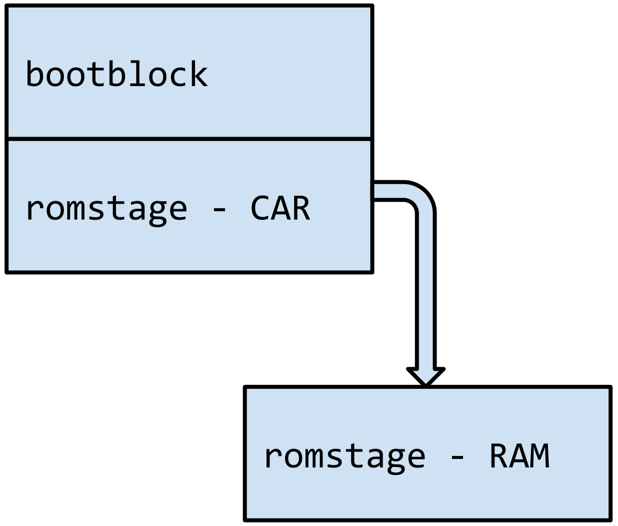
Google™

coreboot porting for x86

Contents

```
src/mainboard/google/rambi/
src/soc/intel/baytrail/
src/cpu/x86/smm/
src/cpu/x86/mp_init.c
```

Early Boot Flow



Bootblock

- enter 32-bit protected mode
- locate romstage (XIP) in cbfs
- uses romcc for small bits of C

Romstage

- two parts: cache-as-ram and romstage with ram
- typical x86 doesn’t have SRAM to use. Need to put cache into special cache-as-ram mode to run C code.

```
src/soc/intel/baytrail/bootblock/bootblock.c
src/soc/intel/baytrail/romstage/romstage.c
```

Romstage - cache-as-ram

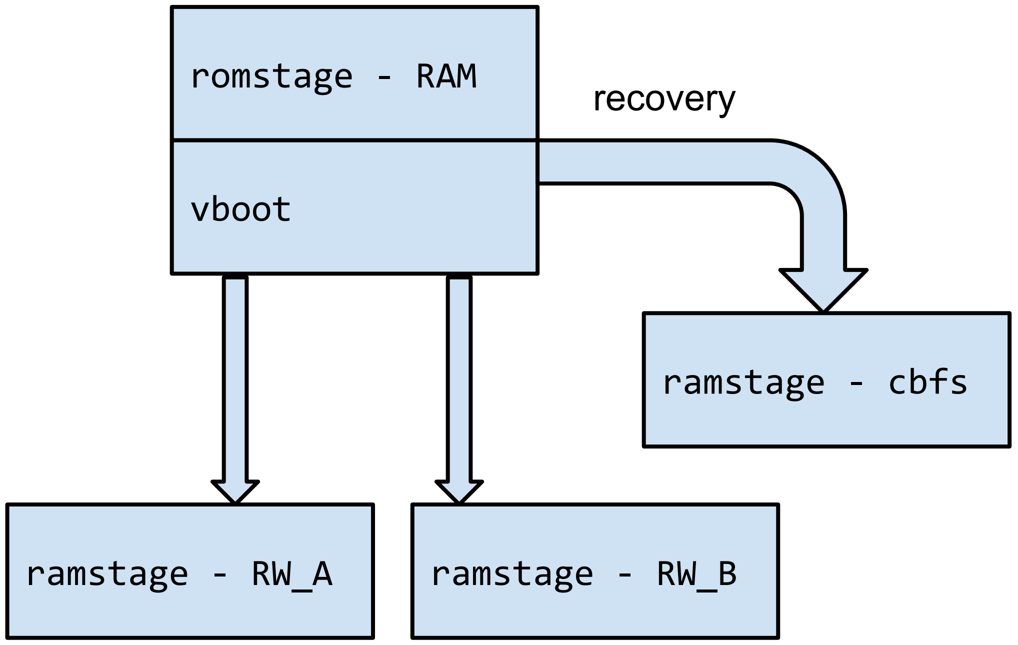
- Configure regions of address space to be used like ram - cache acts like backing store.
- Configure chipset properties -- static BARS
- Configure DRAM using MRC (memory reference code).
- Global writable variables should not be used (though possible with CAR_GLOBAL).
- Initialize cbem area (CAR migration takes place here).

```
cache_as_ram [1] ->
    romstage_main() [2] ->
        mainboard_romstage_entry() [3] ->
            romstage_common() [2] ->
                raminit() [4]
```

```
[1] src/soc/intel/baytrail/romstage/cache_as_ram.inc
[2] src/soc/intel/baytrail/romstage/romstage.c
[3] src/mainboard/google/rambi/romstage.c
[4] src/soc/intel/baytrail/romstage/raminit.c
```

Romstage - RAM

- Provide initial MTRR settings for romstage with RAM (setup_stack_and_mttrs() [2])
- Jump back to cache_as_ram.inc. Tear down CAR. Call romstage_after_car() [2]
- Decide where to run ramstage from [3] and [4].



```
[1] src/soc/intel/baytrail/romstage/cache_as_ram.inc
[2] src/soc/intel/baytrail/romstage/romstage.c
[3] src/vendorcode/google/chromeos/vboot_loader.c
[4] src/vendorcode/google/chromeos/vboot_wrapper.c
```

Ramstage

- Entry point: _start [1] -> main() [2]
- Devicetree.cb - list of static devices [3]
- Device initialization - resource probing and allocation
- Multiprocessor bringup
- SMM initialization
- MTRR configuration
- ACPI table generation
- Memory map layout (e820)
- Load payload (bootloader) and jump to it

```
[1] src/arch/x86/lib/c_start.S
[2] src/lib/hardwaremain.c
[3] src/mainboard/google/rambi/devicetree.cb
```

Hardware Initialization

- Devicetree ordering matters for each call:

```
BS_DEV_INIT_CHIPS
 chip->ops->init()
BS_DEV_ENUMERATE
 chip->ops->enable(each_dev)
 dev->ops->enable(dev)
 dev->ops->scan_bus(dev)
BS_DEV_RESOURCES
 dev->ops->read_resources(dev)
 dev->ops->set_resources(dev)
BS_DEV_ENABLE
 dev->ops->enable_resources(dev)
 dev->ops->ops_pci->set_subsystem(dev, vendorid, devid)
BS_DEV_INIT
 dev->ops->init(dev)
```

Hardware Initialization

```
chip soc/intel/baytrail [1][3]

  device cpu_cluster 0 on

    device lapic 0 on end

  end

  device domain 0 on

    device pci 00.0 on  end # SoC router

    ...

    device pci 1f.3 off end # SMBus

  end

end
```

BS_DEV_INIT_CHIPS [2]

- `soc_init()` [4] -> `baytrail_init_pre_device()` [5]

BS_DEV_ENUMERATE [2]

- `enable_dev()` [4] - DEVICE_PATH_CPU_CLUSTER and DEVICE_PATH_DOMAIN

Per-device PATH-type driver binding

- cpuid
- PCI didvid

```
static struct cpu_device_id cpu_table[] = {
   { X86_VENDOR_INTEL, 0x30673 },
   { X86_VENDOR_INTEL, 0x30678 },
   { 0, 0 },
};

static const unsigned short pci_device_ids[] = {
   SIO_DMA1_DEVID, I2C1_DEVID, 0,
};
```

[1] src/mainboard/google/rambi/devicetree.cb
[2] src/lib/hardwaremain.c
[3] src/soc/intel/baytrail/chip.h
[4] src/soc/intel/baytrail/chip.c
[5] src/soc/intel/baytrail/ramstage.c

Resources

- 2 Main types: `IORESOURCE_IO` (port I/O) and `IORESOURCE_MEM` `(physical address space)`
- Fixed or Allocated
- All `IORESOURCE_FIXED` resources need to be added described during `dev->ops->read_resources(dev)` [1] - sets constraints for allocating non-fixed resources
- Example Attributes [2]: `IORESOURCE_RESERVE`, `IORESOURCE_PREFETCH`, `IORESOURCE_CACHEABLE`
- Helper Functions [3]: `ram_resource()`, `reserved_ram_resource()`, `mmio_resource()`

[1] `src/soc/intel/baytrail/northcluster.c`
[2] `src/include/device/resource.h`
[3] `src/include/device/device.h`

Multiprocessor Initialization

- Typically happens after after resource assignment.
- Since physical address space set final MTRR setup takes place.
- Parallel CPU bringup support in library [1] using `mp_flight_records` [2]:

```
static struct mp_flight_record mp_steps[] = {
    MP_FR_BLOCK_APS(smm_relocate, NULL, smm_relocate, NULL),
    MP_FR_BLOCK_APS(mp_initialize_cpu, NULL, mp_initialize_cpu, NULL),
    /* Wait for APs to finish initialization before proceeding. */
    MP_FR_BLOCK_APS(NULL, NULL, enable_smis, NULL),
};
```

- SMM relocation and initialization can be done in parallel as well.

[1] src/cpu/x86/mp_init.c
[2] src/soc/intel/baytrail/cpu.c

Device Specialization

- Some settings mainboard specific
- Chipset can export per-board settings using devicetree
- Not automatic: need chip.h structure and C code to honor setting

```
chip soc/intel/baytrail
  # SD Card controller
  register "sdcard_cap_low" = "0x036864b2"
  register "sdcard_cap_high" = "0x0"

  # Enable devices in ACPI mode
  register "scc_acpi_mode" = "1"
  register "lpss_acpi_mode" = "1"
end
```

```
struct soc_intel_baytrail_config {
  uint32_t sdcard_cap_low;
  uint32_t sdcard_cap_high;

  int lpss_acpi_mode;
  int scc_acpi_mode;
  int lpe_acpi_mode;
};
```

[1] src/mainboard/google/rambi/devicetree.cb
[2] src/soc/intel/baytrail/chip.h

Device Specialization

```
static void sd_init(device_t dev)
{
        struct soc_intel_baytrail_config *config = dev->chip_info;
        if (config == NULL)
                return;
        if (config->sdcard_cap_low != 0 || config->sdcard_cap_high != 0) {
                printk(BIOS_DEBUG, "Overriding SD Card controller caps.\n");
                pci_write_config32(dev, CAP_OVERRIDE_LOW, config->sdcard_cap_low);
                pci_write_config32(dev, CAP_OVERRIDE_HIGH, config->sdcard_cap_high |
                                   USE_CAP_OVERRIDES);
        }
        if (config->scc_acpi_mode)
                scc_enable_acpi_mode(dev, SCC_SD_CTL, SCC_NVS_SD);
}
```

ACPI

- Static AML generation for ASL files
    - `acpi/` directory under each chipset component [1]
    - `acpi/` directory under each mainboard [2]
- Dynamic AML generation [3]
    - example: src/soc/intel/baytrail/acpi.c

[1] `src/soc/intel/baytrail/api/`
[2] `src/mainboard/google/rambi/acpi/`
[3] `src/arch/x86/boot/acpigen.c`

Questions/Comments?

?

firmware@chromium.org