P10 IPL Flow

V1.03 (06/21/22)

Copyright and Disclaimer

IBM Systems and Technology Group 2070 Route 52, Bldg. 330 Hopewell Junction, NY 12533-6351
The IBM home page can be found at ibm.com®.

1 Introduction

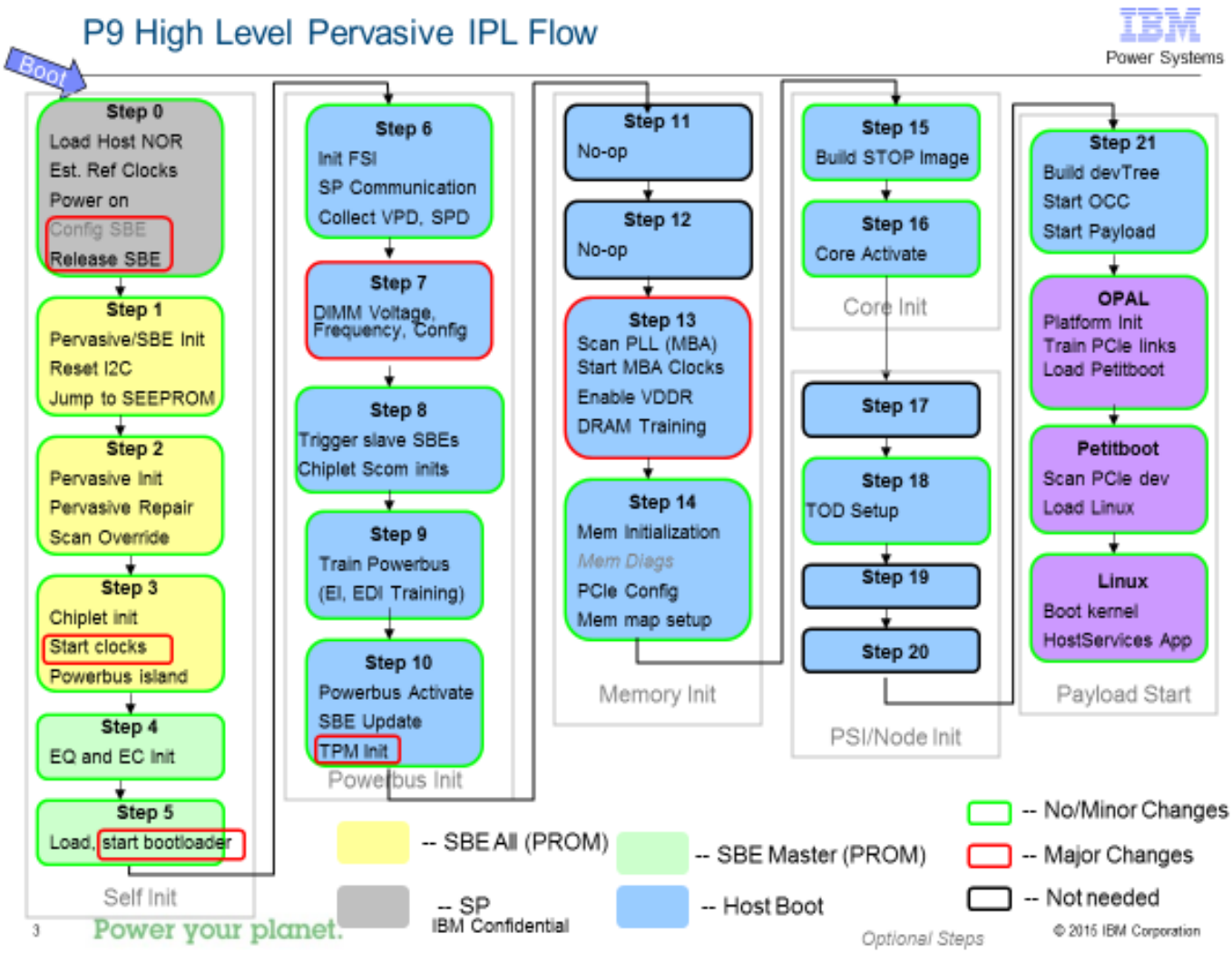
1.1 Description

This document will describe the high-level IPL flow for the Power servers based on P10 chips. It is not intended to contain all low-level details, but instead is designed to illustrate the relationships between various low-level procedures. Complete details can be found in the reference documents listed at the end of this document.
This document covers both the hardware and firmware flow required to boot a system to the hypervisor state. This includes full energy management capability and enough resources to boot partitions. Historical procedure names will continue to be used to identify distinct function boundaries, but actual format may vary.
Note on accuracy: All details down to the procedure call are correct and this document is considered an authoritative reference. Any details within an individual procedure are informational only, the final authority lies within the procedures themselves and their associated reference documents.

This version of the document will cover all POWER10 systems. Please note that this document has a lot of low level details on the initialization of the POWER processor and it's memory subsystem. There are a lot of terms and details in here that are very IBM and POWER centric. We attempted to put as much in the glossary as possible but please feel free to use the mailing list for any questions.
Throughout the document you will see references to a “SP”. This stands for a service processor and when used it's applicable to either the FSP (Flexible Service Processor – used within IBM POWER based servers) or the BMC (the OpenPower service processor). For the most part we've tried to remove FSP specific references for the OpenPower work but some may still remain for reference in here.
Reading over the Hostboot Programmers guide (same document repo) is recommended prior to reading this document.
This version of the document will cover the following systems:

- IBM FSP based system
- IBM Enterprise BMC / OpenBMC based system

The following diagram gives a high level overview of the IPL flow. The minute details are explained in the rest of the document.



1.2 Terminology

- **BMC:** Baseboard Management Controller. Industry standard service processor
- **Explorer:** Off Chip Memory Buffer (OCMB) chip which optimizes memory bandwidth and usage
- **Cronus:** Lab debug tool used to control and debug the IPL
- **eBMC:** Enterprise BMC. Extensions to OpenBMC for IBM enterprise servers
- **FSP:** Flexible Service Processor. IBM designed service processor
- **IPL** : Inital Program Load = Boot process. Covers time between power on and running they hypervisor
- **HCODE: Hardware code** – Refers to the code the EX (EC, EQ) units between the running (full power), stop states (core power saved, cache active), and core and caches powered off states
- **Hostboot**: FW that initializes the memory and powerbus
- **HB Runtime Services (HBRT):** Portion of Host Boot that remains resident during hypervisor execution and provides PRD functionality for CE

- **istep** : IPL Step defined by ecmd interface
- **Normal-mode** : IPL that includes minimal diagnostics, focused on functional requirements only
- **Maintenance-mode** : IPL that includes extensive diagnostics to test the hardware
- **OCC:** On Chip Controller – PPC 405 processor that controls the power management per chip
- **OpenBMC:** Opensource service processor firmware stack used for OpenPOWER servers
- **PCB:** Pervasive Control Bus – internal processor bus that provides an out of band communication layer between the internal logic within the chip
- **PNOR:** P10 Processor NOR chip. NOR flash device where all firmware, including hostboot firmware, is stored and
- from which it is loaded. It is attached to the master processor through an LPC → SPI bus connection. Called PNOR to distinguish from other NOR chips in the system
- **PPE**: PowerPC Lite Engine. Small PPC controller macro used for various embedded control functions with the P10 chip (eg SBE, General Purpose Engines (GPEs with the OCC complex), and IO control)
- **SBE:** Self Boot Engine – A version of a PPE within each P10 chip which is used to do some basic initialization to each chip and to load and start the Hostboot firmware

1.3 IPL Types

IPL Type	SP Power	CEC Standby Power	CEC Logic Power	CFAM Reset	Mainstore Contents	Applicable Platform
Standby POR	Off -> On	Off -> On	Off	Yes	Off	FSP, eBMC
Cold IPL	On	On	Off -> On	FSP: No eBMC: Yes	Cleared	FSP, eBMC
Warm IPL or Warm Re-IPL	On	On	On	FSP: No eBMC: Yes	Cleared	FSP, eBMC
Auto Reboot (after failure)	On	On	FSP: On eBMC: On->Off->On	FSP: No eBMC: Yes	Cleared	FSP, eBMC
Memory-Preserving Re-IPL	On	On	On	No	Maintained	FSP, eBMC
Reset-Reload	Off -> On	On	On	No	Unaffected	FSP, eBMC

IPL Type	SP Power	CEC Standby Power	CEC Logic Power	Mainstore Contents	Applicable Platform
Standby POR	Off -> On	Off -> On	Off	Off	OpenBMC
Cold IPL	On	On	Off -> On	Cleared	OpenBMC
Warm IPL or Warm Re-IPL	On	On	On	Cleared	OpenBMC

1.4 Nomenclature/Conventions

- Items in bold-italics represent deliveries from the hardware team, either procedures or engineering data files. Ex: ***p8_cfaminit.C***
- Steps in italics are software only and do not interact with the hardware at all.
- istep commands in black are performed by the Service Processor (SP)
- istep commands in blue are performed by the Self Boot Engine (SBE)
- istep commands in green are performed by the host code.
    - Note for multi-drawer or multi-Hostboot instances all commands are issued in parallel to all instances, except where otherwise noted
- istep commands in black are performed by the attached service processor

2 Service Processor Power On to Standby

2.1 FSP Based

This flow starts as soon as the power supplies are plugged in and ends when the SP reaches the standby state.

1. Base Wiring
    a) C_FSI_IN_ENA – tied to 0b1 for FSP to indicate that FSI is driven by FSI HW clock
    b) C_JTAG_TMS pin – tied to 0b0 to prevent autostart and allow BMC to select SBE/PNOR side (root ctl regs)

2. Apply AC Power
    a) Standby regulators power on from 12VCS
    b) Reset generator starts 200ms after Standby Pgoods high
    c) Standby reset feeds into FSP/DPSS/APPS. (any intelligent device). FSP/APSS running. DPSS in reset if FSP present. If no FSP, then DPSS starts...Ckip steps 4-5.
    d) CFAM_RESET_B on all CFAMs (P10). Must be at least 100ms after VStandby
3. APSS loads itself from internal flash. We can update the APSS load in the lab from an internal connector. APSS is not updateable from FSP.
4. FSP starts running from SPI Flash at address 0.
5. FSP signs the DPSS load. There is only 1 – no golden image needed.
    a) Where the DPSS gets its load from the FSP
        - Held in FSP flash and clocked in
    b) DPSS load is updateable from the FSP, but not from the host
6. FSP releases DPSS reset.
7. DPSS begins running
    a) PGOOD Reset engine holds PGOODs off
    b) Registers initialized to default safe values (e.g. fans set to high speed).
    c) Waits for “GO” from FSP or external hardware entity to start power sequencing (DIO)
    d) From FSP, DPSS can be controlled to set LEDs, override fan speed (for fans on SB, if any),
8. FSP Machine type VPD read
9. FSI Clocks start to P9/Centaur

10. FSI break command sent to both slaves
11. FSI Arbitration is performed between Redundant FSPs
12. Read FSI Config Space
    a) Builds the device driver structure
    b) Device drivers “init”/clear engines – issues engine resets
13. Read all VPD (includes DRAM spd)
14. Start power process
15. Start HWServer process. At this point HWServer must build the object model based on VPD and chip IDEC
    a) istep BuildHwModel
        - This istep will build the HW model
        - HWServer will read the IDEC from the shift engine and the scom engine
        - During this phase (in non Reset-Reload cases) the primary FSP HWServer will force ownership of the LPC2SPI local bus for PNOR access
16. System now at SP Standby

At the end of this flow the SP and DPSS chip are powered on and viable. For the purposes of this discussion SP Standby is just enough information to power on the CEC logic. SP has **not** issued the “GO” bit yet.
If the P9 and Centaur chip's CFAM portion is powered on and has FSI clocks.

2.2 eBMC Based

1. Base System wiring
    a) C_FSI_IN_ENA – tied to 0b1 for BMC to indicate that FSI is driven by FSI clock
    b) C_JTAG_TMS pin – tied to 0b0 to prevent autostart and allow BMC to select SBE/PNOR side (root ctl regs)
        - If want autostart without BMC control, then tie to 0b1
2. Apply AC Power
    a) Standby regulators power on from 12VCS
    b) Reset generator starts 200ms after Standby Pgoods high
    c) present. If no FSP, then DPSS starts...Ckip steps 4-5.
    d) CFAM_RESET_B on all CFAMs (P10). Must be at least 100ms after VStandby
3. APSS loads itself from internal flash. We can update the APSS load in the lab from an internal connector. APSS is not updateable from eBMC.
4. eBMC starts running from SPI Flash at address 0.
5. eBMC Machine type VPD read
6. FSI Clocks start to P10
7. FSI break command sent
8. FSI Arbitration is performed
9. Read FSI Config Space
    a) Builds the device driver structure
    b) Device drivers “init”/clear engines – issues engine resets
10. Read all VPD (includes DDIMM spd via I2C)
11. BMC to power on:
    a) Enable VDN
    b) Ref clock enabled and toggling (based on VDN)
    c) Unload all FSI device drivers

d) Toggle CFAM_RESET_B
e) Reload all FSI device drivers
f) Set P10 config/control regs (mailbox scratch registers for SBE side selection/mfg flags)
g) Issues CFAM commands to kick off Master processor chip SBE

2.3 SPLess Based

1. Base System wiring
   a) C_FSI_IN_ENA – tied to 0b0 to indicate that FSI is driven by ref clock
   b) C_JTAG_TMS pin – tied to 0b1 to allow autostart
2. Apply AC Power
   a) Standby regulators power on from 12VCS – all power rails come up except VDDR
   b) Reset generator starts 200ms after Standby Pgoods high
   c) APSS loads itself from internal flash.
   d) Ref clock enabled and toggling
   e) HW Toggle CFAM_RESET_B
   f) IPL Starts

3 Cold IPL

This flow covers the steps from FSP/BMC Standby through the initial handshake with PHYP/OPAL.

0 Step 0

`0.1 poweron : Power on system`

a eBMC:

- Unload all FSI device drivers
- Toggle CFAM_RESET_B to all P10 chips
- Reload all FSI device drivers
- Power on

b FSP: Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/PowerOn.tf

- SVPD Recollection (checking for something new plugged in since FSP Standby)
- istep pre_poweron step – only needed if pre_poweron procedures are needed
  - p9_pre_poweron.C
    - Leave as no-op
- Disable any softswitches that need to be disabled
- Must assert PERST on systems that have direct control (POWR) for cold IPLs via FSP GPIOs
- SP sends “GO” command to DPSS to power on the system

- DPSS or the regulator itself sets the default voltage of Variable VRMs
  - Power sequence stage 0 runs to completion, this includes everything except for VDDR (and VPP for DDR4) rails
  - Chip team must determine a safe voltage/frequency for ALL chips in a family
  - Regulator hardware has proper safe voltage value hard-coded with strapping resistors
  - The VRM Vsafe voltage is set by the system designers and will not move until it receives a valid SPIVID command from the processor
- VLogic PON all chips except for dimm rails (VDDR)
- Enable any required softswitches

0.2 `startipl : Start IPL on SP`

- On warm re-ipl this is the entry point to the IPL flow
- Gets SP into a state ready to IPL the CEC

b Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/StartIpl.tf

- Starts attnhdlr process (not listening for attentions)
- issue istep DisableAttns for the hw reconfig loop

0.3 `DisableAttns : Disable Attentions`

- Turn off attention handler
- This is the entry point for FSP controlled HW reconfig loop

0.4 `updatehwmodel : Update the HWServer Model`

- Update HWServer model to account for any new HW
- Take ownership (if primary) of LPC2SPI of any newly detected PNOR
- If PNOR has been locked by SPI command, need FSP/BMC to have the ability to toggle the SPI Reset line (which unlocks the SPI NOR)

0.5 `alignment_check : Check boot reqs for FSP`

- This step will call hardware server to run the alignment check
- Check to see which FSP has the better view for booting the maximum nodes
- If other FSP has better view it will trigger a failover

0.6 `set_ref_clock`

a OSC is active once CEC logic power is on. ***Chip reference clocks start when their voltage rails come up, this step allows for the reference clock frequencies to be adjusted.***

b FW will issue I2C commands to external OSC chip to enable the clock output

- Not a HWP, simple shared code between FW and Cronus

c FW will set ATTR_OSC_CNFG to indicate if both/particular clock will be used. If both will have indication of preferred clock (0 vs 1). This will be used by clock HWPs and must be sent in to SBE via mbox scratch regs (four options, so 2 bits → Only A side (0b00), Only B side (0b01), both with A preferred (0b10), both with B preferred (0b11)

- TOD based off system ref

d p10_setup_ref_clock.C

- Since this is the first procedure run against the chips it also clears the GP write protect
- Setup the clock termination inside P10 chip correctly for system/chip type (OpenPOWER systems must default correctly)
- Will include reset values for any root control register is touches
- Select internal clock mux reachable by root controls (OpenPOWER systems default correctly)
  - Specifically the muxes/clocks used by SBE
- Set RCS into a reset and bypass state, and configure bypass clock per ATTR_OSC_CNFG

e

f p10_set_fsi_gp_shadow.C (FSP based systems only, no-op on BMC due to CFAM reset)

- Corollary in BMC based system is the CFAM_RESET
- Done for all boots – some settings will change based on system type and IPL type
- Set the GP bits to default state
- Will not touch root control registers touched by p10_setup_ref_clock.C

0.7 proc_clock_test

a p10_clock_test.C

- Test to see if the ref clock is valid. Does clock test on RCS – verifies that the clocks specified by ATTR_OSC_CNFG are toggling. If it doesn't match, HWP calls out the clock(s). It is up to FW to reboot with a different ATTR_OSC_CNFG (or terminate the IPL)
- This is run prior to switching the frequency. It is intended to just see if the processor is getting valid reference clocks

0.8 proc_prep_ipl (no-op on BMC)

- Clear all pending messages in the mailbox via adal_mbx_reset (a write to error reset, resets all mbox regs)
- Needed since mbox is on standby power – don't want garbage lying around on warm IPLs

0.9 edramrepair : Noop

- No need to store L3/L2 Line deletes in VPD
- Store current boot detect line deletes in ATTR @runtime
  - ATTR are cleared each boot
- When threshold reached, run the column repair alg
  - If can't repair the column then predictive error

0.10 asset_protection

- On FSP based systems a basic sanity check of plugged parts is performed

0.11 proc_select_boot_prom

a p9_select_boot_master.C

- This HWP selects which Redundant SEEPROM to use
  - This must be set only for the master processor (HB will set later for slaves) depending on current IPL (normal or SBE update directed by Hostboot)

0.12 hb_config_update

- Update Hostboot config data area with any configs/parameters. This may involve updating the PNOR Hostboot attributes
  - Update attributes based on next/previous ASM flags
  - Update istep mode if desired, etc
- Update the PNOR version copy of the dimm SPD and module VPD
- If there is an override section present in /nfs/ then copy to the PNOR ring override section
- Issue commands to configure the SFC direct read window for processor chip access to PNOR
- Save away location of Hostboot base image to attribute for sbe_config_update to use to setup mbox registers

0.13 sbe_config_update

- On BMC systems this is done via direct writes to mbox scratch regs

b p10_setup_sbe_config.C

- See istep 2.2 for details of scratch registers and ATTR mappings
- This includes the Master/slave indication (for FSP/BMC it always sets master)
- Take the FSP/Cronus/hostboot FAPI2 ATTR and write them to the mbox scratch registers
- Data shuffling of the ATTR into an extremely compact form
- In manufacturing mode the SP may be required to update the entire seeproom image via xip_customize. See istep TBD for details
  - Note that the ring override from /nfs/ should be applied during the xip_customize flow if directly updating the SBE

- Note to take into account the dead space between the 64KB SEEPROM images for SBE ECC

0.14 sbe_start

- ♦ Grant the LPC2SPI FSI bus to the LPC bus so the SBE and Hostboot can access the PNOR
- ♦ Done on all warm/cold IPLs under SP control.

b p10_start_cbs.C

- ♦ Set a bit to trigger the CBS on the P9 master chips. Located in FSI GP region
- ♦ CBS applies GP shadows to GP regs, causes endpoint resets
- ♦ The CBS will scan0 flush of pervasive, start clocks
- ♦ For MPIPL the CBS is not used and FSP directly triggers the SBE

0.15 startPRD

- ♦ Build diagnostics model for P8 chips
  - FIRs will be unique for HBI execution, HBI will restore at end of execution

0.16 proc_attn_listen

- ♦ Enable attention interrupts in engine for master PgP chip per node
- ♦ At this point any fatal (checkstop) will force PRD to issue a cancontinue loop

1 Step 1 – Self Boot Engine OTPROM and PIBMEM

Note: release of FSI Go bit triggers SBE executing from OTPROM

a. Processor and FSI reference clocks are stable
   - Running off of RCS in bypass mode – know that clock is toggling, SBE will move to functional PLL in istep 2

b. The following steps are done by the CBS. This happens regardless of the TMS line holding the SBE engine from fetching. The CFAM_RESET or FSI GP bit triggers this.
   - Apply the root_ctrl shadow registers to the effective root_ctrl registers
   - Init TP chiplet (done by clock controller)
   - Start TP Vital – TP mesh clocks (done by clock controller)
   - Scan 0 flush – SBE can't scan PRV PIB or PCB regions as it is part of pervasive itself
     - Covers PCB and TP vital
     - This clears all security bits in the OTPROM controller AND the SDB bit in the mailbox
   - Release tholds for TP and PCB – **running on the ref clock**
     - Pervasive clocks will be started by Clock Controller Logic; SBE itself receives these clocks and therefore can't run before clocks are running
     - Vital and PCB, not tholds to rest of TP
   - PIB Bus operational now

c. SBE is reset to state that fetches directly from OTPROM (it is on the PIB)
   - SBE instructions are parity protected, but SBE instruction parity is turned off (per VBU to make it easier for assembler compilers), but OTPROM is ECC protected
   - SEEPROM, PNOR SBE, and partition NVRAM are ECC protected
   - OTPROM is ECC protected
     - ECC checking is ON by default, scom bit to turn off ECC
     - Mechanism to stop SBE prior to any instructions issued is to use the FSI GP bit

- SEEPROM is ECC protected,
- Scan chains must be ECC protected in SEEPROM, ECC protected in OTPROM/PNOR
- CBS triggers SBE start to fetch instructions
  - SBE will not execute if external pin TE (Module/Wafer Manufacturing Test Enable) =0b1

1.1 proc_sbe_enable_seeprom :F,C - Select SEEPROM address

a This istep is not controllable by FW – once the CBS starts the boot sequencer the SBE will automatically execute this istep. It is listed as an istep for documentation but cannot be manually controlled via istep.

b p10_sbe_enable_seeprom.C (no param) –

- Entrance into this procedure is via SBE Reset (hard) or CBS.
- Hard reset – triggered by SP (and potentially DTRM) without using the CBS
- CBS – runs scan 0 flush SBE, PIB, NET regions. Then does a clock start of the SBE region only, issues hard reset to SBE
- This HWP is not FAPI2 based:
  - It runs directly in OTPROM and cannot use attributes
  - It is burnt into the chips OTPROM during manufacture
- Select which redundant SEEPROM to use based on MBOX Control bit
  - 0b0 – use default SEEPROM (bit 17 of Self Boot Control/Status Register)
  - 0b1 – use alternate SEEPROM (bit 17 of Self Boot Control/Status Register)
- Resets the SPI bus
  - OTPROM only resets the SPI bus to the measurement SEEPROM
  - Code in the measurement SEEPROM does the reset of the SPI buses for TPM and boot SEEPROMs
- If scratch reg is set then it uses SPI speed from scratch, else uses default burned into OTPROM at MFG
- Check that measurement SEEPROM is accessible and image is valid (XIP header magic check)
- Then branch to measurement SEEPROM location TBD–
  - Magic number to address 0 (SBE) and jump point at address 0x4
  - Physically on the SEEPROM this will be 0x0

1.2 proc_sbe_pib_init : -- repair the PIBMEM

a **For a FW (SBE boot) this is a non skippable istep**, it will automatically run as part of the SBE boot

- In FW not a FAPI HWP, instead raw C
  - Executes on the SBE – not against FAPI (OTPROM direct content)
  - Cannot use attributes
- Measurement SEEPROM
  - Runs a basic series of scoms via sequencer

- Scans in PIBMEM repairs
- Clocks start PIB units inside the PIB/NET clock region
- That does the PIBM repair, starts the PIBMEM and then continues on (isolated from rest of TP)

b ***FOR CRONUS – will call the following HW in sequence. They are broken apart for test:***

- p10_sbe_pcb_release_reset.C
- p10_sbe_pibmem_repair.C (scanning separated out)
- p10_sbe_pib_init.C

1.3 proc_sbe_measure : -- code run from the measurement PROM

- The measurement code loads itself into PIBMEM in order to run a SHA256 measurement on the boot loader

b This is a non skippable istep, it will automatically run as part of the SBE boot

- Not a FAPI HWP, instead raw C

c Issues SPI resets

d Does the TPM init and initial extend

e Eventually jumps to boot SBE code

2 Step 2 Self Boot Engine – Pervasive Chiplet Setup

When the SBE first jumps to the SEEPROM it will jump to a routine delivered by FW to potentially collect FFDC based on the reset type. Then it will move to istep 2.1.

2.1 proc_sbe_ld_image :F,C - Load PIBMEM image

a This istep is not controllable by FW – once the CBS starts the boot sequencer the SBE will automatically execute this istep. It is listed as an istep for documentation, but cannot be manually controlled via istep.

- Not a FAPI HWP, instead raw C
  - RAW one that executes on the SBE – not against FAPI (OTPROM direct content)
  - Cannot use attributes
- Turn on SBE internal RISC trace via the SBE internal trace configuration register
- Performs PIBMEM repairs (via load/stores to PIB – aka scoms) only on start vector 0 (start vector 1 is used for warm resets and PIBMEM has already been setup and contains FFDC from/for the reset)
  - Data in pibmem is valid as long as previous steps did not go through scan 0 flush/clock start of PIB domain (CBS start does the scan0 flush)
  - SBE must always treat existing data in PIBMEM as FFDC only and always reload instructions
  - PIBMEM repairs are not required if the SBE is not being used (ie boot via FSI2PIB path)
    - For DFT if the PIBMEM repairs are needed, DFT is responsible for loading
- Loads the pib attached memory image from the SEEPROM This image contains various utilities used

throughout the SBE IPL:

- Kernel
- Base Utilities
- SBE fixed data section (aka ATTR) into PIBMEM

♦ Branch into SBE kernel, start executing Kernel

- Enter control loop
- After this point FW Control loop is in charge of loading/unloading chip ops and calling future HWP
- SBE code checks the scratch registers to determine if in istep mode, if so then it enters istep mode and then waits for data on the FIFO. Otherwise it continues to boot automatically
  - If in non step mode, SBE will only honor FIFO operation to query IPL status/collect FFDC until it completes istep 5 or has an error

♦ All operations to the SBE are atomic from the SP perspective

♦ All power to the chip is on except

- Quads
- PHYs are all powered down

2.2 proc_sbe_attr_setup : F,C -Read scratch regs, update ATTR

a p10_sbe_attr_setup.C (chip target) FAPI2::ReturnCode

♦ If and only if scratch registers are non-zero, HWP will read the contents of the scratch registers and call FAPI2 APIs to set the values into the corresponding SBE platform ATTR values (Mbox reg contents PIBM ATTR)

- Scratch 7, byte 0 is a bit field that indicates validity of the other mailbox register

♦ In the case where HW scratch registers are zero – the values represented by the scratch registers need to be in a fixed location (ECC aligned) of the SEEPROM Image (SEEPROM contents PIBMEM ATTR Mbox scratch regs)

- SEEPROM image ATTR is the master, mailbox is just the overrides
- Fixed location for the ATTR and all mbox ATTRs are at the front and non moveable (can extend, but not move)
- In this case (scratch 7, byte 0 valid bit == 0) then data in the ATTR tank data in the SBE needs to be pushed back into the HW mailbox scratch reg for Hostboot to consume

♦ Hostboot will need the information in the scratch registers as well (for the slave chips, etc)

♦ Check the state of the SAB (Security Access Bit)

- If SBE image has ATTR_SECURITY_MODE == 0b1, then leave SAB bit as is
- Else ATTR_SECURITY_MODE == 0b0, then clear the SAB bit
  - ATTR_SECUIRTY_MODE may only be 0b0 with imprint keys
- Move state of SAB into ATTR_SECURITY_ENABLE
  - Rest of the SBE code to apply security restrictions based on ATTR_SECURITY_ENABLE
- If this is a Slave SBE, it must set the TPM protect bit to prevent the redundant TPM from being used

♦ **Mailbox scratch 1 (CFAM 2838, SCOM 0x50038) - FW functional Cores**

- This register gives FW additional control over functional cores that the SBE can consider for enabling. It is applied on top of the manufacturing partial good VPD. If the bit is on, then the part is non-functional
- Byte 0-3 Core Gard records. Each bit position corresponds to a core starting at core 0 being bit 0. Each nibble is representative of an EQ chiplets' Cores. The EQ is always functional, so this field only tracks core state. Bits 0..31 == ATTR_CORE_GARD

♦ **Mailbox scratch 2 (CFAM 2839, SCOM 0x50039) – FW functional Targets (non core)**

- This register gives FW additional control over functional parts of the chip that the SBE can consider for booting. If a part is disabled (bit is on, then the part is non-functional) then the SBE is expected to leave the part clock gated (ie all scoms will fail to it). It is applied on top of the manufacturing partial good VPD.
  - Bits 0..1 – PCI chiplets functional mask
  - Bit 2..3 -- Reserved
  - Bits 4..7 -- the functional memory chiplets
  - Bits 8..11 – physical PAU chiplet (8..11) functional mask. Separate from logical PAUs because PAU chiplet can be required for IOPPE, common TL logic and yet both PAUs could be unused
  - Bits 12..19 – Logical PAU targets inside of the PAU chiplets (2 per physical PAU chiplet). Note that PG will always disable logical PAU
  - Bits 20..27 – iohs chiplet functional mask.
  -

♦ **Mailbox scratch 3 (CFAM 283A, SCOM 0x5003A) – FW Mode/Control flags**

- The HWP does not need to do anything with this scratch register as it is SBE FW control flags. These will be stored as ATTR_BOOT_FLAGS in the ATTR tank (and by the setup mbox HWP). The SBE FW will check the valid bit and use the mbox scratch register, else it will use the value from the ATTR tank. The SBE FW will use these values prior to this HWP being run. Note that this is only used by the setup_sbe_config to push the data into the mailbox register when running on an FSP (not consumed by FAPI on SBE or as part of SBE SEEPROM customization).
  - Bit 0 indicates istep IPL (0b1) (Used by SBE, HB – FW ISTEP_MODE)
  - Bit 1 indicates that SBE should go directly to runtime functionality (0b1)
  - Bit 2 is reserved for HB usage for the SBE to indicate an MPIPL to Hostboot. It is always 0 in the ATTR tank and is dynamically set by the SBE at the same time the SBE sets the ATTR_MPIPL_MODE ATTR (Used by HB, set by SBE. SBE uses S0/S1 interrupt)
  - Bit 3 in this register is used to indicate FSPless (0b0), otherwise FSP attached (0b1)
  - Bit 4 – Reserved
  - Bit 5 in this register indicates that the SBE should **not** send back internal FFDC on any ChipOp failure response
  - Bit 6 – disable security. SBE is configured to only honor this request if and only if during the update process it was signed with a secure header flag that permits it. Hostboot checks the secure header flag, signing server is responsible for never setting secure header flag with production keys
  - Bit 7 – Allow ATTR overrides in secure system. Hostboot is configured to only honor this request if and only if during the update process it was signed with a secure header flag that permits it. Hostboot checks the secure header flag, signing server is responsible for never setting secure header flag with production keys
  - Bit 8,9 – FW SMT override mode. This does not control the HW capabilities, but instead instructs the FW control of SMT to only advertise a specific SMT mode (and triggers boot in that mode). These bits are only used by Hostboot, not the SBE. Setting SMT1 will result in Linux only being

told of 1 thread per core. Default of 0b00 is SMT4 mode (normal). 0b01 forces to SMT1, 0b10 forces to SMT2, 0b11 is undefined and will result in SMT4.

- Bit 10 – Flag (0b1) to indicate that upon master STOP cycle in istep 16, the SBE should NOT checkstop the system (for debug)
- Bit 11 – Disable Scom white/blacklist in secure systems. SBE is configured to only honor this request if and only if during the update process it was signed with a secure header flag that permits it. Hostboot checks the secure header flag, signing server is responsible for never setting secure header flag with production keys
- Bit 12 – Disable Invalid Scom address check
- Bit 13 – MFG mode. Flag that indicates to SBE was in MFG mode, where extra tests/isteps can be executed
- Bit 14 – Enable SBE output via SUART during boot
- Bit 15 – Skip dump collection in the MPIPL path
- Bit 16:27 – Available
- Bit 28:31 –Reserved – was INIT_LEVEL (will be deprecated when dynamic inits are supported, leaving in doc for fallback option)
  - Open item: How to control SBE Dynamic inits (aka init levels) for non core chiplet inits. We did use risk level for early DD1.x HW on P9. So have to support either init level or dynamic inits for SBE
  - Open item: How to advertise supported dynamic init options for a given FW level (would SBE be point of advertiser (holistic viewpoint instead of splitting between host/sbe)

♦ **Mailbox scratch 4 (CFAM 283B, SCOM 0x5003B) – Nest/Boot Freq**

- Bytes 0,1: Ref clock SPI bus divider consumed by code running out of OTPROM, no ATTR needed as it is directly read. ATTR is ATTR_I2C_BUS_DIV_REF (for image customization) – 12 bits for divider, 4 bits for round trip delay
- Byte 2,3 – core boot frequency (in MHz) from platform (ATTR_FREQ_CORE_BOOT_MHZ

♦ **Mailbox scratch 5 (CFAM 283C, SCOM 0x5003C) – HWP Control Flags**

- Bit 0,1 -- SYSTEM_IPL_PHASE, where 0b00 – HOSTBOOT, 0b01 -- CACHE_CONTAINED, 0b10 is CHIP_CONTAINED
- Bit 2 – RUNN Mode (Cronus cache/chip contained only)
- Bit 3 – Boot loader HWP flag to not place 12K exception vectors. This flag is only applicable when security is disabled (ATTR_DISABLE_HBBL_VECTORS == 0x1)
- Bit 4:6– Policy dictating selection of active and backing caches (ATTR_SBE_SELECT_EX_POLICY), where 0b000 – HB_DEFAULT, 0b001 – CRONUS_MAX_ACTIVE, 0b010 – HB_MAX_FOOTPRINT, 0b100 – HB_MAX_THREADS, 0b110 – HB_MAX_FOOTPRINT_MAX_THREADS
- Bits 7 – Cache contained load method (0=PBA, 1=L2 store queue loader)
- Bits 8:9 – Reserved/Open
- Bits 10:11– Clockstop on checkstop options (0b00=disabled, 0b01=stop on checkstop, 0b10=stop on attn, 0b11=staged xstop)
- Bits 12:13 – IOHS 0 LCPLL; Routes tank PLL reference clock in the chiplet for debug
- Bits 14:15 – IOHS 1 LCPLL; Routes tank PLL reference clock in the chiplet for debug
- …
- Bits 26:27 – IOHS 7 LCPLL; Routes tank PLL reference clock in the chiplet for debug
- Bits 28:29 – PCI 0 LCPLL; Routes tank PLL reference clock in the chiplet for debug

- Bits 30-31 – PCI 1 LCPLL; Routes tank PLL reference clock in the chiplet for debug
  - NOTE: bits 12:31 are don't cares if the chiplet is knocked out. Chip level mux selects are controlled in separate root control register

♦ **Mailbox scratch 6 (CFAM 283D, SCOM 0x5003D) – Master/Slave, node/chip selection**

- Bit 0:1 – ATTR_FILTER_PLL_BUCKET – ring select for TOD, Nest, IO, IO SS (0b00 is product, 1-3 is for MFT usage)
- Bit 2:3 – ATTR_PCI_PLL_BUCKET – ring select for PCIE PLL (0b00 is product, 1-3 is for MFT usage)
- Bit 4:5 – Reserved
- Bit 6 – ATTR_SKEWADJ_BYPASS – Keep Skew Adjust logic in bypass
- Bit 7 – ATTR_DCADJ_BYPASS – Keep Duty Cycle Adjust logic in bypass
- Bit 8 – ATTR_CP_PLLTODFLT_BYPASS – force TOD filter PLL into bypass
- Bit 9 -- ATTR_CP_PLLNESTFLT_BYPASS – force Nest filter PLL into bypass
- Bit 10 -- ATTR_CP_PLLIOFLT_BYPASS – force IO PLL into bypass
- Bit 11 -- ATTR_CP_PLLIOSSFLT_BYPASS – force IO Spread Specturm PLL into bypass
- Bit 12 -- ATTR_NEST_DPLL_BYPASS– force nest DPLL into bypass
- Bit 13 – ATTR_PAU_DPLL_BYPASS -- force PAU DPLL into bypass
- Bit 14 – ATTR_IO_TANK_PLL_BYPASS – force all iohs, omi, pcie (+ filter for pcie) into bypass
- Bit 15 – ATTR_BOOT_PAU_DPLL_BYPASS – force the SBE to boot with the PAU DPLL in bypass (defaults to a 0b1)
- Bits 16:19 – ATTR_FABRIC_EFF_TOPOLOGY_ID(0:3) – Specify topology ID of the socket providing boot memory (should == FABIRC_TOPOLOGY_ID, unless master socket has no memory)
- Bit 20 – ATTR_FABRIC_TOPOLOGY_MODE – specify the fabric topology mode (mode0 = GGG_C, mode 1 = GG_CC)
- Bit 21 – Reserved
- Bit 22:23 – ATTR_FABRIC_BROADCAST_MODE(0:1) – specify hop configuration and pump mode
- Bit 24 – indicates Hostboot master bit, 0b1 == master, 0b0 == slave
  - Default SBE image must ALWAYS have bit 24 on, ie can't be customized via xip_customize in the SEEPROM
  - **Both the external C4 pin (0xF000F bit 40?) and ATTR_PROC_SBE_MASTER_CHIP – ie a 0b1 when master, 0b0 when slave**
    - Note that the 2nd SBE FIFO protection logic keys off of the external input only – so it will block all hub FSI access to downstream P10 2nd SBE FIFOs
  - If set as slave then this overrides the external C4 indicating master/slave
  - If set as master then use the external C4 as indication of master/slave
  - The default SBE image will always have bit 24 indicating master (0b1), which will allow the board C4 pin to control master/slave
  - On BMC based systems the master processor must have the external C4 (e_c4_prv_rx_chip_master) pin tied high
  - For systems where the SP is intended to select master/slave, all module C4 pins must be tied high (indicating master) so that bit 24 will allow the SP to control master slave selection
- Bit 25:27 – ATTR_CP_REFCLOCK_SELECT – Select which of the two reference clocks is supposed to be used:
  - 0b000 – OSC0 – use only OSC0
  - 0b001 – OSC1 – Use only OSC1

- 0b010 – BOTH_OSC0 – Both OSC0 and OSC1 must be good, set up redundant clocks, OSC0 primary
- 0b011 – BOTH_OSC1 – Similar to BOTH_OSC0 but use OSC1 as primary
- 0b100 – BOTH_OSC0_NORED – Try to set up redundant clocking with OSC0 as primary, but don't fail if OSC1 fails
- 0b101 – BOTH_OSC1_NORED – Try to set up redundant clocking with OSC1 as primary, but don't fail if OSC0 fails

- Bits 28:31 – ATTR_FABRIC_TOPOLOGY_ID – specific topology ID for this socket (replacement for Group/Chip ID ATTR)

♦ **Mailbox scratch 7 (CFAM 283E, SCOM 0x5003E) – IOHS DL Mode**

- Byte 0-2: This register gives FW additional control over the functional usage of the IOHS DL layers. The same IOHS PHYs can be used by NDL (Nvlink) or ODL (OpenCAPI) or PDL (Powerbus). Only one of the DL layers can be used with a given PHY. The various usage is system dependent, and for power savings if a particular DL can't be used it will be power gated during boot. If a part is disabled (bit is on, then the part is non-functional) then the SBE is expected to leave the part power gated (ie all scoms will fail to it). It is applied on top of the manufacturing partial good VPD.
  - Bits 0-7: Reserved
  - Bits 8-15: NDL – functional usage of the Nvlink DL layer
  - Bits 16-23: Reserved
- Bytes 3 — Reserved
- **NOTE: If chip is configured in chip-contained mode, this register will assume an alternate meaning**
  - Bits 0-31: ATTR_CHIP_CONTAINED_ACTIVE_CORES_VEC – Vector of active cores, where bit X = core X

♦ **Mailbox scratch 8 (CFAM 283F, SCOM 0x5003F)**

- Byte 0 – each bit in here indicates validity of the same numbered scratch reg 1 greater (bit 0 scratch 1)
  - Bit 0 -- (CFAM 2838, SCOM 0x50038) - FW functional EQ/EC valid
  - Bit 1 -- (CFAM 2839, SCOM 0x50039) - SBE I2C Bus speed based, ref clock valid
  - Bit 2 -- (CFAM 283A, SCOM 0x5003A) - FW Mode/Control flags valid
  - Bit 3 -- (CFAM 283B, SCOM 0x5003B) - Boot frequency valid
  - Bit 4 -- (CFAM 283C, SCOM 0x5003C) - HWP Control Flags valid
  - Bit 5 -- (CFAM 283D, SCOM 0x5003D) - Master/Slave, node/chip selection valid
  - Bit 6 -- (CFAM 283E, SCOM 0x5003E) – DRTM Payload address in MB valid
  - Bit 7 -- (CFAM 283F, SCOM 0x5003F) – bytes 1,2,3 (if used) valid
- Byte 1 – each bit in here indicates validity of the scratch reg 8 greater (bit 0 = scratch 8)
  - Bit 8 -- (CFAM 180, SCOM 0x50180) – PAU, MC Freq valid
  - Bit 9 -- (CFAM 181, SCOM 0x50181) – IOHS PLL valid
  - Bit 10 -- (CFAM 182, SCOM 0x50182) – Future scratch reg valid
  - Bit 11 -- (CFAM 183, SCOM 0x50183) – Future scratch reg valid
  - Bit 12 -- (CFAM 184, SCOM 0x50184) – Future scratch reg valid
  - Bit 13 -- (CFAM 185, SCOM 0x50185) – Future scratch reg valid

- Bit 14 -- (CFAM 186, SCOM 0x50186) – Future scratch reg valid
- Bit 15 -- (CFAM 187, SCOM 0x50187) – Future scratch reg valid

- This is used to know if the data should be updated from scratch to attributes

♦ **Mailbox scratch 9 (CFAM 2980, SCOM 0x50180) – PAU, MC Freq**

- Byte 0,1 -- PAU PLL freq (ATTR_PAU_FREQ_MHZ)
  - The freq is set once at IPL, doesn't change.
  - Feeds the PAUs and TP_CONST. Requirement from OCC that it is greater or equal to 2Ghz.
- Bits 16:31 – ATTR_MC_PLL_BUCKET. 4 x 1 byte array, where each array entry X indicates which bucket to use for the
- OMI bus frequency for Mem controllers (which is dictated by the OCMB frequency)
  - The PLL bucket number is an integer enum, with the actual frequency defined within the bucket
  - Specifically, Explorer is synchronous to DDR
  - Supported buckets: 21.33Gps (DDR 2666Mts), 23.46Gps (DDR 2933Mts), 25.6Gps (DDR 3200Mts). 1 DDR5, 2 MFT designated rings + Spare(s)
  - OMI bus frequency for MC X

♦ **Mailbox scratch 10 (CFAM 2981, SCOM 0x50181) – IOHS PLL**

- The PLL bucket number is an integer enum, with the actual frequency defined within the bucket (Six in the list from product, 2 MFT designated rings + Spare(s))
- Bits 0:31 – ATTR_IOHS_PLL_BUCKET – an 8 x 1 byte array, where each array entry X indicates which IOHS PLL bucket to use for iohs chiplet X
- **Note: If chip is configured in chip-contained mode, this register will assume an alternate meaning**
  - Bits 0-31: ATTR_CHIP_CONTAINED_BACKING_CACHES_VEC – Vector of chip contained backing caches, where bit X = cache X

♦ **Mailbox scratch 11 (CFAM 2982, SCOM 0x50182) – SBE secure FFDC**

- SBE will use this scratch register to pass FFDC on a secureboot validation error up to hostboot to log an error

♦ **Mailbox scratch 12 (CFAM 2983, SCOM 0x50183) – Host Running Indication**

- Host will set this register to "0xA5000001" when it can run without the BMC (for PHYP this is after loading NVRAM and PDR exchange)

♦ **Mailbox scratch 13 (CFAM 2984, SCOM 0x50184) – SPIM TPM settings**

- Bytes 0,1: At-speed SPI bus divider consumed by code running out of measurement SBE, no ATTR needed as it is directly read. ATTR is ATTR_TPM_SPI_BUS_DIV (for scratch reg setting) – 12 bits for divider, 4 bits for round trip delay
- Byte 2,3 – Used to indicate PAU frequency on MPIPL/HReset

♦ **Mailbox scratch 14 (CFAM 2985, SCOM 0x50185) – SBE secure FFDC #2**

- SBE will use this scratch register to pass FFDC on a secureboot validation error up to hostboot to log an error

♦ **Mailbox scratch 15 (CFAM 2986, SCOM 0x50186) – BMC SBE state register**

- BMC will use this register to keep track of the SBE state (being hreset or running)

- **Mailbox scratch 16 (CFAM 2987, SCOM 0x50187) – SBE runtime SPI settings**
  - SBE SPI settings in bits 0:27
  - Bits 28:31 -- Used to indicate PAU frequency limit on SPI Chip Select workaround

2.3 proc_sbe_tp_dpll_bypass :F,C,D Reset the TP clock to bypass

a p10_sbe_tp_dpll_bypass.C (chip target) FAPI2::ReturnCode

- Will check the state of the DPLL clock region, if running then it will revert back to the bypass clock
  - Bring the PAU DPLL back down
  - Update the SPI delay settings and clock divider
- Needed because all TP scanning must be done in bypass (but we want at speed for verification)

2.4 proc_sbe_tp_chiplet_reset :F,C,D TP Chiplet reset

a p10_sbe_tp_chiplet_reset.C (chip target) FAPI2::ReturnCode

- Releases the Pervasive Control Bus (PCB) reset
- Sets TP chiplet enable
- Drops pervasive chiplet fence
- Setup hang counter for PCB slaves/master

2.5 proc_sbe_tp_gptr_time_initf :F,C,D – Init Perv GPTR/Time

a p10_sbe_tp_gptr_time_initf.C

- Scan init the GPTR and Time rings for the Pervasive chiplet

2.6 proc_sbe_dft_probe_setup_1 :D, - Setup DFT probe points

a p10_sbe_dft_probe_setup_1.C (chip target) FAPI2::ReturnCode

- Only run in DFT mode, no-op in normal Cronus/SBE (istep stub for common numbering)
- DFT mode is controlled with IPL option within Cronus

2.7 proc_sbe_npll_initf :F,C,D - Program Powerbus PLL

a p10_sbe_npll_initf.C (chip target) FAPI2::ReturnCode

- Apply config rings for all 8 chip level PLL rings (2 Red. Clock Switch, 4 filter PLLs, 2 (Nest & PAU) DPLLs)
- No frequency buckets – all systems/chips run the same settings
- IOHS, PCIe, and MC PLLs are not set (still running in bypass)

2.8 proc_sbe_rcs_setup

a. ***p10_sbe_rcs_setup.C (chip target) FAPI2::ReturnCode***

- Initialize the RCS
  - Drop RCS reset, keep in bypass
  - check clock redundancy (PIB path) based on ATTR_OSC_CONFIG
  - Starting PLLs,
  - checking for lock, then take out of bypass
- Check for errors (really only applicable if in redundant mode)
  - If have a fail and still have running ref clock, then continue IPL and callout the bad clock
  - At runtime – need a FIR (recoverable) to fire when RCS switchover happens – PRD will handle callout and "remembering" for next boot. Once it fails over that is it, only opportunity for this IPL? Not possible to replace at runtime? Bruce will support in HW (depending on RAS team) – up to FW to enable
  - If RCS fails to B, then B fails can we prevent going back to A? If A_err is latched → don't go back. If FW clears then it is okay to go back → might

2.9 proc_sbe_tp_switch_gears : F,C,D - Update SBE I2C config

a. ***p10_sbe_tp_switch gears.C (chip target) FAPI2::ReturnCode***

- Calls procedure to update SPI bus speed (all code must reside in the PIBMEM)
- Update SPI master bus timing for Measurement PROM, boot PROMs and TPM based on nest freq

2.10 proc_sbe_npll_setup : F,C,D - Nest PLL setup

a. ***p10_sbe_npll_setup.C (chip target) FAPI2::ReturnCode***

- Second initialize the filter PLLs
  - Check for lock, take out of bypass
- Third – start the nest and PAU DPLLs
  - PAU will be programmed with ATTR_PAU_FREQ_MHZ (via scratch reg, which is customized from #V/System specific)
  - NEST will be programmed with ATTR_BOOT_FREQ_MHZ (via scratch regs)
- Clocking: enable Nest Progdly (set nest progdly bypass to zero) – still need this (also skew delay)
- Makes use of a glitchless mux to switch

2.11 proc_sbe_tp_repr_initf : F,C,D - TP Chiplet Repair

a. ***p10_sbe_tp_repr_initf.C (chip target) FAPI2::ReturnCode***

- Load Scan Repair for TP Chiplet

2.12 proc_sbe_setup_tp_abist g: D -- Hook for DFT to run abist on TP

a. ***p10_sbe_tp_abist_setup.C (chip target) FAPI2::ReturnCode***

- Spot for DFT to insert non zero (ie true abist) patterns

2.13 proc_sbe_tp_arrayinit :F,C,D - TP Chiplet array init

a p10_sbe_tp_arrayinit.C (chip target) FAPI2::ReturnCode

- Does not reinit PIBMEM
- Run arrayinit on TP chiplet (includes OCC)
  - After this all TP arrays are initialized (including OCC SRAM tank)
- Scan flush 0 to all TP expect TP Time, GPTR, Repair rings and PIB, and PCB regions

2.14 proc_sbe_tp_initf :F,C,D - TP Chiplet scan inits

a p10_sbe_tp_initf.C (chip target) FAPI2::ReturnCode

- Apply scan overrides to TP Chiplet (includes OCC)

2.15 proc_sbe_dft_probe_setup_2 :D, - Setup DFT probe points

a p10_sbe_dft_probe_setup_2.C (chip target) FAPI2::ReturnCode

- Only run in DFT mode, no-op in normal Cronus/SBE (stub istep left for common numbering)

2.16 proc_sbe_tp_chiplet_init :F,C,D - TP Chiplet Start clocks

a p10_sbe_tp_chiplet_init.C (chip target) FAPI2::ReturnCode

- Switches TP Chiplet OOB mux
- Resets PCB Master Interrupt register
- Drops pervasive and OCC2PIB fence
- Start clocks on perv region (all components of TP)
- Clear force_align in chiplet GP0
- Clear flushmode_inhibit in chiplet GP0
- Drop FSI fence so checkstop and interrupt conditions can flow – SBE has direct path, this is normal TP chiplet path
- Pervasive Trace arrays are now available
- Check for OSC switch clock errors after switching to Nest PLLs
- Theoretically can run the OCC at this point
- If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED
  - Tweak FIR Masks

3 Step 3 Self Boot Engine – Chiplet Setup

3.1 proc_sbe_chiplet_setup : F,C,D -Nest Chiplet Reset

a p10_sbe_chiplet_setup.C (chip target)

- For all good chiplets including EQs
  - Reset PCB Slave to default state
- Setup static multicast groups for all good chiplets based on pervasive target functional state (not ATTR_PG state)
  - All EQs will be in a multicast group
    - Rest of istep 3 will enable all EQs and make use of EQ multicast group
    - Istep 3 wants the correct EQ group
    - Istep 4 wants the functional core subset for EQ to mimic QME boot flow
    - Post istep 5 – would like to restore EQ to "all", with the assertion that the power off cores give a "0" result cleanly on multicast/multihot for FSP usage
- Setup of hang counters including EQ
- Control power gate of nest units (based on PG and scratch register functionality)

3.2 proc_sbe_chiplet_clk_config : F,C,D -Nest Chiplet Reset

a p10_sbe_chiplet_clk_config.C (chip target)

- Setup the chiplet clock muxing
- Clocking: setup chiplet sector buffer strength, pulse mode and pulse mode enable

3.3 proc_sbe_chiplet_reset : F,C,D -Nest Chiplet Reset

a p10_sbe_chiplet_reset.C (chip target) FAPI2::ReturnCode

- For all good chiplets including EQs
  - Setup chiplet net control regs
    - Set chiplet enable on all all good chiplets
- For all enabled good chiplets including EQ
  - Start vital clocks and release endpoint reset
  - PCB Slave error register Reset
  - Scan 0 all rings on all enabled chiplets (except for TP, and the core portion of the EQs)

3.4 proc_sbe_gptr_time_initf: Init GPTR, Time rings for chiplets

a p10_sbe_gptr_time_initf.C

- Scan initalize all rings and initialize REPR on all enabled chiplets (except for TP, EP and EC)

3.5 proc_sbe_chiplet_pll_initf : PLL Initfile for PCIe, MCs, IOHS

a p10_sbe_chiplet_pll_initf.C

- PLL rings are stored in SBE image

- Included tune bits, frequency
- Includes issuing the set pulse

3.6 `proc_sbe_chiplet_pll_setup : Setup PLL for PCIe, MC, IOHS`

a p10_sbe_chiplet_pll_setup.C

- Start the PLLs
- Checks that the PLL locked

3.7 `proc_sbe_repr_initf : F,C,D -Chiplet Repair`

a p10_sbe_repr_initf.C (chip target) FAPI2::ReturnCode

- For all enabled chiplets
  - Load Repair, Time and GPTR rings for all enabled chiplets (except for TP, and the core portion of the EQs)
    - All chip customization data is within the Repair and Time rings – array repair, DTS settings

3.8 `proc_sbe_abist_setup : D -- Hook for DFT to run abist`

a No-op

3.9 `proc_sbe_arrayinit : Chiplet array init`

a p10_sbe_arrayinit.C

- Run arrayinit on all enabled chiplets
  - Scan flush 0 to all rings except GPTR, Time, Repair on all enabled chiplets (except for TP, and the core portion of the EQs)

3.10 `proc_sbe_lbist :D -- Hook for DFT to run lbist`

a No-op

3.11 `proc_sbe_initf : Apply any scan overrides`

a p10_sbe_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- Apply scan overrides to all enabled chiplets
- Generated via “traditional” initfile, but stored as compressed RS4 scan rings
- Spot to put all differences from scan flush 0
  - Intended only for config independent settings “patches”. Chip team goal is to flush to the correct state
  - Cannot contain system configuration differences, but can contain chip customization settings (ie DMI vs EDI personalization)
- Primary debug mechanism is to use Cronus/FSP putspy commandline to modify ring images directly in the chip (ie istep, then putspy).
  - Doesn’t cover core

- Need to know when in the IPL you can perform the scan ring
- Doesn't cover system test (ie non script/interactive mode)

♦ Secondary mechanism is to build an RS4 overlay and have a mechanism/location for the SBE to pick-up various overlays and apply

♦ Apply init file for chiplet IOs

3.12 proc_sbe_startclocks : Start PB and chiplet clocks

a p10_sbe_startclocks.C

♦ Drop fences and tholds on PB Chiplets
- Start nest chiplets with N1 as the master, rest as slaves

♦ Start Xbus, Obus, PCIe clocks

♦ Start clocks on configured chiplets

♦ EQ clock region do not have regions individual abistclk_muxsel and syncclk_muxsel
-

3.13 proc_sbe_chiplet_init : F,C,D -Chiplet Init

a p10_sbe_chiplet_init.C
- EQs
  - setup partial good vector for L3s, ELs, and MMAs
  - setup fabric group- unit- and sys-ids
  - set EQ chiplet ID bit

3.14 proc_sbe_chiplet_fir_init : Setup chiplet FIRs

a p10_sbe_chiplet_fir_init

♦ *Setup FIRs for various chiplets*

3.15 proc_sbe_dts_init : Setup chiplet DTS

a p10_sbe_dts_init

♦ Setup all the DTS (Digital Thermal Sensors) in the chip. Note that this can be done once for all DTS's as in P10 all DTS's maintain power through all STOP states (unlike P9)

3.16 proc_sbe_skew_adjust_setup : Perform skew adjust for chiplets

a p10_sbe_skew_adjust_setup.C

♦ This is a one time operation done on all EQ chiplets (for P9 this had to be done each STOP cycle, but P10 this logic is outside the core)

♦ Drop DCCs reset

- Setup 6 DCCs in parallel (commands over scan with setpulse, scan region = ANEP), dependency to VPD field #MK
- Drop DCCs bypass
- Additional DCC setup step (commands over scan with setpulse, scan region = ANEP)
- Drop SkewAdjust reset
- Setup Skewadjust (commands over scan with setpulse, scan region = ANEP)
- Drop SkewAdjust bypass
- Additional SkewAdjust setup step (commands over scan with setpulse, scan region = ANEP)

3.17 proc_sbe_nest_enable_ridi : Enable nest RI/DI

a p10_sbe_nest_enable_ridi.C

- Drop TP RI/DI for the AVS bus
- Drop RI/DI for TP logics
- Drop RI/DI for nest -- LPC and PSI IOs
-

3.18 proc_sbe_scominit : SBE Nest scominits

a p10_sbe_scominit.C (processor chip)

- Apply any scom inits to nest chiplets
- If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED
  - Tweak FIR Masks

3.19 proc_sbe_lpc : Init the LPC master

a p10_sbe_lpc_init.C

- Requirement from the bootloader is that it only uses MMIOs to LPC master, not Xscom
- Perform scoms to setup LPC bus
  - Move the LPC clock to external input
- Pull the LPC unit out of reset
- Set LPC BAR – hardcoded like Xscom BAR

3.20 proc_sbe_fabricinit : Init fabric(PB) for island mode

a p10_sbe_fabricinit.C

- Send fabric command and check result

- Chip will scan flush to SMP island mode

♦ This initializes P10 chip in “island” fabric mode and allows the core access to the PIB

♦ Pbus will flush to a state where all chiplets come up as good configured and disconnected – logic in powerbus respond to snoop with NULL response (traditional way of handling STOP)

3.21 proc_sbe_check_boot_proc : Determine if boot proc

a At this point the SBE must use the internal bolt-on register to toggle TPM Reset line

b Determine if this is master SBE

♦ SBE FW checks bit 24 of the Scratch register (stored in ATTR) –

♦ if set then this is a slave chip, load /enable runtime chipOps

♦ else master and continue

3.22 proc_sbe_mcs_setup : Setup MCS to allow EX contained

♦ This step needs to be a no-op on MPIPL

b p10_sbe_mcs_setup.C

♦ If

- ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED this step is a no-op

♦ Else:

- Open the MCS BAR to allow Hostboot to dcbz the contents of cache.
- Also disable speculative pre-fetch to prevent PBA reads from triggering operations to MCS

3.23 proc_sbe_select_ex : Select Hostboot core

a p10_sbe_select_ex.C

♦ New plan – not going to tear down EQ – only use the region enables

- Enable only the core region enables for istep 4/5
- Partial good registers remain untouched
- STOP exit will enable the core regions as normal

♦ FW will have correctly set the target functional state(s). HWP uses functional states as master record (doesn’t need to read PG data, gard, etc)

♦ Need to identify caches with good cores that yields 8MB with backing cache space

- Small/normal core mode -> 3/32 good normal cores (1 active, 2 backing)
- Big/fused core mode -> 2/16 good fused cores (1 active, 1 backing)

♦ If ATTR_SYS_FORCE_ALL_CORES is set

- then force select to ALL
  - Multicast groups are already setup by istep 3.1

- Else single “master core”.
  - Clear MC Reg 3 in all EQ chiplets
  - If ATTR_FUSED_CORE_MODE is not set (normal core mode)
    - Find the first 3 good normal cores
    - With the 3 good normal cores, determine which EQs contain those 3 cores (can be 1, 2 or 3 EQs
    - Add EQs to multicast group 6 vis MC Reg 3
  - Else (Fused Core Mode)
    - Find the first 2 good fused cores (the first 2 good paired normal cores, (eg 4 normal cores))
    - With the 2 good fused cores, determine which EQs containing those 2 cores (can be 1 or 2 EQs)
    - Add EQs to multicast group 6 vis MC Reg 3

♦ Write selected (single/all) Core mask into the OCC complex CCSR register for PGPE and OCC use

♦ This is the “master record“ of the enabled cores/quad in the system

♦ This is only for during the IPL (will be updated later in step 15)

4 Step 4 Self Boot Engine – EX Init

Note: Master chip (attached PNOR) inits EX unit for Hostboot execution image. Slave chips patiently wait.

♦ Issue isteps detailed in EQ and EC section
  - These are common to STOP images
  - Execution will return here afterwards

♦ Does NOT start instructions on core

Cache Initialization

This flow covers the steps that are used to initialize the Cache portion of the EQ chiplet. Although it is inserted in the mainline IPL flow, it is executed both in the IPL (to bring up the cache necessary for having the required HostBoot space) and for the QME during run-time.

All core and cache initialization is done using Multicast Group 6. The EQ chiplets that are to participate need to have that group number set into one of the PCBS Multicast registers prior to invoking this flow.

4.1 proc_hcd_cache_poweron : Cache Power-on

a p10_hcd_cache_poweron.C

-

♦ Clocking: Assert cache glsmux async reset

♦ Assert fences between Nest and L3s

♦ Command the cache PFET controller to power-on (VCS first, VDD second)
  - Check for valid power on completion; Polled Timeout: 100us

4.2 proc_hcd_cache_reset : Cache Reset

a p10_hcd_cache_ reset.C

- Assert cache PROGDLY bypass, DCC bypass (Is this from MVPD or constant)?
- Flip muxes to DPLL inputs
- Clocking: setup cache sector buffer strength, pulse mode and pulsed mode enable values (
- Clocking: Drop cache glsmux async reset)
- Assert SRAM enable
- Scan0 flush entire cache region

4.3 proc_hcd_cache_gptr_time_initf : GPTR and Time for EX non core

a p10_hcd_cache_gptr_time_initf.C

- Initfiles in procedure defined on VBU ENGD wiki to produce #G VPD contents
- Check for the presence of cache override GPTR ring from image
  - if found, apply; if not, apply cache GPTR from image
  - Check for the presence of cache override TIME ring from image;
    if found, apply; if not, apply cache base TIME from image

4.4 proc_hcd_cache_repair_initf : Repair ring for EX cache region

a p10_hcd_cache_repair_initf.C

- This HWP is run serialized per EQ (most others are done in multicast)
- Load cache ring images from MVPD
  - These rings must contain ALL chip customization data. This includes the following: DTS
  - Historically this was stored in MVPD keywords are #R, #G. Still stored in MVPD, but SBE image is customized with rings for booting cores

4.5 proc_hcd_cache_arrayinit : EX Initialize arrays

a p10_hcd_cache_arrayinit.C

- Use ABIST engine to zero out all arrays. Upon completion, scan0 flush all rings except Vital, Repair, GPTR, and TIME

4.6 proc_hcd_cache_initf :EX cache scan init

a p10_hcd_cache_initf.C

- Initfiles in procedure defined on VBU ENGD wiki Call putring on EQ rings
  - Putring checks for the presence of cache FUNC override/cache contained/risk level/etc rings from image;
  - if found, apply; if not, apply cache base FUNC rings from image

- Note: all caches that are in the Cache Multicast group will be initialized to the same values via multicast scans

4.7 proc_hcd_cache_startclocks : Cache Clock Start

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p10_hcd_cache_startclocks.C

- Set (to be sure they are set under all conditions) core logical fences
- Enable skewadjust and check for done; Timout: 100us (??)
- Clear clock controller BIST register, L3 region
- Call align_regions()
- Start L3 arrays + nsl regions
- Start L3 sl clock regions
- Check for clocks started
  - If not, error
- Clear force align
- Clear flush mode
- Drop the region fence for the L3 to allow PowerBus traffic
- Set region multicast en for the L3

4.8 proc_hcd_cache_scominit : Cache SCOM Inits

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p10_hcd_cache_scominit.C

- Apply any SCOM initialization to the cache
- Setup L3 configuration mode (LCO), includes back store customzation when in chip contained mode
- Configure Trace Stop on Xstop

4.9 proc_hcd_cache_scom_customize : Cache Customization SCOMs

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p10_hcd_cache_scom_customize.C

- Dynamically built (and installed) routine that is inserted by the “XIP Customization” process.
- Dynamically built pointer where a NULL is checked before execution
- If NULL (a potential early value); return
- Else call the function at the pointer; pointer is filled in by XIP Customization

- Customization items:
  - Epsilon settings scan flush to super safe
    - Customize Epsilon settings for system config
  - LCO setup (chiplet specific)
    - FW setups up based victim caches

From this point on, all data added to the image is for run-time modifications for STOP

4.10 proc_hcd_cache_ras_runtime_scom : EX Runtime Scom Init

a p10_hcd_cache_ras_runtime_scom.C

- Not consumed by SBE (empty istep); QME only
- Run-time updates from Host based PRD, etc that are put in the HOMER by STOP API calls
- Dynamically built pointer where a NULL is checked before execution
- If NULL (the SBE case), return
- Need two (2) separate sections – cache SCOM and L3 repair
- Else call the function at the pointer; pointer is filled in by STOP image build
  - Runtime FIR mask updates from PRD
  - L3 repairs
-

Note: this flow does NOT do anything with any of the cores attached to the caches that were just initialized. Also, this portion of the flow does also NOT initialize the QMEs in the EQ chiplet.

Core Initialization

This flow covers the steps that are used to initialize the Core regions in the EQ chiplet. It is covered prior to the mainline IPL flow as it is a separate image that is executed both in the IPL (to bring up the HostBoot core) and for the QME STOP execution. The running of this initialization flow REQUIRES the flow described in Cache Initialization to have been previously executed.

4.11 proc_hcd_core_poweron : Core Power-on

a p10_hcd_core_poweron.C

- Assert core GLSMux async reset
- Assert fences between core/L2 and L3
- Command the core PFET controller to power-on (VCS first, VDD second)
  - Check for valid power on completion; Polled Timeout: 100us

4.12 proc_hcd_core_reset : Core Reset

a p10_hcd_core_reset.C

- Assert core DC adjust reset
- Flip muxes to DPLL inputs
- Clocking: Drop core glsmux async reset

- Scan0 flush entire core region

4.13 proc_hcd_core_gptr_time_initf : Load Core GPTR and Time rings

a p10_hcd_core_gptr_time_initf.C

- Initfiles in procedure defined on VBU ENGD wiki to produce #G VPD contents
- GPTR is common between cores (ie multicast)
- Check for the presence of core override GPTR ring from image
  - if found, apply; if not, apply core GPTR from image
- Check for the presence of core override TIME ring from image;
  - if found, apply; if not, apply core base TIME from image

4.14 proc_hcd_core_repair_initf : Load Repair ring for core

a p10_hcd_core_repair_initf.C

- This step is run individually per core (serialized)
- Load core ring images from that came from MVPD into the image
  - These rings must contain ALL chip customization data. This includes the following: Array Repair and DTS calibration settings
  - Historically this was stored in MVPD keywords are #R, #G. Still stored in MVPD, but SBE image is customized with rings for booting cores at build time

4.15 proc_hcd_core_arrayinit : Core Initialize arrays

a p10_hcd_core_arrayinit.C

- Use ABIST engine to zero out all arrays
- Upon completion, scan0 flush all rings except Vital, Repair, GPTR, and TIME

4.16 proc_hcd_core_initf :Core scan init

a p10_hcd_core_initf.C

- Initfiles in procedure defined on VBU ENGD wiki
- Check for the presence of core FUNC override rings from image;
- if found, apply; if not, apply core base FUNC rings from image
- Note: FASTINIT ring (eg CMSK ring) is setup at this point to limit the stumps that participate in FUNC ring scanning
- Note : if in fused mode, both core rings will be initialized to the same values via multicast scans

4.17 proc_hcd_core_startclocks : Core Clock Start

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p10_hcd_core_startclocks.C

- Enable skew adjust and DCC; poll for skew adjust and DCC sync done;
- Clear clock controller BIST register, L3 region
- Call align_regions()
- Start core/L2 arrays + nsl regions
- Start core/L2 sl clock regions
- Check for clocks started
  - If not, error
- Clear force align
- Clear flush mode
- Drop the core to cache logical fence
- Set region multicast en for the core

4.18 proc_hcd_core_scominit : Core SCOM Inits

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b P10_hcd_core_scominit.C

- Apply any coded SCOM initialization to core

4.19 proc_hcd_core_scom_customize :Core Customization SCOMS

a If ATTR_SYSTEM_IPL_PHASE == CACHE_CONTAINED then skip (platform check)

b p10_hcd_core_scom_customize.C

- Dynamically built (and installed) routine that is inserted by the "XIP Customization" process
- Dynamically built pointer where a NULL is checked before execution
- If NULL (a potential early value); return
- Else call the function at the pointer; pointer is filled in by XIP Customization

From this point on, all data added to the image is for run-time modifications for STOP

4.20 proc_hcd_core_ras_runtime_scom : EX Runtime Scom Init

a p10_hcd_core_ras_runtime_scom.C

- Not consumed by SBE (istep is placeholder); QME only
- Run-time updates from Host based PRD, etc that are put on the core image by STOP API calls
- Dynamically built pointer where a NULL is checked before execution
- If NULL (the SBE case), return
- Need two (2) separate sections – core SCOM and L2 repair
- Else call the function at the pointer; pointer is filled in by STOP image build
  - Runtime FIR mask updates from PRD

- L2 repairs

Note: for IPL wakeup (eg istep 4), there is no explicit instruction start here. In Cronus mode, this is left up to the user; in FW, the instruction control is managed in step 5. However, for true STOP usage (istep 16 or run-time STOP 11), the QME blocks interrupts to core, puts HRMOR/URMOR in place to point the Core to HOMER and issues an SRESET to all threads. The threads go into HOMER and do SPR restoration. Then slave threads end at STOP level 15, master thread waits for slaves to complete (reach STOP15) then switches to "real" HRMOR/URMOR and other SPRs and then enters STOP15. When QME sees all threads at STOP15, it unblocks interrupts, which allows normal execution to commence.

5 Step 5 Self Boot Engine – Load Hostboot

Note: Master chip (attached PNOR) loads Host boot image. Slave chips patiently wait. Master SBE fetches Hostboot code from PNOR and places in target EX

5.1 proc_sbe_load_bootloader

a p10_pm_ocb_indir_setup_linear.C

- Setup OCB channel 3 to linear mode

b p10_sbe_load_bootloader.C

- Setup PBA to target specific cache (active L3, which will then flow out to the backing L3s as needed)
- SBE fetches bootloader, security algorithm, and hash of HW public keys from SEEPROM
  - SEEPROM Image is ECC protected
  - Each of the items above are independent (ie the key hash and bootloader, security code will want to be updated from HB independently), but from an SBE standpoint they are all contained within the bootloader xip section. They are NOT part of SBE xip customize image (but SBE knows how to find)
- Places bootloader at specific address
  - Hostboot HRMOR + 2MB + 12KB (if Hostboot HRMOR is128MB, then it would be 130MB for bootloader HRMOR)
  - Note that for Hostboot HRMOR == URMOR and P10 will come up with UV mode active (MSR[S] = 0b1)
  - Note that this is okay since during this phase Hostboot is NOT running out of SMF memory (ie bit 15 set in the Real Address)
- SBE creates POWER interrupt table (12K)
  - Done by SBE code because we don't want to waste 12K of SEEPROM space
  - First instruction is an branch absolute to 12KB
  - SBE places some communication area that contains things like the
    - LPC, Xscom BARs
    - HB memory footprint (2 back cores == 8MB, 4 == 16MB, etc). Based on number of backing cores configured by the SBE, SBE will report the max memory space for Hostboot
    - boot info and etc.
- SBE opens a read only memory window to the Hostboot address range (HRMOR/URMOR + 8MB)
  - In other words SBE Chip Ops won't let PBA/ADU traffic in until SBE receives a command to open the unsecure window from the host

- Note that the SBE will use PBA bar 2

5.2 proc_sbe_core_spr_setup

a p10_sbe_core_spr_setup.C

- SBE then inits various core SPRs as follows:
  - Master thread:
    - HRMOR/URMOR to the Boot loader start+ node address
  - Slave thread (including the fused core if enabled)
    - PSSCR on all slave threads to STOP15
    - HRMOR on fused code to Hostboot HRMOR + node address

5.3 proc_sbe_instruct_start

a p10_sbe_instruct_start.C

- Start instructions on active core, one thread
- Thread 0 will be started at CIA scan flush value of 0x0
  - With URMOR/HRMOR this is address 130MB
- Instruction start on one core, one thread. After executing this istep the SBE will load its runtime ChipOps

6 Step 6 Hostboot – Master Init, discovery

6.1 host_bootloader (non-steppable istep)

- Boot loader needs the following information:
  - LPC base address
  - Xscom base address
- Perform any LPC setup (via MMIO only)
- Boot loader finds the FFS partition table in PNOR, locates the HBB partition
- Performs dcbz of working space:
  - HBBL HRMOR -2MB and +2MB (excluding the actual HBBL region)
  - Loads HBB w/ECC to secure memory (4MB relative to HB HRMOR)

==========

Remove ECC to secure memory (5MB relative TBD to get right offsets)

- Uses signature validation code to validate (@ 5MB relative)
- Copy down verified image to 128MB
- Copy down security algorithm, hash of the HW keys, HBB header
- Sets up the bootloader communication area that contains SBE information and addition secureboot info

- Starts executing at 128MB (sets HRMOR and jumps)
- If any of the above steps fail – bootloader will checkstop the system

6.2 host_setup (non-steppable istep): Setup host environment

- If in secure boot the bootloader has already validated image
- Select primary thread (only thread running)
- Dcbz in the Hostboot memory footprint based on information passed up from bootloader (source is SBE information)
- Initial setup
  - stacks
  - MSR
  - execution environment
  - Thread control structures
  - Memory Management setup
- Ready for execution
  - Tracing
  - Device Drivers
    - Xscom (Scom)
    - Mailbox (Scom)
    - I2C (Scom)
    - LPC
    - FSI (Scom)
- At this point the HWPF is alive and active

b p10_thread_control.C

- Honors the scratch reg thread control (SMT 1/2/4)
- If in big core mode then threads are started across 2 small cores to keep PIR it structures in HB the same
- Start and release all other threads on core(s) (1-3)

- Hostboot will pull appropriate scratch register data and write into ATTR
- HB mechanism to read/write to PNOR
  - Hostboot (BMC) writes LPC -> IPMI or PLDM to access BMC memory directly for PNOR
  - Hostboot (FSP) writes LPC ↔ SPI NOR controller to read/write
  -

6.3 host_istep_enable (non-steppable istep): Hostboot istep ready

- Hostboot checks scratch registers for istep attribute, if set Hostboot “halts” and waits for commands from

SP and/or Cronus

- Only isteps after this point can be issued to Hostboot
- At this point communication can be performed with the SP

6.4 host_init_fsi : Setup the FSI links to slave chips

- It is expected that the following steps have already been done by SP – Hostboot will just use FSI bus
  - Configure FSI master (HUB and Cascade)
  - Send break commands to FSI slaves
  - Configure the slaves
  - Force lbus
- Setup Scom device drivers
  - Read ID/EC levels
- Reset all I2C engines/slaves on the P10 Master Chip and all FSI I2C Masters (P10 slaves)
  - Can't reset the scom only I2C master on the P10 Slave chips (see 8.4)

6.5 host_set_ipl_parms : Build ipl parameters

- Sets the IPL parameters for this boot

6.6 host_discover_targets : Builds targeting

- Determines what targets are present and functional -- factors in the PG keyword and MRW constraints. And what is connected (memory buffers, processors installed)
  - Does not look at SBE scratch regs, require a cross between MRW vs actual scratch regs in istep 7 to ensure that HB doesn't use parts that are powered off (need to ensure post istep 7 where the master is reconfigured)
  - Parts that are turned off == PG (ie just don't show up to customer)
- This is the step where the host "configures" itself and builds its present/functional map of the targets
  - Uses FSI presence to detect processors
  - Reads dimm VPD from PNOR/I2C to determine what dimms and OCMBs are present
- For OpenPower systems Hostboot will push the FRU inventory to the BMC
  - Must push for all present parts (for eBMC portions of the VPD may already be on the BMC and not pushed down)
  - Must update FRU present/functional state

6.7 host_update_primary_tpm : Update the Primary TPM

- Currently a no-op. The contents of this istep were moved to the end of host_gard for P10.

6.8 host_gard : Do Gard

- No OCC xstop analysis in the host (use OpenBMC xstop analysis)
- Apply repeat-gard records and deconfigure hardware

- Initialize PRD
- At the end of this step ATTN/PRD will start polling for errors on the master chip
- If redundant TPM this step must enforce that master/alt-master use their local respective TPM
  - If the master proc's TPM is not functional, force a reboot to the Alt Master
- Extend TPM with measurements and configuration data
  - Hash of HW public keys, HBB, HBI, etc
- Perform physical presence detection

6.9 host_voltage_config : Calculate correct chip voltage

- This step will compute and store the various system frequencies and voltages – specifically the powerbus and core frequency based on MRW wattage/powerbus frequency settings
- The programmable voltages for each P10 socket in the system (VCS, VDN, VDD) will also be calculated. The VDN, VCS, VDD rails are always on the AVS bus because the OCC needs to dynamically manipulate for Workload Optimized Frequency. VIO is **not** controlled via AVS (don't expect to need to used dynamically, but may need to be done via FSP/BMC – Eric/Bjorn question)

b p10_setup_evid.C (COMPUTE)

- Use VPD backed attributes (from #V) to calculate VDD, VCS and VDN for this socket
- These need to be stored to ATTR_*_VAL (VCS, VDD, VDN)

- Note that none of the settings are written to hardware – this is done later in the boot.

7 Step 7 Hostboot – MC Config

Note that the "FW Reconfig" loop starts here (since it doesn't touch HW). Any reconfig during step 7 will loop back to this step

7.1 host_mss_attr_cleanup : Spot to clean up ATTR

a p10_mss_attribute_cleanup.C (list of all mcs)

- Called on all present memory buffers (Nimbus and Centaur)
- Hook to clean up attributes on reconfig loop (set to known state) if needed
- Empty – just a placeholder

7.2 mss_volt : Calc dimm voltage

a p10_mss_volt.C (list of functional memports on same voltage rail)

- Consumes SPD to determine valid voltages for installed parts
- Calculate rail Voltage and updates rail system attribute
- Save settings in ATTR

- Procedure handles checking overrides

7.3 mss_freq : Calc dimm frequency

a p10_mss_freq.C (functional memport)

- Consumes SPD to determine the valid frequency for installed parts
- Looks at voltage and dimm functionality
- Takes a system ATTR that defines the allowable dimm frequencies for the system
- Calculate per memory controller frequency from attributes – picks the frequency bucket to use
- Save settings in ATTR
- Procedure handles checking overrides

b p10_mss_freq_system.C (all functional mcbists)

- Determine the required OMI frequency to handle the chosen dimm frequency
  - OMI 25.6 -> DDR 3200
  - OMI 23.46 -> DDR 2933
  - OMI 21.33 -> DDR 2666
  - Cronus will output error and stop if freqs don't match

7.4 mss_eff_config : Determine effective config

a p10_mss_eff_config.C (memport target) -- loop over all functional memports

- Consumes SPD/VPD to determine proper memory configuration supported by this system

b Memory buffer Effective Thermal

- Consumes SPD/VPD and system policies to determine thermal properties
  - ***exp_mss_eff_config_thermal.C (memport target)***
  - ***ody_mss_eff_config_thermal.C (memport target)***

c p10_mss_eff_grouping.C (proc chip target) – loop over all functional procs

- Called on each P10 target, maps memory behind each chip
- Calculates attributes defining the ultimate memory controller BAR settings
- Calculates attributes defining MCB BAR settings
- Accounts for SMF settings
- Does not actually apply any settings to the HW at this point

7.5 mss_attr_update :MSS ATTR Overrides

a *p10_mss_attr_update.C*

- Called per MC
- Stub HWP for FW to override attributes programmatically

b Examine incoming mbox scratch regs versus MRW/Calculated via ATTR

- Specifically the PLL Buckets/frequency, boot modes, etc. This does not include istep, mpipl, risk level modes. Discretion on exact sracth values checked left up to Hostboot team.
  - p10_xip_customize.C

8 Step 8 Hostboot – Nest Chiplets

8.1 host_setup_sbe

a *p10_set_fsi_gp_shadow.C*

- Done for all boots – some settings will change based on system type and IPL type
- Set the GP bits to default state
- Needs to account for changed values set up in p10_setup_ref_clock.C procedure

8.2 host_secondary_sbe_config

- Need to run this from master processor to all slave processors for Secureboot hole (need to ensure that SP didn't leave compromised P8 Slave.
- Looked at doing this earlier, in parallel, but need to know OMI frequencies to select buckets for SBE

b *p10_setup_sbe_config.C*

- Update SBE config data area with any configs/parameters required by SBE (see step 0 for more details)
- This includes the nest (and memory frequency if in synchronous mode)
- Configuration flags (MPIPL, etc)

8.3 host_cbs_start

a *p10_start_cbs.C*

- Set a bit to start the SBE engine on master chips. Located in FSI GP region
- This same bit performs the scan0 flush of pervasive

8.4 proc_check_secondary_sbe_seeprom_complete :Check Secondary SBE Complete

- Check to make sure that the slave SBE engines have completed their IPL
- FW will poll for up to 1 second to see if the “done” signature is in the status reg (not tied to istep number)

- If “done” signature is not found, then FW must extract FFDC from the SBE

b p10_get_sbe_msg_register.C

- Read the SBE state reg

c p10_extract_sbe_rc.C -soft_err

- Called on slave chips to look for any correctable errors on the PNOR and/or SEEPROM
- The soft_error flag just tells the procedure to not generate an error if no HW issue

d Reset all scom only I2C engines/slaves on the P10 Slave Chips

8.5 `host_attnlisten_proc : Start attention poll for P9(s)`

- Enable hostboot to start including all processor attentions in its post istep analysis
- From this point on ATTN/PRD will listen (“poll”) for powerbus attentions after each named istep

8.6 `proc_fbc_eff_config : Determine Powerbus config`

a p10_fbc_eff_config.C (None)

- Sets system wide attributes derived from MRW and system topology for current HB powerbus scope (X & A busses)
  - Epsilon settings
  - Processor floor frequency
- Does not access the HW

8.7 `proc_eff_config_links : Powerbus link config`

a p10_fbc_eff_config_links.C (None)

- Processes link related attributes for current HB powerbus scope (X & A busses)
- Does not access HW

8.8 `proc_attr_update :Proc ATTR Update`

a p10_attr_update.C

- Called per processor
- Stub HWP for FW to override attributes programmatically

8.9 `proc_chiplet_fabric_scominit : Scom inits to fabric chiplets`

a p10_chiplet_fabric_scominit.C

- Initfiles in procedure defined on VBU ENGD wiki
- Apply scom overrides to all chiplets necessary to init the powerbus. Note that this does not cover any PHY inits, just the DL/TL layers
  - p9.fbc.no_hp.scom.initfile
  - p9.fbc.ioe_dl.scom.initfile

- p9.fbc.ioe_tl.scom.initfile
- p9.fbc.ioo_dl.scom.initfile
- p9.fbc.ioo_tl.scom.initfile

8.10 host_set_voltages : Set correct chip voltage(s)

- This step will apply the voltages calculated earlier in the IPL. It is done here so all chips can be programmed at one spot.

b p10_setup_evid.C (APPLY_AVS)

- Via the AVS bus the HWP will program always program VDN, VCS, and VDD. The specific combination of AVS bus and rail select are indicated by ATTR_*_BUS_CTL (which AVS bus) and ATTR_*_BUS_SELECT (which select).
- VIO is handled by the service processor

8.11 proc_io_scominit : Apply scom inits to iohs

a p10_iohs_scominit.C

- Only touches iohs that are being used for powerbus inside the drawer

b p10_io_omi_scominit.C

- Only touches iohs that are being used for powerbus inside the drawer

8.12 proc_load_ioppe : Load IO PPE images to their SRAMS

a p10_io_load_ppe.C (image bin, size)

- Extend SBE putsram chipOp to use multicast to all IO PPE engines
- Master proc will use Xscoms directly, Slave proc will receive data across SBE FIFO as a chipOp
- 4 IO PPEs per chip (one each of the physical PAU chiplets)
- Image is identical for all instances
  - Image is stored HW reference image. Hostboot responsible for extracting and passing to HWP

8.13 proc_iohs_enable_ridi : Enable RI/DI for iohs

a p10_iohs_enable_ridi.C

- Drop RI/DI for iohs chiplets being used

b p10_chiplet_enable_ridi.C

- Drop RI/DI for all chiplets being used (A, O, PCIe, OMI)
- Any other chip wide RI/DI

8.14 proc_init_ioppe : Start and init IO PPE

a p10_io_init_start_ppe.C () (chip target)

- Write link specific data to the SRAM for each IO PPE (using target specific putsram)

- HWP will create a multicast group of ioPPEs
- Address to write to in header of the IO image

♦ Extend SBE putsram chipOp to use multicast to all IO PPE engines

♦ Master proc will use Xscoms directly, Slave proc will receive data across SBE FIFO as a chipOp

♦ For each IOPPE custom data (32 vs 50Gbps) SRAM or abstracted via SBE (ie remove details of SRAM from PHYP/OPAL)
- Need to know speed and Mode (Powerbus, NVLink, OCAPI) – at boot time we will know mode (
- OMI would not need to change (might need to reserve capability)
- ioPPE targets must have relationship with associated iohs and omi/mc targets it controls

♦ Kick off init and iodcal automatically at this point – will do for all iohs AND OMI – plan is to leave this running in the background

9 Step 9 Hostboot – EDI+ and Electrical O-Bus Initialization

9.1 proc_io_dccal_done : Ensure the IO PPE has finished iodcal

a p10_io_init_done.C (list of IOPPE)

♦ Wait for iohs and omi init and dccal to be done

9.2 fabric_dl_pre_trainadv : Advanced pre training

a p10_fabric_dl_pre_trainadv.C (called on each iohs target pair))

♦ Debug routine for IO Characterization

♦ Nothing in it

9.3 fabric_dl_setup_training : Setup training on internal node buses

a p10_fabric_dl_setup_linktrain.C (called on each iohs target pair)

♦ Hostboot will setup training on all intra node buses. The iohs target is passed in and the PDL region is used to perform the training.

♦ Is there swizzling of lanes to PDL? Handled via ATTR

♦ Repairable fails are left for PRD to analyze and perform appropriate error logging

♦ Fatal bus training errors are handled by procedure, must return error and FFDC

9.4 proc_fabric_link_layer : Start SMP link layer

a p10_fabric_link_layer.C (called on processor chip)

♦ Reads logical iohs link configuration attributes, trains the DL/TL layers of selected links

♦ Set scom on both sides of the bus to trigger Data link layer training

- DLL sends training packets, sets link up FIR bit when done
- FIR done bit launches the Transaction Layer (TL)
- FIR bit in nest domain to indicate training done
- After this point the mailbox register are available to communicate
  - Xstop would prevent mailbox communication
- Bus is NOT part of the SMP coherency
- Only performed on setup, valid buses
- The link between the iohs and IOPPE will be used for FFDC, but most interactions are handled by HW

9.5 fabric_dl_post_trainadv : Advanced post training

a p10_fabric_dl_post_trainadv.C (called on each iohs target pair)

- Debug routine for IO Characterization
- Nothing in it

9.6 proc_fabric_iovalid : Lower functional fences on local SMP

a p10_fabric_iovalid.C (chip target)

- Reads logical fabric link config, sets iovalid for selected links
- Only performed on trained, valid buses
- After this point a checkstop on a slave will checkstop master
- Reads the fabric link delays for later HWP to pick best link for coherent traffic

9.7 proc_fbc_eff_config_aggregate : Pick link(s) for coherency

a p10_fbc_eff_config_aggregate.C (chip target)

- Reads attributes from previous HWP and determines per-link address/data capabilities
- Sets up attributes for build SMP

10 Step 10 Hostboot – Activate PowerBus

10.1 proc_build_smp : Integrate P10 Islands into single SMP

a p10_build_smp.C (vector of all chips to include in SMP, SMP_ACTIVATE_PHASE1)

- Look for checkstops
- Use the fabric concurrent maintenance operation to merge P10 PB islands into the SMP
- Fabric config between IO/CAPI are set here – only can set once, must be known by this point in time
- After this point the SMP is built for normal mode

- Runs initfiles to set current/next values for full config in slaves, setup master next value
  - p9.fbc.ab_hp.scom.initfile

b Halt all secondary SBEs

c p10_build_smp.C (vector of all chips to include in SMP, SMP_ACTIVATE_SWITCH)

- Trigger fabric quiesce/switch/init on the master

d p10_build_smp.C (vector of all chips to include in SMP, SMP_ACTIVATE_POST)

- Set fabric hotplug configuration registers after switch
- Enable dlr after activating new config

10.2 host_sbe_update

- On systems that support Alt Master Processors then code will attempt to read the TOC of the Alt Master PNOR to check for connection problems. If an error is detected it will be logged, but this does not stop the IPL (except when in manufacturing mode)
- Hostboot must update SEEPROM because the SP cannot because of secureboot. It is at this step in the IPL so it can be updated via Xscom (trusted path) on all chips in the system

b p10_ipl_customize.C

- If needed build a custom SEEPROM image for each chip in the system off of the base IPL SEEPROM image
- This set will update all SEEPROM images in the HB "node". All needed attributes are written from the host into the SBE image via this HWP.
- In addition if the override section from the PNOR is not empty then it needs to be appended to the SBE image prior to customization.

- If the SEEPROM was updated then Hostboot will request a reipl at this point

10.3 host_secureboot_lockdown : TPM policy flags, prevent SEEPROM updates

a Hostboot will check for TPM policies are valid, secureboot consistency between procs

b p10_disable_ocmb_i2c.C

- Disable ocmb i2c access from any PIB master except SBE for engine A (SUL)

c p10_update_security_ctrl.C

- This HWP will set the SUL security bit so that SBE image cannot be updated
- This will also make the SAB security bit read only
- If a TPM is non functional, set the TDP (TPM Deconfig Protection) to prevent attack vector

d Halted SBEs are awakened via HRESET

10.4 proc_chiplet_scominit : Scom inits to all chiplets (sans core)

a p10_chiplet_scominit.C

- Trigger SMP C/D switch for fabric performance. Mode C is safe boot values, mode D is performance optimized settings based on memory frequency - customized into SBE init settings via Dyanmic inits
  - Placed here in the flow because it must occur after SBE customize.
- Initfiles in procedure defined on VBU ENGD wiki
- Apply scom overrides to all good chiplets (except EX and MC)
  -

b p10_psi_scominit.C

- Each instance of bus must have unique id set for it – personalize it
- Must set present and valid bits based on topology (Attributes indicate present and valid)

10.5 proc_pau_scominit : Apply scom inits to PAU

a p10_pau_scominit.C

- Each instance of OCAPI bus must have unique id set for it – personalize it
- Check the handle IOHS_CONFIG_MODE == OCAPI on all IOHS and mux PAUs correctly
  - IOHS0, 3 – use respective PAU 0,3
  - IOHS1,2 – error
  - IOHS 4,5 – can swizzle between PAU 4,5 -- will use logical PAU target to know which one is functional
  - IOHS 6,7 – can swizzle between PAU 4,5 -- will use logical PAU target to know which one is functional

10.6 proc_pcie_scominit : Apply scom inits to PCIe chiplets

a p10_pcie_scominit.C

- Note that the PHY must be held in reset (CPTL_CTRL0[55]) while scominits are applied
- Initfiles in procedure defined on VBU ENGD wiki
- Perform the PCIe Phase 1 Inits 1-8
  - Sets the lane config based on MRW attributes
  - Sets the swap bits based on MRW attributes
  - Sets valid PHBs, remove from reset
  - Performs any needed overrides (should flush correctly) – this is where initfile may be used
  - Set the IOP program complete bit
  - This is where the dSMP versus PCIE is selected in the PHY Link Layer

10.7 proc_scomoverride_chiplets : Apply sequenced scom inits

a p10_scomoverride_chiplets.C

- Apply any sequence driven scom overrides to chiplets – Should be NONE

10.8 proc_chiplet_enable_ridi : No-op

- No-op for P10 (moved to istep 8)

10.9 host_rng_bist : Trigger Built In Self Test for RNG

a p10_rng_init_phase1.C

- Trigger the Random Number Generator Built In Self Test (BIST). Results are checked later in step 16 when RNG is secured

11 Step 11 Hostboot OCMB Init

11.1 host_prd_hwreconfig : Hook to handle HW reconfig

- This step is always called
- Move all OCMB access back to I2C
- Call PRD to allow them to rebuild model to remove non-functional OCMBs
- Used for HW reconfig path. FW's strategy is to perform the reconfig on ALL functional Centaurs/MCS's in the system.
- The following procedures must be called:
  - ***p10_enable_reconfig.C (MCS, OMI)***
    - Necessary cleanup for a reconfig loop on the MCS/OMI interface

11.2 host_set_mem_volt : Enable voltages on the DDIMMS

a pmic_enable.C (ocmb target)

- Will always disable the PMIC and re-enable voltages on the DDIMM

11.3 proc_ocmb_enable : Release reset, start clocks to OCMB

a p10_ocmb_enable.C

- Enable the ref clocks out to all functional OCMBs (unique enable per sub-channel)
- Release reset out to OCMBs (single reset pin per processor for all attached OCMBs)
- OCMB firmware will boot into a standby mode
  - Performs SHA hash on firmware load and pushes result out to secure register

11.4 ocmb_check_for_ready : Check that OCMB is ready

a exp_check_for_ready.C

- EXP_FW_STATUS command sent to Explorer via i2c from processor
  - Will clock stretch i2c and/or return NACK until chip firmware is ready
  - Expected boot time ~200ms, upper limit of 1s

b exp_getidec.C

- Read IDEC from chip registers

c Verify firmware signature

- Read Explorer firmware signature to save for later (note this is insecure)

12 Step 12 Hostboot – OMI Training

12.1 mss_getecid : Read out ECID of all OCMBs

a exp_getecid.C

- Read the ECID for each explorer and store away for callouts.
- Decode ECID and set other ECID related attributes for later operations on Explorers

b p10_init_mem_encryption.C

- This HWP will enable memory encryption
- Must be done prior to any OMI traffic on the OMI bus

12.2 omi_attr_update : Update DMI related attributes

- no-op

12.3 proc_omi_scominit : OMI Scom setup on Cumulus DMI

a p10_omi_scominit.C (OMI)

- Perform scom inits for OMIs on the processor. This is a place holder, expecting IOPPE to already have performed

b p10_omi_setup_bars.C

- Configure P10 BARs to allow config space access and any other setup required before dccal

12.4 ocmb_omi_scominit : OMI Scom setup on Centaur

a No-op at the moment. Would be a hook point for explorer/odyssey specific inits

12.5 omi_pre_trainadv : Advanced pre-OMI training

a ***p10_io_omi_pre_trainadv.C (OMI/ OCMB pair)***

- Currently empty
- Debug routine for IO Characterization

12.6 omi_setup : OMI setup for training

a ***exp_omi_setup.C***

- Set any registers (via i2c) on the Explorer before OMI is trained
  - Fuse overrides to setup credits, etc

b ***p10_omi_setup.C***

- Setup any registers on P10 side in prep for training
- Needed for various explorer workarounds (setup PRBS)

12.7 omi_io_run_training : Run training on MC buses

a ***p10_omi_train.C (OMI)***

- Turn on P10 config regs to enable training

b ***exp_omi_train.C***

- Send EXP_FW_BOOT_CONFIG command to Explorer via i2c with configured OMI frequency
  - OCMB firmware initializes PHY, sets CFG_DL0_train_mode to enable training, checks for errors
- HWP checks status of command and logs errors appropriately
- Note - If OMI does not train and Explorer firmware is not up to date, Hostboot will update the firmware over i2c

12.8 omi_train_check

a ***exp_omi_train_check.C***

- Check explorer for training errors

b ***p10_omi_train_check.C***

- Check P10 for training errors

12.9 omi_post_trainadv

a ***p10_io_omi_post_trainadv.C (OMI/Centaur pair)***

- Currently empty
- Debug routine for IO Characterization

12.10 host_attnlisten_memb : Start attention poll for membuf

- Currently empty
- Expand Host PRD to include memory buffers (as well as powerbus)
  - Tell the SBE that the OCMB are alive and active.

12.11 host_omi_init : Init the OMI protocol, Set the Inband base addresses

a exp_omi_init.C

- Initialize config space on Explorer, e.g. templates, rates, internal BARs, interrupts

b p10_omi_init.C

- Initialize config space on P10, e.g. templates, rates, interrupts

c ALL ACCESSES from this point on in are Inband access for Explorer unless otherwise specified

12.12 Update_omi_firmware : Set the Inband base addresses

a Firmware update of Explorer

- Securely read hash value, if different than available image perform inband code update

b exp_fw_update.C

- Send EXP_FW_BINARY_UPGRADE to Explorer FW

c Restart IPL at istep 0 if update performed

13 Step 13 Hostboot – DRAM Training

13.1 mss_scominit : Perform scom inits to MC and PHY

a exp_scominit.C

- Configure MC logic in the OCMB

b p10_throttle_sync.C

13.2 mss_draminit : Dram initialize

a exp_draminit.C

- Send EXP_FW_DDR_PHY_INIT command to Explorer via MMIO
  - Parameters come from SPD and system policies
  - Will include known bad bits/nibbles based on previous errors
- Explorer FW initializes PHYs and DRAMs

- Default to safe mode throttles
- All enabled bits/nibbles will be tested and overall results returned
- Real MRS values are returned

♦ HWP examines data, handles errors, stores away MRS data for later

♦ At the end of the step, the following is true:

- DRAM reset is de-asserted
- DRAM clocks are running
- DRAM is Self-Refresh (CKE is low)
- RCD control words are loaded into the hardware
- MRS values are loaded into the hardware
- External ZQ calibration is complete
- Leveling and training complete

b Check for attentions (even if HWP has error)

♦ FW -- Call PRD

- If finds and error, commit HWP RC as informational
- Else commit HWP RC as normal

- Trigger reconfig loop is anything was deconfigured

13.3 mss_draminit_mc : Hand off control to MC

a exp_draminit_mc.C

- Initialize the MC logic in the OCMB

♦ Start main refresh engine

♦ Refresh, periodic calibration, power controls

♦ Turn on ECC checking on memory accesses

♦ Deploy row repairs

♦ Note at this point memory FIRs can be monitored by PRD

14 Step 14 Hostboot – DRAM Initialization

14.1 mss_memdiag : Mainstore Pattern Testing

- The following step documents the generalities of this step
  - In FW PRD will control mem diags via interrupts. It doesn't use mss_memdiag.C directly but the HWP subroutines
  - In cronus it will execute mss_memdiags.C directly

b exp_mss_memdiag.C (ocmb)

- Executed by sending scoms to the OCMB's mcbist engine
- Modes:
  - Minimal: Write-only with 0's
  - Standard: Write of 0's followed by a Read
  - Medium: Write-followed by Read, 4 patterns, last of 0's
  - Max: Write-followed by Read, 9 patterns, last of 0's
- Run on the host
- This procedure will update the bad DQ attribute for each dimm based on its findings
- At the end of this procedure sets FIR masks correctly for runtime analysis
- All subsequent repairs are considered runtime issues

14.2 mss_thermal_init : Initialize the thermal sensor

a exp_mss_thermal_init.C

- List of sensors come from SPD
- Sensors are manually configured via inband EXP_FW_TEMP_SENSOR_PASS_THROUGH_READ/WRITE commands to Explorer
- Cache polling is setup via inband EXP_FW_TEMP_SENSOR_CONFIG_INTERVAL_READ command to Explorer
- Setup emergency throttles and disable safe mode throttles
  - When OCC starts polling OCC cache will revert to runtime settings

b p10_throttle_sync.C

- Must be issued on all P10s, can only be issued after ALL OCMBs on given p10 have thermal init complete
- Triggers sync command from MI to actually load the throttle values into the OCMBs

14.3 proc_load_iop_xram : Load PCIe IOP Xram

a p10_load_iop_xram.C(proc chip, ptr to HCODE image)

- Deassert PHY reset
- Check for sram_init_done
- Called on all chips, loads all PCIe chiplet xrams (4 per PCIe chiplet)
  - Image is extracted from the HCODE
  - Multicast to both PCIe chiplets at a time thru all 4 SRAMs

14.4 proc_pcie_config : Configure the PHBs

a p10_pcie_config.C

- Called on all chips, target is per PHB
- Procedural based – will call initfile if need be
- Covers PCIe Phase 2 Inits 18-30
  - Setup config regs
  - Command and Data credits
  - Clear FIRs (if needed)
  - Unmask PCIe FIRs

14.5 proc_setup_mmio_bars : Setup MMIO BARs

a p10_setup_mmio_bars.C (proc chip)

- Program unit MMIO BARS (fsp/psi/pau/int...)
- Memory controller/MCD BAR programming will be handled by the SBE for p10

14.6 host_secure_rng : Secure the random number

a p10_rng_init_phase2.C

- This HWP will check the result of the Random number generator (RNG) diagnostics
- It will also set the RNL security bit to prevent the RNG from being reprogramed via Xscom by the hypervisor

b p10_disable_ocmb_i2c.C

- Disable ocmb i2c access from any PIB master except SBE for engines B,C,E (SOL)

14.7 host_enable_memory_encryption : Enable memory encryption

- Hostboot will write the memory encryption keys out to the memory controllers. Note the HW mechanism is enabled in istep 12.1 – this step just sets the keys
- Prior to doing this hostboot needs to setup the DARN infrastructure and obtain a random number for this HWP to use
- This needs to be done prior to exit cache contained for the cache contents to be written to memory

14.8 proc_exit_cache_contained : Execution from memory

a p10_exit_cache_contained. C

- For P10 they exit cache contained procedure is fairly involved (unlike on P9). When switching to memory the backing caches need to be "shutdown" and flushed to memory. In addition various other inits need to be changed to take the backing caches out of contained mode, specifically the memory config register for the system. All of this must be done with no traffic pending on the powerbus.

- In order to do this in a controlled sequence, the SBE will be used to perform most of the low level mechanics at the direction of Cronus/Hostboot.

- The main p10_exit_cache_contained.C HWP will build a list of inits (fully formed Xscom operations to setup the MC, MCD BARS). Once this is done it will call p10_sbe_exist_cached_contained on the SBE to do all the heavy lifting:
  - ***p10_sbe_exit_cache_contained.C***
    - Invokes low level HWPs to accomplish cache contained exit sequence (each step executed by unique HWP routine)
    - ***p10_sbe_stop_hb.C***
      - Stop instructions on all threads on set of active cores
    - ***p10_revert_sbe_mcs_setup.C***
      - Restore MI flush state on instance servicing HB dcbz
      - Shifted master chip, MPIPL checks into p10_sbe_exit_cache_contained
    - ***p10_sbe_setup_memory_bars.C***
      - Apply set of passed in (from Hostboot/Cronus) XSCOM inits via ADU
    - ***p10_sbe_purge_hb.C***
      - Flush HB cache footprint and adjust configuration of active, backing caches for memory contained execution
    - ***p10_sbe_instruct_start.C***

- Restart execution on master core, all threads P10_sbe_exit_cache_contained.C.
  - Data rolls out to memory

14.9 `proc_htm_setup : Setup HTM allocations`

a p10_htm_setup. C

- Setup any BARs and inits to enable hardware in memory trace

14.10 `host_mpipl_service : Perform MPIPL tasks`

- This is a no-op for warm/cold IPLs. See description in REF LOC for full details

14.11 `proc_psiinit : Enable PSI for CRONUS ONLY`

a p10_psi_init.C (on 1 functional link)

- Set FSP PSI Frequency and other mode bits

- Call adal_psi_init on FSP

- Raise IOVALID on both sides of the bus

- Enable the link on p10
- Call adal_psi_link_enable on FSP
- Verify link is active

14.12 proc_bmc_pciinit : Enable PCIe for BMC for CRONUS ONLY

a p10_phb5_init.C (on 1 functional link)

- Enable the PCIe Link for BMC traffic
- Used for DMAing into/out of Hostmemory to the BMC, 1 partionable endpoint
- Loads a specific set of ATTR per system
  - Requires host memory for some tables/routing. Need to know size and alignment
    - Need to reserve space for these (ie like HOMER)
    - ATTR_PROC_PHB_RTT_BASE_ADDR u64 0x8000000001c00000
    - ATTR_PROC_PHB_PELTV_BASE_ADDR u64 0x800000001e000000
    - ATTR_PROC_PHB_PEST_BASE_ADDR u64 0x800000001f000000
  - Cronus would have a ATTR file for Rainier/Everest based on system topology with the "right" ATTR (or hardcoded into C code?)

b p10_cfg_bmc_endpoint.C (on 1 functional link)

- Perform link check (see if it is alive and active)
- Perform PCIe CFG operations to setup the BMC for use for
  - Done via get/putmemproc -ci
  - DMAs and host access
  - Current only for MMIO ops and DMA, no interrupt support / no translation

15 Step 15 Hostboot – Build STOP Images

15.1 host_build_stop_image : Build runtime STOP images

a p10_setup_runtime_wakeup_mode.C

- Call this HWP to setup which mode (UV vs HV) istep 16 exits from. This HWP will use SMF_ENABLE ATTR to determine if it will exit in UV mode or HV mode

b Pull necessary information from PNOR / MVPD for building HOMER image

- HCODE (HW reference image)
- MVPD
- OCC (new for P10)
- WOF Data
- Run through secure boot algorithm

- P10 idea is to build a fully functional HOMER including all OCC/pstate information and ONLY start the QMEs for istep 15/16, and then instead of rebuilding in istep 21 – just clear/rebuild runtime sections and start necessary engines
    - Save on OPAL IPL boot time
    - Single common code path (reduce complexity for poweron)

c p10_hcode_image_build.C (void* reference_image, void* v_homer_region, enum image_bld) FAPI2::ReturnCode

- HOMER – Hardware Offload Microcode Engine Region
- Called for each processor chip.
- Parameter: Pointer to Reference image.
- Parameter: Pointer to Output HOMER location (virtual address). The procedure places the respective images (eg PGPE, QME, XGPE) into HOMER at the appropriate offsets
    - This is any Hostboot specified mainstore location (does not have to be attached to the processor being STOPped).
    - When PHYP is loaded, the HOMER region will be trampled, PHYP will call ***p10_hcode_image_build.C*** to recreate them in a PHYP specified location in mainstore (each image will probably be placed in mainstore local to its associated processor for performance).
    - OPAL keeps same location, requires that it is at the top of memory
- Parameter: image_bld – which images to update – either PSTATE, STOP, or both
- Fused vs Normal
    - System ATTR defines
- Customize image with data for each core
    - Scan rings – Time, GPTR, Repair
    - Tweak to make runtime acceptable – expect to be only scom registers
- Write image to the appropriate offset based on the output pointer parameter

d Cronus will load the images via putmemproc
This leads to having image building having to deal with Endianess

e proc_stop_gen_cpu_reg(void* v_homer_region, …)

- API that updates a STOP image with various core state registers (MSR, HRMOR, LPCR)
    - The core registers are set to these values on STOP 15 exit
- This will only be called by Hostboot. Cronus will not use it. Hence separate from ***p10_hcode_image_build.C .***

15.2 proc_set_homer_bar :Set HOMER location in OCC and QME

a p10_pm_set_homer_bar.C(uint64_t p_homer_region, …)

- Called for each processor chip.
- Parameter: Physical address within HOMER image where OCC code will be loaded
- Parameters: PBA BAR number, OCC complex HOMER image size(4MB), QME image location (default: mem; others: L3)
- Set PBA BAR address and PBA BAR 0
- Set the BARs and access sizie in all 8 QMEs to point to the CPMR (Core Power Management Region) of HOMER (eg HOMER base + 2MB for 2MB).

15.3 host_establish_ec_chiplet : Update multicast on ECs for runtime state

a *p10_update_ec_state.C ()*

- Multicast groups are not modified
  - Region PSCOM and Multicast enablement is managed by QME Hcode (eg nothing is restored)
- Need to deal with the deconfiguration case
- Loop over all present blueprint cores
  - Clear functional flag
  - Loop over all functional cores
    - If present = functional
      - Set functional flag
      - Set partial good bit in CPLT_CTRL3
      - Set local core buffer bit
  - If not functional flag
    - If powered on and caches clocking
      - Call p10_hcd_l2_purge.C and p10_hcd_l3_purge.C to flush caches to memory
      - Call p10_hcd_core_stopclocks.C and p10_hcd_cache_stopclocks to stop and fence the deconfigured core/cache
      - Call p10_hcd_core_poweroff.C and p10_hcd_cache_ poweroff to power off the deconfigured core/cache
    - If powered on and caches not clocking
      - Call p10_hcd_core_poweroff.C and p10_hcd_cache_ poweroff to power off the deconfigured core/cache
    - Clear partial good bit in CPLT_CTRL3
- Write all Core good mask from local buffer into OCC complex (this was a subset previously)
- This is the “master record“ of the enabled cores/quad in the system for runtime

15.4 host_start_stop_engine : Initialize the QME engine

a *p10_pm_xgpe_init.C *chiptarget, ENUM:PM_START -> FAPI2::ReturnCode*

- Called for each processor chip

- Parameters: PM_START (to perform initialization and engine starting vs PM_HALT that is used during the OCC restart flow)

- Start the Auxiliary GPE (XGPE) engine to allow for 1) QME error log extraction and 2) support PM Suspend functionality for MPIPL
  - Sets OCC flag register bit that initialization is complete for HWP to poll on

b p10_pm_qme_init(chip_target, ENUM:PM_START) à FAPI2::ReturnCode Called for each processor chip

- Called for each processor chip

- Parameters: PM_START (to perform initialization vs PM_HALT that is used during the OCC restart flow)

- Sets up QME block copy engine to bring the Hcode executable and common scan rings from HOMER into QME SRAM

- Starts the QME engine(s)
  - IOTA initialization -> STOP and PMCR functions started
  - Sets flags in QME Flag reg that initialization is complete for HWP to poll on

16 Step 16 Hostboot – Core Activate

16.1 host_activate_boot_core : Activate boot core

- ***p10_gen_fbc_runtime_settings.C***
  - Generates a array of Xscoms for the SBE to execute when system is quiesced (after STOP15) to configure the powerbus protocol for non chip contained execution
    - Desired SL domain setup for runtime, based on attribute value (ATTR_PROC_FBC_SL_DOMAIN_MODE = HEMISPHERE / CHIP)
    - Xscoms must have consistent value on *all* chips in scope of Hostboot image (using XSCOM to reach remote chips)

- Hostboot sends a message to the SBE to enter the deadman loop for exit STOP15 (passes a parameter to indicate the wait time)
  - Hostboot will block and wait for PSU SBE interface return
  - Hostboot command will trigger the SBE to run the following HWP in its Chip OP thread (this will block SP chipOp until it either passes or triggers the checkstop)
  - Hostboot will pass the array of Xscoms to execute once STOP15 is reached. This may be an empty array

b p10_trigger_stop15 – Hostboot path (Hostboot running)

- Hostboot function, not a HW Procedure
- ***p10_block_wakeup_intr.C -set***
  - This will prevent all interrupts/wake up sources to the core, thus allowing the next step (STOP 15) to work
- Hostboot sets up interrupt presenter so OCC ISC port in PSIHB to interrupt master core thread 0

- If we are in fused – there always be even/odd pair – SBE should have chosen the EVEN EC as the master – responsibility for HB to enforce config
- Thus HB will always interrupt the same thread 0 PIR in fused/normal mode

- Hostboot sets up the stop exit LPCR, HRMOR, MSR values in HOMER based on PIR
  - If in fused mode need to set SPR values into 0,2,4,6 if on even EC (or 1,3,5,7 if on odd EC)
- Issue system call to cause all threads to enter STOP 15. Core will then enter STOP 15 state
  - Clear LPCR (cover not entering due to external interrupts)
  - Write PSSCR with Level = 15,
  - Issue ***stop*** instruction

♦ SBE will:

starts SBE Deadman timer upon receiving the ChipOp (SBE FW handling of deadman message)

- SBE starts timer based on ChipOp parameters

- will repeatedly call the following HWP to check for STOP 12 state across all active cores (for fused case)
  - ***p10_sbe_check_master_stop15.C*** (passed in time(from PIBMEM or via Cronus)
  - It can return three different values:
    - STOP 15 reached (success) – FAPI2 SUCCESS
    - STOP 15 not reached, but no error HW state (still in progress) -- STOP15_PENDING
    - STOP 15 not reached, but HW error (failure) – any other FAPI2 RC
  - On STOP15_PENDING, loop with timeout check
  - On any other FAPI2 RC, the RC and FFDC from this HWP needs to be saved by the SBE into async ChipOp FFDC space. SBE will set an "async FFDC" bit in the SBE status register. When the SP recognizes that the master STOP cycle failed, it can then request the "async FFDC"
- upon master STOP 15 reached, performs the "Oop" (see istep 14 for Alley)
  - ***p10_sbe_fbc_apply_rt_settings.C***
    - Up to this point, Hostboot will have been running with 1 SL domain (fabric protocol) per chip
    - In this procedure, master SBE will issue the Xscoms generated by Hostboot
  - ***p10_sbe_powerdown_backing_caches.C***
    - Uses ATTR_BACKING_CACHES (uint32 which is a bit vector written by ***p10_sbe_select_ex***) to determine the powered on caches.
    - Doorbell to the QMEs to have a "virtual" STOP 11 a "virtual" STOP 11 on those core/caches.
    - Poll those cores for STOP 11 state.
  - Send master core an interrupt via using the PSU Interrupt (Separate bit in PSU doorbell). This results in a trigger of STOP 15 exit on thread 0 on the master core
  - call ***p10_block_wakeup_intr.C –clear*** to allow the core to actually receive the interrupt (order between the unblock and interrupt generation doesn't matter)
  - Note that even after triggering Hostboot, SBE must continue deadman timer to check that Hostboot recovers from the master STOP15 cycle. If Hostboot does not stop deadman timer in X seconds (passed in as parameter), SBE must checkstop system. The X seconds is the full time

- On failure the SBE FW will trigger a checkstop (request to add a mode bit in the FW settings to prevent this for lab debug)
- On pending if the timer has expired then trigger a checkstop.
-

c ***Hostboot sends a message to the SBE to exit the deadman loop for exit STOP15***

- Hostboot runs when active, otherwise Cronus will have to execute
- Stops the deadman timer
- Hostboot must issue its own IPIs to threads 1-3 (normal) or 1-7 (fused)

p10_trigger_stop15_exit – Cronus path only (Hostboot not running)

- Since Hostboot is not running (cores are all in STOP 15 by default) Cronus will this procedure
- ***p10_gen_fbc_runtime_settings.C***
- **p10_sbe_fbc_apply_rt_settings.C**
- ***p10_activate_stop15_cores()*** will force all cores to exit STOP 15 using special wakeup.

16.2 host_activate_secondary_cores : Activate secondary cores

- Hostboot active:
  - Setup stack space for all slave core threads –
  - Wake up all threads on all cores via IPI commands
    - Cores are sitting in a STOP15 state (flush that way)
    - Issue IPI to all slave threads/cores to force winkle exit. Will start executing at SRESET vector (0x100). Bring them into Hostboot collective
- FFDCHostboot not running:
  - Cores come alive and into maintenance mode (LPCR not set)
  - ***p10_activate_stop15_cores.C* – Cronus path only (Hostboot not running)**
    - Called on a core target
    - SP/Cronus issue IPIs to all cores/threads in system except for those on master core

16.3 No-op : NO-OP

- This istep is a no-op, content moved to istep 14

16.4 mss_scrub : Start background scrub

a ***exp_scrub.C(mba)***

- Note that this is not executed directly by Hostboot (instead triggered by PRD), Cronus will execute HWP directly
- Start background scrubbing in a continuous 12h scrub cycle
- Currently Hostboot will not wait (block) before flowing out to memory

- The completion of the scrub commands must be handled by Host based PRD
- HostPRD will not be called after this point (not called for this step)

16.5 host_ipl_complete : Notify SP drawer ipl complete

- Stop hostPRD (in anticipation that HBRT will take over PRD responsibilities)

b FSP only actions -- Sends a message to FSP that drawer IPL is complete

- Pushes down all attributes
- Hostboot enters a "quiesced" state
- Setup any data structures/locks for potential drawer merge
- Sends asynchronous trigger message to the FSP indicating that this step is done on this drawer and SP should proceed with the IPL. This message is **not** sent in istep mode
- At this point the SP takes over the IPL

17 Step 17 SP – Init PSI

17.1 collect_drawers : Wait for all drawers to complete

- Wait for all drawers to complete their IPL
  - In normal (non istep) mode this step blocks waiting for all drawers to send the asynchronous hostboot complete trigger
  - In istep mode this step is a noop (the completion of the previous step is the synchronization point)
- FSP will apply fabric topology configuration based on Abus and processor chip deconfiguration
  - Generic rule is that any node with a deconfigured A bus or processor chip is removed from the system config. Once down to a single node the fabric topology is governed by the config in hostboot during step 8.

17.2 proc_psiinit : Enable PSI link

a p10_psi_init.C (on 1 functional link)

- Set FSP PSI Frequency and other mode bits
- Call adal_psi_init on FSP
- Raise IOVALID on both sides of the bus
- Enable the link on p10
- Call adal_psi_link_enable on FSP
- Verify link is active

17.3 psi_diag : Test the PSI Links

- Only run when new HW is detected or in MFG mode
- Always skipped on R/R
- Disable active link
- For every possible link

b Run p9_psi_init.C

- Run new PSI PHY pattern test
- Perform DMAs across active link
- Disable active link
- Leave 1 link active

18 Step 18 Establish System SMP & TOD

Note steps 18.1 - 18.10 are only performed by the FSP on Denali systems

18.1 sys_proc_eff_config_links : Powerbus link config

a p10_fbc_eff_config_links.C (None)

- Processes link related attributes for current HB powerbus scope (X & A busses)
- Does not access HW

18.2 sys_proc_chiplet_fabric_scominit : Scom inits to fabric chiplets

a p10_chiplet_fabric_scominit.C

- Initfiles in procedure defined on VBU ENGD wiki
- Apply scom overrides to all chiplets necessary to init the powerbus. Note that this does not cover any PHY inits, just the DL/TL layers
  - p9.fbc.no_hp.scom.initfile
  - p9.fbc.ioe_dl.scom.initfile
  - p9.fbc.ioe_tl.scom.initfile
  - p9.fbc.ioo_dl.scom.initfile
  - p9.fbc.ioo_tl.scom.initfile

18.3 sys_fabric_dl_pre_trainadv : Advanced pre training

a p10_fabric_dl_pre_trainadv.C (called on each iohs target pair))

- Debug routine for IO Characterization
- Nothing in it

18.4 sys_fabric_dl_setup_training : Setup training on internal node

buses

a *p10_fabric_dl_setup_linktrain.C (called on each iohs target pair)*

- Hostboot will setup training on all intra node buses. The iohs target is passed in and the PDL region is used to perform the training.
- Is there swizzling of lanes to PDL? Handled via ATTR
- Repairable fails are left for PRD to analyze and perform appropriate error logging
- Fatal bus training errors are handled by procedure, must return error and FFDC

18.5 sys_proc_fabric_link_layer : Start SMP link layer

a *p10_fabric_link_layer.C (called on processor chip)*

- Reads logical iohs link configuration attributes, trains the DL/TL layers of selected links
- Set scom on both sides of the bus to trigger Data link layer training
- DLL sends training packets, sets link up FIR bit when done
- FIR done bit launches the Transaction Layer (TL)
- FIR bit in nest domain to indicate training done
- After this point the mailbox register are available to communicate
  - Xstop would prevent mailbox communication
- Bus is NOT part of the SMP coherency
- Only performed on setup, valid buses
- The link between the iohs and IOPPE will be used for FFDC, but most interactions are handled by HW

18.6 sys_fabric_dl_post_trainadv : Advanced post training

a *p10_fabric_dl_post_trainadv.C (called on each iohs target pair)*

- Debug routine for IO Characterization
- Nothing in it

18.7 sys_proc_fabric_iovalid : Lower functional fences on local SMP

- Hostboot must know the valid Abuses based on number of nodes passed up by FSP
- FSP will raise IO Valids on the bus (eventually should be done by Host, however since we are not protecting against physical attacks this was a compromise to avoid complicated communication between HB/FSP)

a *p10_fabric_iovalid.C (chip target)*

- Reads logical fabric link config, sets iovalid for selected links
- Only performed on trained, valid buses

- After this point a checkstop on a slave will checkstop master
- Reads the fabric link delays for later HWP to pick best link for coherent traffic

18.8 sys_proc_fbc_eff_config_aggregate : Pick link(s) for coherency

a p10_fbc_eff_config_aggregate.C (chip target)

- Reads attributes from previous HWP and determines per-link address/data capabilities
- Sets up attributes for build SMP

- This istep only applies when there are multiple Hostboot instances in the system
  - On all boots except MPIPL boots (which already have the fabric alive)send mailbox message (not a sync point)to each hostboot image (physical drawer) to allow SMP integration:
    - Indicating existing nodes and proposed master
      - Master is lowest functional node for IPL
    - Master hostboot entity will build a consensus on present nodes and TPM content
      - Communication is done via 4 64bit Xscom Abus “mailbox” address, readable/writeable prior to IO valid on the Abus
        - Each bus has two half links. Each half link has two scom registers
          - So Abus 0 has link 0, 1, Abus 1 has links 2,3, etc. Each half link has two scom registers
        - Local data registers (2 per half link) can be read locally via Xscom
        - Local data registers can be written by local Xscom
        - The other side of the link can also be written and read via mailbox control procedure
      - When the AMP (abus mailbox protection security bit) is on then the local half link 0, reg 0 cannot be read via PIB SCOM (Xscom or FSI)
        - So when HB wants to transmit the nonce it must only ever use the half link 0, reg 0 on the remote side (ie use the mailbox control hw to write to far side of the link). The slave HB instances must always read the nonce via Xscom to local half link 0, reg 0
          - It is possible that this can trigger SP ATTN to host
        - It must not be possible for Node to get to half link 0, reg 0 after AMP is set if IO valid is not on
          - Must not be possible for a node to read the other nodes mbox ctl regs
          - Most not be possible for a node to have FSI connections to other node and get access (ie Abus mbox regs must be on SBE blacklist for read and write)

- Communication is from the Master to the slaves, slaves only ever respond to the master (no slave to slave communication)
- HB exchanges TPM information. At this point, slave TPMs need to be marked in HDAT and poison the slave TPM PCRs
- Lock down Abus mbox
- ***p10_update_security_ctrl.C (lock_down Abus)***
  - Lock down Abus mbox so it is write only

- HB instances will respond to original mailbox request
- Must make sure that this and all traces are flushed to the FSP prior to HB continuing
- All hostboot instances will now STOP 12 ALL cores
  - This is to allow the CCM operation to complete in the next step by the FSP
  - **p10_block_wakeup_intr.C -set**
    - This will prevent all interrupts/wake up sources to the core, thus allowing the next step (winkle) to work
  - ***Hostboot is now quiesced…. Will not do anything until interrupt from the FSP***

- FSP takes over the flow here
  - **p10_check_master_stop_15.C**
    - Procedure to query that cores are indeed STOP across the system before proceeding
  - **p10_build_smp.C**
    - CCM all p9 HB instances SMPs into a system wide SMP
    - Quiesce fabric on all slave drawers
    - write AB CURR/NEXT state on 'slave' drawers in preparation for integration (i.e. to reflect full multi-drawer configuration)
    - write AB NEXT state on 'master' drawer in preparation for integration
    - issue sequence of hotplug integration commands from ADU on 'master' drawer
      - PB quiesce op - this would only be broadcast to 'master' drawer
      - switch AB op - this would also only be broadcast to 'master' drawer, and would cause its chips to switch to the multi-drawer configuration
      - PB init op - this would reach all drawers and establish the full multi-drawer fabric
  - **p9_block_wakeup_intr.C -reset**
    - FSP then removes the block wakeup sources (master processor last per drawer)

        - This removes the block wakeup sources and allows hostboot to receive (and exit winkle) the pending mailbox message
    - FSP sends each Hostboot image a synchronous mailbox “coalesce_host” message
        - This places a pending interrupt to the master core/thread
    - HB instance will wake up (un STOP) its master core, thread 0
        - Hostboot will then issue doorbell commands to all threads in its instance
        - Hostboot will establish IPC between each of the HB instances
            - On failure system will xstop or TI
        - HB instance responds to synchronous message
        - After this point FSP only communicates to the master Hostboot Instance

Note the following steps are performed on ALL systems

18.9 proc_tod_setup

- On FSP Based systems this is run on the FSP, BMC based systems this is run in Hostboot
- FW owns algorithm of TOD topology, HWP pushes values into HW

b p10_tod_setup.C

- FW passes in a topology tree, which TOD oscillator to use, and primary/secondary topology
- HWP determines delay values from attributes (MRW)
- HWP programs HW
- HWP outputs register values needed for PHYP and PRD analysis

18.10 proc_tod_init

- On FSP Based systems this is run on the FSP, BMC based systems this is run in Hostboot
- Performed to init the TOD network. Done during the FW IPL due to AVPs, note that it will be done again by PHYP when they start

b p10_tod_init.C

- Setup EX chiplet TOD

c This is the last istep that FSP/HWSV will perform a HW reconfig loop

18.11 cec_ipl_complete : IPL is complete, any needed FW cleanup

- Used to trigger bulk state sync between target model and system model. After this point every update is synchronized immediately.
- No longer check for reconfig loops

18.12 startprd_system : Start PRD for the system

- Build PRD model for entire system
- At this point FSP is responsible for checkstop attentions and core special attentions in the system

18.13 attn_listenall : Enable runtime Attention Handling

- Start listening for attentions for all chips from FSP
  - All true/complement masks are rewritten in scom/fsi2pib engines for all chips

19 Step 19 SP – Prepare for Host

19.1 prep_host : Prepare for Host

a Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/PrepHost.tf

- Enable Platform Dumps – dumpsystem init
- Sync all data to backup -- rmgrcmd –sync-with-ipl-signal
  - This triggers Backup FSP IPL (see Section 9)
- Activate panel function 42
- Wait for completion of backup IPL -- rmgrcmd –wait-for=backiplcomp
- Starts the PSI mailbox process

20 Step 20 Hostboot – Load Payload

20.1 host_load_payload : Load payload

a Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/HBLoadHostOS.tf

- build_host_data : Build the host data areas
  - This step builds the HDAT data areas from attributes, VPD, etc
- Load payload. This can either be directly from PNOR (controlled by attribute) or via the SP
  - PNOR path – just loads what is in payload section on flash
  - SP path
    - When the Host sent the complete IPL message for host_ipl_complete part of the payload is the address to load PHYP at (along with a size)
    - For initial BU (non secure mode) PHYP will be loaded via raw DMAs
    - For secureboot PHYP must be loaded via TCEs
    - Payload will be placed in memory based on Hostboot attributes
      - Base address is defined by ATTR_PAYLOAD_BASE When Payload is started this is the HRMOR
      - Starting address is defined by ATTR_PAYLOAD_ENTRY
      - HDAT is placed at well known address off of the image start address
      - **All addresses must be security checked by Hostboot before starting payload**

- Hostboot then performs verification on the payload

20.2 host_load_complete : Payload is complete

- Switch FSP<->Hostboot communication to use the PSI bus instead of the PBA mechanism to avoid conflicts with FSP<->OCC communication on the PBA bus
- Mechanism for FSP to indicate to hostboot that the payload is complete and can be run. Used in normal boot mode as an asynchronous trigger to hostboot to continue the IPL and run the next step
  - A HB no-op in istep mode
  - This is only issued to the master HB instance

21 Step 21 Hostboot – Start Payload

21.1 host_micro_update : Update System micro code

- This includes PCIe Micro, UCID and NVDIMM FW loads
- Update micro code if it is out of date
- This step is a catchall for any microcode updates

b p10_perst_phb.C (phb target, bool i_assert_deassert) – This is called for PCIe switch updates

21.2 host_runtime_setup

- Note that this step is only issued to master HB instance
- Take down any/all TCE setup
- Loop through attributes and write them to predefined memory area inside of the HDAT structures
  - Note: HB master issues IPC to HB slaves for them to update their sections
- Append the TPM log to HDAT structures
  - Note: HB master issues IPC to HB slaves for them to update their sections
- In OPAL mode Hostboot will load the OCC and start it here. If the load/start fails then HB will send a errorlog to the SP and the SP will terminate the IPL
  - Must check Fmax frequency across the entire system
    - Mix of HWP and HB code
    - HWP does a fmax calc on “HB instance group” and sets system ATTR, then HB does the “max” function across all HB instances, se
  - Will call the high level PM Reset and load
  - OCC must monitor for the broadcast scom read (OR) of EX scratch register 7 for the removal of the payload started signature before using the FSI2Host mailbox for ATTN traffic. Note that OCCs on non master chips will never have to wait (as Hostboot only uses the FSI2Host mailbox on the master chip)

21.3 host_verify_hdat

- Only issued to master HB instance
  - If needed IPC to slaves to perform their tasks
- Secureboot verification of PHYP/AVP image load

21.4 host_start_payload

- Prior to starting shutdown sequence Hostboot must write hostboot (ASCII) to scratch register 7 on the master core. All other cores on the master chip must be written to same value or 0s. This value will be polled by the SP in the next step to ensure that hostboot has truly quiesced
- Hostboot enters shutdown sequence
  - Quiesce mailbox and all DMAs
  - Flush data to PNOR
  - Disable interrupts
  - Send sync message to SP (or respond to istep)
  - Enter Kernel
  - Prepare to jump to payload – at this point hostboot must not TI
  - Clear scratch register 7 on master core
- Payload is started by
  - switching HRMOR to desired address and jumping to entry point
  - Note that master thread must be the last one to jump
  - payload cannot start until all threads have been transitionedFor multi-node systems the HB master does the following:
    - Issue slave node shutdown request via IPC
    - HB master polls the “Hostboot done scratch reg” for all slave nodes to enter payload
    - HB Master issues own shutdown
- No Hostboot code is reused, only mechanism is data passed in HDAT areas. Hostboot runtime is a separate binary image

21.5 host_post_start_payload : Post payload start

a Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/HBPostStartHostOS.tf

- istep hostboot_done – wait for hostboot handoff to payload
  - Wait to receive the Hostboot payload started sync point
  - Issue broadcast scom read (OR) of EX scratch register 7 looking for it to be non 0xBAD60BAD BAD60BAD. Poll 10 seconds waiting for signature to change, if it doesn't terminate IPL and grab hostboot dump.
  - At this point we know that hostboot is no longer in control of host
    - Quiesce all hostboot services (ie continuous trace, error log, etc)
    - Transition state flag to indicate payload is executing
    - enable FSI2Host mailbox for OCC communication
    - FSP takes back ownership of LPC2SPI (PNOR control)
    - If in AVP mode, HWSV sends message to TMGT to notify OCC load is complete

- Enable power management FW (tmgtclient –tmgt_xfile_on_enable_tpmf)
- Time PHYP start (initial mailbox message down from PHYP)

```
21.6 switchbcu        : Switch BCU - send continue to phyp   (G)
```

a Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/SwitchBCU.tf

- Mailbox handshake with PHYP
  - wait for OLC
  - switch BCU and "continue" OLC (Operation Load Complete)

```
21.7 completeipl            : notify HMC  (G)
```

a Execute /etc/fspinit/tf/eclipz/IplSteps/Normal/CompleteIPL.tf

- Wait for continue ack
- Dumpsystem signalhmc
  - Signals to HMC if a dump is available and ready to collects

4 Host Services

The following are not IPL time procedures, but functions called by PHYP/OPAL on Hostboot to perform various tasks. The numbering has been kept common (for convention), but they are not guaranteed to run in this order

State at this point

- PHYP/OPAL running
- Memory is initialized
- SMP alive
- All cores have gone through winkle and are running

22 Enable STOP15

This step is controlled and issued by PHYP when they are ready to build the STOP image. It is required that the STOP image be present in memory prior to loading and starting the OCC.

```
22.1 host_build_winkle          : Build runtime winkle images
```

- Pull Reference Image from SP or PNOR
  - Run through secure boot algorithm

b p10_hcd_image_build.C

- Called for each processor chip.
- Parameter: Pointer to Reference image.
- Parameters: Work buffers needed in the creation of the output image
  - Input buffer 2 will contain the contents of the unsecure memory contents to be copied to ATTR_UNSECURE_HOMER_ADDR for the sized definded by ATTR_UNSECURE_HOMER_SIZE in SMF enabled systems

- Parameter: Pointer to Output HOMER location. The procedure places the respective images (eg PGPE, QME and XGPE) into HOMER at the appropriate offsets.
  - This is any Hostboot specified mainstore location (does not have to be attached to the processor being STOPed
- When PHYP is loaded, the HOMER will be trampled, PHYP will call ***p10_hcd_image_build*** to recreate them in a PHYP specified location in mainstore (each image will probably be placed in mainstore local to its associated processor for performance).
- Customize image with data for each core
  - Scan rings – Time, GPTR, Repair
  - Tweak to make runtime acceptable – expect to be only scom registers
- Write image to output pointer parameter

22.2 proc_set_homer_bar : Tell OCC complex HOMER loc

a p10_set_homer_bar.C

- Called for each processor chip.
  - Parameter: Physical address of HOMER region where OCC code will be loaded. QME image is this value + 2MB (not a pointer address, it cannot be dereferenced)
  - Parameters: PBA BAR number, OCC Complex HOMER image size, QME image location (default: mem; others: L3, SRAM)**This need to update the BAR registers in each QME for P10.**

b p9_stop_gpe_init -init : Initialize the STOPGPE p10_pm_qme_init.C chip_target, ENUM:PM_ START) -> FAPI2::ReturnCode

- Called for each processor chip
- Parameters: PM_START (to perform initialization and engine starting vs PM_HALT that is used during the OCC restart flow)
- Loop over all EQ chiplets
  - ***p10_pm_pfet_init.C (cache target, PM_START)*** (called as a subroutine)
    - Initialize PFET controller parameters (delays
  - Initialize QME Flag and Scratch registers per attributes
    - STOP mappings
    - MMA enablement and timings
  - Clear other QME Flags and Scratch register fields
    - Error injection bits **(looking to deprecated)** --- this allow the machine to boot
    - Ready status bits (STOP, PMCR, Heartbeat Loss)
    - Ignore STOP Entries and Exits (control)
    - Core Block STOP Exit and Entry Enabled (status --- will be set by QME Hcode)
  - Retain (don't touch) the following QME Flags and Scratch register fields
    - L3 contained and Chip contained modes
    - Special Wakeup Check Enable
    - QME Debug Trap Enable, QME Debug Halt Enable

- STOP Entry First
- Resonant Clocks Operable, DDSs Operable (status) – owned by PGPE
- Start the QME engine

♦ Bootloader runs from HOMER OCC offset + 1MB (2MB from HOMER base)
- Copies STOP image from HOMER to QME SRAM
- Restarts from QME SRAM
  - IOTA initialization -> STOP functions started

♦ Sets flags in QME Flag reg that initialization is complete for HWP to poll on

♦ ***proc_stop_gen_cpu_reg()*** will be called by Hostboot, PHYP, or OPAL prior to ***stop***ping any core
- API that updates a STOP image with various chip state registers (MSR, HRMOR, LPCR, etc)
- The chip registers are set to these values on STOP exit
- **This will only be called directly by PHYP at their discretion. Hence separate from *p10_hcode_image_build*.**

23 Reset and Initialize OCC

This step will run each of the substeps to each chip within a physical node (an OCC boundary) before proceeding to the next step. This is done as a regular process in looking to the "start_occ" step whereby the OCCs will start in reasonable time proximity (one followed by the next via singular XSCOM to each chip) to minimize OCC startup timeouts.

23.1 Setup OCC bars : Establish legal addressing

a p9_pm_pba_bar_config.C chiptarget, address

♦ Address dictated by PHYP

♦ Called once for each of 4 BARs

♦ Place image in EM Nodal Region at offset 0

♦ Need to update QME BARs

23.2 power_management_reset : Reset Power Management

(includes clearing any latent errors that may be pending; done for the case of OCC reset)

a p10_pm_halt.C – chiptarget
- ***p9_pm_firinit & i_chip_target, ENUM:PM_HALT*** : Save the current FIR mask setting for later restoration and then set all masks to keep errors from occurring during the reset and initialization
  - ***p10_pm_qme_firinit.C &i_chip_target, ENUM: HALT***
    - For all EQ chiplets, save and set all QME FIR Masks to specific backing attribute
  - ***P10_pm_occ_firinit.C &i_chip_target, ENUM: HALT)***
    - save and set all FIR Masks to specific backing attribute
  - ***p9_pm_pba_firinit.C &i_chip_target ENUM: RESET (candidate for reuse, may not be worth it)***

        - save and set all FIR Masks to specific backing attribute
    - ***p9_pm_occ_control.C chiptarget, ENUM:OCC_HALT*** (candidate for reuse)
        - OCC PPC405 is halted to allow for a clean stop
        - Will cause HW heartbeats to cease and HW (if PGPE is active) will enter safe mode – expect to take less than 10 ms
    - ***p9_pm_occ_control.C *chiptarget, ENUM:OCC_STOP*** (candidate for reuse)
        - OCC PPC405 put into reset
    - For all configured cores, ***p10_cpu_special_wakeup.C *ectarget, ENUM:ENABLE –entity ENUM:OCC***
        - Not used by PHYP – custom procedure used
    - Takes the QME out of the equation
        - Uses the OCC special wake-up
        - Poll for completion.
        - If timeout, indicates that restart of OCC is to not occur via fapi::ReturnCode
            - RC_PROCPM_SPC_WAKEUP_TIMEOUT
            - PRD effect: Mark chiplet for garding
        - Note: QME detected errors will produce malfunctions alerts to PHYP whereby the set of events defined in ***p10_stop_recovery.C*** occur to deal with getting the idle handling complex recovered for use. This is done by XGPE monitoring QME health and asserting the malfunction alert.

♦ ***p9_pm_occ_gpe_init.C *chiptarget, ENUM:PM_RESET (candidate for reuse)***

- Halt both OCC GPE engines
    - Set OCC Flags to request a graceful halt, after timeout will force
    - If forced off, then need to relinquish I2C engines (if owned by OCC) and send interrupt via OCC_MISC
        - With PPC405 stopped and safe mode in place, this engine is not being used.
        - Halt the engine via XCR[CMD] = “halt”
        - Note that when the OCC GPEs stop, the Explorer (OCMB) memory will throttle into safe mode due to lack of polling

♦ ***p10_pm_xgpe_init.C *chiptarget, ENUM:PM_HALT***

- Halt Auxiliary GPE (XGPE) engine
    - Halt the engine via XCR[CMD] = “halt”
    - ***p10_pm_pgpe_init.C *chiptarget, ENUM:PM_SAFE_MODE***
        - Command the Pstate GPE engine to put the chip into Safe mode
            - If PGPE is operational,
                - Clears “SAFE_MODE_COMPLTE” and sets “SAFE_MODE_IN_PROGRESS” in OCC Flag Register (this gives this procedure positive feedback that PGPE is acting on this request)
                - Use existing Pstate protocols to move to the safe frequency, voltage (PGPE has the VPD points to it know where that is) and throttle settings.

- PGPE
- Else if PGPE is not operational (SAFE_MODE_IN_PROGRESS not set in 10ms timeout or SAFE_MODE is not set in 500ms)
  - Read present external voltage using O2S Bridge B
  - If voltage is above the safe voltage (eg the voltage needed for the safe frequency), read all EQ FREQ_CNTL_REG to determine the present DPLL frequencies.
    - If all are at or below the SAFE frequency, leave;

- ***p10_pm_pgpe_init.C *chiptarget, ENUM:PM_HALT***
  - For **all** EQ chiplets
    - Force OCC SPR Mode in each core to remove OPAL communication path
    - Adjust clock grid for the safe frequency to allow for HOMER updates of clock grid parameters
    - OCC Heartbeat disable
      - Will be enabled by PGPE Hcode (not FAPI)
    - Halt Pstate GPE engine
      - With PPC405 stopped and safe mode in place, this engine is not being used.
      - Halt the engine via XCR[CMD] = "halt"
      - Note: this will engage the PGPE "OCC Heartbeat" in the QME which will cause the QMEs to move to the safe throttle values.

♦ ***p10_pm_qme_init.C chiptarget, ENUM:PM_HALT***

- For **all** EQ chiplets
  - Halt QME engine
    - Halt the engine via XCR[CMD] = "halt"
  - ***p9_pm_pba_init.C *chiptarget, ENUM:PM_RESET (candidate for reuse)***
    - Issue resets to all 4 PBA Slaves; poll for completion
    - This does not touch the PBA BARs
  - ***p9_pm_occ_sram_init.C *chiptarget, ENUM:PM_RESET (candidate for reuse)***
    - Placeholder
  - ***p10_pm_ocb_init .C *chiptarget, ENUM:PM_HALT***
    - Disable all OCB indirect channels and return them to their power-on state
      - Note, may need to leave one of the channels enabled for SBE<->Host comm
      - The HALT phase could be reused. However, the START phase cannot as there are now 3 AVSBuses.

- ***p10_pm_pss_init .C *chiptarget ENUM:PM_HALT***
- See that any outstanding operations have finished in ADC engine
  - See that any outstanding operations have finished in P2S engine

24 Load OCC

24.1 load_occ : Place OCC image into memory

- For each chip in a physical node
- There are two divergent paths to load the OCC code image. The first is lab/Cronus only without FW. In this case the HWP is run. In the second case FW controls building up the image at the direction of PHYP

b p9_occ_load.C CRONUS ONLY, mimics what FW does

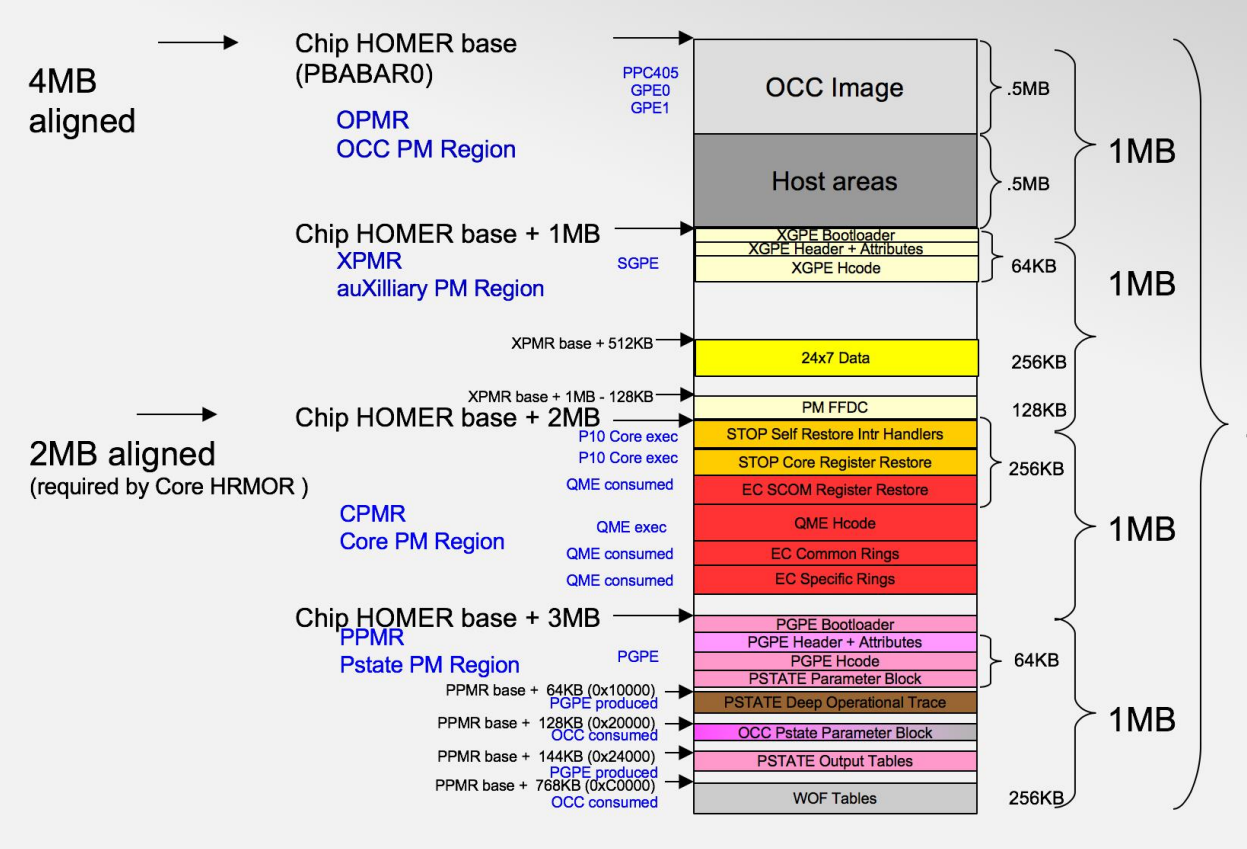
- Load image in memory from PNOR at an address that is passed to this procedure

c occ_load: FW

- There are four different scenarios where this will get run:
  - PHYP: calls HBRT Adjunct
  - OPAL with FSP: HBRT directly within OPAL
  - OPAL openPOWER: Hostboot calls this prior to starting OPAL
  - AVP mode: Hostboot call this prior to loading AVP
- HBRT called with memory region to place the HOMER image
  - HBRT obtains OCC, reference image
    - FSP based systems via lidmgr
    - OpenPOWER systems via PNOR
  - Entity that loads the image verify signature through secure algorithm
    - Lidmanager PHYP
    - PNOR HBRT
- HBRT will create the STOP image from the reference image (see step 15 of IPL)
  - HBRT will recreate the whole image each time (both OCC/PState,
  - ***p10_hcode_image_build.C*** (void* reference_image, void* v_homer_region, ALL)
    - This includes the PGPE, XGPE and QME.
    - Step 15 built the QME components (STOP function)
    - The PGPE and XGPE is tied to the OCC function
      - Manufacturing request to allow biasing
      - Build Pstate Parameter Block (PPB)
      - Good cores come via the OCC CCSR ~~deconfig~~ register
- HBRT will place OCC initial startup information into HOMER image
  - Nest Frequency

- Interrupt type – FSI2Host mailbox(TMGT) or via PSIHB(HTMGT)
- FIR Master
- FIR Capture Data (generated by HBRT) – non FSP based systems
- Processor map, and FIR register to read

♦ HBRT places STOP and OCC images as directed by caller. The following is an overview of a completed HOMER layout>



25 Start OCC

25.1 start_occ : Start OCC

a ***p10_pm_start.C *chiptarget, ENUM:PM_START***

b ***p10_pm_***qme***_init *chiptarget, ENUM:***START

♦ Sets the IAR to the QME bootloader in HOMER.

- HOMER base (QME PPEBAR + 2MB + 256KB) + 16B

♦ Starts the QME and polls QME Flag bit for HCode init completion

- Polling interval: 100us; Timeout: 500ms

♦ ***p10_pm_ocb_init.C *chiptarget, ENUM:PM_START***

- Set registers to initial settings (3 AVSBuses, Flag and Scratch regs, Channel regs)

♦ ***p10_pm_pss_init.C *chiptarget, ENUM:PM_START***

- Setup PSS Configuration (PSS Frequency (attribute) to PSS macro settings)

♦ ***p9_pm_pba_init.C *chiptarget, ENUM:PM_INIT (reuse candidate)***

- "PowerBus Slave" buffer set configuration. Assigns slaves to OCI masters for runtime (vs IPL time for HBI loading)
- PBA Configuration
  - Hang pulse dividers

♦ ***p10_pm_firinit.C :*** Set the FIR masks and action bits per RAS FIR spreadsheet; done as FAPIs vs scom.initfiles to be supportable under PHYP

- ***p10_pm_qme_firinit.C *chiptarget, ENUM:PM_START***
  - For all configured EQ chiplets, sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_HALT*** into an attribute)
- ***p10_pm_occ_firinit.C *chiptarget, ENUM:PM_START***
  - sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_HALT*** into an attribute
- ***p9_pm_pba_firinit.C *chiptarget, ENUM:PM_INIT (reuse candidate)***
  - sets FIR Masks and actions registers (first time takes on initial mask value; subsequent calls restores the value saved during ***ENUM:PM_RESET*** into an attribute

c p***10_pm_***pgpe***_init *chiptarget, ENUM:***START

♦ Sets the IAR to the PGPE bootloader in HOMER.

- HOMER base (PBABAR0 + 3MB) + 16B

♦ Starts the PGPE and polls OCC Flag bit for HCode init completion

- Will NOT start Pstate Protocol until commanded by OCC FW

d ***p9_pm_occ_control.C *chiptarget, ENUM:OCC_START*** (reuse candidate)

- Starts OCC load by releasing the reset to the PPC405
- OCC code boot loads itself from Memory into SRAM tank

26 Config OCC

`26.1 config_occ : Load OCC config`

(H)TMGT now builds the OCC config data and uses its communication path to OCC to give pass config information

a ***OCC FW sends OCC IPI to PGPE to start Pstate Protocol***

- PGPE reads Pstate Parameter Block (PBB) from HOMER, installs in OCC SRAM, and starts the Pstate Protocol with the CMEs.