An open-source Emulator of Legacy Apple Devices

A Dive into Reverse Engineering and Understanding the iPod Touch

Martijn de Vos
About Me

• Postdoctoral Researcher @ EPFL, Switzerland
• Researcher in distributed ML Systems
• Reverse engineering enthousiast
  • Mobile banking apps during PhD
Motivation

• Inspired by Jonathan Afek’s blog post
  • “Running iOS in QEMU to an interactive bash shell”
• Fun challenge
• (long-term) hardware preservation
Where to start?

• Which device to emulate?
• Modern embedded devices are hard to emulate
  • Neural engines
  • FaceID/TouchID engines
  • Secure enclaves
  • Trust caches
• iPod Touch 1G looks like a promising starting point
  • Released in 2007, ARMv6 instruction set
  • Should be simple enough to emulate *

* Famous last words
Related Projects

• Very early attempt by @cmwdotme to emulate S5L8900
  • Evolved into Correlium
• iPhone 6s plus emulation by Johathan Afek
• iPhone 11 emulator by Trung Nguyen
• OpeniBoot
QEMU

• Open-source framework for hardware emulation
• Define peripherals and their expected behaviour
• Support for popular hardware and protocols
  • USB, NICs, SPI, I²C, SDIO, ...

• Poor documentation 😞
Debugging with GDB
Reverse Engineering with Ghidra
Philosophy

• Stay close to the real hardware
• Avoid relying on image patching if possible
• Hacks and workarounds might bite us later, better get it right early on

• As expected, emulator ended up with a bunch of hacks 😊
iPod Touch 1G/2G Boot Chain

SecureRom bootrom → Low-level bootloader → iBoot → XNU Kernel → Springboard

with launchd

→ Alarm.app

→ Safari.app
Bootrom

• Very first code that executes on the device
  • Initializes some key peripherals
  • Loads LLB or puts the device into DFU (restoration) mode

• Jumps to unknown memory addresses
• Probably some proprietary encryption/decryption logic by Samsung
• No access to/dumps of the memory being jumped to 😞
  • Didn’t have a physical IT1G at that time
Low-level Bootloader (LLB)

- Initializes some peripherals and loads iBoot
- Same problem, jumps to unknown memory locations
- Let’s skip the bootrom and LLB, and go straight to iBoot!

```c
// load iBoot
file_data = NULL;
if (g_file_get_contents(nms->iboot_path, (char **)&file_data, &fsize, NULL)) {
    allocate_ram(sysmem, "iboot", IBOOT_BASE, 0x4000000);
    address_space_rwlock(nsas, IBOOT_BASE, MEMTXATTRS_UNSPECIFIED, (uint8_t *)file_data, fsize, 1);
}
```
iBoot (main bootloader)

• Responsible for loading the kernel from NAND
• iBoot source code got leaked in 2018

**Apple has confirmed that some of the source code for its iOS mobile operating system has been leaked online.**

The boot-up source code used on its older iOS 9 operating platform was posted on code-sharing website Github.

Apple typically keeps most of its iOS source code private and ordered Github to remove the content.
Device Tree

- Lists all peripherals and properties
- Included in the IPSW, populated by iBoot
- I used a public DT dump published on GitHub as reference
These devices are complicated!

Schematic of the iPod Touch 2G

- CPU (S5L8720)
- DMA Controller (PL080)
- Vector Interrupt Controller (VIC) (PL192)
- GPIO Controller
- SPI Controller 1
- SPI Controller 4
- SDIO Controller
- WiFi (BCM4325)
- UART
- Vector Interrupt Controller (VIC) (PL192)
- USB OTG
- Clock
- SysIC
- USB Phys.
- ChipID
- SHA1 Engine
- AES Engine
- PKE Engine
- PMU (D1759)
- Accelerometer (LIS302DL)
- I2C Controller 1
- I2C Controller 2
- SPI Controller 1
- SPI Controller 4
- DMA Controller (PL080)
- FMSS Controller
- NOR Flash Controller
- Multitouch
- RAM
- NAND flash memory
- NOR memory
- MIPI Controller
- LCD Controller
- TV Out
- Display, sound and Graphics
- Serial Peripheral Interface
- Secure Digital Input Output
- Inter-Integrated Circuit (I2C)
- Memory and Persistence
- Crypto: SHA1, AES, PKE
- TDA:
  - Accelerometer (LIS302DL)
  - PMU (D1759)
- Clock:
  - USB OTG
  - UART
- Misc:
  - HDMI
  - TV Out
Peripherals

• The kernel communicates with peripherals through memory-mapped I/O (MMIO)
• Each peripheral has a dedicated space in memory
static void ipod_touch_sysic_class_init(ObjectClass *klass, void *data) {
}

static const TypeInfo ipod_touch_sysic_type_info = {
    .name = TYPE_IPOD_TOUCH_SYSIC,
    .parent = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(IPodTouchSYSICState),
    .instance_init = ipod_touch_sysic_init,
    .class_init = ipod_touch_sysic_class_init,
};

static void ipod_touch_sysic_register_types(void) {
    type_register_static(&ipod_touch_sysic_type_info);
}

type_init(ipod_touch_sysic_register_types)
Talking to Peripherals

```c
static uint64_t ipod_touch_sysic_read(void *opaque, hwaddr addr, unsigned size)
{
    IPodTouchSYSICState *s = (IPodTouchSYSICState *) opaque;
    fprintf(stderr, "%s: offset = 0x%08x\n", __func__, addr);

    switch (addr) {
        case POWER_ONCTRL:
            return 42;
        default:
            break;
    }
    return 0;
}

static void ipod_touch_sysic_write(void *opaque, hwaddr addr, uint64_t val, unsigned size)
{
    IPodTouchSYSICState *s = (IPodTouchSYSICState *) opaque;
    fprintf(stderr, "%s: writing 0x%08x to 0x%08x\n", __func__, val, addr);

    switch (addr) {
        case POWER_ONCTRL:
            // do something
            break;
    }
}
```
static uint64_t ipod_touch_spi_read(void *opaque, hwaddr addr, unsigned size) {
    IPodTouchSPIState *s = IPOD_TOUCH_SPI(opaque);
    //printf("%s (base %d): read from location 0x%08x\n", __func__, s->base, addr);

    uint32_t r;
    bool run = false;

    r = s->regs[addr >> 2];
    switch (addr) {
    case R_RXDATA: {
        const uint8_t *buf = NULL;
        int word_size = apple_spi_word_size(s);
        uint32_t num = 0;
        if (fifo8_is_empty(&s->rx_fifo)) {
            hw_error("Rx buffer underflow\n");
            qemu_log_mask(LOG_GUEST_ERROR, "%s: rx underflow\n", __func__);
            r = 0;
            break;
        }
        buf = fifo8_pop_buf(&s->rx_fifo, word_size, &num);
        memcpy(&r, buf, num);
        if (fifo8_is_empty(&s->rx_fifo)) {
            run = true;
        }
        break;
    }
    case R_STATUS: {
        break;
    }
    }
Attaching Peripherals to the Machine

```c
dev = qdev_new("ipodtouch.sysic");
IPodTouchSYSICState *sysic_state = IPOD_TOUCH_SYSIC(dev);
nms->sysic = (IPodTouchSYSICState *) g_malloc0(sizeof(struct IPOdTouchSYSICState));
memory_region_add_subregion(sysmem, SYSIC_MEM_BASE, &sysic_state->iomem);
bu$dev = SYS_BUS_DEVICE(dev);
for(int grp = 0; grp < GPIO_NUMINTGROUPS; grp++) {
    sysbus_connect_irq(busdev, grp, s5l8900_get_irq(nms, S5L8900_GPIO_IRQS[grp]));
}
```
XNU Kernel

• First loads and starts all device drivers declared in the device tree
  • Uses IOKit
• Starting a driver usually involves resetting the peripheral
• After all drivers are loaded, it starts launchd
~20 peripherals later...

- Most key peripherals fully functional
  - Clock, timer, vector interrupt controller (VIC), DMA, crypto engines, ...
- Only partial support for other peripherals
  - Just enough to make it past the initialization
  - TVOut, GPU, accelerometer, light sensor ...
- Avoided GPU rendering with a flag

- Lots of work to do still, but we boot to userland! 😊
Persistence

• Two types of storage: NOR and NAND
• Key differences between iPod Touch 1G and 2G
• Emulator expects proper file system layouts
• Figuring out the layouts took most time (especially for NAND)

• Ended up with two scripts to generate the NOR and NAND images
NAND

Source: Modern SSDs (Fall 2022), Jin-Soo Kim, Seoul National University
Multitouch

- Particularly challenging
  - Converting touch to coordinates is quite difficult
  - Complex initialization procedure
- Communicated with through SPI
- To get this working, I required the real device
- Installed OpeniBoot to read/analyze frames
Hello World!

iPod Touch 1G
iPhoneOS 1.0

iPod Touch 2G
iOS 2.1.1
QEMU-iOS

• An emulator for legacy Apple devices
• https://github.com/devos50/qemu-ios
• Support for iPod Touch 1G and 2G
• Current focus on iPod Touch 2G stability

• Contributions are welcome!
Thank you!

https://devos50.github.io
(some blog posts)