WHITE PAPER

Intel Platform Enabling Guide

Open Source UEFI Firmware Enabling Guide: Intel Atom® Processor E3900 Series Platforms

August 2018

Intel provides an open source UEFI firmware project for the Intel Atom® Processor E3900 Series platforms (formerly Apollo Lake SoC). This document describes the architecture and boot flow of that firmware project.
See *https://firmware.intel.com/projects/IntelAtomProcessorE3900.*

Authors

David Wei

Vincent Zimmer

Mike Wu

Brian Richardson

Contents

Introduction.....1

Firmware Project Overview 7

Firmware Boot Flow 12

Configuring the Intel® FSP 17

IA Processor Initialization Flow 23

Board Level Configuration 32

Enabling Verified Boot 37

Intended Audience

This enabling guide is intended for firmware engineers, platform designers, and system developers.

Prerequisites

- Users of this document should have prior experience with firmware development using UEFI & EDK II. This includes the UEFI Specification and UEFI Platform Initialization (PI) Specification, available at uefi.org/specifications.
- Users should also be familiar with the Intel® Firmware Support Package (Intel® FSP), available at intel.com/fsp.
- This document assumes the audience has experience with Intel® 64 and IA-32 Architectures.

Acronyms and Terminology

Acronym / Term	Definition
ACPI	Advanced Configuration and Power Interface
API	Application Programming Interface
Apollo Lake	The pre-release code name for Intel Atom® Processor E3900 Series CPUs
BCT	Binary Configuration Tool
BIOS	Basic Input/Output System, a common term for the legacy Intel Architecture (IA) boot firmware.
BPDT	Boot Partition Description Table
BSF	Boot Setting File
CAR	Cache as RAM
CRB	Customer Reference Board
CPU	Central Processing Unit
CS	Chip Select
DQ	Data Input/Output
DRAM	Dynamic Random Access Memory
DSDT	Differentiated System Description Table
DTS	Digital Thermal Sensor
DXE	Driver Execution Environment
EDK II	EDK II is a cross-platform firmware development environment for the UEFI and PI specifications.
eMMC	Embedded Multi-Media Controller
FDF	Flash Description File
FIT	Intel Flash Image Tool
FSP	Intel® Firmware Support Package. See http://intel.com/fsp
FW	Firmware
GPIO	General Purpose Input-Output
HECI	Host Embedded Controller Interface
HFM	High Frequency Mode
HOB	Hand-off Block
IA	Intel Architecture
I2C	Inter-Integrated Circuit
IBB	Initial Boot Block
IBBL	Initial Boot Block Loader
IFWI	Integrated Firmware Image
IPU	Image Processing Unit
LBP	Logical Boot Partition

LFM	Low Frequency Mode
MEU	Manifest Extension Utility
NV	Non-volatile Variables
OBB	OEM Boot Block
OEM	Original Equipment Manufacturer
OS	Operating System
PCD	Platform Configuration Database
PEI	Pre-EFI Execution Environment
PMIC	Power Management Integrated Circuit
PMC	Power Management Controller
PPI	PEI Module to PEI Module Interfaces
RAM	Random Access Memory
SEC	Security phase
SCI	System Control Interrupt
SMBIOS	System Management BIOS
SMI	System Management Interrupt
SMIP	Signed Master Image Profile
SMM	System Management Mode
SoC	System on a Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSDT	Secondary System Description Tables
SVID	Serial Voltage ID
TianoCore	A community supporting an open source implementation of the UEFI specification
TPM	Trusted Platform Module
TXE	Intel® Trusted Execution Engine
UDK	UEFI Development Kit
UEFI	Unified Extensible Firmware Interface. See http://uefi.org
UPD	Updatable Product Data

Related Documents and Tools

- Unified Extensible Firmware Interface (UEFI) Specification, Advanced Configuration and Power Interface (ACPI) Specification, and Platform Initialization (PI) Specification: http://www.uefi.org/specifications
- Intel® Pentium® and Celeron® Processor N- and J- Series Datasheet Volume 1 of 3 https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-1.html
- Intel® Pentium® and Celeron® Processor N- and J- Series Datasheet Volume 2 of 3 https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-2.html
- Intel® Pentium® and Celeron® Processor N- and J- Series Datasheet Volume 2 of 3 https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-3.html
- Intel Atom® Processor E3900 and A3900 Series Datasheet Addendum https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/atom-processor-e3900-a3900-series-datasheet-addendum.pdf
- Intel® Firmware Support Package External Architecture Specification v2.0 http://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/fsp-architecture-spec-v2.pdf
- Apollo Lake Intel® Firmware Support Package (FSP) Integration Guide https://github.com/IntelFsp/FSP/blob/ApolloLake/ApolloLakeFspBinPkg/Docs/Apollo_Lake_FSP_Integration_Guide.pdf
- Intel® 64 and IA-32 Architectures Software Developer's Manual https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

Revision History

Date	Description
August 2018	Initial document release.

List of Tables

Table 1: SPI Flash Regions....9
Table 2: Firmware Volumes in IA Firmware....16
Table 3: FSP API and Wrapper....20
Table 4: Intel FSP UPD Input....21
Table 5: FSP Platform Hooks....21
Table 6: HOBs Produced by FSP....21
Table 7: Platform Porting Configuration Consideration....24
Table 8: SMM Foundation Modules....25
Table 9: SMM Extension Modules....25
Table 10: Platform Protocols Associated with SMM....26
Table 11: PPM Modules....26
Table 12: Power Features Controlled by FSPS_UPD....27
Table 13: ACPI Tables for Power Management....28
Table 14: Thermal Control Registers....28
Table 15: Critical Memory Controller Signals....33
Table 16: Memory Rank Configuration....36

List of Figures

Figure 1: SPI Flash Regions and Access Control 9
Figure 2: Logical Boot Partition Layout in IFWI 11
Figure 3: UEFI Variable Access via Intel TXE 11
Figure 4: TXE Loads IBBL 12
Figure 5: TXE and Host Processor Load IBB 13
Figure 6: Shadow IBB to Permanent Memory DRAM 14
Figure 7: Load OBB into DRAM 15
Figure 8: Layout of Cache as RAM Region 18
Figure 9: IA Firmware and Intel FSP Boot Flow 22
Figure 10: Thermal Management Logical Flow 31
Figure 11: Board Specific Configuration 32
Figure 12: Micron Single Die, Dual Channel Package 34
Figure 13: Memory Controller Channel 34
Figure 14: Micron Dual Die, Dual Channel Package 35
Figure 15: Memory Controller Channel 36
Figure 16: Firmware Boot Flow, Verified Boot Enabled 38
Figure 17: Intel TXE Keys, Owners, and Storage Locations 40

Firmware Project Overview

The EDK II open source firmware project for Intel Atom® Processor E3900 Series platforms combines the following platform firmware components to form the Integrated Firmware Image (IFWI) stored in SPI NOR on the motherboard:

- Intel Architecture (IA) Firmware for UEFI
- Intel® Trusted Execution Engine (Intel® TXE) firmware
- Power Management Controller (PMC) firmware
- Microcode and P-unit firmware

This section introduces the functions of IFWI components, and describes the layout of the SPI NOR flash.

Firmware Components within IFWI

Some components of the IFWI contain code for platform level initialization, while others are for device level initialization. The IFWI also contains regions for storing platform data.

Platform Firmware

Platform firmware components are mandatory code for platform initialization. They are responsible for initializing power management controllers (PMC and P-unit), Intel TXE, and Intel Atom® Processor E3900 Series CPUs.

Intel Trusted Execution Engine (TXE) Firmware

Intel TXE Firmware is the code executed by Intel TXE. It brings up TXE and exposes runtime security services such as firmware TPM (fTPM) and Intel® Platform Protection Technology with Boot Guard. Boot Guard in Intel TXE Firmware loads and authenticates other firmware components during boot.

IA Firmware communicates with Intel TXE firmware through HECI. This binary firmware is provided by Intel and signed by Intel's private key.

Intel Architecture (IA) Firmware

IA Firmware is executed by the host processor. It initializes the platform and loads the operating system.

The open source IA firmware provided for Intel Atom® Processor E3900 Series CPUs follows the UEFI Specification, revision 2.6. The codebase also leverages the Intel UEFI Development Kit 2018 (UDK2018), a stable release of the TianoCore EDK II project.

IA Firmware is divided into three execution stages:

1. Initial Boot Block Loader (IBBL)
   First stage of IA Firmware, which executes in Static Random Access Memory (SRAM). Due to the size limitation of SRAM, IBBL is only 4KB in size. From the perspective of UDK2018 architecture, IBBL only contains IA processor Resect Vector code and part of SEC stage code of UEFI IA Firmware which are necessary to load IBB image into temporary memory (Cache as RAM).
2. Initial Boot Block (IBB)
   Second stage of IA Firmware, which executes from temporary memory (CAR). After DRAM has been initialized and CAR is torn down, PEI Modules which register PPIs and PPI notification callbacks have to be shadowed into DRAM and re-dispatched, during which

those PPIs and PPI notification callbacks will be re-installed.

From the perspective of UDK2018 architecture, IBB contains part of SEC stage code, pre-memory PEI modules, and memory initialization code (FSP-M). IBB loads OBB into DRAM.

3. OEM Boot Block (OBB)
   The third stage of IA Firmware which executes from DRAM. From the perspective of UDK2018 architecture, OBB contains post-memory PEI modules, FSP-S, DXE stage and BDS stage.

Besides the above three code regions for UEFI, there is a separate area in the Logical Data Region of SPI flash for UEFI to store Non-volatile Variables (NV). IA Firmware does not access this region directly, but can access it via HECI commands to Intel TXE.

Power Management Controller (PMC) Firmware

PMC firmware is provided by Intel and signed with Intel's private key. It initializes the PMC to provide the following functions:

- Conducting Warm/Cold/Global resets.
- Conducting system boot flow.
- Conducting Sleep states (S3, S4, and S5) entry.
- Collecting system wake events and conducting system wake from sleep states.
- Managing power rails.
- Handling System Management Interrupt (SMI).
- Handling System Control Interrupt (SCI).
- Communicating with external Power Management IC (PMIC) with SVID or I2C interface.

Microcode and P-Unit Firmware

Microcode and P-unit Firmware are released by Intel in a single signed binary, which is signed with Intel's private key.

The P-Unit manages power for IA cores, graphics engine (GT), and memory subsystem. It also participates in reset flow and system sleep state transition.

SPI NOR Flash Layout

The platform supports up to two SPI flash devices. The SPI flash connected to Chip Select 0 is used for storing firmware, and contains a valid SPI Flash Descriptor in the first 4KB of SPI NOR flash. The SPI Flash Descriptor is a data structure that is programmed to the header region of the SPI flash part. The SPI controller is aware of the data structure in the SPI Flash Descriptor region.

The SPI Flash Descriptor describes the flash layout, provides configuration parameters for the platform, and enables the SoC to securely manage storage among multiple SPI masters.

SPI Flash Regions

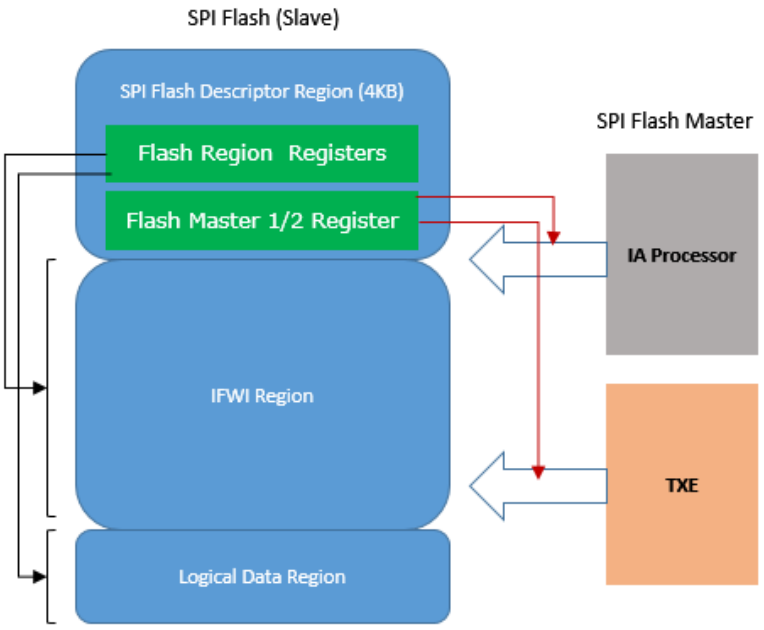
The 4K byte SPI Flash Descriptor at the head of the SPI device splits the SPI flash into regions and defines access control for each region.

- Flash Region registers in the Flash Descriptor Region Section are used to control the base address and size limit of each region. Table 1 lists all the regions.
- Flash Master 1 (Host CPU/ BIOS) register and Flash Master 2 (Intel® TXE) register in the Flash Descriptor Master Section control SPI master access. Table 1: SPI Flash Regions and Access Control illustrates how these register control access to each region.

Table 1: SPI Flash Regions

SPI Flash Regions	Content
SPI Flash Descriptor Region	4KB in size; Platform configuration parameters; Define flash layout; Define flash region accessibility.
IFWI Region (Logical Boot Partitions)	Contains various firmware components (IA Firmware, Intel TXE firmware, PMC firmware, etc.). IFWI region is divided into 2 Logical Boot Partitions (LBP), which are defined by Boot Partition Description Table (BPDT).
Logical Data Region	Contains system non-volatile data managed by Intel TXE, including UEFI NV Storage.

Figure 1: SPI Flash Regions and Access Control



IFWI Region

The IFWI region in SPI flash physically follows the SPI Flash Descriptor Region. It contains all platform firmware components and device firmware components.

The IFWI region is divided into two Logical Boot Partitions, which are identical in size. The Logical Boot Partition layout is defined by the Boot Partition Descriptor Table (BPDT) at the head of the Logical Boot Partition.

Logical Boot Partition 1 and Logical Boot Partition 2 contain multiple entries, also referred to as sub-partitions. These can be signed firmware components or a Secondary Boot Partition ("Secondary BPDT").

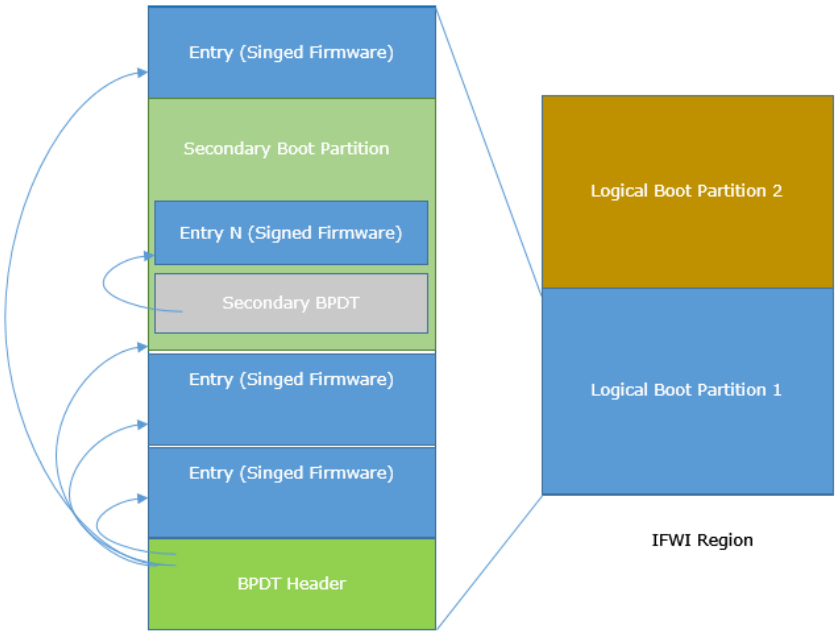
- BPDT: Boot Partition Descriptor Table
- SBPDT: Secondary BPDT, points to sub-partitions recursively
- OEM SMIP (Signed Master Image Profile)
- TXE Firmware
- IPU Firmware
- PMC Firmware
- CPU Microcode and p-Unit Firmware
- IA Firmware
    - IBBL
    - IBB
    - OBB
- Misc. Device Firmware
- Security Data
    - Boot Policy Manifest
    - OEM Key Manifest

The open source UEFI project contains details for the BPDT and code for accessing entries inside Logical Boot Partitions:

https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Silicon/BroxtonSoC/BroxtonSiPkg/Library/BpdtLib

Figure 2 is an example of Logical Boot Partition layout that could be used to illustrate the structure of both Logical Boot Partition 1 and Logical Boot Partition 2.

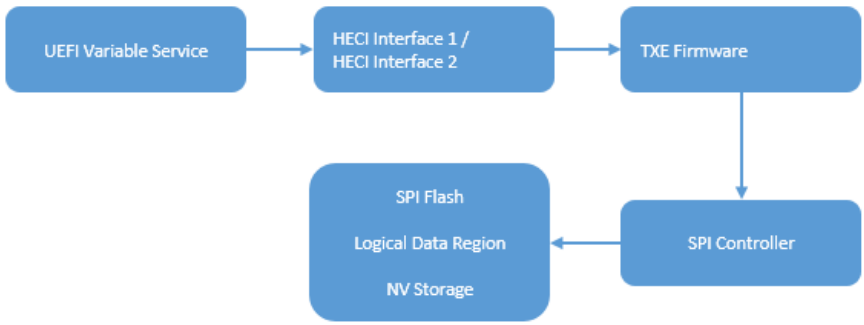
Figure 2: Logical Boot Partition Layout in IFWI



Logical Data Region

Logical Data Region on SPI flash is managed by TXE. UEFI Non-Volatile variables are stored in Logical Data Region. UEFI/BIOS does not directly access this region, but reads and writes data there through the HECI interface, as Figure 3 shows.

Figure 3: UEFI Variable Access via Intel TXE



Flash Image Tool (FIT)

The Intel Flash Image Tool (FIT) is a tool which can be used to configure the settings inside SPI Flash Descriptor, to stitch firmware components together to form the IFWI region. Please contact your Intel Representative to get the FIT tool.

Firmware Boot Flow

On a typical IA platform, SPI NOR flash is mapped to the top of the host processor's 4GB memory space. The host processor reset vector starts code execution from an address decoded by SPI NOR flash, just below 4GB.

On Intel Atom® Processor E3900 Series platforms, the SPI NOR flash is not directly mapped to the host processor memory space at the beginning of boot. A segment of Shared Static RAM (SRAM), which is shared by the host processor and Intel TXE, is mapped to the top of 4GB memory space. The introduction of SRAM makes the boot flow of Intel Atom® Processor E3900 Series platforms different from previous IA processors.

IA Firmware sends HECI messages via Intel TXE to request ownership of the SPI NOR flash. After the host processor gets ownership of SPI NOR, the flash is mapped to the top 4GB of CPU memory space.

This maps IA firmware into three execution phases:

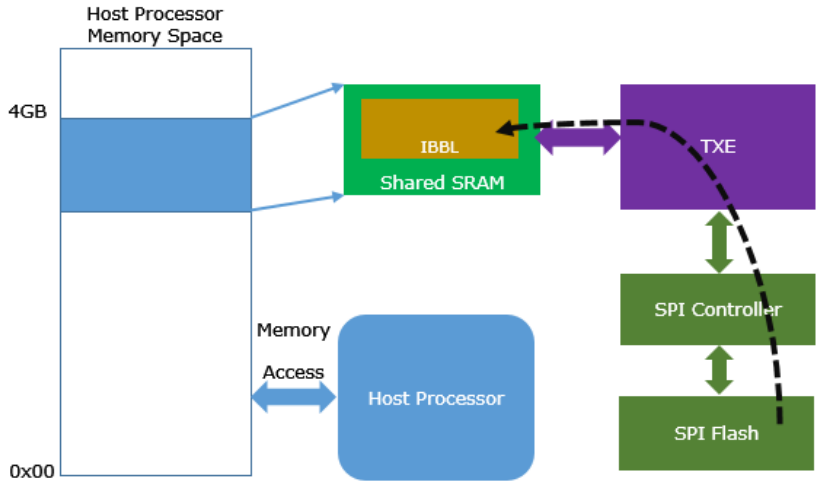
- IBBL is executed in Shared SRAM
- IBB is executed in temporary memory (Cache as RAM)
- OBB is executed in system DRAM

Pre- IA Firmware Stage

At system power-on, Intel TXE executes prior to giving control to the host processor:

1) Intel TXE loads firmware ingredients from SPI NOR flash to TXE SRAM and starts execution from Intel TXE SRAM.
2) Intel TXE creates shared SRAM at top of the host processor's 4GB memory address space. This shared SRAM is accessible by both Intel TXE and the host IA processor, just as Figure 4 shows.
3) Intel TXE copies the IA Firmware IBBL and microcode from SPI NOR flash to the shared SRAM. Figure 4 shows the loading process.
4) PMC brings the host processor out of reset.
5) Host processor loads microcode.
6) Host processor starts executing IA firmware IBBL from the host processor's reset vector (memory region mapped just below 4GB).

Figure 4: TXE Loads IBBL



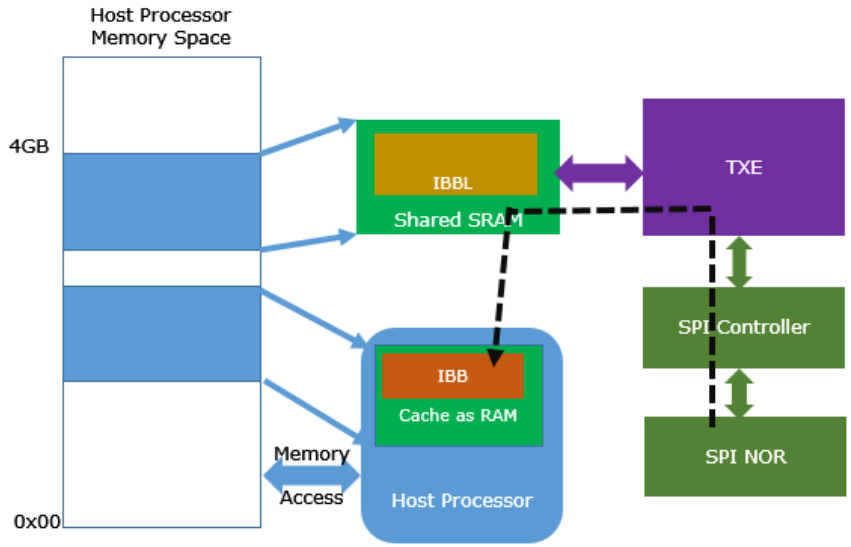
IBBL Stage

IBBL is the first stage of IA Firmware, executed from Shared SRAM. The IA processor starts fetching IBBL code from the reset vector.

1) IBBL sets up Cache as RAM (CAR) as temporary memory.
2) IBBL requests Intel TXE load IBB from SPI Flash into shared SRAM.
3) IBBL copies IBB from SRAM to CAR area.
4) IBBL transfers control to IBB.

Figure 5 shows how IBB is loaded from SPI flash into Cache as RAM.

Figure 5: TXE and Host Processor Load IBB



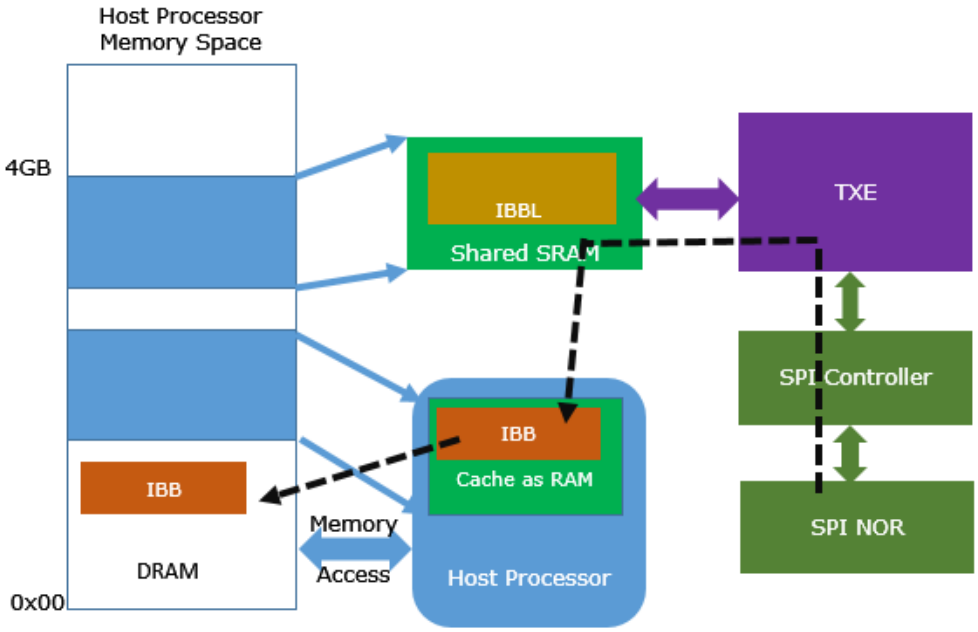
IBB Stage

IBB is the second stage of IA Firmware, containing the PEI Core and Memory Reference Code (FSP-M). Prior to system memory initialization, IBB resides in temporary memory (CAR).

After IBB completes system memory (DRAM) initialization, the following steps will be taken to disable CAR and avoid data loss:

1) PEI Core shadows itself from CAR into DRAM, as Figure 6 shows.
2) PEI Core shadows any PEI Modules which have been dispatched and registered themselves with PEI Service RegiserForShadow().
   Note:
   a) *A PEI Module must register itself with RegiserForShadow() if it has any PPI produced. It has to reinstall the PPIs it has installed previously. Otherwise after CAR is teared down, those PPIs' EFI_PEI_PPI_DESCRIPTOR and the PPI code, which was located in CAR, will be unavailable. This could lead to system crash*
   b) *PEI code that runs prior to PEI memory installation, which is handled by PEI Service InstallPeiMemory (), cannot register any PPI notification with PeiNotifyPpi(). Otherwise the EFI_PEIM_NOTIFY_ENTRY_POINT pointer in EFI_PEI_NOTIFY_DESCRIPTOR will still point to code in CAR, which will be unavailable. Accessing this pointer will lead to system crash.*

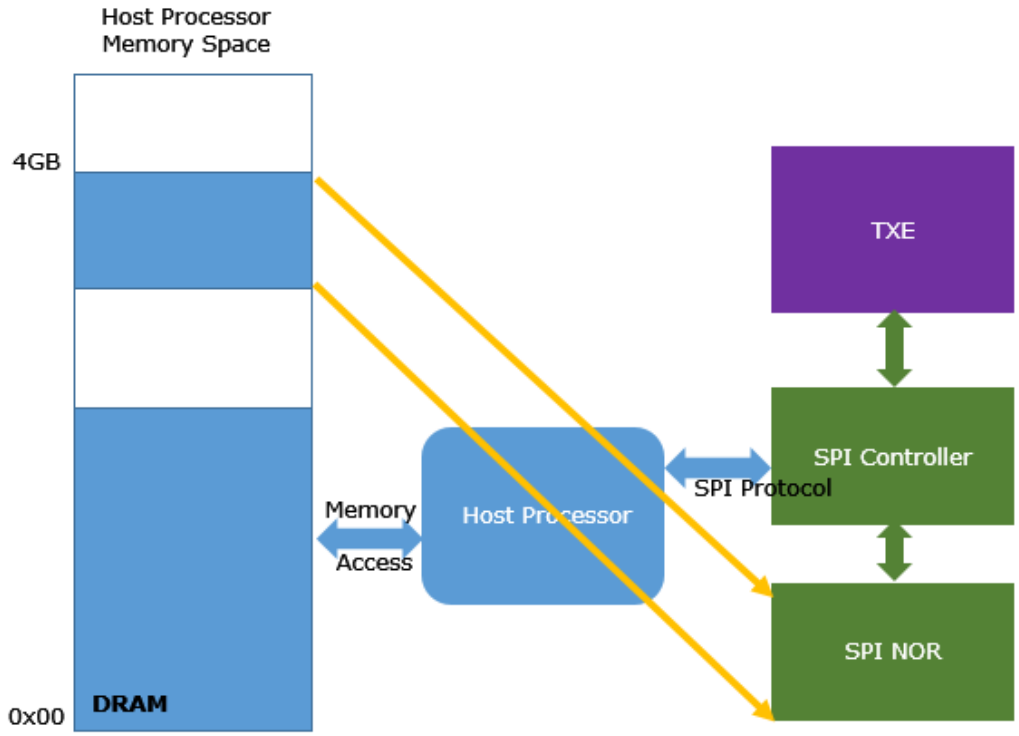
Figure 6: Shadow IBB to Permanent Memory DRAM



3) IBB issues a HECI message to TXE to request ownership of SPI NOR flash. After ownership is granted, SPI NOR flash will be mapped to the top 4GB of host processor memory space. Host processor could get data from SPI NOR flash by memory reading operation.
4) IA processor could also read and write SPI NOR flash through SPI protocol.
5) IBB reads OBB from SPI flash to system DRAM and gives control to OBB.

Figure 7 illustrates the memory map after the host controller has acquired ownership of SPI flash.

Figure 7: Load OBB into DRAM



OBB Stage

OBB consists of post-memory PEI modules, FSP-S, DXE modules, and BDS of UEFI IA Firmware. These modules initialize CPU, Chipset and platform:

- GPIO initialization
- Intel Chipset configuration
- Publish UEFI Services
- Multi-Processor initialization
- Initialize Graphics device and provide GOP service
- Initialize Storage devices
- Publish ACPI tables
- Logo display
- OEM specific customized features
- SMM initialization
- Network
- Console
- SMBIOS, TCG
- BIOS Setup

UEFI IA Firmware Volumes in IBBL, IBB and OBB

The EDK II Flash Description File (FDF) determines how UEFI IA Firmware components are allocated across IBBL, IBB and OBB. For details, please refer to the project's FDF file and IFWI stitch file in TianoCore GitHub:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/PlatformPkg.fdf

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Tools/Stitch/IFWIStitch_Simple.bat

Table 2: Firmware Volumes in IA Firmware

Boot Block	Firmware Volumes
IBBL	IBBL.Fv
IBB	FVIBBM.Fv FSP_M.fv
OBB	FSP_S.fv FVIBBR.fv FVOBB.Fv FVOBBX.Fv FVOBBY.Fv

Configuring the Intel® FSP

The Intel Firmware Support Package (Intel® FSP) is a binary package of initialization code for Intel silicon. This package provides access to silicon initialization functions that are not available in open source code or public documentation. This section described how to configure Intel FSP for Intel Atom® Processor E3900 Series platforms.

Intel FSP Release Package and Related Tools

- The Intel FSP release package for Intel Atom® Processor E3900 Series platforms is available in github: https://github.com/IntelFsp/FSP/tree/ApolloLake
- The release package contains a binary image (.fd) and configuration file (.bsf):
  https://github.com/IntelFsp/FSP/tree/ApolloLake/ApolloLakeFspBinPkg/FspBin/Fsp.fd
  https://github.com/IntelFsp/FSP/tree/ApolloLake/ApolloLakeFspBinPkg/FspBin/Fsp.bsf
- Intel FSP contains the platform Memory Reference Code, PEIM for the graphics controller, and other Silicon Reference Code. Please refer to the release notes for more details: https://github.com/IntelFsp/FSP/blob/ApolloLake/ApolloLakeFspBinPkg/Docs/Apollo_Lake_FSP_MR5_Release_Notes.pdf
- The Intel FSP Binary Configuration Tool (BCT) is required to edit the configuration file: https://github.com/IntelFsp/BCT
- The Intel FSP Binary Split Tool (SplitFspBin.py) is also required to divide the binary file into smaller components: https://github.com/tianocore/edk2/tree/master/IntelFsp2Pkg/Tools

Common Parameters Controlled by Intel FSP

Each Intel FSP binary contains a configurable data region, the Updateable Product Data (UPD), which will be used by the Intel FSP during initialization for SoC configuration.

Updateable Product Data (UPD) contains the default parameters for the FSP initialization. The UPD parameters can also be statically customized using the Binary Configuration Tool (BCT) and Boot Setting File (BSF), if a custom platform requires any different UPD parameters than the default ones.

The parameters of UPD are defined by the following head files:

https://github.com/IntelFsp/FSP/blob/ApolloLake/ApolloLakeFspBinPkg/Include/FspmUpd.h

https://github.com/IntelFsp/FSP/blob/ApolloLake/ApolloLakeFspBinPkg/Include/FspsUpd.h

Integrating Intel FSP into Open Source UEFI IA Firmware

The Intel FSP image consists of three different components (FSP-T, FSP-M and FSP-S), which can be located at different base addresses according to the execution environment. The boot flow has three different execution stages:

- Execution in Shared SRAM
- Execution in temporary memory (Cache as RAM)
- Execution in system memory (DRAM)

This boot flow requires that Intel FSP components be mapped into different execution stages of the boot flow.

- FSP-T executes in Shared SRAM
- FSP-M executes in temporary memory. After memory is initialized, generic elements like the PEI core and Intel FSP data will migrate to permanent memory
- FSP-S executes in system memory (DRAM)

FSP-T is not used by this project, because open source UEFI IA Firmware already covers the function of FSP-T. So we will only discuss FSP-M and FSP-S in this paper.

The FSP-M firmware volume (FV) is integrated into IBB, and the FSP-S FV is integrated into OBB. Note: FSP-T is not used in this firmware project.

Rebase FSP

The **`SplitFspBin.py`** utility is used to rebase the separate sections of Intel® FSP (FSP-T, FSP-M, FSP-S). This section describes how the build process uses this utility.

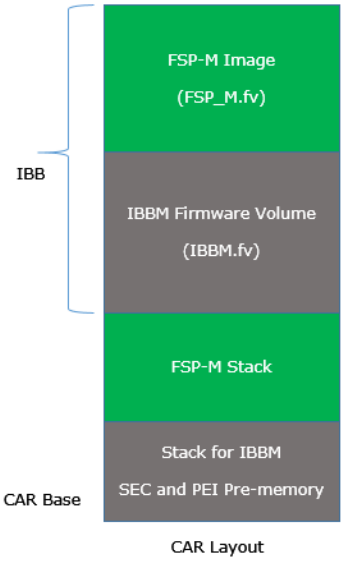
Rebase FSP-M

FSP-M is part of IBB, which is executed from Cache as RAM (CAR). The layout of CAR is described in Figure 8, from which we could calculate out the runtime base address of FSP-M. The pre-build script rebases FSP-M to the runtime address within CAR:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/BuildBxtBios.bat

```
::0xFEF7A000 = gIntelFsp2WrapperTokenSpaceGuid.PcdFlashFvFspBase =
::$(CAR_BASE_ADDRESS) + $(BLD_RAM_DATA_SIZE) + $(FSP_RAM_DATA_SIZE) +
::$(FSP_EMP_DATA_SIZE) + $(BLD_IBBM_SIZE)
Python %WORKSPACE%\Core\IntelFsp2Pkg\Tools\SplitFspBin.py rebase -f Fsp.fd -c
m -b 0xFEF7A000 -o .\ -n ApolloLakeFsp.fd
```

Figure 8: Layout of Cache as RAM Region



Rebase FSP-S

The default base address for FSP-S is set to 0x0200000. This codebase does not rebase FSP-S at build time, so the UEFI IA Firmware will shadow the FSP-S FV to 0x0200000 at runtime before calling the FSP-S API.

Please refer to the following code for FSP-S shadowing during boot:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/PlatformSettings/PlatformPreMemPei/FvCallback.c

```
//
// Copy to FSP-S to preferred base. The preferred base is defined in FSP
// Integration Guide.
// This region must be reserved for BIOS S3 resume.
//
BuildMemoryAllocationHob (
  (EFI_PHYSICAL_ADDRESS)FspSImageBase,
  (UINT64)FspHeader->ImageSize,
  EfiReservedMemoryType
);
CopyMemSse4 ((VOID*) FspSImageBase, FvHeader, (UINT32) FvHeader->FvLength);
PcdSet32S (PcdFspsBaseAddress, (UINT32) FspSImageBase);
```

Split FSP FD File

After rebasing, the Intel FSP FD file will be split into three FV files (**`FSP_T.fv, FSP_M.fv FSP_S.fv`**) by invoking **`SplitFspBin.py`** in the build script file:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/BuildBxtBios.bat

```
python %WORKSPACE%\Core\IntelFsp2Pkg\Tools\SplitFspBin.py split -f Fsp.fd -o
.\ -n FSP.Fv
```

Integrate Intel FSP FV files into UEFI IA Firmware

The firmware build invokes a stitching process to integrate the FSP-M FV file into IBB, and the FSP-S FV file into OBB:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Tools/Stitch/IFWIStitch_Simple.bat

Add Intel FSP Wrapper Modules into UEFI IA Firmware

The Intel FSP Architecture Specification 2.0 defines five standard APIs which abstract major silicon initialization functions. Please refer to Section 8 (FSP Interface) of the Intel FSP Architecture Specification 2.0 for details.

To facility the usage of these APIs, UDK2018 provides several wrapper modules. UEFI IA Firmware developers only need to add “FSP Wrapper” modules, as described in Table 3, instead of creating custom code to trigger the relevant APIs.

Table 3: FSP API and Wrapper

FSP API	Exposed By	Related Wrapper Module in IA Firmware
TempRamInit	FSP-T	Not used by this firmware project.
FspMemoryInit	FSP-M	Called by FspmWrapperPeim. https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/SampleCode/IntelFsp2WrapperPkg/FspmWrapperPeim
TempRamExit	FSP-M	Called by FspmWrapperPeim. https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/SampleCode/IntelFsp2WrapperPkg/FspmWrapperPeim
FspSiliconInit	FSP-S	Called by FspsWrapperPeim. https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/SampleCode/IntelFsp2WrapperPkg/FspsWrapperPeim
NotifyPhaseApi	FSP-S	Called by FspNotifyDxe. https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/SampleCode/IntelFsp2WrapperPkg/FspNotifyDxe

Customize Intel FSP Configuration Data at Boot Time

Each Intel FSP module contains a configurable data region which will be used by the FSP during initialization. This configuration region is a data structure called the Updateable Product Data (UPD), which contains parameters for controlling Intel FSP initialization.

UPD data can be dynamically overridden by UEFI IA Firmware during boot time.

The following code defines Intel FSP UPD data structures:
https://github.com/IntelFsp/FSP/blob/ApolloLake/ApolloLakeFspBinPkg/Include

One of the major jobs of the Intel FSP Wrapper is to customize UPD parameters. The **`FspWrapperPlatformLib`** provides platform hooks to update UPD:

https://github.com/tianocore/edk2/tree/master/IntelFsp2WrapperPkg/Library/BaseFspWrapperPlatformLibSample

Typically, the UEFI IA Firmware Setup interface exposes options for user configuration. The Intel FSP Wrapper obtains settings from Setup variable and uses those settings to configure UPD. Table 4 lists the FSP APIs and corresponding UPD update code.

Table 4: Intel FSP UPD Input

FSP API	UPD	Updated by Platform Lib of Intel FSP Wrapper	Wrapper Module
FspMemoryInit	FSPM_UPD (FspmUpd.h)	UpdateFspmUpdData() FspWrapperPlatformLib	FspmWrapperPeim
FspSiliconInit	FSPS_UPD (FspsUpd.h)	UpdateFspsUpdData() FspWrapperPlatformLib	FspsWrapperPeim

Extract Intel FSP Output (HOB)

Intel FSP builds a series of Hand-Off-Blocks (HOBs) as it progresses through silicon initializing. The Intel FSP API returns the HOB list at the end of execution. IA Firmware can extract preferred HOBs and add them to its own HOB list.

The **FspWrapperHobProcessLib** provides platform hooks, as Table 5 and 6 list, for the Intel FSP Wrapper Module to extract preferred HOBs produced by FSP.

https://github.com/tianocore/edk2/tree/master/IntelFsp2WrapperPkg/Library/PeiFspWrapperHobProcessLibSample

Table 5: FSP Platform Hooks

FSP API	UPD	Platform Hook of Intel FSP Wrapper	Wrapper Module
FspMemoryInit	FSPM_UPD (FspmUpd.h)	PostFspmHobProcess () FspWrapperHobProcessLib	FspmWrapperPeim
FspSiliconInit	FSPS_UPD (FspsUpd.h)	PostFspsHobProcess () FspWrapperHobProcessLib	FspsWrapperPeim

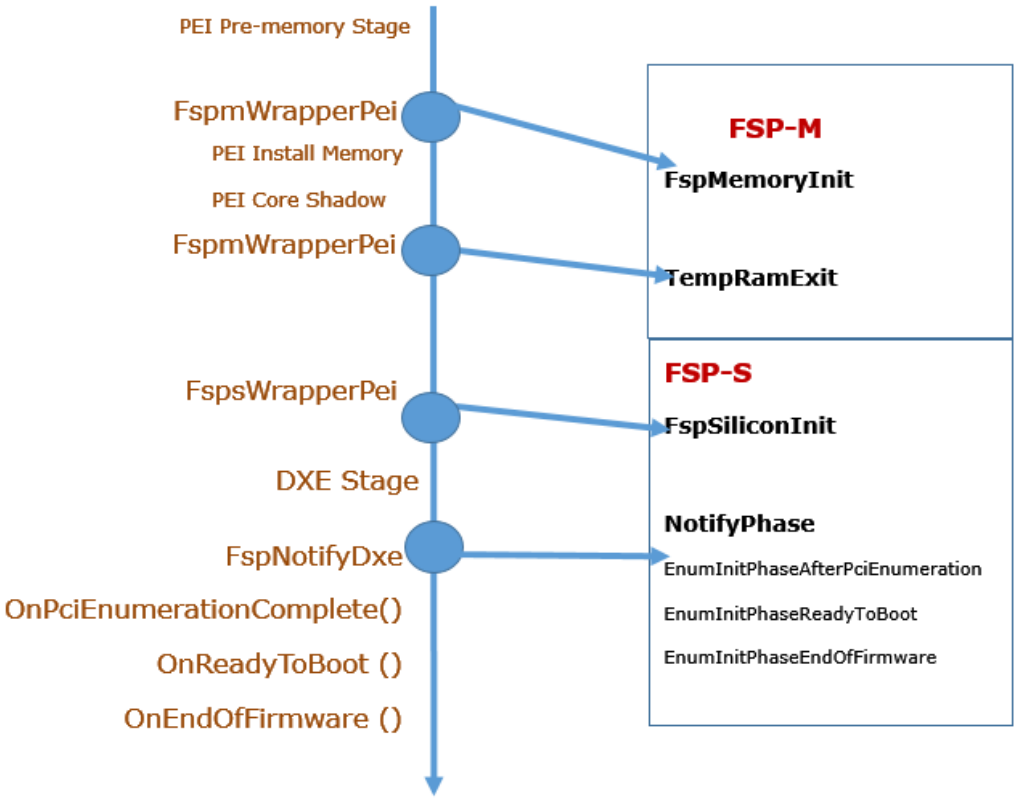
Table 6: HOBs Produced by FSP

Platform Hook	HOBs extracted
PostFspmHobProcess ()	**EFI_HOB_TYPE_RESOURCE_DESCRIPTOR EFI_RESOURCE_SYSTEM_MEMORY**
	EFI_HOB_TYPE_RESOURCE_DESCRIPTOR EFI_RESOURCE_MEMORY_RESERVED
PostFspsHobProcess ()	**EFI_HOB_TYPE_GUID_EXTENSION gScDeviceTableHobGuid**
	EFI_HOB_TYPE_GUID_EXTENSION gFspNonVolatileStorageHobGuid
	EFI_HOB_TYPE_GUID_EXTENSION gPeiAcpiCpuDataGuid
	EFI_HOB_TYPE_GUID_EXTENSION gEfiSmmPeiSmramMemoryReserveGuid
	EFI_HOB_TYPE_GUID_EXTENSION gCpuInitDataHobGuid
	EFI_HOB_TYPE_GUID_EXTENSION gSmbiosProcessorInfoHobGuid
	EFI_HOB_TYPE_GUID_EXTENSION gEfiGraphicsInfoHobGuid

Boot Flow from the Viewpoint of Intel FSP

Figure 9 illustrates the boot flow of UEFI IA Firmware and the timing of FSP Wrapper interacting with Intel FSP.

Figure 9: IA Firmware and Intel FSP Boot Flow



IA Processor Initialization Flow

Intel Architecture (IA) Processor initialization requirements are documented in the "Intel® 64 and IA-32 Architectures Software Developer's Manuals". These manuals are applicable to Intel Atom® Processor E3900 Series CPUs.

High Level Flow for IA Processor Initialization

Reset Vector

Code Sample: https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Silicon/BroxtonSoC/BroxtonSiPkg/Cpu/ResetVector/Vtf0

IA Processor Configuration:

- BSP execute code from Reset Vector.
- BSP enters 16bit real mode.
- IA Firmware sets BSP to protect mode, initialize processor Non-Eviction Mode (NEM) and setup Cache as RAM.

PEI Stage IA Processor Initialization

After IA Firmware has finished initializing DRAM, IA Firmware disables processor Non-Eviction Mode (NEM), sets MTRR registers and enables cache for BSP according to platform configuration. This is done by the FSP-M Wrapper calling FSP API **TempRamExit**.

Code Sample: https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/SampleCode/IntelFsp2WrapperPkg/FspmWrapperPeim

```
TempRamExitParam = GetTempRamExitParam ();
TimeStampCounterStart = AsmReadTsc ();
PERF_START_EX (&gFspApiPerformanceGuid, "EventRec", NULL, 0, 0xB000);
Status = CallTempRamExit (TempRamExitParam);
PERF_END_EX (&gFspApiPerformanceGuid, "EventRec", NULL, 0, 0xB07F);
```

IA Processor PEI to DXE Handoff

During the PEI stage, the IA processor runs in 32-bit protected mode. The DXE stage may execute in 64-bit or 32-bit mode. If DXE executes 64-bit code, the IA processor will switch to 64-bit IA-32e mode at the end of PEI.

Handoff Code:
https://github.com/tianocore/edk2/blob/UDK2018/MdeModulePkg/Core/DxeIplPeim/Ia32/DxeLoadFunc.c

The firmware developer must verify the following PCD setting:

```
## Indicates if DxeIpl should switch to long mode to enter DXE phase.
#  It is assumed that 64-bit DxeCore is built in firmware if it is true;
otherwise 32-bit DxeCore
#  is built in firmware.<BR><BR>
```

```
#  TRUE  - DxeIpl will load a 64-bit DxeCore and switch to long mode to hand
over to DxeCore.
#  FALSE - DxeIpl will load a 32-bit DxeCore and perform stack switch to hand
over to DxeCore.s
# @Prompt DxeIpl switch to long mode.

gEfiMdeModulePkgTokenSpaceGuid.PcdDxeIplSwitchToLongMode|TRUE|BOOLEAN|0x00010
03b
```

DXE Stage IA Processor Initialization

Multi-Processor Initialization

Multi-processor initialization is primarily done by the **CpuDxe** module and **MpInitLib**.

https://github.com/tianocore/edk2/tree/UDK2018/UefiCpuPkg/CpuDxe

https://github.com/tianocore/edk2/tree/UDK2018/UefiCpuPkg/Library/MpInitLib

- Initialize floating point unit.
- Create new GDT table for DXE stage and load new segment selectors based on new GDT.
- Initialize Interrupt Descriptor Table for interrupt handling. Initializes all CPU interrupt and exceptions entries and provides the default interrupt and exception handlers.
- Produce EFI_CPU_ARCH_PROTOCOL.
- Multi-processor initialization:
  - Enable the local APIC for Virtual Wire Mode.
  - Setup AP wakeup buffer.
  - Wakeup AP to do basic AP feature initialization.
  - Install EFI_MP_SERVICES_PROTOCOL.

Table 7 lists the PCDs that needs to be reviewed for specific platform.

Table 7: Platform Porting Configuration Consideration

PCD	Description
gUefiCpuPkgTokenSpaceGuid.PcdCpuApStackSize	Configure stack size for Application Processor (AP).
gUefiCpuPkgTokenSpaceGuid.PcdCpuApLoopMode	Specifies the AP wait loop state during POST phase. The value is defined as below 1: Place AP in the Hlt-Loop state. 2: Place AP in the Mwait-Loop state. 3: Place AP in the Run-Loop state.
gUefiCpuPkgTokenSpaceGuid.PcdCpuApTargetCstate	Specifies the AP target C-state for Mwait during POST phase.

System Management Mode (SMM)

UEFI SMM modules listed in Table 8 implement SMM foundations and extensions defined by the *Platform Initialization Specification, Volume 4: Management Mode Core Interface".*

Table 8: SMM Foundation Modules

SMM Foundation	File Path	Description
SMM Access	`$(PLATFORM_SI_PACKAGE)/Cpu/SmmAccess/Dxe/SmmAccess.inf`	Platform Specific driver which Controls access to SMRAM regions. Produces EFI_SMM_ACCESS2_PROTOCOL.
SMM Control	`$(PLATFORM_SI_PACKAGE)/SouthCluster/SmmControl/RuntimeDxe/SmmControl.inf`	Platform specific driver which produces EFI_SMM_CONTROL2_PROTOCOL for software SMI triggering.
PI SMM IPL	`MdeModulePkg/Core/PiSmmCore/PiSmmIpl.inf`	Load SMM Core into SMRAM, register SMM Core entry point for SMIs, install SMM Base 2 Protocol and SMM Communication Protocol, and register for the critical events required to coordinate between DXE and SMM environments.
PI SMM Core	`MdeModulePkg/Core/PiSmmCore/PiSmmCore.inf`	Reload SMM Core into SMRAM and register SMM Core EntryPoint on the SMI vector.
PI SMM CPU	`UefiCpuPkg/PiSmmCpuDxeSmm/PiSmmCpuDxeSmm.inf`	Setup SMRAM region for the CPU save state and code for the SMI entry point of each logical thread. Relocate SMM Base of each thread from default 0x38000 to preferred SMRAM.
CPU IO SMM	`UefiCpuPkg/CpuIo2Smm/CpuIo2Smm.inf`	

UEFI IA Firmware also provides extensions to facilitate the SMI handler registration, as described in Table 9.

Table 9: SMM Extension Modules

SMM Extension	File Path	Description
South Cluster SMI Dispatcher	`$(PLATFORM_SI_PACKAGE)/SouthCluster/ScSmiDispatcher/Smm/ScSmiDispatcher.inf`	Platform specific driver which produces a bunch of SMM Child Dispatch Protocols defined by *VOLUME 4: Platform Initialization Specification - System Management Mode Core Interface*
PI SMM Communication	`UefiCpuPkg/PiSmmCommunication/PiSmmCommunicationSmm.inf`	Produces `EFI_SMM_COMMUNICATION_PROTOCOL` which provides a means of communicating between drivers outside of SMM and SMI handlers inside of SMM.

The SMM foundation also hooks callback functions on DXE protocols. UEFI IA Firmware must ensure the protocols listed in Table 10 are properly installed.

Table 10: Platform Protocols Associated with SMM

Required Platform Event	Description
gEfiDxeSmmReadyToLockProtocolGuid	This protocol is a mandatory protocol published by a DXE driver prior to invoke the **EFI_SMM_ACCESS2_PROTOCOL.Lock()** function to lock SMM
gEfiEndOfDxeEventGroupGuid	This protocol is a mandatory protocol published by the PI platform code prior to invoking any 3rd-party content, including options ROM's and UEFI executables that are not from the platform manufacturer.

Processor Power Management (PPM)

Processor Power Management is implemented by Intel FSP, **`PowerMgmtDxe`** and **`CpuAcpiTables`**. It manages processor performance state (P-State) and processor sleep state (C-State). Table 11 lists the PPM modules.

Table 11: PPM Modules

Module	File Path
Intel FSP	https://github.com/IntelFsp/FSP/tree/ApolloLake
PowerMgmtDxe	$(PLATFORM_SI_PACKAGE)/Cpu/PowerManagement/Dxe/PowerMgmtDxe.inf
CpuAcpiTables	$(PLATFORM_SI_PACKAGE)/Cpu/AcpiTables/CpuAcpiTables.inf

Power Management Initialization

Intel FSP code related to power management is triggered by **`FspSiliconInit`** API, which uses FSPS_UPD parameters to initialize power management features. Please refer to Table 12 for the power management features controlled by FSPS_UPD.

Table 12: Power Features Controlled by FSPS_UPD

FSP-S UPD	Description
Eist	Enable or Disable Intel SpeedStep Technology. 0:Disable(Default), 1:Enable.
BootPState	Boot P-State with HFM or LFM. 0:HFM(Default), 1:LFM.
EnableCx	Enable or Disable Enhanced C-states. 0:Disable(Default), 1:Enable.
C1e	Enable or Disable Enhanced C1 state. 0:Disable(Default), 1:Enable.
BiProcHot	Enable or Disable Bi-Directional PROCHOT#. 0:Disable, 1:Enable(Default).
PkgCStateLimit	Max Pkg C state. 0:PkgC0C1, 1:PkgC2, 2:PkgC3(Default), 3:PkgC6, 4:PkgC7, 5:PkgC7s, 6:PkgC8, 7:PkgC9, 8:PkgC10, 9:PkgCMax, 254:PkgCpuDefault, 255:PkgAuto.
CStateAutoDemotion	C-State Auto Demotion. 0:Disable(Default) C1 and C3 Auto-demotion, 1:Enable C3/C6/C7 Auto-demotion to C1, 2:Enable C6/C7 Auto-demotion to C3, 3:Enable C6/C7 Auto-demotion to C1 and C3.
MaxCoreCState	Max Core C-State. 0:Unlimited, 1:C1, 2:C3, 3:C6, 4:C7, 5:C8, 6:C9, 7:C10, 8:CCx(Default).
PkgCStateDemotion	Enable or Disable Package Cstate Demotion. 0:Disable(Default), 1:Enable.
TurboMode	Enable or Disable long duration Turbo Mode. 0:Disable, 1:Enable(Default).

Power Management Features Reporting

The **CpuAcpiTables** module contains ACPI SSDT tables for EIST, C State and T State. The **FspSiliconInit** API returns **CPU_INIT_DATA_HOB**, which contains FVID table and the **CPU_GLOBAL_NVS_AREA** pointer. This data is used by the **PowerMgmtDxe** driver to update power management elements of SSDT of **CpuAcpiTables**. Table 13 lists the ACPI tables which report power management features to the OS.

https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Acpi/AcpiTablesPCAT

```
typedef struct {
  UINT32                Revision;
  EFI_PHYSICAL_ADDRESS  CpuConfig;
  EFI_PHYSICAL_ADDRESS  PowerMgmtConfig;
  EFI_PHYSICAL_ADDRESS  SoftwareGuardConfig;
  EFI_PHYSICAL_ADDRESS  CpuGnvsPointer;
  EFI_PHYSICAL_ADDRESS  MpData;
  EFI_PHYSICAL_ADDRESS  FvidTable;
  UINT32                SiliconInfo;
} CPU_INIT_DATA_HOB;
```

Table 13: ACPI Tables for Power Management

Table to Be Updated	Description
FADT	C3 Latency for C3 enabling
SSDT "Cpu0Ist"	Patch the native _PSS (Processor Supported States) package with the EIST values from CPU_INIT_DATA_HOB.FvidTable
SSDT "CpuSsdt"	Processor scope ACPI PPM GlobalNvs NVS Region with data from CPU_INIT_DATA_HOB.CpuGnvsPointer

Processor Digital Thermal Sensor (DTS) for Thermal Reporting

An Intel Atom® Processor E3900 Series CPU has a digital thermal sensor (DTS) in each core. These thermal sensors provide a means for periodically monitoring the thermal characteristics of each CPU core. The die temperature read from each sensor is compared against a programmable threshold. If any of the thresholds are exceeded, various actions can be triggered (interrupts, clock-throttling, P state transitions, or I/O pin signaling).

This section introduces the core level interrupt mechanism of the digital thermal sensor. Package level thermal interrupts are implemented with similar code logic.

Register Interfaces

Table 14 list the registers that are involved in implementing DTS function.

Table 14: Thermal Control Registers

Register	Description
IA32_THERM_STATUS (MSR 0x19C, Core Level)	The status of the temperature sensor. It also indicates the temperature of digital thermal sensor.
IA32_THERM_INTERRUPT (MSR 0x19B, Core Level)	Program the thermal thresholds for SMI interrupt generation.
LOCAL_APIC_THERMAL_DEF (MMIO 0xFEE00330, Core Level)	Enable thermal SMI interrupt of a core.
SWGPE_EN bit of GPE0a_EN	Software GPE Enable bit of PMC General Purpose Event 0 Enable Register. This bit enables the SW GPE function. If SWGPE_STS, SWGPE_EN, and SCI_EN are all ones, an SCI will be generated.
SWGPE_STS bit of GPE0a_STS	Software GPE Status bit of PMC General Purpose Event 0 Status Register. This bit indicate the assertion of SWGPE event. If SWGPE_STS, SWGPE_EN, and SCI_EN are all ones, an SCI will be generated.
SWGPE_CTRL bit of GPE_CTRL	Software General Purpose Event Control bit of PMC General Purpose Event Control. Programming one to this bit will assert the SW GPE event.

UEFI SMM Driver for DTS

https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Silicon/BroxtonSoC/BroxtonSiPkg/Cpu/PowerManagement/Smm

This driver enables DTS, initializes DTS interrupt thresholds, and registers a SMI handler for the DTS SMI. The thermal interrupt SMI handler performs the following functions:

- Reads processor core temperatures
- Fills in corresponding global variables:
  - GLOBAL_NVS_AREA.BspDigitalThermalSensorTemperature
  - GLOBAL_NVS_AREA.ApDigitalThermalSensorTemperature
- Updates thermal thresholds (**IA32_THERM_INTERRUPT**) based on current core temperature for further thermal SMI generation
- Asserts Software General Purpose Event (SCI interrupt) to notify OS Power Management (OSPM)

Platform ACPI Table

Several data structures and devices are required in ACPI DSDT to support thermal zones.

See https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Acpi/AcpiTablesPCA

Global NVS Memory Operation Region

```
OperationRegion (GNVS, SystemMemory, 0x00000000, 0xAA55)
Field (GNVS, AnyAcc, Lock, Preserve)
{
      …
      DTS1,    8,     //   (06) Digital Thermal Sensor 1 Reading
      DTS2,    8,     //   (07) Digital Thermal Sensor 2 Reading
      …
}
```

This code example maps to the global data structure (**GLOBAL_NVS_AREA**), while DTS1 and DTS2 point to **BspDigitalThermalSensorTemperature** and **ApDigitalThermalSensorTemperature** respectively.

The DTS SMI handler reads the current core temperatures from DTS and updates these fields. The ACPI thermal zone method **_TZ._TMP** consumes these temperature values.

Thermal Zone in ACPI DSDT

The thermal zone method **_TZ._TMP** reports the higher value of GNVS.DTS1 and GNVS.DTS2 to OSPM as current system temperature.

See https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Acpi/AcpiTablesPCAT/THERMAL.ASL

```
Scope (\_TZ)
{
   // Thermal Zone 1 = DTS Thermal Zone.
   ThermalZone (TZ01)
   {
       ...
       Method (_TMP,0,Serialized)
       {
          If (DTSE) {
            //
            // If DTS support is enabled, simply return the higher of the two
            // DTS Temperatures.
            //
            If (LGreaterEqual (DTS1, DTS2)) {
              Return (Add(2732, Multiply(DTS1, 10)))
            }
            Return(Add(2732, Multiply(DTS2, 10)))
          }
      ...
      } // End of _TMP
      ...
    }// end ThermalZone(TZ01)
} // end Scope(\_TZ)
```

General Purpose Event in ACPI DSDT

General purpose event handler **_GPE._L02** corresponds to the SCI asserted by the DTS thermal SMI handler. This handler notifies OSPM that a thermal event has been triggered. OSPM then can read the temperature via **_TZ._TMP** and handle the corresponding event.

See https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Common/Acpi/AcpiTablesPCAT/Gpe.asl

```
Scope (\_GPE)
{
  //
  // Software GPE caused the event.
  //
  Method (_L02)
  {
    // Clear GPE status bit.
    Store (0,GPEC)
    //
    // Handle DTS Thermal Events.
    //
    External(DTSE, IntObj)
```

```
If(CondRefOf(DTSE))
    {
      If (LGreaterEqual (DTSE, 0x01))
      {
        Notify (\_TZ.TZ01,0x80)
      }
    }
  }
```

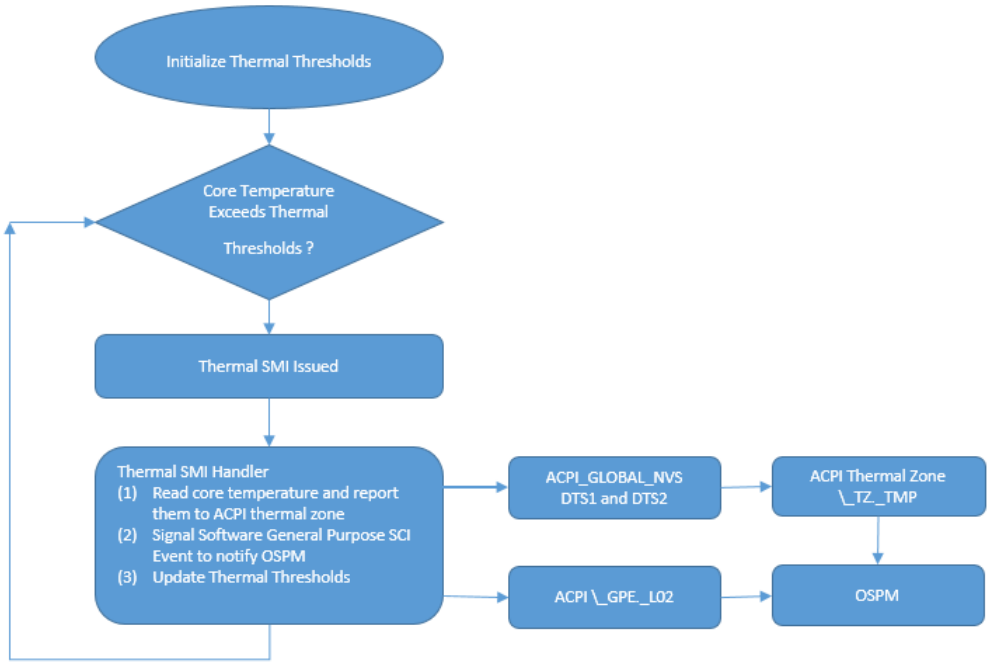
Thermal Management Logic Flow

UEFI IA Firmware initializes the high and low thermal thresholds, and registers for the SMI thermal interrupt handler, which is triggered when a threshold is exceeded. The thermal SMI handler performs the following tasks:

- Reads core temperatures
- Fills into corresponding global variables for ACPI Thermal Zone
  - **`GLOBAL_NVS_AREA.BspDigitalThermalSensorTemperature`**
  - **`GLOBAL_NVS_AREA.ApDigitalThermalSensorTemperature`**
- Signals a Software General Purpose event (SCI interrupt) to notify OSPM that a thermal event has occurred
- Calculates new thermal thresholds based on current temperature, and updates threshold registers

Figure 10 illustrates the thermal management logical flow.

Figure 10: Thermal Management Logical Flow



Board Level Configuration

Board Specific Configuration

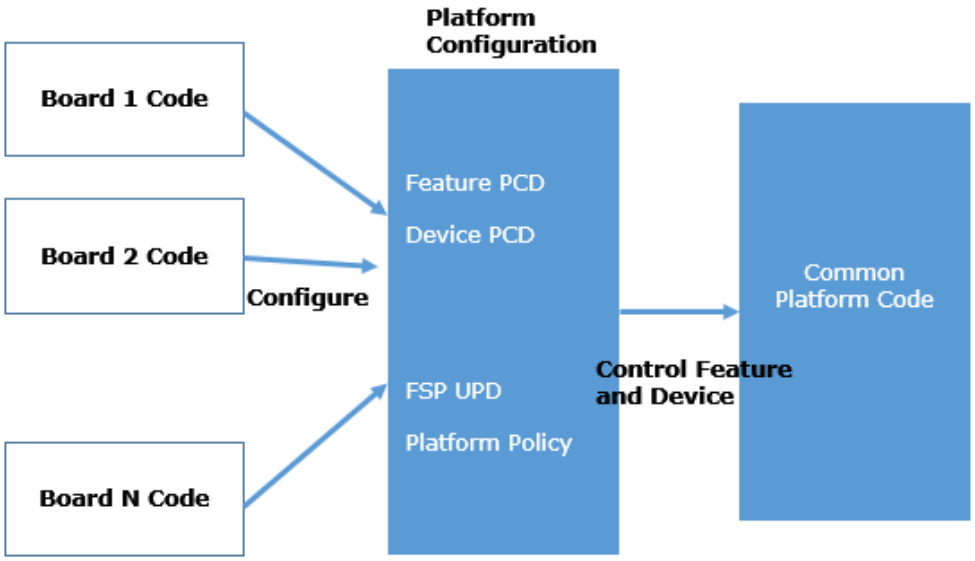
Board specific configuration files are located in the "Board" sub-folder:
https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Board

Detection of "Board ID" and "FAB ID" is supported, so specific code blocks can be executed based on these values. This feature allows a single BIOS binary image to support multiple platforms. Board-specific code can be executed based on the detected Board ID and FAB ID, such as the following:

- Program GPIO
- Switch on/off patchable PCD
- Update UPD for Intel FSP

Code outside of the "Board" folder is common platform code, and should not have any dependency on Board ID or FAB ID. Common platform code consumes PCDs which have been configured by board specific code to control common features and devices.

Figure 11: Board Specific Configuration



Example: Configuration for LPDDR4 "Memory Down" (On Board Memory)

One common configuration change is related to platform memory configuration. This example shows changing LPDDR4 related-fields in FSP-M (FSP_M_CONFIG) for a specific platform:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Board/AuroraGlacier/BoardInitPreMem/BoardInitMiscs.c

```
EFI_STATUS
EFIAPI
AuroraUpdateFspmUpd (
  IN CONST EFI_PEI_SERVICES  **PeiServices,
  IN FSPM_UPD                *FspUpdRgn
  )
```

This example is based on Micron Mobile LPDDR4 SDRAM (MT53B256M32D1, MT53B512M32D2, MT53B1024M32D4).

Memory Controller LPDDR4 Interface

The Intel Atom® Processor E3900 Series memory controller has four x32 LPDDR4 channels. Each channel's data bus width is 32 bit. 32 bit is also the data bus width of memory Rank, and each channel can be connected to two memory Ranks.

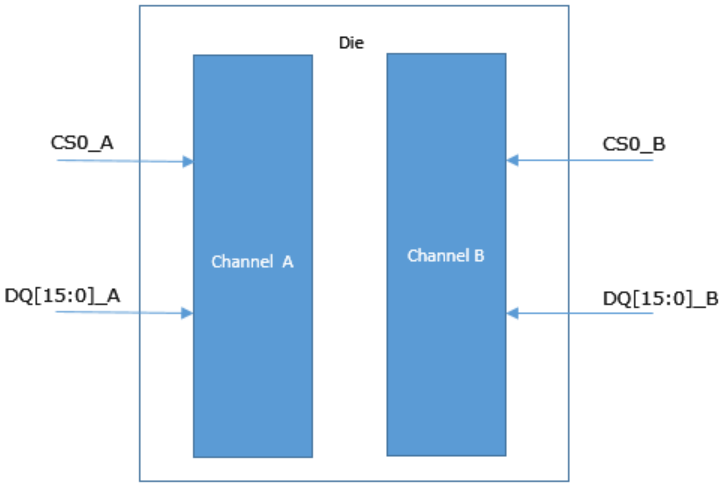
Table 15: Critical Memory Controller Signals

Some Signal Name	Description
MEM_CH0_DQA[31:0]	Channel 0 x32 data bus to the LPDDR4 DRAM data bus.
MEM_CH0_CKE[1:0]A	Channel 0 Clock Enable. 2 signal pins in total. One pin for each memory Rank.
MEM_CH0_CS[1:0]A	Channel 0 Chip Select. 2 signal pins in total. One pin for each memory Rank.
MEM_CH0_DQB[31:0]	Channel 1 x32 data bus to the LPDDR4 DRAM data bus.
MEM_CH0_CKE[1:0]B	Channel 1 Clock Enable. 2 signal pins in total. One pin for each memory Rank.
MEM_CH0_CS[1:0]B	Channel 1 Chip Select. 2 signal pins in total. One pin for each memory Rank.
MEM_CH1_DQA[31:0]	Channel 2 x32 data bus to the LPDDR4 DRAM data bus.
MEM_CH1_CKE[1:0]A	Channel 2 Clock Enable. 2 signal pins in total. One pin for each memory Rank.
MEM_CH1_CS[1:0]A	Channel 2 Chip Select. 2 signal pins in total. One pin for each memory Rank.
MEM_CH1_DQB[31:0]	Channel 3 x32 data bus to the LPDDR4 DRAM data bus.
MEM_CH1_CKE[1:0]B	Channel 3 Clock Enable. 2 signal pins in total. One pin for each memory Rank.
MEM_CH1_CS[1:0]B	Channel 3 Chip Select. 2 signal pins in total. One pin for each memory Rank.

LPDDR4 SDP (Single-Die, Dual-Channel Package)

The memory package shown in Figure 12 has one single die. There are two memory devices on each die. Each device has an x16 data bus width, and its own dedicated Chip Select (CS) signal (not be confused with Chip Select signal of the SoC memory controller).

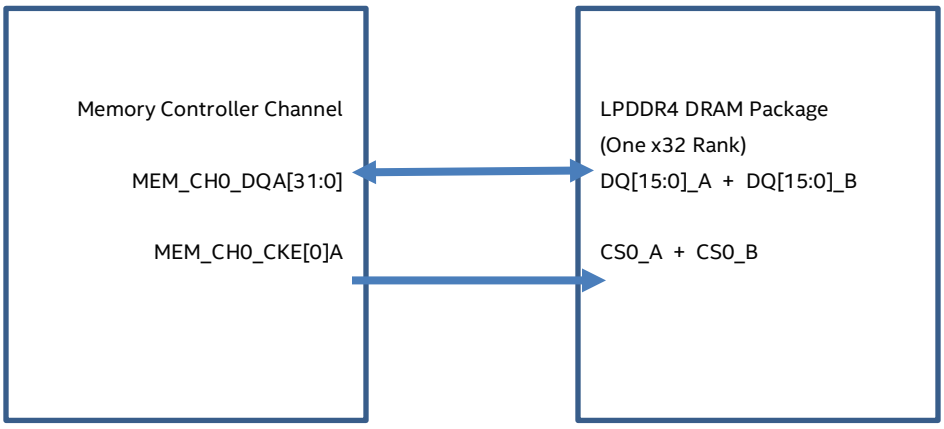
Figure 12: Micron Single Die, Dual Channel Package



The term “channel” refers to the DQ signal arrays (not be confused with a Channel of the SoC memory controller.)

The SoC memory controller requires LPDDR4 SDRAM with x32 rank, so two x16 devices in the same die will be combined to form a x32 memory rank. Their Chip Select signals, **CS0_A** and **CS0_B**, will be merged and connected to the SoC memory controller Chip Select signal.

Figure 13: Memory Controller Channel



For the memory configuration shown in Figure 13, the following Intel FSP parameters are required:

```
FSP_M_CONFIG ->FspmConfig.Ch0_RankEnable = 0x01; // [0]: Rank 0; [1]: Rank 1
FSP_M_CONFIG ->FspmConfig.Ch0_DeviceWidth = 0x01; // x16 device width
```

Note: the above code is board-specific and added into board specific function AuroraUpdateFspmUpd(). Please refer to below board specific file for detail:

https://github.com/tianocore/edk2-platforms/blob/devel-IntelAtomProcessorE3900/Platform/BroxtonPlatformPkg/Board/AuroraGlacier/BoardInitPreMem/BoardInitMiscs.c

LPDDR4 DDP (Dual-Die, Dual-Channel Package)

Figure 14: Micron Dual Die, Dual Channel Package

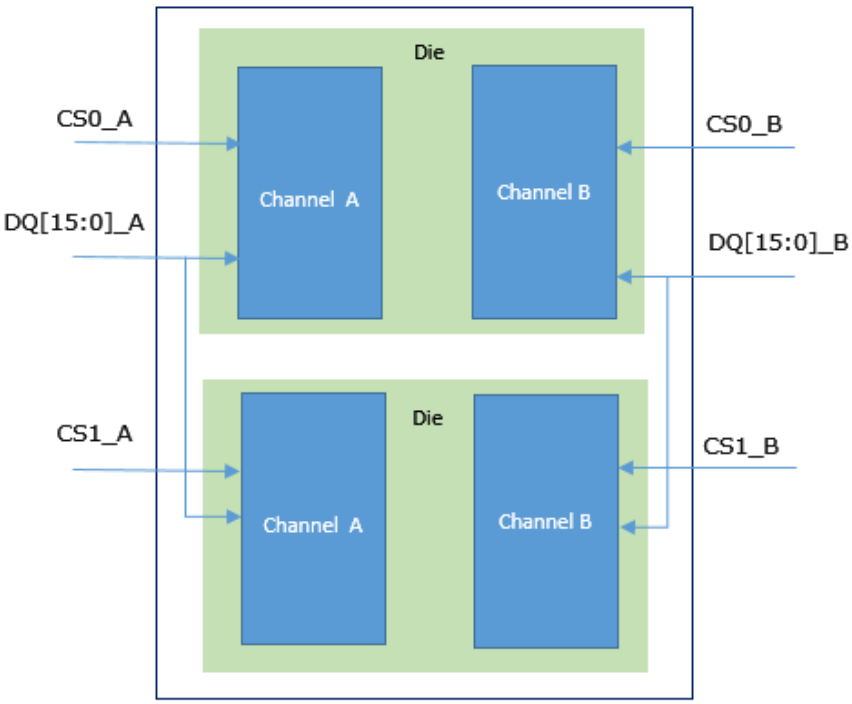


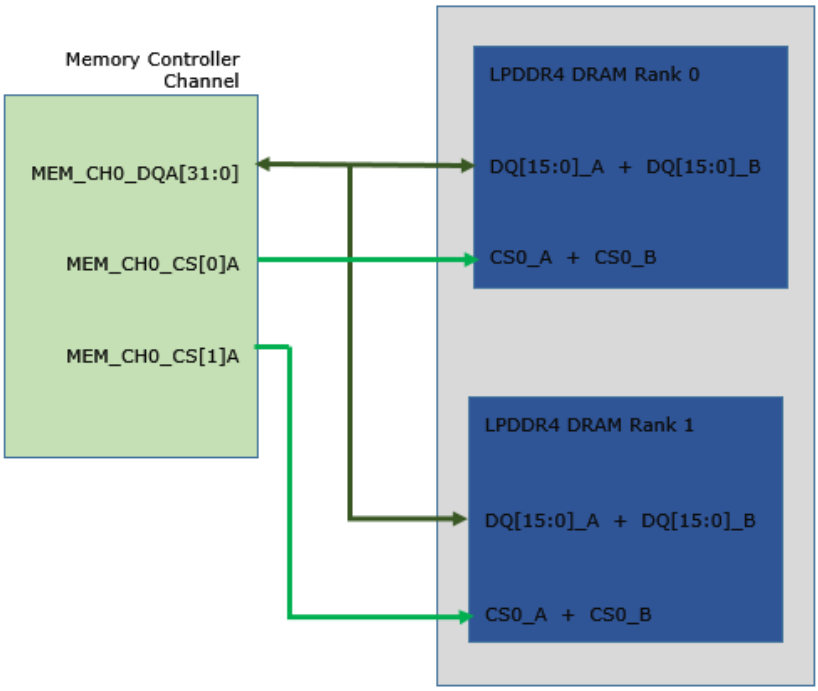
Figure 14 shows a slightly different memory configuration than Figure 12. The memory "package" has two "dies".

The Intel Atom® Processor E3900 Series memory controller requires LPDDR4 SDRAM with x32 Rank. So in this example, two x16 DQ in the same die will be combined to form an x32 memory rank. The Chip Select signals are merged and connected to the SoC memory controller Chip Select (Table 16). A DDP package provides two x32 memory Ranks, which will be connected to a single channel of the SoC memory controller (Figure 14).

Table 16: Memory Rank Configuration

Rank	Combined DQ	Merged CS	Combined LPDDR4 Device
Rank 0	DQ[15:0]_A DQ[15:0]_B	CS0_A	Die 0, Channel A
		CS0_B	Die 0, Channel B
Rank 1	DQ[15:0]_A DQ[15:0]_B	CS1_A	Die 1, Channel A
		CS1_B	Die 1, Channel B

Figure 15: Memory Controller Channel



For the memory configuration shown in Figure 15, the following Intel FSP parameters are required:

```
FSP_M_CONFIG ->FspmConfig.Ch0_RankEnable = 0x03; // [0]: Rank 0; [1]: Rank 1
FSP_M_CONFIG ->FspmConfig.Ch0_DeviceWidth = 0x01; // x16 device width
```

Enabling Verified Boot

This section describes the implementation of Verified Boot on Intel Atom® Processor E3900 Series platforms using Intel TXE. Enabling Verified Boot includes the process of manifesting, signing and assembling firmware components. This document does not cover all aspects or features of Intel TXE.

Root-of-Trust & System Firmware Integrity

System firmware components are typically located in re-programmable storage devices, such as eMMC NAND Flash or SPI NOR Flash. The re-programmable storage mechanism provides flexibility for firmware updating, but also introduces the risk of firmware attack. Attackers may attempt to replace firmware components, either through software or hardware methods, with their own malicious firmware.

System firmware is a key element in establishing platform root-of-trust, since it initializes the platform after reset and launches the operating system. For a secure platform implementation, system firmware must be verified prior to execution. This requires storing a root-of-trust element on a non-reprogrammable device, such as read-only memory (ROM) or memory with a write-once fuse.

Verified Boot is a feature of Intel TXE used to authenticate firmware components, ensure they originate from trusted sources, and verify their integrity. This verification is necessary for firmware components stored on reprogrammable media devices.

The root-of-trust for Intel TXE is stored in read-only memory (ROM), in combination with an Original Equipment Manufacturer (OEM) public key hash stored in a Field Programmable Fuse (FPF). The Intel TXE ROM and FPF are write-once storage media which can never be reprogrammed, guaranteeing data integrity after shipment. If any firmware component cannot pass identification authentication or integrity verification, Intel® TXE prevents execution of that component.

Root-of-trust components for Verified Boot are the Intel TXE ROM and OEM key fused in FPF. Firmware components in SPI flash are authenticated prior to execution. These components include the firmware executed by Intel TXE, platform UEFI firmware (IA firmware), and peripheral drivers. Firmware components, the OEM Key Manifest, and other OEM data stored in SPI flash must be signed and manifested by either Intel or the OEM prior to distribution. A hash of the OEM root public key must be burned into FPF at the end of manufacturing stage by OEM.

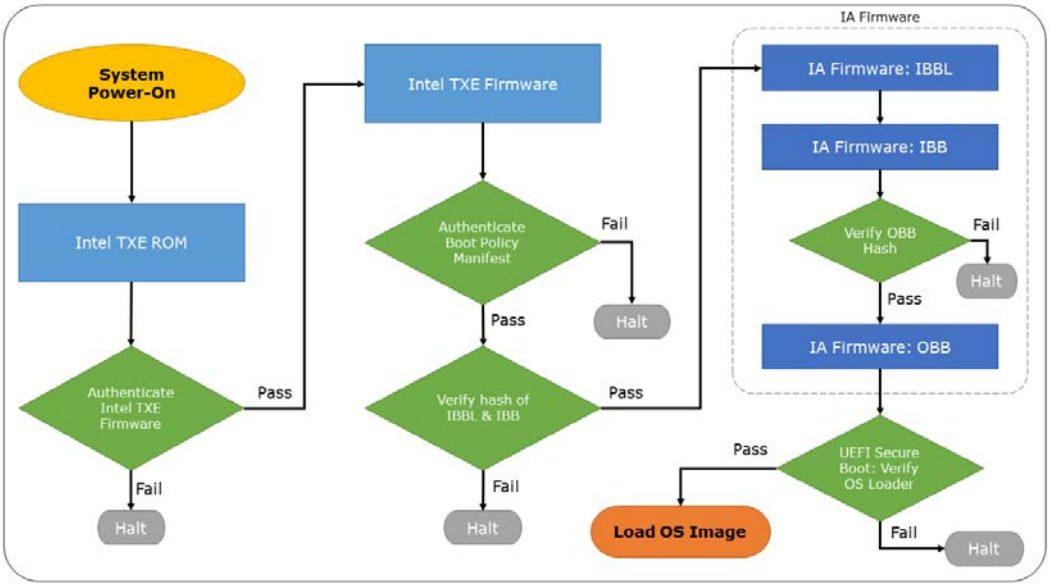
Boot Flow with Verified Boot

The firmware boot flow with TXE Verified Boot enabled is more complex than boot flow without TXE Verified Boot.

1. After platform power on, Intel TXE and the PMC come out of reset and execute their respective firmware. Intel TXE runs its code from the Intel TXE ROM.
2. Intel TXE ROM loads TXE firmware from SPI flash and authenticates the digital signature. If signature authentication is successful, Intel TXE ROM will hand off control to the Intel TXE Firmware. Otherwise the boot process will halt.
3. Intel TXE Firmware authenticates the Boot Policy Manifest of the IA Firmware. If the Boot Policy Manifest passes signature authentication, Intel TXE loads IBBL into Shared SRAM and verifies the hash of IBBL and IBB. If the hash verification passes, Intel TXE will notify the Power Management Controller to de-assert processor reset. Otherwise, Intel TXE will not de-assert processor reset.
4. After reset, the processor starts executing IBBL for early SoC initialization.
5. IBBL code calls Intel TXE to load IBB into temporary RAM (Cache as RAM).

6. IBB initializes system memory (DRAM), copies OBB from SPI NOR flash into DRAM, and verifies the OBB hash. If OBB passes hash verification, IBB will hand off control to OBB. Otherwise, IBB will stop the boot process.
7. The IA processor executes OBB. OBB installs UEFI boot services, creates system tables (ACPI, SMBIOS), installs UEFI runtime services, and locates the OS Boot Loader on a mass storage device.
8. If UEFI Secure Boot is enabled, OBB authenticates the OS Boot Loader. If the Boot Loader passes signature authentication of UEFI Secure Boot, OBB will load the OS.

Figure 16: Firmware Boot Flow, Verified Boot Enabled



Example Attack Scenario: Altered OBB Image

Attacker reads firmware from the SPI flash, then uses specific software tools to insert a malicious System Management Mode (SMM) UEFI driver into the OBB region of the IA firmware. The attacker flashes the altered firmware image into the SPI flash.

If Verified Boot is not enabled, the system could still boot to the OS. The risk is that the malicious SMM UEFI driver will take control at OS runtime when a SMI is triggered and the CPU transitions SMM mode, which has the ability to access system memory and compromise confidential data.

If Verified Boot is enabled, IBB will measure the hash value of OBB and discover the OBB image has been altered. This failed verification causes the system to halt during IBB. The board will need to be reprogrammed with firmware signed by the OEM.

Verified Boot Ingredients

Verified Boot Firmware Components

Intel TXE ROM

Vendor: Intel Corporation

Storage: TXE, Non-reprogrammable

Intel TXE includes read-only memory (ROM) containing initialization and cryptography code. When Intel TXE is powered on, execution starts from the Intel TXE ROM.

The firmware code in the Intel TXE ROM is programmed during SoC manufacturing. The ROM cannot be changed once it is programmed by Intel at the factory.

The Intel TXE ROM also includes code for verifying firmware loaded from SPI flash. Intel TXE uses an Intel owned key contained in the ROM to authenticate the Intel TXE Firmware. Intel TXE ROM functions as the platform root-of-trust.

Intel TXE Firmware

Vendor: Intel Corporation

Storage: SPI Flash, Reprogrammable

Intel TXE Firmware runs on Intel TXE, implementing additional system bring-up code and security applications. The Intel TXE Firmware also includes code for loading and authenticating platform firmware during boot. Intel TXE Firmware for MinnowBoard 3 is stored in the SPI flash device.

Intel Architecture (IA) Firmware

Vendor: OEM

Storage: SPI NOR Flash, Reprogrammable

IA Firmware is the first piece of code executed by the IA processor coming out of reset. This open source project is based on EDK II available from tianocore.org, which follows the UEFI and UEFI PI Specifications.

The IA Firmware image is divided three stages. Initial Boot Block Loader (IBBL), Initial Boot Block (IBB), and OEM Boot Block (OBB).

IBBL is verified by TXE Firmware and loaded into Shared SRAM by TXE. IBBL triggers TXE Firmware to verify and load IBB from SPI NOR flash into CPU cache, using “Cache as RAM” as storage. IBB verifies OBB and loads OBB from SPI NOR flash into DRAM.

IA Firmware is stored in the SPI NOR flash.

Power Management Controller (PMC) Firmware

Vendor: Intel

Storage: SPI Flash, Reprogrammable

This firmware is used to initialize the PMC, which controls platform power planes.

Integrated Firmware Image (IFWI)

The OEM uses Intel FIT tool stitches all firmware components together to generate the IFWI binary, which is programmed into the IFWI region SPI NOR flash.

Verified Boot Keys

Intel TXE Verified Boot uses RSA 2048 public key infrastructure to sign and verify firmware components. The firmware vendor must sign the firmware before distribution and post a public key. Figure 17 lists the firmware components and corresponding key storage locations.

Figure 17: Intel TXE Keys, Owners, and Storage Locations

Firmware Component	Private Key Owner	Public Key Consumer	Location of Public Key Hash
Intel TXE Firmware	Intel	Intel TXE	Intel TXE ROM
PMC Firmware	Intel	Intel TXE	Intel TXE ROM
OEM Key Manifest	OEM	Intel TXE	SoC FPF
Boot Policy Manifest (IA Firmware Hash Inside)	OEM	Intel TXE	OEM Key Manifest
Audio Firmware	OEM	Intel TXE	OEM Key Manifest
Camera Firmware	OEM	Intel TXE	OEM Key Manifest

Public key hash values for Intel components are stored in the Intel TXE ROM. Public key hash values for OEM firmware components are stored in OEM Key Manifest in SPI flash. Hash of public key for OEM Key Manifest is stored in SoC FPF fuse. The vendor securely stores the private key used to sign the firmware.

Public Key for OEM Key Manifest

The OEM Key Manifest contains the public key hash for all OEM firmware components in the SPI NOR flash. The OEM Key Manifest must be signed to protect the integrity of stored OEM keys.

The public key hash for authenticating the OEM Key Manifest is programmed into FPF at the end of manufacturing. The FPF is a “write-once” chip. Once the OEM public key’s hash is programmed into FPF, it can never be reprogrammed. The OEM claims ownership of the platform by fusing the key into FPF. The OEM key stored in FPF can be treated as root-of-trust for OEM components.

Public Keys for OEM Firmware Components

OEM firmware components on SPI NOR flash or other reprogrammable media must be signed. The OEM signs OEM firmware components with OEM private keys, and inserts the corresponding OEM public key hash values in the IWFI OEM Key Manifest.

Prior to execution, Intel TXE authenticates each OEM firmware component based on the corresponding public key stored in the OEM Key Manifest.

Manifesting and Signing Firmware Components

To enable Verified Boot, firmware components have to be manifested and signed. The Intel TXE firmware release package provides an Intel Manifest Extension Utility (MEU) to support manifesting and signing different types of files (e.g. Boot Policy Manifest, OEM Key Manifest, audio firmware, etc.). The Apollo Lake Signing and Manifesting Guide, which is part of the Intel® TXE Firmware (FW) Production Version (PV) release package, gives detailed instructions on how to use Intel MEU.

Signing and Manifesting Tools

OpenSSL

OepnSSL is available for download: http://www.openssl.org/source

Intel Manifest Extension Utility (MEU)

The Intel Manifest Extension Utility (MEU) is used to sign and manifest firmware binaries. The manifest tool leverage OpenSSL to sign a file.

Intel MEU is part of the Intel® TXE Firmware (FW) Production Version (PV) release package version 3.0.20.1139 (kit number 120568), which is available on https://platformsw.intel.com. Customers could contact Intel representative for information on downloading the release package.

Assembling the Full IFWI Image

After each firmware component has been manifested and singed, the OEM stitches them together to form an 8MB SPI Flash Image to program into the SPI NOR flash.

Intel Firmware Image Tool (FIT)

The Intel firmware Image tool (FIT), which is part of the Intel® TXE Firmware (FW) Production Version (PV) release package, assembles firmware components into a single firmware image.

- Please download Intel® TXE Firmware (FW) Production Version (PV) release package 3.0.20.1139 (kit number 120568) from https://platformsw.intel.com or contact your Intel representative to get FIT.
- Please follow the Signing and Manifesting Guide within the release package to enable Boot Guard.
- Please follow the Intel® Trusted Execution Engine (Intel® TXE) Firmware Bring-Up Guide in the release package to create a full IFWI image.