RISC-V and UEFI[1]

Dong Wei, Abner Chang, Vincent Zimmer[2]
Hewlett Packard Enterprise Intel Corporation
November 19, 2015

Abstract

RISC-V describes a composable architecture for various instruction set and system profiles to have a central processing unit (CPU) in a system on a chip (SOC). A CPU in isolation cannot provide a full solution to support running various operating systems. To that end, the CPU needs to be situated inside of a platform. And for this SOC-platform to launch operating systems (OS), some hand-off between the platform and OS loader is required. To avoid having to hard-code platform details into the OS, industry standards for platform booting under the umbrella of the UEFI Forum, such as the UEFI (Unified Extensible Firmware Interface) Specification and ACPI (Advanced Configuration and Power Interface) Specification come into play. And to build out the UEFI interfaces, open source code base technologies such as EDK II provide potential building blocks. This paper will discuss how EDK II can be used to provide a binding to RISC-V, including the approach and challenges therein.

1 RISC-V Background

Details on the RISC-V instruction set and system features can be found at [1]. RISC-V builds on the heritage of the classic RISC CPU's and features instruction set architecture widths, including 32-bit, 64-bit, and 128-bit. The design is open such that the elements can be built into SOC's that scale from small Internet of Things (IOT) to scale-out servers that need huge address space access.

2 UEFI

EDK II [3] is an open-source implementation of the UEFI specification. The UEFI Forum includes both the main UEFI Specification (intended to provide an abstraction between the operating system and the underlying firmware) and the Platform Initialization (PI) specifications(describe a modular means by which to implement the UEFI Specification).

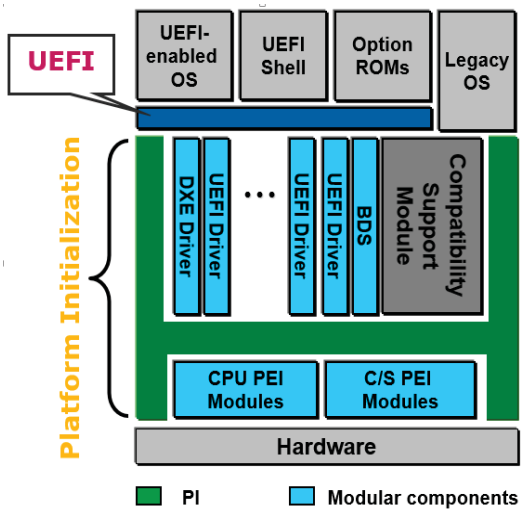

Figure 2 System Diagram of UEFI and PI APIs

Figure 1 above shows the layering with the UEFI API's at the top and the underlying PI API's at the bottom.

The open source implementation is generic, and there are various CPU-specific elements, such as https://github.com/tianocore/edk2/tree/master/ArmPkg for ARM Ltd. CPU's, both 32-bit and AArch64, and https://github.com/tianocore/edk2/tree/master/UefiCpuPkg for Intel-architecture based 32-bit and 64-bit CPU's. And overview of silicon support can be found in [5].

Chapter 2 of the UEFI 2.5 specification describes the specifics of a CPU binding, including the machine state for a given CPU. The UEFI PI specification describes the various architectural protocols (Aps) that must be provided in order to make the generic DXE main work with different platforms and CPU architectures. In general, these AP's provide support for a timer, CPU interrupt, and non-volatile storage capability. UEFI in general has a polled driver model so only a timer tick is necessary from the platform.

The EDK II open source implementation is written in

[1] This paper has been submitted to the 3nd RISC-V workshop
[2] The authors can be reached at dong.wei@hpe.com, abner.chang@hpe.com, vincent.zimmer@intel.com

pure ANSI C and works with GCC, Intel Compiler, Microsoft Compiler, Clang LLVM/XCode, and many other tool chains.

3 RISC-V and UEFI

To support RISC-V, the approach would be to create a RiscVCpuPkg, RiscVPlatformPkg and OvmfRiscVPkg. The first is generic RISC-V CPU support on UEFI, the second could be the real platform support such as ZYBO development board. The last could be based upon a software simulator like QEMU.

The value of having UEFI support is that UEFI and ACPI, with the hardware reduced bit of the latter, will allow for re-use of upstream operating support, such as the Linux kernel. Instead of having sub-architectures for each SOC, there can be a single kernel that parameterizes the platform differences with UEFI and ACPI. This is how the ARM community went from many sub-archs to a single CPU support directory, akin to the arch/x86 for the Intel-architecture based systems.

The present implementation has mapped the necessary EDK II infrastructure to RISC-V. Detailed status includes the following:

Table 1 Status of the EDK II port

Task	Note
Reset Vector	RISC-V high location reset vector.
Security Core (SEC phase)	- Platform early initialization. - RISC-V temporary memory for early Pre EFI Initialization.
Pre EFI Initialization (PEI phase)	- Publish permanent memory for Driver Execution Environment (DXE). - Platform initialization. - RISC-V specific PEI service table retrieval.
DXE Initial Program Loader (DXEIPL)	RISC-V specific PEI to DXE handoff driver - RISC-V page table establishment. - RISC-V exception table establishment.
BDS boot device select (BDS phase)	RISC-V platform UEFI Boot Manager library
PI DXE Architecture Protocol	RISC-V CPU Arch Protocol - Interrupt disable/enable/handler RISC-V Timer Arch Protocol - Set timer period/handler RISC-V Reset Arch Protocol - Reset CPU/platform Real Time Clock Arch protocol - Set/Get time.
EDKII MdePkg	ProcessorBinding – RISC-V specific defines and types. - Variable aligned at its natural size (RISC-V specific). - Structure at 8-byte alignment. BaseLib - RISC-V disable/enable interrupt. - RISC-V specific stack switch, includes SeJump and LongJump. - RISC-V unaligned memory manipulation (memory unaligned read/write). IoLibIntrinsic - RISC-V memory map I/O read/write. PeCoffLoade - RISC-V [20:12] relocation. BaseSunchronization
CpuIO protocol/PPI	CPU I/O protocols on DXE phase, PPI on PEI phase.
PEI Reset PPI	PEI reset interface.
Manage Mode (SMM) PPI/Protocol	
ACPI (processor)	
S3 resume	
MP Service	

Table 2 Status of the EDK II Tools for RISC-V

Task	Note
RISC-V ELF to PE/COFF (EDKII GenFw)	- Handle RISC-V ELF image. - PE/COFF machine type for RISC-V. - RISC-V ELF relocation types to RISC-V EFI image relocation type. - Relocate image when generate PE/COFF. According to RISC-V relocation type.
EDKII Gen Firmware Volume	Relocate EFI firmware file according to RISC-V EFI image relocation type when generate EDKII firmware volume

There is a gap where the PE/COFF specification does not have a machine type for RISC-V yet.

Table 3 Status of the GNU Linker for RISC-V

Task	Note
GNU linker for RISC-V	Support “–no-relax”. Disable global optimization. Link option “–relax” affects function pointer usage on EDKII.

4 Related Work

Although there is not a RISC-V binding in chapter 2 of the UEFI 2.5 specification, other CPU’s, such as ARM and Itanium, are supported.

5 Conclusion

We have discussed industry-standard UEFI firmware and how to prepare for a new architecture, such as RISC-V. EDK II port for RISC-V is underway. The outcome of this work is to define the RISC-V binding in a future UEFI Specification and contribute the corresponding EDK II port to open source.

Acknowledgments

We would like to thank the UEFI Forum members for the work on the UEFI, PI, and ACPI specifications, along with the community for work on the EDK II implementation.

6 References

[1] RISC-V website http://www.riscv.org/

[2] UEFI website http://www.uefi.org/

[3] EDK II project http://www.tianocore.org/