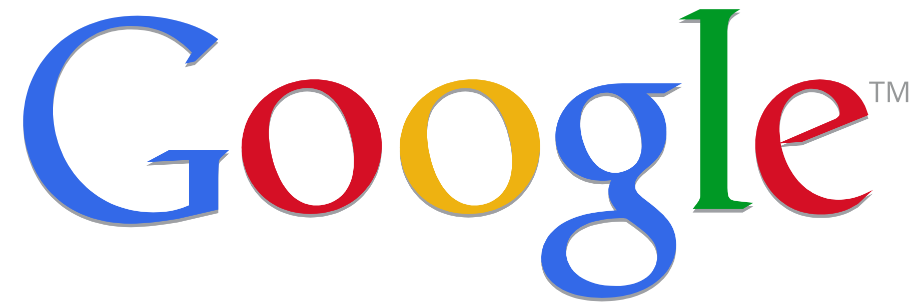
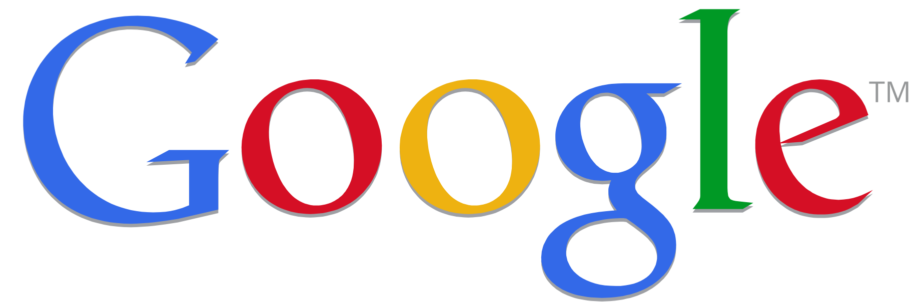


Chrome OS

Firmware Summit
February 20, 2014
firmware@chromium.org

Agenda

Overview	Duncan Laurie
coreboot Porting: x86	Aaron Durbin
coreboot Porting: ARM	Stefan Reinauer
coreboot Porting: Mainboard	Shawn Nematbakhsh
Depthcharge	Gabe Black
Embedded Controller	Bill Richardson
Chromium Process	Dave Parker Bernie Thompson



Chrome OS Firmware

Overview

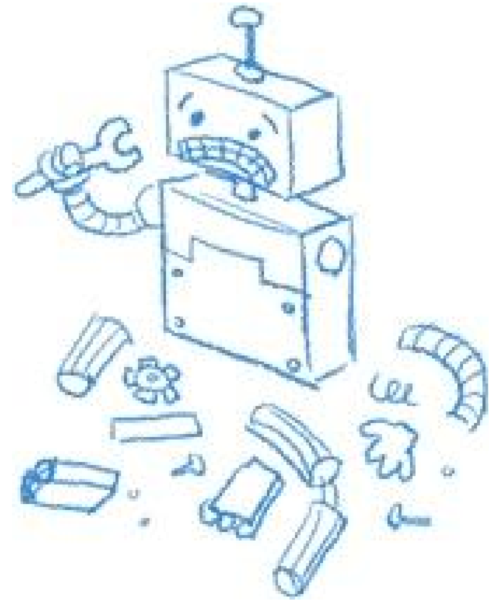
Why Invest In Firmware?

Control of the Platform

- Consistent behavior across architectures
- Maximize power and performance
- Flexible Firmware/OS interfaces
- Strong commitment to open source

Knowledge of the Platform

- Firmware is hard
- Bugs will be found
- Time is money
- Focus on security



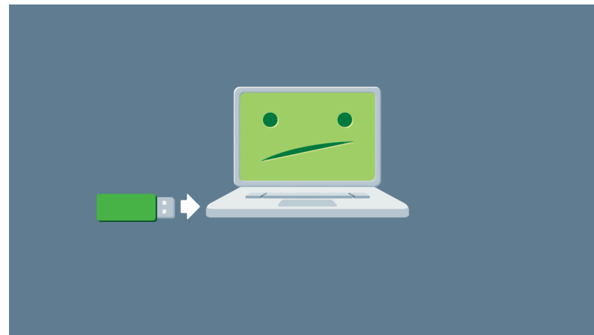
Chrome OS Boot Modes

Verified Mode

- Can only boot Google-signed Chrome OS images
- Full verification of firmware and kernel
- Read-Write firmware path
- Go to Recovery Mode if verification fails
  - Read-Write firmware modified or corrupt
  - Block device modified or corrupt
  - Hardware failure
    - TPM
    - SPI Flash
    - SSD or eMMC

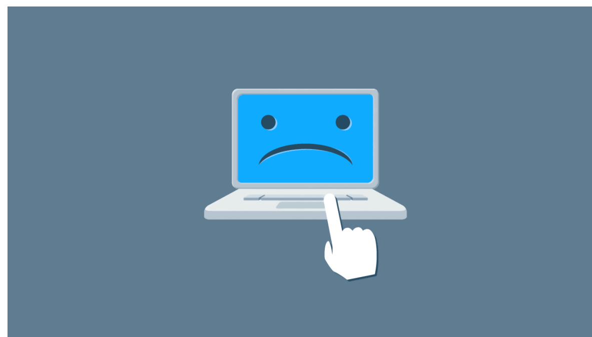
Recovery Mode

- Read-Only firmware used to boot signed USB
    - Must be signed by Google with device recovery key
- Can be initiated by the system
    - Security or Hardware fault
- Can be initiated by the user with physical presence
    - Chromebook: Keyboard ESC + Refresh + Power
    - Chromebox: Recovery button + Power button



Developer Mode

- Jailbreak mode built into every device
- Enable with Ctrl+D from Recovery Mode screen
- Transition will erase local state in TPM and disk
- Enables root shell in Chrome OS
- Enables user to boot unverified images
    - *crossystem dev_boot_usb=1*
    - *crossystem dev_boot_signed_only=0*



Legacy Mode

- Unsupported method for booting alternate OS
  - Can boot any payload
  - Separate CBFS partition in RW_LEGACY region
  - SeaBIOS on Intel Haswell generation
- Must be enabled from Developer Mode
  - *crossystem dev_boot_legacy=1*

Firmware Components

- coreboot
- Depthcharge
- Verified Boot
- Embedded Controller
- Vendor Binaries

coreboot

- GPLv2 BIOS replacement
    - Started as LinuxBIOS in 1999 by Ron Minnich
    - Renamed to coreboot in 2008 by Stefan Reinauer
- Mostly C, Assembly, and ASL
- Kconfig and modified Kbuild
- High-level organization around block diagram
    - Modular CPU, Chipset, Device support
- NOT a bootloader
    - Support for various payloads
    - Payload can boot Linux, DOS, Windows, etc

coreboot Stages

- Boot Block
  - Prepare Cache-as-RAM and Flash access
- ROM stage
  - Memory and early chipset initialization
- RAM stage
  - Device enumeration and resource assignment
  - Start other CPU cores
  - ACPI table creation
  - SMM handler
- Payload

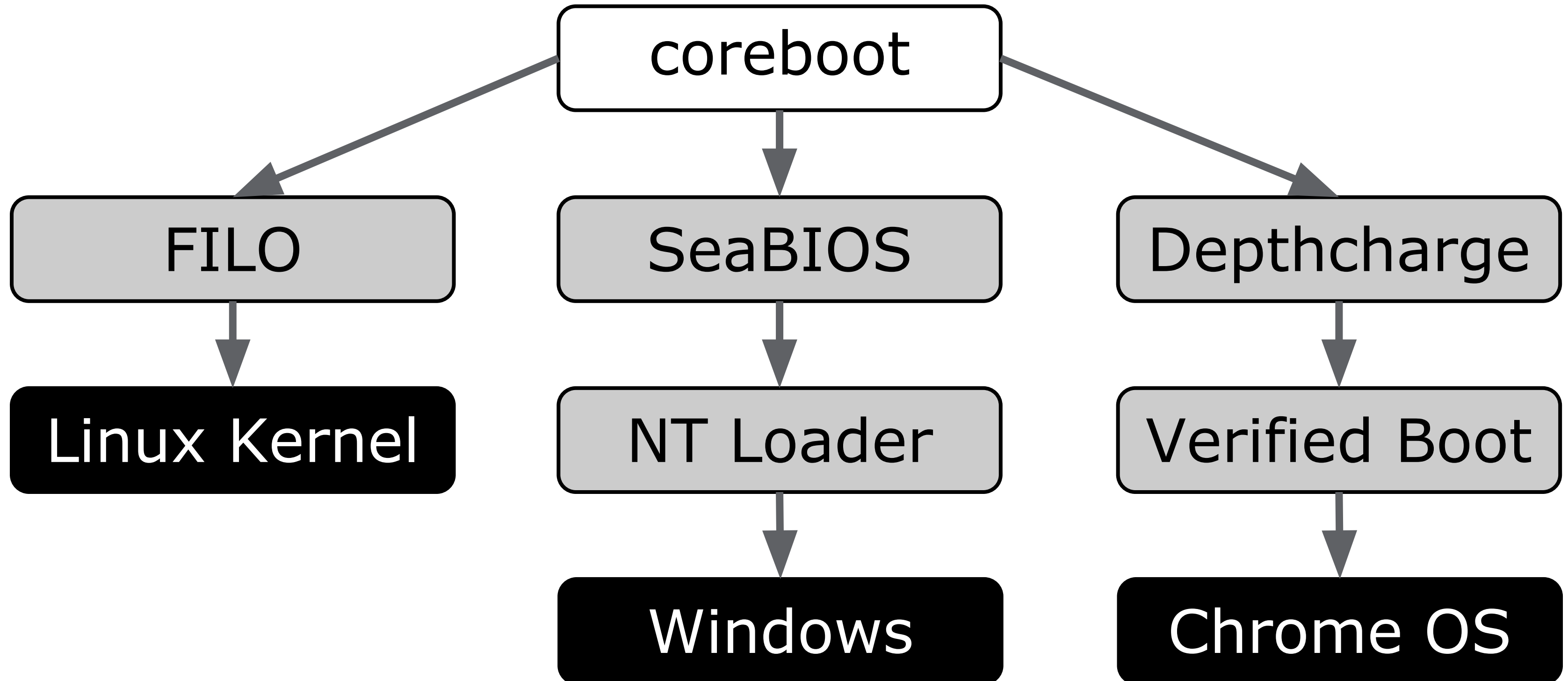
coreboot and UEFI

coreboot	UEFI
Boot Block	SEC
ROM Stage	PEI
Reference Code	
RAM Stage	DXE
Option ROMs	
Payload	BDS
Operating System	

libpayload

- Library of common payload functions
  - Subset of libc functions
  - Tiny ncurses implementation
  - Various hardware drivers
  - Read and parse coreboot table
- BSD License
- www.coreboot.org/Libpayload

coreboot Payloads



coreboot Upstream

- Rebase Chromium git repo with upstream
  - Typically after product cycle
  - Create new git branch in coreboot repo
    - remotes/m/master -> chromeos-2013.04
- Push patches upstream to review.coreboot.org
  - Gerrit for patches and code review

Depthcharge

- GPLv2 license
- Payload designed to boot Chrome OS
- Verified Boot reference implementation
- Does not boot other operating systems
    - Can chain to another payload

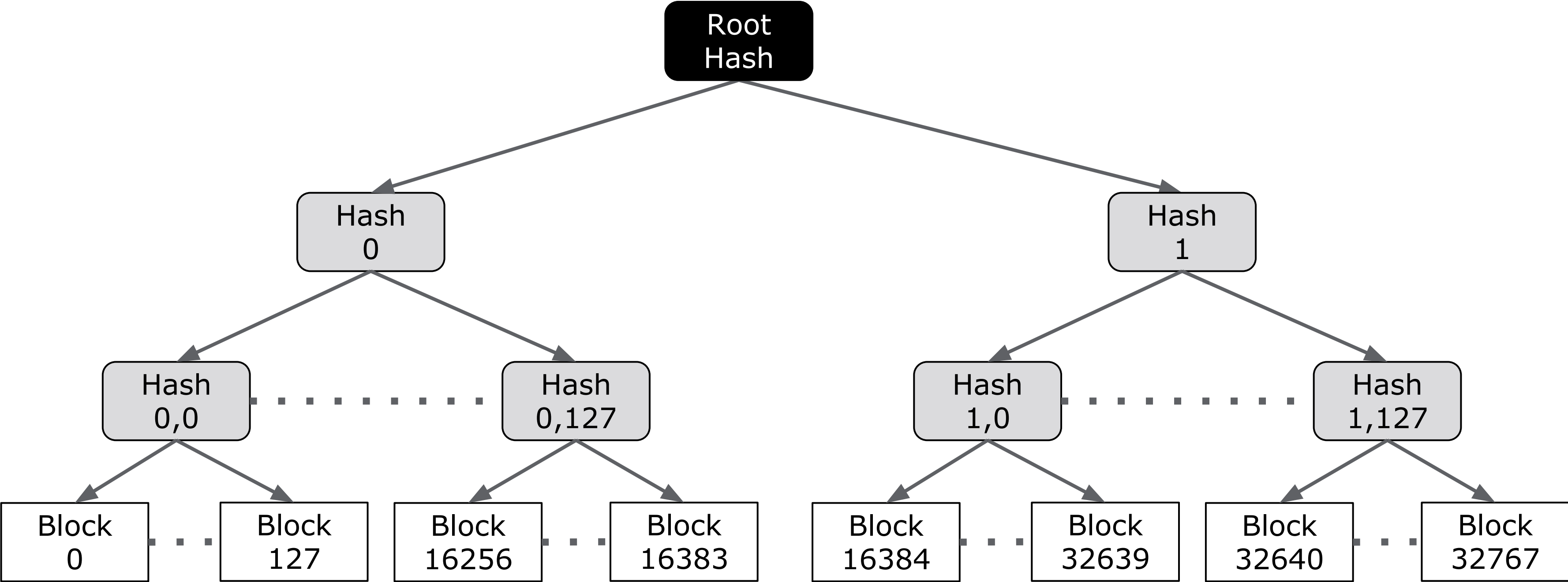
Verified Boot Firmware

- Method to ensure that only signed code is executed
- Root Of Trust is in Read-Only firmware
  - Hardware write protection
  - Reset vector must be in RO flash
  - Void warranty and open case to disable
- RO firmware verifies signed RW firmware
- RW firmware verifies signed kernel
- Reference implementation available
  - chromiumos/platform/vboot_reference.git

Verified Boot Kernel

- Root filesystem is a Read-Only image
    - Hash each block in the image
    - Block hashes are bundled and structured in a tree
    - Hash of the first block is stored with kernel and signed
- Root hash specified on kernel command line
- Subsequent read blocks are hashed
    - Checked against the tree
    - Stored in page cache

Verity

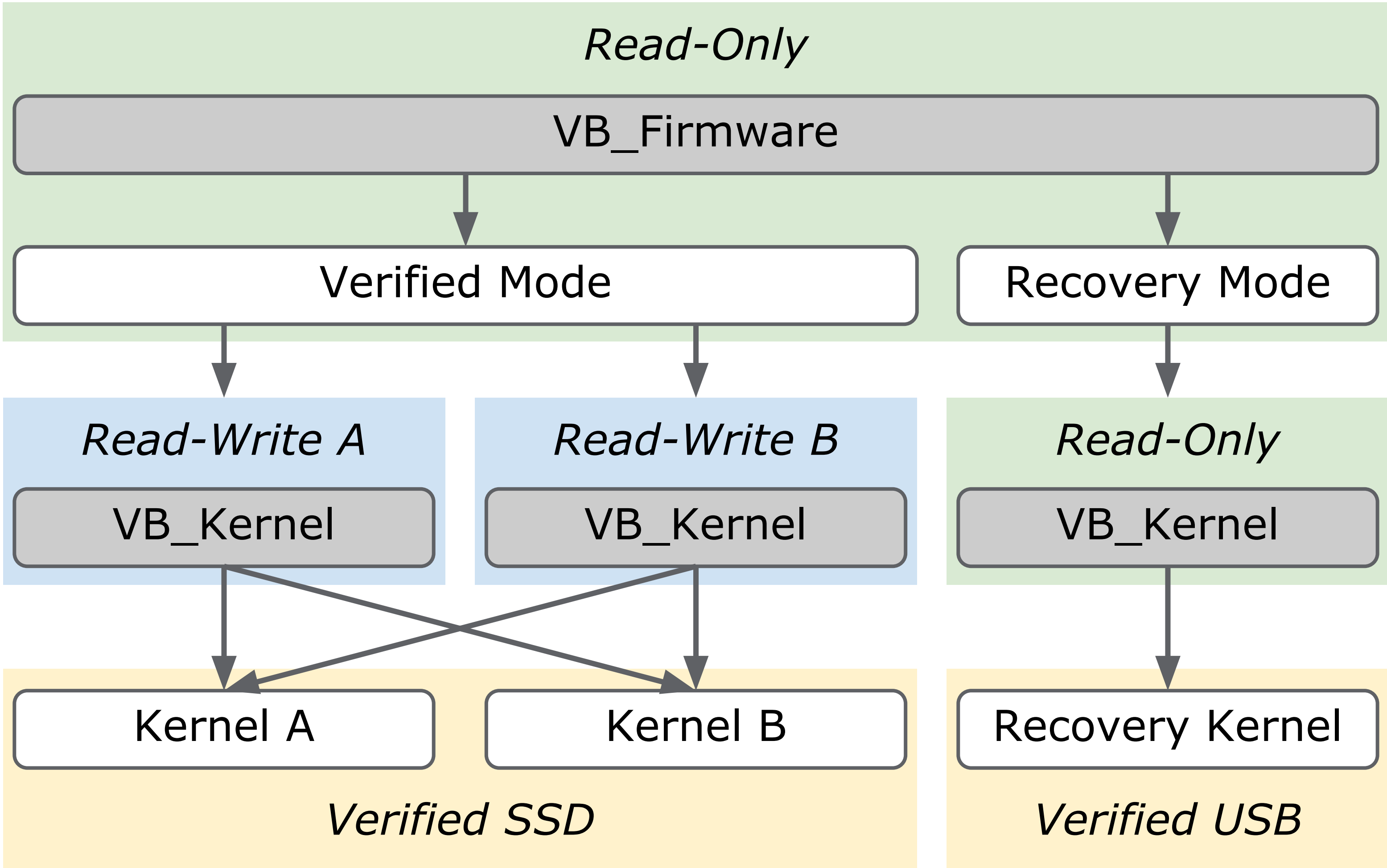



```
hash_block_size = 128
data_block_size = 4096
num_data_blocks = 32768
```

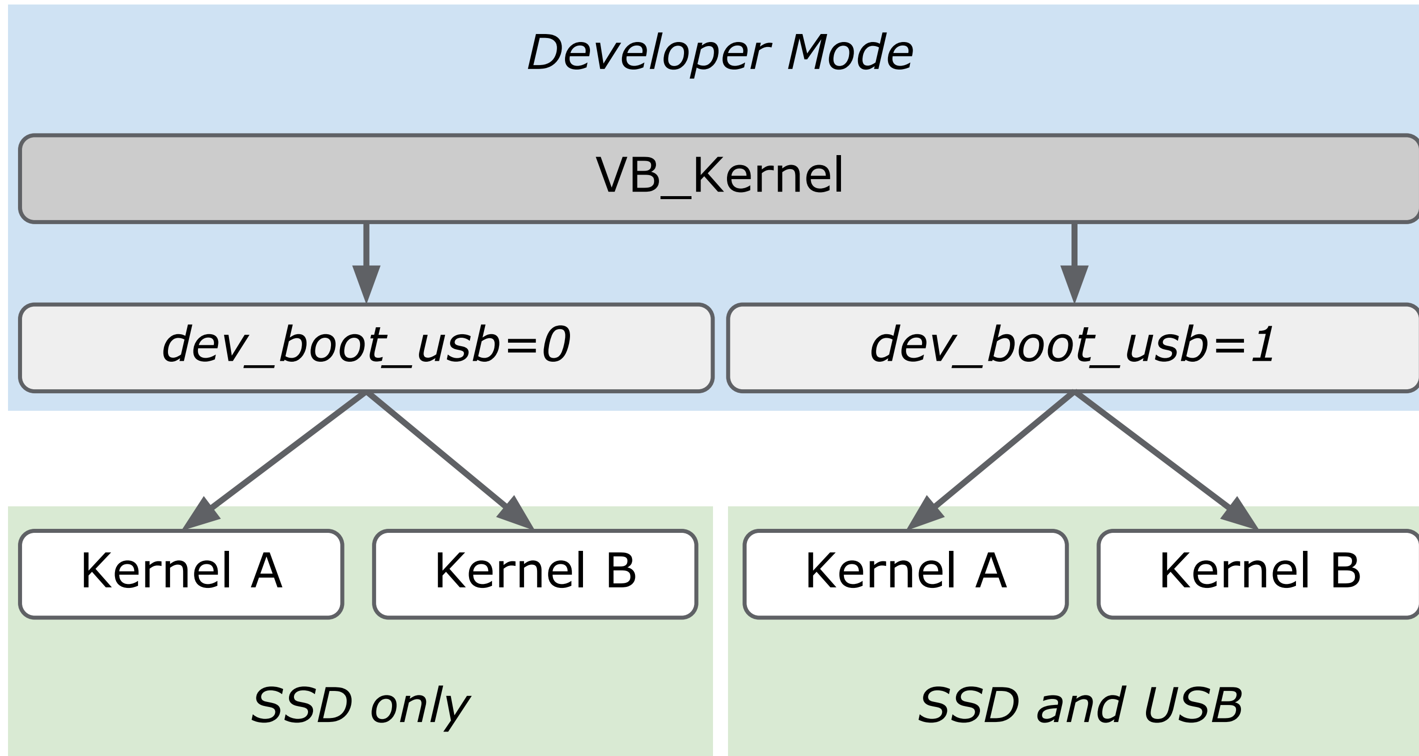
Verity

- Transparent block device
  - linux.git/Documentation/device-mapper/verity.txt
  - linux.git/drivers/md/dm-verity.c
- Experimental feature in Android 4.4
  - source.android.com/devices/tech/security/dm-verity.html

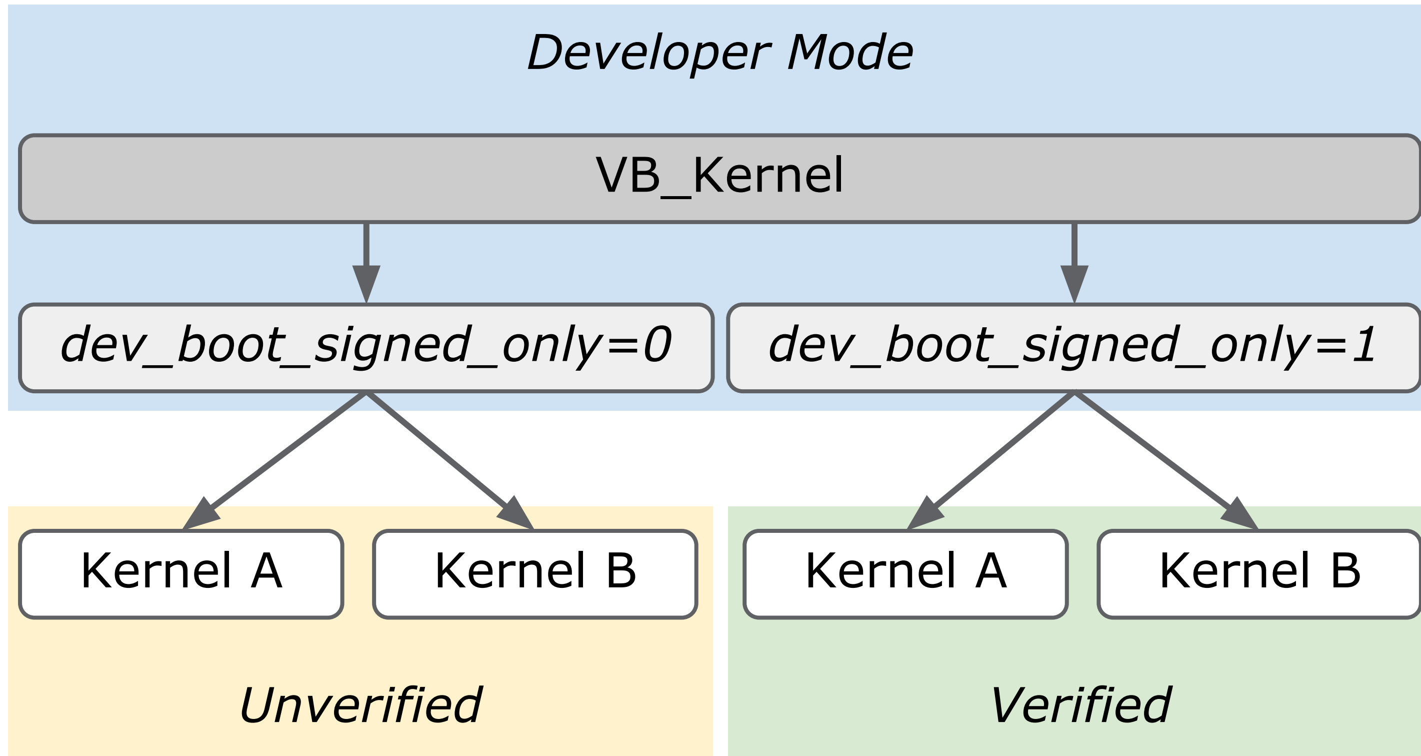
Verified Boot Modes



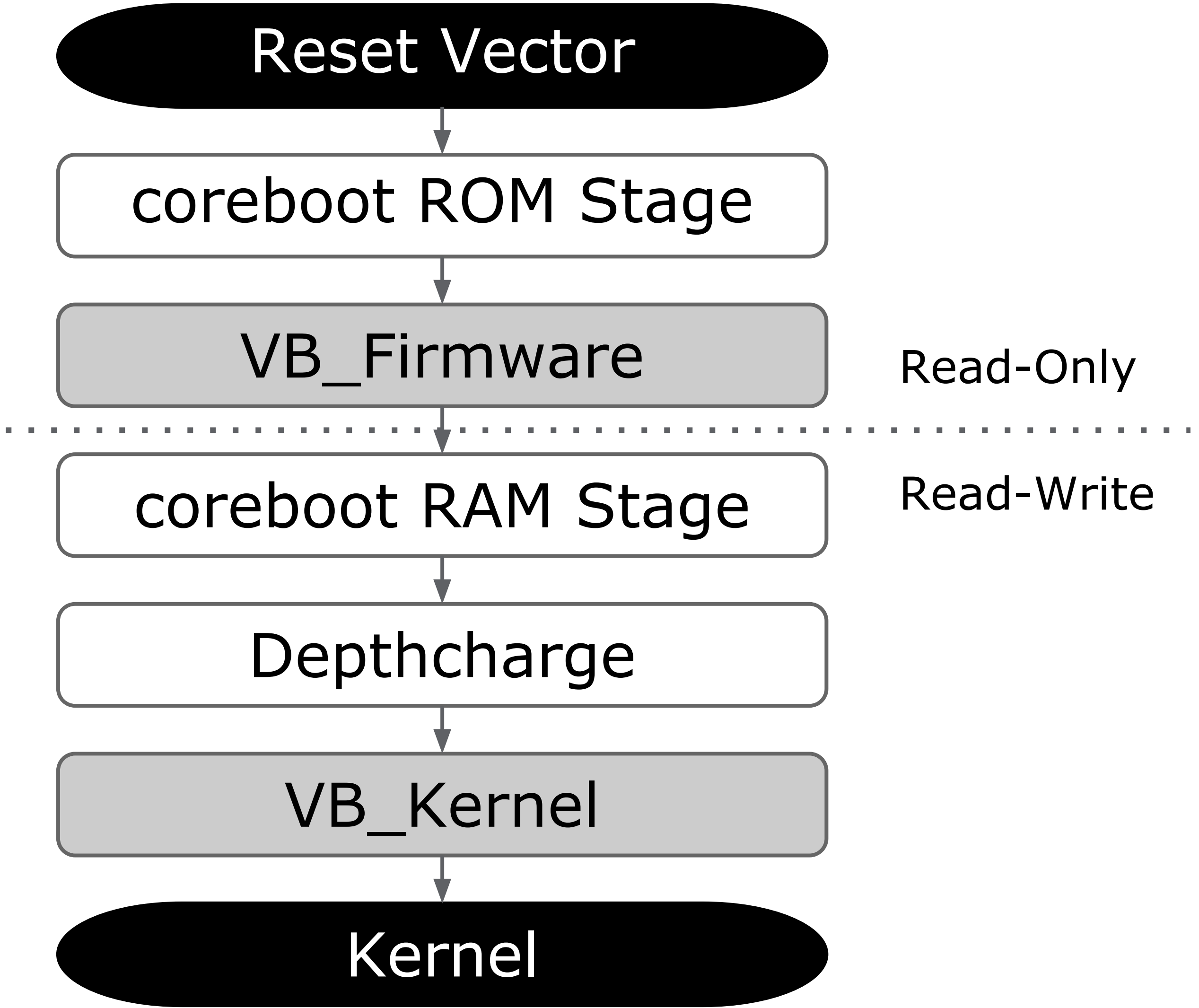
Developer Mode Boot



Developer Mode Verification



Verified Boot Integration



Embedded Controller

- Power sequencing
- Keyboard
- Fan Control
- Battery Charging
- Lid and Power Button control
- Device power control
- System LED behavior

Chrome EC

- Embedded Controllers are vital but closed
- Chrome EC is open source
  - chromiumos/platform/ec.git
- Chrome EC is designed for security
  - RO and RW regions
  - RW update is signed and handled by host firmware
  - EC Software Sync is part of Verified Boot
- Support for different ARM SOCs
  - Texas Instruments Stellaris Cortex-M4
  - ST Micro STM32 Cortex-M3
  - More in progress...

Firmware Support

- Flash Map
- Disk Image Layout
- TPM: Trusted Platform Module
- GBB: Google Binary Block
- VPD: Vital Product Data
- Firmware Update
- Tools

Flash Map

- Simple specification for layout of flash devices
- No assumptions about the underlying technology
  - Used in Legacy BIOS, UEFI, coreboot, EC
- Regions can overlap
- Checksum for static regions
- Reference implementation available
  - flashmap.googlecode.com

FMAP Structure

```
struct fmap_header {
        char        fmap_signature[8];      /* "__FMAP__" */
        uint8_t     fmap_ver_major;         /* Major version number of this structure */
        uint8_t     fmap_ver_minor;         /* Minor version number of this structure */
        uint64_t    fmap_base;              /* Physical address of the flash chip */
        uint32_t    fmap_size;              /* Size of the flash chip in bytes */
        char        fmap_name[32];          /* Descriptive name of this flash device */
        uint16_t    fmap_nareas;            /* Number of areas described by fmap_areas[] */

        struct fmap_area_header {
                uint32_t    area_offset;    /* Offset of this area in flash device */
                uint32_t    area_size;      /* Size of this area in bytes */
                char        area_name[32];  /* Descriptive name of this area */
                uint16_t    area_flags;     /* Flags for this area */
        } fmap_areas[0];
} __packed;

#define FMAP_AREA_STATIC        0x0001      /* Area contents will not change */
#define FMAP_AREA_COMPRESSED    0x0002      /* Area holds potentially compressed data */
#define FMAP_AREA_RO            0x0004      /* Area is considered read-only */
```

BASE	SIZE	SECTION	DESCRIPTION
0x000000	0x200000	SI_ALL	Descriptor + ME
0x200000	0x0f0000	RW_SECTION_A	Read-Write Firmware A
0x2f0000	0x0f0000	RW_SECTION_B	Read-Write Firmware B
0x3e0000	0x010000	RW_MRC_CACHE	Memory Training Cache
0x3f0000	0x004000	RW_ELOG	Event Log
0x3f4000	0x004000	RW_SHARED	Shared Data
0x3f8000	0x002000	RW_VPD	Read-Write VPD
0x400000	0x200000	RW_LEGACY	Legacy Firmware
0x600000	0x004000	RO_VPD	Read-Only VPD
0x610000	0x000800	FMAP	Flash Map
0x610800	0x000040	RO_FRID	RO Firmware ID
0x611000	0xef0000	GBB	Google Binary Block
0x700000	0x100000	BOOT_STUB	Read-Only Firmware

BOOT_STUB Section

- Reset vector is here
  - coreboot ROM stage and VB_Firmware
- Complete RO firmware for Recovery Mode
  - coreboot RAM stage
  - Depthcharge
  - VB_Kernel
- RO firmware will only boot from signed USB

Flash RW Section

RW_SECTION_A			
0x200000	0x010000	VBLOCK_A	Key Block
0x210000	0x0c0000	FW_MAIN_A	BIOS Image A
0x2d0000	0x01ffc0	EC_MAIN_A	EC Image A
0x2effc0	0x000040	RW_FWID_A	RW Firmware ID
RW_SECTION_B			
0x2f0000	0x010000	VBLOCK_B	Key Block
0x300000	0x0c0000	FW_MAIN_B	BIOS Image B
0x3c0000	0x01ffc0	EC_MAIN_B	EC Image B
0x3dffc0	0x000040	RW_FWID_B	RW Firmware ID

fmap.dts (RW_SECTION_A)

```
rw-a {
        label = "rw-section-a";
        reg = <0x00200000 0x000f0000>;
};
rw-a-vblock {
        label = "vblock-a";
        reg = <0x00200000 0x00010000>;
        type = "keyblock boot,ecrwhash,ramstage,refcode";
        keyblock = "firmware.keyblock";
        signprivate = "firmware_data_key.vbprivk";
        version = <1>;
        kernelkey = "kernel_subkey.vbpubk";
        preamble-flags = <0>;
};
rw-a-boot {
        label = "fw-main-a";
        reg = <0x00210000 0x000c0000>;
        type = "blob boot,ecrwhash,ramstage,refcode";
};
rw-a-ec-boot {
        label = "ec-main-a";
        reg = <0x002d0000 0x0001ffc0>;
        type = "blob ecbin";
};
rw-a-firmware-id {
        label = "rw-fwid-a";
        reg = <0x002effc0 0x00000040>;
        read-only;
        type = "blobstring fwid";
};
```

Disk Image

- GPT partition table
- Custom partition types
  - Chrome OS Kernel, Chrome OS rootfs
- Custom attributes for Verified Boot
  - Priority - Boot kernel with highest priority
  - Tries - Boot try count, decremented before attempt
  - Success - Written 30 seconds after boot
- cgpt tool for interacting with Chromium GPT
- Only one partition has Read-Write data
  - User data is encrypted

Disk Image Layout

NAME		TYPE	PRIORITY	TRIES	SUCCESS
STATE	RW	/mnt/stateful_partition			
KERN-A		vbutil_kernel image	1	0	1
ROOT-A	RO	rootdev			
KERN-B		vbutil_kernel image	0	0	0
ROOT-B	RO	rootdev			
KERN-C		vbutil_kernel_image	0	0	0
ROOT-C	RO	rootdev			
OEM	RO	/usr/share/oem			
RWFW		unused			
EFI-SYSTEM		chromiumos-x86			

Firmware TPM Usage

- Not used for cryptographic functions
- Preventing key rollback attacks
  - Firmware and Kernel key versions in TPM NVRAM
- Boot mode state
  - Platform Configuration Register 0
- TPM is locked by firmware
  - Except in Recovery Mode
  - Physical presence disabled by RW firmware
- TPM hardware
  - Drivers provided by coreboot/Depthcharge
  - Interface driven by Verified Boot

TPM: VB_Firmware

- TPM_Init
- TPM_Startup
- TSC_PhysicalPresence(PRESENT)
- TPM_NV_ReadValue
  - Check firmware key version
- TPM_NV_WriteValue
  - Set bGlobalLock
    - Except in Recovery Mode boot
- TPM_Extend(PCR0)
  - Get Boot Mode

TPM: VB_Kernel

- TPM_NV_ReadValue
  - Check kernel key version
- TPM_Extend(PCR0)
  - Get Boot Mode
  - Set Boot Mode
- TSC_PhysicalPresence(LOCK)

Google Binary Block (GBB)

- Binary storage interface
- Stored in Read-Only firmware region
- Chrome OS related features
  - Hardware Identification
  - Boot configuration flags
  - Firmware screen bitmaps
  - Root key
  - Recovery key

GBB: HWID v3

- Hardware identifier for unique bundle
  - Platform name, Build phase, RO Firmware Version
  - Catalogue of all HW and FW components
- Generated for each board during factory process
  - Verified against known HWID bundle
- Used to uniquely identify hardware variant
  - For recovery, updates, and metrics
- Firmware images contain default HWID
  - Firmware update scripts preserve original HWID
  - HWID bundles live in internal git repository

GBB: Firmware Bitmaps

- Firmware screens for Verified Boot
  - Recovery Mode help
  - Developer Mode warning
  - Transition to/from Developer mode
- Images with text overlay
  - Images are LZMA compressed bitmaps
  - Text is localized
  - Can switch between locales with arrow keys
- Needs to be available to RO firmware

GBB: Firmware Keys

- Public-key cryptographic signatures
  - Private keys known only to Google
  - Public keys in GBB are used for verification
  - Data is not encrypted, only hashed
- Root and Recovery Public Keys
  - RSA-8192 + SHA-512
  - Subsequent keys are smaller
- Each signing key is versioned
  - Verified Boot will reject lower versions

GBB: Boot Flags

- Flags that alter Chrome OS boot path
- Can override NV flags set with crossystem
- Used to enable alternate booting for the Factory
- Can be used by end-user to customize boot
  - After disabling write protect

GBB_FLAG_DEV_SCREEN_SHORT_DELAY	Reduce Developer screen delay to 2s
GBB_FLAG_FORCE_DEV_SWITCH_ON	Force enable Developer mode
GBB_FLAG_FORCE_DEV_BOOT_LEGACY	Force enable Legacy mode
GBB_FLAG_DEFAULT_DEV_BOOT_LEGACY	Default to Legacy mode boot
GBB_FLAG_DISABLE_EC_SOFTWARE_SYNC	Disable EC Read-Write firmware update
GBB_FLAG_DISABLE_FW_ROLLBACK_CHECK	Disable firmware rollback protection

Tool: gbb_utility

```
Utility to manage Google Binary Block (GBB)
Usage: gbb_utility [-g|-s|-c] [OPTIONS] bios_file [output_file]

GET MODE:
-g, --get   (default)   Get (read) from bios_file, with following options:
     --hwid             Report hardware id (default).
     --flags            Report header flags.
 -k, --rootkey=FILE     File name to export Root Key.
 -b, --bmpfv=FILE       File name to export Bitmap FV.
     --recoverykey=FILE File name to export Recovery Key.

SET MODE:
-s, --set               Set (write) to bios_file, with following options:
 -o, --output=FILE      New file name for ouptput.
 -i, --hwid=HWID        The new hardware id to be changed.
     --flags=FLAGS      The new (numeric) flags value.
 -k, --rootkey=FILE     File name of new Root Key.
 -b, --bmpfv=FILE       File name of new Bitmap FV.
     --recoverykey=FILE File name of new Recovery Key.

CREATE MODE:
-c, --create=prop1_size,prop2_size...
                        Create a GBB blob by given size list.
SAMPLE:
  gbb_utility -g bios.bin
  gbb_utility --set --hwid='New Model' -k key.bin bios.bin newbios.bin
  gbb_utility -c 0x100,0x1000,0x03DE80,0x1000 gbb.blob
```

Example: gbb_utility

```
Read current BIOS from flash into bios.bin
# flashrom -r bios.bin

Extract and display HWID from bios.bin
# gbb_utility --get --hwid bios.bin
hardware_id: PEPPY E6A-B3G-A3I

Extract and display GBB flags from bios.bin
# gbb_utility --get --flags bios.bin
flags: 0x00000000

Set GBB flags in bios.bin to 0x39 (factory default)
# gbb_utility --set --flags=0x39 bios.bin
 - flags changed from 0x00000000 to 0x00000039: success
successfully saved new image to: bios.bin

Write updated bios.bin back to flash
# flashrom -i GBB -w bios.bin
```

Tool: set_gbb_flags.sh

```
Changes ChromeOS Firmware GBB Flags value.

  Usage: set_gbb_flags.sh [option_flags] GBB_FLAGS_VALUE

  GBB_FLAG_DEV_SCREEN_SHORT_DELAY     0x00000001
  GBB_FLAG_LOAD_OPTION_ROMS           0x00000002
  GBB_FLAG_ENABLE_ALTERNATE_OS        0x00000004
  GBB_FLAG_FORCE_DEV_SWITCH_ON        0x00000008
  GBB_FLAG_FORCE_DEV_BOOT_USB         0x00000010
  GBB_FLAG_DISABLE_FW_ROLLBACK_CHECK  0x00000020
  GBB_FLAG_ENTER_TRIGGERS_TONORM      0x00000040
  GBB_FLAG_FORCE_DEV_BOOT_LEGACY      0x00000080
  GBB_FLAG_FAFT_KEY_OVERIDE           0x00000100
  GBB_FLAG_DISABLE_EC_SOFTWARE_SYNC   0x00000200
  GBB_FLAG_DEFAULT_DEV_BOOT_LEGACY    0x00000400

  To get a developer-friendly device, try 0x11 (short_delay + boot_usb).
  For factory-related tests (always DEV), try 0x39.

flags:
  -d,--[no]debug:  Provide debug messages (default: false)
  -f,--file:  Path to firmware image. Default to system firmware. (default: '')
  --[no]check_wp:  Check write protection states first. (default: true)
  -h,--[no]help:  show this help (default: false)
```

Vital Product Data (VPD)

- Region in Read-Only and Read-Write flash
- RO_VPD
  - Serial Number
  - Initial Locale
  - Initial Time Zone
  - Keyboard Layout
- RW_VPD
  - Activation Date
  - Registration Codes

Tool: vpd

```
Chrome OS VPD 2.0 utility --
 : 0e307b1 : Oct 11 2013 17:35:59 UTC

Usage: vpd [OPTION] ...
   OPTIONs include:
      -h               This help page and version.
      -f <filename>    The output file name.
      -E <address>     EPS base address (default:0x240000).
      -s <key=value>   To add/change a string value.
      -p <pad length>  Pad if length is shorter.
      -i <partition>   Specify VPD partition name in fmap.
      -l               List content in the file.
      --sh             Dump content for shell script.
      -O               Overwrite and re-format VPD partition.
      -g <key>         Print value string only.
      -d <key>         Delete a key.

   Notes:
      You can specify multiple -s and -d. However, vpd always
         applies -s first, then -d.
      -g and -l must be mutually exclusive.
```

Example: vpd

```
Display contents of read-only VPD
# vpd -i RO_VPD -l
“initial_locale”=“en-US”
“initial_timezone”=“America/Los_Angeles”
“keyboard_layout”=“xkb:us::eng”
“serial_number”=“2A763027093000099”

Display contents of read-write VPD
# vpd -i RW_VPD -l
“ActivateDate”=“2013-40”

Erase and reformat read-only VPD
# vpd -i RO_VPD -O

Add serial number to read-only VPD
# vpd -i RO_VPD -s serial_number=1234567890

Read back serial number value
# vpd -i RO_VPD -g serial_number
1234567890
```

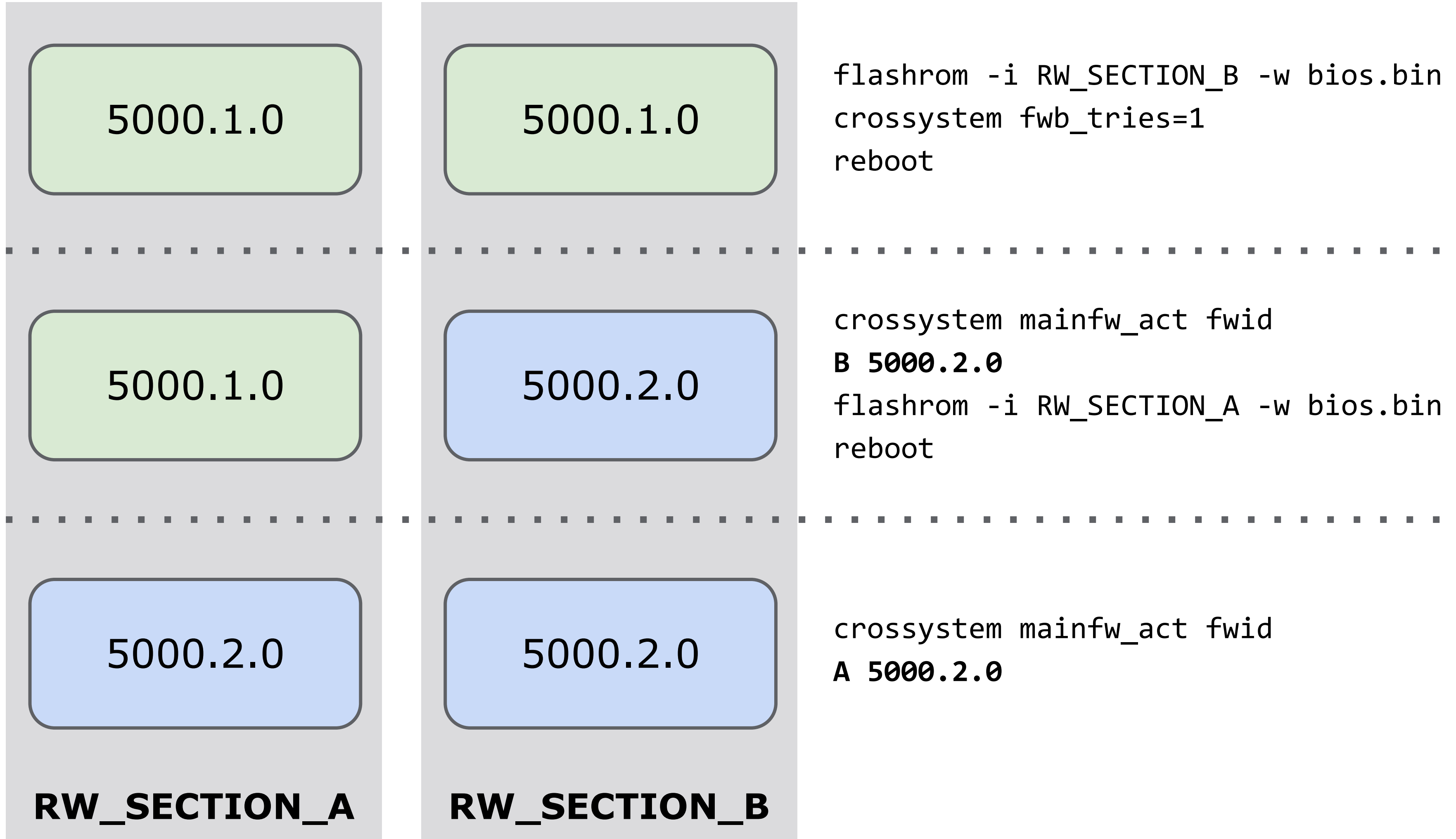
chromeos-firmwareupdate

- Self contained firmware update image
- Binaries and scripts embedded into shell archive
- Firmware images
  - bios.bin, ec.bin
- Static compiled utilities
  - flashrom, gbb_utility, vpd, mosys, crossystem
- Update scripts
  - updater4.sh, updater_custom.sh
- Helper scripts
  - shflags, common.sh, crosfw.sh

flashrom

- Open Source flash chip programmer
- Supports many chipsets
- FTDI (-p ft2232_spi:servo-v2)
- Dediprog (-p dediprog)
- Embedded Controllers (-p ec)
- FMAP integration (-i region)

Firmware Update



crossystem

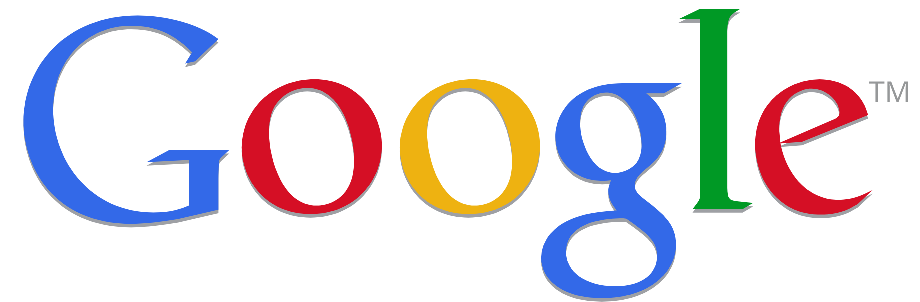
```
arch                   = x86                        # Platform architecture
clear_tpm_owner_request = 0                         # Clear TPM owner on next boot
clear_tpm_owner_done   = 0                          # Clear TPM owner done
cros_debug             = 0                          # OS should allow debug features
dbg_reset              = 0                          # Debug reset mode request (writable)
ddr_type               = unknown                    # Type of DDR RAM
debug_build            = 1                          # OS image built for debug features
dev_boot_usb           = 0                          # Enable developer mode boot from USB/SD (writable)
dev_boot_legacy        = 0                          # Enable developer mode boot Legacy OSes (writable)
dev_boot_signed_only   = 1                          # Boot only from official kernels (writable)
devsw_boot             = 0                          # Developer switch position at boot
devsw_cur              = 0                          # Developer switch current position
disable_dev_request    = 0                          # Disable virtual dev-mode on next boot
ecfw_act               = RW                         # Active EC firmware
fmap_base              = 0xffe10000                 # Main firmware flashmap physical address
fwb_tries              = 0                          # Try firmware B count (writable)
fwid                   = Google_Peppy.5216.78.0     # Active firmware ID
fwupdate_tries         = 0                          # Times to try OS firmware update (writable)
hwid                   = PEPPY E6A-B3G-A38 0128     # Hardware ID
kern_nv                = 0x00000000                 # Non-volatile field for kernel use
kernkey_vfy            = sig                        # Type of verification done on kernel key block
loc_idx                = 0                          # Localization index for firmware (writable)
mainfw_act             = A                          # Active main firmware
mainfw_type            = normal                     # Active main firmware type
nvram_cleared          = 1                          # Have NV settings been lost?  Write 0 to clear
oprom_needed           = 0                          # Should we load the VGA Option ROM at boot?
platform_family        = Haswell                    # Platform family type
```

crossystem

```
recovery_reason         = 0                            # Recovery mode reason for current boot
recovery_request        = 0                            # Recovery mode request (writable)
recovery_subcode        = 0                            # Recovery reason subcode (writable)
recoverysw_boot         = 0                            # Recovery switch position at boot
recoverysw_cur          = (error)                      # Recovery switch current position
recoverysw_ec_boot      = 0                            # Recovery switch position at EC boot
ro_fwid                 = Google_Peppy.5216.61.0       # Read-only firmware ID
savedmem_base           = 0x00f00000                   # RAM debug data area physical address
savedmem_size           = 1048576                      # RAM debug data area size in bytes
sw_wpsw_boot            = 0                            # Firmware write protect SW setting enabled at boot
tpm_fwver               = 0x00010001                   # Firmware version stored in TPM
tpm_kernver             = 0x00010001                   # Kernel version stored in TPM
tried_fwb               = 0                            # Tried firmware B before A this boot
vdat_flags              = 0x00004c42                   # Flags from VbSharedData
vdat_timers             = LFS=183498608,280001888 LF=280216304,415831424 LK=0,400900 # Timer values
wpsw_boot               = 1                            # Firmware write protect HW switch position at boot
wpsw_cur                = 1                            # Firmware write protect HW switch current position
```

mosys

- Firmware and Hardware inspection utility
- System memory information
  - *mosys memory spd*
- Event Log decode
  - *mosys eventlog list*
- Embedded Controller information
  - *mosys ec info*



coreboot

Overview and Features

Coding Style

- www.coreboot.org/Coding_Style
- Very close to Linux Kernel style
- 80 columns
- 8 character tabs

Configuration

- Modified Kconfig and Kbuild
- CONFIG_x options always defined
- Disabled options are defined as zero
- Must use #if and not #ifdef

```
#if CONFIG_x
    ...
#endif

if (CONFIG_x) {
    ...
}
```

Common Defines

#define	Description
__PRE_RAM__	ROM Stage
__SMM__	System Management Mode
__ACPI__	For ASL compiler
__ASSEMBLER__	Assembly code
__ROMCC__	Boot Block C compiler

Source Tree Layout

`arch/`	Architecture specific common code
`console/`	Console drivers
`cpu/`	CPU support
`device/`	Device enumeration and initialization
`drivers/`	Device drivers
`ec/`	Embedded Controller support
`include/`	Headers
`lib/`	Common code
`mainboard/`	Mainboard support
`northbridge/`	Northbridge support
`soc/`	SOC support
`southbridge/`	Southbridge support
`superio/`	SuperIO support
`vendorcode/`	Vendor specific code

Mainboard

- mainboard/google/peppy
  - cpu/intel/haswell
  - northbridge/intel/haswell
  - southbridge/intel/lynxpoint
  - ec/google/chromeec
- mainboard/intel/bayleybay
  - soc/intel/baytrail
  - ec/google/chromeec
- mainboard/google/snow
  - soc/samsung/exynos5250
  - ec/google/chromeec

Console Output

include/console/loglevel.h:

```
#define BIOS_EMERG          0   /* system is unusable                  */
#define BIOS_ALERT          1   /* action must be taken immediately    */
#define BIOS_CRIT           2   /* critical conditions                 */
#define BIOS_ERR            3   /* error conditions                    */
#define BIOS_WARNING        4   /* warning conditions                  */
#define BIOS_NOTICE         5   /* normal but significant condition    */
#define BIOS_INFO           6   /* informational                       */
#define BIOS_DEBUG          7   /* debug-level messages                */
#define BIOS_SPEW           8   /* way too many details                */
```

.config:

```
CONFIG_MAXIMUM_CONSOLE_LOGLEVEL_8=y
CONFIG_DEFAULT_CONSOLE_LOGLEVEL_6=y
```

Example:

```
#include <console/console.h>
printk(BIOS_INFO, “Starting coreboot…\n”);
```

Register Script

- Data driven process for initializing devices
  - `#include <reg_script.h>`
  - `struct reg_script init_script[] = {...}`
- Supports PCI, IO, MMIO, MSR, IOSF
  - `REG_PCI_WRITE32(PCI_COMMAND, PCI_COMMAND_MASTER)`
  - `REG_MMIO_RMW32(BASE + 0x200, ~0xf, 0x1)`
- Can maintain device context
  - `reg_script_run_on_dev(dev, init_script)`
  - `REG_RES_WRITE32(PCI_BASE_ADDRESS_0, 0x200, 0x1)`
- Register Script tables can be chained
  - `REG_SCRIPT_NEXT(next_script)`

reg_script Example

```
/* Warm Reset a USB3 port */
static void xhci_reset_port_usb3(struct device *dev, int port)
{
        struct reg_script reset_port_usb3_script[] = {
                /* Issue Warm Port Rest to the port */
                REG_RES_OR32(PCI_BASE_ADDRESS_0, XHCI_USB3_PORTSC(port),
                             XHCI_USB3_PORTSC_WPR),
                /* Wait up to 100ms for it to complete */
                REG_RES_POLL32(PCI_BASE_ADDRESS_0, XHCI_USB3_PORTSC(port),
                               XHCI_USB3_PORTSC_WRC, XHCI_USB3_PORTSC_WRC,
                               XHCI_RESET_TIMEOUT),
                /* Clear change status bits, do not set PED */
                REG_RES_RMW32(PCI_BASE_ADDRESS_0, XHCI_USB3_PORTSC(port),
                              ~XHCI_USB3_PORTSC_PED, XHCI_USB3_PORTSC_CHST),
                REG_SCRIPT_END
        };
        reg_script_run_on_dev(dev, reset_port_usb3_script);
}
```

devicetree.cb

- Build and platform configuration settings
- Mainboard specific
- Describes device layout of the system
- Not related to Open Firmware Device Tree
- Parser in util/sconfig/
  - Creates chip and device structures from devicetree.cb
  - Translate path to name
    - southbridge/intel/lynxpoint -> southbridge_intel_lynxpoint

devicetree.cb Example

mainboard/google/peppy/devicetree.cb:

```
chip northbridge/intel/haswell
     device cpu_cluster 0 on
          chip cpu/intel/haswell end
     end
     device domain 0 on
          device pci 00.0 on end # Host Bridge
          device pci 02.0 on end # GPU
          ...
          chip southbridge/intel/lynxpoint
               device pci 1f.0 on # LPC
                    chip ec/google/chromeec end
               end
               ...
               register “sata_port_map” = “0x1”
               device pci 1f.2 on end # SATA
          end
     end
end
```

Device Tree Example: Peppy

southbridge/intel/lynxpoint/chip.h:

```
struct southbridge_intel_lynxpoint_config {
        uint8_t sata_port_map;
};
```

southbridge/intel/lynxpoint/pch.c:

```
static void pch_enable(struct device *dev) {}
struct chip_operations southbridge_intel_lynxpoint_ops = {
        CHIP_NAME("Intel LynxPoint Southbridge"),
        .enable_dev = pch_enable,
};
```

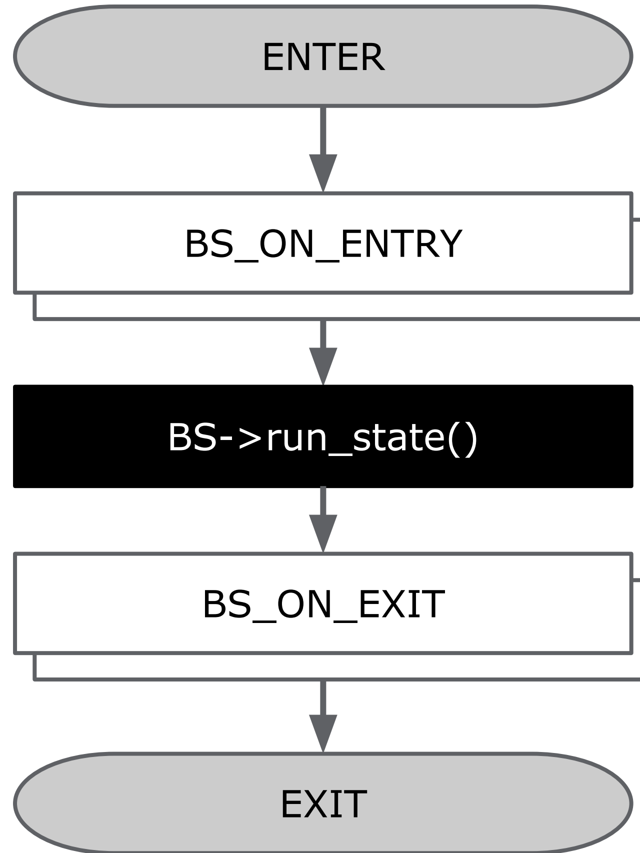
southbridge/intel/lynxpoint/sata.c:

```
static void pch_sata_init(struct device *dev)
{
        struct southbridge_intel_lynxpoint_config *config = dev->chip_info;
        pci_write_config16(dev, PCH_SATA_PORT_MAP, config->sata_port_map);
}
static struct device_operations pch_sata_ops = {
        .read_resources    = pci_dev_read_resources,
        .set_resources     = pci_dev_set_resources,
        .enable_resources  = pci_dev_enable_resources,
        .init              = pch_sata_init,
};
static const struct pci_driver pch_sata __pci_driver = {
        .ops               = &pch_sata_ops,
        .vendor            = PCI_VENDOR_ID_INTEL,
        .device            = PCI_DEVICE_ID_INTEL_LYNXPOINT_SATA_AHCI,
};
```

Boot State Machine

- Boot process is separated into discrete states
  - include/bootstate.h
  - lib/hardwaremain.c
- Callbacks can be made on state entry or exit
  - BS_ON_ENTRY
  - BS_ON_EXIT

Boot State Callbacks



Boot States

STATE	DESCRIPTION	ACTION
BS_PRE_DEVICE	Before any device tree actions take place	
BS_DEV_INIT_CHIPS	Initialize all chips in device tree	`chip->ops->init()`

BS_DEV_ENUMERATE	Device tree probing	`chip->ops->enable_dev()`
		`dev->ops->enable()`
		`dev->ops->scan_bus()`

BS_DEV_RESOURCES	Device resource allocation and assignment	`dev->ops->read_resources()` `dev->ops->set_resources()`
BS_DEV_ENABLE	Device enable or disable	`dev->ops->enable_resources()` `dev->ops->pci->set_subsystem()`
BS_DEV_INIT	Device initialization	`dev->ops->init()`
BS_POST_DEVICE	All device tree actions performed	
BS_OS_RESUME_CHECK	Check for OS resume	`acpi_find_wakeup_vector()`
BS_OS_RESUME	Resume to OS	`acpi_resume()`
BS_WRITE_TABLES	Write coreboot tables	`write_tables()`
BS_PAYLOAD_LOAD	Load a payload into memory	`cbfs_load_payload()`
BS_PAYLOAD_BOOT	Execute the loaded payload	`selfboot()`

Boot State Callback Example

```
#include <bootstate.h>
#include <console/console.h>

/* Early Magic Initialization Before Device Setup */
static void magic_init_early(void *unused)
{
        printk(BIOS_INFO, “Early Magic Init\n”);
}

/* Late Magic Initialization After Device Setup */
static void magic_init_late(void *unused)
{
        printk(BIOS_INFO, “Late Magic Init!\n”);
}

BOOT_STATE_INIT_ENTRIES(magic_init) = {
        BOOT_STATE_INIT_ENTRY(BS_PRE_DEVICE, BS_ON_ENTRY,
                              magic_init_early, NULL),
        BOOT_STATE_INIT_ENTRY(BS_POST_DEVICE, BS_ON_EXIT,
                              magic_init_late, NULL),
};
```

CBFS

- Simple flash-based file system
- Offset to header at 4 bytes from end of ROM
  - 0xFFFFFFFC -> 0xFFFFB000-0xFFFFFFF0
- Isolate firmware components
  - Boot stages
  - Payloads
  - Option ROMs
  - On-board memory SPD
  - Microcode updates
  - Reference code binaries

CBFS Usage

```
/* Load boot stage from CBFS */
void *cbfs_load_stage(struct cbfs_media *media, const char *name);

/* Load payload from CBFS */
void *cbfs_load_payload(struct cbfs_media *media, const char *name);

/* Load Option ROM from CBFS */
void *cbfs_load_optionrom(struct cbfs_media *media,
                           uint16_t vendor, uint16_t device, void *dest);

/* Return pointer to file content inside CBFS */
void *cbfs_get_file_content(struct cbfs_media *media, const char *name, int type);

/*
 * EXAMPLE: Locate Memory Reference Code binary in CBFS
 */
entry = cbfs_get_file_content(CBFS_DEFAULT_MEDIA, “mrc.bin”, 0xab);
```

cbfstool

```
# Print contents of CBFS
cbfstool bios.bin print

# Add RAM stage to CBFS
cbfstool bios.bin add-stage -f coreboot_ram.elf -n fallback/coreboot_ram -c lzma

# Add payload to CBFS
cbfstool bios.bin add-payload -f seabios.elf -n fallback/payload -c lzma

# Add reference code binary to CBFS
cbfstool bios.bin add -f mrc.bin -n mrc.bin -t 0xab -b 0xfffa0000

# Add reference code binary to CBFS from Makefile
cbfs-files-$(CONFIG_HAVE_MRC) += mrc.bin
mrc.bin-file := $(CONFIG_MRC_FILE)
mrc.bin-type := 0xab
mrc.bin-position := 0xfffa0000
```

CBMEM

- Generic runtime storage interface
  - Tables for ACPI, SMBIOS, MP, etc
  - S3 Resume scratch space
  - Event Log
  - Memory Console
  - Timestamps
- CBMEM region is reserved in e820
- Can function like EFI HOB
  - Entries added in ROM stage available in RAM stage

CBMEM Usage

```
#define CBMEM_ID_ACPI           0x41435049
#define CBMEM_ID_CBTABLE        0x43425442
#define CBMEM_ID_RESUME         0x5245534d
#define CBMEM_ID_TIMESTAMP      0x54494d45
#define CBMEM_ID_MRCDATA        0x4d524344
#define CBMEM_ID_CONSOLE        0x434f4e53
#define CBMEM_ID_ELOG           0x454c4f47

/* Find existing or initialize new cbmem area. */
void cbmem_initialize(void);

/* Add a cbmem entry of a given size and id. */
void *cbmem_entry_add(uint32_t id, uint64_t size);

/* Find a cbmem entry of a given id. */
void *cbmem_find(uint32_t id);
```

CBMEM Console

- Serial is essential for development and debug
  - Not always available, especially in mobile devices
- Console output saved to memory buffer
- In ROM stage some CAR space is used for console
- In early RAM stage a static buffer is used
- In RAM stage CBMEM is reinitialized
  - Concatenate ROM stage and early RAM stage buffers

CBMEM Timestamp Table

- Chrome OS focus on boot time
- Timestamp table stored in CBMEM
  - Each entry includes ID and timestamp
  - Architecture specific timestamp source
- Timestamps collected at specific points
  - First possible collection point
  - Boot State Transitions
  - Before and after Verified Boot
  - Last point before boot or resume

CBMEM at Runtime

- Kernel driver for Google Memory Console
  - [linux.git/drivers/firmware/google/memconsole.c]
  - Export CBMEM console log to sysfs
    - /sys/firmware/log
- Userspace utility at util/cbmem/
  - Print memory console
  - Decode timestamps
  - Code coverage information

Event Log

- Persistent log of system events
- Based on SMBIOS System Event Log
- Runtime logging
  - x86 runtime logging relies on SMI
  - ARM runtime logging accesses flash directly
- Userland can find and parse the log
  - chromiumos/platform/mosys.git/lib/eventlog/eventlog.c

Google SMI Driver

- Kernel/Firmware SMI interface
  - linux.git/drivers/firmware/google/gsmi.c
- Allow kernel events to be stored in event log
- Hook into kernel notifier chains
  - panic
  - thermal
  - reboot
  - die

ELOG

include/elog.h:

```
int elog_add_event_raw(uint8_t event_type, void *data, uint8_t data_size);
int elog_add_event(uint8_t event_type);
int elog_add_event_byte(uint8_t event_type, uint8_t data);
int elog_add_event_wake(uint8_t source, uint32_t instance);
```

southbridge/intel/lynxpoint/elog.c:

```
/* Read PM1_STS register value from PMBASE */
u16 pm1_sts = inw(pmbase + PM1_STS);

/* Check for Power Button wake event */
if (pm1_sts & PWRBTN_STS)
        elog_add_event_wake(ELOG_WAKE_SOURCE_PWRBTN, 0);

/* Check for RTC wake event */
if (pm1_sts & RTC_STS)
        elog_add_event_wake(ELOG_WAKE_SOURCE_RTC, 0);

/* Check for ACPI wake (from S3/S4/S5) */
if (pm1_sts & WAK_STS)
        elog_add_event_byte(ELOG_TYPE_ACPI_WAKE, acpi_slp_type);
```

Event Log Example

```
# mosys eventlog list
12 | 2013-01-15 10:47:43 | ACPI Wake    | S5
13 | 2013-01-15 10:47:43 | EC Event     | Lid Open
14 | 2013-01-15 10:47:43 | System boot  | 142
15 | 2013-01-15 11:51:42 | ACPI Enter   | S3
16 | 2013-01-15 21:05:37 | ACPI Wake    | S3
17 | 2013-01-15 21:05:37 | Wake Source  | GPIO | 11
18 | 2013-01-15 21:05:38 | Kernel Event | Oops
19 | 2013-01-15 21:05:38 | Kernel Event | Panic
20 | 2013-01-15 21:05:39 | System boot  | 143
```

```
10:47 - Power on because lid was opened
11:51 - System is suspended
21:05 - Wake from suspend due to GPIO 11 (Touchpad)
21:05 - Kernel oops+panic on resume
```

firmware@chromium.org