Provocation = $12
Provocation

Chinese low-end hardware platform

- 260 MHz 32-bit CPU, 8MiB RAM
- Quad-band GSM
- Bluetooth
- OLED display
- MP3 player
- Li-Poly battery
- $12 qty 1

Western low-end hardware platform

- 16 MHz 8-bit CPU, 2.5k RAM
- USB serial interface
- Voltage regulator
- $29 qty 1
I Can Has Documentation?
Downloadable source files...
Phone OS Source...

- “MKT11B” is a ~7.5GiB source archive, downloadable from Baidu
But is it Open?

- Docs are either restricted, or unspecified.
- Shanzhai dev environment is Visual Studio plus a cracked copy of ARM's RealView compiler
China Don't Care

• This technicality does not stop Shanzhai (or most Chinese)
  – There is a view that Western IP law can be unethical: drug companies overcharging for life-saving drugs, $20 IP burden for mobile phones (or $30 DVDs) is seen as stealing food from the poor to give to the rich, etc.
  – Enforcement of laws is selective and subjective
Fruits of Permissive IP Environment

Image: Rachel Kalmar

Image: Halfdan
The West *Does* Care

• Can't build a western business on “stolen” IP
  - So ask Mediatek for a license?
    • Either no response or
    • (For example) $250k pre-payment on order volume for access to docs

• Not practical for individuals & startups!
Weltschmerz

• So you're saying China startups get to make whole phones... and Western startups get to make just phone accessories, and that's a Good Thing?
Who own copyright to rage faces?
Can We Hack the System?
Can We Hack the System?
What's at Stake

- We are not lawyers. We're sharing with you our personal views.
  - However, law is a tool – and tools are meant to be used
  - Also, we only know about laws in the US – things are different elsewhere

- Copyright
  - Complex legal landscape:
    - Traditional copyright law – do we have a right to make a copies for interoperability and reverse engineering, and to what extent can we use their code as reference code?
    - DMCA – did we have to defeat any technological measures that control access to the documentation or code?

- CFAA – did we have to access, without authorization, any servers to obtain documents and code?

- Contract law – are we under NDA, EULA, TOU, TOS, etc. that could waive or nullify some of our Fair-use rights?

- Complex set of issues:
  - Check out https://www.eff.org/issues/coders/reverse-engineering-faq

- Patent
  - Not just Mediatek, but also any potential patent holder (e.g. GSM)
Traditional Copyright

- Expression is copyrightable, but not facts
  - A list of phone numbers and names is not copyrightable
  - “Notwithstanding a valid copyright, a subsequent compiler remains free to use the facts contained in another's publication to aid in preparing a competing work, so long as the competing work does not feature the same selection and arrangement” – Justice O'Connor (*Feist v. Rural*)
Our Assertion

• Not (yet) court-tested, but here is our assertion:
  – A list of registers, their addresses, and bitfields are like a phone directory
  – A list of address and data pairs used to initialize a hardware function, is a fact
    • e.g. “set up the PLL by writing these data to these addresses in this order”
Rules of Engagement

• Courts have found that reverse engineering to understand the ideas embodied in code and to achieve interoperability is fair use
  - Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1992)
  - Sony Computer Entertainment, Inc. v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000)

• Our rules of engagement:
  • Only make copies that are absolutely necessary for reverse engineering
  • Reduce datasheets, binaries, and code into facts, then write code or create maskworks using our own creative expression based off of these facts
  • Do not include any copy/paste code, this includes comments
  • Use a pseudocode language for implementation, to avoid “subconscious plagiarism” of code motifs
Pseudocode language for hardware initializations

```c
#include "scriptic.h"
#include "fernvale-pll.h"

sc_new "set_plls", 1, 0, 0
sc_write16 0, 0, PLL_CTRL_CON2
sc_write16 0, 0, PLL_CTRL_CON3
sc_write16 0, 0, PLL_CTRL_CON0
sc_usleep 1
sc_write16 1, 1, PLL_CTRL_UPLL_CON0
sc_write16 0x1840, 0, PLL_CTRL_EPLL_CON0
sc_write16 0x100, 0x100, PLL_CTRL_EPLL_CON1
sc_write16 1, 0, PLL_CTRL_MDDS_CON0
sc_write16 1, 1, PLL_CTRL_MPLL_CON0
sc_usleep 1
sc_write16 1, 0, PLL_CTRL_EDDS_CON0
sc_write16 1, 1, PLL_CTRL_EPLL_CON0
sc_usleep 1
sc_write32 0x4000, 0x4000, PLL_CTRL_CLK_CONDB
sc_usleep 1
sc_write32 0x8048, 0, PLL_CTRL_CLK_CONDC
/* Run the SPI clock at 104 MHz */
sc_write32 0xd002, 0, PLL_CTRL_CLK_CONDH
sc_write32 0xb6a0, 0, PLL_CTRL_CLK_CONDC

if defined(MT6260)
volatile kal_uint32 i, reg_val, loop_1us;

loop_1us = 13;

if(mode == PLL_MODE_MAUI)
{
    // MCU @ 26Mhz
}
else if( mode == PLL_MODE_USB_META) /* Need to keep USB connection */
{
    // change MCU and bus back to @ 26Mhz
    *PLL_CLK_CONDC = 0xB048; // 0xA001_0108, switch to 26Mhz
    // wait for switch takes effect
    while(*PLL_CLK_CONDC & 0x2);
    *PLL_CLK_CONDC = 0x0048; // 0xA001_0108, bit 15 set to 0 to disable digital frequency divider
}

// enable HW mode TOPSM control and clock CG of PLL control
*PLL_PLL_CON2 = 0x0000; // 0xA0170048, bit 12, 10 and 8 set to 0 to enable TOPSM control
// bit 4, 2 and 0 set to 0 to enable clock CG of PLL control
*PLL_PLL_CON3 = 0x0000; // 0xA017004C, bit 12 set to 0 to enable TOPSM control

// enable delay control
*PLL_PLLTD_CON0= 0x0000; //0x A0170700, bit 0 set to 0 to enable delay control

//wait for 3us for TOPSM and delay (HW) control signal stable
for(i = 0 ; i < loop_1us*3 ; ++i);

//enable and reset UPLL
reg_val = *PLL_UPLL_CON0;
reg_val |= 0x0001;
*PLL_UPLL_CON0 = reg_val; // 0xA0170140, bit 0 set to 1 to enable UPLL and generate reset of UPLL

(continues on for several pages)....
• No circumvention, no DMCA problem
  
  – None of the files or binaries were encrypted or had access controlled by any technological measure
  
  – There's a SHA-1 check, but to us, that doesn't control access to the data; it merely validates its contents
CFAA & Contracts

- All files were downloaded off of Baidu or Google, from publicly accessible servers
  - Origin of files is unknown, and we have no connection to the people who posted the files
- We have no NDA with Mediatek, and the phones ship with no EULA, TOU, or T&C that would waive our right to reverse engineer
Is it Legal?

- We have carefully designed our research to avoid running afoul of the law...but impossible to be 100% sure until we:
  - do it
  - (possibly) get sued
  - Win (if sued)
  - Ironically, if it's not litigated, it's not legal precedent in the US

- Also, we're not a lawyers, so don't take any legal advice from us.
  - But, we think we have the Fair Use right (at least in the US courts) to perform this work, and we're happy to exercise it
Patents

• GSM and ARM patent holders might have some claim, but it's unclear for what and how and against who

– It's a whole other talk to give...
Goals

1) Access the MT6260 as a *microcontroller* (e.g. cost-equivalent upgrade to ATMega328U) – GSM/BT is a tertiary goal

2) Create an open (by Western standards) hardware and software platform around the MT6260

   Develop a legal methodology for pulling IP from the China ecosystem into the Western ecosystem
Picking the Target

- We transitioned to the MT6260 (not the MT6250) to future-proof the work a bit.
- Average chipset lifetime is ~1-2 years, and we figure it'll take us that long to make progress.
- MT6260 has a 364MHz CPU (vs 260MHz)
- The MT6260DA includes 4MiB NV storage on-chip
Audience Poll

• For a $3 chip that includes:
  – Multiple ARM cores
  – 8MiB RAM
  – 4MiB EEPROM
  – Bluetooth
  – GSM
  – battery charger
  – audio codec
  – touchscreen controller, and so forth...

• How many chips are inside?
Initial Sketches

- Original idea was to make it compatible with the Spark Core ecosystem
  - 24-pin DIP SoM + castellations on edge for surface-mountable deployment
  - Couldn't pack enough I/O into this footprint
Actualized Implementation
Moar pr0n – with expansion boards
Design Process

- All footprints and symbols created based on spec tables
  - No copy/paste from reference material
- Schematics and layout based on:
  - Experience
  - Educated guesses
  - Reverse engineering (compare/contrast) of several existing systems
  - Reference materials (e.g. designs published on the Internet and obtained off of download sites) – primarily as sanity checks
Firmware Reversing

- Started by dumping code from an existing phone, the Melrose “MP4 Terminator X”
### Static Analysis

**64Mbit SPINOR**

- Mostly unencrypted, with LZMA-compressed objects

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_0000</td>
<td>media signature “SF_BOOT”</td>
</tr>
<tr>
<td>0x0000_0200</td>
<td>bootloader signature “BRLYT”, “BBBB”</td>
</tr>
<tr>
<td>0x0000_0800</td>
<td>sector header 1 (“MMM.8”)</td>
</tr>
<tr>
<td>0x0000_09BC</td>
<td>reset vector table</td>
</tr>
<tr>
<td>0x0000_0A10</td>
<td>start of ARM32 instructions – stage 1 bootloader?</td>
</tr>
<tr>
<td>0x0000_3400</td>
<td>sector header 2 (“MMM.8”) – stage 2 bootloader?</td>
</tr>
<tr>
<td>0x0000_A518</td>
<td>thunk table of some type</td>
</tr>
<tr>
<td>0x0000_B704</td>
<td>end of code (padding until next sector)</td>
</tr>
<tr>
<td>0x0001_0000</td>
<td>sector header 3( “MMM.8”) – kernel?</td>
</tr>
<tr>
<td>0x0001_0368</td>
<td>jump table + runtime setup (stack, etc.)</td>
</tr>
<tr>
<td>0x0001_0828</td>
<td>ARM thumb code start – possibly also baseband code</td>
</tr>
<tr>
<td>0x0007_2F04</td>
<td>code end</td>
</tr>
<tr>
<td>0x0007_2F05 - 0x0009_F0005</td>
<td>padding “DFFF”</td>
</tr>
<tr>
<td>0x0009_F006</td>
<td>code section begin “Accelerated Technology / ATI / Nucleus PLUS”</td>
</tr>
<tr>
<td>0x000A_2C1A</td>
<td>code section end; pad with zeros</td>
</tr>
<tr>
<td>0x000A_328C</td>
<td>region of compressed/unknown data begin</td>
</tr>
</tbody>
</table>

Identified with binwalk, extracted with dd, decompressed with 7z
Live System Analysis

- Used Tek MDO4104B-6 to analyze timing of RS-232 lines vs. SPI ROM access
  - Identify how much prep work is done by internal ROM vs. extracted ROM image
  - Identify entry points and transitions between bootloader stages
Overall Timing
Decode RS-232 Strings
Decode SPI ROM addresses & data
Some Kind of Verification...

- Modifying putative boot area causes boot to fail

--- Welcome to MTK Bootloader V005 (since 2005) ---

**== Original code ==**

```
F1: 0000 0000
V0: 0000 0000 [0001]
00: 0000 0000
U0: 0000 0001 [0000]
G0: 0002 0000 [0000]
T0: 0000 00C0
Jump to BL

Init Start
Init done, 0x2210992
Jump to ExtBL, 0x3460
```

--- Welcome to MTK Bootloader V005 (since 2005) ---

**== One-byte modification ==**

```
F1: 5004 0000
F8: 380C 0000
F9: 4800 000B
F9: 4800 000B
F9: 4800 000B
F9: 4800 000B
00: 102C 0004
01: 1005 0000
U0: 0000 0001 [0000]
T0: 0000 00C3
Boot failed, reset ...
```
There's a Phone in my Novena...
Enter the Machine

- Romulate to assist with First boot ROM flow reverse engineering
  - Selective MITM between MT6260 and SPINOR
  - Bypass CS line to FPGA to swap in original or emulated ROMs
  - Power/reboot control for CI automation
  - Use Novena + FPGA to memory-map MT6260 boot ROM into Novena's RAM space
- Instantaneous, live experimentation upon MT6260 ROM code!
Finding the Verification

• Determine extent of verification
  – Use Romulator to poke regions & determine extent of hash region

• Determine type of hash
  – Static analysis of ROMs shows constants for SHA-1, so look for a SHA-1 signature
Found it!

• SHA-1 hash appended to intbl region

F1: 0000 0000
V0: 0000 0000 [0001]
00: 0000 0000
U0: 0000 0001 [0000]
G0: 0002 0000 [0000]
T0: 0000 00C0
Jump to BL

Init Start
Init done, 0x17ba72
food toyomama, 0x3460

~~~ Welcome to MTK Bootloader V005 (since 2005) ~~~
**=================================================================**
Dynamic Code Manipulation via radare2

- SPI ROM is now a 64k window available via mmap() on Linux
- Port radare2 to treat 64k mmap() window as an I/O target
  - Include routine to auto-update intbl/extbl hash every time ROM is patched
  - https://github.com/xobs/radare2/tree/fernvale
radare2 example
<table>
<thead>
<tr>
<th>Section</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Keypad Scanner</td>
</tr>
<tr>
<td>3.4</td>
<td>General Purpose Inputs/Outputs</td>
</tr>
<tr>
<td>3.5</td>
<td>General-purpose Timer</td>
</tr>
<tr>
<td>3.6</td>
<td>MCU OSTIMER</td>
</tr>
<tr>
<td>3.7</td>
<td>UART</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0x00000000</td>
<td>PSRAM map, repeated and mirrored at 0x008000000 offsets</td>
</tr>
<tr>
<td>0x01000000</td>
<td>Memory-mapped SPI chip</td>
</tr>
<tr>
<td>0x70000000</td>
<td>On-chip SRAM (maybe cache?)</td>
</tr>
<tr>
<td>0x80000000</td>
<td>Config block (chip version, etc.)</td>
</tr>
<tr>
<td>0x82200000</td>
<td>Config block (mirror?)</td>
</tr>
<tr>
<td>0xa0000000</td>
<td>Config block (mirror?)</td>
</tr>
<tr>
<td>0xa0001000</td>
<td>(?)SPI mode?</td>
</tr>
<tr>
<td>0xa0020000</td>
<td>GPIO control block</td>
</tr>
<tr>
<td>0xa0030000</td>
<td>WDT block</td>
</tr>
<tr>
<td>0xa0030800</td>
<td>(?)WDT register (?)</td>
</tr>
<tr>
<td>0xa0060000</td>
<td>?? Possible IRQs at 0xa0060200 ???</td>
</tr>
<tr>
<td>0xa0070000</td>
<td>Empty (all zeroes)</td>
</tr>
<tr>
<td>0xa0080000</td>
<td>UART1 block</td>
</tr>
<tr>
<td>0xa0090000</td>
<td>UART2 block</td>
</tr>
</tbody>
</table>

*Memory Map*
Doing what we can

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Mnemonic</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>THR</td>
<td>THR</td>
<td>TX holding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A write-on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Modified</td>
</tr>
</tbody>
</table>
Fernly

• Command line environment
  – Contains peek, poke, hexdump
  – One-off programs to search for patterns

• Must fit within extbl
  – That's okay, it's relatively small
First up: UART

• Same UART as in many other Mediatek products
• Part of reference manual we had
• No IRQ required
• `putchar()` and `getchar()`
Next up: GPIO

- Also very easy
- Also part of reference manual we had
- No IRQ required
- Not very useful at this point
Next up: GPT

- Necessary for periodic tick
- Also in reference manual
The (One) IRQ is Standardized on ARM

<table>
<thead>
<tr>
<th>Exception</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>0</td>
</tr>
<tr>
<td>Undefined Instruction</td>
<td>4</td>
</tr>
<tr>
<td>SWI</td>
<td>8</td>
</tr>
<tr>
<td>Prefetch Abort</td>
<td>12</td>
</tr>
<tr>
<td>Data Abort</td>
<td>16</td>
</tr>
<tr>
<td>Reserved</td>
<td>20</td>
</tr>
<tr>
<td><strong>IRQ</strong></td>
<td>24</td>
</tr>
<tr>
<td>FIQ</td>
<td>28</td>
</tr>
</tbody>
</table>
## The interrupt problem

### MT6205B

**CIRQ+0014h**

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IRQF</td>
<td>IRQE</td>
<td>IRQD</td>
<td>IRQC</td>
<td>IRQB</td>
<td>IRQA</td>
<td>IRQ9</td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**CIRQ+0038h**

<table>
<thead>
<tr>
<th>Bit</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IRQ1F</td>
<td>IRQ1E</td>
<td>IRQ1D</td>
<td>IRQ1C</td>
<td>IRQ1B</td>
<td>IRQ1A</td>
<td>IRQ19</td>
<td>IRQ18</td>
<td>IRQ17</td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### MT6235

**CIRQ+0014h**

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IRQF</td>
<td>IRQE</td>
<td>IRQD</td>
<td>IRQC</td>
<td>IRQB</td>
<td>IRQA</td>
<td>IRQ9</td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**CIRQ+0038h**

<table>
<thead>
<tr>
<th>Bit</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>IRQ1F</td>
<td>IRQ1E</td>
<td>IRQ1D</td>
<td>IRQ1C</td>
<td>IRQ1B</td>
<td>IRQ1A</td>
<td>IRQ19</td>
<td>IRQ18</td>
<td>IRQ17</td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Name</td>
<td>IRQF</td>
<td>IRQE</td>
<td>IRQD</td>
<td>IRQC</td>
<td>IRQB</td>
<td>IRQA</td>
<td>IRQ9</td>
<td>IRQ8</td>
<td>IRQ7</td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Try to analyze what we have

- Locate ROM, dump it
- Analyze SPI ROM with IDA
- Find other ROMs online and analyze them
- Look at manuals for similar chips
Found a function

- void func(int, (void *)(), char *)
  - func(30, isr30, “SPI”)
  - func(18, isr18, “GPT Handler”)
• Remember that 7.5GiB source archive?
• Customized to the MT6260
• Source of an OS:
  - IRQ module exists in source form
    • cirq/inc/intrCtrl_MT6260.h
  - Complete memory map definition in header files
    • regbase/inc/reg_base_mt6260.h
  - Not as good as a datasheet, but it will do!
IRQ Problem: Solved

- We know how to unmask IRQs
- We know how to acknowledge IRQs
- For some reason, IRQs are off-by-5
  - func(30, ispi, “SPI”) → IRQ35
  - func(18, igpt, “GPT Handler”) → IRQ23
NuttX port

- Used by Osmocom
- Multitasking support
  - Thanks to GPT and IRQ
- No memory protection
  - ARM7EJ has no MMU
  - Only example of ARMv5 on ARM7
- At this point, Goal #1 is basically reached
  - Many features yet to be implemented
    - LCD
    - SPI
    - Audio
There's a Phone in my Novena...
Getting code onto Fernvale
Boot - Mediatek

ROM -> intbl -> extbl

1bl -> 2bl

factory -> memtest

OS
fernvale-usb-loader

- Open-licensed
- Writes to /dev/ttyUSB0

Diagram:

1. ROM
2. usbdl
3. fernly
4. NuttX

Flow from: ROM -> usbdl -> fernly -> NuttX
Towards an “open” boot

- Closed Mediatek – intbl and 1bl
  - Set up clocks, PSRAM
- No reference manuals
- How can we set up the chip at boot?
Scriptic

- Simple command language
  - Very similar problem to SoC boot scripts
- Can distill facts down into scripts
- Scripts are not Turing-complete
  - Can call C functions
  - E.g. PSRAM calibration
- Implemented as assembler macros
Scriptic - Commands

#define sc_end_cmd 0
#define sc_read32_cmd 1
#define sc_write32_cmd 2
#define sc_read16_cmd 3
#define sc_write16_cmd 4
#define sc_call_cmd 5
#define sc_usleep_cmd 6
/* Remap EMI to 0x10000000, and SPI to 0x00000000 */
sc_write32 2, 0, EMI_CTRL_REMAP

/* Memory configuration */
sc_write16 1, 0, 0x1ffffffffe
sc_usleep 50
sc_write16 0x2b13, 0, 0x1ffffffffe
sc_usleep 50
/* Calibrate DQ in delay */
sc_call calibrate_psram, 0

sc_write32 0x300f0000, 0, EMI_CTRL_DLLV
sc_read32 0x80, 0x80, EMI_CTRL_DLLV
sc_write32 0x700f0000, 0, EMI_CTRL_DLLV
sc_read32 0x80, 0x00, EMI_CTRL_DLLV
sc_write32 0x100f0000, 0, EMI_CTRL_DLLV
/* Enable high-impedence for GPIO mode */

sc_write16 0, \n   GPIO_CTRL_PULL_CTRL1_IO66 | \ 
   GPIO_CTRL_PULL_CTRL1_IO67 | \ 
   GPIO_CTRL_PULL_CTRL1_IO72, \ 
   GPIO_CTRL_RESEN1_R0

sc_write16 0, \n   GPIO_CTRL_PULL_CTRL1_IO66 | \ 
   GPIO_CTRL_PULL_CTRL1_IO67 | \ 
   GPIO_CTRL_PULL_CTRL1_IO72, \ 
   GPIO_CTRL_RESEN1_R1
Wrap-Up

- Draft process for translating “China IP” into “Western IP”
  - Obtain documentation via public download (common practice in China)
  - Work within fair-use framework
  - Extract facts via scriptic framework to prevent subconscious plagiarism
Open Platform Compliant to Western IP Standards

- Fernvale
  - 3-board system, consisting of mainboard, expansion, and AFE
  - Schematics and layout licensed CC-BY-SA 3.0 + Apache for patents
  - Custom bootloader and flashing tool under BSD license
  - Clang + GCC toolchain (BSD + GPL licensed)
  - Runs NuttX (BSD licensed)

- Interested in hardware? Come see us, we have a few samples to give to qualified developers
Special Thanks

• Shout out to .mudge for enabling this research!
Q&A

Thanks for your attention!

@xobs   @bunniestudios