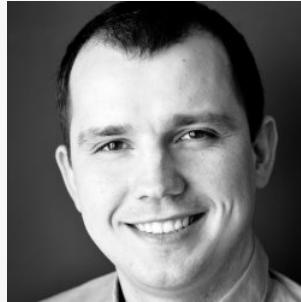


coreboot on POWER9

OpenPOWER Summit 2021

Piotr Król





Piotr Król
3mdeb Founder and CEO

- OSF and OSHW promoter
- interested in Trusted Computing
- Conference speaker and organizer
- TrenchBoot Steering Committee Core Member
- 13yrs in business
- 7yrs in Open Source Firmware and Trusted Computing integration
- C-level positions in





- coreboot licensed service providers since 2016 and leadership participants
- UEFI Adopters since 2018
- Yocto Participants and Embedded Linux experts since 2019
- Official consultants for Linux Foundation fwupd/LVFS project since 2020
- IBM OpenPOWER Foundation members since 2020

- Why are we porting coreboot to POWER9?
- Our workflow
- Work done so far
- Problems faced
- Plans for the future








- Thierry Laurion, Founder Insurgo Technologies Libres/Open



- Krystian Hebel, Firmware Engineer @ 3mdeb, as main contributor to POWER9 coreboot port and this presentation
- Igor Bagnucki, Junior Embedded Firmware Developer @ 3mdeb
- Maciej Pijanowski, Engineering Manager @ 3mdeb

- Hostboot is **complex**
 - full-blown OS
 - virtual memory, paging, userspace
 - interrupts
 - swap to flash memory (!) before RAM is trained
- Boot time:
 - servers rarely do cold boot so it shouldn't matter
 - unless you are hypescaler using POWER hardware
 - POWER9 is used also in workstations
- Code size:
 - smaller code means faster loading, faster validation, no swapping
 - more space for other components OR smaller, cheaper flash
- For the benefit of coreboot community:
 - new architecture supported
 - code checked against endianness issues
 - new, open platform
- coreboot is well recognized firmware platform with sizable community

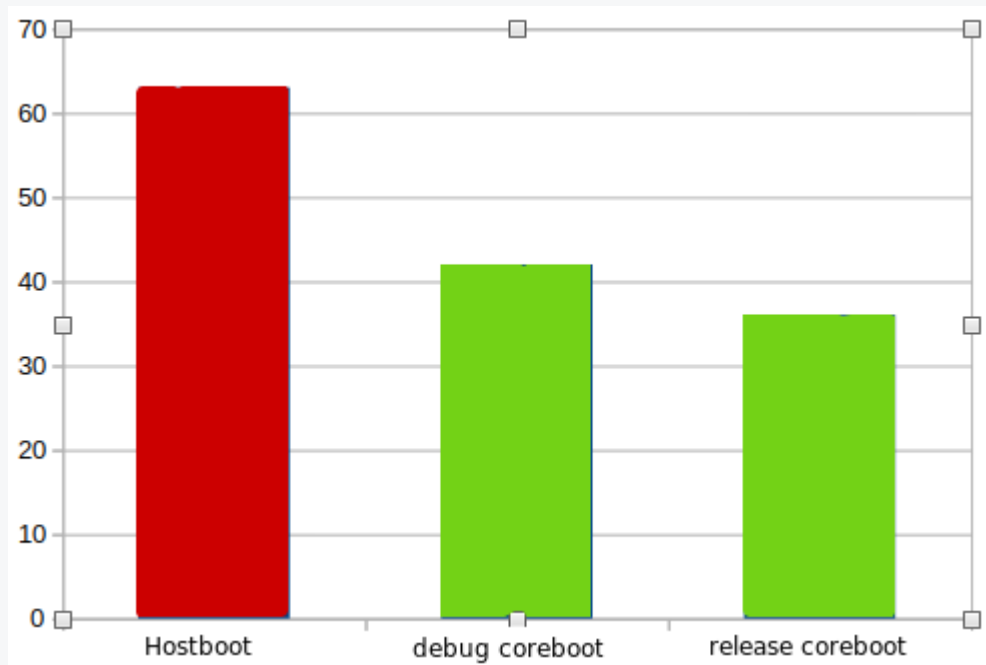
Nazwa	Rozmiar	Typ	Modyfikacja
 hostboot.bin	569,4 kB	Nieznany typ	18 października 2021, 11:02
 hostboot_bootloader.bin	21,5 kB	Nieznany typ	18 października 2021, 11:02
 hostboot_extended.bin	12,7 MB	Nieznany typ	18 października 2021, 11:02
 hostboot_runtime.bin	4,6 MB	Nieznany typ	18 października 2021, 11:02
 hostboot_securerom.bin	9,1 kB	Nieznany typ	18 października 2021, 11:02

```
$ ls build -l | grep signed.ecc
-rw-r--r--  1 root root   28674 paź 26 15:00 bootblock.signed.ecc
-rw-r--r--  1 root root 2363904 paź 26 15:00 coreboot.rom.signed.ecc

(28674+2363904) * (8/9) = 2126736 bytes ~ 2.02 MiB
(8/9) - ECC multiplier
```

- Hostboot files weigh more than 15 MiB
 - Skiboot is placed on a different partition
- coreboot files weigh around 2 MiB (compressed would fit in HBB)
 - Includes uncompressed Skiboot as a payload (almost 1.5 MiB)
 - Still has ~315 KiB free space in CBFS
 - Whole CBFS in one PNOR partition

- Release Hostboot from CPU start to Skiroot: 63s
- Debug coreboot from CPU start to Skiroot: 42s (33% faster)
- With no output on serial coreboot can go down to 36s (42% faster)



- The code is public but not quite open
 - Some are machine-generated from so-called "init files"
 - Are init files public?
 - We couldn't find much about it (including spec or compiler).
- Huge codebase (almost 1.7 milion lines of code)
 - Some files with thousands of lines, some empty except for license
- Code audit (aka "Reverse Engineering") of Hostboot is difficult
 - "What author had in mind" problem

```
(...)  
  
target_type 0 TARGET_TYPE_PROC_CHIP;  
target_type 1 TARGET_TYPE_SYSTEM;  
  
ispy INT.INT_PC.INT_PC_AIB_TX_CRD_RSD_CRD_VPC_LD_RMT [when=S && ATTR_CHIP_EC_FEATURE_P9N_INT_DD10] {  
    spyv;  
    0b00;  
}  
  
ispy INT.INT_VC.INT_VC_EQC_CONFIG_PAGE_OFFSET_CFG [when=S] {  
    spyv;  
    0x5BBF;  
}
```

<https://github.com/3mdeb/talos-hostboot/blob/07-25-2019/src/import/chips/p9/initfiles/p9a.int.scom.initfile>

- Machine-generated
- Barely-human-readable

```
if (((l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_48625)
    || (l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_68625)))
{
    l_scom_buffer.insert<56, 4, 60, uint64_t>(literal_0x1 );
}
else if (((l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_44500)
    || (l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_64500)))
{
    l_scom_buffer.insert<56, 4, 60, uint64_t>(literal_0x4 );
}
else if (((l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_55500)
    || (l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_75500)))
{
    l_scom_buffer.insert<56, 4, 60, uint64_t>(literal_0xB );
}
else if (((l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_51375)
    || (l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_71375)))
{
    l_scom_buffer.insert<56, 4, 60, uint64_t>(literal_0x8 );
}
else if (((l_TGT0_ATTR_CEN_EFF_RD_VREF[literal_1] == literal_45875)
```

https://github.com/open-power/hostboot/blob/master/src/import/chips/centaur/procedures/hwp/initfiles/centaur_ddrphy_scom.C#L1901

That's not how you round up numbers

```
// Round up  
l_vdd = (l_vdd << 1) + 1;  
l_vdd = l_vdd >> 1;
```

- Introduced in massive code refactor of 5k SLOC
- It get through 4 reviewers and 6 testing systems
- Huge code base is hard to maintain

https://github.com/open-power/hostboot/blob/master/src/import/chips/p9/procedures/hwp/pm/p9_pstate_parameter_block.C#L1803

This floor don't seem quite right

```
double internal_floor(double x)
{
    if(x >= 0)
    {
        return (int)x;
    }

    return (int)(x - 0.9999999999999999);
}
```

https://github.com/open-power/hostboot/blob/master/src/import/chips/p9/procedures/hwp/pm/p9_pstate_parameter_block.H#L522

Can you spot the difference between statements in both blocks?

```
if ((l_TGT2_ATTR_EFF_DIMM_TYPE[l_def_PORT_INDEX][literal_0] == literal_1))
{
    l_scom_buffer.insert<30, 6, 58, uint64_t>(((l_TGT2_ATTR_EFF_DRAM_CWL[l_def_PORT_INDEX] +
        l_TGT2_ATTR_MSS_EFF_DPHY_WLO[l_def_PORT_INDEX]) - literal_8) );
}
else if ((l_TGT2_ATTR_EFF_DIMM_TYPE[l_def_PORT_INDEX][literal_0] != literal_1))
{
    l_scom_buffer.insert<30, 6, 58, uint64_t>(((l_TGT2_ATTR_EFF_DRAM_CWL[l_def_PORT_INDEX] +
        l_TGT2_ATTR_MSS_EFF_DPHY_WLO[l_def_PORT_INDEX]) - literal_8) );
}
```

https://github.com/3mdeb/talos-hostboot/blob/07-25-2019/src/import/chips/p9/procedures/hwp/initfiles/p9n_mca_scom.C#L494

- One setter has form (id, val)
- Similar one in the next line has (val, id)

```
io_cmeRings.setRingOffset( pRingPayload, io_cmeRings.getCommonRingId( ringIndex ));  
io_cmeRings.setRingSize( io_cmeRings.getCommonRingId( ringIndex ), ringSize );
```

https://github.com/open-power/hostboot/blob/master/src/import/chips/p9/procedures/hwp/pm/p9_hcode_image_build.C#L294

- Context of code is important, so basically one part can be analyzed effectively by only one person
 - Effective knowledge sharing for so complex code is challenging
- A lot of time was spent on understanding basics like FAPI buffers
 - Existing documentation shows very basic examples, but code uses much more complicated methods
 - Mix of templates and methods of class, either one would suffice
- Porting without proper documentation is prone to error, so please test our code before using it in production
- What we need is comprehensive Hostboot documentation
 - in source code
 - as standalone document describing high level concepts
- We documented as much as we could in:
<https://github.com/3mdeb/openpower-coreboot-docs>
 - we recommend devnotes/porting.md
 - in case of mistakes - PRs are welcomed

- SCOM registers are the very basic system for any operations on POWER9
- SCOM base address is set by SBE
- Lot of guesswork was needed to figure out how to use SCOM registers
- It would not be possible without:
 - OpenPOWER Firmware mailing list members support
 - QEMU POWER9 implementation by Cédric Le Goater

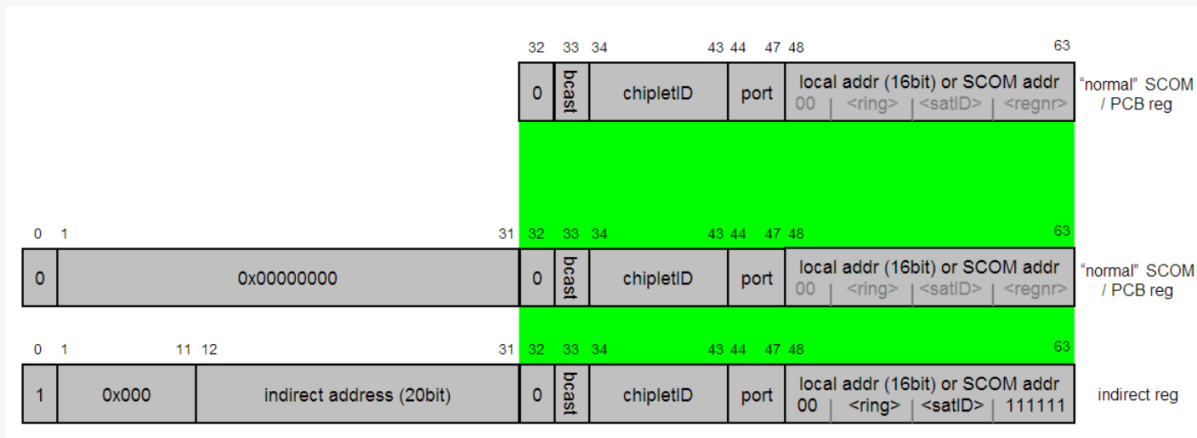
https://github.com/open-power/sbe/blob/master/src/import/chips/p9/procedures/hwp/nest/p9_sbe_scominit.C#L177

- 11264 documented registers
- 504 of registers used by Hostboot were undocumented
- The documentation is partially inaccurate, as we found several registers that were used differently than documented
- It is poorly documented, how to access SCOM registers
 - The graphics show example of confusing SCOM description
 - with meaningful name, purpose and definition
 - even with all bits reserved

Register Name			CERR Message 0 Register
Mnemonic			NPU.STCK1.CS.CTL.MISC.CERR_MESSAGE0
Address			0000000005011198 (SCOM)
Description			Error message/capture register.
Bits	SCOM	Field Mnemonic: Description	
0:63	RWX_WCLRREG	CERR_MESSAGE_BITS0: Reserved.	

<https://ibm.ent.box.com/s/ddcdl3g0otdzviahkfe3jjh2oy5p3mt>

- Documentation has only one note about indirect registers
 - Only a graphic presenting how direct and indirect addresses differ
 - Not enough to create working implementation
- Even with correct implementation, indirect SCOM couldn't be accessed initially
 - Needs earlier isteps to work



<https://ibm.ent.box.com/s/ddcdl3g0otdzviahkfe3jjh2oy5p3mt>

- Probably the most time spent on analysis was here
 - but a large chunk of that was for understanding SCOM, targeting, istep dispatching, Hostboot directory structure etc.
- Some small issues with MEMD format
 - one byte missed due to wrongfully hardcoded value and the rest of the fields followed
 - a lot of time wasted on debugging
 - perfect opportunity to learn how to use pdbg
- **Lesson learned:** unless you need tables with timing parameters for speed bins, you don't need to buy DDR specification from JEDEC - similar documents can be found on DRAM vendors' sites for free

- Quite large subsystem:
 - OCC
 - 4x GPE (SGPE, PGPE and two generic ones)
 - common SRAM (conflicting documentation/comments about its size)
 - cooperates with CMEs (one per L3 cache, 12 in total)
- Each component has a separate log area with different format and must be debugged independently
 - no existing documentation at all
- Required to enable power management in the OS and starting other cores than bootstrapping one
- Most of the work is done by those engines, host "only" orchestrates start of OCC and two special GPEs by:
 - building HOMER (see next slide) which was poorly documented at that time
 - pushing enough code into SRAM and/or OCC's registers to start a bootloader on it

HOMER - Hardware Offload Microcode Engine Region. Has few hundreds fields that had to be properly set with little documentation. Divided into 4 subregions, 1MB each:

- OCC host area
 - main OCC code is loaded from here by a bootloader
- QPMR - Quad Power Management Region
 - SGPE code
 - cache SCOM region - restores SCOM registers on quad (L3/L2) power-up after deeper sleep states
- CPMR - Core Power Management Region
 - CME code
 - core self-restore (GPRs and SPRs) and SCOM regions
- PPMR - Pstate Power Management Region
 - PGPE code
 - global, local(*) and OCC Pstate Parameter Blocks ({G,L,O}PPB)

- HOMER and SCOM details were provided as result of mailing list discussion
 - don't be afraid to ask, if you can't find something
 - expect waiting, getting through corporate process
- Hostboot seem to use 'Code is documentation' approach
 - we would like to change that with 3mdeb Dasharo product line
- Beware code vs code comment mismatch
- Read Skiboot, QEMU or Linux source it gives more than documentation and Hostboot's code combined

39	ROX	Reserved field.
40	ROX	Reserved field.
41	ROX	Reserved field.
42	ROX	Reserved field.
43	ROX	Reserved field.
44	ROX	Reserved field.
45	ROX	Reserved field.
46:63	RO	constant = 0b00000000000000000000

<https://ibm.ent.box.com/s/ddcdl3g0otdzviahkfe3jjh2oy5p3mt> page 608

- Hostboot passes HDAT to Skiboot, coreboot uses FDT instead
- coreboot always reads MVPD from EEPROM, no cache in PNOR (yet?)
 - Slightly slows down boot process
 - Always has fresh data - is Hostboot's validation just by checking for serial number match trustworthy enough?
- Less passes of RAM pattern testing - faster boot, useful for Talos II used as a workstation
- Petitboot can be substituted with Heads
- Order of some steps was changed
 - Most of early isteps in Hostboot are for data acquisition. coreboot has no concept similar to attributes so data is obtained as needed
 - Istep 16.2 (start secondary cores) was moved to just before jumping to payload. This hack removes the need to implement SRESET handler, cores jump to initial handler in Skiboot's image. Auxiliary threads don't have address of FDT in %r3, it is imperative for boot thread to jump into the payload before any other thread wakes up. Needs testing for bigger number of cores.

```

ending istep 14.3
starting istep 14.5
ending istep 14.5
0xF000F = 223d104900008040
CBMEM:
IMD: root @ 0xffeff000 254 entries.
IMD: root @ 0xffefec00 62 entries.
FMAP: area COREBOOT found @ 20200 (1965568 bytes)
FFS header at 0x80060300ffff7000
PNOR base at 0x80060300fc000000
HBI partition has ECC
HBI is in 0x00426200 through 0x0175f
CBFS: Found 'fallback/ramstage' @0xd0000000 0xd3c0 in mcache @0x00031080
BS: romstage times (exec / console): 0 (unknown) / 39 ms

coreboot-4.12-8269-gce64c09122 Fri Oct 22 09:49:19 UTC 2021 ramstage starting (log level: 7)...
Enumerating buses...
Root Device scanning...
DD23, boot core: 1
FFS header at 0x80060300ffff7000
PNOR base at 0x80060300fc000000
HCODE partition has ECC
HCODE is in 0x01a82200 through 0x0

```

<https://asciinema.org/a/lqMvLFEPfjR14Eqg8Z5Wwlm77>

In no particular order:

- Code cleanup, make coreboot less verbose
- Organize documentation
- Unhardcode device tree - in progress
 - /cpus and /memory@(...) are already created by code
 - /xscom@(...)/mcbist@(...) subtrees removed, nothing seems to be using them
 - SPD under I2C were also removed, consequently Linux doesn't load drivers for them automatically
 - "cosmetic" stuff left - model, system-id, VPDs etc.
- Support for systems with 2 CPUs installed
- Additional testing
- Testing
- More testing on different configurations
- Any volunteers for testing?

- First stage of coreboot called bootblock can be put in SEEPROM
 - validated/measured boot
- coreboot on different components (SBE? OCC?)
 - would require another compiler for PPE
 - may not fit coreboot's design of separating FW initialization code (coreboot) from runtime (payload)
- Memory training values cached between boots
 - possibly faster training, but is it worth the effort?
- `ibm,firmware-versions/coreboot` and entry in `VERSIONS` in PNOR
 - possible components names are hardcoded in Skiboot, would require changes in its code
 - host should be able to differentiate between different flavors of firmware, e.g. for automatic updates
 - would such modification be accepted for upstream Skiboot?

- Community bi-weekly calls
 - every odd weeks (WW43, WW45, ...) at 2PM UTC
 - <https://meet.3mdeb.com/OpenPOWERCommunityCall>
 - Minutes: <https://pad.riseup.net/p/Mf9UCB4ui4YPH4CFhGeD-keep>
- Open Source FirmWare (OSFW) Slack #openpower channel
 - self-registration: <https://slack.osfw.dev/>
- Repositories
 - <https://github.com/3mdeb/openpower-coreboot-docs>
 - soon we will add Dasharo subsection at <https://docs.dasharo.com/>

- If yes, and you want to support us we would be glad to provide coreboot and Hostboot related support
 - if no, please let us know what we can do better
- What we can do:
 - Hostboot documentation (improvements, maintenance)
 - Hostboot and coreboot firmware validation
 - Hostboot and coreboot training for new beginners
 - Hostboot and coreboot code audit
 - Hostboot and coreboot platform bring-up and custom features development
 - Try to convince you why to switch to coreboot and how much resources you can save on that :)

Q&A