Runtime BIOS Fixups for ACPI Tables

Using the Intel iASL compiler

Revision 0.05

July 24, 2013

Contents

1 Introduction

BIOS ASL code is dynamic in nature, as BIOS must patch the ACPI DSDT and SSDT with values it determines during POST from querying HW and user options. There are code maintenance and performance issues that arise from the typical implementation. These issues are detailed in the AML Patching Issues section. Better implementations for these issues is given in the Modifying AML items in a loop section.

Fixups are typically done by searching AML in memory for ACPI Methods, Names, or placeholder strings. These items have a fixed location in the AML, so it is natural to change the memory searching algorithm to a loop that goes to the memory location of each item and performs the modification. This requires the BIOS build process to do some preprocessing on the ASL files so the offset of the ACPI items are output to a C header file for inclusion in C code. By moving to a preprocessing model, the fixup code becomes simplified and more efficient.

2 AML Patching Issues

2.1 Issue 1: Magic values and compiler error checking

By using magic values, it makes the iASL compiler find false errors in the resources when it does error checking. The –cr switch can be used to disable Resource checking, but this prevents iASL from finding errors while compiling ASL into AML. One of the tasks of a compiler is to find bugs. If ASL can be rewritten to avoid use of –cr, then that is the suggested path.

```
WORDBusNumber(            //Bus number resource (0
  ResourceProducer,       // bit 0 of general flags is 1
  MinFixed,               // Range is fixed
  MaxFixed,               // Range is fixed
  PosDecode,              // PosDecode
  0x4946,                 // Granularity    (FIX5 – patched during POST)
  0x3558,                 // Min
  0x003E,                 // Max
  0x0000,                 // Translation
  0x003F                  // Range Length = Max-Min+1
)
```

iASL conforms to the ACPI spec, so any items or values not compliant with the spec should be exposed as an error. The above Granularity value is ASCII for 'IF', and the Min value is ASCII for '5X', but using Magic Values is generally bad practice because it makes code maintenance more difficult, and is an indicator of bad design. The AML file will have 'FIX5' in memory, and examining every byte of DSDT will find FIX5 so that Granularity, Min, Max, and Range Length can be modified in BIOS code.

2.2 Issue 2: Scanning memory for a Magic Value signature

Scanning memory to find a Magic Value is inefficient and is a source of excessive execution time. This slows down boot time, so it is important to focus efforts on removing this issue in BIOS.

```
#define SIGNATURE_16(A, B)        ((A) | (B << 8))
#define SIGNATURE_32(A, B, C, D)  (SIGNATURE_16 (A, B) | (SIGNATURE_16 (C, D) << 16
))

CurrPtr = (UINT8 *) DsdtPointer;
for (; CurrPtr <= (DsdtPointer + ((ACPI_COMMON_HEADER *) DsdtPointer)->Length);
CurrPtr ++) {
  Signature = (UINT32 *) CurrPtr;
  switch (*Signature) {
    //
    // Fix ACPI Parameter "FIX0"
    //
```

```
    case (SIGNATURE_32 ('F', 'I', 'X', '0')):
```

It is more efficient to have a preprocessor determine the AML offsets of fixups, and then provide a table of offsets. Code can loop over the offset table and add the offsets to the base pointer on each iteration and modify the memory location as needed.

2.3 Issue 3: Adding values to pointer to modify items

Adding a value to a pointer to point to a different memory location requires one to add the correct value, otherwise you will point to the wrong memory location.

```
case (SIGNATURE_32 ('F', 'I', 'X', '1')):
  * (UINT16 *) DsdtPointer      = 0;
  *((UINT16 *) DsdtPointer + 1) = (UINT16)BusBase_Uncore[0];
  *((UINT16 *) DsdtPointer + 2) = (UINT16)BusBase_Uncore[0];
  *((UINT16 *) DsdtPointer + 4) = 1;
  break;
```

If the memory location is pointing to a structure, then a pointer to the structure should be used by casting the structure pointer to the memory location. This makes the compiler do the work of adding the correct values to the structure pointer when addressing different items in the structure. The solution is to use a structure cast so the compiler calculates the pointer arithmetic based on the structure format and data packing. Using a structure cast also helps document code, because then it's clear what items are being modified based on structure variable. The code above is not clear on what is being modified in memory.

3 iASL compiler and offset table generation

iASL 20130328 and later (download link) support the –so switch to generate a .offset.h file. The offset table must be generated before BIOS is built so that the .offset.h file is available for C code to #include. To achieve this, you provide iASL the file that gets turned into AML.

3.1 ASL operators for which iASL generates offsets

The iASL –so switch generates AML offsets for the following items:

- Name
- OperationRegion
- Method
- Package
- WORDBusNumber
- WORDIO
- DWORDMemory
- QWORDMemory

3.2 How to modify ASL file so iASL generates offsets

The following items don't require changes to ASL to fix iASL compile errors and generate offsets:

- Name
- OperationRegion
- Method
- Package

The following items require changes to ASL to fix iASL compile errors and generate offsets:

- WORDBusNumber
- WORDIO
- DWORDMemory
- QWORDMemory

Errors are fixed by changing values to conform to the ACPI spec, and populating the DescriptorName field with the fixup tag.

3.3 Sample DSDT ASL file: Platform.asl

Fixup tags are indicated with "FIXx – patched during POST" in the sample ASL file. The DescriptorName field of the ResourceTemplate operator is required for the iASL –so switch to calculate the AML offset.

```
DefinitionBlock ("Platform.asl","DSDT",2,"Company","PLATFORM",3)
{
  //
  // Current P-State & T-State limit
  //
  Name(\PSTE, 0) // Current P-State limit for _PPC()
  Name(\TSTE, 0) // Current T-State limit for _TPC()
```

```
  Name(\PETE, 0) // P-States enabled + T-States enabled (OS called _PPC/_TPC)

  //
  // Fixup for Uncore Bus number
  //
  Name (\BBU0, 0x50584946)    // (FIXP - patched during POST)

  //
  // Access CMOS range
  //
  OperationRegion (ACMS, SystemIO, 0x72, 2)
  Field ( ACMS, ByteAcc, NoLock, Preserve) {
    INDX, 8,
    DATA, 8
  }

  OperationRegion (PSYS, SystemMemory, 0x30584946, 0x400) // (FIX0 - patched during
POST)

  Scope (\_SB) {
    Name (_CRS, ResourceTemplate() {
      WORDBusNumber (     // Bus number resource (0)
        ResourceProducer, // bit 0 of general flags is 1
        MinFixed,         // Range is fixed
        MaxFixed,         // Range is fixed
        PosDecode,        // PosDecode
        0x0000,           // Granularity (FIX1 - patched during POST)
        0x0000,           // Min         (FIX1 - patched during POST)
        0x0000,           // Max         (FIX1 - patched during POST)
        0x0000,           // Translation
        0x0001,           // Range Length = Max-Min+1 (FIX1 - patched during POST)
              ,
              ,
        FIX1              // DescriptorName populated so iASL outputs offset
      )
    })

    Name (P0RS, ResourceTemplate() {
      WORDIO(               // Consumed-and-produced resource (all I/O above CFF)
        ResourceProducer,   // bit 0 of general flags is 0
        MinFixed,           // Range is fixed
        MaxFixed,           // Range is fixed
        PosDecode,
        EntireRange,
        0x0000,             // Granularity  (FIX6 - patched during POST)
        0x0000,             // Min          (FIX6 - patched during POST)
```

```
            0x0000,             // Max          (FIX6 - patched during POST)
            0x0000,             // Translation
            0x0001,             // Range Length (FIX6 - patched during POST)
                  ,
                  ,
            FIX6                // DescriptorName populated so iASL outputs offset
          )

          DWORDMemory(          // Consumed-and-produced resource(all of memory space)
            ResourceProducer,   // bit 0 of general flags is 0
            PosDecode,          // positive Decode
            MinFixed,           // Range is fixed
            MaxFixed,           // Range is fixed
            NonCacheable,
            ReadWrite,
            0x00000000,         // Granularity (FIX7 - patched during POST)
            0x00000000,         // Min (FIX7 - patched during POST)
            0x00000000,         // Max (FIX7 - patched during POST)
            0x00000000,         // Translation
            0x00000000,         // Range Length (FIX7 - patched during POST)
                      ,
                      ,
            FIX7                // DescriptorName populated so iASL outputs offset
          )

          QWORDMemory(          // Consumed-and-produced resource(all of memory space)
            ResourceProducer,   // bit 0 of general flags is 0
            PosDecode,          // positive Decode
            MinFixed,           // Range is fixed
            MaxFixed,           // Range is fixed
            NonCacheable,
            ReadWrite,
            0x00000000000,      // Granularity (FIX8 - patched during POST)
            0x00000000000,      // Min (FIX8 - patched during POST)
            0x00000000000,      // Max (FIX8 - patched during POST)
            0x00000000000,      // Translation
            0x00000000000,      // Range Length (FIX8 - patched during POST)
                        ,
                        ,
            FIX8                // DescriptorName populated so iASL outputs offset
          )
        })

    } // end of _SB
} // end of DSDT
```

3.3.1 Windows batch file to create .offset.h file for Platform.asl

The iASL –so switch extracts all Name, OperationRegion, Method, WORDBusNumber, WORDIO, DWORDMemory, QWORDMemory and Package operators and puts their AML offset into the .offset.h file. BIOS performs fixup on a smaller set of items, so the .offset.h file must be processed to filter any items not required. The .offset.h file for Platform.asl file is created and filtered with the batch file below.

```
@echo off
iasl.exe -oi -so Platform.asl
if %ERRORLEVEL% NEQ 0 (
  echo ERROR: iASL failed when creating Platform.offset.h
  goto :EOF
)
if not exist Platform.offset.h (
  echo ERROR: iASL failed to create Platform.offset.h
  goto :EOF
)

REM Move Platform.offset.h to use temp filename so it can be filtered.
move /y Platform.offset.h Platform.offset.h.tmp

REM Filter out everything except tool info, typedef, and start of structure
definition.
REM When search string has " in it, must use ^" to indicate string quote for
command processor, and \" as quote character in string.
findstr /v ^"{\" {NULL, };" Platform.offset.h.tmp > Platform.offset.h

REM Filter out everything except the AML operators that need fixup..
REM When search string has " in it, must use ^" to indicate string quote for
command processor, and \" as quote character in string.
findstr /r ^"\"PSYS\" .MCTL\" .FIX[0-9,A-Z] BBI[0] BBU[0]" Platform.offset.h.tmp >>
Platform.offset.h

REM Filter out everything except structure terminator and end of structure
definition.
findstr "{NULL, };" Platform.offset.h.tmp >> Platform.offset.h

REM Delete intermediate files
del Platform.offset.h.tmp
```

3.3.1.1 findstr, first execution

The first findstr filters everything except the file comments, offset structure definition, and start of the offset table structure.

3.3.1.2 findstr, second execution

The second findstr filters everything except the fixup items. This makes the table reflect only what will be touched, which means the loop over the offset table will only examine items that will be modified.

3.3.1.3 findstr, third execution

The third findstr filters everything except the table terminator and end of structure.

3.3.2 Platform.offset.h

The .offset.h file has a #ifndef around the offset structure definition so if you preprocess multiple ASL files so you can include their .offset.h file into the same .c file, you don't get a structure redefinition error. The offset table variable name is created from the DSDT DefinitionBlock() operator fields of TableSignature and TableID; this assures that all offset variable names are unique.

```
/*
 *
 * Intel ACPI Component Architecture
 * ASL Optimizing Compiler version 20130626-32 [Jul 24 2013]
 * Copyright (c) 2000 - 2013 Intel Corporation
 *
 * Compilation of "D:/bios/AcpiTables/Dsdt/SRPPlatform.iiii" - Wed Jul 24 14:06:23
2013
 *
 */
#ifndef __AML_OFFSET_TABLE_H
#define __AML_OFFSET_TABLE_H

typedef struct {
    char                    *Pathname;      /* Full pathname (from root) to the
object */
    unsigned short          ParentOpcode;   /* AML opcode for the parent object */
    unsigned long           NamesegOffset;  /* Offset of last nameseg in the parent
namepath */
    unsigned char           Opcode;         /* AML opcode for the data */
    unsigned long           Offset;         /* Offset for the data */
    unsigned long long      Value;          /* Original value of the data (as
applicable) */
} AML_OFFSET_TABLE_ENTRY;

#endif /* __AML_OFFSET_TABLE_H */

/*
 * Information specific to the supported object types:
 *
 * Integers:
 *    Opcode is the integer prefix, indicates length of the data
 *        (One of: BYTE, WORD, DWORD, QWORD, ZERO, ONE, ONES)
 *    Offset points to the actual integer data
 *    Value is the existing value in the AML
 *
 * Packages:
 *    Opcode is the package or var_package opcode
```

```
 *    Offset points to the package opcode
 *    Value is the package element count
 *
 * Operation Regions:
 *    Opcode is the address integer prefix, indicates length of the data
 *    Offset points to the region address
 *    Value is the existing address value in the AML
 *
 * Control Methods:
 *    Offset points to the method flags byte
 *    Value is the existing flags value in the AML
 *
 * Processors:
 *    Offset points to the first byte of the PBlock Address
 *
 * Resource Descriptors:
 *    Opcode is the descriptor type
 *    Offset points to the start of the descriptor
 */
AML_OFFSET_TABLE_ENTRY   DSDT_PLATFSRP_OffsetTable[] =
{

    {"BBI0",                      0x0008, 0x00000234, 0x0C, 0x00000239,
0x000000004C584946}, /* INTEGER */
    {"BBU0",                      0x0008, 0x0000023E, 0x0C, 0x00000243,
0x0000000050584946}, /* INTEGER */
    {"PSYS",                      0x5B80, 0x000002D1, 0x0C, 0x000002D7,
0x0000000030584946}, /* OPERATIONREGION */
    {"_SB_.UNC0.FIX1",            0x0011, 0x00000000, 0x88, 0x00001428,
0x0000000000000000}, /* WORDBUSNUMBER */
    {"_SB_.IIO0.FIX5",            0x0011, 0x00000000, 0x88, 0x00003D03,
0x0000000000000000}, /* WORDBUSNUMBER */
    {"_SB_.IIO0.FIX6",            0x0011, 0x00000000, 0x88, 0x00003D2B,
0x0000000000000000}, /* WORDIO */
    {"_SB_.IIO0.FIX7",            0x0011, 0x00000000, 0x87, 0x00003D6F,
0x0000000000000000}, /* DWORDMEMORY */
    {"_SB_.IIO0.FIX8",            0x0011, 0x00000000, 0x8A, 0x00003D89,
0x0000000000000000}, /* QWORDMEMORY */
    {"_SB_.IIO0.DMI0.MCTL",       0x5B80, 0x00004024, 0x0C, 0x0000402A,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR1A.MCTL",       0x5B80, 0x000047F7, 0x0C, 0x000047FD,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR1B.MCTL",       0x5B80, 0x0000502A, 0x0C, 0x00005030,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR2A.MCTL",       0x5B80, 0x00005886, 0x0C, 0x0000588C,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR2B.MCTL",       0x5B80, 0x000060BC, 0x0C, 0x000060C2,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR2C.MCTL",       0x5B80, 0x000068F2, 0x0C, 0x000068F8,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR2D.MCTL",       0x5B80, 0x00007128, 0x0C, 0x0000712E,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR3A.MCTL",       0x5B80, 0x00007984, 0x0C, 0x0000798A,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR3B.MCTL",       0x5B80, 0x000081BA, 0x0C, 0x000081C0,
0x0000000054584946}, /* OPERATIONREGION */
```

```
    {"_SB_.IIO0.BR3C.MCTL",        0x5B80, 0x00008A16, 0x0C, 0x00008A1C,
0x0000000054584946}, /* OPERATIONREGION */
    {"_SB_.IIO0.BR3D.MCTL",        0x5B80, 0x00009272, 0x0C, 0x00009278,
0x0000000054584946}, /* OPERATIONREGION */
    {NULL,0,0,0,0,0} /* Table terminator */
};
```

3.4 Sample SSDT ASL file: CpuPm.asl

Fixup tags are indicated with "FIXx – patched during POST" in the sample ASL file. The DescriptorName field of the ResourceTemplate operator is required for the iASL –so switch to calculate the AML offset.

```
DefinitionBlock ("CPUPM.asl", "SSDT", 2, "Company", "SSDT__PM", 0x4000)
{
  External(PSEN, FieldUnitObj)
  External(\_SB.SCK0.CP00, DeviceObj)

  Scope(\_SB.SCK0.CP00)
  {
    Name(TYPE, 0x80000000)
    Name(NCPU, 0x80000000)
    Name(DOMN, 0x80000000)

    Name (PSDC, Package()
    {
      //                              DOMN            NCPU
      Package(){0x00000005, 0x00000000, 8, 0x000000FC, 0x24}
    })

    Name(ZPSS, Package() {
      Package(){0, 0, 0, 0, 0, 0}
    })

    Name(NPSS,Package() {
      Package(){0x80000000, 0x80000000, 0x80000000, 0x80000000, 0x80000000,
0x80000000},  // P0
      Package(){0x80000000, 0x80000000, 0x80000000, 0x80000000, 0x80000000,
0x80000000},  // P1
      Package(){0x80000000, 0x80000000, 0x80000000, 0x80000000, 0x80000000,
0x80000000},  // P2
      Package(){0x80000000, 0x80000000, 0x80000000, 0x80000000, 0x80000000,
0x80000000},  // P3
    })

    //
    // Performance States Supported
```

```
    //
    Method(_PSS,0)
    {
      If (PSEN)
      {
        Return(NPSS)  // Return native P-states table
      }
      return(ZPSS)    // P-states are not available, so return package with 1
element of value 0
    }
  } // end of _SB
} // end of SSDT
```

3.4.1 Windows batch file to create .offset.h file for CpuPm.asl

The iASL –so switch extracts all Name, OperationRegion, Method, WORDBusNumber, WORDIO, DWORDMemory, QWORDMemory and Package operators and puts their AML offset into the .offset.h file. BIOS performs fixup on a smaller set of items, so the .offset.h file must be processed to filter any items not required. The .offset.h file for Platform.asl file is created and filtered with the batch file below.

```
@echo off
iasl.exe -oi -so CpuPm.asl
if %ERRORLEVEL% NEQ 0 (
  echo ERROR: iASL failed when creating CpuPm.offset.h
  goto :EOF
)
if not exist CpuPm.offset.h (
  echo ERROR: iASL failed to create CpuPm.offset.h
  goto :EOF
)

REM Move CpuPm.offset.h to use temp filename so it can be filtered.
move /y CpuPm.offset.h CpuPm.offset.h.tmp

REM Filter out everything except tool info, typedef, and start of structure
definition.
REM When search string has " in it, must use ^" to indicate string quote for
command processor, and \" as quote character in string.
findstr /v ^"{\" {NULL, };" CpuPm.offset.h.tmp > CpuPm.offset.h

REM Filter out everything except the AML operators that need fixup.
REM When search string has " in it, must use ^" to indicate string quote for
command processor, and \" as quote character in string.
findstr /r "DOMN NCPU PSDC NPSS" CpuPm.offset.h.tmp >> CpuPm.offset.h
```

```
REM Filter out everything except structure terminator and end of structure
definition.
findstr "{NULL, };" CpuPm.offset.h.tmp >> CpuPm.offset.h

REM Delete intermediate files
del CpuPm.offset.h.tmp
```

3.4.1.1 findstr, first execution

The first findstr filters everything except the file comments, offset structure definition, and start of the offset table structure.

3.4.1.2 findstr, second execution

The second findstr filters everything except the fixup items. This makes the table reflect only what will be touched, which means the loop over the offset table will only examine items that will be modified.

3.4.1.3 findstr, third execution

The third findstr filters everything except the table terminator and end of structure.

3.4.2 CpuPm.offset.h

The .offset.h file has a #ifndef around the offset structure definition so if you preprocess multiple ASL files so you can include their .offset.h file into the same .c file, you don't get a structure redefinition error. The offset table variable name is created from the DSDT DefinitionBlock() operator fields of TableSignature and TableID; this assures that all offset variable names are unique.

```
/*
 *
 * Intel ACPI Component Architecture
 * ASL Optimizing Compiler version 20130626-32 [Jul 24 2013]
 * Copyright (c) 2000 - 2013 Intel Corporation
 *
 * Compilation of "D:/bios/AcpiTables/Ssdt/CpuPm.iiii" - Wed Jul 24 14:06:23 2013
 *
 */
#ifndef __AML_OFFSET_TABLE_H
#define __AML_OFFSET_TABLE_H

typedef struct {
    char                   *Pathname;      /* Full pathname (from root) to the
object */
    unsigned short         ParentOpcode;   /* AML opcode for the parent object */
    unsigned long          NamesegOffset;  /* Offset of last nameseg in the parent
namepath */
    unsigned char          Opcode;         /* AML opcode for the data */
    unsigned long          Offset;         /* Offset for the data */
```

```
    unsigned long long     Value;          /* Original value of the data (as
applicable) */
} AML_OFFSET_TABLE_ENTRY;

#endif /* __AML_OFFSET_TABLE_H */

/*
 * Information specific to the supported object types:
 *
 * Integers:
 *    Opcode is the integer prefix, indicates length of the data
 *        (One of: BYTE, WORD, DWORD, QWORD, ZERO, ONE, ONES)
 *    Offset points to the actual integer data
 *    Value is the existing value in the AML
 *
 * Packages:
 *    Opcode is the package or var_package opcode
 *    Offset points to the package opcode
 *    Value is the package element count
 *
 * Operation Regions:
 *    Opcode is the address integer prefix, indicates length of the data
 *    Offset points to the region address
 *    Value is the existing address value in the AML
 *
 * Control Methods:
 *    Offset points to the method flags byte
 *    Value is the existing flags value in the AML
 *
 * Processors:
 *    Offset points to the first byte of the PBlock Address
 *
 * Resource Descriptors:
 *    Opcode is the descriptor type
 *    Offset points to the start of the descriptor
 */
AML_OFFSET_TABLE_ENTRY   SSDT_SSDT__PM_OffsetTable[] =
{

    {"_SB_.SCK0.CP00.NCPU",        0x0008, 0x00000C23, 0x0C, 0x00000C28,
0x0000000080000000}, /* INTEGER */
    {"_SB_.SCK0.CP00.DOMN",        0x0008, 0x00000C2D, 0x0C, 0x00000C32,
0x0000000080000000}, /* INTEGER */
    {"_SB_.SCK0.CP00.PSDC",        0x0008, 0x00000C56, 0x12, 0x00000C5A,
0x0000000000000001}, /* VAR_PACKAGE */
```

```
    {"_SB_.SCK0.CP00.PSDD",        0x0008, 0x00000C6B, 0x12, 0x00000C6F,
0x0000000000000001}, /* VAR_PACKAGE */
    {"_SB_.SCK0.CP00.PSDE",        0x0008, 0x00000C80, 0x12, 0x00000C84,
0x0000000000000001}, /* VAR_PACKAGE */
    {"_SB_.SCK0.CP00.NPSS",        0x0008, 0x00000ED9, 0x12, 0x00000EDD,
0x0000000000000010}, /* VAR_PACKAGE */
    {NULL,0,0,0,0,0} /* Table terminator */
};
```

4 Use .offset.h files in C code to modify AML items in a loop

4.1 #include .offset.h file

First you must include a header file from the ACPICA release that has AML structure definitions.

```
// from https://acpica.org/downloads/source_code.php
// file acpica-win-20130626.zip/source/include
#include "amlresrc.h"
```

Next you must include the .offset.h file.

```
#include "Platform.offset.h"
#include “CpuPm.offset.h”
```

You can also declare pointers to the offset tables so the patching code is generic, instead of referencing the offset variable name. This makes the patching code easier to maintain.

```
AML_OFFSET_TABLE_ENTRY    *mDsdtAmlOffsetTablePointer = &DSDT_PLATFORM_OffsetTable;
AML_OFFSET_TABLE_ENTRY    *mSsdtAmlOffsetTablePointer = &SSDT_SSDT__PM_OffsetTable;
```

4.2 WORDBusNumber, WORDIO

DSDT patching code requires an AML structure typecast for WORDBusNumber and WORDIO (16-bit).

```
AML_RESOURCE_ADDRESS16 *AmlResourceAddress16Pointer;
```

4.3 DWORDMemory

DSDT patching code requires an AML structure typecast for DWORDMemory (32-bit).

```
AML_RESOURCE_ADDRESS32 *AmlResourceAddress32Pointer;
```

4.4 QWORDMemory

DSDT patching code requires an AML structure typecast for QWORDMemory (64-bit).

```
AML_RESOURCE_ADDRESS64 *AmlResourceAddress64Pointer;
```

4.5 Name, OperationRegion

Name and OperationRegion operators have their fixup tag as part of the operator, and are the AmlValue item in the offset table.

```
Name (\BBU0, 0x50584946)    // (FIXP - patched during POST)
OperationRegion (PSYS, SystemMemory, 0x30584946, 0x400) // (FIX0 - patched during
POST)
```

4.6 Method

Method operators don't have fixup tags. The Offset item in the offset table points to the location in AML of the last 4 characters of the Pathname. This makes it quick to change the Method name, or change values after the Method by casting the pointer to a structure defined in amlresrc.h.

4.7 Package

Package operators don't have fixup tags. The Offset item in the offset table points to the location in AML of the package opcode. If you want to change package data, then your code must parse the AML packages.

4.8 Sample code

Incorporating all of the above changes results in a more efficient (and thus faster boot time) and easier to maintain DSDT and SSDT patch functions.

This example is for an EDKII BIOS because it uses AsciiStrLen() to get the length of the Pathname field, and other string operation functions.

```
#define SIGNATURE_16(A, B)        ((A) | (B << 8))
#define SIGNATURE_32(A, B, C, D)  (SIGNATURE_16 (A, B) | (SIGNATURE_16 (C, D) << 16
))

#include "Platform.offset.h"
#include “CpuPm.offset.h”

EFI_STATUS
PatchDsdtTable (
  IN OUT  EFI_ACPI_DESCRIPTION_HEADER *TableHeader
  )
{
  UINT8                   *DsdtPointer;
  UINT32                  i;
  UINT32                  *Signature;
  AML_OFFSET_TABLE_ENTRY  *mDsdtAmlOffsetTablePointer = &DSDT_PLATFORM_OffsetTable;
```

```
  //
  // Loop through the AML looking for values that we must fix up.
  //
  for (i = 0; mDsdtAmlOffsetTablePointer[i].Pathname != 0; i++) {
    //
    // Point to offset in DSDT for current item in AmlOffsetTable.
    //
    DsdtPointer = (UINT8 *) (TableHeader) + mDsdtAmlOffsetTablePointer [i].Offset;

    if (mDsdtAmlOffsetTablePointer [i].Opcode == AML_DWORD_PREFIX) {
      //
      // If AmlOpcode is 0x0C, then operator is Name() or OperationRegion().
      //

      //
      // AmlOffsetTable.Value has FIX tag, so check that to decide what to modify.
      //
      Signature = (UINT32 *) (&mDsdtAmlOffsetTablePointer [i].Value);
      switch (*Signature) {
        case (SIGNATURE_32 ('F', 'I', 'X', '0')):
          * (UINT32 *) DsdtPointer = (UINT32) (UINTN) mAcpiParameter;
          break;
      }
    } else if (mDsdtAmlOffsetTablePointer [i].Opcode == AML_INDEX_OP) {
      //
      // If AmlOpcode is 0x88, then operator is WORDBusNumber() or WORDIO().
      // DsdtPointer must be cast to AML_RESOURCE_ADDRESS16 to change values.
      //
      AmlResourceAddress16Pointer = (AML_RESOURCE_ADDRESS16 *) (DsdtPointer);

      //
      // Last 4 chars of AmlOffsetTable.Pathname has FIX tag.
      //
      Signature = (UINT32 *) (mDsdtAmlOffsetTablePointer [i].Pathname +
AsciiStrLen(mDsdtAmlOffsetTablePointer [i].Pathname) - 4);
      switch (*Signature) {
        case (SIGNATURE_32 ('F', 'I', 'X', '1')):
          AmlResourceAddress16Pointer->Granularity = 0;
          AmlResourceAddress16Pointer->Minimum = (UINT16)BusBase_Uncore[0];
          AmlResourceAddress16Pointer->Maximum = (UINT16)BusBase_Uncore[0];
          AmlResourceAddress16Pointer->AddressLength = 1;
          break;
      }
    } else if (mDsdtAmlOffsetTablePointer [i].Opcode == AML_SIZE_OF_OP) {
      //
```

```
      // If AmlOpcode is 0x87, then operator is DWORDMemory().
      // DsdtPointer must be cast to AML_RESOURCE_ADDRESS32 to change values.
      //
      AmlResourceAddress32Pointer = (AML_RESOURCE_ADDRESS32 *) (DsdtPointer);

      //
      // Last 4 chars of AmlOffsetTable.Pathname has FIX tag.
      //
      Signature = (UINT32 *) (mDsdtAmlOffsetTablePointer [i].Pathname +
AsciiStrLen(mDsdtAmlOffsetTablePointer [i].Pathname) - 4);
      switch (*Signature) {
        case (SIGNATURE_32 ('F', 'I', 'X', '7')):
          AmlResourceAddress32Pointer->Granularity = 0;
          if (MemLimit32_Iio[0] > MemBase32_Iio[0]) {
            AmlResourceAddress32Pointer->Minimum = (UINT32) MemBase32_Iio[0];
            AmlResourceAddress32Pointer->Maximum = (UINT32) MemLimit32_Iio[0];
            AmlResourceAddress32Pointer->AddressLength = (UINT32)
(MemLimit32_Iio[0] - MemBase32_Iio[0] + 1);
          }
          break;
      }
    } else if (mDsdtAmlOffsetTablePointer [i].Opcode == AML_CREATE_DWORD_FIELD_OP)
{
      //
      // If AmlOpcode is 0x8A, then operator is QWORDMemory().
      // DsdtPointer must be cast to AML_RESOURCE_ADDRESS64 to change values.
      //
      AmlResourceAddress64Pointer = (AML_RESOURCE_ADDRESS64 *) (DsdtPointer);

      //
      // Last 4 chars of AmlOffsetTable.Pathname has FIX tag.
      //
      Signature = (UINT32 *) (mDsdtAmlOffsetTablePointer [i].Pathname +
AsciiStrLen(mDsdtAmlOffsetTablePointer [i].Pathname) - 4);
      switch (*Signature) {
        case (SIGNATURE_32 ('F', 'I', 'X', '8')):
          AmlResourceAddress64Pointer->Granularity = 0;
          if (MemLimit64_Iio[0] > MemBase64_Iio[0]) {
            AmlResourceAddress64Pointer->Minimum = (UINT64) MemBase64_Iio[0];
            AmlResourceAddress64Pointer->Maximum = (UINT64) MemLimit64_Iio[0];
            AmlResourceAddress64Pointer->AddressLength = (UINT64)
(MemLimit64_Iio[0] - MemBase64_Iio[0] + 1);
          }
          break;
      }
    }
```

```
  }
}

#pragma pack(1)

typedef struct {
  UINT8    StartByte;
  UINT32   NameStr;
  UINT8    Size;
  UINT32   Value;
} EFI_ACPI_NAMEPACK_DWORD;

typedef struct {
  UINT8    StartByte;
  UINT32   NameStr;
  UINT8    OpCode;
  UINT16   Size;
  UINT8    NumEntries;
} EFI_ACPI_NAME_COMMAND;

typedef struct {
  UINT8     NameOp;          // 08h ;First opcode is a NameOp.
  UINT32    PackageName;     // PSDC
  UINT8     Length;          // 12h
  UINT8     DwordPrefix1;    // 0Fh
  UINT8     Revision;        // 01h
  UINT8     PackageOp;       // 12h
  UINT8     PackageLen;      // 0Ch
  UINT8     PackLen;         // 05h
  UINT16    WordValue1;      // 0A05h
  UINT16    WordValue2;      // 0A00h
  UINT8     BytePrefix2;     // 00h
  UINT8     Domain;          // 08h
  UINT8     BytePrefix3;     // 0Ah
  UINT8     CoordType;       // 0FCh/0FDh/0FEh
  UINT8     BytePrefix4;     // 0Ah
  UINT8     NumProcessors;   // 10h
} PSD_PACKAGE_LAYOUT;

#pragma pack()

#define ACPI_NAME_COMMAND_FROM_NAME_STR(a)  BASE_CR (a, EFI_ACPI_NAME_COMMAND, Name
Str)
#define ACPI_NAME_COMMAND_FROM_NAMEPACK_STR(a)  BASE_CR (a, EFI_ACPI_NAMEPACK_DWORD
, NameStr)
```

```
#define THREAD_MASK_64        (0x3F)
#define CPUIDX_SKT_SHIFT_BIT  6

EFI_STATUS
PatchSsdtTable (
  IN OUT  EFI_ACPI_DESCRIPTION_HEADER *TableHeader
  )
{
  UINT8                   *SsdtPointer;
  UINT32                  i;
  UINT32                  Signature;
  UINT8                   DOMNValue;
  UINT8                   NCPUValue;
  UINT32                  CpuSkt;
  UINT32                  CpuIndex;
  EFI_ACPI_NAMEPACK_DWORD *NamePtr;
  EFI_ACPI_NAME_COMMAND   *PssTable;
  EFI_ACPI_NAME_COMMAND   *PsdPackage;
  PSD_PACKAGE_LAYOUT      *PsdPackageItemPtr;
  AML_OFFSET_TABLE_ENTRY  *mSsdtAmlOffsetTablePointer = &SSDT_SSDT__PM_OffsetTable;

  DOMNValue = 0;
  NCPUValue = 0;

  //
  // Loop through the AML to fix up items.
  //
  // Items should appear in this order:
  //   _SB_.SCKx.CPyz.NCPU
  //   _SB_.SCKx.CPyz.DOMN
  //   _SB_.SCKx.CPyz.PSDC
  //   _SB_.SCKx.CPyz.NPSS
  //
  for (i = 0; SsdtAmlOffsetTablePointer[i].Pathname != 0; i++) {
    //
    // Point to offset in SSDT for current item in AmlOffsetTable.
    //
    if (SsdtAmlOffsetTablePointer[i].Opcode == AML_DWORD_PREFIX) {
      SsdtPointer = (UINT8 *) (TableHeader) + SsdtAmlOffsetTablePointer[i].Offset -
 5; // -5 to iASL offsets to be at Name
    } else if (SsdtAmlOffsetTablePointer[i].Opcode == AML_PACKAGE_OP) {
      SsdtPointer = (UINT8 *) (TableHeader) + SsdtAmlOffsetTablePointer[i].Offset -
 4; // -4 to iASL offsets to be at Name
    } else {
      SsdtPointer = (UINT8 *) (TableHeader) + SsdtAmlOffsetTablePointer[i].Offset;
    }
```

```
    Signature = *(UINT32 *) (SsdtAmlOffsetTablePointer[i].Pathname + AsciiStrLen(Ss
dtAmlOffsetTablePointer[i].Pathname) - 4);
    switch (Signature) {
      //
      // Step 1. "NCPU", determine SCKx and CPyz, set local DOMNValue
      //
      case (SIGNATURE_32 ('N', 'C', 'P', 'U')):
        //
        // NCPU Pathname is "_SB_.SCKx.CPyz.NCPU".  x and yz are hex, so they must
be converted to decimal.
        //
        // Convert SCKx from hex to decimal.
        //
        if ((SsdtAmlOffsetTablePointer[i].Pathname[8] - '0') <= 9) {
          CpuSkt = (UINT32) (SsdtAmlOffsetTablePointer[i].Pathname[8] - '0');
        } else {
          CpuSkt = (UINT32) (SsdtAmlOffsetTablePointer[i].Pathname[8] - 'A' + 10);
        }
        //
        // Convert CPyz from hex to decimal.
        //
        if ((SsdtAmlOffsetTablePointer[i].Pathname[12] - '0') <= 9) {
          CpuIndex = (UINT32) ((SsdtAmlOffsetTablePointer[i].Pathname[12] -
 '0') << 4);
        } else {
          CpuIndex = (UINT32) ((SsdtAmlOffsetTablePointer[i].Pathname[12] -
 'A' + 10) << 4);
        }
        if ((SsdtAmlOffsetTablePointer[i].Pathname[13] - '0') <= 9) {
          CpuIndex += (UINT32) (SsdtAmlOffsetTablePointer[i].Pathname[13] - '0');
        } else {
          CpuIndex += (UINT32) (SsdtAmlOffsetTablePointer[i].Pathname[13] -
 'A' + 10);
        }

        CpuIndex = (CpuSkt << CPUIDX_SKT_SHIFT_BIT) + (CpuIndex & THREAD_MASK_64);

        // Update DOMN
        DOMNValue = (UINT8) CpuIndex >> 1;

        NamePtr = ACPI_NAME_COMMAND_FROM_NAMEPACK_STR (SsdtPointer);
        if (NamePtr->StartByte != AML_NAME_OP) {
          continue;
        }
        if (NamePtr->Size != AML_NAME_DWORD_SIZE) {
```

```
      continue;
    }

    NamePtr->Value = (UINT32) NCPUValue;
    break;

  //
  // Step 2. "DOMN", write local DOMNValue to ACPI space
  //
  case (SIGNATURE_32 ('D', 'O', 'M', 'N')):
    NamePtr = ACPI_NAME_COMMAND_FROM_NAMEPACK_STR (SsdtPointer);
    if (NamePtr->StartByte != AML_NAME_OP) {
      continue;
    }
    if (NamePtr->Size != AML_NAME_DWORD_SIZE) {
      continue;
    }

    NamePtr->Value = (UINT32) DOMNValue;
    break;

  //
  // Step 3. "PSDC", write local NCPUValue / DOMNValue to ACPI space
  //
  case SIGNATURE_32 ('P', 'S', 'D', 'C'):
    PsdPackage = ACPI_NAME_COMMAND_FROM_NAME_STR (SsdtPointer);
    if (PsdPackage->StartByte != AML_NAME_OP) {
      continue;
    }

    PsdPackageItemPtr       = (PSD_PACKAGE_LAYOUT *) ((UINT8 *) PsdPackage);
    PsdPackageItemPtr->Domain = DOMNValue;
    PsdPackageItemPtr->NumProcessors = NCPUValue;
    break;

  //
  // Step 4. "NPSS", adjust PState data
  //
  case SIGNATURE_32 ('N', 'P', 'S', 'S'):
    PssTable = ACPI_NAME_COMMAND_FROM_NAME_STR (SsdtPointer);
    if (PssTable->StartByte != AML_NAME_OP) {
      continue;
    }

    break;
} // switch
```

```
    } // for

    return EFI_SUCCESS;
}
```

This page intentionally left blank.