presented by

ARM®

Porting a PCI driver to ARM AArch64 platforms

UEFI Spring Plugfest – May 18-22, 2015
Presented by Olivier MARTIN (ARM Ltd)

Updated 2011-06-01

Agenda

- Context
- About PCI
- Recommendations & Good Practices
- About EFI Byte Code (EBC)
- At this event

Context

ARM platforms until recently

- Mainly mobile consumer oriented or embedded platforms

… no strong push for PCI support

ARM platforms until recently

- Most platforms had Ethernet & USB supports

… and sometimes SATA

… and even some confidential PCI support

Marvell Sheevaplug Development Kit

Freescale i.MX6

… but always memory mapped devices

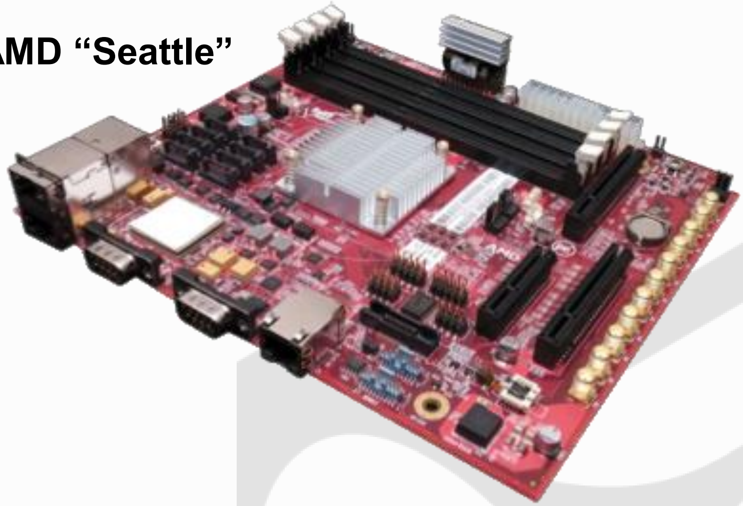
New Opportunities & Markets

- Parity with other architectures / platforms
- Need to address Server market requirements

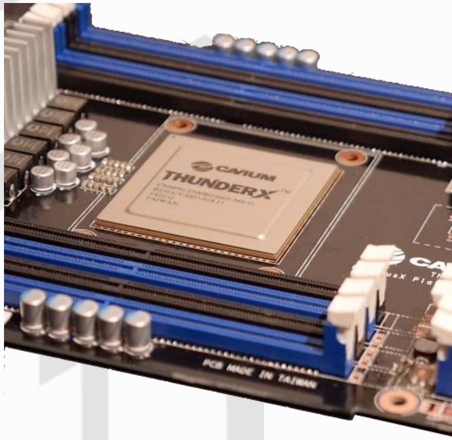
Latest ARM platforms

Applied Micro
X-C1™

AMD “Seattle”



Cavium
ThunderX™



HiSilicon D02

ARM Juno

About PCI

PCI in the context of UEFI

- PCI Devices might be enumerated at boot time
- UEFI driver in PCI Option ROM
- ACPI ‘MCFG’ Table exposed by UEFI firmware

- … Need for OS Generic PCI Root Bridge driver

PCI in the context of ARM

- It should not be different compared to other architectures

PCI in the context of ARM

- It should not be different compared to other architectures … in theory

PCI in the context of ARM

- It should not be different compared to other architectures … in theory*

* - MSI only supported from ARM GICv2m specification

- PCI Bus not necessarily coherent with the CPU
- No PCI IO space support
- Likely to be ECAM only

Recommendations & Good practices

Recommendation 1

- Do not use direct memory access / MmioLib
  - ➢ Some PCI Root Complex require translation logic to convert from PCI to AXI buses
  - ➢ Avoid architecture / platform specificities

Recommendation 1 (cont…)

In practice, move from:

```
Value32 = MmioRead32 (Port->RegBase +
               SII3132_PORT_SSTATUS_REG);
*(UINT32)(SataPort->RegBase + SII3132_PORT_INTSTATUS_REG)=
               IrqMask;
```

to:

```
Status = PciIo->Mem.Read (PciIo, EfiPciWidthUint32,
   Port->RegBase + SII3132_PORT_SSTATUS_REG, 1, &Value32);
Status = PciIo->Mem.Write (PciIo, EfiPciWidthUint32,
   SataPort->RegBase + SII3132_PORT_INTSTATUS_REG, 1,
   IrqMask);
```

Recommendation 1 (cont…)

In practice, move from:

```
CopyMem ((VOID*)(SataPort->RegBase + (EmptySlot * 0x80)),
        SataPort->HostPRB, sizeof (SATA_SI3132_PRB));
```

to:

```
PciIo->Mem.Write (PciIo, EfiPciIoWidthUint8, 1, // Bar 1
        SataPort->RegBase + (EmptySlot * 0x80),
        sizeof (SATA_SI3132_PRB), SataPort->HostPRB);
```

Recommendation 1 (cont…)

In practice, move from:

```
CopyMem ((VOID*)(SataPort->RegBase + (EmptySlot * 0x80)),
        SataPort->HostPRB, sizeof (SATA_SI3132_PRB));
```

to:

```
PciIo->Mem.Write (PciIo, EfiPciIoWidthUint8, 1, // Bar 1
        SataPort->RegBase + (EmptySlot * 0x80),
        sizeof (SATA_SI3132_PRB), SataPort->HostPRB);
```

or even better:

```
PciIo->Mem.Write (PciIo, EfiPciIoWidthUint32, 1, // Bar 1
        SataPort->RegBase + (EmptySlot * 0x80),
        sizeof (SATA_SI3132_PRB) / 4, SataPort->HostPRB);
```

Recommendation 2

- Do not use TimerLib!
  - ➢Since ARMv8, ARM introduced a Generic Timer (similarly from ARMv7 there has been the ‘Generic Timer Extension’).
    - ➢But the UEFI specification already offers API for this purpose - see `BootServices.Stall()`

Good practice 1

- Be aware you might have multiple instances of the same PCI card plugged in your platform!
  - ➢ Consider carefully global variables!

```
EFI_PCI_IO_PROTOCOL* gPciIo; // No!!!!
```

Good practice 2

- Build your PCI driver with:
  - ➢ different toolchains (MS Visual Studio, GCC, LLVM, etc)
  - ➢different architectures (32-bit, 64-bit, ARM, Intel, etc)

Good practice 3

- Use UEFI protocols / Boot Services in preference to EDK(2) Libraries

👍 Code smaller, driver more portable

👎 Rely on the platform UEFI firmware

➢ But that’s why we have UEFI conformance tests!

Note

- All these recommendations are not specific to ARM architectures and PCI UEFI drivers!

In practice…

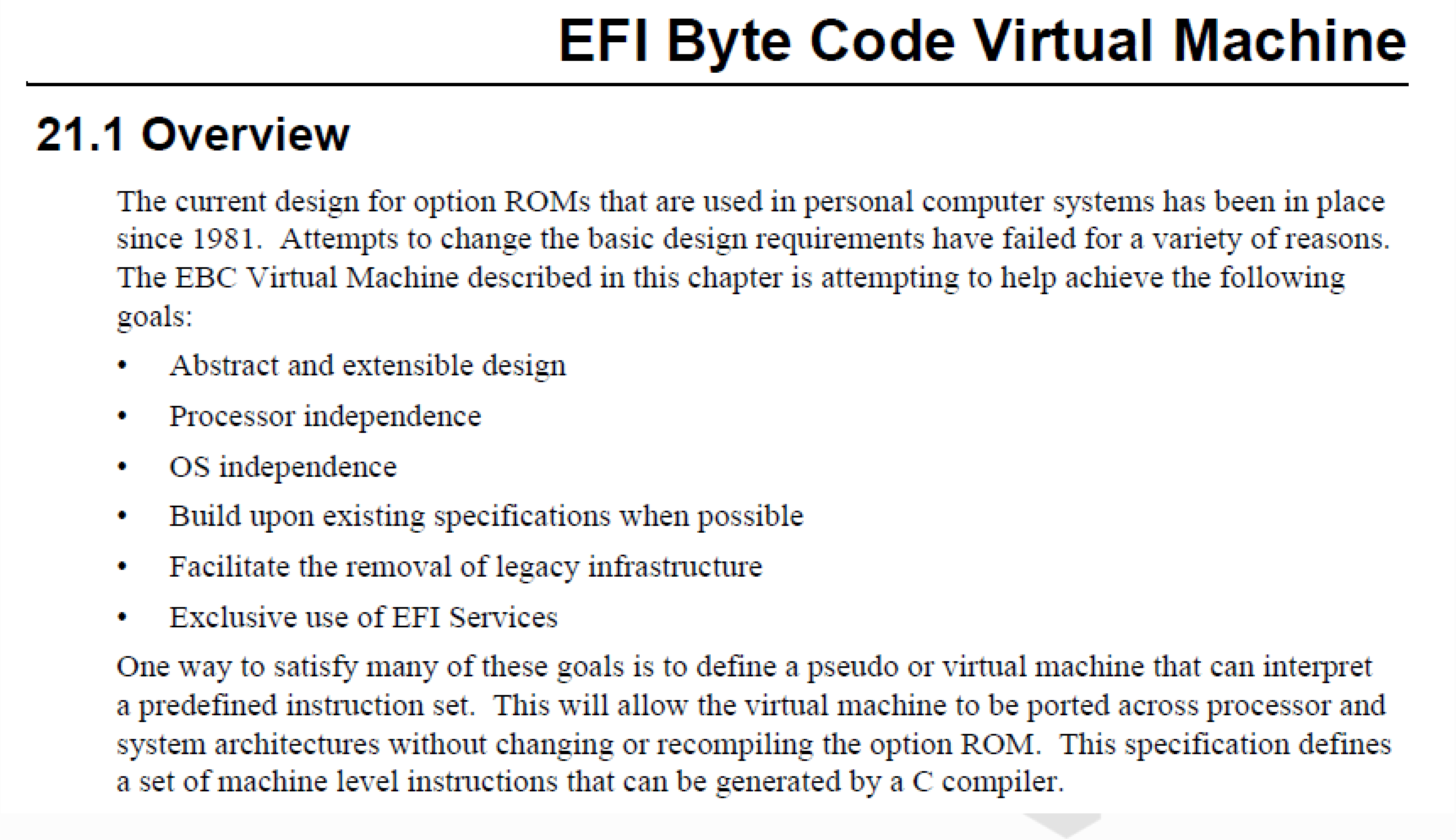
- Ensure your PCI driver lives into an architecture independent EDK(2) package

- Review the library dependencies in your driver INF file.

- Build your driver with at least two toolchains and architectures and DEBUG/RELEASE

About EFI Byte Code (EBC)

What is / Why EBC?



EFI Byte Code Virtual Machine

21.1 Overview

The current design for option ROMs that are used in personal computer systems has been in place since 1981. Attempts to change the basic design requirements have failed for a variety of reasons. The EBC Virtual Machine described in this chapter is attempting to help achieve the following goals:

- Abstract and extensible design
- Processor independence
- OS independence
- Build upon existing specifications when possible
- Facilitate the removal of legacy infrastructure
- Exclusive use of EFI Services

One way to satisfy many of these goals is to define a pseudo or virtual machine that can interpret a predefined instruction set. This will allow the virtual machine to be ported across processor and system architectures without changing or recompiling the option ROM. This specification defines a set of machine level instructions that can be generated by a C compiler.

The questions of EBC support

- Should EBC work on ARM?
  - Yes

The questions of EBC support

- Should EBC work on ARM?
  - Yes, but it is not implemented at the moment

The questions of EBC support

- Should EBC work on ARM?
  - Yes, but it is not implemented at the moment
- What do I need to build an EBC PCI driver?
  - A $995 compiler…

The questions of EBC support

- Should EBC work on ARM?
  - ➢ Yes, but it is not implemented at the moment
- What do I need to build an EBC PCI driver?
  - ➢ A $995 compiler…
- Should your driver support EBC?
  - ➢ Some people say yes, and other say no…

At this event

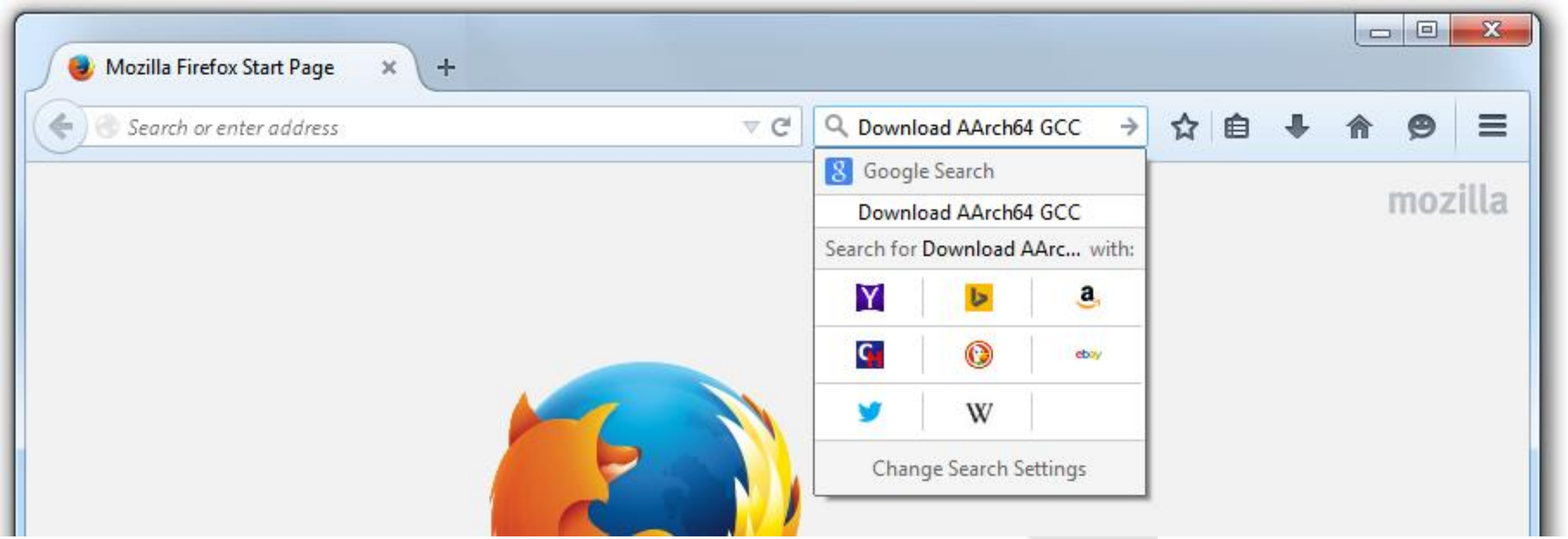
Take advantage of this event

- There will be ARM platforms with PCI support (likely to be AArch64 platforms)

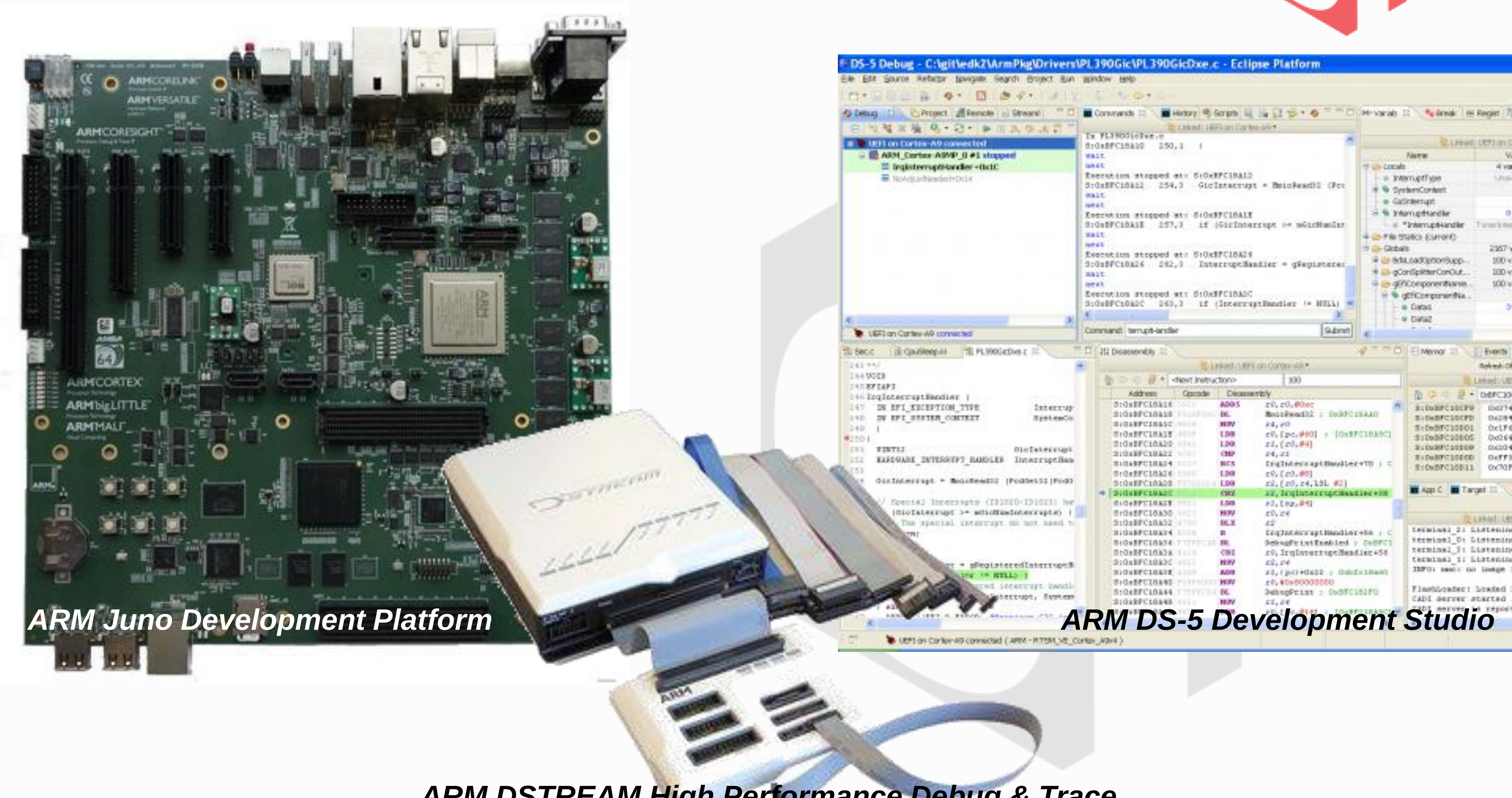
- ARM Engineers with platforms and debug tools to help you to test your PCI drivers and support you

First step to support ARM…

- Build your driver (for free)



Going further…



DS-5 Debug - C:\git\edk2\ArmPkg\Drivers\PL390Gic\PL390GicDxe.c - Eclipse Platform

```
In PL390GicDxe.c
S:0xBFC18A10   250,1   {
wait
next
Execution stopped at: S:0xBFC18A12
S:0xBFC18A12   254,3   GicInterrupt = MmioRead32 (Pcc
wait
next
Execution stopped at: S:0xBFC18A1E
S:0xBFC18A1E   257,3   if (GicInterrupt >= mGicNumInt
wait
next
Execution stopped at: S:0xBFC18A26
S:0xBFC18A26   262,3   InterruptHandler = gRegisterec
wait
next
Execution stopped at: S:0xBFC18A2C
S:0xBFC18A2C   263,3   if (InterruptHandler != NULL)
```

ARM Juno Development Platform

ARM DS-5 Development Studio

ARM DSTREAM High Performance Debug & Trace

Thanks for attending the
UEFI Spring Plugfest 2015

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

ARM®