

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

September 20, 2018

---

Intel provides open source UEFI firmware for the Intel Atom® Processor E3900 Series, Intel® Pentium® Processor N4200, and Intel® Celeron® Processor N3350 platforms (formerly Apollo Lake SoCs). This document describes changes commonly made when adapting the firmware for customized versions of supported platforms.

## Authors

---

Xianhui Liu  
David Wei  
Mike Wu  
Brian Richardson

## Contents

---

Introduction.....	1
Defining a New Board.....	6
Hardware-Related Changes.....	11
Other Platform Customizations .....	21
Revision History.....	31

## Overview

---

The purpose of this white paper is to describe changes commonly made when adapting Intel-provided UEFI firmware for customized hardware platforms based on the Intel products mentioned above. The firmware codebase supports multiple hardware designs, but examples in this document will refer to two specific platforms:

- MinnowBoard 3 Module, an open source platform encouraging experimentation and derivative designs ([minnowboard.org](http://minnowboard.org))
- Leaf Hill, an Intel customer reference board (CRB)

<https://firmware.intel.com/projects/IntelAtomProcessorE3900>

## Intended Audience

---

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

## Prerequisites

---

- Users of this document should have prior firmware development experience using UEFI & EDK II, including knowledge of the UEFI Specifications available at [uefi.org/specifications](http://uefi.org/specifications).
- Users should also be familiar with the Intel® Firmware Support Package (FSP), available at [intel.com/fsp](http://intel.com/fsp).
- This document also assumes the reader has experience with Intel® 64 and IA-32 Architectures.

## Related Documents

---

- Open Source UEFI Firmware Enabling Guide: Intel Atom® Processor E3900 Series Platforms  
[https://firmware.intel.com/sites/default/files/uefi\\_firmware\\_enabling\\_guide\\_for\\_the\\_intel\\_atom\\_processor\\_e3900\\_series.pdf](https://firmware.intel.com/sites/default/files/uefi_firmware_enabling_guide_for_the_intel_atom_processor_e3900_series.pdf)
- 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 3 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>
- Unified Extensible Firmware Interface (UEFI) Specification, Advanced Configuration and Power Interface (ACPI) Specification, and Platform Initialization (PI) Specification:  
<http://www.uefi.org/specifications>
- 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](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>

## Acronyms and Terminology

Acronym / Term	Definition
<b>ACPI</b>	Advanced Configuration and Power Interface
<b>API</b>	Application Programming Interface
<b>Apollo Lake</b>	The pre-release code name for Intel Atom® Processor E3900 Series CPUs
<b>BCT</b>	Binary Configuration Tool
<b>BIOS</b>	Basic Input/Output System, a common term for the legacy Intel Architecture (IA) boot firmware. UEFI has replaced BIOS on most systems, with legacy BIOS support until 2020.
<b>BPDT</b>	Boot Partition Description Table
<b>BSF</b>	Boot Setting File
<b>CAR</b>	Cache as RAM
<b>CRB</b>	Customer Reference Board
<b>CPU</b>	Central Processing Unit
<b>CS</b>	Chip Select
<b>DQ</b>	Data Quality
<b>DRAM</b>	Dynamic Random Access Memory
<b>DSDT</b>	Differentiated System Description Table
<b>DTS</b>	Digital Thermal Sensor
<b>DXE</b>	Driver Execution Environment
<b>EDK II</b>	EDK II is a cross-platform firmware development environment for the UEFI and PI specifications.
<b>eMMC</b>	Embedded Multi-Media Controller
<b>FDL</b>	Flash Description File
<b>FIT</b>	Intel Flash Image Tool
<b>FSP</b>	Intel® Firmware Support Package. See <a href="http://intel.com/fsp">http://intel.com/fsp</a>
<b>FW</b>	Firmware
<b>GPIO</b>	General Purpose Input-Output
<b>HECI</b>	Host Embedded Controller Interface
<b>HFM</b>	Hyperbolic Frequency-Modulated waveform
<b>HOB</b>	Hand-off Block
<b>IA</b>	Intel Architecture
<b>I2C</b>	Inter-Integrated Circuit
<b>IBB</b>	Initial Boot Block
<b>IBBL</b>	Initial Boot Block Loader
<b>IFWI</b>	Integrated Firmware Image
<b>IPU</b>	Interconnect Processing Unit
<b>LBP</b>	Logical Boot Partition
<b>LFM</b>	Linear Frequency-Modulated waveform

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

## List of Figures

---

Figure 1: Guidgen Utility.....	7
Figure 2: PMIC/VR Configuration with FIT Tool.....	19
Figure 3: Flex I/O Configuration Using Intel FIT .....	20
Figure 4: Microcode Update with Intel® FIT .....	20
Figure 5: FSP MemoryInit Settings Using Intel® FIT .....	21
Figure 6: SiliconInit Settings using Intel BCT.....	22

## List of Tables

---

Table 1: SMBIOS Structure Table for Platform.....	27
---	----

## Defining a New Board

---

This section describes the process of adding a new platform (“board”) to the existing firmware project, based on a reference design.

### Download Project Source Code

---

Source code, binary images, and tools required to build the firmware project are available from the Intel® Architecture Firmware Resource Center, at <https://firmware.intel.com/projects/IntelAtomProcessorE3900>.

You need to download and build the default firmware configuration for Leaf Hill or MinnowBoard 3 Module defined in the ‘Build Instructions’ file before making any firmware customizations.

Example: build a 64-bit release image for “Leaf Hill” using Visual Studio 2015

```
BuildBIOS.bat /vs15 /LH /D /x64 Broxton Release
```

### Add a New Board to the Project

---

The firmware project supports defining multiple boards under a single tree. Platforms with minor variations can share common code, which is an easy way to build firmware based on a reference platform. Board definitions are under the “Board” directory:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board
```

Developers can clone the reference project by copying the directory, renaming the directory, and adding references in several configuration files.

### Copy Existing Board Directory as Reference

---

In this example, we will create a new platform (“New Board”) based on the Leaf Hill design. Locate the Leaf Hill board directory in the firmware project:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\LeafHill
```

Copy the `LeafHill` directory into a new folder:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard
```

### Changes to Functions, Variables, and File GUIDs

---

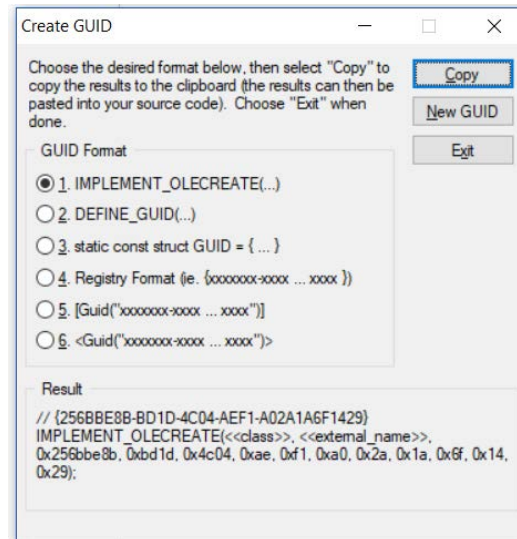
Functions and global variables of libraries under the `NewBoard` folder need to be changed so they do not conflict with libraries of existing boards. These are referenced in `.INF` files copied from the reference project:

```
BoardInitPreMem.inf  
BoardInitPostMem.inf  
BoardInitDxe.inf
```

UEFI and EDK II associate a globally unique identifier (GUID) with various functions, files, and protocols. New file GUIDs must also be generated for `.INF` files under the `NewBoard` folder to avoid conflict with

GUIDs from existing projects. The `guidgen` utility included with Microsoft Visual Studio, as shown in Figure 1, is recommended for GUID generation for Microsoft Windows development environments.

**Figure 1: Guidgen Utility**



The following examples show use of new GUIDs and function names in 'NewBoard':

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard
\BoardInitDxe\BoardInitDxe.inf
```

```
[Defines]
  INF_VERSION                = 0x00010005
  BASE_NAME                  = NBBoardInitDxe
  FILE_GUID                  = D406820E-9CD3-45b3-8D83-74C6CFFB7C80

  MODULE_TYPE                = DXE_DRIVER
  VERSION_STRING             = 1.0
  LIBRARY_CLASS              = NULL|DXE_DRIVER DXE_RUNTIME_DRIVER
DXE_SAL_DRIVER DXE_SMM_DRIVER UEFI_APPLICATION UEFI_DRIVER
  CONSTRUCTOR                = NBBoardInitDxeConstructor
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard
\BoardInitDxe\ BoardInitDxe.c
```

```
NBGetBoardName (
  IN UINT8                      BoardId
)
NBBoardInitDxeConstructor (
  IN EFI_HANDLE                ImageHandle,
  IN EFI_SYSTEM_TABLE          *SystemTable
)
```

## Add Components to DSC File

### Add DXE Library

edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Components.dsc

Add path to the `BoardInitDxe.inf` file under `NewBoard`.

```
$(PLATFORM_PACKAGE_COMMON)/PlatformSettings/PlatformSetupDxe/PlatformSetupDxe
.inf {
  <LibraryClasses>
    NULL|$(PLATFORM_NAME)/Board/MinnowBoard3/BoardInitDxe/BoardInitDxe.inf
    NULL|$(PLATFORM_NAME)/Board/LeafHill/BoardInitDxe/BoardInitDxe.inf
    NULL|$(PLATFORM_NAME)/Board/NewBoard/BoardInitDxe/BoardInitDxe.inf
    NULL|$(PLATFORM_NAME)/Board/BensonGlacier/BoardInitDxe/BoardInitDxe.inf
    NULL|$(PLATFORM_NAME)/Board/AuroraGlacier/BoardInitDxe/BoardInitDxe.inf

  NULL|$(PLATFORM_NAME)/Board/MinnowBoard3Next/BoardInitDxe/BoardInitDxe.inf
}
```

### Add PEI Libraries

edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\
Components.IA32.dsc

Add references to `BoardInitPreMem.inf` and `BoardInitPostMem.inf` to the DSC file.

```
$(PLATFORM_PACKAGE_COMMON)/PlatformSettings/PlatformPreMemPei/PlatformPreMemP
ei.inf {
  <LibraryClasses>

  NULL|$(PLATFORM_NAME)/Board/MinnowBoard3/BoardInitPreMem/BoardInitPreMem.inf

  NULL|$(PLATFORM_NAME)/Board/LeafHill/BoardInitPreMem/BoardInitPreMem.inf

  NULL|$(PLATFORM_NAME)/Board/NewBoard/BoardInitPreMem/BoardInitPreMem.inf

  . . .

  $(PLATFORM_PACKAGE_COMMON)/PlatformSettings/PlatformPostMemPei/PlatformPostMe
mPei.inf {
    <LibraryClasses>

  NULL|$(PLATFORM_NAME)/Board/MinnowBoard3/BoardInitPostMem/BoardInitPostMem.in
f

  NULL|$(PLATFORM_NAME)/Board/LeafHill/BoardInitPostMem/BoardInitPostMem.inf

  NULL|$(PLATFORM_NAME)/Board/NewBoard/BoardInitPostMem/BoardInitPostMem.inf
```



## Path to Binary Stitching Files

---

Modify the post-build stitch file to recognize the new platform's `BoardId` value ('NB') and location of IFWI binary stitching files (`SpiChunkN.bin`) if they differ from the versions provided by the reference project. Changes differ for Microsoft Windows & Linux build environments.

Microsoft Windows build environment:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Tools\Stitch\
IFWIStitch_Simple.bat
```

```
if /i "%~3"=="LH" (
    set BoardId=LH
)

if /i "%~3"=="NB" (
    set BoardId=NB
)

else if %BoardId%==MN (
    if %FabId%==B (
        copy /y /b ..\..\..\Board\MinnowBoard3\IFWI\FAB_B\SpiChunk1.bin .
        copy /y /b ..\..\..\Board\MinnowBoard3\IFWI\FAB_B\SpiChunk2.bin .
        copy /y /b ..\..\..\Board\MinnowBoard3\IFWI\FAB_B\SpiChunk3.bin .
        copy /y /b
SpiChunk1.bin+.\BIOS_COMPONENTS\IBBL.Fv+.\BIOS_COMPONENTS\IBB.Fv+SpiChunk2.bi
n+.\BIOS_COMPONENTS\OBB.Fv+.\BIOS_COMPONENTS\NvStorage.Fv+SpiChunk3.bin
spi_out.bin
    )
)
```

Linux build environment:

```
edk2-platforms\BuildBIOS.sh
```

```
elif [ "$(echo $1 | tr 'a-z' 'A-Z')" == "/NB" ]; then
    BoardId=NB
    Build_Flags="$Build_Flags /NB"
    shift
```

```
edk2-platforms\Platform\BroxtonPlatformPkg\BuildBxtBios.sh
```

```
elif [ "$(echo $1 | tr 'a-z' 'A-Z')" == "/NB" ]; then
    BoardId=NB
    Shift
```

```
elif [ $BoardId == "NB" ]; then
    BOARD_ID=NEWBOARD
    echo BOARD_ID = NEWBOARD >> $WORKSPACE/Conf/BiosId.env
```

```
if [ $BoardId == "NB" ]; then
    if [ $FabId == "A" ]; then
        BOARD_REV=A
        echo BOARD_REV = A >> $WORKSPACE/Conf/BiosId.env
```

```
    fi
fi

if [ $BoardId == "NB" ]; then
    if [ $FabId == "A" ]; then
        cp -f
        $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Board/NEWBOARD/IFWI/FAB_A/SpiChunk
        1.bin $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Common/Tools/Stitch
        cp -f
        $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Board/NEWBOARD/IFWI/FAB_A/SpiChunk
        2.bin $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Common/Tools/Stitch
        cp -f
        $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Board/NEWBOARD/IFWI/FAB_A/SpiChunk
        3.bin $PLATFORM_PATH/Platform/BroxtonPlatformPkg/Common/Tools/Stitch
    fi
fi

edk2-platforms\Platform\BroxtonPlatformPkg\BuildIFWI.sh

elif [ "$(echo $1 | tr 'a-z' 'A-Z')" == "/NB" ]; then
    BoardId=NB
    Build_Flags="$Build_Flags /NB"
    Shift
```

## Test Build Process for the New Board

---

Test the changes using the build process described in Section 2.1, using the `BoardId` for “New Board” (`NB`) instead of “Leaf Hill” (`LH`). This should complete without error.

```
BuildBIOS.bat /vs15 /NB /D /x64 Broxton Release
```

The binary SPI image (8 MB) of the firmware will be located in the following directory:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Tools\Stitch.
```

Verify this operation before making further changes to the firmware.

## Hardware-Related Changes

---

Section 2 introduced the concept of “cloning” an existing project as reference to build a UEFI IA Firmware image for a custom board. The next step is editing the cloned project based on the board configuration. This section covers required UEFI IA Firmware changes for custom platforms that vary from a reference hardware design.

1. Detection of Board ID & Fab ID
2. Change UART serial port for UEFI IA Firmware debug messages
3. Change system memory parameters
4. Change display devices, peripherals
5. Modify I/O & GPIO configuration
6. Microcode updates

Examples are based on open source project code available in Tianocore github:

<https://github.com/tianocore/edk2-platforms/tree/devel-IntelAtomProcessorE3900>

Code modified for a new platform should reside in the corresponding board directory. This document uses the following path as an example for “NewBoard”:

`edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard`

### Detection of Board ID & Fab ID

---

BoardId and FabId values are typically defined by GPIO configuration, so the firmware can read the board-specific GPIO to determine their values.

### Define Board ID and FAB\_ID

---

`Platform\BroxtonPlatformPkg\Common\Include\Guid\PlatformInfo.h`

Define a unique value for “NewBoard”:

```
typedef enum {
    BOARD_ID_NEW_BOARD           = 0x01,      // NEW_BOARD
    BOARD_ID_MINNOW_NEXT        = 0x03,      // Minnow Board Next
    BOARD_ID_LFH_CRB            = 0x07,      // Leaf Hill
    BOARD_ID_MINNOW              = 0x0F,      // Minnow Board
    BOARD_ID_BENSON              = 0x0C,      // Benson Glacier
    BOARD_ID_AURORA              = 0x0E,      // Aurora Glacier
    BOARD_ID_APL_UNKNOWN        = 0xFF
} APL_BOARD_ID_LIST;
```

Replace BOARD\_ID\_LFH\_CRB with BOARD\_ID\_MY\_BOARD in files under the `NewBoard` folder.

Define FAB\_ID in FAB\_ID\_LIST data structure.

```
typedef enum {
    FAB_ID_RVP_B_C              = 0x0,      // For Broxton FAB B/C, special define
    FAB_ID_A                    = 0x1,      // FAB A
    FAB_ID_B                    = 0x2,      // FAB B
    FAB_ID_C                    = 0x3,      // FAB C
}
```

```

FAB_ID_D      = 0x4,      // FAB D
FAB_ID_E      = 0x5,      // FAB E
FAB_ID_F      = 0x6,      // FAB F
UNKNOWN_FAB   = 0xFF     // Unknown FAB
} FAB_ID_LIST;

```

## Get Board ID and FAB ID

Platform/BroxtonPlatformPkg/Board/NewBoard/BoardInitPreMem/PlatformId.c

Customize the `NewBoardGetEmbeddedBoardIdFabId()` function to detect the new Board ID and FAB ID values.

```

EFI_STATUS
EFIAPI
NewBoardGetEmbeddedBoardIdFabId(
    IN CONST EFI_PEI_SERVICES **PeiServices,
    OUT UINT8 *BoardId,
    OUT UINT8 *FabId
)
{
    BXT_CONF_PAD0 padConf0;
    BXT_CONF_PAD1 padConf1;
    IN UINT32 CommAndOffset;

    DEBUG ((DEBUG_INFO, "GetEmbeddedBoardIdFabId++\n"));

    //
    // Board_ID0: PMIC_STDBY
    //
    CommAndOffset = GetCommOffset (NORTHWEST, 0x00F0);
    padConf0.padCnf0 = GpioPadRead (CommAndOffset +
    BXT_GPIO_PAD_CONF0_OFFSET);
    padConf0.r.PMode = 0; // Set to GPIO mode
    padConf0.r.GPIORxTxDis = 0x1; // Set to GPI
    GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
    padConf0.padCnf0);
    padConf1.padCnf1 = GpioPadRead (CommAndOffset +
    BXT_GPIO_PAD_CONF1_OFFSET);
    //
    // Set to Pull Up 20K
    //
    padConf1.r.Term = 0xC;
    GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
    padConf1.padCnf1);
    //
    // Board_ID1: PMIC_SDWN_B
    //
    CommAndOffset = GetCommOffset (NORTHWEST, 0x00D0);
    padConf0.padCnf0 = GpioPadRead (CommAndOffset +
    BXT_GPIO_PAD_CONF0_OFFSET);
    padConf0.r.PMode = 0;
    padConf0.r.GPIORxTxDis = 0x1;
    GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
    padConf0.padCnf0);

```

```

//
// Board_ID2: PMIC_RESET_B
//
CommAndOffset = GetCommOffset (NORTHWEST, 0x00C8);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig0.r.PMode = 0;
padConfig0.r.GPIORxTxDis = 0x1;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);

//
// Board_ID3: GP_CAMERASB10
//

CommAndOffset = GetCommOffset (NORTH, 0x01E0);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig1.padCnf1 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF1_OFFSET);

padConfig0.r.PMode = M0; // Set to GPIO mode
padConfig0.r.GPIORxTxDis = GPI; // Set to GPI
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);

padConfig1.r.IOSTerm = EnPu; // Enable pull-up
padConfig1.r.Term = P_20K_H; // Set to 20K pull-up
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
padConfig1.padCnf1);

//
// Read out Board_ID
//
*BoardId = (UINT8) (((GpioPadRead (GetCommOffset (NORTHWEST, 0x00F0) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) | \
(((GpioPadRead (GetCommOffset (NORTHWEST, 0x00D0) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 1) | \
(((GpioPadRead (GetCommOffset (NORTHWEST, 0x00C8) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 2) | \
(((GpioPadRead (GetCommOffset (NORTH, 0x01E0) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 3)));

DEBUG ((DEBUG_INFO, "BoardId from PMIC strap: %02X\n", *BoardId));

//
// Fab_ID0: PMIC_I2C_SDA
//
CommAndOffset = GetCommOffset (NORTHWEST, 0x0108);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig0.r.PMode = 0;
padConfig0.r.GPIORxTxDis = 0x1;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);
//
// Set to Pull Up 20K

```

```

//
padConfig1.r.Term = 0xC;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
padConfig1.padCnf1);
//
// Fab_ID1: PMIC_I2C_SCL
//
CommAndOffset = GetCommOffset (NORTHWEST, 0x0100);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig0.r.PMode = 0;
padConfig0.r.GPIORxTxDis = 0x1;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);
//Set to Pull Up 20K
padConfig1.r.Term = 0xC;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
padConfig1.padCnf1);
//
// Fab_ID2: PMIC_BCUDISW2
//
CommAndOffset = GetCommOffset (NORTHWEST, 0x00D8);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig0.r.PMode = 0;
padConfig0.r.GPIORxTxDis = 0x1;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);
//
// Set to Pull Up 20K
//
padConfig1.r.Term = 0xC;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
padConfig1.padCnf1);
//
// Fab_ID3: PMIC_BCUDISCRIT
//
CommAndOffset = GetCommOffset (NORTHWEST, 0x00E0);
padConfig0.padCnf0 = GpioPadRead (CommAndOffset +
BXT_GPIO_PAD_CONF0_OFFSET);
padConfig0.r.PMode = 0;
padConfig0.r.GPIORxTxDis = 0x1;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF0_OFFSET,
padConfig0.padCnf0);
//
// Set to Pull Up 20K
//
padConfig1.r.Term = 0xC;
GpioPadWrite (CommAndOffset + BXT_GPIO_PAD_CONF1_OFFSET,
padConfig1.padCnf1);

*FabId = (UINT8) (((GpioPadRead (GetCommOffset (NORTHWEST, 0x0108) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) | \
                (((GpioPadRead (GetCommOffset (NORTHWEST, 0x0100) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 1) | \
                (((GpioPadRead (GetCommOffset (NORTHWEST, 0x00D8) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 2) | \

```

```

        (((GpioPadRead (GetCommOffset (NORTHWEST, 0x00E0) +
BXT_GPIO_PAD_CONF0_OFFSET) & BIT1) >> 1) << 3));

    DEBUG ((EFI_D_INFO, "FabId from PMIC strap: %02X\n", *FabId));

    return EFI_SUCCESS;
}

```

## Save Board ID and FAB ID into PCD

**Platform/BroxtonPlatformPkg/Board/NewBoard/BoardInitPreMem/BoardInit.c**

The PCD values for **PcdBoardId** and **PcdFabId** are set in PEI pre-memory initialization by the function **NewBoardPreMemInit**. Subsequent code can read the **BoardId** and **FabId** PCD values.

```

PcdSet8 (PcdBoardId, BoardId);
PcdSet8 (PcdFabId, FabId);

```

## Change Serial Port for Debug Messages

Platforms often contain multiple serial ports, which can be used by the UEFI IA Firmware to output serial debug message or create a serial console. Set the PCD **PcdSerialIoUartNumber** to select the UART controller.

**edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.dec**

```

gPlatformModuleTokenSpaceGuid.PcdSerialIoUartNumber | 2 | UINT8 | 0x10001011

```

## Change System Memory Parameters

There are two supported methods for changing system memory parameters.

- Use the [Intel® Binary Configuration Tool](#) (Intel® BCT) to change default settings defined for Intel® FSP (.BSF file). This is the recommended method for setting default platform parameters for common silicon initialization. Please refer to Section 4.1 for details.
- Override specific Intel FSP settings in the default Intel FSP configuration. This method is recommended for overriding parameters based on board-specific configuration. Overridden platform values are defined via UPD (**FspUpdRgn**). Examples can be found in the following file: **edk2-platforms\Platform\BroxtonPlatformPkg\Board\LeafHill\BoardInitPreMem\BoardInitMiscs.c**

The example below assigns board-specific memory settings for a new board:

```

FspUpdRgn->FspmConfig.Package           = 0x01;
FspUpdRgn->FspmConfig.Profile           = 0x0B;
FspUpdRgn->FspmConfig.MemoryDown       = 0x01;
FspUpdRgn->FspmConfig.DualRankSupportEnable = 0x01;
FspUpdRgn->FspmConfig.Ch0_RankEnable    = 0x03;
FspUpdRgn->FspmConfig.Ch0_DeviceWidth   = 0x01; // x16
FspUpdRgn->FspmConfig.Ch0_DramDensity   = 0x02; // 8Gb

```

These methods are discussed further in the “Intel® FSP Configuration” section of this document. Please refer to the “Board Level Configuration” section of the “Open Source UEFI Firmware Enabling Guide: Intel Atom® Processor E3900 Series Platforms” for additional information:

[https://firmware.intel.com/sites/default/files/uefi\\_firmware\\_enabling\\_guide\\_for\\_the\\_intel\\_atom\\_processor\\_e3900\\_series.pdf](https://firmware.intel.com/sites/default/files/uefi_firmware_enabling_guide_for_the_intel_atom_processor_e3900_series.pdf)

## Change Display Device Initialization

Display device configuration on Intel platforms is set in a Video BIOS Table (VBT) configured by the Intel® Binary Modification Program (Intel® BMP). Each board in this firmware project uses a default VBT (.bin), which may need to be altered for different display configurations (port type, initial resolution, LVDS parameters, etc.).

It is recommended to create a new VBT instead of overwriting the VBT included with a reference project. The new file can be stored under the board directory:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard\VBT
```

This ensures the modified VBT is included in the build, instead of the original video configuration. Create a GUID for the new VBT file and add it to the project:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.dec
```

```
gPeiLeafHillVbtGuid      = { 0x6ae80680, 0x5e3f, 0x4e63, { 0xa5, 0xf5, 0x78,
0xe5, 0x21, 0x4f, 0x13, 0xfe } }
gPeiNewBoardVbtGuid     = { 0xE08CA6D5, 0x8D02, 0x43ae, { 0xAB, 0xB1,
0x95, 0x2C, 0xC7, 0x87, 0xC9, 0x33 } }
gPeiMinnowBoard3VbtGuid = { 0xE08CA6D5, 0x8D02, 0x43ae, { 0xAB, 0xB1, 0x95,
0x2C, 0xC7, 0x87, 0xC9, 0x33 } }
```

Next, add the modified VBT path and GUID to the FDF file:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.fdf
!if $(PEI_DISPLAY_ENABLE) == TRUE
  FILE FREEFORM = 7BB28B99-61BB-11D5-9A5D-0090273FC14D {
    SECTION RAW = $(PLATFORM_PACKAGE_COMMON)/Binaries/Logo/Logo.bmp
  }

  # VBT For Leaf Hill (File Guid is gPeiLeafHillVbtGuid)
  FILE FREEFORM = 6AE80680-5E3F-4E63-A5F5-78E5214F13FE {
    SECTION RAW = $(PLATFORM_NAME)/Board/LeafHill/Vbt/VbtBxtMipi.bin
    SECTION UI = "IntelGopVbt1"
  }

  # VBT For New Board (File Guid is gPeiNewBoardVbtGuid)
  FILE FREEFORM = E08CA6D5-8D02-43ae-ABB1-952CC787C933 {
    SECTION RAW = $(PLATFORM_NAME)/Board/NewBoard/Vbt/VbtBxtMipi.bin
    SECTION UI = "IntelGopVbt1"
  }
}
```

Finally, reference the VBT GUID in the BoardInit.c file.

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard\
BoardInitPostMem\BoardInit.c
//
// Board specific VBT table.
```



```
//  
BufferSize = sizeof (EFI_GUID);  
PcdSetPtr(PcdBoardVbtFileGuid, &BufferSize, (UINT8 *)&  
gPeiNewBoardVbtGuid);
```

## Modify GPIO Configuration

---

General Purpose Input/Output (GPIO) pin configuration is controlled by tables in the file `BoardGpios.h`. Example:

```
Platform\BroxtonPlatformPkg\Board\NewBoard\BoardInitPostMem\  
BoardGpios.h
```

Edit the copy of this file in the `NewBoard` folder to change GPIO behavior. Please review the following Intel documents before modifying the GPIO configuration:

- Intel® Pentium® and Celeron® Processor N- and J- Series Datasheet (Vol 1-3)
  - <https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-1.html>
  - <https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-2.html>
  - <https://www.intel.com/content/www/us/en/processors/pentium/pentium-celeron-n-series-j-series-datasheet-vol-3.html>

## Add & Remove Peripherals in ACPI SSDT

---

Modern operating systems use the Advanced Configuration & Power Interface (ACPI) specification for accessing hardware at OS runtime. Certain classes of device are not enumerated by the firmware and rely on data in an ACPI Secondary System Device Table (SSDT) for proper configuration. This includes devices like cameras, audio codecs, and touch panels that are attached to a non-enumerable bus (I2C, SPI, etc.).

This project includes numerous examples for peripheral devices, defines in ACPI Source Language (ASL). Note this document does not cover the creation or syntax of ASL programming.

```
Platform\BroxtonPlatformPkg\Common\Acpi\AcpiTablesPCAT\PlatformSsdT
```

The file `PlatformSsdT.asl` can be edited to determine which device definitions are included in the firmware at build time. New devices can be defined by creating a new `.ASL` file for the device and adding it to `PlatformSsdT.asl`.

## Add New Peripherals

---

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Acpi\  
AcpiTablesPCAT\PlatformSsdT
```

Create a new folder under `PlatformSsdT` named `NBAcpi`. This will contain any `.ASL` files designated for the `NewBoard` project.

Create a new file (`NBAcpi.asl`) in the `NBAcpi` folder.

Include `NBAcpi.asl` in `PlatformSsdT.asl`

```
include ("NBacpi/NBacpi.asl")
```

## Remove Peripherals

```
edk2-platforms\Silicon\BroxtonSoC\BroxtonSiPkg\NorthCluster\Include\Protocol\
GlobalNvsArea.h
```

Add new member `NBacpiTableEnable` in structure `EFI_GLOBAL_NVS_AREA` of Miscellaneous Dynamic Values, using definitions below that match `GNVS` definitions in `Platform.ASL`

```
UINT8      NBacpiTableEnable;    ///< (925) NewBoard Acpi Table Enable:
                                   ///< 0:Disable; 1:Enable
```

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Acpi\AcpiTablesPCAT\
GloblNvs.asl
```

Add new member `NBEN` in `GNVS`.

```
NBEN,      8,      // (925) NewBoard Acpi Table Enable: 0:Disable; 1:Enable
```

Add `NBEN` to control `NBacpi.asl` in `PlatformSsdt.ASL`

```
if (LEqual(NBEN, 1)) {
    include ("NBacpi/NBacpi.asl")
}
```

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.dec
```

Add new PCD `PcdNBacpiEn` in to control the ACPI table.

```
## This PCD used to include NBacpi Table
gPlatformModuleTokenSpaceGuid.PcdNBacpiEn | 1 | UINT8 | 0x80001000
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard\
BoardInitPostMem\BoardInit.c
```

PCD `PcdNBacpiEn` controls ACPI table function in `NewBoardPostMemInitCallback`.

```
//
// Set PcdNBacpiEn
//
PcdSetBool (PcdNBacpiEn, FALSE);
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Acpi\AcpiPlatformDxe\
AcpiPlatform.c
```

Set `NBacpiTableEnable` from `PcdNBacpiEn` in `AcpiPlatformEntryPoint`.

```
//
// NBacpi Table is Enabled by default
//
mGlobalNvsArea.Area->NBacpiTableEnable = PcdGet8(PcdNBacpiEn);
```

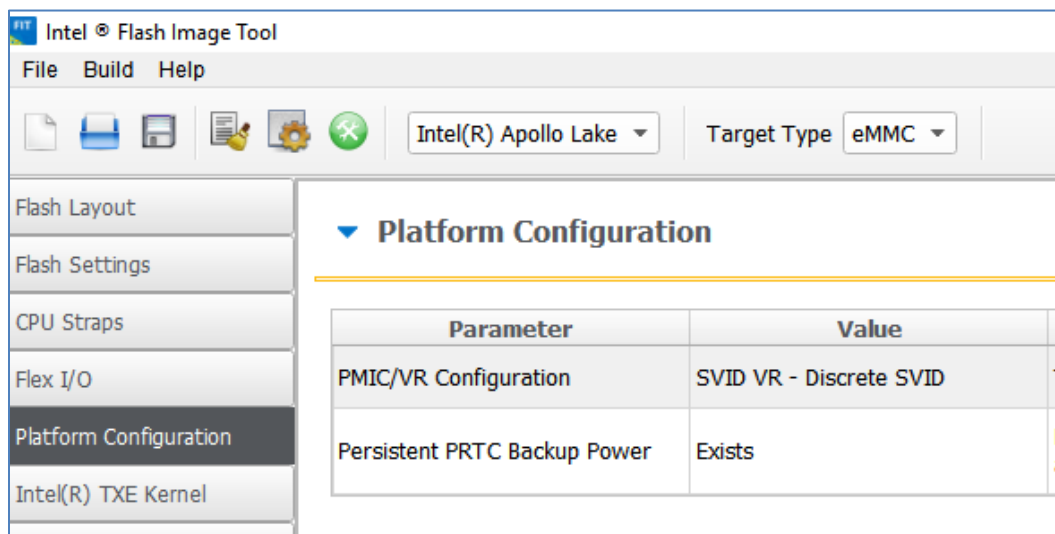
## Configuration via Intel® Flash Image Tool

Section describes the process for building a platform firmware image. Some aspects of this process are controlled by the Intel® Flash Image Tool (Intel® FIT). Please contact an Intel representative for information on how to acquire this toolkit.

### Change PMIC/VR Configuration

Select a pre-defined PMIC/VR Configuration with Intel® FIT tool as shown in Figure 2.

**Figure 2: PMIC/VR Configuration with FIT Tool**



### Modify I/O Configuration (PCIe & USB)

Use Intel® FIT to modify the configuration of USB ports and PCIe lanes as shown in Figure 3. Note that some PCIe lanes are multiplexed with USB ports.



## Other Platform Customizations

### Intel® FSP Configuration

The Intel® Firmware Support Package (Intel® FSP) provides chipset and processor initialization. This firmware project uses two methods to configure Intel FSP parameters:

- [Intel® Binary Configuration Tool](https://github.com/IntelFsp/BCT) (Intel® BCT), which sets the default platform parameters for silicon initialization. <https://github.com/IntelFsp/BCT>
- UPD structure in pre-memory init, which can override specific Intel FSP settings (`FspUpdRgn`) based on board-specific configuration.

### Using Intel BCT to Configure Intel FSP Parameters

The Intel FSP configuration file (.bsf) can be modified using Intel BCT.

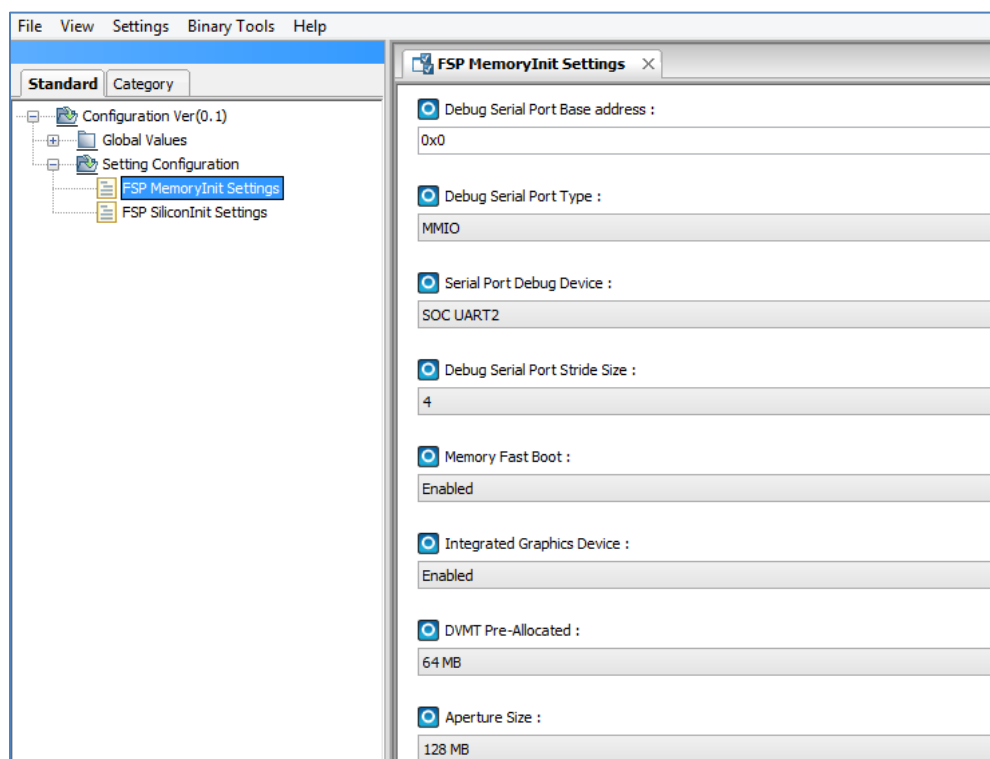
```
edk2-platforms\Silicon\BroxtonSoC\BroxtonFspPkg\  
ApolloLakeFspBinPkg\FspBin
```

Please refer to the Intel BCT User Guide for more information.

### Default MemoryInit Settings in Intel BCT

The following figure shows an example using the `MemoryInit` setting with Intel BCT.

**Figure 5: FSP MemoryInit Settings Using Intel® FIT**



## MemoryInit Settings Commonly Changed for a New Board

---

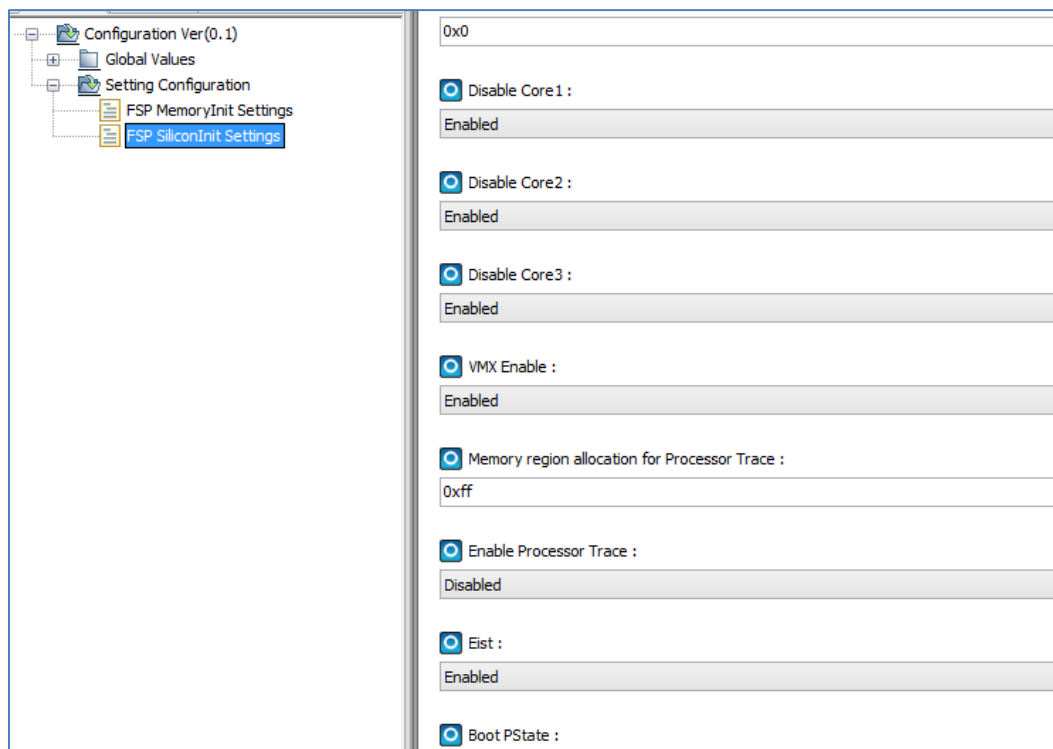
- Serial Port Debug Device
- Package
- Profile
- MemoryDown
- Ch0\_DramDensity
- Ch1\_DramDensity
- Ch0\_RankEnable
- Ch1\_RankEnable

## Default SiliconInit Settings in Intel BCT

---

The following figure gives an example using SiliconInit settings with Intel BCT.

**Figure 6: SiliconInit Settings using Intel BCT**



## Using UPD to Override Intel FSP Parameters

---

Override via UPD (`FspUpdRgn`) is recommended for overriding parameters based on a board-specific configuration. Examples can be found in the Leaf Hill CRB:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\LeafHill\
BoardInitPreMem\BoardInitMiscs.c
```

UPD structures and related definitions are defined by Intel FSP:

```
edk2-platforms\Silicon\BroxtonSoC\BroxtonFspPkg\Include
```

This method requires more detailed knowledge of platform parameters. Please refer to the “Apollo Lake Intel® Firmware Support Package (FSP) Integration Guide” for more information:

<https://github.com/IntelFsp/FSP/tree/ApolloLake/ApolloLakeFspBinPkg/Docs>

## Change Default Value for User Setup Options

---

The default value of user setup options can be modified to best fit the platform configuration. The initial value is controlled by the following flags:

```
flags = DEFAULT | MANUFACTURING | RESET_REQUIRED
```

The following example changes the “Discrete TPM” setting from “Disabled” to “Enabled” by modifying the flags associated with each option.

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\PlatformSettings\
PlatformSetupDxe\Security.vfi
```

```
oneof  varid  = Setup.TPM,
      prompt  = STRING_TOKEN(STR_TPM_PROMPT),
      help    = STRING_TOKEN(STR_TPM_HELP),
      option text = STRING_TOKEN(STR_DISABLE), value = 0x00, flags = DEFAULT |
MANUFACTURING | RESET_REQUIRED;
      option text = STRING_TOKEN(STR_TPM_PTT), value = 0x01, flags =
RESET_REQUIRED;
endoneof;
```

It is recommended to set both the `DEFAULT` and `MANUFACTURING` flags for any value that should be the default value during normal operation. Please set the `RESET_REQUIRED` flag for any value that requires a system reset to take effect.

For more information on defining user setup values, please refer to the “EDK II VFR Programming Language Specification”.

<https://edk2-docs.gitbooks.io/edk-ii-vfr-specification/content/>

## Custom Boot Logo

---

A small boot logo can be displayed in the center of the screen, prior to the OS loading. Images are in BMP format (24 bit, maximum resolution 208 x 131). Due to module space limit, the BMP size cannot exceed 79K. A default logo is included that is common to all boards in the firmware project:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Binaries\Logo\Logo.bmp
```

Rather than overwriting the existing logo, it is recommended to create a new logo that is stored under the board folder:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard\Logo
```

Create a GUID for the new logo file and add it to the project:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.dec
gPeiMinnowBoard3LogoGuid = { 0xE1F5F8B0, 0x1707, 0x4A2D, { 0xB6, 0x07, 0x1A,
0x2B, 0xE4, 0x54, 0x60, 0x0B } }
gPeiNewBoardLogoGuid = { 0x02c2a0ef, 0x98ba, 0x417a, { 0x96, 0xb1, 0x57,
0x2f, 0x80, 0x16, 0x4c, 0x5c } }
```

Next, add a reference to the logo to the project FDF file:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.fdf
FILE FREEFORM =
PCD(gEfiIntelFrameworkModulePkgTokenSpaceGuid.gPeiNewBoardLogoGuid) {
    SECTION RAW = $(PLATFORM_NAME)/Board/MinnowBoard3/Logo/Logo.bmp
}
FILE FREEFORM =
PCD(gEfiIntelFrameworkModulePkgTokenSpaceGuid.gPeiNewBoardLogoGuid) {
    SECTION RAW = $(PLATFORM_NAME)/Board/NewBoard/Logo/Logo.bmp
}
```

Finally, reference the VBT GUID in the `BoardInit.c` file.

```
edk2-platforms\Platform\BroxtonPlatformPkg\Board\NewBoard\
BoardInitPostMem\BoardInit.c
//
// Board specific Logo
//
BufferSize = sizeof (EFI_GUID);
PcdSetPtr(PcdOemLogoFileGuid, &BufferSize, (UINT8 *)& gPeiNewBoardLogoGuid);
```

## Default Boot Order

---

Boot order controls the priority of platform boot devices (USB, SATA, Network, etc.). Users can configure boot order with a setup option. The default boot order can also be controlled using the `BootOptionPriority()` function:

```
edk2-platforms\Platform\BroxtonPlatformPkg\Common\Library\
PlatformBootManagerLib\PlatformBootOption.c
```

```
UINTN
BootOptionPriority (
    CONST EFI_BOOT_MANAGER_LOAD_OPTION *BootOption
)
{
    //
```



```

// EFI boot options.
//
switch (BootOptionType (BootOption->FilePath)) {
    case MSG_MAC_ADDR_DP:
    case MSG_VLAN_DP:
    case MSG_IPv4_DP:
    case MSG_IPv6_DP:
        return 5;
    case MSG_SATA_DP:
    case MSG_ATAPI_DP:
        return 2;
    case MSG_USB_DP:
        return 1;
}

if (StrCmp (BootOption->Description, INTERNAL_UEFI_SHELL_NAME) == 0) {
    return 6;
}
if (StrCmp (BootOption->Description, UEFI_HARD_DRIVE_NAME) == 0) {
    return 3;
}
if (StrCmp (BootOption->Description, UEFI_NVME_DRIVE_NAME) == 0) {
    return 4;
}

return 100;
}

```

## Remove UEFI Shell

---

The UEFI Shell is an interactive text interface, similar to DOS and Linux shell environments, for accessing a pre-OS environment.

<https://github.com/tianocore/tianocore.github.io/wiki/ShellPkg>

A pre-built binary version of the UEFI Shell is integrated in the firmware project, primarily designed for pre-release debugging. Removing the UEFI Shell from production platform firmware is recommended, which is easily done by editing the project `.FDF` file.

`edk2-platforms\Platform\BroxtonPlatformPkg\PlatformPkg.fdf`

```

#
# UEFI Shell
#
FILE APPLICATION = 7C04A583-9E3E-4f1c-AD65-E05268D0B4D1 {
SECTION PE32 = ShellBinPkg/UefiShell/$(IA32_X64_IC)/Shell.efi
}

```

## SMBIOS Table

---

[System Management BIOS \(SMBIOS\)](#) defines a standard for delivering management information via system firmware. SMBIOS values are commonly used by the operating system and may need to be customized based on platform changes. These values are set by the firmware during the boot process. Table 1 provides the SMBIOS types used for the platform.

`edk2-platforms\Platform\BroxtonPlatformPkg\Common\Features\Smbios\SmBiosMiscDxe`

Each SMBIOS data type relies on three source files:

- `*.uni` – defines Unicode strings used in the entry
- `*Data.c` – defines the structure of the entry
- `*Function.c` – defines functions used to dynamically update the entry

Table X lists common SMBIOS types that require customization when porting from a reference hardware design to a new platform. In most cases, only string changes (`*.uni`) are required. Note that the table is not a comprehensive list of SMBIOS Types, since many SMBIOS entries are dynamically generated and may not require platform-specific porting changes.

**Table 1: SMBIOS Structure Table for Platform**

Structure Name & Type	Related Source Files
BIOS Information (Type 0)	MiscBiosVendor.uni MiscBiosVendorData.c MiscBiosVendorFunction.c
System Information (Type 1)	MiscSystemManufacturer.uni MiscSystemManufacturerData.c MiscSystemManufacturerFunction.c
Baseboard/Module Information (Type 2)	MiscBaseBoardManufacturer.uni MiscBaseBoardManufacturerData.c MiscBaseBoardManufacturerFunction.c
System Enclosure (Type 3)	MiscChassisManufacturer.uni MiscChassisManufacturerData.c MiscChassisManufacturerFunction.c
Port Connector Information (Type 8)	MiscPortInternalConnectorDesignator.uni MiscPortInternalConnectorDesignatorData.c MiscPortInternalConnectorDesignatorFunction.c
System Slots (Type 9)	MiscSystemSlotDesignation.uni MiscSystemSlotDesignationData.c MiscSystemSlotDesignationFunction.c
OEM Strings (Type 11)	MiscOemString.uni MiscOemStringData.c MiscOemStringFunction.c
System Configuration Options (Type 12)	MiscSystemOptionString.uni MiscSystemOptionStringData.c MiscSystemOptionStringFunction.c
BIOS Language Information (Type 13)	MiscNumberOfInstallableLanguagesData.c MiscNumberOfInstallableLanguagesFunction.c  MiscSystemLanguageString.uni MiscSystemLanguageStringData.c MiscSystemLanguageStringFunction.c
System Reset (Type 23)	MiscResetCapabilitiesData.c MiscResetCapabilitiesFunction.c
System Boot Information (Type 32)	MiscBootInformationData.c MiscBootInformationFunction.c

For more information, please refer to Section 6.2 (“Required structures and data”) of the SMBIOS Specification, available from dmtf.org:

[https://www.dmtf.org/sites/default/files/standards/documents/DSP0134\\_3.2.0.pdf](https://www.dmtf.org/sites/default/files/standards/documents/DSP0134_3.2.0.pdf)

## Signed Capsule Update

---

[EDK II](#) provides an implementation of capsule-based firmware update and firmware recovery features that can detect if a firmware update or a recovery image delivered via UEFI Capsule has been modified ([SignedCapsulePkg](#)). It can also verify that the capsule applies to the platform that receives the capsule and verifies that a firmware update does not violate any of the platform's firmware rollback rules.

<https://github.com/tianocore/edk2/tree/master/SignedCapsulePkg>

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Defines.dsc
    DEFINE CAPSULE_GENERAL_ENABLE = TRUE
```

The feature can be disabled by set

```
    DEFINE CAPSULE_GENERAL_ENABLE = FALSE
```

For more information, review the documentation available at the TianoCore wiki:

<https://github.com/tianocore/tianocore.github.io/wiki/Capsule-Based-Firmware-Update-and-Firmware-Recovery>

## Enable Verified Boot

---

Enabling Verified Boot includes the process of manifesting, signing and assembling firmware components. System firmware is a key element in establishing platform root-of-trust, since it initializes the platform after reset and launches the operating system. Verifying system firmware components prior to execution is one element of a more secure platform implementation.

Root-of-trust components for Verified Boot are the Intel Trusted Execution Engine (Intel TXE) ROM and OEM key. 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 programmed at the end of manufacturing stage by the OEM.

Verified Boot relies on the Intel Manifest Extension Utility (MEU) to sign and manifest firmware binaries. The manifest tool leverages OpenSSL to sign a file. Intel MEU is part of the Intel® TXE Firmware (FW) Production Version (PV) release package (kit number 120568), available from <https://platformsw.intel.com>. Customers can contact an Intel representative for information on downloading the release package.

Verified Boot can be enabled in the firmware project using `BOOT_GUARD_ENABLE`. The token is disabled by default:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Defines.dsc
```

```
    DEFINE BOOT_GUARD_ENABLE = FALSE
```

The feature can be enabled by set macro as TURE

```
    DEFINE BOOT_GUARD_ENABLE = TRUE
```

Firmware boot flow is altered when using Verified Boot. Please refer to the "Enabling Verified Boot" section of the "Open Source UEFI Firmware Enabling Guide: Intel Atom® Processor E3900 Series Platforms" for more information:

[https://firmware.intel.com/sites/default/files/uefi\\_firmware\\_enabling\\_guide\\_for\\_the\\_intel\\_atom\\_processor\\_e3900\\_series.pdf](https://firmware.intel.com/sites/default/files/uefi_firmware_enabling_guide_for_the_intel_atom_processor_e3900_series.pdf)

## Trusted Platform Module (TPM)

---

The TPM is used to securely store artifacts for platform authentication. Platform firmware initializes the TPM during boot for use by the OS for trusted computing applications. The firmware project contains definitions to enable support for a discrete TPM (3<sup>rd</sup> party hardware) or firmware TPM (fTPM, integrated in the SoC).

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Defines.dsc
    DEFINE FTPM_ENABLE = TRUE
    DEFINE TPM12_ENABLE = TRUE
```

The feature can be disabled by set macro as FALSE

```
    DEFINE FTPM_ENABLE = FALSE
    DEFINE TPM12_ENABLE = FALSE
```

For more information on the TPM, please refer to the following documents:

- <https://trustedcomputinggroup.org/resource/trusted-platform-module-2-0-a-brief-introduction/>
- <https://docs.microsoft.com/en-us/windows/security/hardware-protection/tpm/trusted-platform-module-overview>

## UEFI Secure Boot

---

UEFI Secure Boot defines how a platform's firmware can authenticate a digitally signed image, such as an operating system loader or UEFI driver stored in an Option ROM. This provides the capability to ensure that those UEFI images are only loaded in an owner authorized fashion and provides a common means to ensure platforms security and integrity over systems running UEFI-based firmware.

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Defines.dsc
    DEFINE SECURE_BOOT_ENABLE = TRUE
```

The feature can be disabled by setting the macro as FALSE.

```
    DEFINE SECURE_BOOT_ENABLE = FALSE
```

For more information on UEFI Secure Boot, please refer to the following documents:

- [Signing UEFI Applications and Drivers for UEFI Secure Boot](#) (whitepaper)
- [A Tour Beyond BIOS into UEFI Secure Boot](#) (whitepaper)
- <https://github.com/tianocore/tianocore.github.io/wiki/SecurityPkg>
- <https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot>

## UEFI Networking

---

EDK II enables a full network stack, as defined by the UEFI Specification. The open source `NetworkPkg` package supports features such as PXE boot, iSCSI, VLAN, and HTTP/HTTPS boot over IPv4 & IPv6 networks.

The firmware project includes `NetworkPkg` by default, and defines tokens to enable or disable specific network features:

```
edk2-platforms\Platform\BroxtonPlatformPkg\PlatformDsc\Defines.dsc
    DEFINE NETWORK_ENABLE = TRUE
```

```
DEFINE NETWORK_IP6_ENABLE = TRUE
DEFINE NETWORK_ISCSI_ENABLE = FALSE
DEFINE NETWORK_VLAN_ENABLE = FALSE
```

For more information, please refer to the following documentation:

- [Getting Started Guide of EDK II HTTP Boot](#) (TianoCore whitepaper)
  - [Getting Started with UEFI HTTPS Boot on EDK II](#) (TianoCore whitepaper)
  - [UEFI PXE Boot Performance Analysis](#) (Intel whitepaper)
  - <https://github.com/tianocore/tianocore.github.io/wiki/Enable-UEFI-PXE-Boot-in-EDK-II>
  - <https://github.com/tianocore/tianocore.github.io/wiki/NetworkPkg>
  - <https://github.com/tianocore/tianocore.github.io/wiki/NetworkPkg-Getting-Started-Guide>
- 



© 2018, Intel Corporation.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Currently characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725, or online at [www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm).

Intel, the Intel logo, [List the Intel trademarks in your document] are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

## Revision History

---

<b>Date</b>	<b>Description</b>
September 2018	Initial document release.
September 20, 2018	Fix coding typo in Custom Boot Logo section.