

presented by



Is your Pi “ServerReady”?

Embracing UEFI and ACPI at the Edge

UEFI 2020 Virtual Plugfest

Wednesday, April 29

Presented by Andrei Warkentin (VMware) and Samer El-Haj-Mahmoud (Arm)

Meet the Presenters



Samer El-Haj-Mahmoud
Senior Principal Architect
Member Company: Arm



Andrei Warkentin
Arm Enablement Architect
Member Company: VMware

Agenda



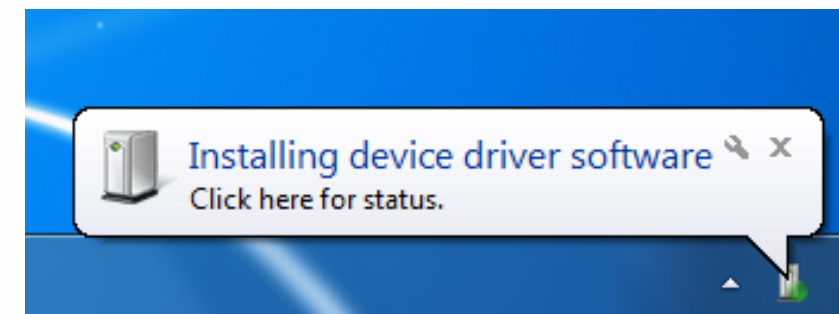
- Intro to Arm ServerReady
- What about SBCs?
- Making Pi “Boring”
- Challenges
- Demo
- Call to Action

Arm ServerReady



- Makes Arm servers a multi-player in horizontally-integrated ecosystem.
- Blueprint for “just works” Arm-based infrastructure
 - Industry standards (PCIe, UEFI, ACPI, SMBIOS, etc...)
 - Server Base System Architecture (SBSA).
 - Server Base Boot Requirement (SSBR).
 - Arm’s Server Architectural Compliance Suite (ACS).
- Just as “boring” to consume as x86 counterparts.
 - Not weird to OEM/ODMs.
 - Not weird to IT.
 - Not weird for software developers.
 - Easily support Commercial Off-the-Shelf (COTS) and proprietary system software.
 - ...a horizontally-integrated ecosystem.


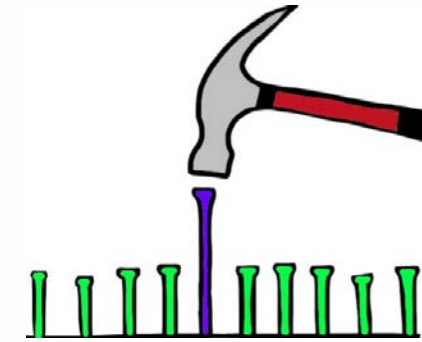
A single binary OS distribution to boot on any system – enough to install missing drivers.



Server Base System Architecture (SBSA)



- Hardware requirements for OS vendors.
- Multiple levels, each corresponding to a set of increasing conformity:
 - Processor / memory features.
 - UART, Interrupt Controller, SMMU.
 - PCIe[®] technology integration.
 - Common I/O peripherals like USB/SATA.
 - ...
- New levels added as architecture evolves, including/refining previous levels.
- Currently at Level 6.



A screenshot of the ARM Developer Documentation website, showing the 'Documentation' section for 'A-Profile Architecture Specifications'.

Arm Arch:

- Armv8.x-A
- SMMU
- GIC
- Extensions:
 - RAS
 - MPAM

Industry Standards



Future



Example: SBSA L6



Processor Element (processors cores)

- Up to 2^{28} PEs
- AArch64 at all Exception Levels
- Must to provide EL2 and EL3
- Advanced SIMD extensions.
- Instruction Caches VIPT or PIP
- 16-bit ASID support.
- 4KB and 64KB translation granules at stage 1 and stage 2.
- All PEs are coherent and in the same Inner Shareable domain.
- Where export restrictions allow, cryptography extensions.
- Little-endian support.
- 6 PMU counters, 6 breakpoints, 4 watchpoints
- CRC32, cond. Scalar vector ext.

- Standard RAS
- 16 bit VMID
- Virtual host extension
- Cond. pointer signing
- Cond. Clean to point of serialization
- Nested virtualization improvements
- Pointer signing
- Coresight standard support
- Cond. MPAM
- Crypto additions (SHA3/512 SM3/4)
- Activity monitor extension
- **SVE 128bit**
- **ARMv8.5 and Security**

System level requirements

- Interrupts
 - GICv3 (generic MSI support!)
 - Standard interrupt numbers
 - No non-standard extensions
- IO virtualization
- SMMUv3 (ATS!)
 - **Access/Dirty Flags mandated**
- **Easier page sharing PE and SMMU**
- PCIe for all assignable devices
- **PCIe enhancements (Appendix E)**
 - **ECAM**
 - **On-chip Peripherals**

Memory

- No deadlocks when accessed by processor or device
 - MMIO Peripherals 64Kb apart
- ### System counter scaling
- **Nanosecond generic counter**

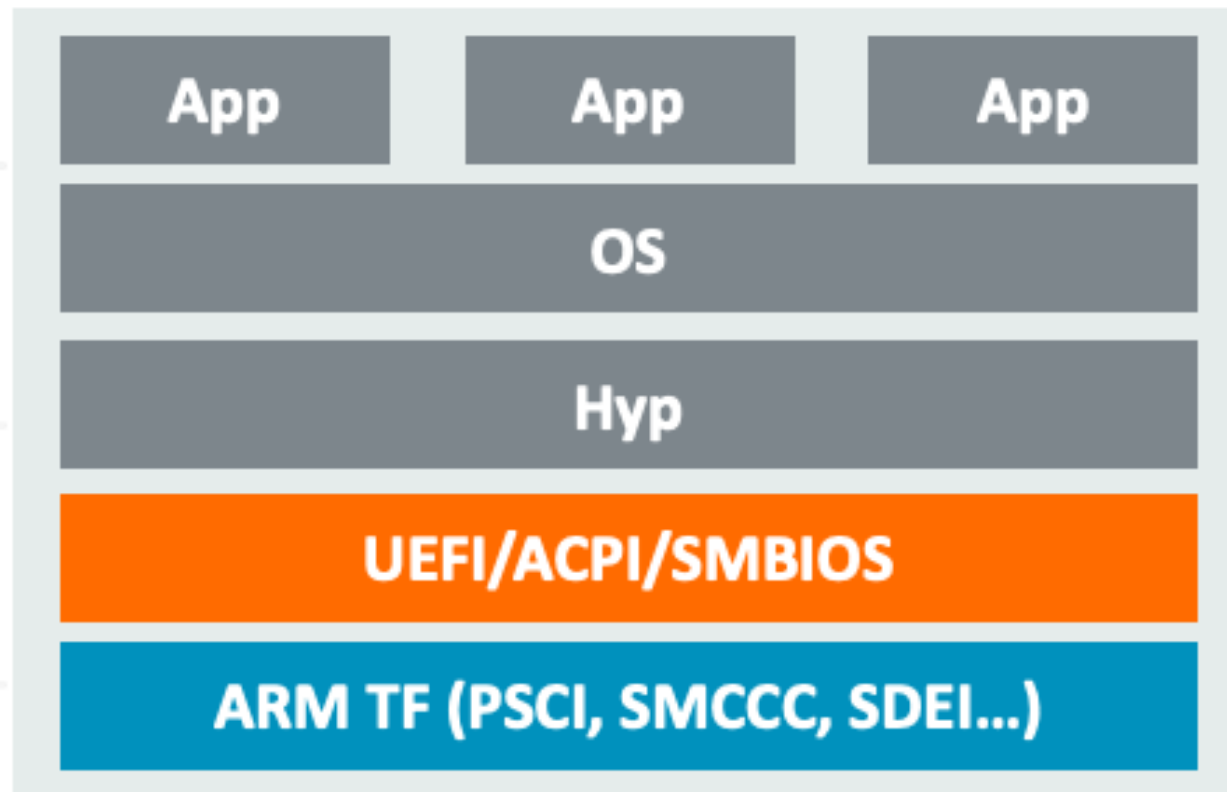
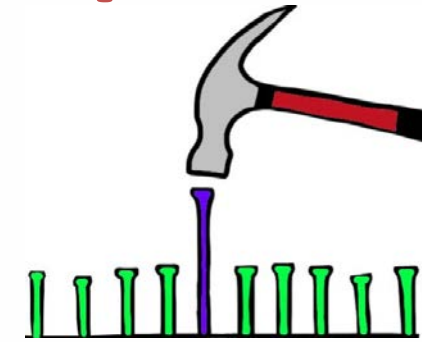
Other

- Power
- Arm standard wake up timer
- Peripherals
- Arm standard UART
- Arm standard WDog
- EHCI v1.0 or later
- XHCI v1.0 or later
- AHCI v1.3 or later

Server Base Boot Requirements (SBBR)



OSV firmware requirements: adopt industry standards where they exist, complement with Arm-specific interfaces where mandatory.



A screenshot of the Arm Developer website showing the 'Documentation' page for 'A-Profile Architecture Specifications'. Below the screenshot, the text 'Arm Specs' is followed by a list of specifications:

- PSCI
- SMCCC
- TF-A
- Arm FFH
- Arm MM

Industry Standards

- UEFI
- ACPI

- SMBIOS

- TCG FW spec

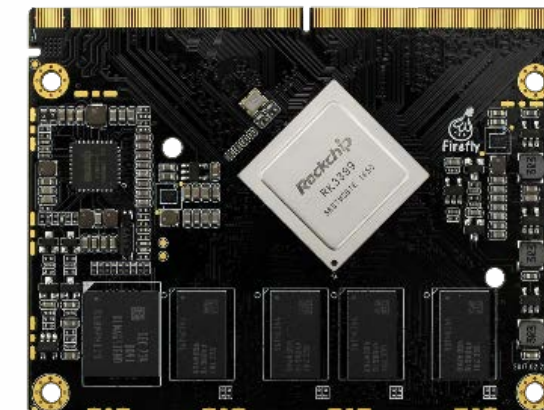
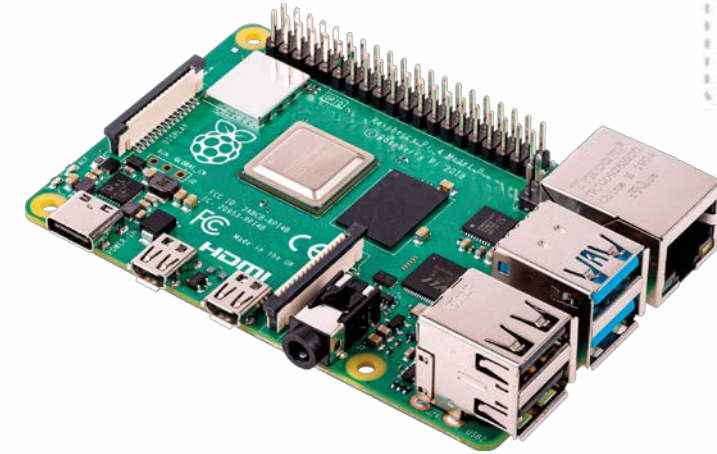
- PCI FW spec

- OPEN Compute Project®

Beyond Servers?



- Non-Server Arm devices are traditionally non-traditional.
- Even higher-end devices are treated as embedded.
 - Vertically-integrated ecosystem.
 - Very little off the shelf software.
 - Custom firmware (sometimes not even U-Boot).
 - Custom system software (non-upstreamed kernels, custom distros).
 - Poor support after just a few years.
 - Vendor lock-in.
- This class of systems is positioned for IoT/Edge, but there's no common ecosystem. This is getting in the way of adoption and proliferation of Edge/IoT deployments.



The Light at the End of the Tunnel



- Standards aren't just for servers.
- At TechCon'19 Arm extended "ServerReady" to other segments, where COTS "general purposes" system software needs to run. (see [Arm whitepaper](#))
- "Server" requirements → "System" requirements (for all "A class" Arm systems)?
 - Server devices (e.g. datacenter, infra edge)
 - Client devices (e.g. laptops)
 - Edge devices (e.g. IoT gateways)
 - Facilitates creating a horizontally-integrated ecosystem.
 - Based on SBSA, SBBR, SBSG (security)
- What about EBBR?
 - Embedded Base Boot Requirements stop at describing firmware/OS hand-off.
 - Don't standardize on describing or interfacing hardware.
 - Examples:
 - Raspberry Pi with U-Boot + FDT, loading UEFI OS loader
 - HP Envy x2 WoA laptop: UEFI + ACPI, but can't boot a non-Snapdragon OS image.



Let's Do Something!



- **Why Raspberry Pi 4?**

- High-volume, low-cost (\$50) device for everyone.
- People may have not heard of Arm, but they have heard of the Raspberry Pi.
- Pi will be around for a while (unlike a few other boards I can think of).

- **Is it feasible?**

- Pi 4 finally has GICv2. XHCI for USB
- Can safely ignore PCIe (no physical slot) – Level 2/3 SBSA compliance should be possible.

- **How should it be done?**

- Out in the open community (true OSS project).
- Set an example and show others how it's done.

I'm a server buff - and why should I care?

- No good widely available client platforms today to build a good mass of developers.
- A \$50 device that “just works” almost like a “server” is a great enabler for the Arm server ecosystem.

What could you boot with this?

- SBBR/SBSA OSES (ESXi, RHEL, Windows)
- EBBR (Device Tree) OSES: SUSE, Ubuntu, Debian, NetBSD, FreeBSD, OpenBSD
- EBBR (ACPI) OSES?



Pi 4 Firmware Task Force

- 10/2016 – **Microsoft** publishes 32-bit UEFI for Pi 2/3, as part of Windows IoT Core.
- 11/2016 – **Ard Biesheuvel** publishes minimal 64-bit UEFI for Pi 3 with TF-A.
- 11/2017 – **Andrei Warkentin** begins work on RaspberryPiPkg for Pi 3, building on Ard's initial work and ms-iot.
- 11/2017 - Integration of a DWC2 UEFI USB host driver (originating with Linaro)
- 12/2017 – Booting Pi 3 SUSE Linux with Device Tree
- 04/2018 – First Pi 3 Windows on Arm boot!
- 10/2018 – First Pi 3 ESXi-Arm boot!
- 02/2019 – RaspberryPiPkg upstreamed to TianoCore edk2-platforms by **Pete Batard**.
- 09/2019 – **VMware + Arm** start community project to add Pi 4 support to Pi 3 code base..
- 10/2019 – ESXi-Arm demoed booting on Pi 4 at the Arm TechCon.
- 12/2019 – **Jeremy Linton** contributes UEFI PCIe RC driver for XHCI support.
- 12/2019 – First ever SBBR boots for Linux and NetBSD.
- 01/2020 – GENET ACPI support, Windows 10/WinPE on Pi 4 (with limitations)

Staging and releases <https://github.com/pftf>

Official blog <https://rpi4-uefi.dev/>

Andrei Warkentin (VMware), Samer El-Haj-Mahmoud (Arm), Pete Batard (Akeo), Ard Biesheuvel (Arm),
Jeremy Linton (Arm), Jared McNeill (NetBSD)

PFTF Github



- <https://github.com/pftf>
- Forks from TianoCore ([edk2](#), [edk2-platform](#), [edk2-non-osi](#))
 - Kept relatively up-to-date with upstream
 - Used as staging branches before submitting patches to TianoCore
- Releases for [RPI4](#) and [RPI3](#), built directly from TianoCore upstream
- Issue and feature request tracking (<https://github.com/pftf/RPI4/issues>)
- [Arm Enterprise ACS](#) test reports: <https://github.com/pftf/acs-reports>
 - ACS test suite pre-built images available from https://github.com/ARM-software/arm-enterprise-acs/tree/release/prebuilt_images

Pi ServerReady Blog



- <https://rpi4-uefi.dev/>
- Background, History, FAQ, Links
- Detailed status kept at: <https://rpi4-uefi.dev/status/>

Making Pi ServerReady

SBBR-compliant (UEFI+ACPI) AArch64 firmware for the Raspberry Pi 4

HOME ABOUT STATUS FAQ LINKS #RPI4-UEFI-DEV ON DISCORD

ABOUT

Community-developed and Arm ServerReady-compatible TianoCore 64-bit UEFI firmware for the iconic Raspberry Pi 4B.

There's no cute project name, given it's just upstream TianoCore support for the platform, but you're welcome to call it `rpi4-uefi` or `rpi-uefi` for the entire [family of upstreamed TianoCore implementations](#), which today encompass Pi 3/3B+ and Pi 4B.

Table of Contents

- 1. Details
- 2. Why
 - 2.1. Low-Cost and High-Volume Developer Platform
 - 2.2. Arm Edge Standardization
 - 2.3. Credible Example
- 3. Project History

Symbol	Meaning
✓	Supported or Complete
⚠	Some work has been done, but has issues
✂	Work is known to be in progress
✗	Not supported or not done at all

Note: Issue tracker is for bugs in a specific release, and tracks only upstream code. It doesn't track feature development. See <https://github.com/pftf/RPi4/issues> for the complete view.

Release	Notes	Firmware Status Report	OS Support Report	Standards Compliance Reports	Issue Tracker	Issues Fixed
v1.5	✓	Same as v1.4	✓	Same as v1.2	✓	✓
v1.4	✓	✓	✓	Same as v1.2	✓	✓
v1.3	✓	Same as v1.2	Same as v1.2	Same as v1.2	✓	✗

Pi 4 Firmware Status



Trusted Firmware

- Using upstream Pi 4 TF-A
- PSCI for CPU boot and power-off/reset.
- PL011 support.
- TBD: SDEI

Devices

- RNG
- miniUART and PL011 serial.
- File-backed NVRAM.
- UEFI framebuffer via Pi mailbox (supporting a range of virtual resolutions).
- DWC2 (USB2) controller on Type-C.
- xHCI (USB3) via custom PCIe controller.
- Arasan SDHCI for SD card.
- GENET (SoC Ethernet) SNP driver. (upstreaming in progress)
- TBD: EMMC2 (new Pi 4 SDHCI controller)

Miscellaneous

- PXE booting
- iSCSI
- RAM disk
- HTTP(s) boot

System Description

- SMBIOS
- EBBR via Device Tree (including overlays)
- Early ACPI support (xHCI, GENET)

Challenges

- PCIe (and thus xHCI) have 3GB DMA limit
- Legacy Pi devices have 1GB DMA limit and translation.
- This doesn't violate SBBR/SBSA, but not done before.

Upstreaming and development in progress.



Exposing PCIe[®] USB XHCI via ACPI

1. Straight-up ACPI MMIO device
 1. Configure PCI in UEFI.
 2. Patch DSDT with USB base address.
 3. Deal with PCIe bus mastering being disabled on UEFI exit.
 2. Expose PCIe
 1. "Almost ECAM" (wrong alignment)
 2. Expose just the USB device config space as ECAM.
- ...in all cases it's still non-cache coherent.
- (2) Violates SBSA Levels > 0: because it's non-cache coherent, ECAM range has wrong alignment and can't scan for B:D:F > 0:0:0.

```
Device (XHC0) {
  Name (_HID, "PNP0D10")
  Name (_UID, 0x0)
  Name (_CCA, 0x0)
  Method (_INI, 0, Serialized) {
    OperationRegion (PCFG, SystemMemory,
                    PCIE_EXT_CFG, 0x1000)
    Field (PCFG, ByteAcc, NoLock, Preserve) {
      Offset (0), VNID, 16, DVID, 16, CMND, 16, STAT, 16,
    }
    Store (0x6, CMND)
  }
  Name (_CRS, ResourceTemplate () {
    QWordMemory (ResourceConsumer, , MinFixed, MaxFixed,
                NonCacheable, ReadWrite, 0x0, 0x600000000,
                0x600000fff, 0x0, 0x1000)
    Interrupt (ResourceConsumer, Level, ActiveHigh,
              Exclusive, ,, ) { 175 }
  })
}
```

XHCI and legacy DMA constraints.



- A major headache.
- The legit way to support is through the `_DMA` object defined for the bus controller enumerating affected devices.
- This can deal with both bus translations and limits.
- Full support in NetBSD thanks to Jared.
- Have 5 lines of code patch for Linux.
- TBD for Windows.

```
Device (SCB0) {  
    Name (_HID, "ACPI0004")  
    Name (_CRS, ResourceTemplate() { ... })  
    Name (_DMA, ResourceTemplate() { ... })  
    Device (XHC0) { ... }  
}
```

6.2.4 `_DMA` (Direct Memory Access)

This optional object returns a byte stream in the same format as a `_CRS` object. `_DMA` is only defined under devices that represent buses. It specifies the ranges the bus controller (bridge) decodes on the child-side of its interface. (This is analogous to the `_CRS` object, which describes the resources that the bus controller decodes on the parent-side of its interface.) Any ranges described in the resources of a `_DMA` object can be used by child devices for DMA or bus master transactions.

The `_DMA` object is only valid if a `_CRS` object is also defined. OSPM must re-evaluate the `_DMA` object after an `_SRS` object has been executed because the `_DMA` ranges resources may change depending on how the bridge has been configured.

If the `_DMA` object is not present for a bus device, the OS assumes that any address placed on a bus by a child device will be decoded either by a device on the bus or by the bus itself, (in other words, all address ranges can be used for DMA).

For example, if a platform implements a PCI bus that cannot access all of physical memory, it has a `_DMA` object under that PCI bus that describes the ranges of physical memory that can be accessed by devices on that bus.

A `_DMA` object is not meant to describe any “map register” hardware that is set up for each DMA transaction. It is meant only to describe the DMA properties of a bus that cannot be changed without reevaluating the `_SRS` method.

We might need to introduce an ACPI boot profile that limits useful RAM to 3GB and hides all legacy devices, at least to deal with OS versions that don't support `_DMA`.

Call to Action



Raspberry Pi 4

- Always looking for more help!
- **#rpi4-uefi-dev** on Arm Developer-Ecosystem Discord server.
- <https://rpi4-uefi.dev> for Discord server link
- Contact us: awarkentin@vmware.com and samer.el-haj-mahmoud@arm.com.



Other platforms

- nVidia Tegra-based platforms (Jetson Nano, Xavier)
- Rockchip RK3399 platforms (Pinebook Pro, Orange Pi, Rock64, Rock960, Nano Pi-M4, etc)

Other firmware

- UEFI does not have to be TianoCore
- U-Boot's UEFI implementation adding ACPI tables in addition to DT
- ...small steps towards both EBBR and SBRR-compliant systems built with U-Boot.

Join Arm standards committee!

- Arm Server Advisory Council (ServerAC)
- Shaping the standards and requirements for servers (and future edge devices)




PINEBOOK PRO

Magnesium Alloy
14" 1080p IPS LCD
64/128GB of eMMC*
USB-C & USB 3.0
802.11ac WiFi
Optional M.2 NVMe Slot
Bluetooth 4.2
4GB of LPDDR4 RAM
Rockchip RK3399
Hexacore A72/A53

Price: \$199

Demo!





ESC (setup), F1 (shell), ENTER (boot)

Raspberry Pi 4 Model B
 BCM2711 (ARM Cortex-A72) 1.50 GHz
 Lanpone b7d593f3e9 on 11/23/2019 4096 MB RAM

Select Language Standard English This is the option one adjusts to change the language for the current system

- ▶ Device Manager
- ▶ Boot Manager
- ▶ Boot Maintenance Manager

Continue
Reset

```

Shell> pci
Shell> pci
  Seg  Bus  Dev  Func
  ---  ---  ---  ---
    00  00  00  00 --> Bridge Device - PCI/PCI bridge
        Vendor 14E4 Device 2711 Prog Interface 0
    00  01  00  00 --> Serial Bus Controllers - USB
        Vendor 1106 Device 3483 Prog Interface 30
Shell> _
        
```

```

Ctrl[52] Primary Console Input Device
Ctrl[53] Primary Console Output Device
Ctrl[54] Primary Standard Error Device
Ctrl[40] VenHw(28A03FF4-12B3-4305-A417-BB1A4F94081E)
Ctrl[41] VenHw(2A46715F-3581-4A55-8E73-2B769AAA30C5)
Ctrl[46] VenHw(EBF0ED7C-0DD1-4787-84F1-F48D537DCACF)
Ctrl[49] PciRoot(0x0)
Ctrl[87] PciRoot(0x0)/Pci(0x0,0x0)
Ctrl[88] eXtensible Host Controller (USB 3.0)
Ctrl[89] PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x0,0x0)
Ctrl[8A] PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x0,0x0)/USB(0x0,0x0)
Ctrl[8B] Generic Usb Keyboard
Ctrl[52] Primary Console Input Device
Ctrl[8C] PciRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x0,0x0)/USB(0x0,0x0)
USB(0x0,0x1)
Ctrl[58] Raspberry Pi Franebuffer
Ctrl[53] Primary Console Output Device
Ctrl[54] Primary Standard Error Device
Press ENTER to continue or 'Q' break:
Ctrl[58] VenHw(CD7CC258-31DB-22E6-9F22-63B0B8EED6B5)
Ctrl[76] VenHw(4B47D616-A8D6-4552-9D44-CCAD2E0F4CF9)
Ctrl[7D] Raspberry Pi USB Host
Ctrl[84] VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)
Ctrl[85] FAT File System
Shell> _
        
```


VMware ESXi 7.0.0 (VMKernel Release Build 00000)

Sony UK Raspberry Pi 4 Model B

ARM Limited Cortex-A72 r0p3
3.9 GiB Memory

procf's loaded successfully.

Windows Setup



Windows

Language to install: English (United States)

Time and currency format: English (United States)

Keyboard or input method: US

Enter your language and other preferences and click "Next" to continue.

© 2019 Microsoft Corporation. All rights reserved. Next



Questions?



Thanks for attending the UEFI 2020 Virtual Plugfest

For more information on UEFI Forum and UEFI Specifications, visit <http://www.uefi.org>

presented by

