White Paper

A Tour Beyond BIOS Implementing the Tiny Quark Design

Jiewen Yao
Intel Corporation

Vincent J. Zimmer
Intel Corporation

Elvin Li
Intel Corporation

Chris Li
Intel Corporation

September 2014

"Don't be encumbered by history. Go off and do something wonderful." Robert Noyce [BIO]

Executive Summary

This Intel implementation of EDKII [EDK2] demonstrates the possibilities available using the scalable architecture of both the code base and the associated underlying industry standards [UEFI][UEFI PI Specification]. The UEFI firmware size for this Intel® Galileo EDKII implementation [QUARK] is 64KiB, and given its diminutive size relative to the full Quark EDKII build, it is referred to as "TinyQuark" throughout the rest of this document. TinyQuark boots Yocto Linux [YOCTO] on the Intel® Galileo [GALILEO] board using the on board flash. You can build this solution from the source code available to download using the link below.

Specifically, the TinyQuark code is at http://uefidk.intel.com/content/get-started-intel-galileo-development-board.

This paper presents the internal design of TinyQuark which can be generalized by developers to make their own small-footprint UEFI BIOS.

Goal
The feature set in TinyQuark is different with traditional one. The goal of TinyQuark is to show what is possible.

Goal is...	Goal is NOT...
Follow UEFI spec.	Follow PI spec.
Reduce feature.	All features.
Quark specific.	General for all platform.
Boot Yocto only.	Boot all OS.
Full chipset initialization.	Full chipset feature.
Full platform initialization.	Full platform feature.

Prerequisite
This paper assumes that the audience has EDKII/UEFI firmware development experience on Quark. Also, the content assumes familiarity with UEFI/PI firmware infrastructure, such as SEC, PEI, DXE, and the runtime phase, on Quark.

Table of Contents

Overview

Introduction to Quark

The Intel® Quark System-on-a-chip (SoC) X1000 is the first product in a new roadmap of innovative, small core products targeted at rapidly growing areas ranging from the industrial IoT [IOT] to wearables. It will bring low power and Intel compute capabilities for thermally-constrained, fan-less, and head-less applications. With its security and manageability features, this SoC is ideally suited for the Internet of Things (IoT) and for the next wave of cost-effective intelligent connected devices.

Introduction to EDKII

EDKII is open source implementation for UEFI firmware, which can boot multiple UEFI-aware operating systems. Section 2.6 of the UEFI Specification [UEFI] defines the minimum set of capabilities that UEFI-aware firmware, such as EDKII, must support.. We use EDKII BIOS for the Galileo board, which uses Quark processor.

Summary

This section provided an overview of Quark and EDKII.

ROM Flash size optimization

Technology Summary

A set of UEFI/EDKII related size reduction technologies to make TinyQuark are listed below. "Write Good C-Code" is NOT listed below, because it is a generic advocacy to have robust, testable code [SECURE]. Instead, this section defines how to apply Occam's Razor to the EDKII Quark firmware and still maintain basic UEFI conformance.

In next several sections, we will discuss the size reduction techniques one by one.

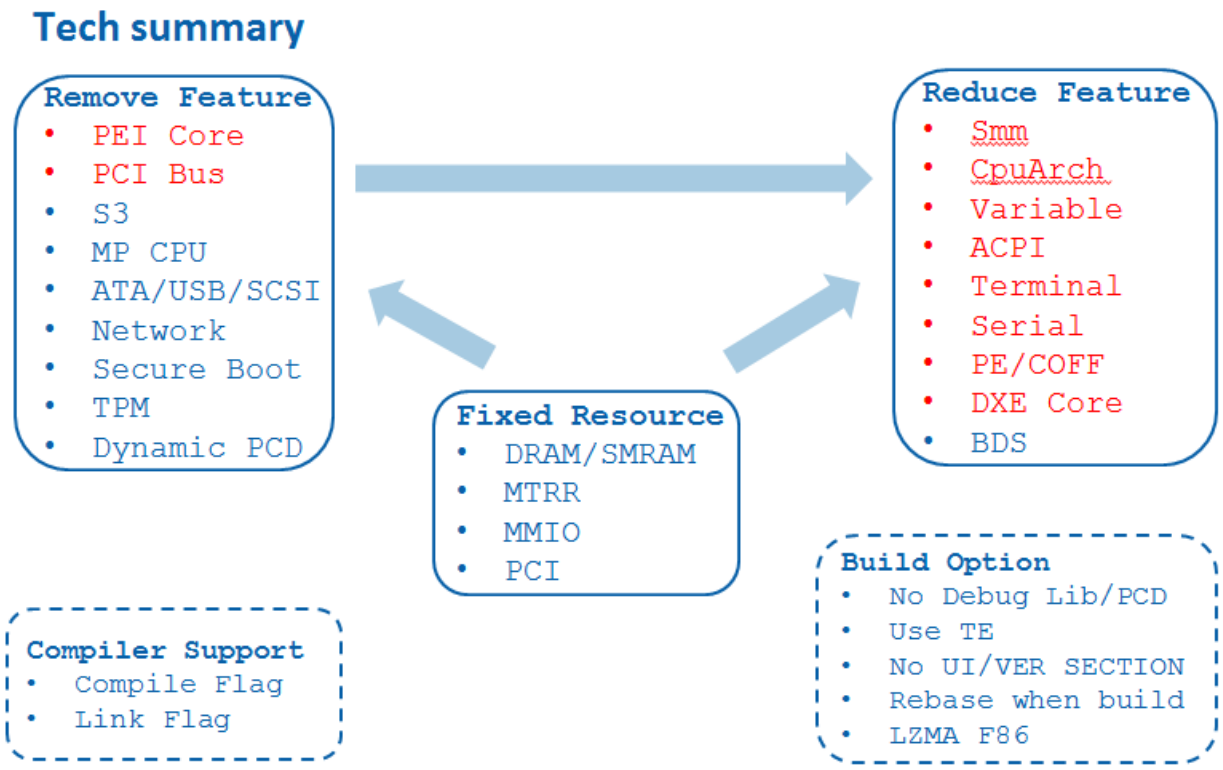

Figure 1 Technology Summary

Fixed Resource

To begin, the support of fixed system resources is important because it will lead to the direct on feature set of the platform.

1) DRAM/SMRAM

If a platform can have fixed DRAM resource, then we can remove complicated Memory-Type Range Register (MTRR) [SDM] calculation algorithm in the CPU driver. The MTRR settings can be a table driven configuration for this design with a known physical memory map.

Refer to: QuarkPlatformPkg\Library\QuarkSecLib\SecPlatform.c [QUARK]

If a platform can have fixed SMRAM, then we can remove the SmmAccess driver that supports the PI SMM interfaces [UEFI PI Specification], and just use library to get that value.
Refer to: QuarkPlatformPkg\Library\SmmPlatformHookLib

If a platform can have fixed memory mapped I/O (MMIO) and PCI resources, then we can remove PCI driver. The PCI resource setting can be done by a table driven configuration.
Refer to: QuarkPlatformPkg\Pci\PlatformFixedPciResource

Remove features

Not all UEFI features are needed in TinyQuark. Some features can be removed directly, like S3 [ACPI], ATA bus [ATA], USB bus [USB], SCSI bus, network [UEF], HII [UEFI], UEFI secure boot [UEFI], TPM [TCG], or dynamic PCD [UEFI PI Specification].

Removing some features need fixed resource support, like PCI bus driver.

The biggest component removed in TinyQuark is the PEI Core, though. TinyQuark has SEC linked DecompressLib, and the code in SEC jumps directly into DxeIpl. DxeIpl links into the memory referece code (MRC), finishes memory initialization, and then jumps into the DxeCore. The DXE Core provides the basic UEFI capabilities, such as the boot services.

The detail flow of SEC->DXE is shown below:

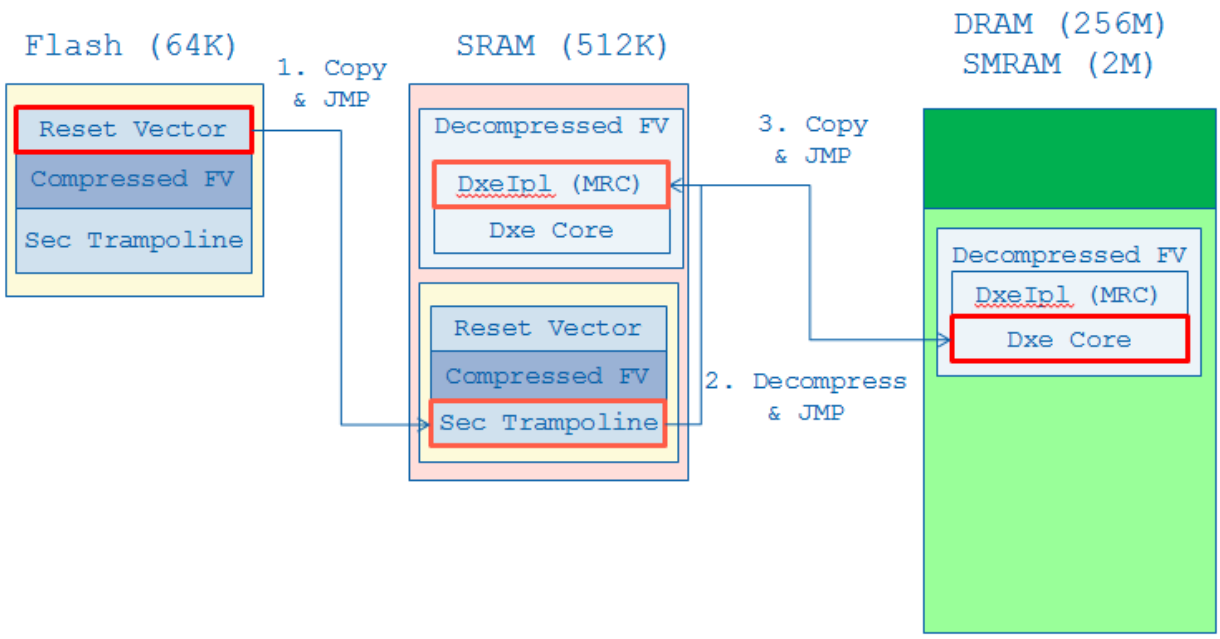

Figure 2 SEC->DXE solution

Please refer to QuarkPlatformPkg.dsc and QuarkPlatformPkg.fdf for more detail on how many features are removed.

For ResetVector module, refer to QuarkPlatformPkg/Cpu/Sec/ResetVector
For SecTrampoline module, refer to QuarkPlatformPkg/Cpu/SecTrampoline
For DxeIpl module, refer to QuarkPlatformPkg/Cpu/SecCore

Reduce features

Even if a component is still needed in TinyQuark, we can make a simplified version. For example:

1) CpuArch. No SetMemoryAttribut() support, because MTRR is fixed and programmed in DxeIpl.

Refer to IA32FamilyCpuBasePkg/SimpleCpuArchDxe

2) Variable. Just expose empty variable driver. Or later we can have some fixed data integrated in this driver. So there is no need to allocate specific Variable FV region.

Refer to
TinyBootPkg/Universal/Variable/NullVariableRuntimeDxe/NullVariableRuntimeDxe.inf

3) ACPI. No generic ACPI_TABLE or ACPI_SDT driver. We created AcpiLib to support SetAcpi() only in AcpiPlatform driver.

Refer to TinyBootPkg/Library/AcpiTableLib

4) Terminal. No driver model. Only support PcAnsi.

Refer to TinyBootPkg/Universal/Console/SimpleCombinedTerminalDxe

5) Serial. No driver model. Link SerialPortLib directly.

Refer to TinyBootPkg/Universal/Console/SimpleCombinedTerminalDxe

6) PE/COFF lib. Support PE32 only, no PE32+ or TE.

Refer to TinyBootPkg/Library/BasePeCoffLibPe32

7) DXE Core: No GUIDED_SECTION, no Decompression, no FVB, no EBC, no HII, no DebugInfo table.

Refer to TinyBootPkg/Core/SimpleDxeCore

8) SMM: SMM is redesigned. We removed SmmCore. See below for detail:

Current EDKII has SmmIpl, SmmCore, and SmmCpu driver. SmmIpl will load SmmCore into SMRAM. SmmCore will load all SMM drivers into SMRAM, including SmmCpu driver, SmmPch driver, SmmPlatform driver, etc.
After the SmmCpu driver is loaded, SMBASE rebase happens. Then the next SMI will trigger into the SmmCpu driver. Then the SmmCpu driver passes control to the SmmCore, and the SmmCore calls each Smm root handler driver, as registered by SmmPch driver, to dispatch the respective SMI handler.

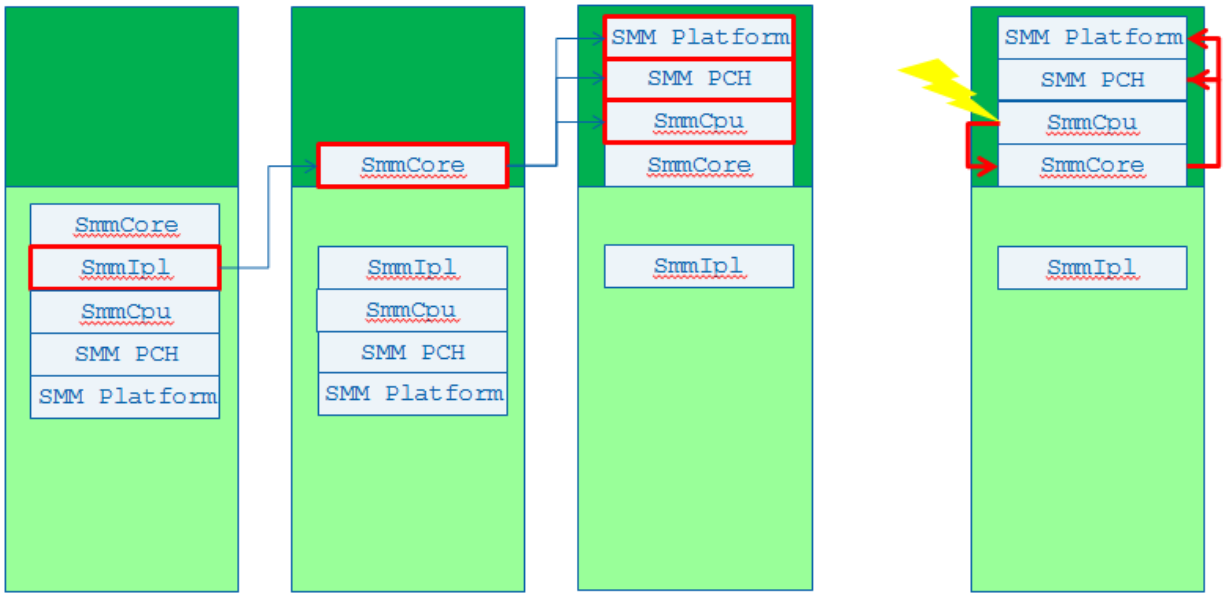

Figure 3 Full SMM Core solution

In simplified version, we remove the SmmCore. We let the SmmIpl find the SmmCpu driver and load it into SMRAM to do the SMBASE rebase operation. Another activity entails the conversion of all SmmPlatform drivers into a library and link this library into SmmCpu drive. Then the next SMI will trigger a machine mode switch so that control is passed into the SmmCpu driver, and the SmmCpu driver will call a SmmPlatform library to dispatch the SMI handler.

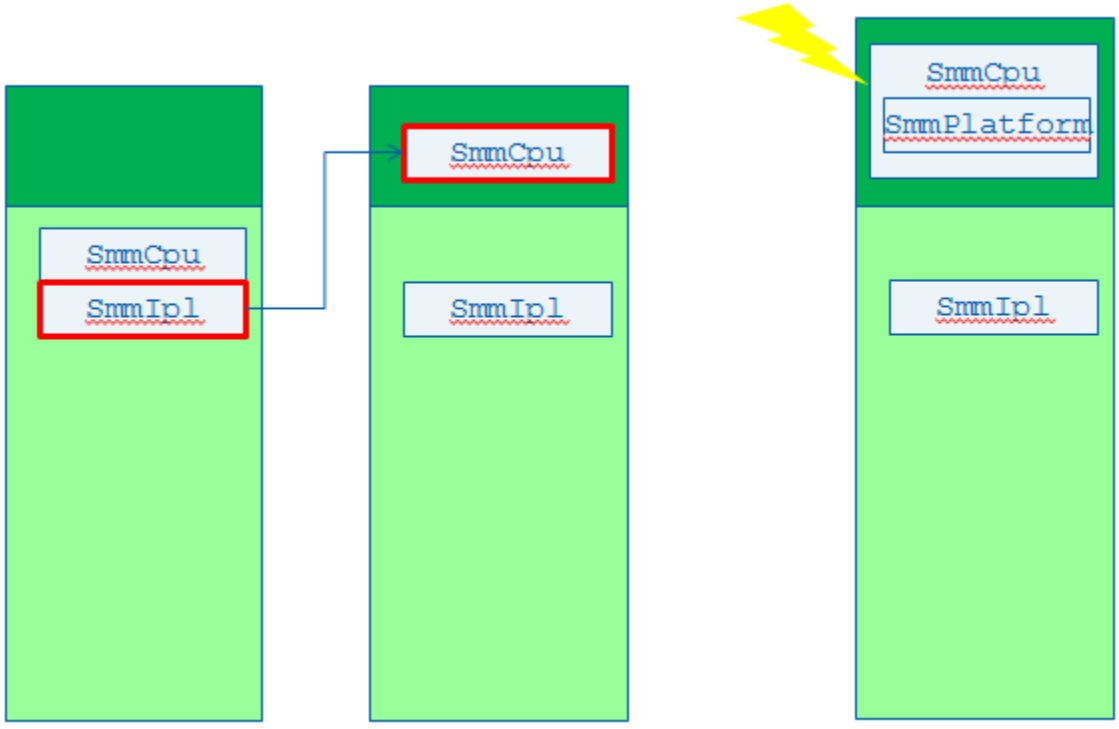

Figure 4 Simplified SMM solution

Please refer to QuarkPlatformPkg.dsc and QuarkPlatformPkg.fdf for more detail on simplified version driver.

For the SmmIpl/SmmCpu module, refer to IA32FamilyCpuBasePkg/SimplePiSmmCpuDxeSmm
For SmmPlatform lib, refer to QuarkPlatformPkg/Library/SmmPlatformHookLib.

Compiler Support

1) Do not use the /Zi compiler flag
2) Do not use the /DEBUG link flag
3)

These flags can be added in the debug phase, but do not use them in final release, so that debug information in PE image is removed.

Build Option

1) Use NULL Debug lib.

```
DebugLib|MdePkg/Library/BaseDebugLibNull/BaseDebugLibNull.inf
```

Refer to QuarkPlatformPkg.dsc

2) Do not use dynamic PCD.

```
PcdLib|MdePkg/Library/BasePcdLibNull/BasePcdLibNull.inf
```

Refer to QuarkPlatformPkg.dsc

3) Use TE image

```
[Rule.Common.SEC]
  FILE SEC = $(NAMED_GUID) RELOCS_STRIPPED {
    TE  TE    Align = 8       $(INF_OUTPUT)/$(MODULE_NAME).efi
    RAW BIN   Align = 16      |.com
  }
```

Refer to QuarkPlatformPkg.fdf

4) Do not use UI/VER section.

```
[Rule.Common.DXE_DRIVER]
  FILE DRIVER = $(NAMED_GUID) {
    DXE_DEPEX DXE_DEPEX Optional       $(INF_OUTPUT)/$(MODULE_NAME).depex
    PE32      PE32                     $(INF_OUTPUT)/$(MODULE_NAME).efi
#   UI        STRING="$(MODULE_NAME)" Optional
#   VERSION   STRING="$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
}
```

Refer to QuarkPlatformPkg.fdf

5) Rebase when build.

```
[FV.EDKII_BOOT_SLIM]
BlockSize          = 0x1000
FvBaseAddress      = 0x80010000
FvForceRebase      = TRUE
...
```

Refer to QuarkPlatformPkg.fdf

6) Use LZMA F86 compression

```
FILE FV_IMAGE = 9E21FD93-9C72-4c15-8C4B-E77F1DB2D791 {
        SECTION GUIDED D42AE6BD-1352-4bfb-909A-CA72A6EAE889
PROCESSING_REQUIRED = TRUE { # LzmaF86
        SECTION FV_IMAGE = EDKII_BOOT_SLIM
       }
     }
```

Refer to QuarkPlatformPkg.fdf

Result

Table 1 TinyQuark ROM module size

NonCompressed	Size (byt	Category
SecTrampoline	8012	Platform
ResetVector	1208	Platform
Summary		
Generic	0	
Silicon	0	
Platform	9220	

Compressed	Size (byt	Category
DxeIpl (MRC - 17600)	22084	Platform
SimpleDxeCore	46652	Generic
SimpleCpuArch	5124	Generic
Metronome	1252	Generic
RuntimeDxe	3844	Generic
PlatformVariableRuntime	1764	Generic
ResetSystemRuntime	1572	Generic
SimplePcRtc	3748	Generic
PlatformInitDxe	5732	Platform
PlatformFixedPciResource	3140	Platform
QNCInitDxe	5316	Silicon
AcpiPlatform	4100	Platform
AcpiTable	8850	Platform
SimpleSmmIpl	6164	Generic
SimplePiSmmCpu	6724	Platform
IohInitDxe	1268	Silicon
CombindedTerminalDxe	5332	Generic
Legacy8259	2020	Generic
TinyBds	2116	Platform
Summary		
Generic	77472	
Silicon	24184	
Platform	35146	

	Final (by	Percentage	Meaning
Generic	**30989**	**48.46%**	UEFI generic API
Silicon	**9674**	**15.13%**	Silicon Initialization
Platform	**23278**	**36.41%**	Platform Initialization
total	**63941**		

Summary

This section describes how to reduce ROM flash size.

RAM Footprint optimization

Analysis

In the earlier section we reduce the amount of real-estate consumed in the SPI NOR flash, thus allowing more space for the operating system and other data. But the SPI NOR isn't the only factor to impact the bill of materials (BOM). The other factor that impacts a board cost is volatile memory or RAM usage.

As such, we did an analysis on RAM footprint on the 64K TinyQuark, and we realized that the image in Quark firmware is copied 4 times during boot, which can be avoided in practice.

1) DxeIpl, will prepare Decompressed FV to DxeCore. This is first copy.
2) Once DxeCore finds a FV, it will copy FV into RAM. The reason is that DxeCore does not know if it is on flash or DRAM.
3) Then in DriverDispatch phase, DxeCore constructs a list for all FFS and SECTION there. The section stream for FFS is 3rd copy for each PE/COFF image.
4) Finally, when DxeCore starts loading UEFI image, it allocates another memory and use PeCoff library to load and relocate PE image. Now it is 4th copy.

The detailed memory layout before optimization is below:

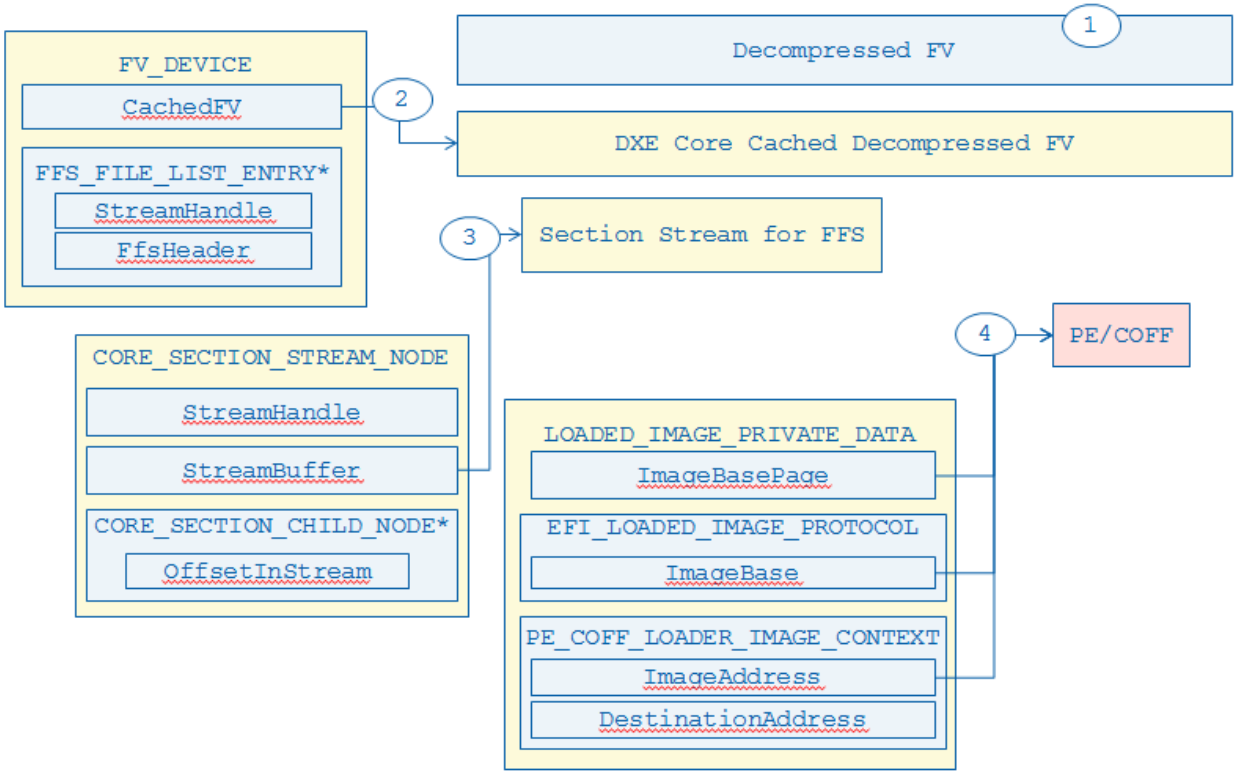

Figure 5 Current DXE Core

Optimization

So the optimization could be: reuse old buffer as much as possible.

1) The DecompressedFv buffer is the base.
2) The CachedFv could be a pointer to original DecompressedFv buffer.
3) The section stream for FFS can still point to original buffer.
4) Then for the PE/COFF image, we relocate all PE/COFF images at build time. So a normal DXE driver or UEFI driver is execute-in-place (XIP), so that it can be run directly from the SPI NOR. For the DxeRuntime driver, it is special because it needs to reside in runtime memory (i.e., memory of type EFI_MEMORY_RUNTIME [UEFI]). So the DxeCore just needs to allocate Runtime Paged memory and do the relocation for the runtime driver.

The detailed memory layout after optimization is below:

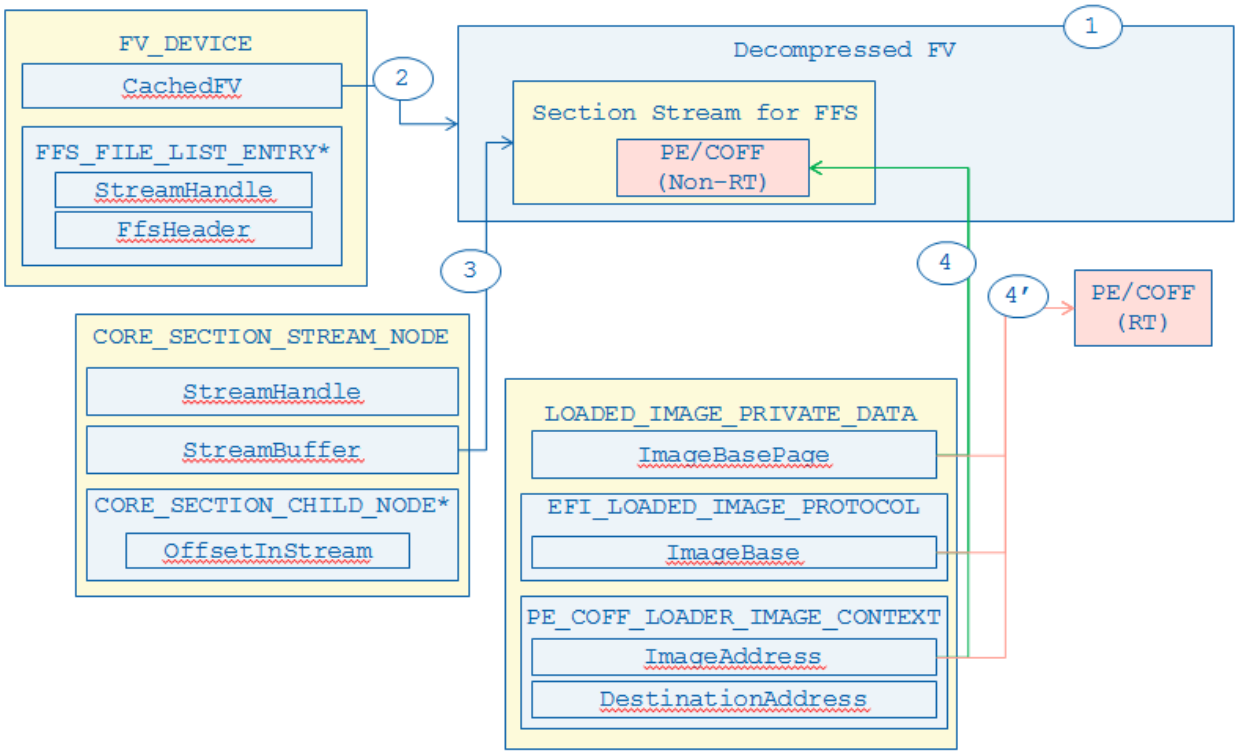

Figure 6 Enhanced Simplified DXE Core

Result

Before optimization, the memory size used is 760K (624K allocated during boot + 136K decompressed FV). After optimization, the memory size used is 340K (204K allocated during boot + 136K for the decompressed FV).

Table 2 TinyQuark component RAM size

Component (MemType)	Size (K)	Page
1) Decompressed FV	136	34
ACPI Reclaim (9)	48	12
ACPI NVS (0xA)	8	2
Runtime Code (5)	16	4
Runtime Data (6)	12	3
Boot Code (3)	0	0
Boot Data (4)	120	30
-- Stack	32	8
2) Total allocated during boot	204	51
Total memory Used	340	85

Summary

This section describes the how to reduce RAM footprint size.

Conclusion

This Intel implementation of EDKII is a demonstration showing the possibilities available using the scalable architecture of EDKII source code technology and the flexibility of the UEFI Specification. The 64K “TinyQuark” demonstrates the scalability of the EDKII architecture and how to create a UEFI conformant Firmware solution that has a very small flash size and can minimize DRAM usage. This allows for a slim boot load environment for a subsequent UEFI OS or a slim execution environment for bare metal execution that can suffice just using UEFI services.

Glossary

Galileo - Intel® Galileo development board uses Quark.

PI – Platform Initialization. Volume 1-5 of the UEFI PI specifications.

Quark – The Intel® Quark SoC X1000 is the first product in a new roadmap of innovative, small core products targeted at rapidly growing areas ranging from the industrial IoT to wearables.

SMM – System Management Mode. x86 CPU operational mode that is isolated from and transparent to the operating system runtime

SPI NOR – serial peripheral interface not-OR flash. This is memory mapped flash for execute in place firmware and features byte read/write and block erase.

UEFI – Unified Extensible Firmware Interface. Firmware interface between the platform and the operating system.

XIP – Execute-in-Place. A style of execution from a memory-mapped flash device, such as SPI NOR

References

[ACPI] Advanced Configuration and Power Interface (ACPI) version 5.1 www.uefi.org
[EDK2] UEFI Developer Kit www.tianocore.org

[BIO] Leslie Berlin, "The Man Behind the Microchip," Oxford University Press, November 2006

[GALILEO] Intel® Galileo development board http://uefidk.intel.com/content/get-started-intel-galileo-development-board

[IOT] Internet of Things http://www.intel.com/content/www/us/en/internet-of-things/overview.html

[QUARK] Intel Quark Technology http://uefidk.intel.com/projects/quark

[SATA] Serial ATA www.sata-io.org

[SECURE] Writing Secure Code http://msdn.microsoft.com/en-us/security/aa570401.aspx

[SDM] IA-32 Software Developer Manual http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[TCG] Trusted Computing Group and its associated Trusted Platform Module (TPM) www.trustedcomputinggroup.org

[UEFI] Unified Extensible Firmware Interface (UEFI) Specification, Version 2.4.b www.uefi.org

[UEFI Book] Zimmer,, et al, "Beyond BIOS: Developing with the Unified Extensible Firmware Interface," 2nd edition, Intel Press, January 2011

[UEFI Overview] Zimmer, Rothman, Hale, "UEFI: From Reset Vector to Operating System," Chapter 3 of *Hardware-Dependent Software*, Springer, February 2009

[UEFI PI Specification] UEFI Platform Initialization (PI) Specifications, volumes 1-5, Version 1.3 www.uefi.org

[USB] Universal Serial Bus (USB) www.usb.org/home

[YOCTO] Yocto Linux project https://www.yoctoproject.org/

Authors

Jiewen Yao (jiewen.yao@intel.com) is EDKII BIOS architect, EDKII FSP package maintainer with Software and Services Group at Intel Corporation.

Vincent J. Zimmer (vincent.zimmer@intel.com) is a Senior Principal Engineer with the Software and Services Group at Intel Corporation.

Elvin Li (elvin.li@intel.com) is a EDKII BIOS architect and developer with the Software and Services Group at Intel Corporation.

Chris Li (chris.li@intel.com) is a EDKII BIOS architect and developer with the Software and Services Group at Intel Corporation.