT_RMT_SOCKET_5_STRUCTURE;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE  SchemaHeader;
    UINT32                     RefCodeRevision; // Matches JKT 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                     MaxDq;         // Number of Dq bits used by the rank, e.g. 72
    UINT32                    MarginLoopCount; // Units of cache line
    BDAT_RMT_SOCKET_5_STRUCTURE SocketList[MaxNode]; // Array of sockets per system
} BDAT_RMT_5_STRUCTURE;

#pragma pack(pop)
5.7 **Margin Configuration Schema**

This memory schema exposes some configuration elements associated with the margin configuration.

```c
#pragma pack(push, 1)

/// Margin Configuration Schema GUID
/// {890ce5bf-008f-46c6-9570-ec07b13a389c}

#define BDAT_Margin_Config_GUID
{ 0x890ce5bf, 0x008f, 0x46c6, 0x95, 0x70, 0xec, 0x07, 0xb1, 0x3a, 0x38, 0x9c }

typedef struct {
  UINT8   MarginConfigRevision; // Revision number for this struct
  UINT8   MarginTestMajorVer;   // Major version number. Updated whenever a
                                // significant change in RMT is introduced.
  UINT8   MarginTestMinorVer;   // Minor version number. Update whenever one or more
                                // minor features in RMT is introduced.
  UINT8   MarginTestRevVer;     // Revision Number. Updated whenever a new RMT build is
                                // released (e.g. bug fixes)
  UINT32  DqNumberOfUi;        // A measure of the number of bits transmitted during a
                                // single margin test on the data bus (e.g. during the
                                // Write margining there are a series of Writes followed
                                // by Reads. This metric will only count the number of
                                // UI transmitted during the Write traffic)
  UINT32  CmdNumberOfUi;       // A measure of the number of bits transmitted during a
                                // single margin test on the command bus.
  UINT8   DqStress;            // Indicate stress profile for the data bus traffic.
                                // 0 - Rotating victim-aggressor LFSR write/read
                                // loopback test. Victim bit lane rotates within the
                                // byte lane. Write subsequence with back to back
                                // cachelines, followed by read subsequence of the same
                                // length.
                                // 1 - Turn-around Test pattern: rotating victim-
                                // aggressor LFSR write/read loopback test with specific
                                // rank transitions across write and read combinations.
  UINT8   CmdStress;           // Indicate stress profile for the command bus traffic.
                                // 0 - LFSR victim aggressor
                                // 1 - CADB deselect victim
                                // aggressor LFSR write/read loopback test with specific
                                // rank transitions across write and read combinations.
  UINT8   Scrambler:1;        // Indicates if scrambler is on or off.
                                // 0 - off
                                // 1 - on
  UINT8   RsvdBits:7;         // Reserved
} BDAT_Margin_Config_Struc;

#pragma pack(pop)
```
5.8 Product-specific Data Schema

This schema exposes product-specific data for a memory subsystem.

```c
#pragma pack(push, 1)

///
/// Product-specific Schema GUID
/// {7ac3abc0-f996-4c52-a0f5-146eb0c8a7d4}
///
#define BDAT_PRODUCT_SPECIFIC_DATA_GUID\
{\
  0x7ac3abc0, 0xf996, 0x4c52, 0xa0, 0xf5, 0x14, 0x6e, 0xb0, 0xc8, 0xa7, 0xd4\
}

typedef struct {
  EFI_GUID ProductId; // A unique identifier for the product data being described.
  UINT8 Revision;     // 1 - Revision value of this structure
  UINT8 LengthInBytes; // Length in bytes of this structure, including ProductData[]
  // UINT8 ProductData[]; // The product-specific data as defined by the creator
  // of the ProductId structure.
} BDAT_PRODUCT_SPECIFIC_DATA_STRUCT;

#pragma pack(pop)
```
5.9 **Columnar Style Memory Schema 6**

Previous memory data schema 4 and 4B, RMT schema 4 and 5 allocated space for all the possible sockets, channels, dimms, ranks, and lanes based on the platform regardless whether the sockets, channels, dimms or ranks were populated or not. It could waste a lot of space. Moreover, the RMT results allocate space for all lanes regardless whether the per lane results were requested. In order to utilize the memory space more efficiently, provide the flexibilities of handle different types of results or data and enable a common parser for all schema, columnar style schema are introduced.

Columnar style schema have two sections: one for the metadata and one for the columnar data. The metadata section contains the key/value pairs. One of the key/value pair indicates the number of columnar entries. The columnar section contains data organized as rows and columns.

The columnar style schema have this GUID in the schema header structure.
```c
/// Columnar style schema GUID
/// {8F4E928-0F5F-46D4-8410-479FDA279DB6}
```

```c
#define COLUMNAR_RESULT_GUID

{ 0x8F4E928, 0xF5F, 0x46D4, 0x84, 0x10, 0x47, 0x9F, 0xDA, 0x27, 0x9D, 0xB6 }
```

The columnar results schema have the following header which defines the GUIDs, the size of the meta and row data and the size of row element and row count. The information in the header is important for the common parser.

```c
#pragma pack(push, 1)
typedef struct {
  UINT32  Revision;
  BOOLEAN TransferMode;
  struct {
    VOID     *Reserved;
    UINT32   MetadataSize;
    EFI_GUID MetadataType;
  } MdBlock;
  struct {
    VOID     *Reserved;
    EFI_GUID ResultType;
    UINT32   ResultElementSize;
    INT32    ResultCapacity;
    INT32    ResultElementCount;
  } RsBlock;
} COLUMNAR_RESULTS_DATA_HEAD_STRUCTURE;
#pragma pack(pop)
```

Following the header is the the meta data structure, then the result data rows.

To increase the flexibility of support different projects which different memory space requirement, the memory data are divided into 3 schema to store memory device info, training data and margin test results respectively. Individual project can choose to implement any combination of them.
The following diagram shows what the BDAT structure look like if all 3 schema are implemented.

![Diagram of BDAT structure]

5.9.1 RMT Schema 6

RMT schema 6 stores the RMT results.

NOTE: RMT schema 6 used in the Broxton.

5.9.1.1 RMT Schema 6 metadata

```c
#define BDATRMT_RESULT_METADATA_GUID \\
{0x02CB1552,0xD659,0x4232,{0xB5,0x1F,0xCA,0xB1,0x1F,0xCA,0x87} } 

#pragma pack (push, 1)
typedef struct BDATRMT_RESULT_METADATA{
  BOOLEAN EnableCtlAllMargin;
  UINT16 SinglesBurstLength;
  UINT32 SinglesLoopCount;
  UINT16 TurnaroundsBurstLength;
  UINT32 TurnaroundsLoopCount;
  SCRAMBLER_OVERRIDE_MODE ScramblerOverrideMode;
  UINT8 PiStepUnit[2]; // indexed as [fronstside=0/backside=1]
  UINT16 RxVrefStepUnit[2]; // indexed as [fronstside=0/backside=1]
};
```

BDAT Interface Specification
5.9.1.2 **RMT Schema 6 columns**

Each result element consists of: 1) a bit field structure header that describes the type and source of the corresponding margin data; and 2) eight unsigned 8-bit values representing margin parameter offsets. The 8 values are organized as 4 pairs with one element of the pair for the low side of the margin parameter and one for the high side. The rank margin covers more than four margin parameters so it requires multiple margin results elements. The lane and rank-to-rank turnaround margins only cover 4 margin parameters so they only require a single margin result element.

**5.9.1.2.1 Header:**

Header structure is a 32-bit bit mapped structure.

**ResultType:**

This bit field is the result type. The value occupies bits 0 through 2. The values are:

- 0 = RankResultType0
- 1 = RankResultType1
- 2 = LaneResultType
- 3 = TurnaroundResultType
- 4 = ParamLimitsResultType0
- 5 = ParamLimitsResultType1
- 6 = ParamLimitsResultType2
- 7 = RankResultType2

**Socket:**

This is the field is the zero based socket index. It occupies the bits 4 through 6.

**Controller:**

This is the field is the zero based memory controller index within a socket. It occupies the bits 7 through 8.

**Channel:**

This is the field is the zero based channel index within a memory controller. It occupies the bits 9 through 11.
**DimmA:**
This is the field is the zero based dimm index within a memory channel. It occupies the bit 12.

**RankA:**
This is the field is the zero based rank index within a dimm. It occupies the bits 13 through 15.

**DimmB:**
This is the field is the zero based dimm index within a memory channel. It occupies the bit 16.

**RankB:**
This is the field is the zero based rank index within a dimm. It occupies the bits 17 through 19.

**Lane:**
This is the field is the zero based lane index within a rank group. It occupies the bits 20 through 27.

**IoLevel:**
This is the field is the I/O level. It occupies the bit 28.

**IsNvM:**
This is the field indicates whether the data is for the NVMDIMM. It occupies the bit 29.

**Reserved:**
This is the reserved field. It occupies the bits 30 through 31.

**5.9.1.2.2 Data:**
This column is the margin parameter offsets. The value is an array of four structures where the structure contains two 8-bit unsigned integers. When the corresponding ResultType bit field is *ResultType[0,1,2], the structure values are the margin parameter’s last pass offsets. When the corresponding ResultType bit field is ParamLimits*ResultType[0,1,2], the structure values are the margin parameter’s limiting offsets. The first value in the structure is the magnitude of the low side of the corresponding margin parameter’s offset and the second value is the high side. When the corresponding ResultType bit field is *ResultType[0,1,2] and no failure is detected then the limiting margin parameter value is placed in the corresponding entry.
Memory Schemas

The distribution of margin parameter types within the array of structures depends on the ResultType bit field value as follows:

<table>
<thead>
<tr>
<th>Group=0</th>
<th>Group=1</th>
<th>Group=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index=0</td>
<td>RxDqsDelay</td>
<td>CmdAll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EridDelay lane=0</td>
</tr>
<tr>
<td>Index=1</td>
<td>TxDqDelay</td>
<td>CmdVref</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EridDelay lane=1</td>
</tr>
<tr>
<td>Index=2</td>
<td>RxVref</td>
<td>CtlAll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EridVref lane=0</td>
</tr>
<tr>
<td>Index=3</td>
<td>TxVref</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EridVref lane=1</td>
</tr>
</tbody>
</table>

The Group=0 margin parameter values apply when the corresponding ResultType bit field is RankResultType0, LaneResultType, TurnaroundResultType, or ParamLimitsResultType0.

The Group=1 margin parameter values apply when the corresponding ResultType bit field is RankResultType1 or ParamLimitsResultType1. The CtlAll values will be set to 0 if the EnableCtlAllMargin configuration parameter is FALSE.

The Group=2 margin parameter values apply when the corresponding ResultType bit field is RankResultType2 or ParamLimitsResultType2.

#define BDATRMT_RESULT_COLUMNS_GUID \ {0x87024B19,0xDA3B,0x420B,{0x92,0xC5,0xA6,0x20,0xB3,0x29,0x83} }

#pragma pack (push, 1)

struct RMT_RESULT_ROW_HEADER;

defined enum RMT_RESULT_TYPE{  
  RankResultType0 = 0,  
  RankResultType1 = 1,  
  LaneResultType = 2,  
  TurnaroundResultType = 3,  
  ParamLimitsResultType0 = 4,  
  ParamLimitsResultType1 = 5,  
  ParamLimitsResultType2 = 6,  
  RankRmtResultType2 = 7,  
  ResultTypeMax = 16,  
  RMT_RESULT_TYPE_DELIM = INT32_MAX
};
typedef enum RMT_RESULT_TYPE RMT_RESULT_TYPE;

struct RMT_RESULT_ROW HEADER{  
  UINT32 ResultType :14;  
  UINT32 Socket :3;  
  UINT32 Controller :2;  
  UINT32 Channel :3;  
  UINT32 DimmA :1;
typedef struct BDAT_RESULT_COLUMNS{
    struct RMT_RESULT_ROW_HEADER Header;
    UINT8 Margin[4][2];
} BDAT_RESULT_COLUMNS;

#pragma pack (pop)

5.9.1.2.3 Product specific RMT Schema 6 columns

Purley RMT column row header

#define RMT_RESULT_COLUMNS_GUID \
{0xDBBE487E,0xF3C1,0x475E,{0xB8,0xEA,0x69,0x88,0x40,0x07,0x7E,0x2F} }

#pragma pack (push, 1)

struct RMT_RESULT_ROW_HEADER;

enum RMT_RESULT_TYPE{
    Rank0RmtResultType = 0,
    Rank1RmtResultType = 1,
    LaneRmtResultType = 2,
    TurnaroundRmtResultType = 3,
    ParamLimits0ResultType = 4,
    ParamLimits1ResultType = 5,
    ParamLimits2ResultType = 6,
    Rank2RmtResultType = 7,
    RmtResultTypeMax = 8,
    RMT_RESULT_TYPE_DELIM = INT32_MAX
};

struct RMT_RESULT_ROW_HEADER{
    UINT32 ResultType :3;
    UINT32 Socket :3;
    UINT32 Controller :1;
    UINT32 Channel :2;
    UINT32 DimmA :2;
    UINT32 RankA :3;
    UINT32 DimmB :2;
   (UINT32 RankB :3;
    UINT32 Lane :7;
    UINT32 IoLevel :2;
    UINT32 IsDdrT :1;
    UINT32 Reserved :3;
};

#pragma pack (pop)
Whitley/Whitley_Copperlake/Jacobsville RMT column row header

```c
#define RMT_RESULT_COLUMNS_GUID \ 
{0xD98145F2,0x62F4,0x47CD,{0xAA,0xD1,0xDA,0x77,0x91,0xB2,0x77,0xF1} }

#pragma pack (push, 1)
struct RMT_RESULT_ROW_HEADER;
enum RMT_RESULT_TYPE{
    Rank0RmtResultType = 0,
    Rank1RmtResultType = 1,
    LaneRmtResultType = 2,
    TurnaroundRmtResultType = 3,
    ParamLimits0ResultType = 4,
    ParamLimits1ResultType = 5,
    ParamLimits2ResultType = 6,
    Rank2RmtResultType = 7,
    RmtResultTypeMax = 8,
    RMT_RESULT_TYPE_DELIM = MAX_INT32
};

struct RMT_RESULT_ROW_HEADER{
    UINT32 ResultType :3;
    UINT32 Socket :3;
    UINT32 Controller :3;
    UINT32 Channel :3;
    UINT32 DimmA :2;
    UINT32 RankA :3;
    UINT32 DimmB :2;
    UINT32 RankB :3;
    UINT32 Lane :1;
    UINT32 IoLevel :2;
    UINT32 IsDdrT :1;
};
#pragma pack (pop)
```

5.9.2 RMT Schema 6B

RMT schema 6B has the same metadata structure as that of RMT schema 6, but the column data structure was updated to add more result types, removed the IsNvm field.

NOTE: RMT schema 6 used in the CFL, CNL.

5.9.2.1 RMT Schema 6B metadata

It is the same as that of RMT schema 6.

5.9.2.2 RMT Schema 6B columns

Each result element consists of: 1) a bit field structure header that describes the type and source of the corresponding margin data; and 2) eight unsigned 8-bit values representing margin parameter offsets. The 8 values are organized as 4 pairs with one element of the pair for the low side of the margin parameter and one for the high side. The rank margin covers more than four margin parameters so it requires multiple
margin results elements. The lane and rank-to-rank turnaround margins only cover 4 margin parameters (RxDqs, TxDq, RxVref and TxVref) so they only require a single margin result element.

5.9.2.2.1 Header:

Header structure is a 32-bit bit mapped structure.

**ResultType:**
This bit field is the result type. The value occupies bits 0 through 4. The values are:
- 0 = RankResultType0
- 1 = RankResultType1
- 2 = RankResultType2
- 3 = RankResultType3
- 4 = ByteResultType
- 5 = LaneResultType
- 6 = TurnaroundResultType
- 7 = ParamLimitsResultType0
- 8 = ParamLimitsResultType1
- 9 = ParamLimitsResultType2
- 10 = ParamLimitsResultType3

**Socket:**
This is the field is the zero based socket index. It occupies the bits 5 through 7.

**Controller:**
This is the field is the zero based memory controller index within a socket. It occupies the bits 8 through 9.

**Channel:**
This is the field is the zero based channel index within a memory controller. It occupies the bits 10 through 12.

**DimmA:**
This is the field is the zero based dimm index within a memory channel. It occupies the bit 13.

**RankA:**
This is the field is the zero based rank index within a dimm. It occupies the bits 14 through 16.
**DimmB:**
This is the field is the zero based dimm index within a memory channel. It occupies the bit 17.

**RankB:**
This is the field is the zero based rank index within a dimm. It occupies the bits 18 through 20.

**Lane:**
This is the field is the zero based lane index within a rank group. It occupies the bits 21 through 28.

**IoLevel:**
This is the field is the I/O level. It occupies the bit 29.

**Reserved:**
This is the reserved field. It occupies the bits 30 through 31.

### 5.9.2.2.2 Data:
This column is the margin parameter offsets. The value is an array of four structures where the structure contains two 8-bit unsigned integers. When the corresponding ResultType bit field is $\text{Rmt}*\text{ResultType}[0,1,2,3]$, the structure values are the margin parameter’s last pass offsets. When the corresponding ResultType bit field is $\text{ParamLimits}*\text{ResultType}[0,1,2,3]$, the structure values are the margin parameter’s limiting offsets. The first value in the structure is the magnitude of the low side of the corresponding margin parameter’s offset and the second value is the high side. When the corresponding ResultType bit field is $\text{Rmt}*\text{ResultType}[0,1,2,3]$ and no failure is detected then the limiting margin parameter value is placed in the corresponding entry.

The distribution of margin parameter types within the array of structures depends on the ResultType bit field value as follows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Group=0</th>
<th>Group=1</th>
<th>Group=2</th>
<th>Group=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RxDqsDelay</td>
<td>CmdAll</td>
<td>ErIdDelay lane=0</td>
<td>RecEn</td>
</tr>
<tr>
<td>1</td>
<td>TxDqDelay</td>
<td>CmdVref</td>
<td>ErIdDelay lane=1</td>
<td>WrLvl</td>
</tr>
<tr>
<td>2</td>
<td>RxVref</td>
<td>CtlAll</td>
<td>ErIdVref lane=0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TxVref</td>
<td></td>
<td>ErIdVref lane=1</td>
<td></td>
</tr>
</tbody>
</table>
The Group=0 margin parameter values apply when the corresponding ResultType bit field is RankResultType0, ByteResultType, LaneResultType, TurnaroundResultType, or ParamLimitsResultType0.

The Group=1 margin parameter values apply when the corresponding ResultType bit field is RankResultType1 or ParamLimitsResultType1. The CtlAll values will be set to 0 if the EnableCtlAllMargin configuration parameter is FALSE.

The Group=2 margin parameter values apply when the corresponding ResultType bit field is RankResultType2 or ParamLimitsResultType2.

The Group=3 margin parameter values apply when the corresponding ResultType bit field is RankResultType3, or ParamLimitsResultType3.

#define RMT_RESULT_COLUMNS_GUID \ 
{0x0E60A1EB,0x331F,0x42A1,{0x9D,0xE7,0x45,0x3E,0x84,0x76,0x11,0x54} }

#pragma pack (push, 1)
struct RMT_RESULT_ROW_HEADER;
enum RMT_RESULT_TYPE{
    RankResultType0 = 0,
    RankResultType1 = 1,
    RankResultType2 = 2,
    RankResultType3 = 3,
    ByteResultType = 4,
    LaneResultType = 5,
    TurnaroundResultType = 6,
    ParamLimits0ResultType = 7,
    ParamLimits1ResultType = 8,
    ParamLimits2ResultType = 9,
    ParamLimits3ResultType = 10,
    ResultTypeMax = 31,
    RMT_RESULT_TYPE_DELIM = INT32_MAX
};

typedef enum RMT_RESULT_TYPE RMT_RESULT_TYPE;

struct RMT_RESULT_ROW_HEADER{
    UINT32 ResultType :5;
    UINT32 Socket :3;
    UINT32 Controller :2;
    UINT32 Channel :3;
    UINT32 DimmA :1;
    UINT32 RankA :3;
    UINT32 DimmB :1;
    UINT32 RankB :3;
    UINT32 Lane :8;
    UINT32 IoLevel :1;
    UINT32 Reserved :2;
};

typedef struct RMT_RESULT_ROW_HEADER RMT_RESULT_ROW_HEADER;

typedef struct RMT_RESULT_COLUMNS{
    struct RMT_RESULT_ROW_HEADER Header;
    UINT8 Margin[4][2];
}RMT_RESULT_COLUMNS;
5.10 **DIMM SPD RAW Data Schema 7**

This memory schema store the SPD raw data.

The SPD data structure header contains a GUID, total size in bytes and CRC. SPD data entries are packed by bytes in contiguous space after the header. Each entry contains a header that describes the entry type and entry size. The entry type structures define data type for each SPD data entry structure.

```c
#pragma pack(push, 1)
/// Memory SPD Data Schema GUID
/// {1B19F809-1D91-4F00-A3F3-7A676606D3B1}
///
#define BDAT_MEM_SPD_GUID \{
   0x1b19f809, 0x1d91, 0x4f00, { 0xa3, 0xf3, 0x7a, 0x67, 0x66, 0x6, 0x3, 0xb1 } \}
/// Memory SPD data identification GUID
/// {46F60B90-9C94-43CA-A77C-09B84899348}
///
#define MEM_SPD_DATA_ID_GUID { 0x46f60b90, 0x9c94, 0x43ca, { 0xa7, 0x7c, 0x9, 0xb8, 0x48, 0x99, 0x93, 0x48 } };
/// Memory SPD Data Header
///
typedef struct {
   EFI_GUID  MemSpdGuid;  /// GUID that uniquely identifies the memory SPD data revision
   UINT32    Size;       /// Total size in bytes including the header and all SPD data
   UINT32    Crc;        /// 32-bit CRC generated over the whole size minus this crc
   field
   /// Note: UEFI 32-bit CRC implementation (CalculateCrc32)
   /// Consumers can ignore CRC check if not needed.
   UINT32    Reserved;   /// Reserved for future use, must be initialized to 0
} MEM_SPD_RAW_DATA_HEADER;
/// Memory SPD Raw Data
///
typedef struct {
   MEM_SPD_RAW_DATA_HEADER  Header;

   /// This is a dynamic region, where SPD data entries are filled out.
   ///
} MEM_SPD_DATA_STRUCTURE
```
typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    MEM_SPD_DATA_STRUCTURE SpdData;
} BDAT_MEM_SPD_STRUCTURE;

/// List of all entry types supported by this revision of memory SPD data structure
///
typedef enum {
    MemSpdDataType0 = 0,
    MemTrainDataTypeMax,
    MemTrainDataTypeDelim = MAX_INT32
} MEM_SPD_DATA_TYPE;

/// Generic entry header for all memory SPD raw data entries
///
typedef struct {
    MEM_SPD_DATA_TYPE Type;
    UINT16 Size;     /// Entries will be packed by byte in contiguous space. Size of the entry includes the header.
} MEM_SPD_DATA_ENTRY_HEADER;

/// Structure to specify SPD dimm memory location
///
typedef struct {
    UINT8 Socket;
    UINT8 Channel;
    UINT8 Dimm;
} MEM_SPD_DATA_ENTRY_MEMORY_LOCATION;

/// Type 0: SPD RDIMM/LRDIMM DDR4 or DDR5
/// The NumberOfBytes are 512 and 1024 for DDR4 and DDR5 respectively.
///
typedef struct {
    MEM_SPD_DATA_ENTRY_HEADER Header;
    MEM_SPD_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    UINT16 NumberOfBytes;
    // This is a dynamic region, where SPD data are filled out.
    // The total number of bytes of the SPD data must match NumberOfBytes
    //
} MEM_SPD_ENTRY_TYPE0;

#pragma pack(pop)

5.11 Memory Training Data Schema 8

This memory schema store the memory training data.

The memory training data structure header contains a GUID, total size in bytes and CRC. Memory training data entries are packed by bytes in contiguous space after the header. Each entry contains a
header that describes the entry type and entry size. The entry type structures define data type for each memory training data entry structure.

```c
#pragma pack(push, 1)

/// Memory Training Data Schema GUID
/// {27AAB341-5EF9-4383-AE4D-091241B2FA0C }
///
#define BDAT_MEM_TRAINING_GUID \
{ \
  0x27aab341, 0x5ef9, 0x4383, 0xae, 0x4d, 0x9, 0x12, \
  0x41, 0xb2, 0xfa, 0xc \
}
```
Memory Schemas

/// Memory training data identification GUID
/// {37E839B5-4357-47D9-A13F-6F9A4333FBC4}
///
#define MEM_TRAINING_DATA_ID_GUID { 0x37e839b5, 0x4357, 0x47d9, { 0xa1, 0x3f, 0x6f, 0x9a, 0x43, 0x33, 0xfb, 0xc4 } };

/// Memory Training Data Header
///
typedef struct {
  EFI_GUID  MemDataGuid;  // GUID that uniquely identifies the memory training data revision
  UINT32    Size;        // Total size in bytes including the header and all training data
  UINT32    Crc;         // 32-bit CRC generated over the whole size minus this crc field
  UINT32    Reserved;    // Reserved for future use, must be initialized to 0
} MEM_TRAINING_DATA_HEADER;

/// Memory Training Data
///
typedef struct {
  MEM_TRAINING_DATA_HEADER  Header;
  // This is a dynamic region, where trainingd data entries are filled out.
  //
} MEM_TRAINING_DATA_STRUCTURE;

typedef struct {
  BDAT_SCHEMA_HEADER_STRUCTURE  SchemaHeader;
  MEM_TRAINING_DATA_STRUCTURE   TrainingData;
} BDAT_MEM_TRAINING_STRUCTURE;

/// List of all entry types supported by this revision of memory training data
///
typedef enum {
  MemTrainingDataType0 = 0,
  MemTrainingDataType1 = 1,
  MemTrainingDataType2 = 2,
  MemTrainingDataType3 = 3,
  MemTrainingDataType4 = 4,
  MemTrainingDataType5 = 5,
  MemTrainingDataTypePpin = 6,
  MemTrainingDataTypeBoardUuid = 7,
  MemTrainingDataTypeTurnaround = 8,
  MemTrainingDataTypeDcPmmTurnaround = 9,
  MemTrainingDataTypeDimmReg = 10,
  MemTrainingDataTypeMax,
  MemTrainingDataTypeDelim = MAX_INT32
} MEM_TRAINING_DATA_TYPE;
typedef struct {
    MEM_TRAINING_DATA_TYPE      Type;
    UINT16                      Size;    // Entries will be packed by byte in contiguous space
} MEM_TRAINING_DATA_ENTRY_HEADER;

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER           Header;
    UINT8                                    EccEnable;
    UINT8                                    MaxSocket;
    UINT8                                    MaxChannel;
    UINT8                                    MaxSubChannel;    // It is 1 if there is no sub-channel
    UINT8                                    MaxDimm;
    UINT8                                    MaxRank;
    UINT8                                    MaxStrobePerSubChannel;    // It is the MaxStrobe of the channel if there is no sub-channel
    UINT8                                    MaxBitsPerSubChannel;    // It is the MaxBits of the channel if there is no sub-channel
} MEM_DATA_ENTRY_TYPE0;

typedef struct {
    UINT8     Socket;
    UINT8     Channel;
    UINT8     SubChannel;    // Upper nibble is the pseudo channel if pseudo channel is applicable. Lower nibble is the subchannel.
    UINT8     Dimm;    // 0xFF = n/a
    UINT8     Rank;    // 0xFF = n/a Upper nibble is the subrank specified by CID signals. Lower nibble is the CS rank.
} MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION;

typedef enum {
    PerBitMemTrainData = 0,
    PerStrobeMemTrainData = 1,
    PerRankMemTrainData = 2,
    PerSubChannelMemTrainData = 3,
    PerChannelMemTrainData = 4,
    PerSubChannelSignalTrainData = 5,
    PerChannelSignalTrainData = 6,
}
Memory Schemas

```c
MemTrainDataScopeMax,
MemTrainDataScopeDelim = MAX_INT32
} MEM_TRAINING_DATA_SCOPE;

///
/// Struct to store signal and its data
///
typedef struct {
    MRC_GT Signal;
    INT16 Value;
} SIGNAL_DATA;

///
/// Type 1: General training data
///
typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    MRC_GT Level;
    MRC_GT Group;
    MEM_TRAINING_DATA_SCOPE Scope; // If Scope is PerSubChannelMemTrainData
    // or PerChannelMemTrainData, the
    // training data is applicable to whole
    // SubChannel or Channel regardless the
    // Dimm or Rank. The MemoryLoaction.Dimm
    // and MemoryLoaction.Rank should be
    // ignored.
    //
    // For PerSubChannelSignalTrainData and
    // PerChannelSignalTrainData, each
    // element is an instance of SIGNAL_DATA
    // struct.
    //
    UINT8 NumberOfElements;
    UINT8 SizeOfElement; // Number of bytes of each
    // training data element.
    // 1: UINT8
    // 2: UINT16
    // 4: UINT32
    // 6: SIGNAL_DATA

    // This is a dynamic region, where training data are filled out.
    // The total number of bytes of the training data must be equal to
    // NumberOfElements * SizeOfElement
    //}
} MEM_DATA_ENTRY_TYPE1;

///
/// Type 2: DRAM mode register data (deprecated)
///
typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    UINT8 NumberOfModeRegisters; // DDR5: 256
    UINT8 NumberOfDrams;
```
Memory Schemas

// This is a dynamic region, where DRAM mode register data are filled out.
// Each mode register data is one byte. The total number of bytes of the data must
// be equal to NumberOfModeRegisters * NumberOfDrams. The data is indexed as
// [ModeRegister][Dram]
//
} MEM_DATA_ENTRY_TYPE2;

//
// Type 3: RCD data 

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    UINT8 NumberOfRegisters;

    // This is a dynamic region, where RCD RW register data are filled out.
    // Each RW register data is one byte. The total number of bytes of the data must
    // be equal to NumberOfRegisters.
    // For DDR5, the data are ordered as: RW00-57; PG0RW60-7F; PG1RW60-7F; PG2RW60-7F;
    // PG3RW60-7F
}
} MEM_DATA_ENTRY_TYPE3;

//
// Type 4: Signal training data

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    MRC_LT Level;
    MEM_TRAINING_DATA_SCOPE Scope;// If Scope is PerSubChannelMemTrainData
    // or PerChannelMemTrainData, the
    // training data is applicable to whole
    // SubChannel or Channel regardless the
    // Dimm or Rank. The MemoryLocation.Dimm
    // and MemoryLocation.Rank should be
    // ignored.
    UINT8 NumberOfSignals; // Number of SIGNAL_DATA
        // struct

    // This is a dynamic region, where signal training data are filled out.
    // Each signal training data element is defined by a SIGNAL_DATA struct.
    // The total number of bytes of the training data must be equal to
    // NumberOfSignals * sizeof (SIGNAL_DATA)
}
} MEM_TRAINING_DATA_ENTRY_TYPE4;

//
// Type 5: IO latency, Round trip and IO Comp training data

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION MemoryLocation;
    MEM_TRAINING_DATA_SCOPE Scope;

    // This is a dynamic region, where IO latency, Round trip and IO Comp training data
    // are filled out.
        // Each training data element is defined by a SIGNAL_DATA struct.
        // The total number of bytes of the training data must be equal to
        // NumberOfSignals * sizeof (SIGNAL_DATA)
}
} MEM_TRAINING_DATA_ENTRY_TYPE5;
UINT8 IoLatency;
UINT8 RoundTrip;
UINT8 IoComp;
} MEM_TRAINING_DATA_ENTRY_TYPE5;

//
// Type 6: Processor ID which is 64bits.
//
typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    UINT8 Socket;
    UINT32 PpinHi;
    UINT32 PpinLo;
} MEM_TRAINING_DATA_ENTRY_TYPE6;

//
// Type 7: FRU (Field Replaceable Unit) ID.
//
typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    UINT8 FRUSerialNumber[FRUMAXSTRING];
} MEM_TRAINING_DATA_ENTRY_TYPE7;

//
// Type 8: Turnaround timing for DDR
// The turnaround time is defined as the number of DCLKs between corresponding CAS commands measured on the bus for a specific turnaround type.
//
/*@ Defined in PerSocketData.h */
typedef struct {
    UINT8 t_rrsg; // read to read, same bank, same rank, same DIMM
    UINT8 t_rwsg; // read to write, same bank, same rank, same DIMM
    UINT8 t_wrsg; // write to read, same bank, same rank, same DIMM
    UINT8 t_wrsr; // write to read, different bank, same rank, same DIMM
    UINT8 t_wrss; // write to write, different bank, same rank, same DIMM
    UINT8 t_wrds; // write to read, different subrank, same DIMM
    UINT8 t_rdrr; // read to write, different 3DS logical rank
} DDR_TURNAROUND_PER_CHANNEL;
Memory Schemas

```
UINT8 Socket;
UINT8 Channel;
DDR_TURNAROUND_PER_CHANNEL TurnaroundSettings;
} MEM_TRAINING_DATA_ENTRY_TYPE8;

// Type 9: Turnaround timing for DCPMM
// The turnaround time is defined as the number of DCLKs between corresponding CAS
// commands measured on the bus for a specific turnaround type.
//
/* Defined in PerSocketData.h
// DCPMM Turnaround Time
typedef struct {
    UINT8 t_rdrd_dd;  // read to read, different DIMM
    UINT8 t_wrrd_dd;  // write to read, different DIMM
    UINT8 t_rdrd_s;   // read to read, same DIMM
    UINT8 t_wrrd_s;   // write to read, same DIMM
    UINT8 t_rdwr_dd;  // read to write, different DIMM
    UINT8 t_rdwr_s;   // read to write, same DIMM
    UINT8 t_wrwr_dd;  // write to write, different DIMM
    UINT8 t_wrwr_s;   // write to write, same DIMM
    UINT8 t_gntrd_dd; // grant to read, different DIMM
    UINT8 t_gntrd_s;  // grant to read, same DIMM
    UINT8 t_gntwr_dd; // grant to write, different DIMM
    UINT8 t_gntwr_s;  // grant to write, same DIMM
    UINT8 t_rdrgnt_dd; // read to grant, different DIMM
    UINT8 t_rdrgnt_s; // read to grant, same DIMM
    UINT8 t_wrgrnt_dd; // write to grant, different DIMM
    UINT8 t_wrgrnt_s; // write to grant, same DIMM
    UINT8 t_gntgnt_dd; // grant to grant, different DIMM
    UINT8 t_gntgnt_s; // grant to grant, same DIMM
} DCPMM_TURNAROUND_PER_CHANNEL;
*/

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER Header;
    UINT8 Socket;
    UINT8 Channel;
    DCPMM_TURNAROUND_PER_CHANNEL TurnaroundSettings;
} MEM_TRAINING_DATA_ENTRY_TYPE9;

// Type 10: DRAM mode register, RCD and DB register data

typedef struct {
    UINT8  Page; // Page number of DDR5 or function number of DDR4
    // For direct access registers, Page value is 0.
    UINT8  Address; // Bit 8 is the CW bit. 0 - DRAM, 1 - RCD/DB
    // Bits[7:0] are the register address.
    // DDR5:
    // DRAM: 0x00-FF mapped to MR00-FF
    // RCD: 0x00-5F mapped to RW00-5F(direct access),
    // 0x60-7F mapped RW60-7F (page access)
    // DB: 0x80-DF mapped to RW80-DF(direct access),
    // 0xE0-FF mapped RWE0-FF (page access)
} MEM_TRAINING_DATA_ENTRY_TYPE10;
```
UINT8     NumberOfElement;  // The number of devices for a given rank with the
// specified address and page.
// DRAM: It is the number of DRAM devices for a given rank.
// RCD:  It is number of RCD for a given rank.
//       Its value is 1.
// DB:   It is the number of DB devices for a given rank.

// This is a dynamic region where register data of DRAM/RCD/DB devices are filled out.
// The register page and address are specifid in by the Page and Address fields.
// The NumberOfElement is the number of devices for a given rank.
// The size of each register data is one byte. The total number of bytes of the data
// must be equal to NumberOfElement.
// The data is indexed as [Element]
//
} DIMM_REG;

typedef struct {
    MEM_TRAINING_DATA_ENTRY_HEADER           Header;
    MEM_TRAINING_DATA_ENTRY_MEMORY_LOCATION  MemoryLocation;
    UINT16                                   NumberOfRegisters;
    UINT8                                    RegisterSize;

    // This is a dynamic region where DIMM(DRAM/RCD/DB) register data are filled out.
    // It includes the data of one or more registers.
    // RegisterSize is the size of DIMM_REG struct includes its dynamic region.
    // Each register entry is an instance of the DIMM_REG struct.
    // All registers in this dynamic region must have the same size.
    // The total number of bytes of the data must be equal to
    // NumberOfRegisters * RegisterSize.
    // The data is indexed as [Register]
    //
} MEM_TRAINING_DATA_ENTRY_TYPE10;

#pragma pack(pop)
Five schemas were added to support addition of PCIe data to the BDAT structure. These schemas are designed to support the wide variety of system topologies possible with PCIe while avoiding unnecessary data duplication or empty data fields whenever possible. This comes at the expense of the additional complexity of multiple schemas.

The PCIe Topology Schema, PCIe Lane Margin Schema and PCIe Port Margin Schema are designed to be as generic as possible such that they could be applied to any PCIe implementation. The PCIe Software Equalization Phase 2/3 schema and PCIe Software Equalization Score Schema are designed with the intention of being as generic as possible but unavoidably contains data that assumes the BIOS implements a PCIe software equalization algorithm like the one found in Haswell and Broadwell client BIOS.

The reason why software equalization data is broken into two separate schemas is because all the data needed to generate the Equalization Phase 2/3 schema is available on every boot since it is stored in NVRAM. The score data is only available if software equalization actually runs during that boot.

### 6.1 PCIe Topology Schema

The PCIe Topology schema contains high level data that indicates what the system topology is. This includes information about which root ports are enabled, which lanes are routed to which ports, and what endpoints are downstream. This information is derived from the bifurcation and lane reversal settings that the system is presently using.

```
#pragma pack(push, 1)
///
/// PCIe Topology Schema GUID
///
/// (436EC602-0D69-48C7-A8E6-AB50EA226B16)
///
#define BDAT_PCIE_TOPOLOGY_GUID \
{ 0x436EC602, 0xD69, 0x48C7, {0xA8, 0xE6, 0xAB, 0x50, 0xEA, 0x22, 0x6B, 0x16}\ }
#define BDAT_PCIE_MAX_LINK_WIDTH     32
///
/// Common Structure Definitions
///
typedef struct {
  UINT8   Bus;
  UINT8   Device;
  UINT8   Function;
  UINT8   Reserved;
} BDAT_PCI_DEVICE;

typedef union {
  UINT32   Data;
  struct {
    UINT16   DeviceId;
    UINT16   VendorId;
  } Ids;
} BDAT_PCI_DEVICE_ID;
```
typedef struct {
    UINT8 Major;
    UINT8 Minor;
    UINT8 Rev;
    UINT8 Build;
} BDAT_PCIE_CODE_VERSION;

typedef struct {
    BDAT_PCI_DEVICE RootPort;
    BDAT_PCI_DEVICE PhyPort;
    BDAT_PCI_DEVICE_ID EndpointId;
    UINT8 MaxLinkSpeed;
    UINT8 Reserved1;
    UINT8 Reserved2;
    UINT8 Reserved3;
} BDAT_PCIE_ROOT_PORT;

typedef struct {
    UINT8 PhysicalLane;
    UINT8 LogicalLane;
} BDAT_PCIE_LANE_TOPOLOGY;

typedef struct {
    BDAT_PCIE_ROOT_PORT Port;
    UINT8 LaneCount;
    UINT8 Reserved1;
    UINT8 Reserved2;
    UINT8 Reserved3;
    BDAT_PCIE_LANE_TOPOLOGY Lanes[BDAT_PCIE_MAX_LINK_WIDTH];
} BDAT_PCIE_PORT_TOPOLOGY;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    BDAT_PCIE_CODE_VERSION CodeVersion;
    UINT16 RootPortCount;
    UINT16 Reserved;
    BDAT_PCIE_PORT_TOPOLOGY Ports[RootPortCount];
} BDAT_PCIE_TOPOLOGY_STRUCTURE;
#pragma pack (pop)
6.2 PCIe Software Equalization Phase 2/3 Schema

The Software Equalization Phase 2/3 schema stores the link training values which the software equalization algorithm found to be optimal. The schema allows a wide variety of parameters to be optimized. The EqPhase parameter on the BDAT_PCIE_SWEQ_LANE_PHASE23 structure indicates which side of the link the optimized value is for using the following convention:

2 = Root Port Tx, Endpoint Rx side of link
3 = Endpoint Tx, Root Port Rx side of link

The valid flags inform the parser which parameters where optimized. If the valid flag is not set, then the value it corresponds to should be zero and ignored by the parser. Note that it may be possible for a single lane to have two array entries (one for phase 2, one for phase 3.) At time of writing, Haswell and Broadwell client only implement Phase 3 and only use the BestPreset parameter.

```c
#pragma pack(push, 1)
/// Software Equalization Phase 2/3 Schema GUID
/// {9268BE80-6FBC-4528-8D8E-F1ABDD72AE7F}
///
#define BDAT_PCIE_SWEQ_PHASE23_GUID
    
    0x9268BE80, 0x6FBC, 0x4528, {0x8D, 0x8E, 0xF1, 0xAB, 0xDD, 0x72, 0xAE, 0x7F}
}
typedef struct {
    UINT8 EqPhase;
    UINT8 PhysicalLane;
    UINT8 BestPresetValid;
    UINT8 BestCursorsValid;
    UINT8 BestCtleValid;
    UINT8 BestPreset;
    UINT8 BestPreCursor;
    UINT8 BestCursor;
    UINT8 BestPostCursor;
    UINT8 BestCtle;
    UINT8 Reserved1;
    UINT8 Reserved2;
} BDAT_PCIE_SWEQ_LANE_PHASE23;

typedef struct {
    UINT8 LaneCount;
    UINT8 Reserved1;
    UINT8 Reserved2;
    UINT8 Reserved3;
    BDAT_PCI_DEVICE PhyPort;
    BDAT_PCIE_SWEQ_LANE_PHASE23 BestTxEqs[BDAT_PCIE_MAX_LINK_WIDTH];
} BDAT_PCIE_SWEQ_PHY_PHASE23;
```
typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE   SchemaHeader;
    UINT16                         PhyPortCount;
    UINT16                         Reserved;
    BDAT_PCIE_SWEQ_PHY_PHASE23     PhyPorts[PhyPortCount];
} BDAT_PCIE_SWEQ_PHASE23_STRUCTURE;
#pragma pack (pop)
### 6.3 PCIe Software Equalization Score Schema

The score schema provides the “score” that the software equalization algorithm assigned to each tested TxEQ/CTLE. Software equalization will choose the TxEQ/CTLE that provides the best score on a per lane basis. This data provides visibility into how what the valuation was for each preset and insight into the decision made by SW EQ. It also contains the TxEQs/CTLE that Software Equalization selected as the best for Phase 2/3 of the equalization procedure. Note that while this schema covers all known possible optimizations, BIOS may not implement support for all of them. At time of writing Broadwell client BIOS only supports phase 3 preset optimization. The **Score field** of BDAT_PCIE_SWEQ_LANE_SCORE needs to be interpreted as a fixed decimal point number. For example, a value of 100 in this field translates to a value of 1.00, 153 would be 1.53 and 1 would be 0.01.

```c
#pragma pack(push, 1)
#endif
/// Software Equalization Score Schema GUID
/// {BA5A6B9F-3903-43F0-90C6-DD65413D08DA}
#define BDAT_PCIE_SWEQ_SCORE_GUID

typedef struct {
    UINT8                           PhysicalLane;
    UINT8                           Reserved;
    INT32                           Score;
} BDAT_PCIE_SWEQ_LANE_SCORE;

typedef struct {
    UINT8                           EqPhase;
    UINT8                           PresetValid;
    UINT8                           CursorsValid;
    UINT8                           CtleValid;
    UINT8                           Preset;
    UINT8                           PreCursor;
    UINT8                           Cursor;
    UINT8                           PostCursor;
    UINT8                           Ctle;
    UINT8                           LaneCount;
    UINT8                           Reserved1;
    UINT8                           Reserved2;
    BDAT_PCI_DEVICE                 PhyPort;
    BDAT_PCIE_SWEQ_LANE_SCORE       Lanes[BDAT_PCIE_MAX_LINK_WIDTH];
} BDAT_PCIE_SWEQ_SCORE;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE    SchemaHeader;
    UINT16                          ScoreCount;
    UINT16                          Reserved;
    BDAT_PCIE_SWEQ_SCORE            Scores[ScoreCount];
} BDAT_PCIE_SWEQ_SCORE_STRUCTURE;
#pragma pack (pop)
```

### 6.3.1 Fixed Decimal Point Parsing Sample Code

```c
void
```
PrintDecimalNumber (  
    IN INT32  Number  
  )  
{  
    INT32 FirstDigit;  
    INT32 SecondDigit;  
    INT32 Whole;  

    FirstDigit  = Number % 10;  
    SecondDigit = (Number / 10) % 10;  
    if (FirstDigit < 0) {  
        FirstDigit *= -1;  
    }  
    if (SecondDigit < 0) {  
        SecondDigit *= -1;  
    }  
    Whole = Number / 100;  

    printf ("%3d.%d%d", Whole, SecondDigit, FirstDigit);  

    return;  
}
6.4 PCIe Port Margin Schema

The port margin structure contains margin data that represents the worst case margin across all lanes assigned to a root port. An arbitrary number of margin data structures can be included. Each margin data structure contains a specific type of margin data (ex. Timing or Voltage), for a specific port. For example, a system with 3 root ports that reports both jitter tolerance and VOC data would provide 6 instances of the **BDAT_PCIE_PORT_MARGIN** structure; 2 for each root port (jitter and VOC) multiplied by 3 root ports. This allows the data structure to be extended for different margining techniques in the future without redefining the schema, one merely defines a new GUID for the new margining technique. For Jitter Tolerance data, the **LowSideMargin** field is unused and will always be zero. **The HighSideMargin and LowSideMargin fields should be interpreted as fixed decimal point numbers, using the same method used for the scores in the PCIe Software Equalization Score Schema.**

```c
#pragma pack(push, 1)
/// Port Margin Schema GUID
/// {D7154D12-03B2-4054-8CD2-9F4B2090BEF7}
#define BDAT_PCIE_PORT_MARGIN_GUID\
{ 0xD7154D12, 0x03B2, 0x4054, {0x8C, 0xD2, 0x9F, 0x4B, 0x20, 0x90, 0xBE, 0xF7}}

/// Jitter Tolerance Margin Type GUID
/// {B52A2E04-45FF-484e-B5FE-E478F5F6C9B}
#define JITTER_TOLERANCE_MARGIN_GUID\
{ 0xB52A2E04, 0x45FF, 0x484E, {0xB5, 0xFE, 0xEE, 0x47, 0x8F, 0x5F, 0x6C, 0x9B}}

/// VOC Margin Type GUID
/// {3578349A-9E98-4f70-91CB-E25B9899BC16}
#define VOC_MARGIN_GUID\
{ 0x3578349A, 0x9E98, 0x4F70, {0x91, 0xCB, 0xE2, 0x5B, 0x98, 0x99, 0xBC, 0x16}}

typedef struct {
    BDAT_PCI_DEVICE RootPort;
    EFI_GUID MarginType;
    INT32 HighSideMargin;
    INT32 LowSideMargin;
} BDAT_PCIE_PORT_MARGIN;

typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    UINT16 MarginCount;
    UINT16 Reserved;
    BDAT_PCIE_PORT_MARGIN Margins[MarginCount];
} BDAT_PCIE_PORT_MARGIN_STRUCTURE;
```
#pragma pack (pop)
### 6.5 PCIe Lane Margin Schema

The lane margin schema works in the same way as the port margin schema and uses the same margin type GUIDs. **Like the port margin schema, margin data is encoded as fixed decimal point.** Unlike the other schemas defined here, the implementation to generate this data in Broadwell Client BIOS is not provided to OEMs in reference code.

```c
#pragma pack(push, 1)

/// Lane Margin Schema GUID
///
/// {7AC0996D-A601-4210-944E-934E517B6C57}
///
#define BDAT_PCIE_LANE_MARGIN_GUID   
   {0x7AC0996D, 0xA601, 0x4210, 0x94, 0x4E, 0x93, 0x4E, 0x51, 0x7B, 0x6C, 0x57}

typedef struct {
    UINT8 LogicalLane;
    UINT8 Reserved1;
    UINT8 Reserved2;
    UINT8 Reserved3;
    INT32 HighSideMargin;
    INT32 LowSideMargin;
} BDAT_PCIE_LANE_MARGIN;
```

```c
typedef struct {
    BDAT_PCI_DEVICE RootPort;
    EFI_GUID MarginType;
    UINT8 LaneCount;
    UINT8 Reserved1;
    UINT8 Reserved2;
    UINT8 Reserved3;
    BDAT_PCIE_LANE_MARGIN Lanes[BDAT_PCIE_MAX_LINK_WIDTH];
} BDAT_PCIE_PORT_LANE_MARGIN;
```

```c
typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    UINT16 MarginCount;
    UINT16 Reserved;
    BDAT_PCIE_PORT_LANE_MARGIN Margins[MarginCount];
} BDAT_PCIE_LANE_MARGIN_STRUCTURE;
```

#pragma pack (pop)
7.1 eMMC Bus Margin Schema

The eMMC bus margin structure contains margin data that represents the margin result of an eMMC bus. An arbitrary number of margin data structures can be included. Each margin data structure contains a specific margin of margin data for a specific bus speed mode (ex. HS200, SR52) with a specific type of margin type (ex. Using CMD21 or Block IO).

```c
#pragma pack(push, 1)
/// eMMC Margin Schema GUID
/// {F6401DF9-2F7F-4079-A3AC-BA68E522769E}
#pragma pack(pop)
#define BDAT_EMMC_MARGIN_GUID
    {
        0xf6401df9, 0x2f7f, 0x4079, { 0xa3, 0xac, 0xba, 0xe5, 0x22, 0x76, 0x9e }
    }

/// EMMC OCR, CID, CSD and EXT_CSD data formats are defined in the eMMC JEDEC spec.
typedef struct {
    UINT32  OcrData;
} EMMC_OCR;

typedef struct {
    UINT32 CidData[4];
} EMMC_CID;

typedef struct {
    UINT32 CsdData[4];
} EMMC_CSD;

typedef struct {
    UINT8  ExtCsdData[512];
} EMMC_EXT_CSD;

typedef struct {
    BDAT_PCI_DEVICE                   Device;
    EMMC_OCR                          Ocr;
    EMMC_CID                          Cid;
    EMMC_CSD                          Csd;
    EMMC_EXT_CSD                      ExtCsd;
    UINT8                             BusWidth
    UINT8                             Reserved1;
    UINT8                             Reserved2;
    UINT8                             Reserved3;
} BDAT_EMMC_DEVICE_INFO;

typedef struct {
    UINT16                            Frequency;   // Mhz
    UINT8                             DataRate;    // 0: SDR, 1: DDR
    UINT8                             DriveStrength; // Ohms
```
typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    BDAT_EMMC_DEVICE_INFO DeviceInfo;
    UINT16 MarginCount;
    UINT8 LaneCount;
    UINT8 Reserved;
    BDAT_EMMC_MARGIN Margins[MarginCount];
} BDAT_EMMC_MARGIN_STRUCTURE;

#pragma pack (pop)
# EWL Schema

The EWL structure contains the Enhance Warning Log in Intel reference code. EWL structure header contains a GUID, total size in bytes, Offset of free space and CRC. Each EWL entry contains a header that describes the entry type and entry size. The entry type structures define data type for each EWL entry structure associated with each entry type.

The entry type definition can be found in the EWL specification document - "Enhance Warning Log in Intel Reference Code" located in server BIOS repo FDBin\Docs\Common folder.

```c
#pragma pack(push, 1)
/// Enhanced Warning Log Identification GUID
/// This GUID is used for HOB, UEFI variables, or UEFI Configuration Table as needed
/// by platform implementations
/// {D8E05800-005E-4462-AA3D-9C6B4704920B}
///
#define EWL_ID_GUID { 0xd8e05800, 0x5e, 0x4462, { 0xaa, 0x3d, 0x9c, 0x6b, 0x47, 0x4, 0x92, 0xb } };~

/// Enhanced Warning Log Revision GUID
/// Rev 1:  {75713370-3805-46B0-9FED-60F282486CFC}
///
#define EWL_REVISION1_GUID { 0x75713370, 0x3805, 0x46b0, { 0x9f, 0xed, 0x60, 0xf2, 0x82, 0x48, 0x6c, 0xfc } };~

/// Enhanced Warning Log Header
///
typedef struct {
    EFI_GUID  EwlGuid;  /// GUID that uniquely identifies the EWL revision
    UINT32    Size;     /// Total size in bytes including the header and buffer
    UINT32    FreeOffset; /// Offset of the beginning of the free space from byte 0
                       /// of the buffer immediately following this structure
                       /// Can be used to determine if buffer has sufficient space
                       /// for next entry
    UINT32    Crc;      /// 32-bit CRC generated over the whole size minus this crc
                       /// field
                       /// Note: UEFI 32-bit CRC implementation (CalculateCrc32)
                       /// (References [7])
                       /// Consumers can ignore CRC check if not needed.
    UINT32    Reserved; /// Reserved for future use, must be initialized to 0
} EWL_HEADER;

///
/// Enhanced Warning Log Spec defined data log structure
///
typedef struct {
    EWL_HEADER Header; /// The size will vary by implementation and should not
                       /// be assumed
    UINT8      Buffer[4 * 1024]; /// The spec requirement is that the buffer follow the
                       /// header
} EWL_PUBLIC_DATA;
```
typedef struct {
    BDAT_SCHEMA_HEADER_STRUCTURE SchemaHeader;
    EWL_PUBLIC_DATA WarningLogs;
} BDAT_EWL_STRUCTURE;

#pragma pack (pop)
### Appendix A – Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPI</td>
<td>Advanced Configuration and Power Interface</td>
</tr>
<tr>
<td>BIOS</td>
<td>Basic Input Output System</td>
</tr>
<tr>
<td>GUID</td>
<td>Globally Unique IDentifier(s)</td>
</tr>
<tr>
<td>ITP</td>
<td>In Target Probe – An Intel implementation of JTAG for Intel platforms</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>UEFI</td>
<td>Unified Extensible Firmware Interface</td>
</tr>
</tbody>
</table>