(Ab)using Google's Chromium EC for your own projects

Building Franken Chrome Devices

Moritz Fischer - Senior Software Engineer
National Instruments

$whoami

Moritz Fischer

- Embedded Software Engineer @ National Instruments
- Work on the USRP
- Very small team, focus on open-source
- Try to work upstream first (U-Boot, Kernel, ...)

Also find slides at http://mfischer.github.io/fosdem17-slides

Brief Intro to Chromebooks

Just another laptop?

Chromebook

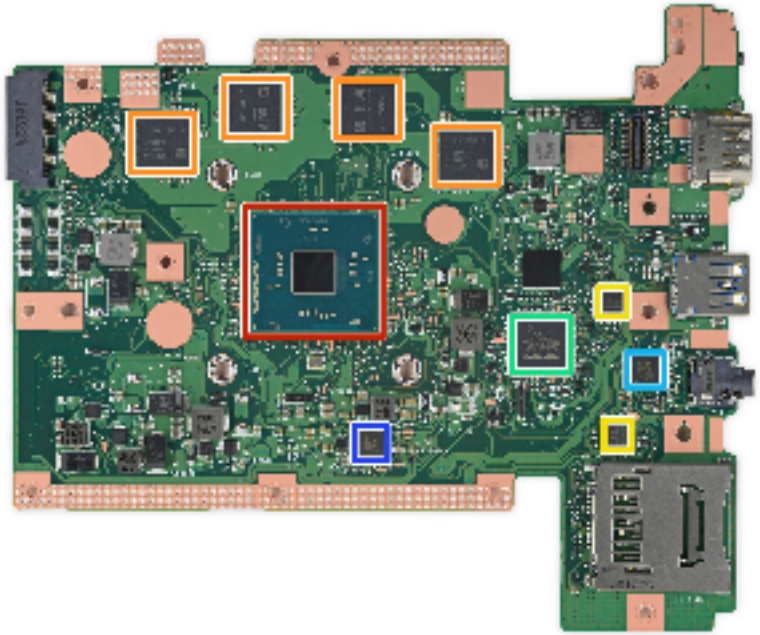
What's so special?

- x86_64 (Intel), ARMv7 (exynos 5, rk3288c, ...), ARMv8 (rk3399)
- Linux Kernel (old, but maintained)
- Userland derived from Gentoo
- Made to run Google Chrome & Google Apps
- Very limited local storage
- Strong focus on security, verified boot

source: lenovo.com

Chromebook Embedded Controller

What's that?

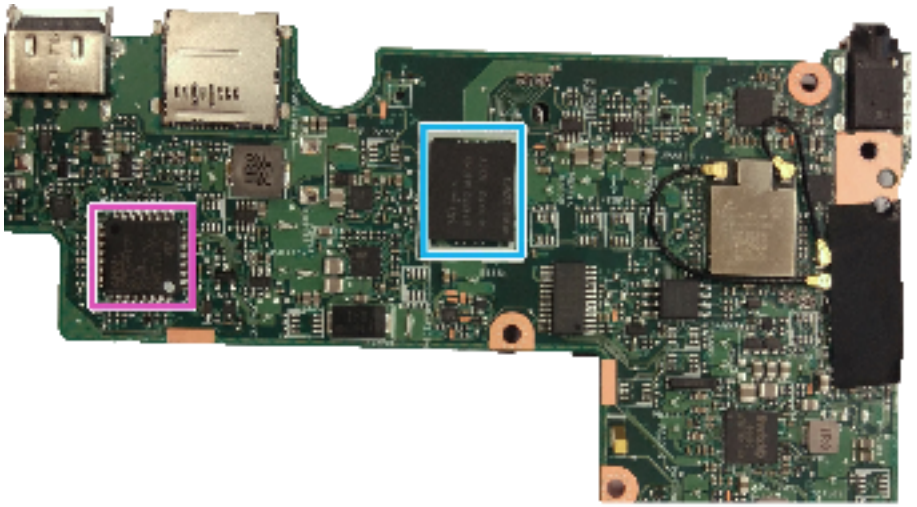


- PCB here: Asus C202 (terra/strago)
- AP: (red) Intel Celeron CPU (Braswell)
- EC MCU (green) **SMSC MEC1322-LZY**
- Hooked up via LPC
- We can buy that MCU!

source: ifixit.com

Chromebook Embedded Controller

Another one!



- PCB here: Asus C100P (veyron_minnie/veyron_pink)
- AP: (bottom right) Rockchip RK3288C
- EC MCU (pink) **STM32F071**
- Hooked up via SPI
- Note sizeof(AP) vs sizeof(EC)

source: ifixit.com

Chromium EC

Let's look at the firmware

Chromium EC

Overview

- Firmware of Embedded Controllers
- License: 3 Clause BSD
- Currently supported ports to MCU with
  - ARM Cortex M{0,3,4}
  - AndeStar NDS32
  - Minute-IA (x86)
- Kernel Coding style!

git: https://chromium.googlesource.com/chromiumos/platform/ec

Chromium EC

Source organization

- board/ - Board specific code
- chip/ - Chip family specific code, clocks, low level I2C, SPI ...
- common/ - framework code for I2C API, SPI API, GPIO code, PWM
- core/ - OS code, scheduler etc
- driver/ - driver code for sensors etc
- power/ - power sequencing code
- include/
- utils/ - utils like ectool
- test/ - unittests

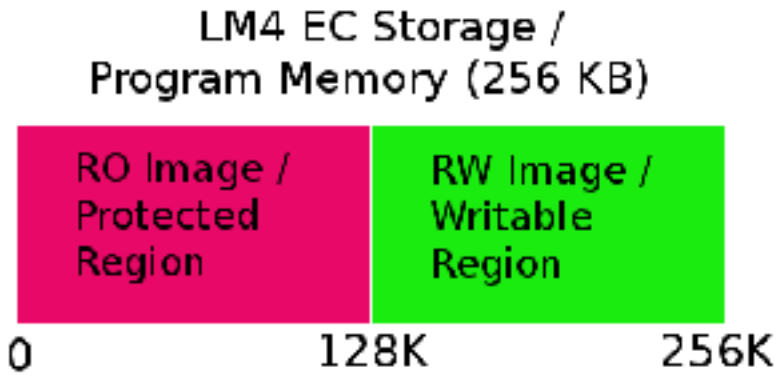
Chromium EC

Configuration / Build

- Configuration options are defined and documented in include/config.h
- Configuration options exist for debug levels, modules, features, etc
- Each board has a board.h file in board/<board>/board.h
- Datastructures get initialized in board/<board>/board.c

Chromium EC

Flash organization



- Factory programmed read-only (RO) part
- Field upgradeable read-write (RW) part
- Always boot RO, jump to RW if AP requests
- Use GPIO pin for write-protect (screw)

- Flash stores two copies of firmware

source: st.com

Chromium EC

Tasks

- Have individual stacks (250-640 bytes)
- Interrupt driven task switching
- Priority based preemption
- Mutexes
- Timers
- Events
- **NO** heap (malloc() / free())

Chromium EC

Tasks - ec.tasklist

C

```
/**
 * List of enabled tasks in the priority order
 *
 * The first one has the lowest priority.
 *
 * For each task, use the macro TASK(n, r, d, s) where :
 * 'n' in the name of the task
 * 'r' in the main routine of the task
 * 'd' in an opaque parameter passed to the routine at startup
 * 's' is the stack size in bytes; must be a multiple of 8
 */

#define CONFIG_TASK_LIST \
        TASK_ALWAYS(HOOKS, hook_task, NULL, LARGER_TASK_STACK_SIZE) \
        TASK_ALWAYS(CHARGER, charger_task, NULL, LARGER_TASK_STACK_SIZE) \
        TASK_NOTEST(CHIPSET, chipset_task, NULL, LARGER_TASK_STACK_SIZE) \
        TASK_ALWAYS(HOSTCMD, host_command_task, NULL, TASK_STACK_SIZE) \
        TASK_ALWAYS(CONSOLE, console_task, NULL, TASK_STACK_SIZE) \
        TASK_NOTEST(KEYSCAN, keyboard_scan_task, NULL, TASK_STACK_SIZE)
```

Chromium EC

Modules

- Common stuff that includes state machines is grouped into modules
- They are self contained, and often optional compile-time options
- Each module will have init function, setup statemachine
- Declare hook with initialization priority
- Examples: I2C, SPI, ADC, CHIPSET, DMA, GPIO ....

Chromium EC

Hooks

```c
static void power_lid_change(void)
{
        task_wake(TASK_ID_CHIPSET);
}
DECLARE_HOOK(HOOK_LID_CHANGE, power_lid_change,
             HOOK_PRIO_DEFAULT);
```

```c
hook_notify(HOOK_LID_CHANGE);
```

- Allows to register functions to be run when specific events occur

- Run in priority order, if multiple callbacks are registered
- Stuff like suspend, resume, lid open, button press, tick, second
- Can also call be deferred, e.g. to debounce events
- Hooks execute in stack of calling task, careful!
- Handled in the HOOKS task

Chromium EC

Console

- ccprintf() and similar functions
- Show selective debug via 'channels'
- Allows for easy debug of a lot of commands
- Adding custom commands is fairly simple
- MCU UART or USB possible

Chromium EC

Console Command Example

C

```
static int cc_pwm_duty(int argc, char *argv)
{
        /* parse, act, etc */
        return EC_RES_SUCCESS;
}

DECLARE_CONSOLE_COMMAND(pwmduty, cc_pwm_duty,
                "[channel [ | -1=disable] | [raw ]]",
                "Get/set PWM duty cycles ");
```

Chromium EC

Communication with the AP

- Packet based, i.e. header w/checksum + data
- Two versions of protocol v2 vs v3
- Busses have very different semantics, protocol hides that
- Some EC's speak both versions !?

Chromium EC

Protocol v2

- older version, send commands as follows
  - Byte 0: EC_CMD_VERSION + (command version)
  - Byte 1: Command number
  - Byte 2: Length of parameters (N)
  - Byte 3..N+2: Parameters
  - Byte N+3: 8 bit checksum over bytes 0 .. N+2
- with a response like
  - Byte 0: Result code
  - Byte 1: Length of params (M)
  - Byte 2:M+1: Parameters
  - Byte M+2: checksum

Chromium EC

Protocol v3

Current version, send packets as follows:

```c
struct ec_host_request {
        uint8_t struct_version;
        uint8_t checksum;
        uint16_t command;
        uint8_t command_version;
        uint8_t reserved;
        uint16_t data_len;
} __packed;
```

On I2C gets wrapped to do v3 structs over v2:

```c
struct ec_i2c_host_request {
        uint8_t command_protocol;
        struct ec_host_request;
} __packed;
```

source: kernel.org

Chromium EC

Communication with the AP - Declaring a hostcmd

C

```c
int temp_sensor_command_get_info(struct host_cmd_handler_args *args)
{
        const struct ec_params_temp_sensor_get_info *p = args->params;
        struct ec_response_temp_sensor_get_info *r = args->response;
        int id = p->id;

        if (id >= TEMP_SENSOR_COUNT)
                return EC_RES_ERROR;

        strzcpy(r->sensor_name, temp_sensors[id].name, sizeof(r->sensor_name));
        r->sensor_type = temp_sensors[id].type;

        args->response_size = sizeof(*r);

        return EC_RES_SUCCESS;
}
DECLARE_HOST_COMMAND(EC_CMD_TEMP_SENSOR_GET_INFO,
                     temp_sensor_command_get_info,
                     EC_VER_MASK(0));
```

Chromium EC

In your own design

Chromium EC

SoC Requirements

Bus Interface:

- SPI
  - Officially recommended, Fast
  - Requires decent SPI controller
- I2C
  - Requires I2C controller that can do repeated start
  - Drawback: Slow
- LPC
  - Drawback: Limited subset of SoCs can do it

GPIO: Power, IRQ, Suspend

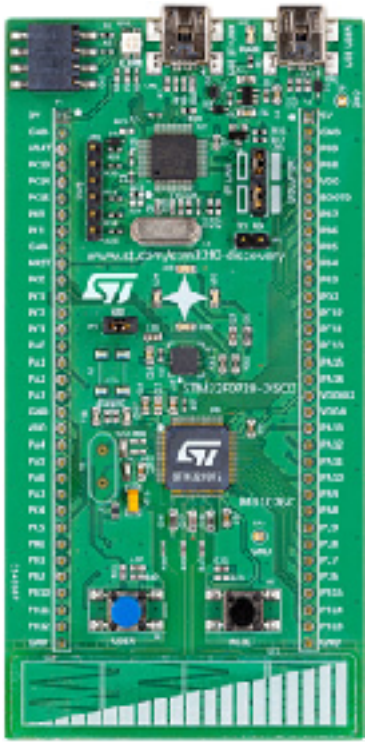
Chromium EC

Picking an MCU

- What do you want the EC to do?
- Minimal recommended set of peripherals
  - 1+ SPI, 1+ I2C
  - HW PWM
  - Lots of GPIO
  - DMA channels for SPI/I2C
  - UART
- Optional: USB (Console, DFU...), UARTs

Chromium EC

MCU Eval Board



- I'm cheap, so ...
- Discovery-Board STM32F072RB ~10$
- MCU: STM32F072RB
  - On-Board SWD debug via OpenOCD
  - 128KB flash
  - USB
  - DMA, I2C, SPI ...
  - Veyron-Jerry uses a STM32F071VB
- Is already a supported target

source: st.com

Chromium EC

GPIOs & Pin Muxing

- Pins are either
  - Inputs / outputs (strap pins, leds, write-protect)
  - Interrupt sources (external reset, buttons, switches ...)
  - Assigned to alternate functions (I2C, SPI, timer, pwm ...)
  - pin assignment and muxing happens in gpio.inc, transformed by build

Chromium EC

GPIOs - Inputs / outputs

- Mostly generic, some EC MCU specific flags
- Name, Pin, Flags (i.e. Level on reset, Pull-ups, Open Drain ...)
- see include/gpio.h

```c
GPIO(WP_L,        PIN(B, 4), GPIO_INPUT)
[...]
GPIO(BAT_LED_RED, PIN(B, 11), GPIO_OUT_HIGH)
[...]
GPIO(EC_INT_L,    PIN(B, 9),  GPIO_OUT_LOW)
```

API calls: gpio_get_level() / gpio_set_level()

Chromium EC

GPIOs - Interrupt sources

- Again, mostly generic
- Name, Pin, Flags (i.e. Edge, Pull-ups, Open Drain ...), handler
- examples: board/*/gpio.inc

```c
GPIO_INT(SPI1_NSS,   PIN(A, 4),  GPIO_INT_BOTH, spi_event)
GPIO_INT(AC_PRESENT, PIN(C, 6),  GPIO_INT_BOTH | GPIO_PULL_UP, extpower_interrupt)
GPIO_INT(SUSPEND_L,  PIN(C, 7),  GPIO_INT_BOTH, power_signal_interrupt)
```

Chromium EC

GPIOs - Alternate Functions

- Defined by architecture
- PIN_MASK(A,0x00f0) = PA4, PA5, PA6, PA7
- Alternate number 0 (from datasheet)
- Module that will deal with it (SPI)
- Flags same as before

```c
ALTERNATE(PIN_MASK(A, 0x00f0), 0, MODULE_SPI,  0)
ALTERNATE(PIN_MASK(A, 0x0600), 1, MODULE_UART, 0)
ALTERNATE(PIN_MASK(B, 0x00c0), 1, MODULE_I2C,  0)
```

Chromium EC

GPIOs - Faking it

- Some generic code expects certain signals to be there (WP_L, ...)
- Makes stuff work, by pretending UNIMPLEMENTED GPIOs exist
- E.g. use generic LED code, but you have only one LED
- Flags same as before

```
UNIMPLEMENTED(WP_L)
UNIMPLEMENTED(LED_BAT)
```
C

Chromium EC

Power sequencing

- Most modern SoCs need certain sequence
- Usually goes like this
  - Turn on X volt rail
  - Wait max time Y for power good signal
  - If timeout happened, handle it, otherwise proceed
  - power subfolder contains sequences for common chromebooks
    - RK3399, RK3288-C
    - Apollolake, Baytrail, Haswell, Skylake
    - Tegra
    - Mediatek

Chromium EC

Power sequencing - your SoC

- Implement statemachine
- Use ACPI S/G states G3/S5/S3/S0 ...
- Describe what needs to happen to proceed, and how to handle failure

C

```
case POWER_S0:
        if (!power_has_signals(IN_PGOOD_S3) ||
            forcing_shutdown ||
            !(power_get_signals() & IN_SUSPEND_DEASSERTED))
                return POWER_S0S3;
        [...]
        if (power_wait_signals_timeout(IN_PGOOD_AP | IN_PGOOD_SYS,
                                       PGOOD_AP_DEBOUNCE_TIMEOUT)
                                       == EC_ERROR_TIMEOUT)
        return POWER_S0S3;
```

Chromium EC

Interfacing Peripherals

- I2C (Master) API
  - i2c_readX() / i2c_writeX() ... X = 8,16,32
- I2C tunnel
  - Simulates i2c bus over SPI/I2C/LPC connection
  - Allows host to access slave devices
  - Might come in handy
- SPI (Master) API
  - spi_transaction() data, txlen, rxdata, rxlen
  - spi_transaction_async() same, but hand over to DMA
  - supports (some) SPI flash devices

Chromium EC

Kernel & U-Boot

- I like to work with upstream stuff
- Chromebooks use Linux Kernel, so that works really well
- Most use depthcharge instead of u-boot, code gets less flight time
- I'm still working on adding software sync to u-boot
- I didn't look at adding verified boot

Chromium EC

Kernel & U-boot - Devicetree (I2C)

Instantiation for I2C:

DEVICETREE

```
&i2c0 {
        ec: embedded-controller@1e {
                reg = <0x1e>;
                compatible = "google,chromium-ec-i2c";
                interrupts = <14 IRQ_TYPE_LEVEL_LOW>;
                interrupt-parent = <&gpio0>;
                wakeup-source;
        };
};
```

Check out Documentation/devicetree/bindings/mfd/cros-ec.txt

Chromium EC

Kernel & U-boot - Devicetree (SPI)

DEVICETREE

```
&spi0 {
        ec: embedded-controller@0 {
                reg = <0x0>;
                compatible = "google,cros-ec-spi";
                interrupts = <14 IRQ_TYPE_LEVEL_LOW>;
                interrupt-parent = <&gpio0>;
                wakeup-source;
        };
};
```

Check out Documentation/devicetree/bindings/mfd/cros-ec.txt

Chromium EC

Linux interface

- Nothing too exciting, MFD device
- MFD subdevices include
  - I2C tunnel (i2c bus)
  - PWM channels (pwm)
  - Battery (power-supply)
  - Lightbar
- Character device allowing for ioctl(2) for ectool / flashrom
- Code mostly
  - drivers/mfd/cros_ec_{i2c,spi}.c
  - drivers/platform/chrome/

Chromium EC

Linux interface - sysfs

CONSOLE

```
[mfischer@chromebook]$ tree /sys/class/chromeos/cros_ec
/sys/class/chromeos/cros_ec
|-- dev
|-- device -> ../../../cros-ec-ctl.0.auto
|-- flashinfo
|-- lightbar
|-- power
|   |-- autosuspend_delay_ms
|   |-- control
|   |-- runtime_active_time
|   |-- runtime_status
|   `-- runtime_suspended_time
|-- reboot
|-- subsystem -> ../../../../../../../../../class/chromeos
|-- uevent
|-- vbc
`-- version
```

Chromium EC

Linux interface - ectool

- Useful tool to test and inspect
- Uses characterdev with ioctl(2) to
  - GPIO state
  - Firmware Versions
  - Reboot the EC
  - Flash info
  - Read / write Firmware
  - Send commands, useful for development
  - ...

Chromium EC

U-Boot

- Nothing too exciting, misc device
- Functionality exposed
  - I2C tunnel (i2c bus integrated in dm)
  - Reading / Writing firmware (crosec read / write)
  - Getting flash information (crosec flashinfo)
  - Reboot EC into RO / RW (crosec reboot RO / RW)
- Code mostly
  - drivers/misc/cros_ec_{i2c,spi}.c
  - cmd/cros_ec.c

Community & Wrap-up

Awesome, what bout it?

Chromium EC

Community (Subjective!)

- Mostly Google driven development
- Roadmap very nebulous
- Code review via gerrit :(
- Mailing list very low traffic
- Very receptive to patches, quick reviews

Thank You!

Get in touch

g+	plus.google.com/117055022771319709709
twitter	@fischmz
www	www.ni.com
github	github.com/mfischer