Booting UEFI-aware OS on coreboot enabled platform

"In God's Name, Why?"

European Coreboot Conference 2017

Piotr Król and Kamil Wcisło

3MDEB

Intro

Piotr Król

- @pietrushnic
- piotr.krol@3mdeb.com
- linkedin.com/in/krolpiotr
- facebook.com/piotr.krol.756859

Kamil Wcisło

- @mek_xgt
- kamil.wcislo@3mdeb.com
- linkedin.com/in/kamil-wcisło-86a83189
- facebook.com/mek.xgt

"In God's Name, Why?"

- Customers want to boot UEFI-aware systems on platforms where only coreboot is supported (Chromebook, PC Engines)
- To have fun with already developed UEFI tools for system forensics and validation
  - MemTest86 v7 Pro Edition
  - UEFI SCT
  - PI SCT
  - FWTS
  - PeiBackdoor (?)
  - SimpleVisor (?)
- To use UEFI drivers and OpROMs
- UEFI implementation validation eg. `RuntimeServices`, NVRAM variables
- To write your own games, os, hypervisors and compilers :)

Agenda

- Brief history of tianocore payload
- Tianocore payload components
- Steps to take while porting
- PCDs explained
- Trust, but verify
- How to enable serial in UEFI ?
- PCIe issues
- Testing results
- Further steps and conclusion

Brief history

- First attempt made during GSoC 2010
- Initially created for Intel Bay Trail CRB to boot UEFI-aware OS
- First implementation had USB3.0 and SATA/ATA support
- Build instruction came from June 2014, but initial commit came from March 2015
- It has very little activity (2017: 6 patches, 2016: 61 patches)

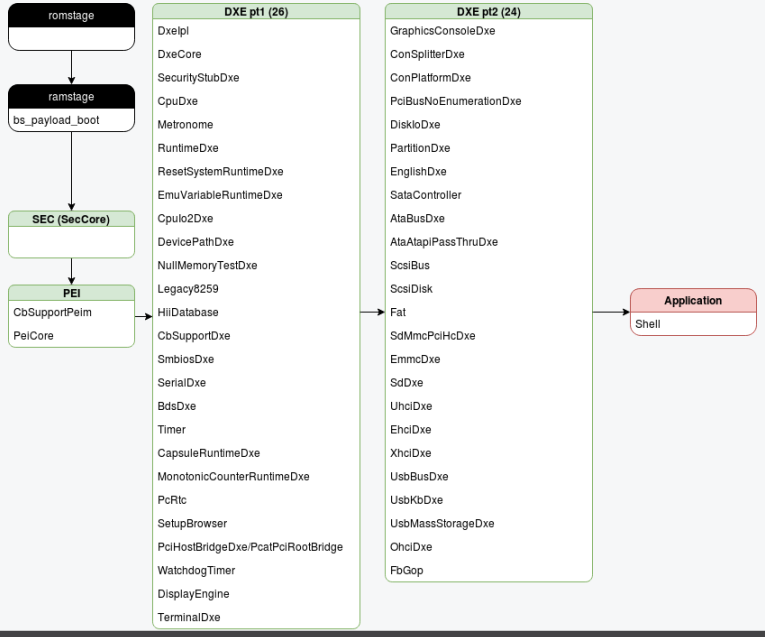
CorebootPayloadPkg

- FbGop
- Library
  - AcpiTimerLib
  - PciHostBridgeLib
  - PlatformBootManagerLib
  - PlatformHookLib
  - ResetSystemLib

CorebootModulePkg

- SecCore
- CbSupportPei
- CbSupportDxe
- SataControllerDxe
- Library
  - BaseSerialPortLib16550
  - CbParseLib
  - CbPlatformSupportLibNull

Tianocore payload booting



Boot time for PC Engines apu2

- romstage: 2.217s
- ramstage: 2.324s
- SEC: 1ms
- PEI: 217ms
- DXE: 9.624s

Steps to take while porting

- prepare development environment and automate debugging process
- try what is already there
- understand what information are passed in coreboot table
- learn what PCDs you have available and how to use those
- target UEFI Shell booting
- familiarize yourself with `ConIn`, `ConOut` and `ErrOut` concept

PCDs explained

- Platform Configuration Database
  - build-time configuration options
  - run-time configuration options
- PcdFeatureFlag - boolean, fixed at build time
- PcdFixedAtBuild - constant, fixed at build time
- PcdPatchableInModule - constant, can be modified at runtime
- PcdDynamic{Default, ExDefault} - can be modified, set at boot from default stored in flash
- PcdDynamic{Hii, ExHii} - can be modified, persistent, stored in flash
- PcdDynamic{Vpd, ExVpd} - can be modified, persistent, stored in flash

Problems we faced

- `PcRtcEntry` assert
- Lack of serial console in/out
- `CbSupportDxe` assert
- PCI/PCIe enumeration issues

CbSupportDxe assert

- failed to add memory space for `LAPIC` (`0xFEE00000`)
- access denied when trying to allocate pool
- workaround: ignore :)
- solution: TBD

PcRtcEntry assert

- RTC cannot be easily omitted since this is against PI spec, and UEFI will not allow further boot
- problem was with Valid RAM and Time (`VRT`) bit in RTC D register
- it happen to be bug in datasheet (or silicon), since `VRT` bit is read-only everywhere but not on `GX-412TC` apu2 SoC

Choosing your Shell

- SHELL_TYPE:
  - BUILD_SHELL- build from source code, useful for debugging
  - FULL_BIN - default, legacy, binary distributed with edk2, all commands
  - MIN_BIN - binary distributed with edk2, minimum commands
  - NONE
  - UEFI_BIN - binary distributed with edk2, all commands
- Shell is defined in platform DSC file:

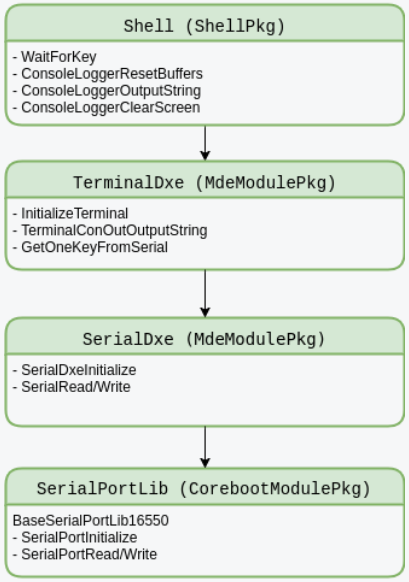
```
#
# Shell options: [BUILD_SHELL, FULL_BIN, MIN_BIN, NONE, UEFI]
#
DEFINE SHELL_TYPE              = BUILD_SHELL
```

ConIn, ConOut and ErrOut

- apu2 do not have video output - serial is crucial
- problem: lack of serial logs after UEFI Shell take over
- `ConIn`, `ConOut` and `ErrOut`
  - global variables defined by UEFI spec
  - non-volatile
  - available during boot time and runtime
- in early boot stage `SerialPortLib` was used that's why logs in PEI and DXE were visible
- what have to be provided is UEFI device path for serial device:

```
VenHw(D3987D4B-971A-435F-8CAF-4967EB627241)/Uart(115200,8,N,1)
```

Serial port stack



Add device path for serial

```c
#define DP_NODE_LEN(Type) { (UINT8)sizeof (Type), (UINT8)(sizeof (Type) >> 8)  }

#pragma pack (1)
typedef struct {
  VENDOR_DEVICE_PATH          SerialDxe;
  UART_DEVICE_PATH            Uart;
  VENDOR_DEFINED_DEVICE_PATH TermType;
  EFI_DEVICE_PATH_PROTOCOL    End;

} PLATFORM_SERIAL_CONSOLE;
#pragma pack ()

#define SERIAL_DXE_FILE_GUID { \
          0xD3987D4B, 0x971A, 0x435F, \
          { 0x8C, 0xAF, 0x49, 0x67, 0xEB, 0x62, 0x72, 0x41  } \
           }

STATIC PLATFORM_SERIAL_CONSOLE mSerialConsole = {
  //
  // VENDOR_DEVICE_PATH SerialDxe
  //
  {
    { HARDWARE_DEVICE_PATH, HW_VENDOR_DP, DP_NODE_LEN (VENDOR_DEVICE_PATH)  },
    SERIAL_DXE_FILE_GUID
   },

  //
  // UART_DEVICE_PATH Uart
  //
  {
    { MESSAGING_DEVICE_PATH, MSG_UART_DP, DP_NODE_LEN (UART_DEVICE_PATH)  },
    0, // Reserved
    0, // BaudRate
    0, // DataBits
    0, // Parity
    0  // StopBits
   },
 }
```

```c
  //
  // VENDOR_DEFINED_DEVICE_PATH TermType
  //
  {
    {
      MESSAGING_DEVICE_PATH, MSG_VENDOR_DP,
      DP_NODE_LEN (VENDOR_DEFINED_DEVICE_PATH)
    }
    //
    // Guid to be filled in dynamically
    //
  },

  //
  // EFI_DEVICE_PATH_PROTOCOL End
  //
  {
    END_DEVICE_PATH_TYPE, END_ENTIRE_DEVICE_PATH_SUBTYPE,
    DP_NODE_LEN (EFI_DEVICE_PATH_PROTOCOL)
  }
};
```

Set ConIn, ConOut and ErrOut

```
mSerialConsole.Uart.BaudRate = PcdGet64 (PcdUartDefaultBaudRate);
mSerialConsole.Uart.DataBits = PcdGet8 (PcdUartDefaultDataBits);
mSerialConsole.Uart.Parity = PcdGet8 (PcdUartDefaultParity);
mSerialConsole.Uart.StopBits = PcdGet8 (PcdUartDefaultStopBits);
//
// Add the hardcoded serial console device path to ConIn, ConOut, ErrOut.
//
CopyGuid (&mSerialConsole.TermType.Guid,
  PcdGetPtr (PcdTerminalTypeGuidBuffer));
EfiBootManagerUpdateConsoleVariable (ConIn,
  (EFI_DEVICE_PATH_PROTOCOL *)&mSerialConsole, NULL);
EfiBootManagerUpdateConsoleVariable (ConOut,
  (EFI_DEVICE_PATH_PROTOCOL *)&mSerialConsole, NULL);
EfiBootManagerUpdateConsoleVariable (ErrOut,
  (EFI_DEVICE_PATH_PROTOCOL *)&mSerialConsole, NULL);
```

PcatPciRootBridge driver from DuetPkg

- access PCI memory space through IO (0xcf8/0xcfc)
- no enumeration performed
- adds dummy root bridge
- requires changes in FDF and DSC files or applying `01_CorebootPayloadPkg_pcinoenum.patch`

```
diff --git a/CorebootPayloadPkg/CorebootPayloadPkg.fdf b/CorebootPayloadPkg/CorebootPayloadPkg.fdf
index 303e626842..a39e3999ba 100644
--- a/CorebootPayloadPkg/CorebootPayloadPkg.fdf
+++ b/CorebootPayloadPkg/CorebootPayloadPkg.fdf
@@ -124,8 +124,8 @@ INF MdeModulePkg/Universal/SmbiosDxe/SmbiosDxe.inf
 #
 # PCI Support
 #
-INF MdeModulePkg/Bus/Pci/PciBusDxe/PciBusDxe.inf
-INF MdeModulePkg/Bus/Pci/PciHostBridgeDxe/PciHostBridgeDxe.inf
+INF DuetPkg/PciRootBridgeNoEnumerationDxe/PciRootBridgeNoEnumeration.inf
+INF DuetPkg/PciBusNoEnumerationDxe/PciBusNoEnumeration.inf
(...)
```

MdeModulePkg PciHostBridgeDxe

- forcing access through IO by setting `PCIE_BASE` to 0 in CorebootPayloadPkg DSC
- leave `PCIE_BASE` as is (`0xE0000000`) - seem to be correct only for Intel
- set `PCIE_CASE` to value reported by coreboot

```
...
BS: BS_DEV_ENUMERATE times (us): entry 0 run 133877 exit 0
Allocating resources...
Reading resources...
fx_devs = 0x1
Adding PCIe enhanced config space BAR 0xf8000000-0xfc000000.
Done reading resources.
Setting resources...
...
```

```
#
# PCI options
#
DEFINE PCIE_BASE                        = 0xF8000000
```

MdeModulePkg PciHostBridgeDxe - assert

```
InitRootBridge: populated root bus 255, with room for 0 subordinate bus(es)
ScanForRootBridges:587 RootBridges: -813834216
RootBridge: PciRoot(0x0)
  Support/Attr: 10063 / 10063
    DmaAbove4G: No
NoExtConfSpace: No
     AllocAttr: 0 ()
           Bus: 0 - 3
            Io: 1000 - 4FFF
           Mem: F7A00000 - F7FFFFFF
    MemAbove4G: FFFFFFFFFFFFFFFF - 0
          PMem: FFFFFFFFFFFFFFFF - 0
   PMemAbove4G: FFFFFFFFFFFFFFFF - 0
Split - 0xCF7E5000
RootBridge: PciRoot(0x6B)
  Support/Attr: 0 / 0
    DmaAbove4G: No
NoExtConfSpace: No
     AllocAttr: 0 ()
           Bus: 6B - 6B
            Io: FFFFFFFFFFFFFFFF - FFFFFFFFFFFFFFFF
           Mem: FFFFFFFFFFFFFFFF - FFFFFFFFFFFFFFFF
    MemAbove4G: FFFFFFFFFFFFFFFF - 0
          PMem: FFFFFFFFFFFFFFFF - 0
   PMemAbove4G: FFFFFFFFFFFFFFFF - 0

DXE_ASSERT!: .../Pci/PciHostBridgeDxe/PciRootBridgeIo.c (100): Bridge->Mem.Limit < 0x0000000100000000ULL
```

What we tried ?

- UEFI-aware Debian 9.2
- UEFI-aware Arch
- Memtest86 Pro
- Python

Debian 9.2

```
root@apu2:~# efibootmgr -v
BootCurrent: 0003
Timeout: 3 seconds
BootOrder: 0000,0001,0002,0003
Boot0000* UiApp MemoryMapped(11,0x830000,0xc0ffff)/FvFile(462caa21-7614-4503-836e-8ab6f4662331)
Boot0001* UEFI Kingston DataTraveler 3.0 0015F284C2ADB031F955D922       PciRoot(0x0)/Pci(0x10,0x0)/USB(1,0)N.....YM....R,Y.
Boot0002* UEFI SanDisk SSD i110 16GB 150901101579       PciRoot(0x0)/Pci(0x11,0x0)/Ata(0,0,0)N.....YM....R,Y.
Boot0003* UEFI Shell    MemoryMapped(11,0x830000,0xc0ffff)/FvFile(7c04a583-9e3e-4f1c-ad65-e05268d0b4d1)
root@apu2:~# dmidecode -t system
# dmidecode 3.0
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x0001, DMI type 1, 27 bytes
System Information
        Manufacturer: PC Engines
        Product Name: PC Engines apu2
        Version: 1.0
        Serial Number: 1069064
        UUID: Not Settable
        Wake-up Type: Reserved
        SKU Number: 4 GB
        Family: Not Specified

Handle 0x0005, DMI type 32, 11 bytes
System Boot Information
        Status: No errors detected

root@apu2:~# uname -a
Linux apu2 4.9.0-4-amd64 #1 SMP Debian 4.9.51-1 (2017-09-28) x86_64 GNU/Linux
root@apu2:~#
```

Arch

```
Arch Linux 4.13.3-1-ARCH (ttyS0)

archiso login: root
Last login: Fri Oct 20 01:01:24 on tty1
root@archiso ~ # efibootmgr -v
BootCurrent: 0001
Timeout: 3 seconds
BootOrder: 0000,0001,0002,0003
Boot0000* UiApp MemoryMapped(11,0x830000,0xc0ffff)/FvFile(462caa21-7614-4503-836e-8ab6f4662331)
Boot0001* UEFI Wilk GOODRAM 16GB 071043349515C551       PciRoot(0x0)/Pci(0x10,0x0)/USB(1,0)N.....YM....R,Y.
Boot0002* UEFI SanDisk SSD i110 16GB 150901101579       PciRoot(0x0)/Pci(0x11,0x0)/Ata(0,0,0)N.....YM....R,Y.
Boot0003* UEFI Shell    MemoryMapped(11,0x830000,0xc0ffff)/FvFile(7c04a583-9e3e-4f1c-ad65-e05268d0b4d1)
root@archiso ~ #
```

Memtest86 Pro

```
PassMark MemTest86 V7.4 Pro   AMD GX-412TC SOC
Clk/Temp : 782.9 MHz / 97C  | Pass     0%
L1 Cache :   64K  11.50 GB/s| Test    76% ########################
L2 Cache : 2048K   5107 MB/s| Test 13 [Hammer test] - Hammering rows
L3 Cache :  N/A             | Address   : 0xC0C10000 - 0xCBDB0000
Memory   : 4077M   1455 MB/s| Pattern   : 0x0C4AF4D8
RAM Info : N/A
-------------------------------------------------------------------------------
CPU:   0123                           | CPUs Found:       4
State: -WWW                           | CPUs Started:     4 CPUs Active:  4
-------------------------------------------------------------------------------
Time:     0:16:47   AddrMode: 64-bit     Pass: 3 / 4           Errors:    0

 [ECC Inject] Injecting ECC error for AMD Steppe Eagle
 [ECC Inject] Injecting ECC error for AMD Steppe Eagle
>[ECC Inject] Injecting ECC error for AMD Steppe Eagle









(ESC)/(c)onfiguration
```

ECC for apu2 considerations

- Memtest86 Pro output:

```
(...)
find_mem_controller - AMD Steppe Eagle (1022:1582) at 0-24-2
find_mem_controller - AMD Steppe Eagle ECC mode: detect: yes, correct: yes, scrub: no, chipkill: no
ECC polling enabled
(...)
```

- ECC testing on apu2 was prevented by bug in AGESA, which was fixed in MullinsPI 1.0.0.A
- Unfortunately apu2 platform cannot use that because fix introduced with MullinsPI 1.0.0.4, that aims to solve Windows 7 graphics driver hang
- Problem happen after adding 2 PSP functions `PspMboxBiosCmdDramInfo` and `PspMboxBiosCmdDramInfo`

Python

```
FSOpen: Open '\Efi\StdLib\lib\python.27\stat.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\stat.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\genericpath.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\genericpath.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\warnings.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\warnings.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\linecache.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\linecache.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\types.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\types.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\copy_reg.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\copy_reg.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\traceback.py' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\traceback.pyc' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\site-packages' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\site-packages' Success
FSOpen: Open '\Efi\StdLib\lib\python.27' Success
FSOpen: Open '\Efi\StdLib\lib\python.27\site-packages' Success
Python 2.7.2 (default, Jul 24 2017, 12:02:08) [C] on uefi
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print sys.platform
uefi
>>> 
```

Possible further steps

- fix CHIPSEC to have top forensics tool support
- look how to improve USB drivers in SeaBIOS since those in Tianocore seems to be more stable
- try to emulate Secure Boot
- try to execute UEFI OpROMs
- try to call AGESA API from UEFI Application

Conclusions

- there are useful tools available just for UEFI
- enabling coreboot platform to boot Tianocore payload is not hard task, there seem to be problems remaining in Tianocore that have to be fixed
- it may give ability to enable devices not supported by coreboot yet
- it can help debugging alternative implementation eg. comparing behavior between SeaBIOS and Tianocore payload

3MDEB

Thank you