

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 enthusiast
 - 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 *



iPod Touch 1G

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

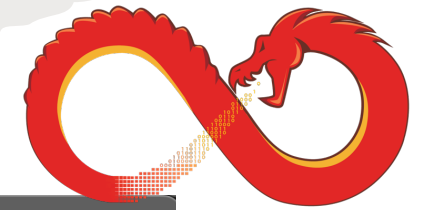
```
build — gdb — 175x48
~/Documents/generate_nor_it2g/nor.bin -serial mon:stdio -cpu max -m 2G -d unimp -S -s
~/Documents/qemu-it2g/build — gdb

Register group: general
r0      0x0      0          r1      0x0      0          r2      0x0      0
r3      0x0      0          r4      0x0      0          r5      0x0      0
r6      0x0      0          r7      0x0      0          r8      0x0      0
r9      0x0      0          r10     0x0      0          r11     0x0      0
r12     0x0      0          sp      0x0      0x0        lr      0x0      0
pc      0x40     0x40      cpsr    0x40001d3 1073742291 fpscr   0x0      0
fpsid   0x41034070 1090732144 fpexc   0x0      0          ID_MMFR1 0x40000000 1073741824
IFAR    0x0      0          ID_MMFR2 0x1260000 19267584  PMEVTYPER0 0x0      0
ID_MMFR3 0x2122211 34742801  VESR_EL2 0x0      0          PMEVTYPER1 0x0      0
NSACR   0xc00    3072     DFAR    0x0      0          PMEVTYPER2 0x0      0
CBAR    0x0      0          ERRIDR_EL1 0x0      0          PMEVTYPER3 0x0      0
DBGBVR4_EL1 0x0      0          PMEVTYPER4 0x0      0          DBGBCR4_EL1 0x0      0
PMEVTYPER5 0x0      0          PMCCNTR 0x0      0          VDISR_EL2 0x0      0
RES_0_C0_C4_X 0x0      0          CNTP_CVAL 0x0      0          JIDR    0x0      0

>0x40 mov r0, pc
0x44 ldr r1, [pc, #708] @ 0x310
0x48 sub r0, r0, r1
0x4c ldr r1, [pc, #704] @ 0x314
0x50 cmp r0, r1
0x54 beq 0x78
0x58 ldr r1, [pc, #692] @ 0x314
0x5c ldr r2, [pc, #692] @ 0x318
0x60 ldr r3, [r0], #4
0x64 subs r2, r2, #4
0x68 str r3, [r1], #4
0x6c bne 0x60
0x70 ldr r1, [pc, #668] @ 0x314
0x74 bx r1

remote Thread 1.1 In:
(gdb) layout r
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x00000000 in ?? ()
(gdb) si
0x00000040 in ?? ()
(gdb) █
```

Reverse Engineering with Ghidra



GHIDRA

CodeBrowser(3): ios_kernel_reverse:iPod Touch 2G/kernel.out

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

- kernel.out
 - __TEXT
 - __DATA
 - __HIB
 - __KLD
 - __PRELINK
 - __LINKEDIT
 - ABSOLUTE

Symbol Tree

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager

- BuiltInTypes
- kernel.out

Listing: kernel.out

```
*bootrom_s5l8900 kernel.out
r0:4 work_loop2
Stack[-0x18]:4 loca_18
AppleS5L8900XSPIController::start XREF [
stmdb sp!,{r4,r5,r6,r7,lr}
add r7,sp,#0xc
sub sp,sp,#0x4
ldr r3=>DAT_c02a9a30,[DAT_c05fa7d0]
cpy r4,this
cpy r5,provider
mov lr,pc
ldr pc=>AppleARMSPIController::start,[r3,7]

cmp this,#0x0
beq LAB_c05fa554
ldr r3,[r5,#0x0]
ldr provider=>s_spi-version_c0600a5c,[DAT_

cpy this,r5
mov lr,pc
ldr pc,[r3,#0xa4]
ldr provider,[DAT_c05fa7d8]
ldr r6,[DAT_c05fa7dc]
ldr provider=>OSData,[provider->vtable,#0]

blx r6=>OSMetaClassBase::safeMetaCast
subs r3,spi_version_data,#0x0
beq LAB_c05fa540
ldr r3,[r3,#0x0]
mov lr,pc
ldr pc,[r3,#0x84]
ldr spi_version_data,[r0,#spi_version_data]
str spi_version_data,[r4,#0x94]

B_c05fa540 XREF [
```

Decompile: start - (kernel.out)

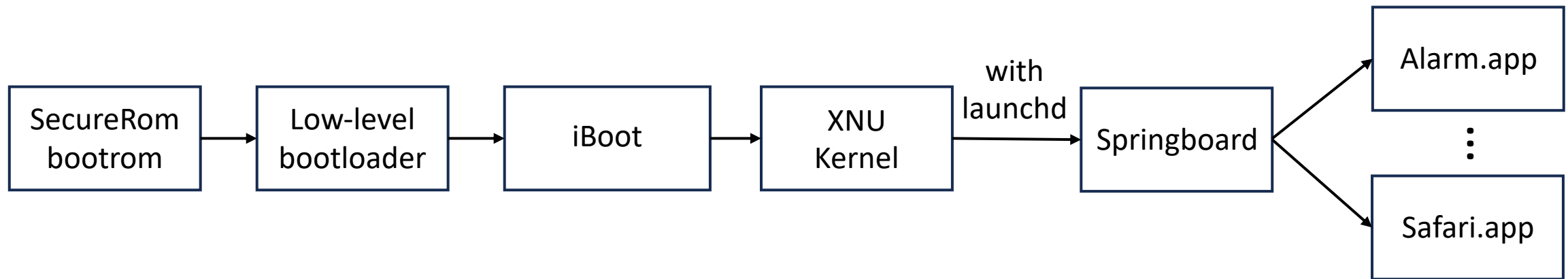
```
56 }
57 this->field42_0xa8 = iVar1;
58 provider_cast =
59     (AppleARMIIODevice *)
60     OSMetaClassBase::safeMetaCast((OSMetaClassBase *)provider,(OSMetaC
61 ;
62 this->provider = provider_cast;
63 if (provider_cast != (AppleARMIIODevice *)0x0) {
64     pclk = (*(code *)provider_cast->vtable->getClockFrequency3)(provider_
65     this->pclk = pclk;
66     if (pclk != 0) {
67         nclk = (*(code *)this->provider->vtable->getClockFrequency3)(this->
68         this->nclk = nclk;
69         if (nclk != 0) {
70             memory_map = (IOMemoryMap *)
71                 (*(code *)this->provider->vtable->super).mapDeviceM
72                 (this->provider,0,0);
73         this->memory_map = memory_map;
74         if (memory_map != (IOMemoryMap *)0x0) {
75             iVar1 = (*(code *)memory_map->vtable->getVirtualAddress());
76             this->memory_vaddr = iVar1;
77             param1 = (char *)(*(code *)this->provider->vtable->super).supe
78                 (this->provider,0);
79             kprintf("AppleS5L8900XSPIController::start: %s: _spiBaseAddress
80                 ,param1,this->memory_vaddr,(long)this->spi_version);
81             filter_interrupt_event_source =
82                 (IOInterruptEventSource *)
83                 IOFilterInterruptEventSource::filterInterruptEventSource
84                 (this,(ulong)this->vtable->onInterrupt,this->vta
85                 this->provider,0);
86             this->interrupt_event_source = filter_interrupt_event_source;
87             if (filter_interrupt_event_source != (IOInterruptEventSource *)
88                 work_loop = (IOWorkLoop *)(*(code *)this->vtable->getWorkLoop
89                 (*(code *)work_loop->vtable->addEventSource)(work_loop,this->
90                 dma_event_source =
91                 (int *)IODMAEventSource::dmaEventSource
```


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



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!

```
// load iBoot
file_data = NULL;
if (g_file_get_contents(nms->iboot_path, (char **)&file_data, &fsize, NULL)) {
    allocate_ram(systemem, "iboot", IBOOT_BASE, 0x400000);
    address_space_rw(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

AFP

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.

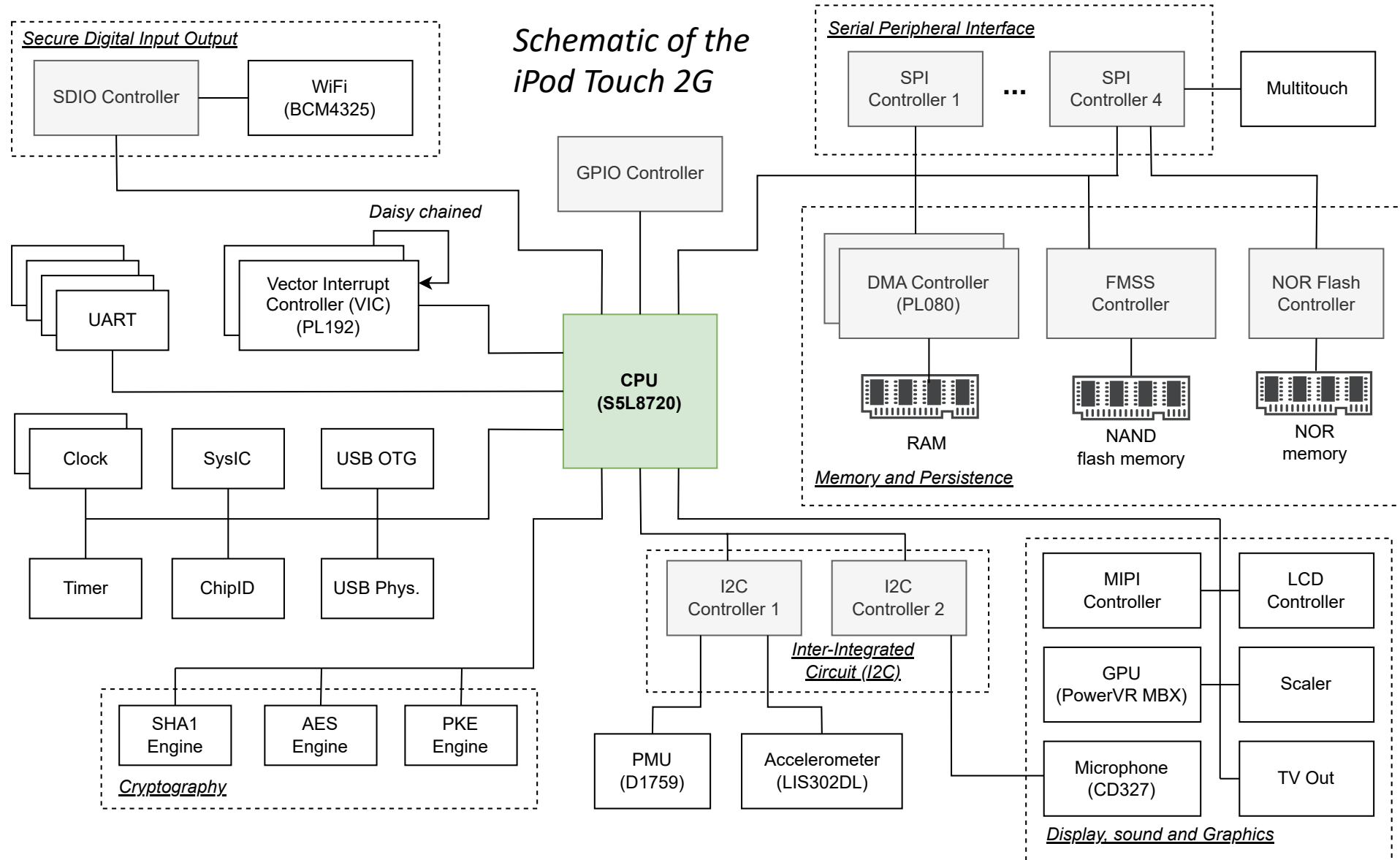
Apple typically keeps most of its iOS source code private and ordered Github to remove the content.

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

```
+--o aes@C00000 <class AppleARMIODevice, registered, matched, active, bus
{
  "IOInterruptControllers" = ("IOInterruptController00904AD0")
  "clock-ids" = <02000000>
  "IODeviceMemory" = (({"address"=952107008,"length"=4096}))
  "clock-gates" = <0a000000>
  "xxxxxxxx-disable_keys" = <"0_□","Ksid">
  "AAPL,phandle" = <90b29000>
  "IOInterruptSpecifiers" = (<27000000>)
  "name" = <"aes">
  "device_type" = <"aes">
  "interrupts" = <27000000>
  "compatible" = <"aes,s5l8900x">
  "reg" = <0000c00000100000>
  "interrupt-parent" = <d04a9000>
}
+--o AppleS5L8900XAES <class AppleS5L8900XAES, registered, matched, act
{
  "IOProviderClass" = "AppleARMIODevice"
  "IOProbeScore" = 0
  "CFBundleIdentifier" = "com.apple.driver.AppleS5L8900XCrypto"
  "IOMatchCategory" = "IODefaultMatchCategory"
  "IOUserClientClass" = "IOAESAcceleratorUserClient"
  "IONameMatched" = "aes,s5l8900x"
  "IOClass" = "AppleS5L8900XAES"
  "IONameMatch" = "aes,s5l8900x"
  "IOPowerManagement" = {"CurrentPowerState"=1}
}
```

These devices are complicated!



Peripherals

- The kernel communicates with peripherals through memory-mapped I/O (MMIO)
- Each peripheral has a dedicated space in memory

```
// MMIO addresses
#define VROM_MEM_BASE 0x0
#define INSECURE_RAM_MEM_BASE 0x8000000
#define SECURE_RAM_MEM_BASE 0xB000000
#define FRAMEBUFFER_MEM_BASE 0xFB00000
#define IB00T_MEM_BASE 0xFF00000
#define SRAM1_MEM_BASE 0x22020000
#define SHA1_MEM_BASE 0x38000000
#define DMAC0_MEM_BASE 0x38200000
#define USBOTG_MEM_BASE 0x38400000
#define DMAC1_0_MEM_BASE 0x38700000
#define DISPLAY_MEM_BASE 0x38900000
#define FMSS_MEM_BASE 0x38A00000
#define AES_MEM_BASE 0x38C00000
#define SDIO_MEM_BASE 0x38D00000
#define VIC0_MEM_BASE 0x38E00000
#define VIC1_MEM_BASE 0x38E01000
#define EDGEIC_MEM_BASE 0x38E02000
#define H264_MEM_BASE 0x38F00000
#define SCALER_CSC_MEM_BASE 0x39000000
#define TVOUT_MIXER2_MEM_BASE 0x39100000
#define TVOUT_MIXER1_MEM_BASE 0x39200000
#define TVOUT_SDO_MEM_BASE 0x39300000
#define SYSIC_MEM_BASE 0x39700000
#define DMAC1_1_MEM_BASE 0x39900000
#define MBX1_MEM_BASE 0x3B000000
#define MBX2_MEM_BASE 0x39400000
#define SPI0_MEM_BASE 0x3C300000
#define USBPHYS_MEM_BASE 0x3C400000
#define CLOCK0_MEM_BASE 0x3C500000
#define I2C0_MEM_BASE 0x3C600000
#define TIMER1_MEM_BASE 0x3C700000
#define I2C1_MEM_BASE 0x3C900000
#define UART0_MEM_BASE 0x3CC00000
#define SPI1_MEM_BASE 0x3CE00000
#define GPIO_MEM_BASE 0x3CF00000
#define PKE_MEM_BASE 0x3D000000
#define CHIPID_MEM_BASE 0x3D100000
#define SPI2_MEM_BASE 0x3D200000
#define UNKNOWN1_MEM_BASE 0x3D700000
#define MIPI_DSI_MEM_BASE 0x3D800000
```


Initializing Hardware with QEMU

```
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

```
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;
    }
}
```

More Complicated Hardware

```
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);

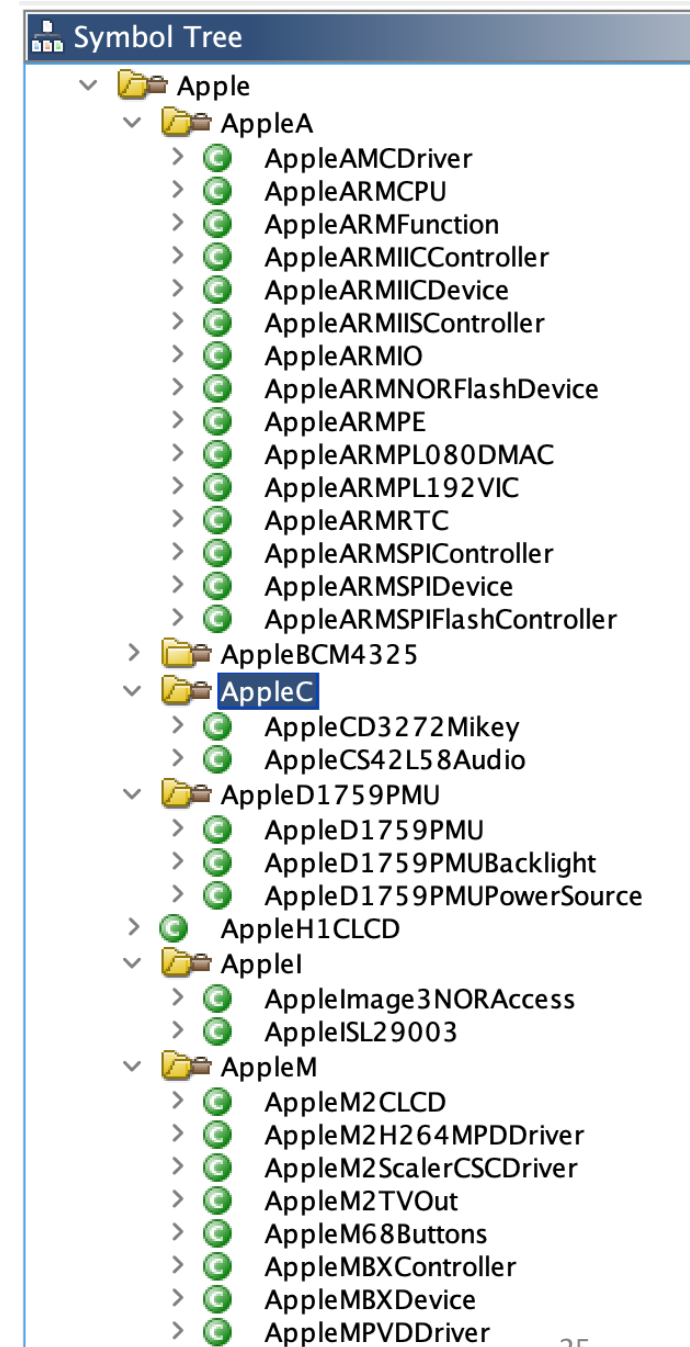
            if (fifo8_is_empty(&s->rx_fifo)) {
                run = true;
            }
            break;
        }
        case R_STATUS: {
```

Attaching Peripherals to the Machine

```
dev = qdev_new("ipodtouch.sysic");
IPodTouchSYSICState *sysic_state = IPOD_TOUCH_SYSIC(dev);
nms->sysic = (IPodTouchSYSICState *) g_malloc0(sizeof(struct IPodTouchSYSICState));
memory_region_add_subregion(system, SYSIC_MEM_BASE, &sysic_state->iomem);
busdev = 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! 😊

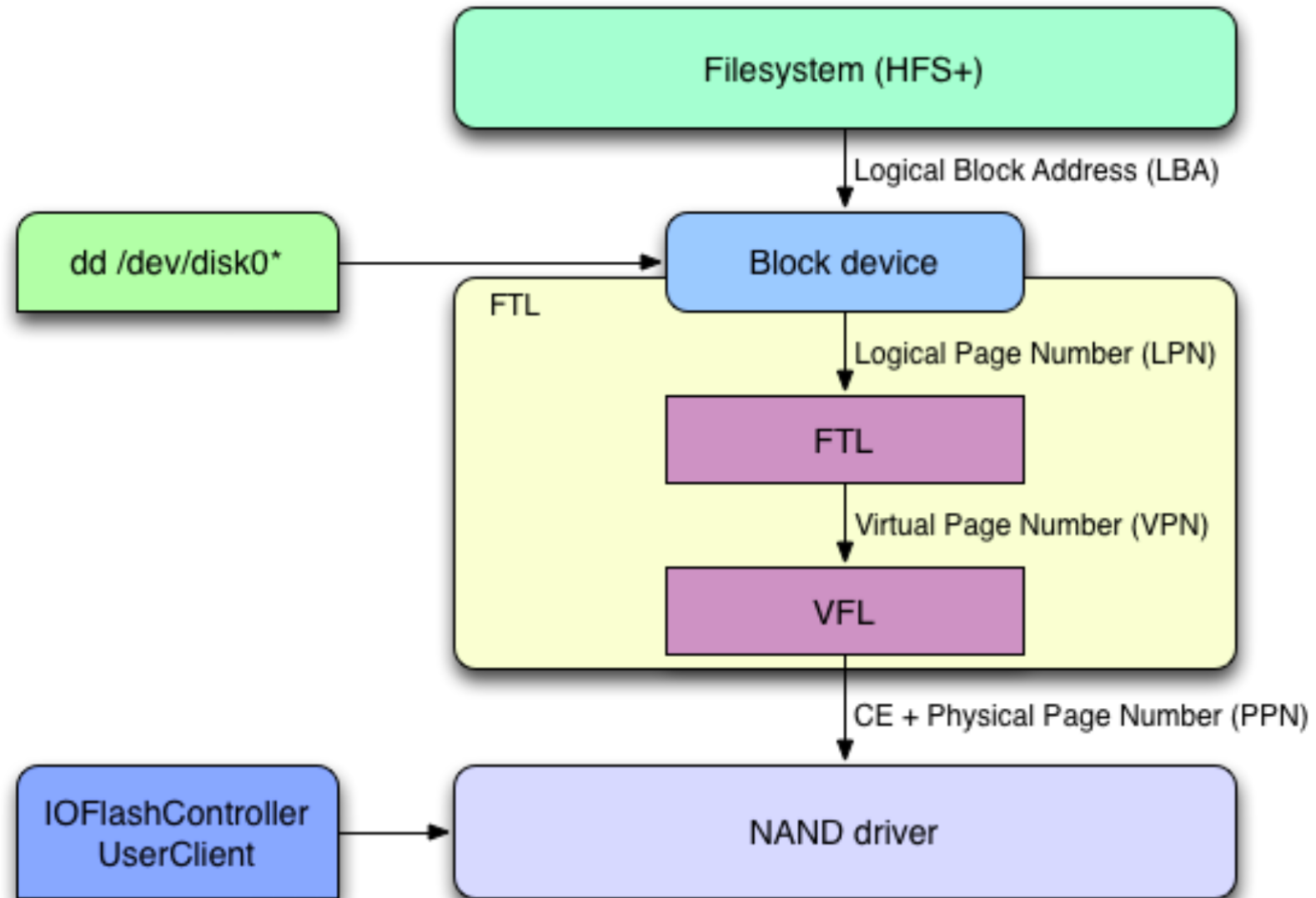
```
/* ipod_touch.c
/* ipod_touch_8900_engine.c
/* ipod_touch_adm.c
/* ipod_touch_aes.c
/* ipod_touch_chipid.c
/* ipod_touch_clock.c
/* ipod_touch_gpio.c
/* ipod_touch_lcd.c
/* ipod_touch_lcd_panel.c
/* ipod_touch_lis302dl.c
/* ipod_touch_multitouch.c
/* ipod_touch_nand.c
/* ipod_touch_nand_ecc.c
/* ipod_touch_pcf50633_pmu.c
/* ipod_touch_sdio.c
/* ipod_touch_sha1.c
/* ipod_touch_spi.c
/* ipod_touch_sysic.c
/* ipod_touch_timer.c
/* ipod_touch_tvout.c
/* ipod_touch_usb_otg.c
```

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)

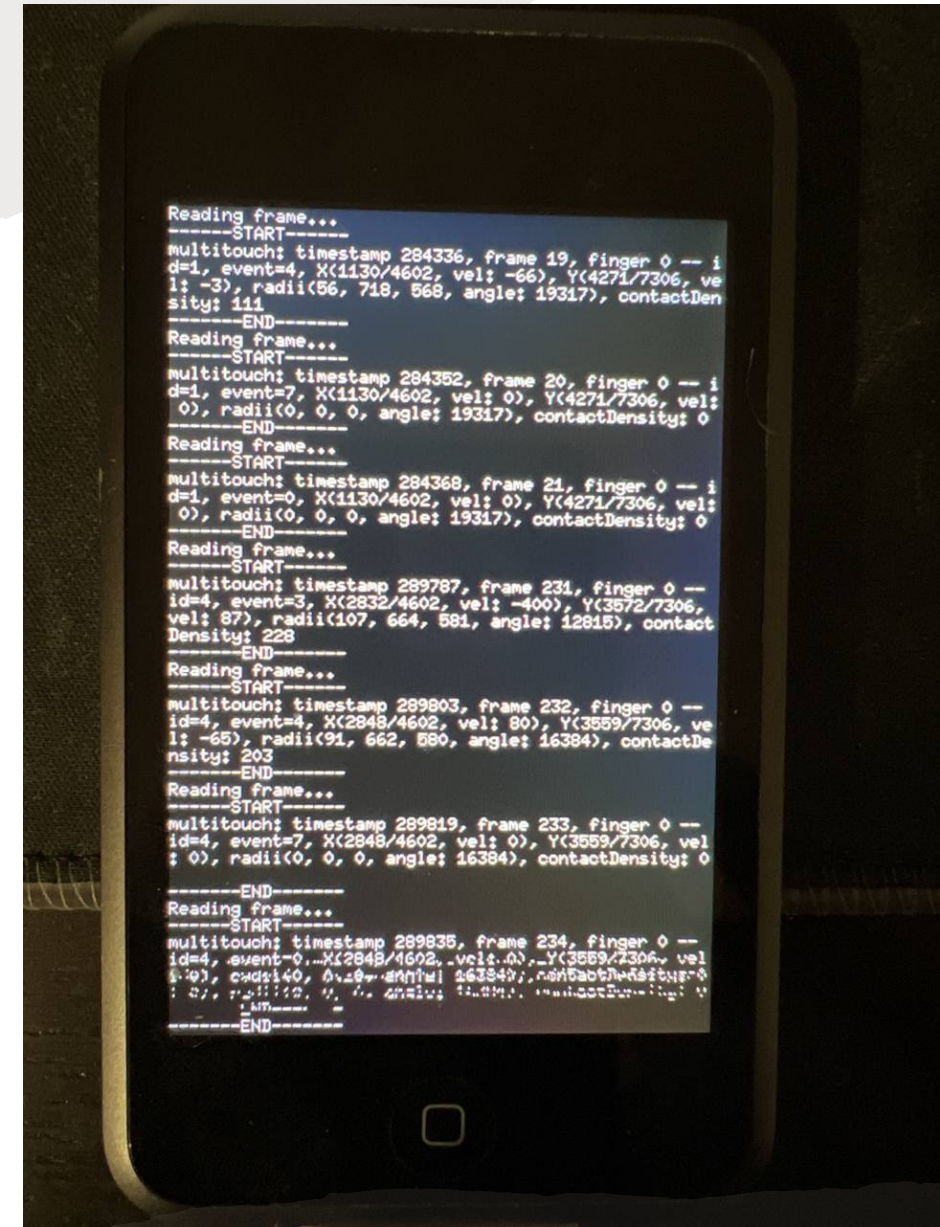
- Ended up with two scripts to generate the NOR and NAND images

NAND

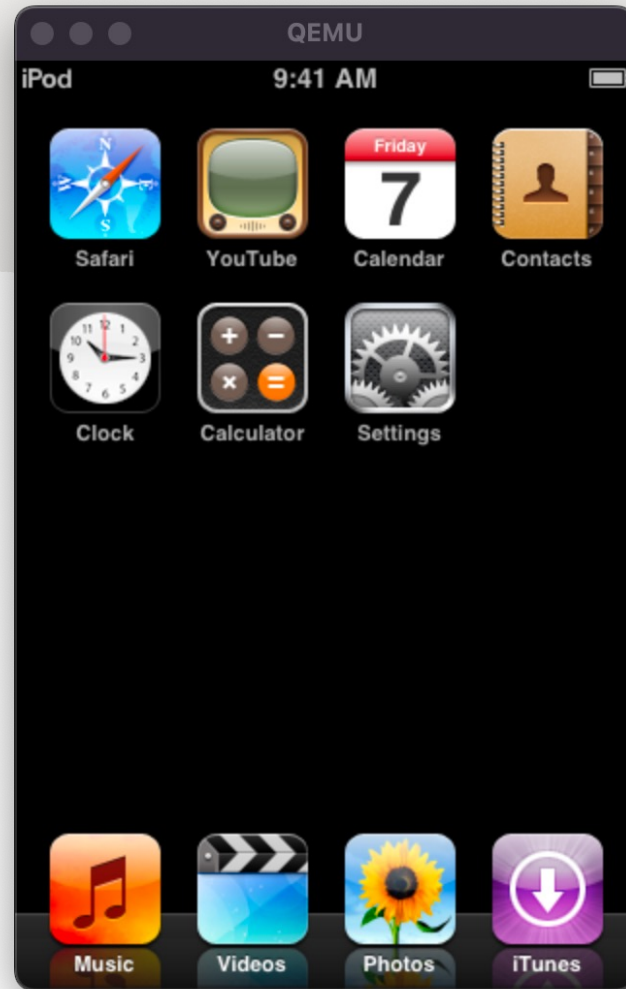


Multitouch

- 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



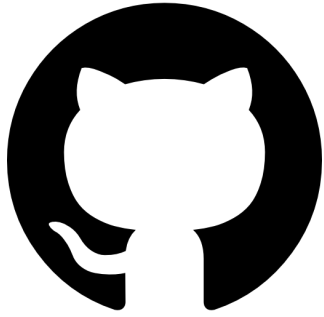
iPod Touch 2G
iOS 2.1.1

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!



devos50

<https://devos50.github.io>

(some blog posts)