HostBoot IPL Flow for OpenPower

v1.46 (07/02/14)

1 Introduction

1.1 Description

This document will describe the high-level IPL flow for the IBM POWER servers based on P8 (i.e. PgP chips). It is not intended to contain all low-level details, but instead is designed to illustrate the relationships between various low-level procedures. The source code and specifically the hardware procedures will always be the final documentation.

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.

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.

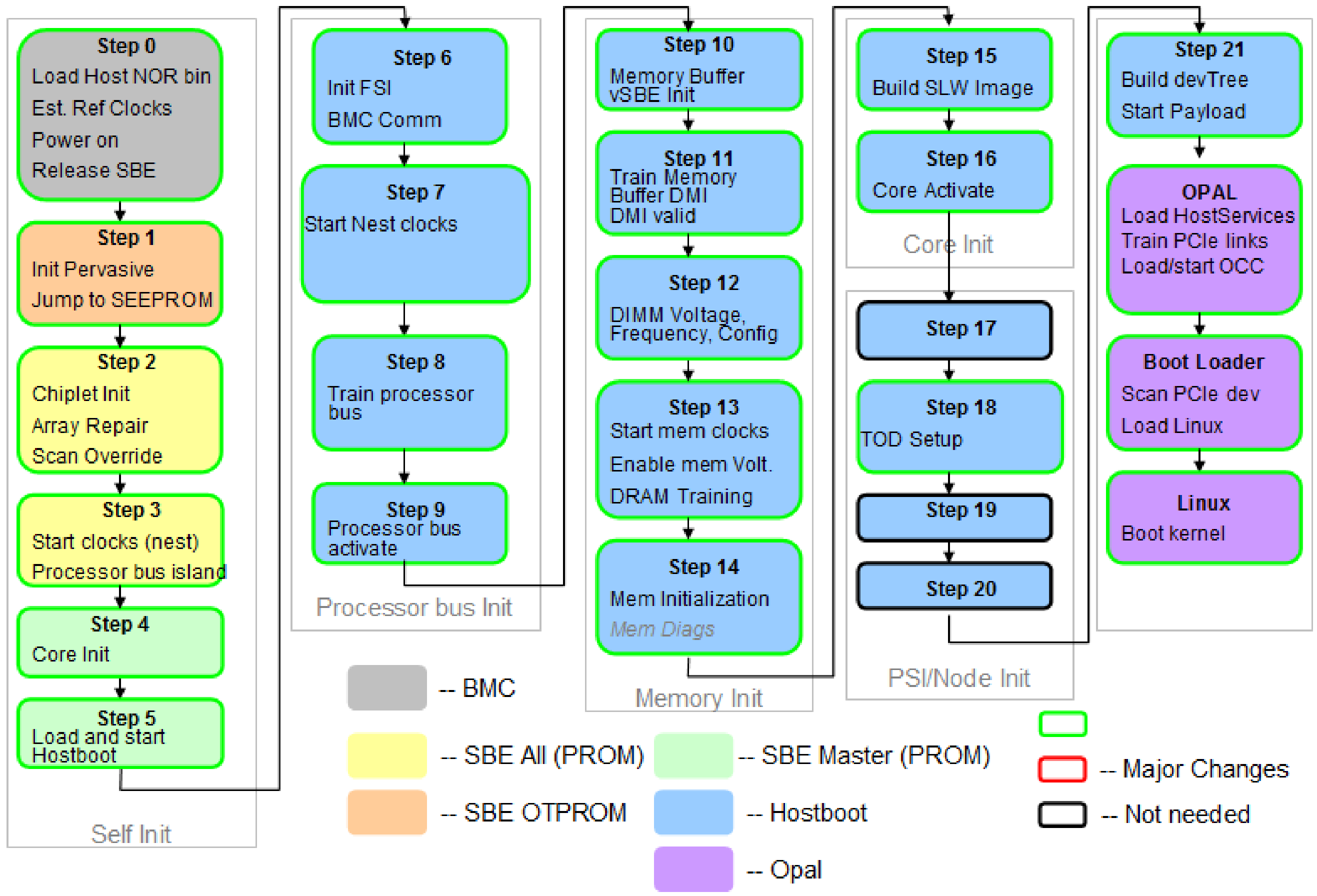
This version of the document will cover all POWER8 systems. Please note that this document has a lot of low level details on the initialization of the POWER processor and it's memory subsystem. There are a lot of terms and details in here that are very IBM and POWER centric. We attempted to put as much 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.

There are also quite a few references to “Secureboot” throughout this document. Secureboot is a function built into the POWER8 chips to ensure the appropriate firmware is running. Certain aspects of this have been implemented but other portions of it have been saved for another day.

Reading over the Hostboot Programmers guide (same document repo) is reccomended prior to reading this document.

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

- **Centaur :** Memory buffer chip which optimizes memory bandwidth and usage (processor memory controller talks to centaur which talks to logical dimms)
- **Cronus :** Lab debug tool used to control and debug the IPL (mostly by the hw engineers)
- **EX :** Contains processor core, L2 and L3 logic.
- **Hardware Reconfig Loop** : When a piece of hardware is deconfigured during the IPL, the firmware can in certain instances go back to a previously run istep and still boot the system with the hardware now deconfigured.
- **Hostboot/HB**: Firmware that initializes the memory and powerbus.
- **HB Runtime Services**: Portion of Hostboot that remains resident during hypervisor execution and provides PRD functionality for Ces
- **IPL** : Inital Program Load = Boot process. Covers time between power on and running they hypervisor
- **Maintenance-mode** : IPL that includes extensive diagnostics to test the hardware
- **MPIPL :** Memory Preserving IPL – This is an IPL type used to grab hypervisor memory in certain error scenarios (for debug). This is currently not supported on OpenPower systems.
- **Normal-mode** : IPL that includes minimal diagnostics, focused on functional requirements only
- **PCB** : Pervasive Control Bus – Internal processor bus that provides a communication layer between the internal logic within the chip.
- **Pegasus** : Family of server products based on around P8 or P8+ processor chips
- **PgP :** P8 Processor Chip
- **PNOR :** Processor NOR. NOR memory device where all firmware, including the hostboot firmware, is stored and from which it is loaded. It is attached to the master processor through an SPI bus.

- **PORE :** Power On Reset Engine – Mini-execution engine within each P8 chip used to execute firmware which assists with some of the IPL steps.
- **SBE:** Self Boot Engine – A version of a PORE within each P8 chip which is used to do some basic initialization to each chip and to load and start the Hostboot firmware.

1.2.1 IPL Types

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

1.2.2 Nomenclature/Conventions

- Items in bold-italics represent deliveries from the hardware team, either procedures or engineering data files. Ex: ***p8_cfaminit.C***
- Initfile processing is shown with the "WHEN=" value in parentheses, e.g. galaxy2.initfile(L) means to process the initfile with "WHEN=L".
- Steps in italics are software only and do not interact with the hardware at all.
- 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

Master Processor Chiplet State (after istep0-5)						
Chiplet	**Clock source**	**Scan0, Array Init**	**Repair Data?**	**Override Scans?**	**Clocks On?**	**Comments**
TP	Nest PLL	X	X	X	X	Fully functional
TP – OCC	Nest PLL	X	X	X		
Pbus	Nest PLL	X	X	X	X	Powerbus in island mode
Pbus – MCS	Nest PLL	X	X	X	X	
Xbus	Bypass pll	X	X	X		
Abus	Bypass pll	X	X	X		
PCIe	Bypass pll	X	X	X		
MBS (Centaur)						
MBA (Centaur)						

EX – Boot	DPLL	X	X	X	X	Executing Hostboot
EX – Other	Bypass pll	X	X			

4. Step 0-4 Power On and SBE Start
5. Step 5 Hostboot Start

   isteps 0-5 involve the BMC initiating a power on to the CEC and starting the SBE which then does the basic initialization required to load hostboot into the L3 cache of the first funcional core and starting instruction on that core so hostboot starts running.

6. Step 6 **Hostboot – Slave SBE**

6.1. host_setup (non executable istep): Setup host environment

a) If in secure boot the ROM as already validated image

b) Select primary thread (only thread running)

c) Initial setup

- stacks
- MSR
- execution environment
- Thread control structures
- Memory Management setup

d) Ready for execution

- Start and release all other threads on core (1-7)
- Tracing
- Device Drivers
  - Xscom
  - Mailbox

e) HB mechanism to read/write to PNOR

- All Xscom based
- Host writes to LPC ↔ SPI NOR controller to read/write
- SBE uses NOR at lowest frequency, Hostboot will use flash config info to speedup to full frequency

6.2. host_istep_enable (non executable istep): Hostboot istep ready

a) Hostboot checks PNOR for istep attribute, if set Hostboot “halts” and waits for commands from BMC

b) Only isteps after this point can be issued to Hostboot

c) At this point communication can be performed with the SP

6.3. host_init_fsi : Setup the FSI links to slave chips

a) It is expected that the following steps have already been done by SP – Hostboot will just use FSI bus (except on OpenPower systems where Hostboot is responsible for this)

- Configure FSI master (HUB and Cascade)
- Send break commands to FSI slaves
- Configure the slaves
- Force lbus

b) Setup Scom device drivers

- Read ID/EC levels

6.4. host_set_ipl_parms : Build ipl parameters

a) Sets the IPL parameters for this boot

6.5. host_discover_targets : Builds targeting

a) Determines what targets are present and functional

b) This is the step where the host "configures" itself and builds its present/functional map of the targets

- Uses FSI presence to detect processors and memory buffers
- Reads dimm VPD from PNOR to determine what dimms are present

6.6. host_gard : Do Gard

a) Apply repeat-gard records and deconfigure hardware

6.7. host_cancontinue_clear : Clear deconfigured states

a) Clears Cancontinue target deconfigured state (used to know when to trigger a CanContinue IPL)

6.8. proc_revert_sbe_mcs_setup : Clean up MCS extent regs

a) ***proc_revert_sbe_mcs_setup.C***

- Clean up the MCS BARs that were used by SBE and Hostboot to cleanly load/purge the L3 cache
- Re-enable speculative reads

6.9. proc_cen_ref_clk_enable : Setup centaur ref clocks

a) ***proc_cen_ref_clk_enable.C***

- Enable the ref clocks to centaur

6.10. host_slave_sbe_config

a) 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.

b) **proc_setup_sbe_config.C**

- Update SBE config data area with any configs/parameters required by SBE (see step 0 for more details)

6.11. host_sbe_start

a) ***proc_start_sbe.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

6.12. proc_check_slave_sbe_seeprom_complete : Check Slave SBE Complete

a) ***proc_check_slave_sbe_seeprom_complete.C***

- Check to make sure that the slave SBE engines have completed
  - SBE state reg to make sure that slave SBE has finished
  - Check for specific "done" signature in status reg (not tied to istep number)
- If they haven't wait 1 second and then generate error
- Also checks for error status codes

b) ***proc_extract_sbe_rc.C -soft_err***

- Called on all chips (master and slaves) 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

6.13. proc_xmit_sbe : vSBE Init of Slave Chips

a) Left as placeholder if step 3 is needed by master

b) ***proc_xmit_sbe.C***

- Loads SBE PNOR image from PNOR into memory (L3)
  - Scan escape
  - Customizes parameters for slave chip (system settings, core good, etc)
- Uses virtual vSBE engine to setup slave chips

- vSBE provides callbacks that Hostboot fills in
  - Sends output stream of get/putscoms to slave chips across FSI to slave chips

7. Step 7 **Hostboot – Nest Chiplets**

7.1. proc_attr_update :Proc ATTR Update

a) ***proc_attr_update.C***

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

7.2. proc_a_x_pci_dmi_pll_initf : PLL Initfile for A, PCIe, DMI

a) ***proc_a_x_pci_dmi_pll_initf.C***

- PLL rings are stored as system attributes
  - Included tune bits, frequency

7.3. proc_a_x_pci_dmi_pll_setup : Setup PLL for A, X, PCIe, DMI

a) ***proc_a_x_pci_dmi_pll_setup.C***

- Checks that the PLL locked
- Start the VAR OSCs / Config the TANK PLLs & lock
- In certain configs these chiplets are potentially not used
- Must run at system frequency

7.4. proc_startclock_chiplets : Start clocks on A, X, PCIe chiplets

a) proc_startclock_chiplets.C

- Start Xbus, Abus, PCIe clocks
  - Start clocks on configured chiplets for all chips (master and slaves)
- Drop fences (RI/DI)

7.5. proc_chiplet_scominit : Apply scom inits to chiplets

a) ***proc_chiplet_scominit.C***

- Initfiles in procedure defined on VBU ENGD wiki
- Apply scom overrides to all good chiplets (except EX and MC)
- Powerbus – Apply system Pbus config to slave PB (scoms to CD NEXT)?

b) ***proc_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)

7.6. proc_xbus_scominit : Apply scom inits to Xbus

a) ***proc_xbus_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)

7.7. proc_abus_scominit : Apply scom inits to Abus

a) ***proc_abus_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)

7.8. proc_pcie_scominit : Apply scom inits to PCIechiplets

a) ***proc_pcie_scominit.C***

- 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

7.9. proc_scomoverride_chiplets : Apply sequenced scom inits

a) ***proc_scomoverride_chiplets.C***

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

Master Processor Chiplet State

Chiplet	Clock source	Scan0, Array Init	Repair Data?	Override Scans?	Clocks On?	Comments
TP	Nest PLL	X	X	X	X	Fully functional
TP – OCC	Nest PLL	X	X	X		
Pbus	Nest PLL	X	X	X	X	Powerbus in island mode
Pbus – MCS	Nest PLL	X	X	X	X	
Xbus	Xbus PLL	X	X	X	X	
Abus	Abus PLL	X	X	X	X	
PCIe	PCIe PLL	X	X	X	X	
MBS (Centaur)						
MBA (Centaur)						
EX – Boot	DPLL	X	X	X	X	Executing Hostboot
EX – Other	Bypass pll	X	X			

8. Step 8 **Hostboot – EDI, EI Initialization**

8.1. fabric_erepair : Restore Fabric/EDI Bus eRepair data

a) Restore/preset bad lanes on A, and X, buses from VPD

- ***io_restore_erepairr.C*** (A, X bus target pairs)
  - procedure for repairs on X and A buses
- Applies powerbus repair data from module vpd (#ER keyword in VRML VWML)
- Applies centaur data from planar prom (planar centuars), centuar dimm prom (centuar dimm)
- Runtime detected fails that were written to VPD are restored here

8.2. fabric_io_dccal : Calibrate Fabric/EDI interfaces

a) ***io_dccal.C*** (A, X bus target pairs passed in)

- Will be called per bus target pair
- For CCM the A bus io_dccal must be done by the SP for “hot add/remove” buses on active nodes. HB

does new node side

- Calibration of TX impedance, RX offset for A and X busses
- Needs to be quiet on the bus – drivers are quiesced and driving 0s – A, X buses
- Must be complete on ALL chips before starting next A, X bus training
- Expect to use a calculation
- At end of offset calibration there may be a lane that is bad
  - FW must record bad lane and write to VPD for future eRepair (handled when PRD starts)
  - Must generate error log, procedure will mark lane bad in HW (which future procedure take advantage of)

b) ***io_new_procedure.C?***

8.3. fabric_pre_trainadv : Advanced pre EI/EDI training

a) ***io_pre_trainadv.C*** (called on each A and X bus target pair)

- Debug routine for IO Characterization
- Nothing in it

8.4. fabric_io_run_training : Run training on internal buses

a) ***io_run_training.C (called on each A and X bus target pair)***

- **Hostboot will run training on all intra node buses. For Murano this is all A and X buses. For Venice (ie Rayner) this is run by the SP in a later step**
- Start wder on all slave bus endpoints
- Start wder on all master bus endpoints
- Wiretest, Deskew, Eye Optimization, and repair
  - Option to run extend bit patterns in optimization phase (replaces RDT)
  - Repairable fails are left for PRD to analyze and move data into VPD
    - PRD will use ***io_eRepair_read.C*** to perform this
  - Fatal bus training errors are handled by procedure, must return error and FFDC (written to VPD for PgP)
  - Expected that fatal error passes returncode back to HWPF, FW then looks up returncode and determines what to do based off of errorInfoRepository

8.5. fabric_post_trainadv : Advanced post EI/EDI training

a) ***io_post_trainadv.C*** (called on each A and X bus target pair)

- Debug routine for IO Characterization
- Nothing in it

8.6. host_startprd_pbus : Load PRD for powerbus domain

a) Hostboot will apply fabric topology configuration based on Abus, Xbus, and processor chip deconfiguration

b) Start Host PRD just for powerbus domain

c) Must be minimal environment (running out of L3)

8.7. host_attnlisten_proc : Start listening for attentions

a) Enable hostboot to start including processor (all) attentions in its post istep analysis

b) From this point on ATTN/PRD will listen ("poll") for powerbus attentions after each named istep

8.8. proc_fab_iovalid : Lower functional fences on local SMP

a) ***proc_fab_iovalid.C*** (called on A and X bus target pair)

- Lower fences on internal fabric X buses

    - Lower fences on external fabric (A) buses
    - Only performed on trained, valid buses
  b) After this point a checkstop on a slave will checkstop master

9. Step 9 **Hostboot – Activate PowerBus**

9.1. proc_build_smp : Integrate PgP Islands into SMP

a) ***proc_build_smp.C***

- Look for checkstops
- CCM all PgP islands into the SMP
- Fabric config between dSMP and 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
- High level flow
    - Preconditions:
        - Slave powerbus init'ed via vSBE
        - Slave chips Pbus have run fabric init via vSBE
            - CD CURR/NEXT is "island config" and AB CURR/NEXT is "island" mode
        - EI/EDI buses are up and alive
        - HB is loaded on master with AB CURR as island mode
        - Apply system Pbus config to both master/slave PB (scoms to CD NEXT)
    - Use ADU to switch CD to NEXT on **master chip** (Master Pbus has system config)
    - via FSI use ADU to switch CD to NEXT on all **slave chips** (Slave chip have system config)
    - Via FSI quiesce slave fabric
    - Via FSI apply system fabric topology to AB NEXT and AB CURR setting on slave chips
    - Via Xscom apply system fabric topology to AB Next on master
    - Apply switch AB to master via ADU
        - Full powerbus topology comes alive following init in the switch AB
    - Via Xscom write master AB NEXT to system fabric topology

9.2. host_slave_sbe_update

a) Placeholder for Secureboot where Hostboot must update SEEPROM because the SP cannot. It is at this step in the IPL so it can be updated via Xscom (trusted path) on all chips in the system

b) ***p8_customize_image.C***

- If needed build a custom SEEPROM image for each chip in the system off of the base IPL SEEPROM image

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

Master, Slave Processor Chiplet State

Chiplet	Clock source	Scan0, Array Init	Repair Data?	Override Scans?	Clocks On?	Comments
TP	Nest PLL	X	X	X	X	Fully functional
TP – OCC	Nest PLL	X	X	X		
Pbus	Nest PLL	X	X	X	X	Local drawer SMP
Pbus – MCS	Nest PLL	X	X	X	X	
Xbus	Xbus PLL	X	X	X	X	

Abus	Abus PLL	X	X	X	X	
PCIe	PCIe PLL	X	X	X	X	
MBS (Centaur)						
MBA (Centaur)						
EX – Boot	DPLL	X	X	X	X	Executing Hostboot
EX – Other	Bypass pll	X	X			

10. Step 10 **Hostboot SBE – Centaur Init**

The following steps are part of the Centaur SBE Init Image. Note that these cannot be stepped in hostboot, but can be in Cronus (due to POREVE implementation). When running hostboot when it is requested to execute step 10.2 it will perform all substeps in step 10. From a command line interface all the rest of the steps should be stubbed and return success. Only when running in Cronus only (vSBE) mode will steps 10.3-10.x do anything. The HW reconfig loop is only possible on processor DD2.x or higher parts. FW is responsible for disabling the HW reconfig loop on D1.X parts.

Hostboot will check for HW reconfig loop after then end of each named istep. If it happens in 10,11,13, or 14 it will go back to the beginning of step 10 (known as HW reconfig loop). If it fails in step 12 it will go back to the beginning of step 12 (FW reconfig loop).

Note that this is the step for HW Reconfig to restart on centaur or dimm training/init fails

The following sections are to initialize the Centaur TP logic (Host TP Initialization)

10.1. host_prd_hwreconfig : Hook to handle HW reconfig

a) This step is always called

b) Move all centaur's inband scom back to FSI scom

c) Call PRD to allow them to rebuild model to remove centaurs

d) 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:

- ***proc_enable_reconfig.C***
  - Call on all present processor, MCS, and centaur targets. Called per processor and associated targets
  - Enables HW for reconfig loop
  - Add a Chip EC feature attribute to check support of special MCS bit
    - Check this when setting the bit to bring the DMI bus down
  - Add an attribute (ATTR_MSS_INIT_STATE) to each centaur to track where the Reconfig loop got too:

    Clocks on (can do fir masking) – set after step 10
    - DMI bus up (inject special bit) – set after framelock
  - Turn's on special bit that allows the MCS DMI to get errors and not get into a hang condition
  - Mask a bunch of FIRs on processor
  - Mask a bunch FIRs on centaur (HWP will check clock state)
  - Injects a fail on the DMI bus (only if DMI bus is alive)
  - Clears IO/MCS FIRs
  - Turns off special bit

cen_sbe_istep_pnor.S

- This procedure is not an official istep under FW/Cronus control
- It is a wrapper that allows for individual istep control
-

10.2. cen_sbe_tp_chiplet_init1 : TP Chiplet Init

a) cen_sbe_tp_chiplet_init1.S

  - Flush all GP reg content to default state

10.3. cen_sbe_pll_initf : Program Nest PLL

a) cen_sbe_pll_initf.S

- Apply the Nest PLL ring
  - Initfile defined on VBU ENGD wiki
-

10.4. cen_sbe_pll_setup : Setup Nest PLL

a) cen_sbe_pll_setup.S

- Performs PLL checking
- The memory PLL (ie DDR3) are set to a default value (not actual runtime speeds)
- Establish Nest PLLs (feeds TP chiplet)
  - Includes scans for all PLL (Nest, EDI) setup
  - Moved out of bypass(just nest)

10.5. cen_sbe_tp_chiplet_init2 : Cen TP Chiplet Init 2

a) ***cen_sbe_tp_chiplet_init1.S***

- Scan 0 init TP unit (flush)
- No repair/timing for TP chiplet (ie fuses)

10.6. cen_sbe_tp_arrayinit : Cen TP Chiplet array init

a) cen_sbe_tp_arrayinit.S

- Run arrayinit on TP chiplet
  - After this all arrays are initialized
- Scan flush 0 to all rings except GPTR, Time, Repair

10.7. cen_sbe_tp_chiplet_init3 : Cen TP Chiplet Start clocks

a) cen_sbe_tp_chiplet_init3.S

- Start clocks on perv region
- At end, the TP chiplet can be used to init the rest of the chip
- Involves writing FSI GP3 to switch mux – all access now go through TP chiplet

The following sections are to initialize the Centaur chip logic (Host Chiplet Setup)

10.8. cen_sbe_chiplet_init : Cen Chiplet Init

a) cen_sbe_chiplet_init.S

- Scan 0 all rings on all good chiplets (except for TP)
- Kick off the fuse repair algorithm (loads repair ring)

10.9. cen_sbe_arrayinit : Chiplet array init

a) cen_sbe_arrayinit.S

- Run arrayinit on all good chiplets (except for TP)
  - After this all arrays are initialized
- Scan flush 0 to all rings except GPTR, Time, Repair

- *Note the MBA PLLs are controlled later by Hostboot*

b) *Note: If LBIST was to be run, it should be run after this step, prior to the next step*

The following sections are to initialize the Centaur chip logic (Host Chiplet Initialization)

10.10. *cen_sbe_dts_init : Cen DTS Init*

a) cen_sbe_dts_init.S

- Setup the thermal sensors based on fuse data

10.11. *cen_sbe_initf : Cen Scan overrides*

a) cen_sbe_chiplet_initf.S

- Perform any scan overrides for Centaur
  - May not have any config dependent scans
- Does not include the pervasive region

10.12. *cen_sbe_do_manual_inits : Manual Cen Scans*

a) cen_sbe_do_manual_inits.S

- Perform any non initfile scan overrides for Centaur
- Should be avoided, provided as a placeholder for workarounds

The following sections are to initialize the Centaur chip Scom logic

10.13. *cen_sbe_nest_startclocks : Start Cen Nest*

a) cen_sbe_nest_startclocks.S

- Starts Centaurs's nest clocks. This includes the L4, DMI, nest portions of logic. It specifically excludes the MBA clocks
- Deassert the memrst_b GP bit to activate the reset_OE signal
- Enable driver and receivers (set appropriate GP bits)
- Lower RI and DI inhibits

10.14. *cen_sbe_scominits : Perform any Cen scom inits*

a) cen_sbe_scominits.S

- Any needed scom initializations – no config dependent settings allowed
- Should be empty

Master, Slave Processor Chiplet State

Chiplet	Clock source	Scan0, Array Init	Repair Data?	Override Scans?	Clocks On?	Comments
TP	Nest PLL	X	X	X	X	Fully functional
TP – OCC	Nest PLL	X	X	X		
Pbus	Nest PLL	X	X	X	X	Local drawer SMP
Pbus – MCS	Nest PLL	X	X	X	X	
Xbus	Xbus PLL	X	X	X	X	
Abus	Abus PLL	X	X	X	X	
PCIe	PCIe PLL	X	X	X	X	
MBS (Centaur)	Cen Nest PLL	X	X	X	X	

MBA (Centaur)	Bypass PLL?	X	X	X		
EX – Boot	DPLL	X	X	X	X	Executing Hostboot
EX – Other	Bypass pll	X	X			

11. Step 11 **Hostboot – DMI Training**

11.1. mss_getecid : Read out ECID of all Centaurs

a) mss_get_cen_ecid.C

- Read the ECID for each centaur and store away for callouts. Also update functional state of MBAs and L3 cache halves for later use
- Sets ATTR_MSS_INIT_STATE to "clocks on"

11.2. dmi_attr_update :DMI ATTR Update

a) dmi_attr_update.C

- Called per MCS and Centaur
- Stub HWP for FW to override attributes programatically

11.3. proc_dmi_scominit : DMI Scom setup on P8 MCS

a) proc_dmi_scominit.C

- Initfiles in procedure defined on VBU ENGD wiki
- Called per P8 MCS
- Perform scom inits for DMI on the processor

11.4. dmi_scominit : Scom setup on centaur

a) ***cen_dmi_scominit.C***

- Initfiles in procedure defined on VBU ENGD wiki
- Perform scom inits for DMI on Centaur

11.5. dmi_erepair : Restore EDI Bus eRepair data

a) Restore/preset bad lanes on MC buses from VPD

- Bad lanes are preset on the receive side
- ***io_restore_erepair.C (MC and Centaur target pair passed in)***
  - procedure for repairs on MC bus
- Applies centaur data from planar prom (planar centuars), centuar dimm prom (centuar dimm)
- Runtime detected fails that were written to VPD are restored here

11.6. dmi_io_dccal : Calibrate DMI interfaces

a) **io_dccal.C** (MC and Centaur target pair passed in)

- Calibration of TX impedance, RX offset for memory buses
  - Needed for EDI buses on p8 and centaur (need to run in order to guarantee clean clock)
- Needs to be quiet on the bus – drivers are quiesced and driving 0s – EDI buses
- Must be complete on ALL centaurs for this PgP island before starting next EDI training(host based sync point)
- At end of offset calibration there may be a lane that is bad
  - FW must record bad lane and write to VPD for future eRepair (handled when PRD starts)

11.7. dmi_pre_trainadv : Advanced pre DMI training

a) ***io_pre_trainadv.C*** (MC and Centaur target pair passed in)

- Debug routine for IO Characterization

- Nothing in it

11.8. dmi_io_run_training : Run training on MC buses

a) ***io_run_training.C*** (MC and Centaur target pair passed in)

- Train internal DMI bus
- Wiretest, Deskew, Eye Optimization, and repair
  - Option to run extend bit patterns in optimization phase (replaces RDT)
  - Wiretest fails are left for PRD to analyze and move data into VPD
  - Fatal bus training errors are handled by procedure and written to VPD

11.9. dmi_post_trainadv : Advanced post DMI training

a) ***io_post_trainadv.C*** (MC and Centaur target pair passed in)

- Debug routine for IO Characterization
- Nothing in it

11.10. proc_cen_framelock : Initialize EDI Frame

a) ***proc_cen_framelock.C***

- Raise IO Valid – Allow link init traffic (scrambled patterns) on EDI bus
- P8 Centaur initial frame lock
  - Starts listening automatically after IOValid raised
  - Started on the P8 logic
  - If a bit error (CRC) in the middle need to reFrameLock
- Round trip delay calculation
  - Host code can trigger and check
- Inband accesses are now viable
- Hardware xmitting idle frames
- Enabled CRC checking
- EDI is at runtime state
- If successful, set ATTR_MSS_INIT_STATE to DMI active on centaur

11.11. host_startprd_dmi : Load PRD for DMI domain

a) Expand Host PRD to include DMI (as well as powerbus)

b) Must be minimal environment (running out of L3)

11.12. host_attnlisten_cen : Start listening for attentions

a) Enable hostboot to start including centaur attentions in its post istep analysis

11.13. cen_set_inband_addr : Set the Inband base addresses

a) ***proc_cen_set_inband_addr.C***

- Any initializations to setup the Inband access path
  - MCS – Scom base address
  - Centaur – any other settings
- **ALL ACCESES from this point on in are Inband access for Centaur unless otherwise specified**

12. Step 12 **Hostboot – MC Config**

Note that the "FW Reconfig" loop starts here (since it doesn't touch HW). Any reconfig during step 12 will loop back to this step

12.1. host_collect_dimm_spd : Collect Master dimm SPD

a) This step is a placeholder – The SP will have placed all dimm SPD into PNOR prior to releasing the SBE, so Hostboot consumes the dimm SPD directly from PNOR

b) Hostboot knows the relationship between dimm, SPD, chip, voltage rail, and socket

c) Hostboot FW then places dimm SPD of interest into HWPF attributes

- This may be "on demand" as attributes are looked up

d) mss_attribute_cleanup.C

- Called on all present centaurs
- Hook to clean up attributes on reconfig loop (set to known state)

e) For the inner reconfig loop

- Call PRD to allow them to rebuild model to remove centaurs
- ***proc_enable_reconfig.C***
    - FW must only call present, non functional (processor, MCS, and centaur) targets
    - Must not be called on functional parts
    - Does the same thing as above (10.1)

12.2. mss_volt : Calc dimm voltage

a) mss_volt.C

- Procedure is called all the dimms on a voltage rail
- Calculate rail Voltage and updates rail system attribute
- Save settings in variables (saved in framework/cache)
- Procedure handles checking overrides

12.3. mss_freq : Calc dimm frequency

a) mss_freq.C

- Procedure is called on each MC in the system
- Looks at voltage and dimm functionality
- Calculate per memory controller frequency from attributes – picks the frequency bucket to use
- Save settings in variables (saved in framework/cache)
- Procedure handles checking overrides

12.4. mss_eff_config : Determine effective config

a) ***mss_eff_config.C***

- Called per MC
- Uses HWPF attributes to determine config (SPD/PNOR overrides was stored previously in step 5)
- Determine effective settings for settings – determine "final" solution
- Looking at overrides and applies
- Extent settings are stored as attributes per target (may be changed later based on training)

b) ***mss_eff_config_thermal.C***

- Called per MBA
- Perform thermal calculations for the effective config
- Split out so SP can call earlier for ITE power curves

c) In order to ensure optimal placement of memory (ie as contiguous as possible for the hypervisor) the following sequence must be followed

- ***opt_memmap.C***
    - Called with init = true to zero out all relevant attributes

- ***mss_eff_grouping.C***
  - Called on each P8 target. Note that all P8 base addresses must be zero
  - This is done the first time to get the available sizes behind all P8s
- ***opt_memmap.C***
  - Called with init=false
  - All P8 on a physical drawer are passed in to procedure
  - FW written procedure that then walks the P8 mappings and orders the base addresses to form as contiguous a memory map as possible
  - This is a FAPI procedure so it can be shared with Cronus
- ***mss_eff_grouping.C***
  - Called on each P8 target. Note that all P8 base addresses are now set optimally
  - This will then slot the maps in the correct places
- ***mss_eff_mb_interleave.C***
  - Called on each centaur target.
  - This sets up the MBA interleaving internal to the cenatur

12.5. mss_attr_update :MSS ATTR Overrides

a) mss_attr_update.C

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

13. Step 13 **Hostboot – DRAM Training**

13.1. host_disable_vddr : Disable VDDR on CanContine loops

a) Power off dram. Must be done here to meet JEDEC spec compliance

- Turned off here to handle CanContinue loop for dimm failure
- Only really issued if VDDR is on
- **On MPIPL this is already on – don't touch**

13.2. mem_pll_initf : PLL Initfile for MBAs

a) cen_mem_pll_initf.C

- MBA PLL setup – get tune bits, frequency from attributes
  - Note that Hostboot doesn't support twiddling bits, looks up which “bucket” (ring) to use from attributes set during mss_freq
  - Data is stored as a ring image that is frequency specific
    - 6 different frequencies – 6 different ring images
- Must run at system frequency

13.3. mem_pll_setup : Setup PLL for MBAs

a) ***cen_mem_pll_setup.C***

- MBA PLL setup
  - Moved PLL out of bypass(just DDR)
- Performs PLL checking

13.4. mem_startclocks : Start clocks on MBAs

a) ***cen_mem_startclocks.C***

- Drop fences and tholds on MBAs

13.5. host_enable_vddr : Enable the VDDR3 Voltage Rail

a) Bring power to dram. Must be done here to meet JEDEC spec compliance
  - **On MPIPL this is already on – don't touch**

b) Send message to SP to turn on voltages
  - Message must have accounted for voltage/current tweaking based on number of plugged dimms (Dynamic VID)
  - Pulled from HWPF attributes per voltage rail
  - SP
    - Trigger voltage ramp to DPSS via I2C
    - Wait for min 200 ms ramp, must be stable 500us after DPSS claims Pgood

c) Wait for ack message from SP – confirms that voltage is on and ready

13.6. mss_scominit : Perform scom inits to MC and PHY

a) ***mss_scominit.C***
  - Called per Centaur, pass in associated MBAs
    - HW units included are MBS, MBX, MBAs, and PHYs
    - Initfiles: cen_ddrphy.initfile, cen_mba.initfile, cen_mbx.initfile and cen_mbs.initfile
    - HW calls multiple initifles per target
  - FAPI provides scom initfile handling
  - Uses attributes from previous step

b) ***proc_throttle_sync.C***
  - Must be issued on all P8s, can only be issued after ALL centaurs on given p8 have scominit complete (can also loop at the end)
  - Triggers sync command from MCS to actually load the throttle values into the centaurs

13.7. mss_ddr_phy_reset : Soft reset of DDR PHY macros

a) ***mss_ddr_phy_reset.C*** : Soft Reset the DDR phy macros
  - DDR PLLs
    - Already configured DDR PLL in scaninit
    - Take PLL out of reset (controlled by a GP3 bit)
  - Sends Soft DDR Phy reset
  - Kick off internal ZQ Cal
  - Perform any config that wasn't scanned in (TBD)
    - Nothing known here

13.8. mss_draminit : Dram initialize

a) ***mss_draminit.C*** :Init the DDR logic
  - De-assert dram reset
  - De-assert bit (Scom) that forces mem clock low – dram clocks start
  - Raise CKE
  - Load RCD Control Words
  - Load MRS – for each dimm pair/ports/rank
    - ODT Values
    - MR2, MR3, MR1, MR0

13.9. mss_draminit_training : Dram training

a) ***mss_draminit_training.C***

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
  - Marks the calibration fail array
- External ZQ Calibration
- Execute initial dram calibration (7 step – handled by HW)
- This procedure will update the bad DQ attribute for each dimm based on its findings

13.10. mss_draminit_trainadv : Advanced dram training

a) ***mss_draminit_training_advanced.C***

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
  - Marks the MCBist mask
- This step will contain any algorithms to improve eye
- Also will contain some characterization (mfg only) tests
- All optional tests must be their own functions and C files to optimize load and execution
- This procedure will update the bad DQ attribute for each dimm based on its findings

13.11. mss_draminit_mc : Hand off control to MC

a) ***mss_draminit_mc.C***

- Set IML complete bit in centaur
- Start main refresh engine
- Refresh, periodic calibration, power controls
- Unmask memory FIRs
- Turn on ECC checking on memory accesses

b) Note at this point memory FIRs can be monitored by PRD

13.12. mss_dimm_power_test : Hand off control to MC

a) ***mss_dimm_power_test.C***

- Pass in all dimms on a given power domain
- Will be a no-op for CDIMMs (FW can always call)
- For IS dimms measure the power

14. Step 14 **Hostboot – DRAM Initialization**

14.1. host_startprd_dram : Load PRD for DRAM domain

a) Expand Host PRD to include DRAM (as well as powerbus and DMI)

b) Must be minimal environment (running out of L3)

14.2. mss_extent_setup

a) mss_extent_setup.C

- Pushes memory extent configuration into the centaurs via Inband
- Addresses are pulled from attributes, set previously by mss_eff_config
- Centaurs always start at address 0, address map controlled by proc_setup_bars below

14.3. mss_memdiag : Mainstore Pattern Testing

a) The following step documents the generalities of this step

- In FW PRD will control mem diags via interrupts. It doesn't use mss_memdiags.C directly but the HWP subroutines
- In cronus it will execute mss_memdiags.C directly

b) mss_memdiags.C

- Prior to running this procedure will apply known DQ bad bits to prevent them from participating in training. This information is extracted from the bad DQ attribute and applied to Hardware
- Will use traditional scrub engine
- Still supports superfast
- Modes:
    - Minimal: Write-only with 0's
    - Medium: Write-followed by Read, 4 patterns, last of 0's
    - Max: Write-followed by Read, 9 patterns, last of 0's
- Final “init” done with scrub engine to ensure good ECC
    - Note that this does not need to be zeros, but it is typically zeroed (hypervisor will zero again)
- 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

c) All subsequent repairs are considered runtime issues

14.4. mss_thermal_init : Initialize the thermal sensor

a) mss_thermal_init.C

- Called on Centaur target
- Setup and configure I2C thermal sensor on dimms
- Configure and start centaur thermal cache
- Configure and start the OCC cache
- Disable safe mode throttles
    - Will cause memory to go to runtime emergency throttles
    - When OCC starts polling OCC cache will revert to runtime settings

b) proc_throttle_sync.C

- Must be issued on all P8s, can only be issued after ALL centaurs on given p8 have thermal init complete (can also loop at the end of all centuars)
- Triggers sync command from MCS to actually load the throttle values into the centaurs

14.5. proc_pcie_config : Configure the PHBs

a) proc_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

14.6. mss_power_cleanup : Clean up any MCS/Centaurs

a) mss_power_cleanup.C

- Called on all present Centaurs and MBAs
- Cleans up and powers down unused cenaturs/mcs/DMI
- **After this step no more HW reconfig loops (back to step 10) are allowed – this step and future ones will not perform the reconfig**

loop check

- Hostboot will start to flow out to memory in the next step
- Any memory errors after this point are considered “runtime errors”
- All errors from this point on have to be a no deconfig and gard OR terminate the IPL (and let the SP do the reconfig)
- If user attempts to do a deconfig outside the loop – then attempt to fail

14.7. proc_setup_bars : Setup Memory BARs

a) mss_setup_bars.C

- Based on system memory map
  - Each MCS has its mirroring and non mirrored BARs
  - Set the correct checkerboard configs. Note that chip flushes to checkerboard
  - need to disable memory bar on slave otherwise base flush values will ack all memory accesses

b) proc_setup_bars.C

- Sets up Powerbus/MCD, L3 BARs on running core
  - Other cores are setup via winkle images
- Setup dSMP and PCIe Bars
  - Setup PCIe outbound BARS (doing stores/loads from host core)
    - Addresses that PCIE responds to on powerbus (PCI init 1-7)
  - Informing PCIe of the memory map (inbound)
    - PCI Init 8-15
- Set up Powerbus Epsilon settings
  - Code is still running out of L3 cache
  - Use this procedure to setup runtime epsilon values
  - Must be done before memory is viable

14.8. proc_exit_cache_contained : Allow execution from memory

a) ***proc_exit_cache_contained. C***

- Allow execution to flow out to memory
- Must clear CoreFabricProtect bit
- Must clear NestFabricProtect bit
- Must set all centaur's secureboot bits to prevent memory D/A

b) Data rolls out to memory

14.9. host_mpipl_service : Perform MPIPL tasks

a) This is a no-op for warm/cold IPLs. See description in Error: Reference source not found, Section Error: Reference source not found for full details

15. Step 15 **Hostboot – Build Winkle Images**

15.1. host_build_winkle : Build runtime winkle images

a) Pull Reference Image from PNOR

- Run through secure boot algorithm

b) p8_slw_build.C

- Called for each processor chip.
- Parameter: Pointer to Reference image.

- Parameter: Pointer to Output winkle image.
  - This is any Hostboot specified mainstore location (does not have to be attached to the processor being winkled).
  - When hypervisor is loaded, the winkle images will be trampled, hypervisor will call p8_slw_build to recreate them in a hypervisor 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

c) ***p8_pore_gen_cpu_reg()***

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

15.2. proc_set_pore_bar : Tell SWL Eng where winkle image is

a) Cronus will load the images via procputmem

b) p8_set_pore_bar.C

- Called for each processor chip.
- Parameter: Physical address of SLW image (not a pointer address, it cannot be dereferenced)
- Parameters: PBA BAR number, SLW image size, SLW image location (default: mem; others: L3, SRAM)
- ***p8_pba_bar_config.C (called as subroutine)***
  - Set BAR address

15.3. host_poreslw_init -init : Initialize the PORE-SLW engine and related functions to allow assisted idles to operate

a) p8_poreslw_init.C chip_target, ENUM:PM_INIT

- Called for each processor chip
- Parameters: none other than target
- Set and then clear PMC reset to clear any latent idle transitions that may be queued (in the case this flow is executed as part of a warm IPL or MPIPL.
- **Note** this clears ALL PMC settings. However, previously issued SPIVID operations will remain intact.
- ***p8_pmc_deconfig_setup.C (called as subroutine)***
  - Reads the EX chiplet enable (GP0(0)) and sets the PMC_CORE_DECONFIGURATION_REG for any non-configured EX chiplets
  - This is necessary to enable the PMC to accept idle entry/exit requests from on the validly enabled PCBS. This setting is also necessary for proper operation of the Pstate mechanism.
- Make sure the PMC Idle Sequencer is enabled
  - Clears PMC_MODE_REG(14) - HALT_IDLE_STATE_MASTER_FSM
- Enables OHA to accept idle operations
  - Clear OHA_MODE_REG(6) - OHA Idle State Override Enable
- Activates the PCBS-PM macro
  - Clear PMGP0(0) (PM_DISABLE) as this is necessary for both idle and Pstate operation. Hardware default is to come up inactive.

- Clear the OCC Special Wake-up which is initialized ON by hardware default to keep idle operations from occurring until the SLW image is installed.

16. Step 16 **Hostboot – Core Activate**

16.1. host_activate_master : Activate master core

a) ***proc_prep_master_winkle.C***

- This procedure checks to make sure that the SBE program to exit winkle is already running from PIBMEM (left running from initial SBE steps). For both real SBE and vSBE it triggers deadman loop
    - ***proc_sbe_trigger_winkle.S (loaded into PIBMEM prior or via vSBE)***
        - Monitor master winkle and trigger winkle exit when it detects winkle enter. (Note that this is running in parallel with Hostboot, started at end of istep 5)
        - Starts the deadman timer loop
        - Sets indication that SBE program is ready
        - Waits for winkle entered (completely entered)
        - SBE triggers winkle exit via IPI to all threads on a core. Uses istep control bits to only start specific threads if in AVP mode
        - If Hostboot does not stop deadman timer in 10 seconds checkstop system

b) proc_trigger_winkle – Hostboot path (Hostboot running)

- Hostboot function, not a HW Procedure
- ***p8_block_wakeup_intr.C -set***
    - This will prevent all interrupts/wake up sources to the core, thus allowing the next step (winkle) to work
- Issue system call to cause all threads to enter winkle. Core will then enter winkle state
    - Clear LPCR (cover not entering due to external interrupts)
    - issue winkle instruction

c) ***proc_force_winkle.C*** – **SP/Cronus path (Hostboot not running)**

- Since Hostboot is not running this procedure will force the master core to winkle
    - Master core is left in initialed, maintenance mode state after proc SBE

d) ***proc_trigger_winkle.C* – Cronus path only**

- This procedure is a NO-OP when the real SBE is executing. It is hook to allow the virtual SBE to trigger the winkle exit – ie resume execution of ***proc_sbe_trigger_winkle.S*** in vSBE mode (see above)
    - Note that this is NOT supported in SP based IPL, Cronus only

e) proc_stop_deadman_timer.C

- Hostboot runs when active, otherwise Cronus will have to execute
    - Stops the deadman timer
    - Stops the SBE program

16.2. host_activate_slave_cores : Activate slave cores

a) Hostboot active:

- Setup stack space for all slave core threads –
- Wake up all threads on all cores
    - Cores are sitting in a winkled state (flush that way)
    - Issue IPI to all slave cores to force winkle exit. Will start executing at SREST vector (0x100). Bring them into Hostboot collective

b) Hostboot not running:

- Cores come alive and into maintenance mode (LPCR not set)
- ***proc_trigger_winkle.C***
  - Called on a core target
  - SP/Cronus issue IPIs to all cores/threads in system except for those on master core

16.3. mss_scrub : Start background scrub

a) ***mss_scrub.C***

- Note that this is not executed directly by Hostboot (instead triggered by PRD), Cronus will execute HWP directly
- Start background scrubbing. First pass will trigger as fast as possible, then switch to 12h scrub cycle
- Currently Hostboot will not wait (block) before flowing out to memory
- The completion of the scrub commands must be handled by SP or (PHYP/Sapphire) PRD
- HostPRD will not be called after this point (not called for this step)

16.4. host_ipl_complete : Notify SP drawer ipl complete

a) Stop hostPRD (in anticipation that SP will take over PRD responsibilities)

b)

b) cen_switch_rec_attn.C

- Switch recoverable/special attentions from host back to SP

c) proc_switch_rec_attn.C

- Switch recoverable/special attentions from host back to SP

d) proc_switch_cfsim.C

- Switch the FSI bus control back to the SP
- Needed for Secureboot

e) Sends a message to SP 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 SP indicating that this step is done on this drawer and SP should proceed with the IPL. This message is **not** sent in istep mode

NOTE that **istep 17** is where the SP waits for Hostboot to complete the IPL

NOTE that **istep 18 and 19** is where the SP does IBM-system specific initialization. Hostboot will setup the TOD here.

20. Step 20 **Hostboot – Load Payload**

20.1. host_load_payload : Load payload

a) Execute /etc/SPinit/tf/eclipz/IplSteps/Normal/HBLoadHostOS.tf

b) build_host_data : Build the host data areas

- This step builds the HDAT data areas from attributes, VPD, etc

c) 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 hypervisor at (along with a size)
        - For initial BU (non secure mode) hypervisor will be loaded via raw DMAs
        - For secureboot hypervisor 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

d) Hostboot then performs verification on the payload via Secureboot ROM

20.2. host_load_complete : Payload is complete

a) Switch SP<->Hostboot communication to use the PSI bus instead of the PBA mechanism to avoid conflicts with SP<->OCC communication on the PBA bus

b) Mechanism for SP 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_runtime_setup

a) Note that this step is only issued to master HB instance

b) Take down any/all TCE setup

c) 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

d) Append the TPM log to HDAT structures

- Note: HB master issues IPC to HB slaves for them to update their sections

e) In AVP 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

- 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)

21.2. host_verify_hdat

a) Only issued to master HB instance

- If needed IPC to slaves to perform their tasks

b) Secureboot verification of hypervisor/AVP image load

21.3. host_start_payload

a) 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

b) 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

c) 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

d) No Hostboot code is reused, only mechanism is data passed in HDAT areas. Hostboot runtime is a part of hypervisor

21.4. host_post_start_payload : Post payload start

a) Execute /etc/SPinit/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
    - SP 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 hypervisor start (initial mailbox message down from hypervisor)

21.5. switchbcu : Switch BCU - send continue to hypervisor (G)

a) Execute /etc/SPinit/tf/eclipz/IplSteps/Normal/SwitchBCU.tf

- Mailbox handshake with hypervisor
  - wait for OLC
  - switch BCU and "continue" OLC

21.6. 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