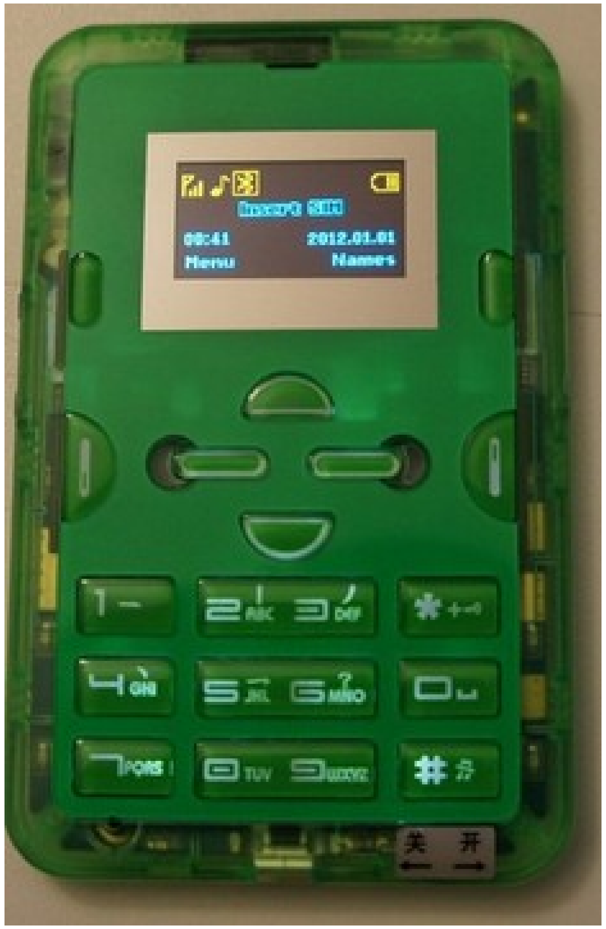
Fernvale:

A Reverse Engineered MT6260 Dev Platform

bunnie & xobs
@bunniestudios & @xobs

31c3

Provocation



= $12

Provocation

Chinese low-end hardware platform

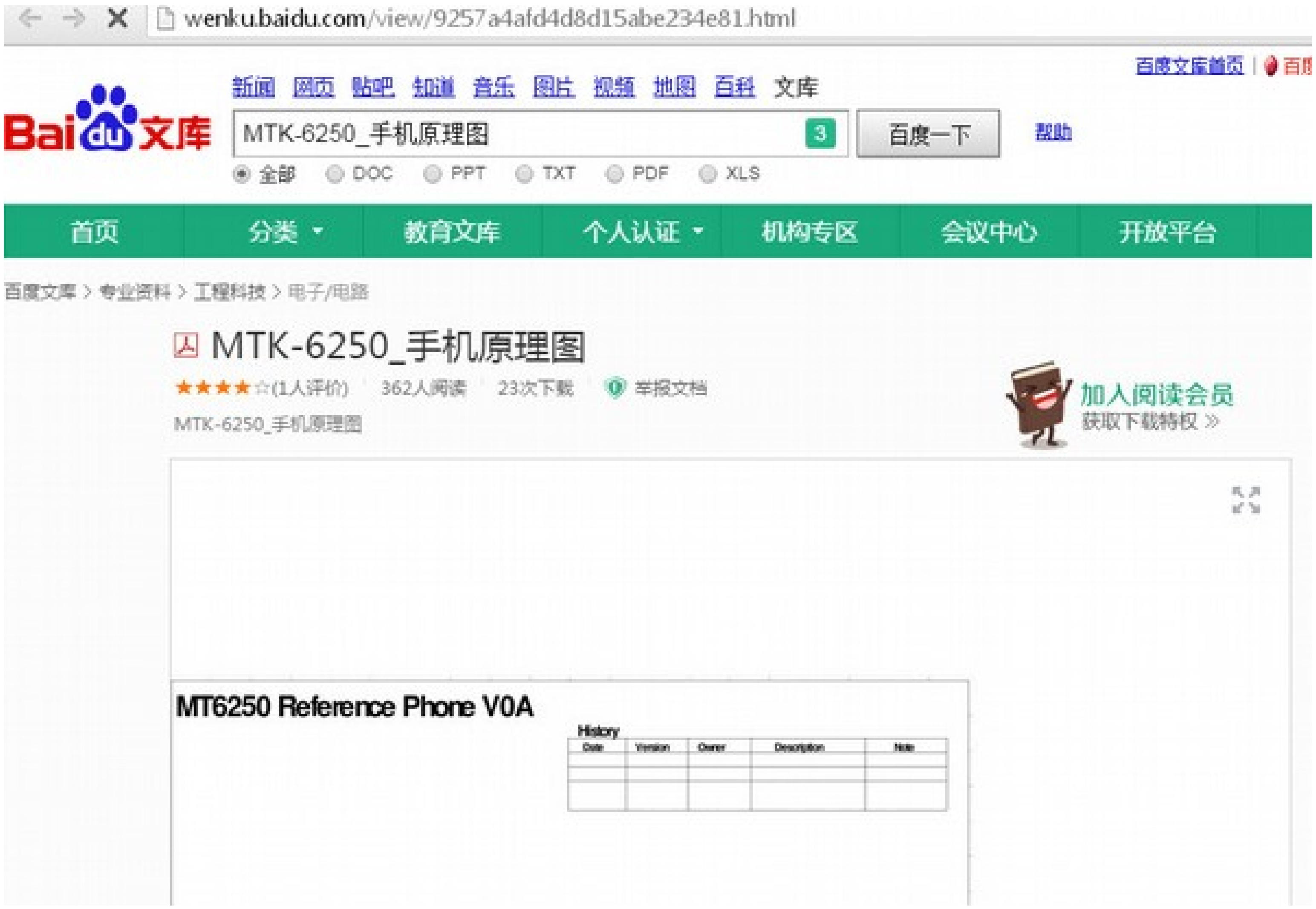
- 260 MHz 32-bit CPU, 8MiB RAM
- Quad-band GSM
- Bluetooth
- OLED display
- MP3 player
- Li-Poly battery
- $12 qty 1

Western low-end hardware platform

- 16 MHz 8-bit CPU, 2.5k RAM
- USB serial interface
- Voltage regulator
- $29 qty 1

Image: Arduino.cc

I Can Has Documentation?


wenku.baidu.com/view/9257a4afd4d8d15abe234e81.html

百度文库首页 | 百

新闻 网页 贴吧 知道 音乐 图片 视频 地图 百科 文库

MTK-6250_手机原理图 3 百度一下 帮助

[✓ 全部] DOC PPT TXT PDF XLS

首页 分类 教育文库 个人认证 机构专区 会议中心 开放平台

百度文库 > 专业资料 > 工程科技 > 电子/电器

MTK-6250_手机原理图

★★★★☆(1人评价) 362人阅读 23次下载 举报文档

MTK-6250_手机原理图

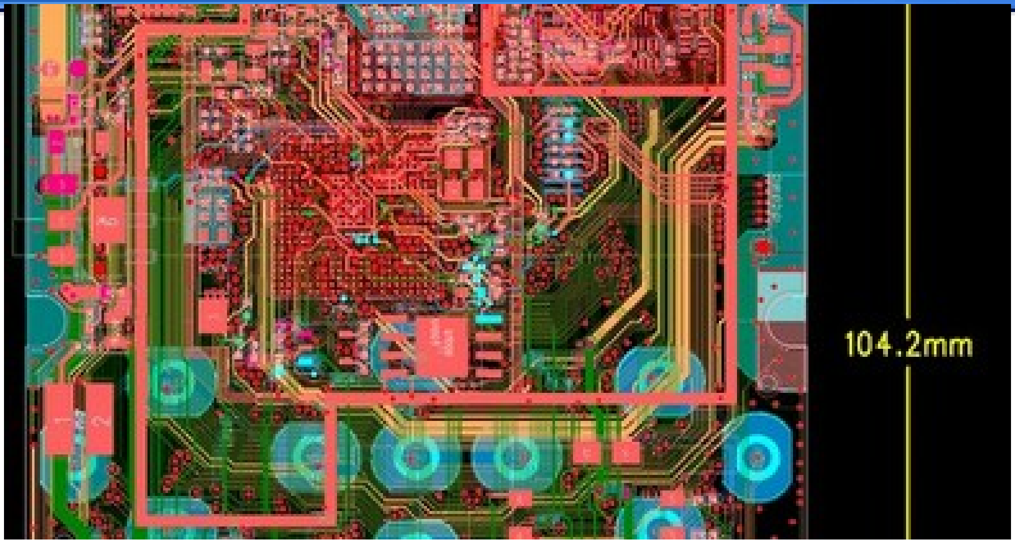
加入阅读会员
获取下载特权 »

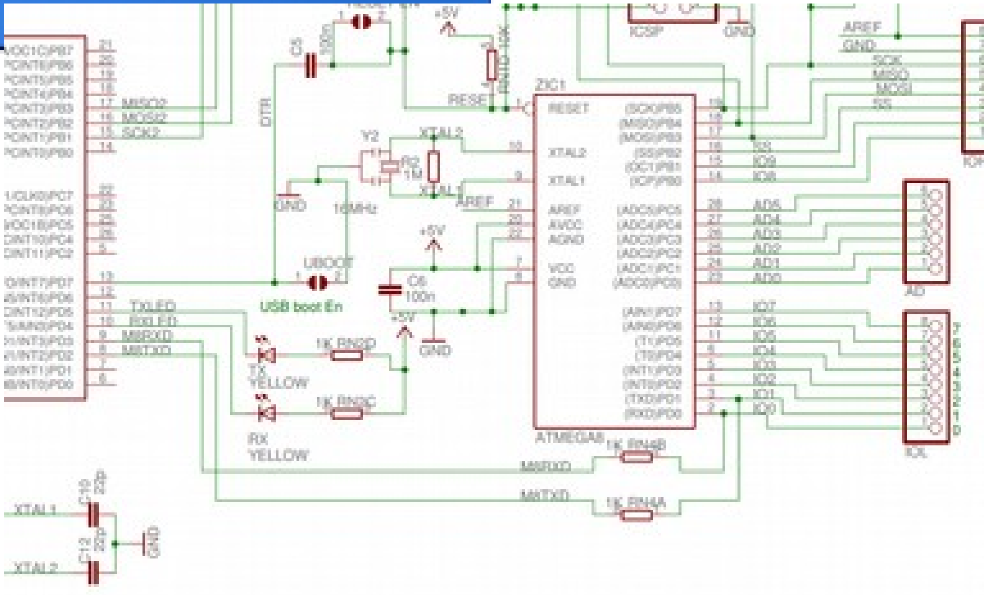
MT6250 Reference Phone V0A

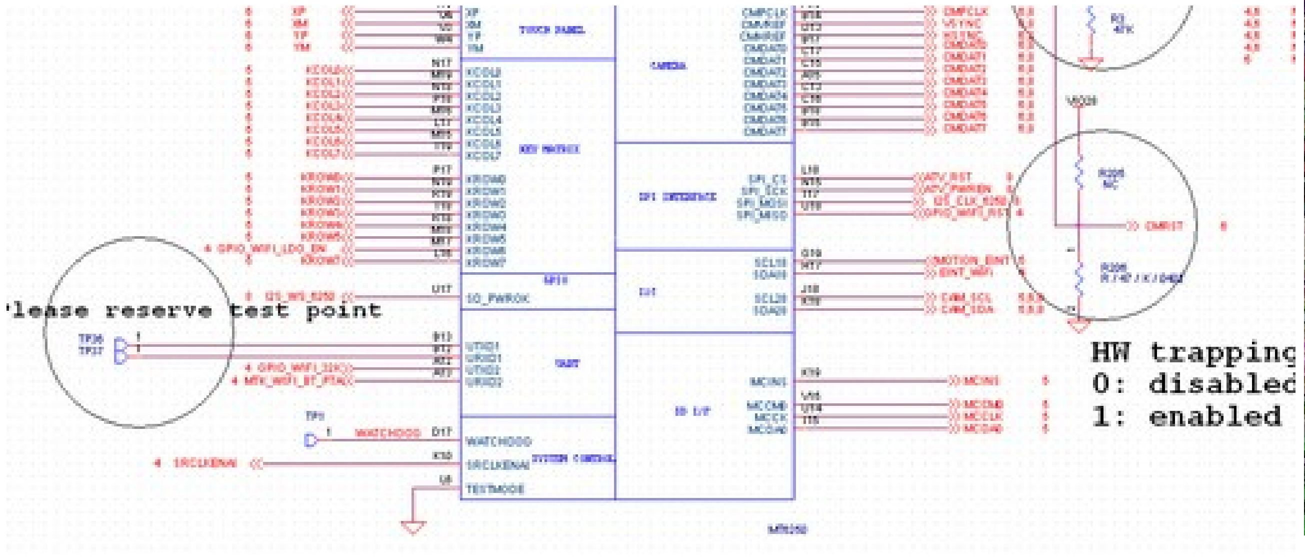
History

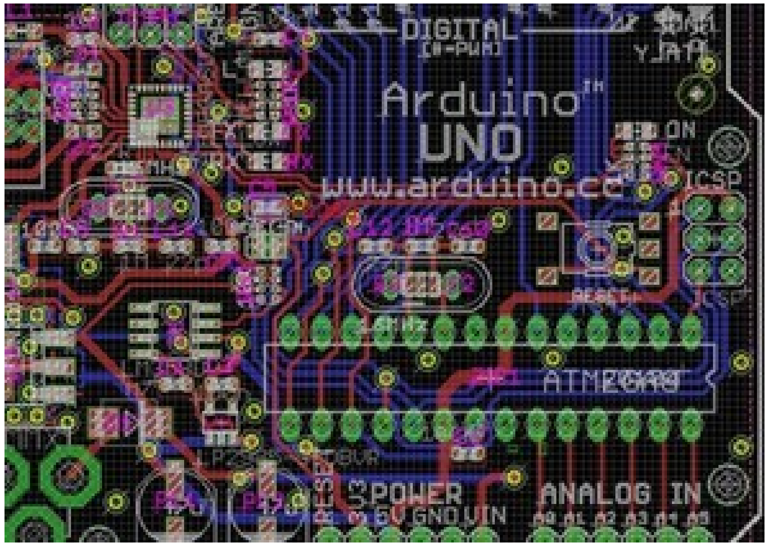
Date	Version	Owner	Description	Note

Downloadable source files...



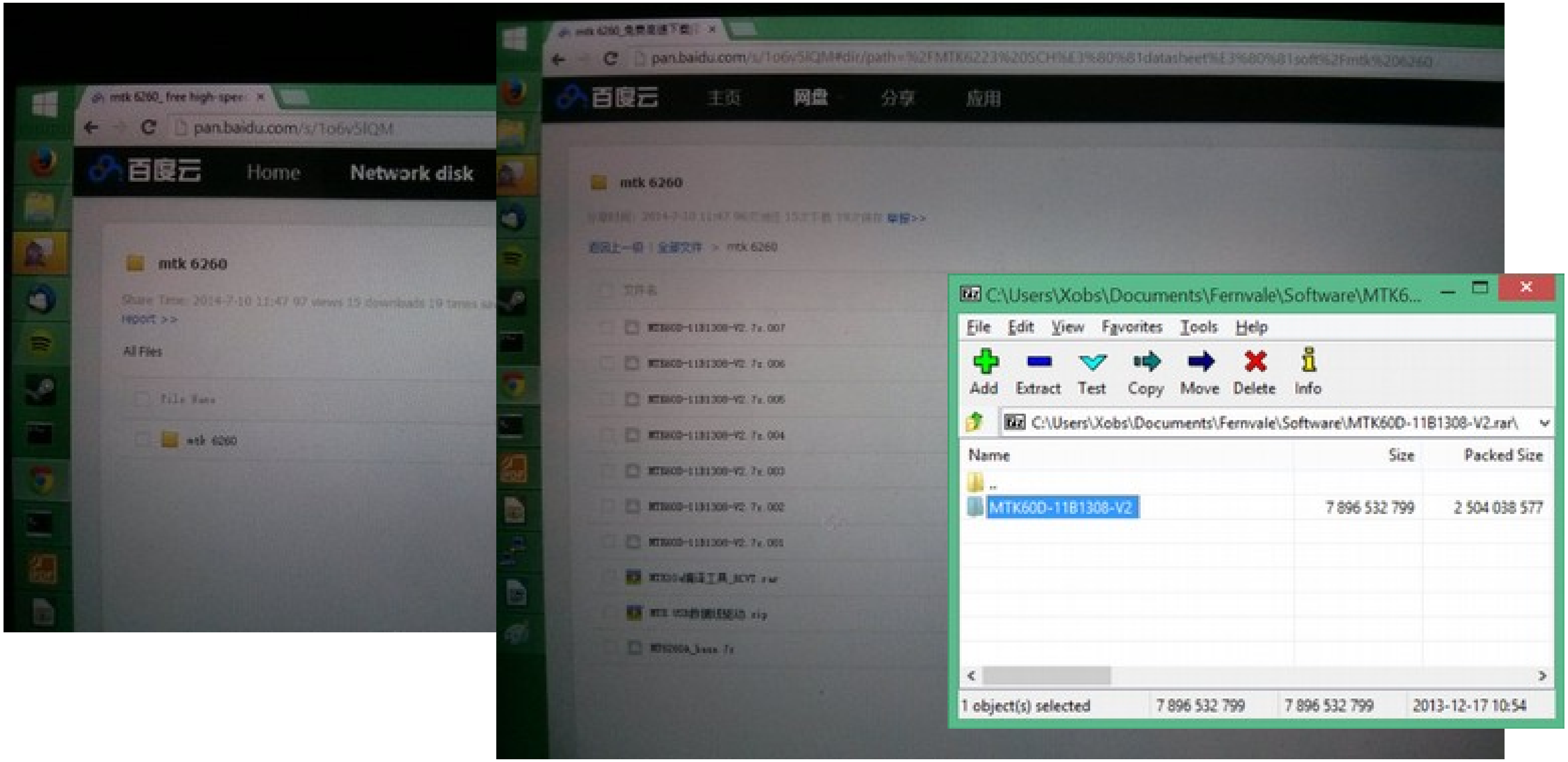






Phone OS Source...

- “MKT11B” is a ~7.5GiB source archive, downloadable from Baidu



pan.baidu.com/s/1o6v5IQM

mtk 6260

All Files

C:\Users\Xobs\Documents\Fernvale\Software\MTK6...

File Edit View Favorites Tools Help

Add Extract Test Copy Move Delete Info

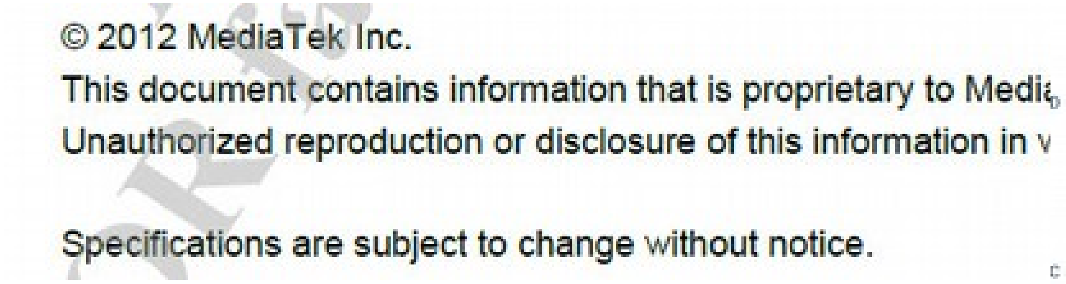
C:\Users\Xobs\Documents\Fernvale\Software\MTK60D-11B1308-V2.rar\

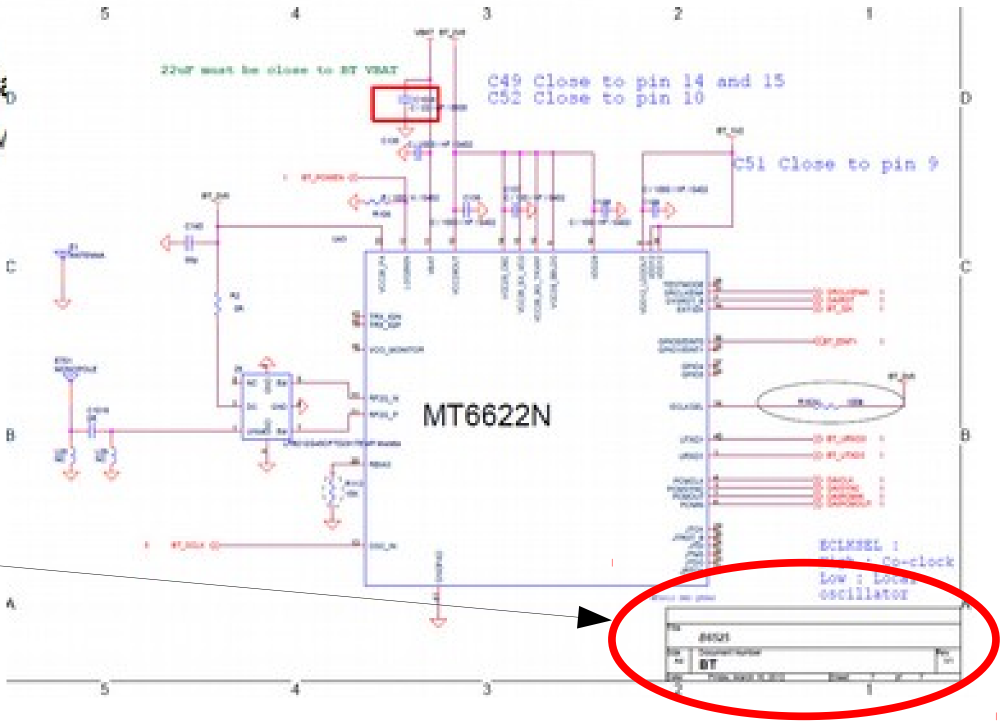
Name	Size	Packed Size
..		
MTK60D-11B1308-V2	7 896 532 799	2 504 038 577

1 object(s) selected 7 896 532 799 7 896 532 799 2013-12-17 10:54

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





No copyright notice anywhere

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

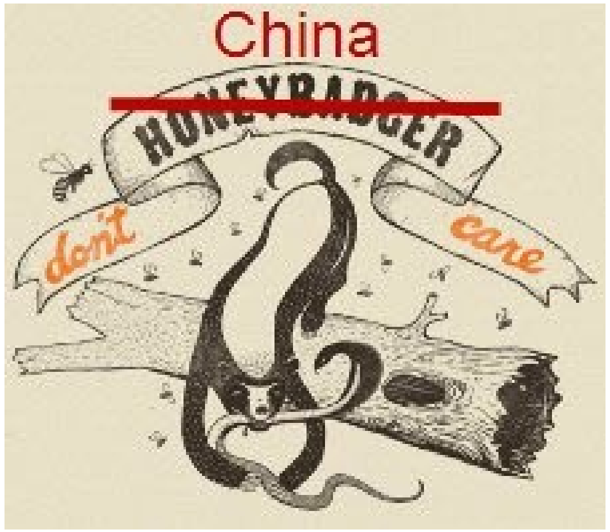


Image: icebergtees.com

Fruits of Permissive IP Environment

Image: Rachel Kalmar

Image: Halfdan

The West *Does* Care

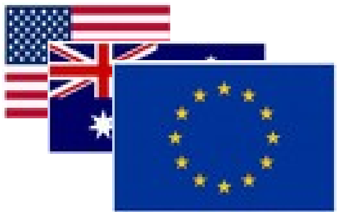
- Can't build a western business on “stolen” IP
  - So ask Mediatek for a license?
    - Either no response or
    - (For example) $250k pre-payment on order volume for access to docs
- Not practical for individuals & startups!

Clip art: openclipart.org
Egg Minder: quirky.com
Isuper tank: 247deals.com.au

Weltschmerz

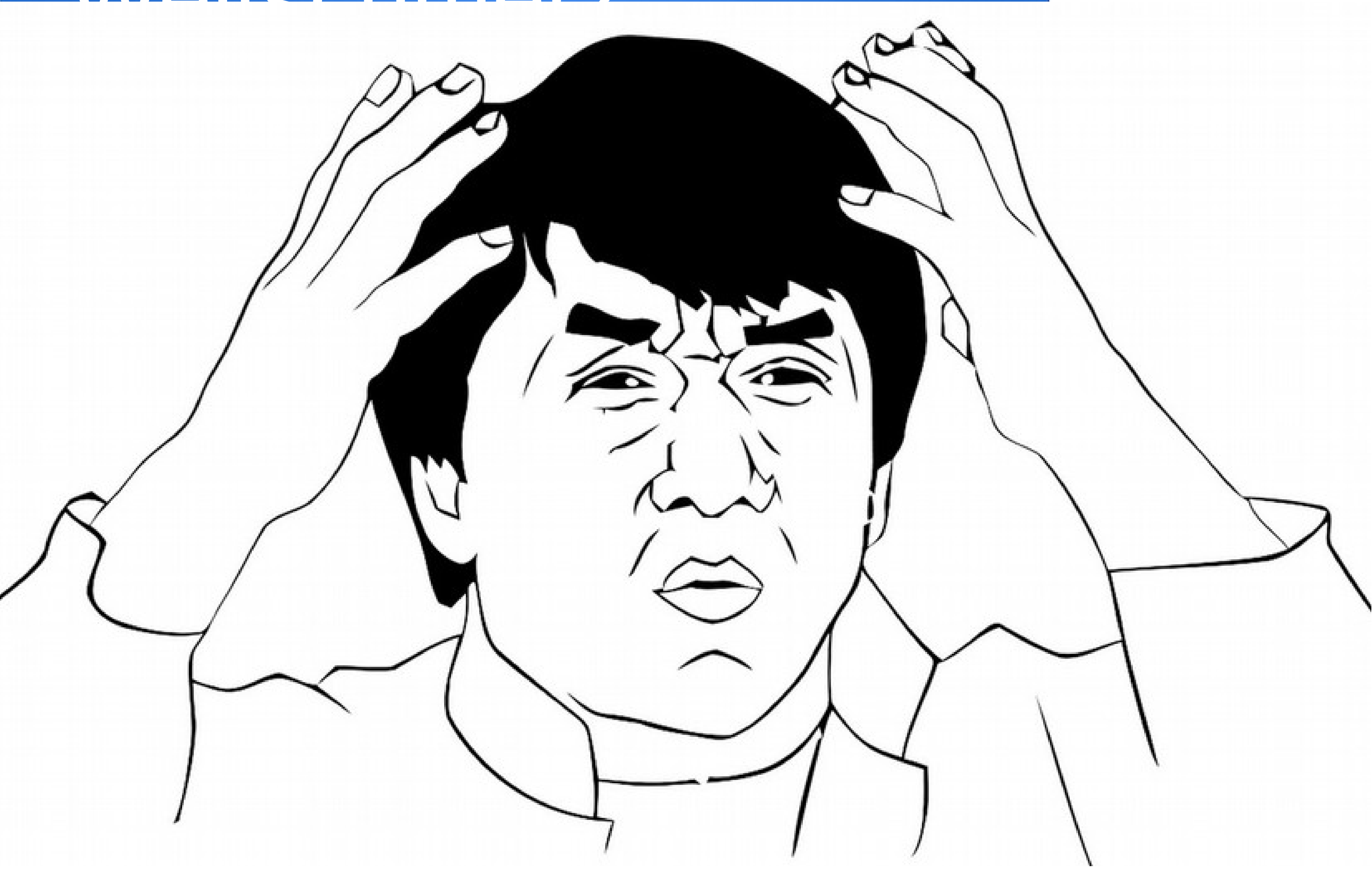
- So you're saying China startups get to make whole phones... and Western startups get to make just phone accessories, and that's a Good Thing?

vs.



Weltschmerz

Who own copyright to rage faces?



Can We Hack the System?

Can We Hack the System?

Who own copyright to rage faces?

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

Scriptic

- Pseudocode language for hardware initializations

Original source code

```
#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 = 0x8048; // 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)....

Manually extract facts

Scriptic pseudocode

```
#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_write16 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
  sc_end
```

DMCA

- 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 MT62**60** (not the MT62**50**) 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?

X-Ray

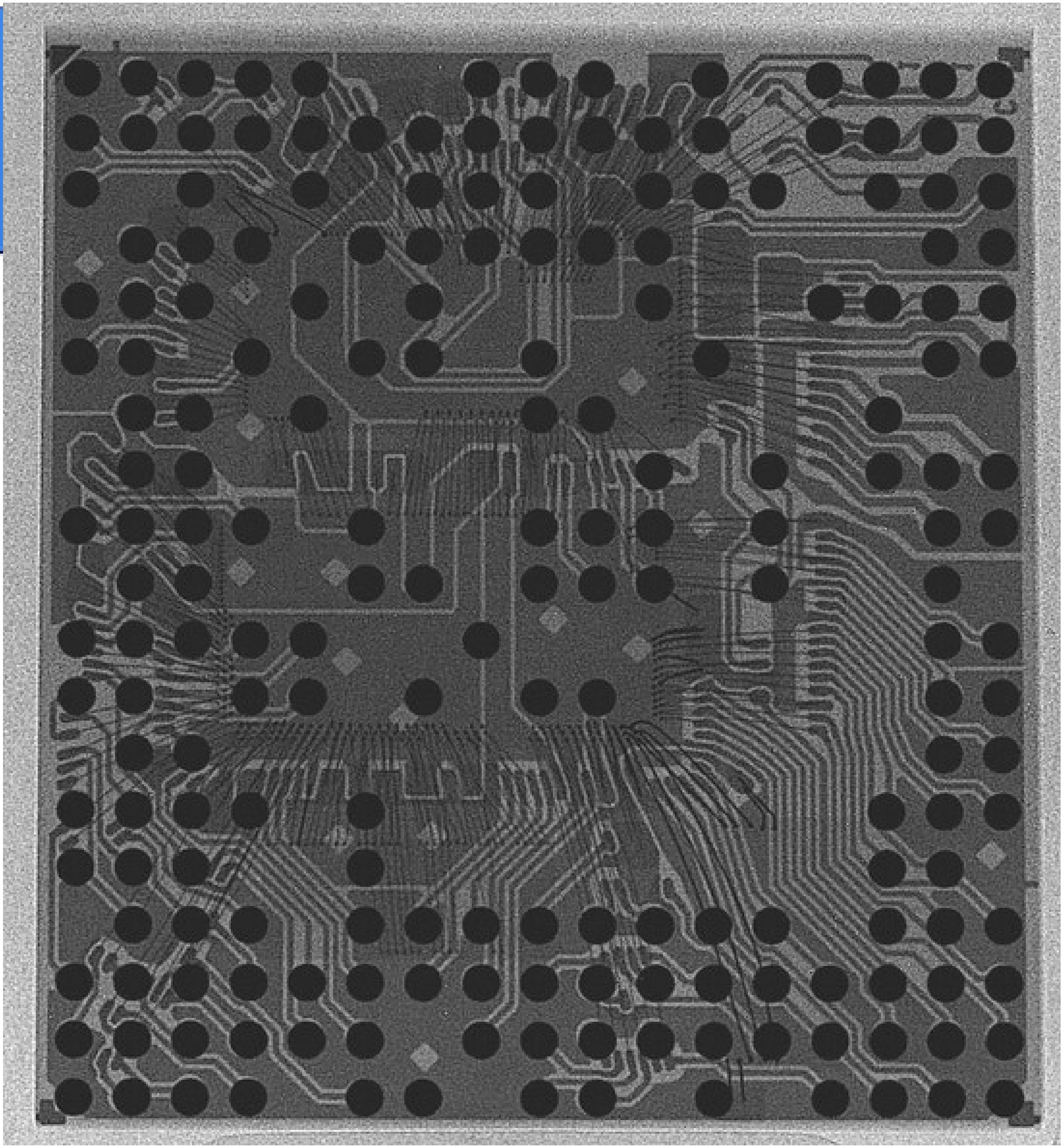
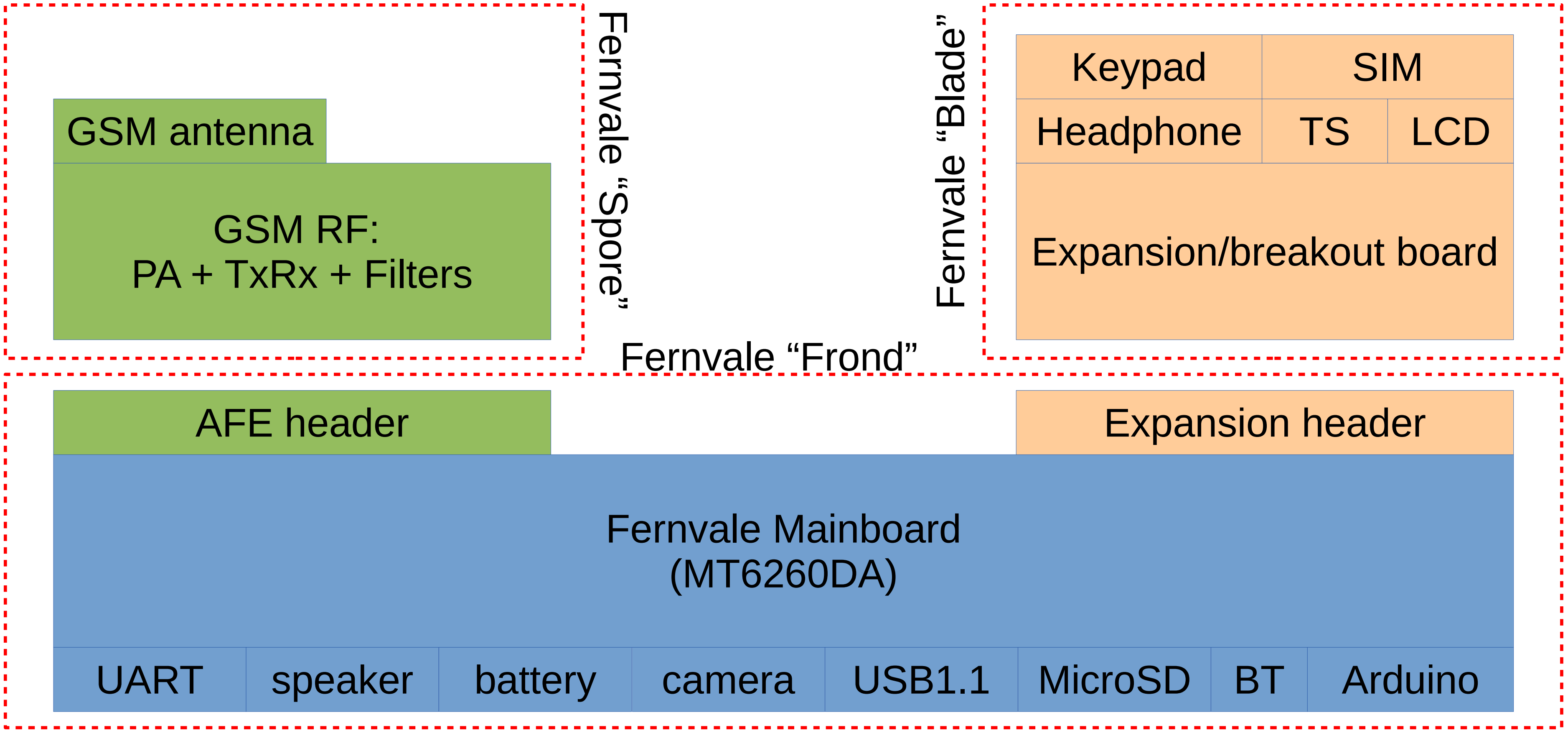


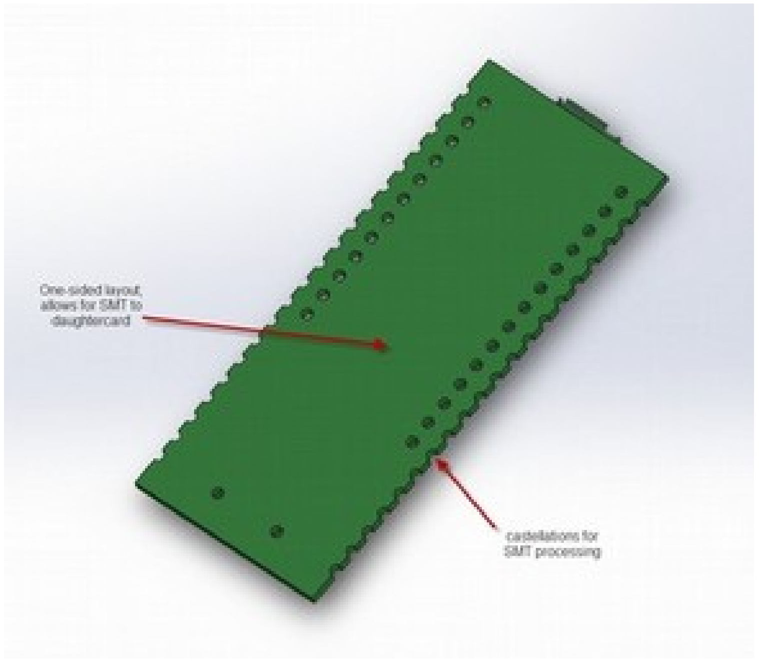
Image credit: Nadya Peek

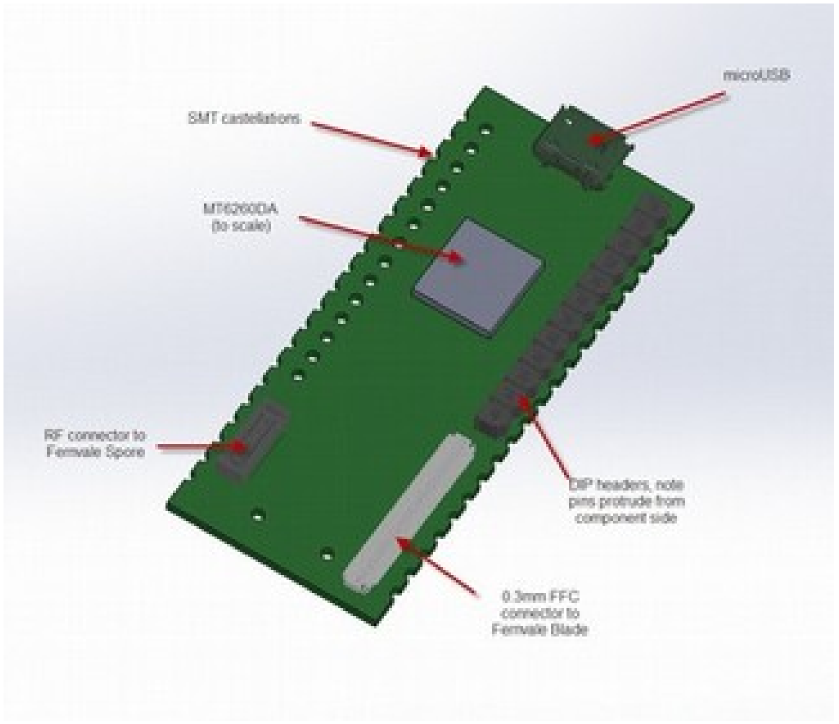
Hardware System Diagram



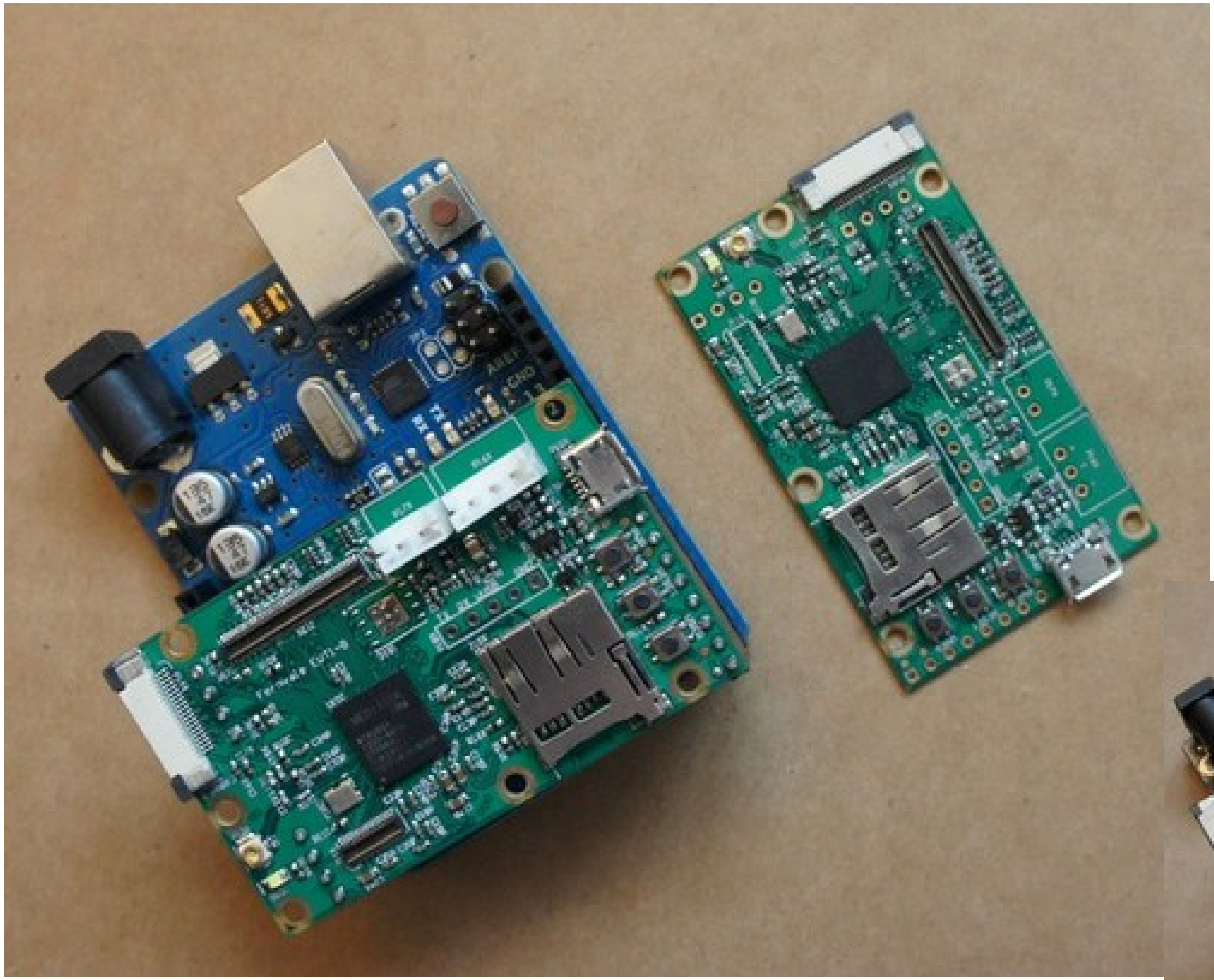
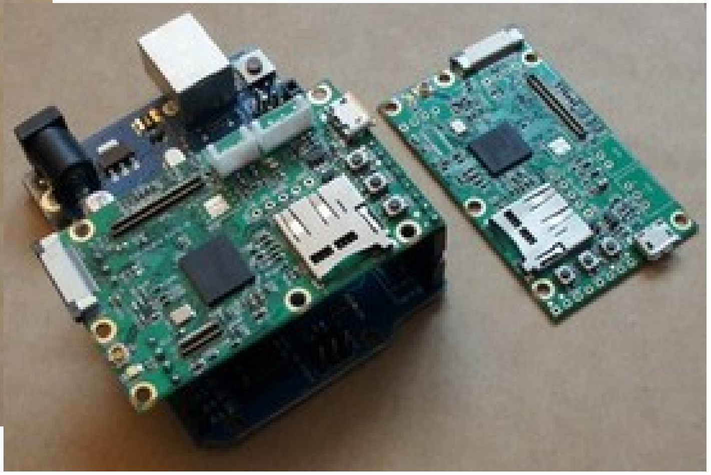
Initial Sketches

- 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

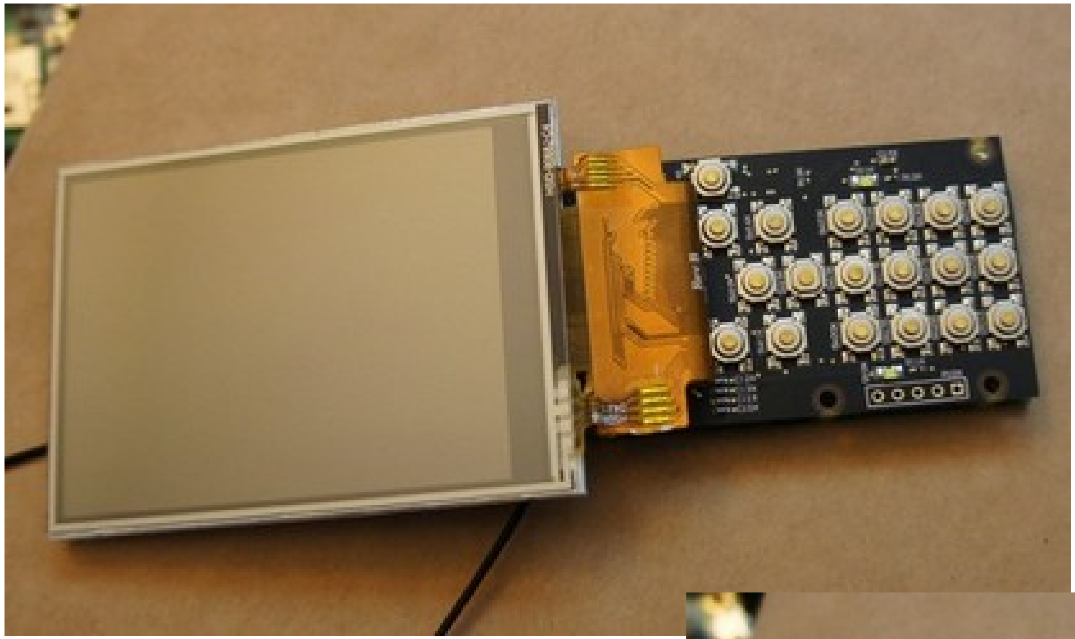
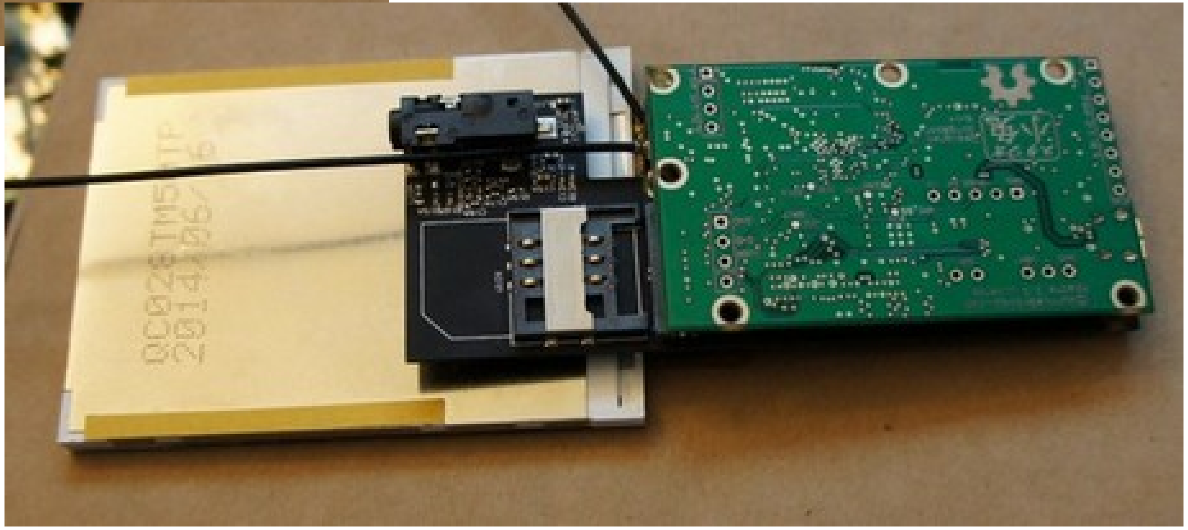

One-sided layout, allows for SMT to daughtercard
castellations for SMT processing


microUSB
SMT castellations
MT6260DA (to scale)
RF connector to Fernvale Spore
DIP headers, note pins protrude from component side
0.3mm FFC connector to Fernvale Blade

Actualized Implementation

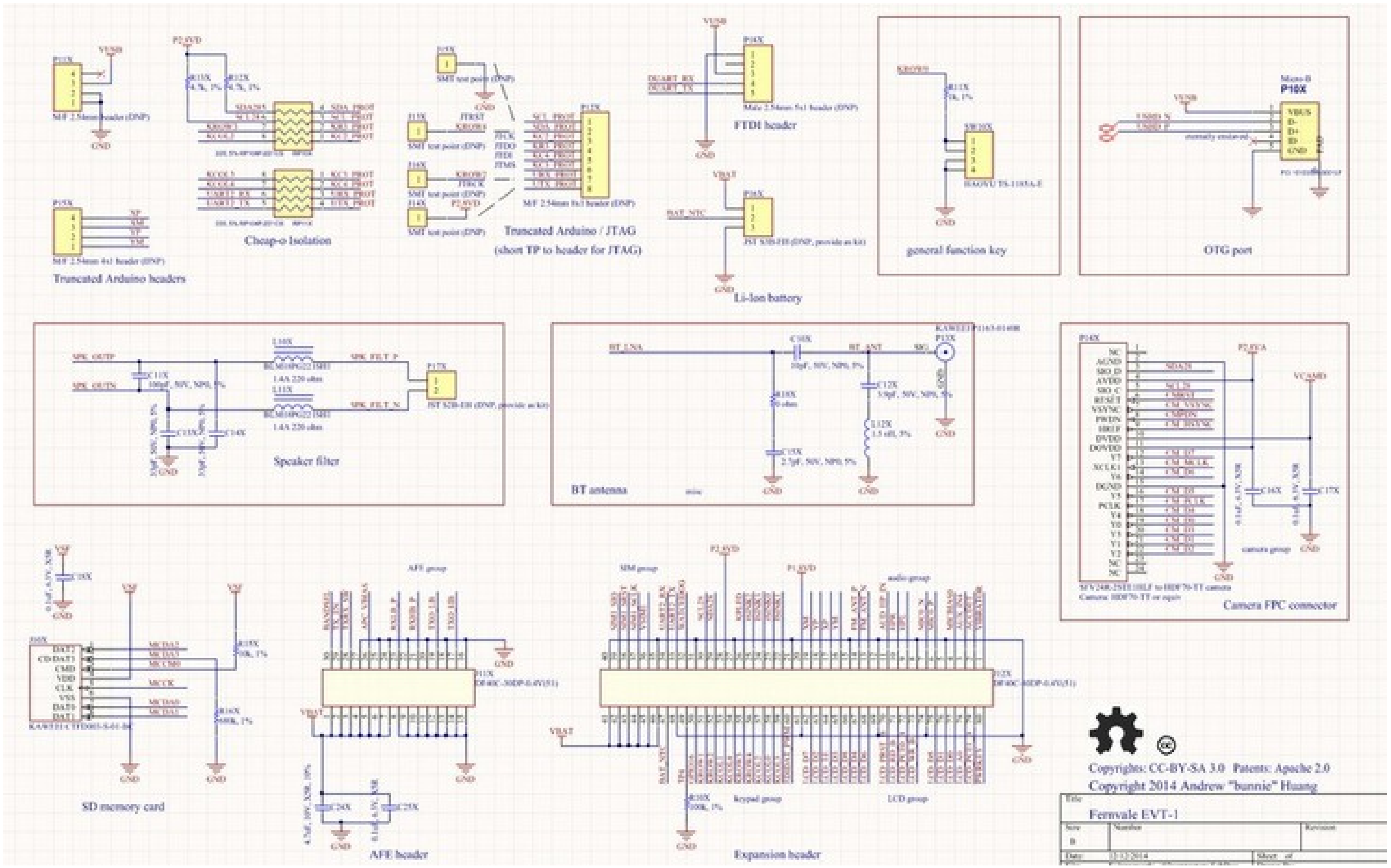
Moar pr0n – with expansion boards

Design Process

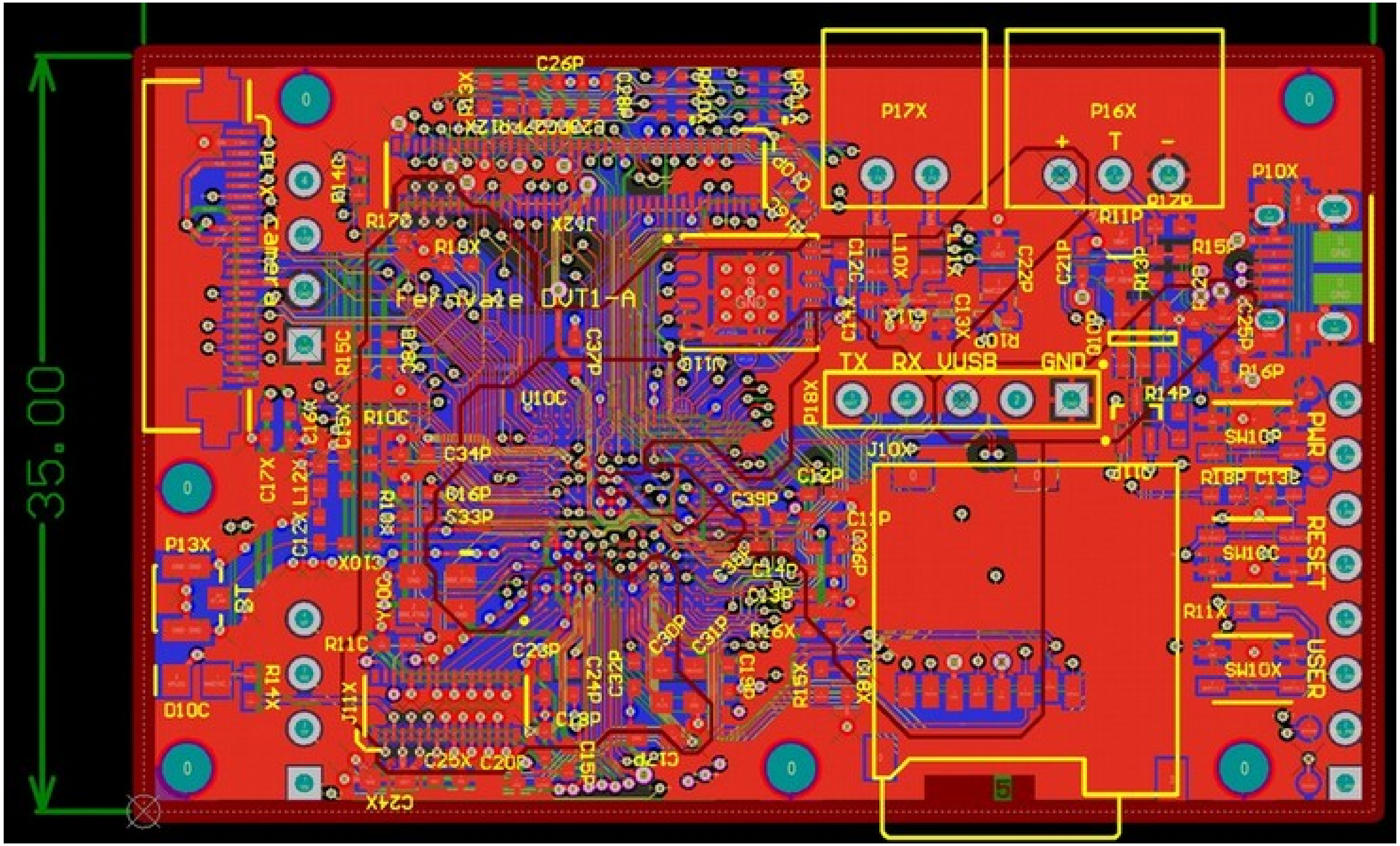
- All footprints and symbols created based on spec tables
  - No copy/paste from reference material
- Schematics and layout based on:
  - Experience
  - Educated guesses
  - Reverse engineering (compare/contrast) of several existing systems
  - Reference materials (e.g. designs published on the Internet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

Schematics



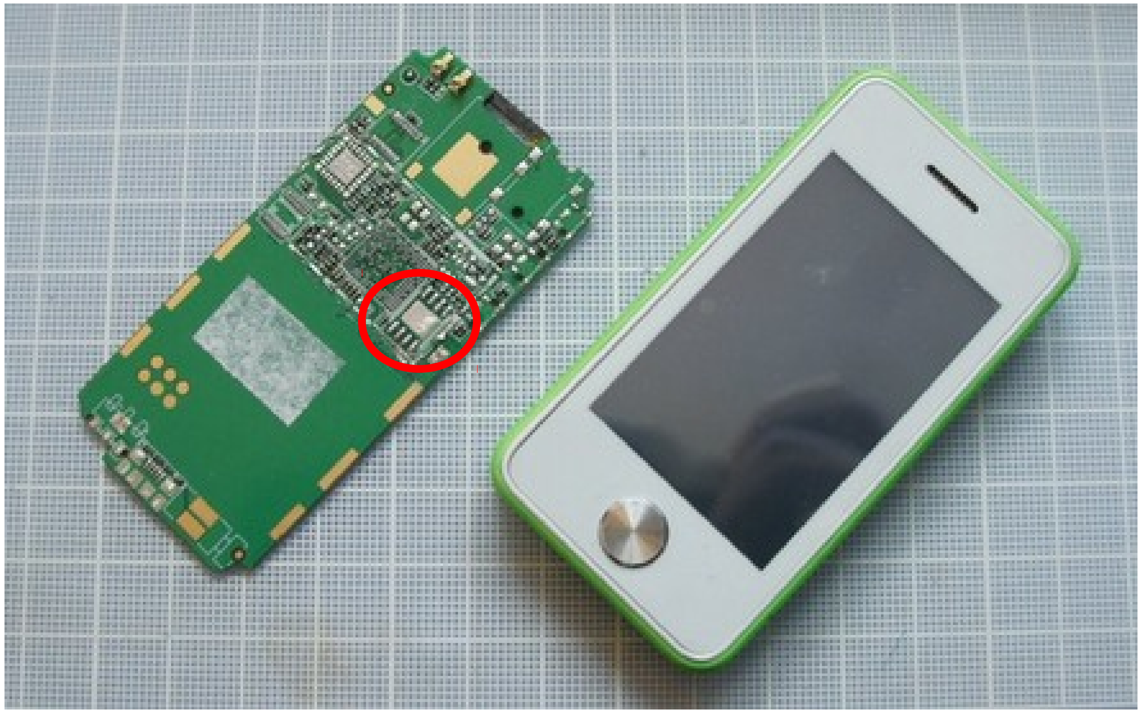
Truncated Arduino headers
Cheap-o Isolation
Truncated Arduino / JTAG
(short TP to header for JTAG)
FTDI header
Li-Ion battery
general function key
OTG port
Speaker filter
BT antenna
Camera FPC connector
SD memory card
AFE header
Expansion header
Copyrights: CC-BY-SA 3.0 Patents: Apache 2.0
Copyright 2014 Andrew "bunnie" Huang
Fernvale EVT-1

Layout


35.00
P17X
P16X
+
T
-
P10X
TX
RX
VUSB
GND
PWR
RESET
USER
Camera
Ferdivate DVT1-A
U10C
P13X
D10C
BT
SW10P
SW10C
SW10X
J10X
P18X

Firmware Reversing

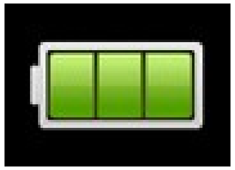
- Started by dumping code from an existing phone, the Melrose “MP4 Terminator X”

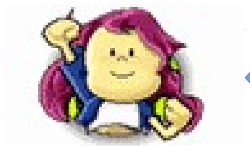


Static Analysis

- 64Mbit SPINOR
  - Mostly unencrypted, with LZMA-compressed objects

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



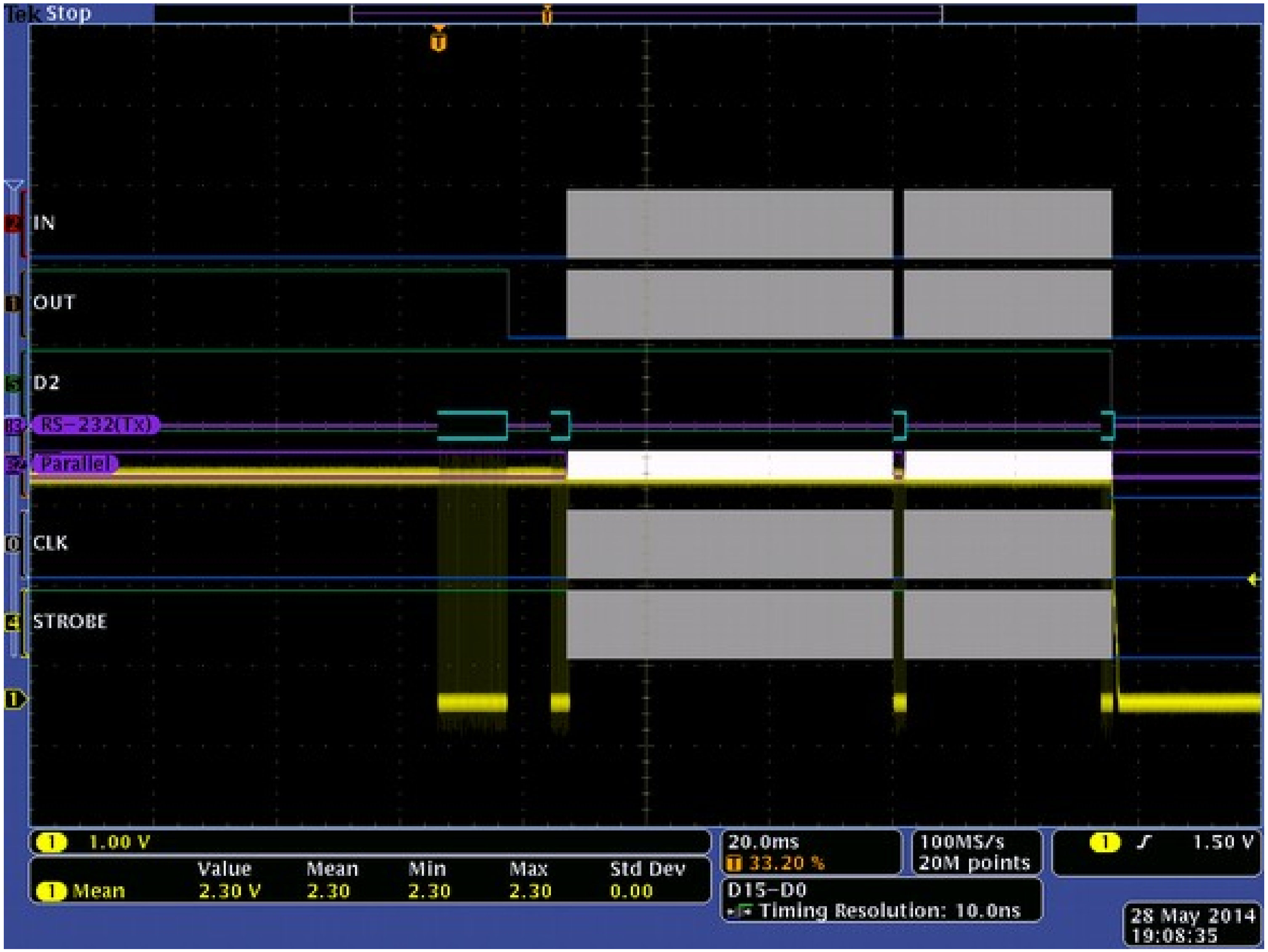


Identified with binwalk, extracted with dd, decompressed with 7z

Live System Analysis

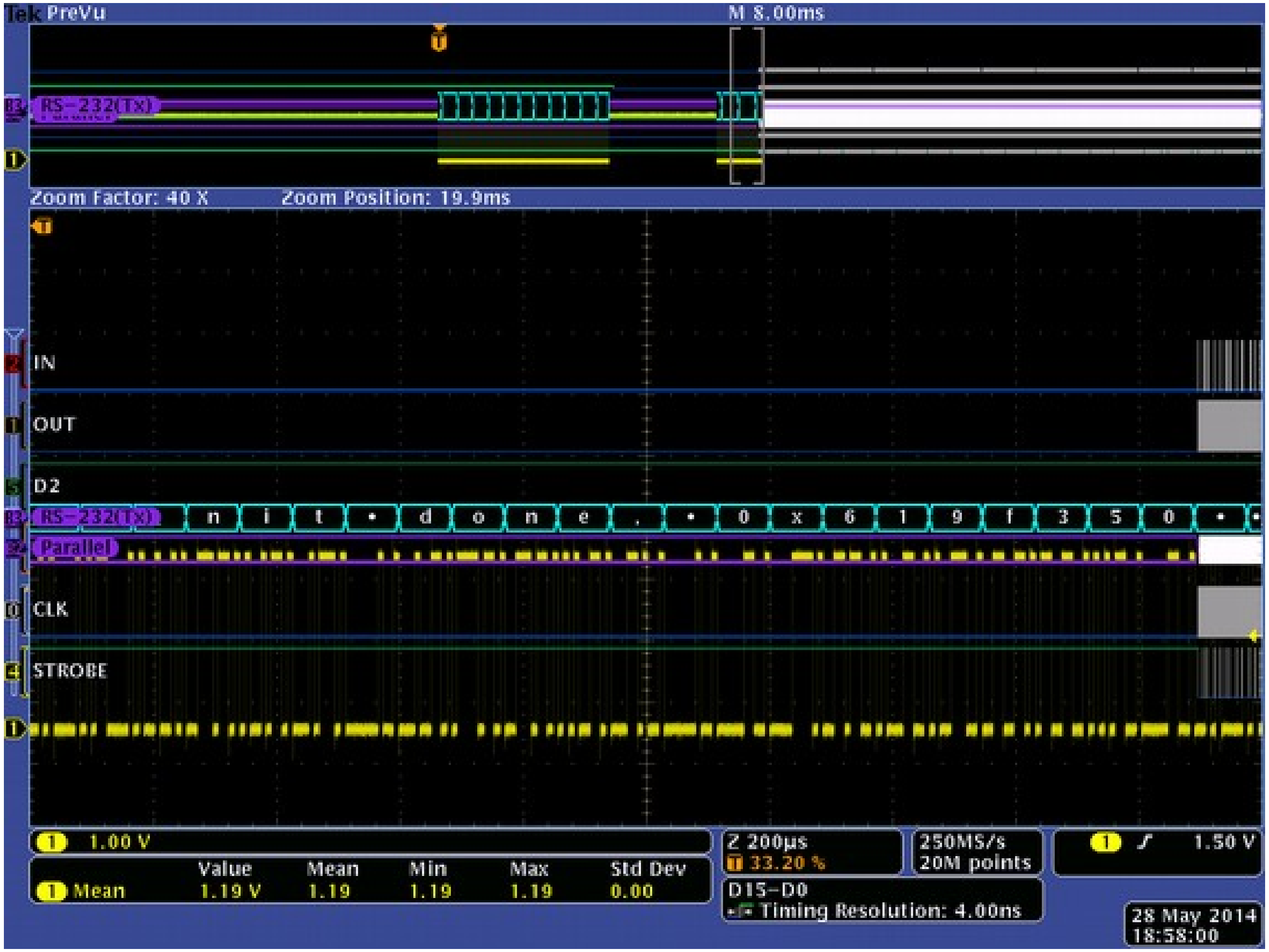
- Used Tek MDO4104B-6 to analyze timing of RS-232 lines vs. SPI ROM access
  - Identify how much prep work is done by internal ROM vs. extracted ROM image
  - Identify entry points and transitions between bootloader stages

Overall Timing



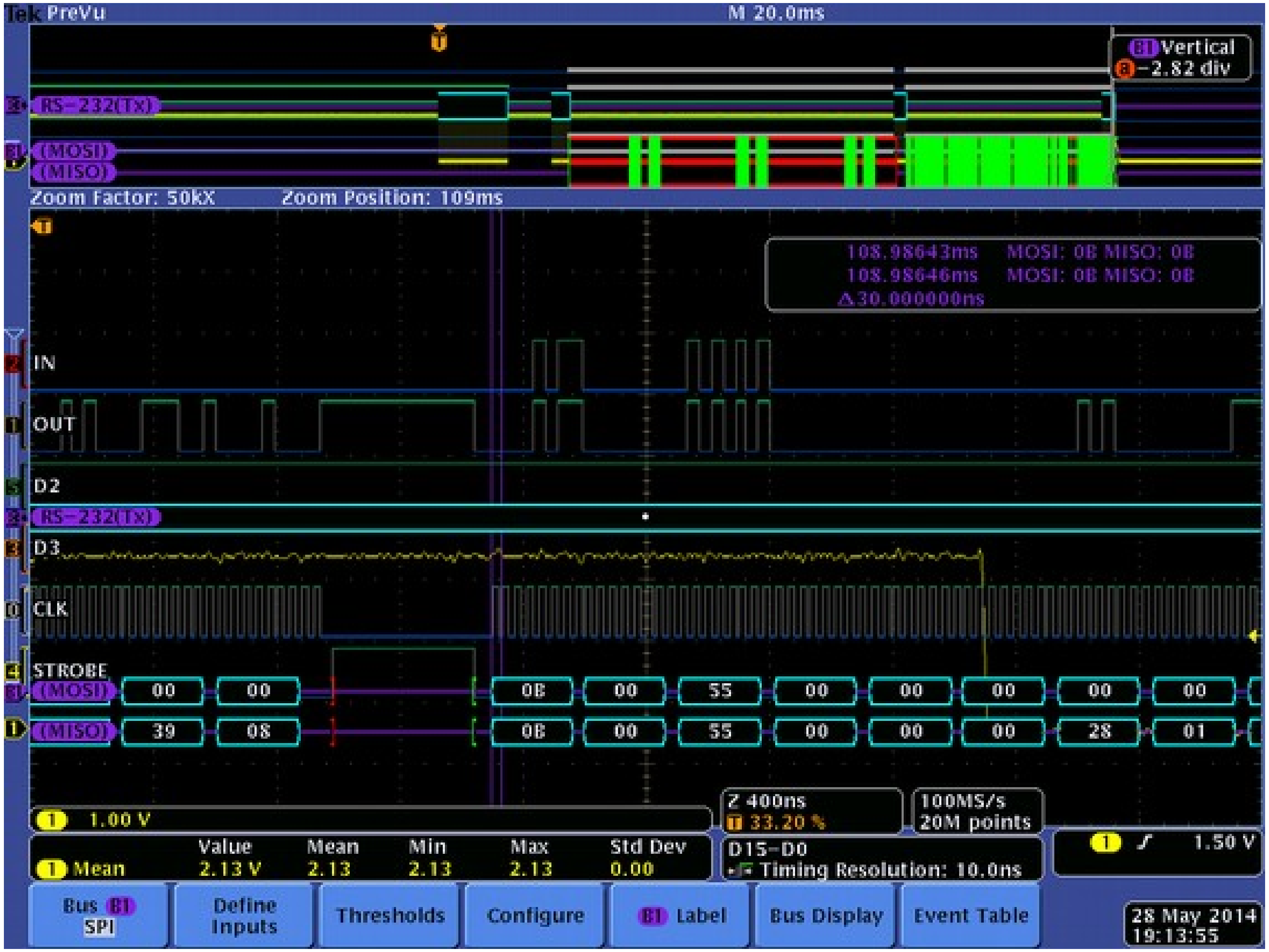
```
1  1.00 V
                Value    Mean   Min    Max    Std Dev
1 Mean          2.30 V   2.30   2.30   2.30   0.00

20.0ms          100MS/s        1  ƒ  1.50 V
T 33.20 %       20M points

D15-D0
Timing Resolution: 10.0ns

28 May 2014
19:08:35
```

Decode RS-232 Strings



```
1  1.00 V
        Value    Mean   Min    Max    Std Dev
1 Mean  1.19 V   1.19   1.19   1.19   0.00

Z 200µs          250MS/s        1  ⎍  1.50 V
T 33.20 %        20M points
D15–D0
Timing Resolution: 4.00ns

28 May 2014
18:58:00
```

Decode SPI ROM addresses & data



```
M 20.0ms
B1 Vertical
-2.82 div
Zoom Factor: 50kX    Zoom Position: 109ms

108.98643ms   MOSI: 0B MISO: 0B
108.98646ms   MOSI: 0B MISO: 0B
Δ30.000000ns

(MOSI)  00  00  0B  00  55  00  00  00  00  00
(MISO)  39  08  0B  00  55  00  00  00  28  01

1  1.00 V
Z 400ns          100MS/s
T 33.20 %        20M points

          Value    Mean   Min    Max    Std Dev
1 Mean    2.13 V   2.13   2.13   2.13   0.00

D15-D0
Timing Resolution: 10.0ns
1  1.50 V
28 May 2014
19:13:55
```

Some Kind of Verification...

- Modifying putative boot area causes boot to fail

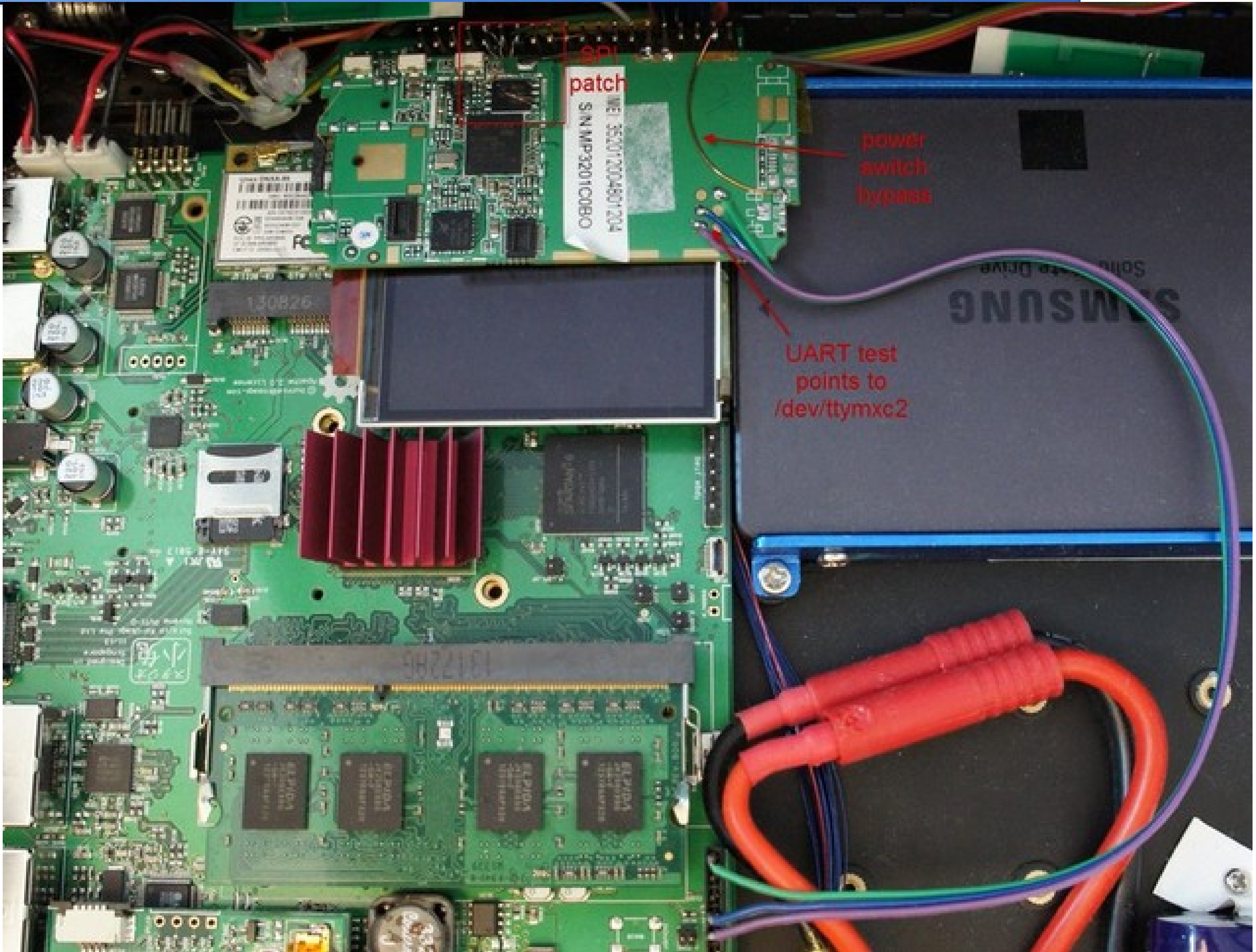
```
F1: 0000 0000
V0: 0000 0000 [0001]
00: 0000 0000
U0: 0000 0001 [0000]
G0: 0002 0000 [0000]
T0: 0000 00C0
Jump to BL

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


~~~ Welcome to MTK Bootloader V005 (since 2005) ~~~
**=================================================
==**
```

Original code

```
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 …
```

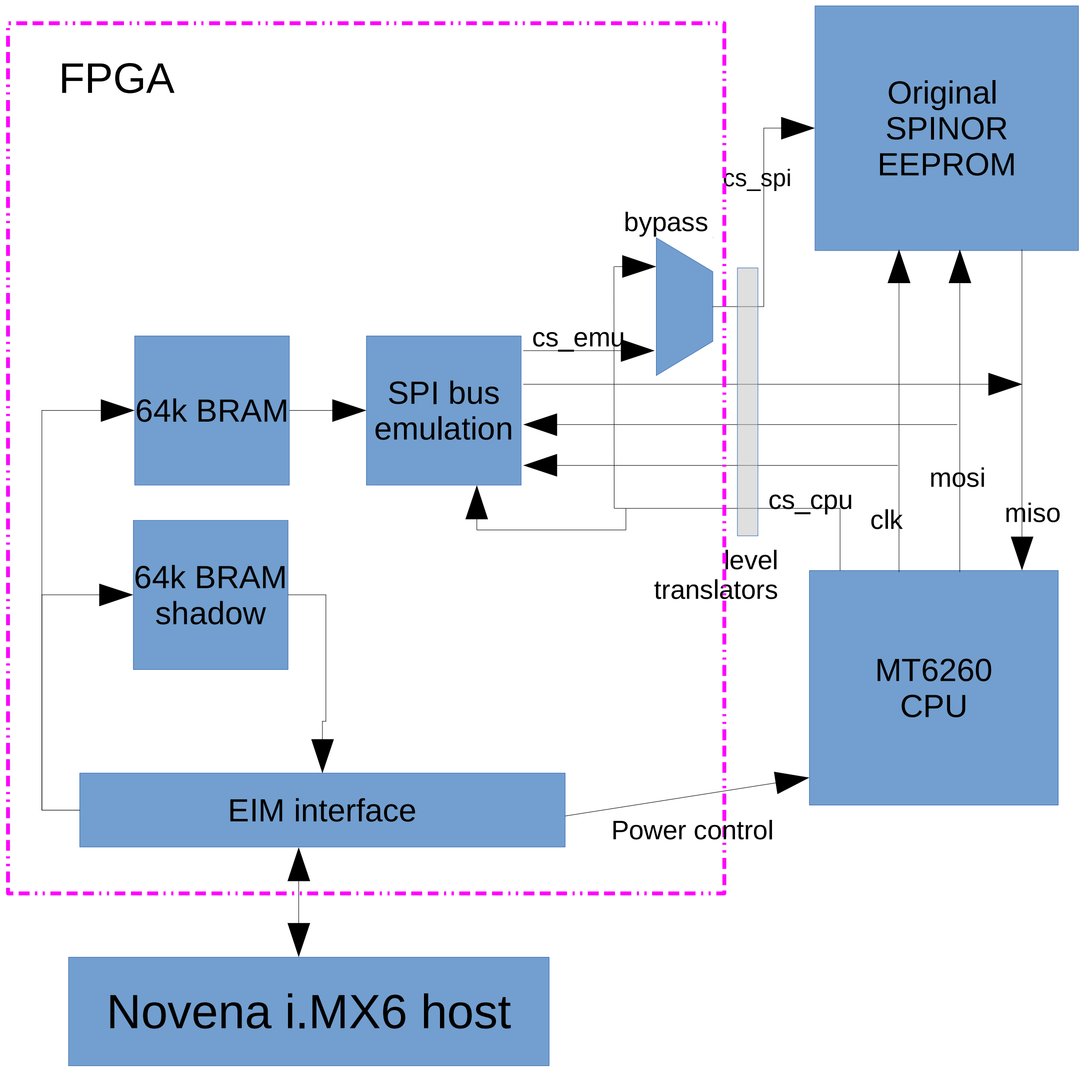
One-byte modification

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!

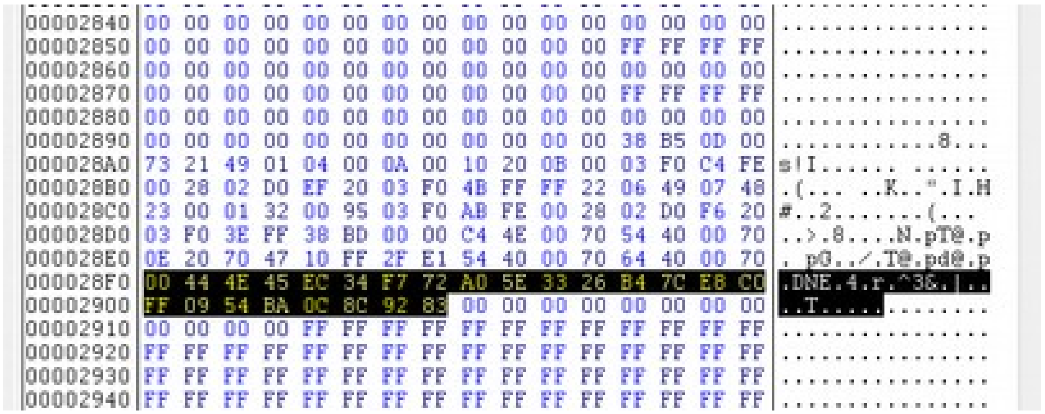


Finding the Verification

- 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



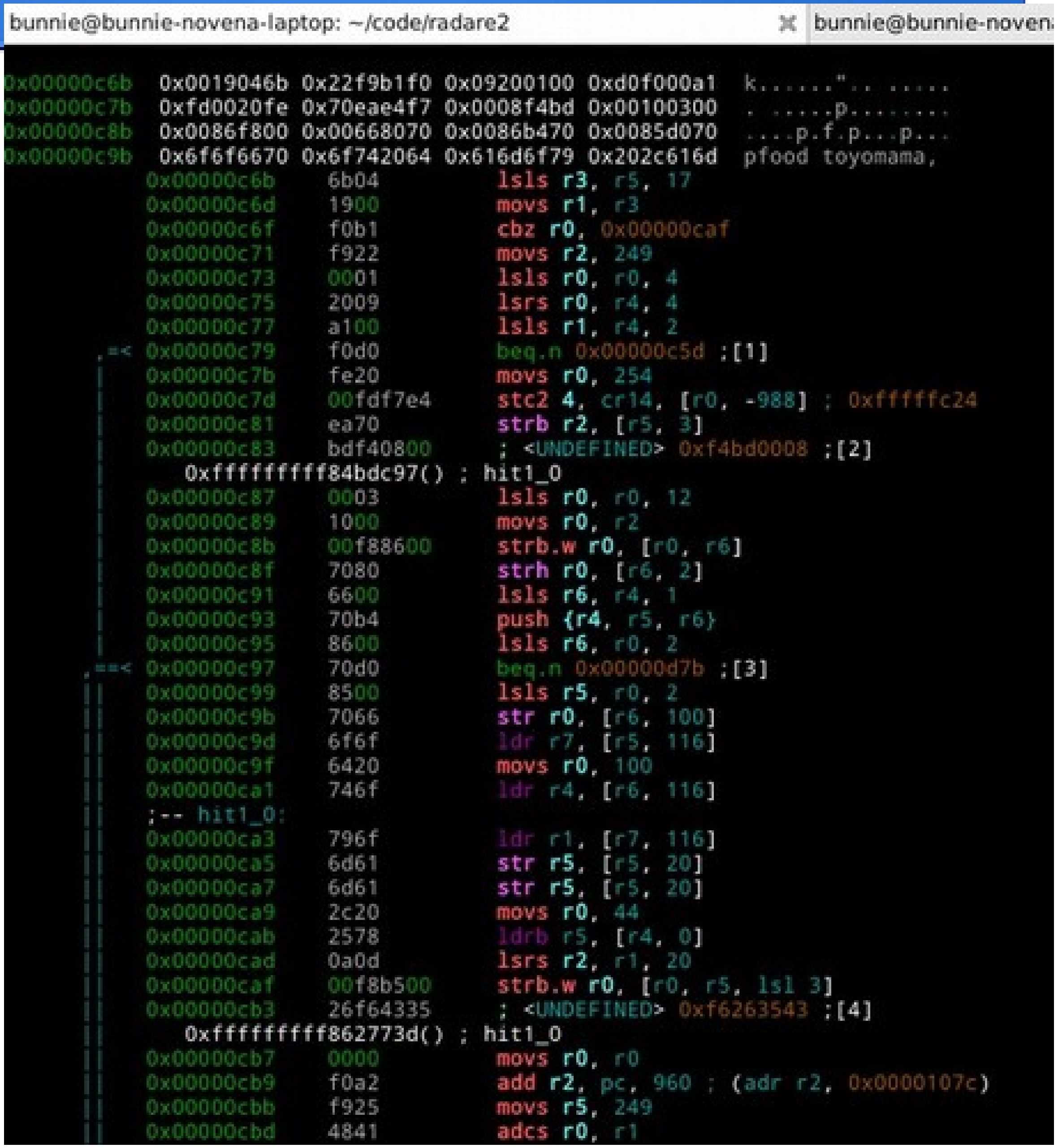
```
00002840 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00002850 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF ................
00002860 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00002870 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF ................
00002880 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00002890 00 00 00 00 00 00 00 00 00 00 00 00 38 B5 0D 00 ............8...
000028A0 73 21 49 01 04 00 0A 00 10 20 0B 00 03 F0 C4 FE s!I...... ......
000028B0 00 28 02 D0 EF 20 03 F0 4B FF FF 22 06 49 07 48 .(... ..K..".I.H
000028C0 23 00 01 32 00 95 03 F0 AB FE 00 28 02 D0 F6 20 #..2.......(...
000028D0 03 F0 3E FF 38 BD 00 00 C4 4E 00 70 54 40 00 70 ..>.8....N.pT@.p
000028E0 0E 20 70 47 10 FF 2F E1 54 40 00 70 64 40 00 70 . pG../.T@.pd@.p
000028F0 00 44 4E 45 EC 34 F7 72 A0 5E 33 26 B4 7C E8 C0 .DNE.4.r.^3&.|..
00002900 FF 09 54 BA 0C 8C 92 83 00 00 00 00 00 00 00 00 ..T.............
00002910 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF ................
00002920 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
00002930 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
00002940 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
```

```
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

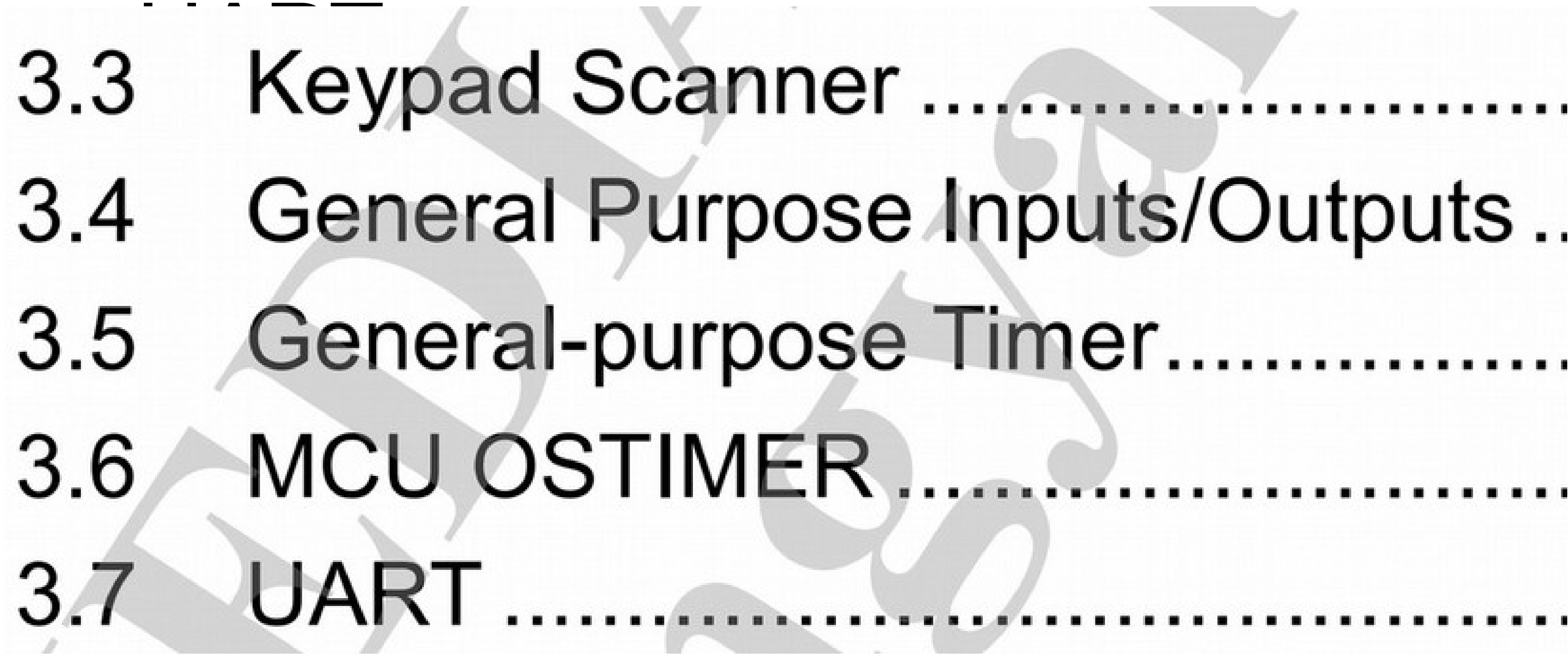
- SPI ROM is now a 64k window available via mmap() on Linux
- Port radare2 to treat 64k mmap() window as an I/O target
  - Include routine to auto-update intbl/extbl hash every time ROM is patched
  - https://github.com/xobs/radare2/tree/fernvale

radare2 example

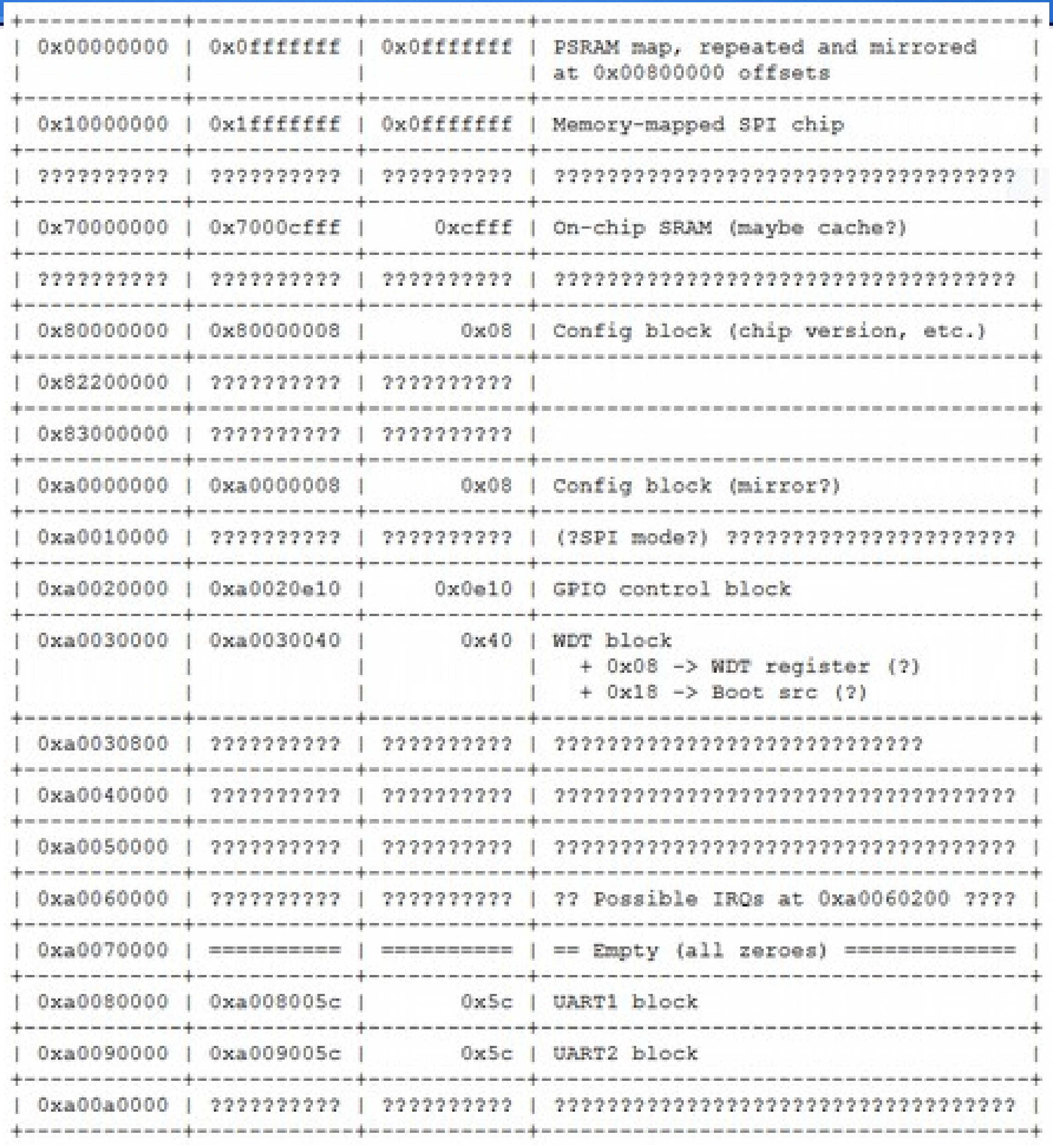


```
bunnie@bunnie-novena-laptop: ~/code/radare2                    bunnie@bunnie-noven

0x00000c6b   0x0019046b 0x22f9b1f0 0x09200100 0xd0f000a1  k......".. .....
0x00000c7b   0xfd0020fe 0x70eae4f7 0x0008f4bd 0x00100300  . .....p........
0x00000c8b   0x0086f800 0x00668070 0x0086b470 0x0085d070  ....p.f.p...p...
0x00000c9b   0x6f6f6670 0x6f742064 0x616d6f79 0x202c616d  pfood toyomama,
           0x00000c6b     6b04          lsls r3, r5, 17
           0x00000c6d     1900          movs r1, r3
           0x00000c6f     f0b1          cbz r0, 0x00000caf
           0x00000c71     f922          movs r2, 249
           0x00000c73     0001          lsls r0, r0, 4
           0x00000c75     2009          lsrs r0, r4, 4
           0x00000c77     a100          lsls r1, r4, 2
       ,=< 0x00000c79     f0d0          beq.n 0x00000c5d ;[1]
       |   0x00000c7b     fe20          movs r0, 254
       |   0x00000c7d     00fdf7e4      stc2 4, cr14, [r0, -988] ; 0xfffffc24
       |   0x00000c81     ea70          strb r2, [r5, 3]
       |   0x00000c83     bdf40800      ; <UNDEFINED> 0xf4bd0008 ;[2]
       |      0xffffffff84bdc97() ; hit1_0
       |   0x00000c87     0003          lsls r0, r0, 12
       |   0x00000c89     1000          movs r0, r2
       |   0x00000c8b     00f88600      strb.w r0, [r0, r6]
       |   0x00000c8f     7080          strh r0, [r6, 2]
       |   0x00000c91     6600          lsls r6, r4, 1
       |   0x00000c93     70b4          push {r4, r5, r6}
       |   0x00000c95     8600          lsls r6, r0, 2
      ,==< 0x00000c97     70d0          beq.n 0x00000d7b ;[3]
      ||   0x00000c99     8500          lsls r5, r0, 2
      ||   0x00000c9b     7066          str r0, [r6, 100]
      ||   0x00000c9d     6f6f          ldr r7, [r5, 116]
      ||   0x00000c9f     6420          movs r0, 100
      ||   0x00000ca1     746f          ldr r4, [r6, 116]
      ||   ;-- hit1_0:
      ||   0x00000ca3     796f          ldr r1, [r7, 116]
      ||   0x00000ca5     6d61          str r5, [r5, 20]
      ||   0x00000ca7     6d61          str r5, [r5, 20]
      ||   0x00000ca9     2c20          movs r0, 44
      ||   0x00000cab     2578          ldrb r5, [r4, 0]
      ||   0x00000cad     0a0d          lsrs r2, r1, 20
      ||   0x00000caf     00f8b500      strb.w r0, [r0, r5, lsl 3]
      ||   0x00000cb3     26f64335      ; <UNDEFINED> 0xf6263543 ;[4]
      ||      0xffffffff862773d() ; hit1_0
      ||   0x00000cb7     0000          movs r0, r0
      ||   0x00000cb9     f0a2          add r2, pc, 960 ; (adr r2, 0x0000107c)
      ||   0x00000cbb     f925          movs r5, 249
      ||   0x00000cbd     4841          adcs r0, r1
```

Doing what we can

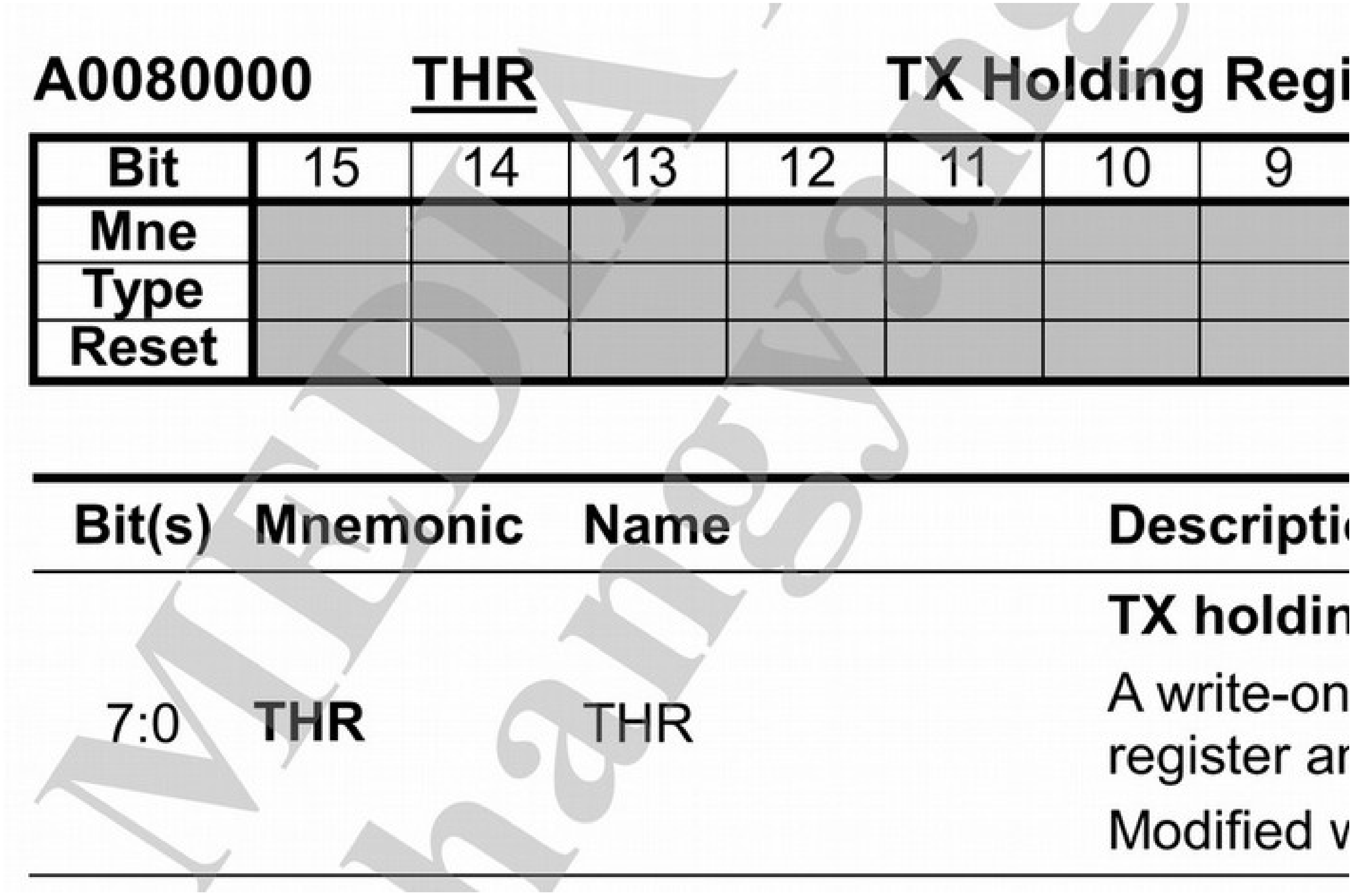


Memory Map



```
+------------+------------+------------+-----------------------------------+
| 0x00000000 | 0x0fffffff | 0x0fffffff | PSRAM map, repeated and mirrored  |
|            |            |            | at 0x00800000 offsets             |
+------------+------------+------------+-----------------------------------+
| 0x10000000 | 0x1fffffff | 0x0fffffff | Memory-mapped SPI chip            |
+------------+------------+------------+-----------------------------------+
| ?????????? | ?????????? | ?????????? | ????????????????????????????????? |
+------------+------------+------------+-----------------------------------+
| 0x70000000 | 0x7000cfff |     0xcfff | On-chip SRAM (maybe cache?)       |
+------------+------------+------------+-----------------------------------+
| ?????????? | ?????????? | ?????????? | ????????????????????????????????? |
+------------+------------+------------+-----------------------------------+
| 0x80000000 | 0x80000008 |       0x08 | Config block (chip version, etc.) |
+------------+------------+------------+-----------------------------------+
| 0x82200000 | ?????????? | ?????????? |                                   |
+------------+------------+------------+-----------------------------------+
| 0x83000000 | ?????????? | ?????????? |                                   |
+------------+------------+------------+-----------------------------------+
| 0xa0000000 | 0xa0000008 |       0x08 | Config block (mirror?)            |
+------------+------------+------------+-----------------------------------+
| 0xa0010000 | ?????????? | ?????????? | (?SPI mode?) ???????????????????? |
+------------+------------+------------+-----------------------------------+
| 0xa0020000 | 0xa0020e10 |     0x0e10 | GPIO control block                |
+------------+------------+------------+-----------------------------------+
| 0xa0030000 | 0xa0030040 |       0x40 | WDT block                         |
|            |            |            |   + 0x08 -> WDT register (?)      |
|            |            |            |   + 0x18 -> Boot src (?)          |
+------------+------------+------------+-----------------------------------+
| 0xa0030800 | ?????????? | ?????????? | ????????????????????????????      |
+------------+------------+------------+-----------------------------------+
| 0xa0040000 | ?????????? | ?????????? | ????????????????????????????????? |
+------------+------------+------------+-----------------------------------+
| 0xa0050000 | ?????????? | ?????????? | ????????????????????????????????? |
+------------+------------+------------+-----------------------------------+
| 0xa0060000 | ?????????? | ?????????? | ?? Possible IRQs at 0xa0060200 ???? |
+------------+------------+------------+-----------------------------------+
| 0xa0070000 | ========== | ========== | == Empty (all zeroes) ============ |
+------------+------------+------------+-----------------------------------+
| 0xa0080000 | 0xa008005c |       0x5c | UART1 block                       |
+------------+------------+------------+-----------------------------------+
| 0xa0090000 | 0xa009005c |       0x5c | UART2 block                       |
+------------+------------+------------+-----------------------------------+
| 0xa00a0000 | ?????????? | ?????????? | ????????????????????????????????? |
+------------+------------+------------+-----------------------------------+
```

Doing what we can



A0080000 **THR** **TX Holding Regi**

Bit	15	14	13	12	11	10	9
Mne							
Type							
Reset							

Bit(s)	**Mnemonic**	**Name**	**Descripti**
7:0	**THR**	THR	**TX holdin** A write-on register ar Modified v

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

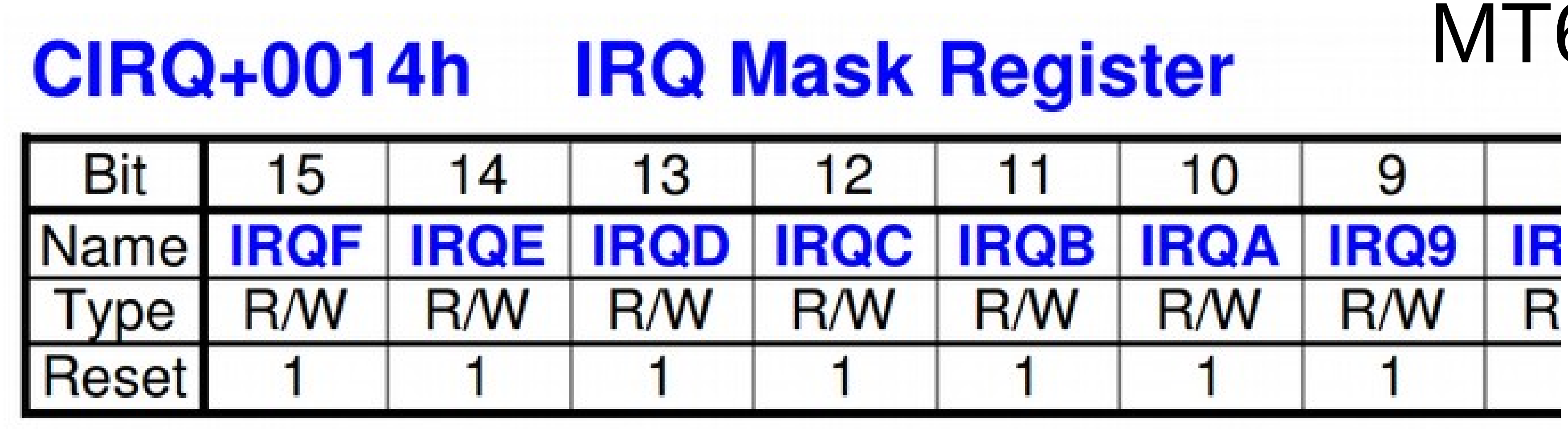
Christopher Polk / Getty Images

The (One) IRQ is Standardized on ARM

Exception	**Offset**
Reset	0
Undefined Instruction	4
SWI	8
Prefetch Abort	12
Data Abort	16
Reserved	20
IRQ	**24**
FIQ	28

The interrupt problem

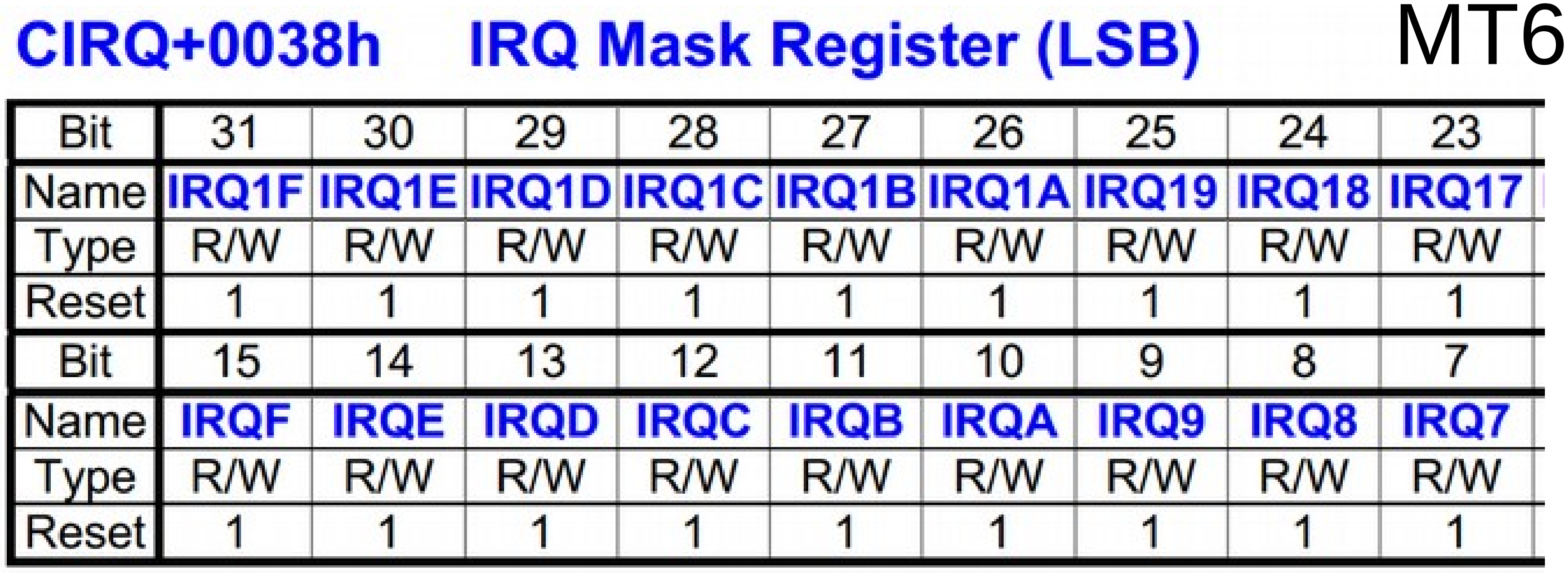
MT6205B



CIRQ+0014h IRQ Mask Register

Bit	15	14	13	12	11	10	9	
Name	IRQF	IRQE	IRQD	IRQC	IRQB	IRQA	IRQ9	IR
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
Reset	1	1	1	1	1	1	1	

MT6235



CIRQ+0038h IRQ Mask Register (LSB)

Bit	31	30	29	28	27	26	25	24	23
Name	IRQ1F	IRQ1E	IRQ1D	IRQ1C	IRQ1B	IRQ1A	IRQ19	IRQ18	IRQ17
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1	1
Bit	15	14	13	12	11	10	9	8	7
Name	IRQF	IRQE	IRQD	IRQC	IRQB	IRQA	IRQ9	IRQ8	IRQ7
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1	1

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”)

Back to MTK11B.1308

- 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!

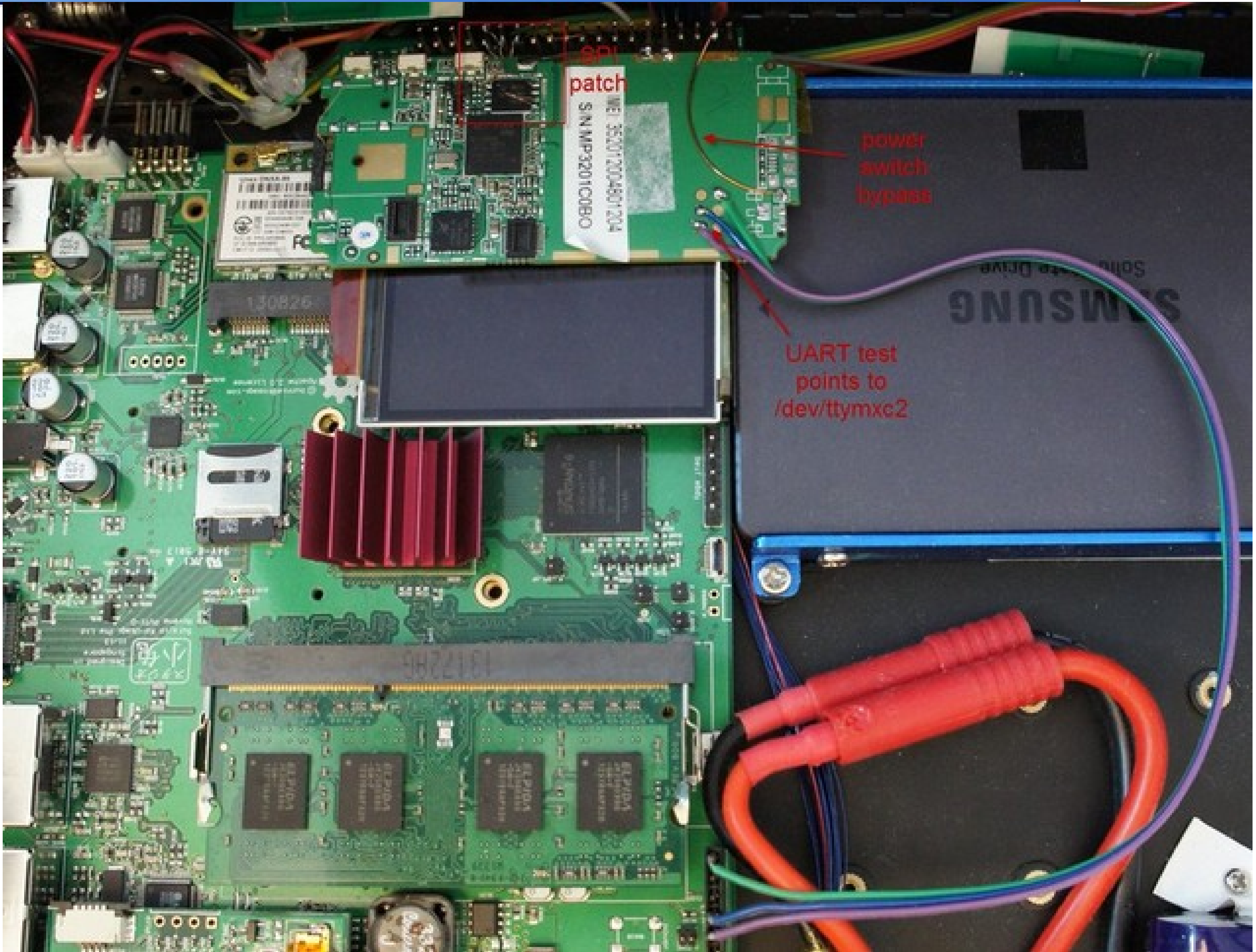
IRQ Problem: Solved

- 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

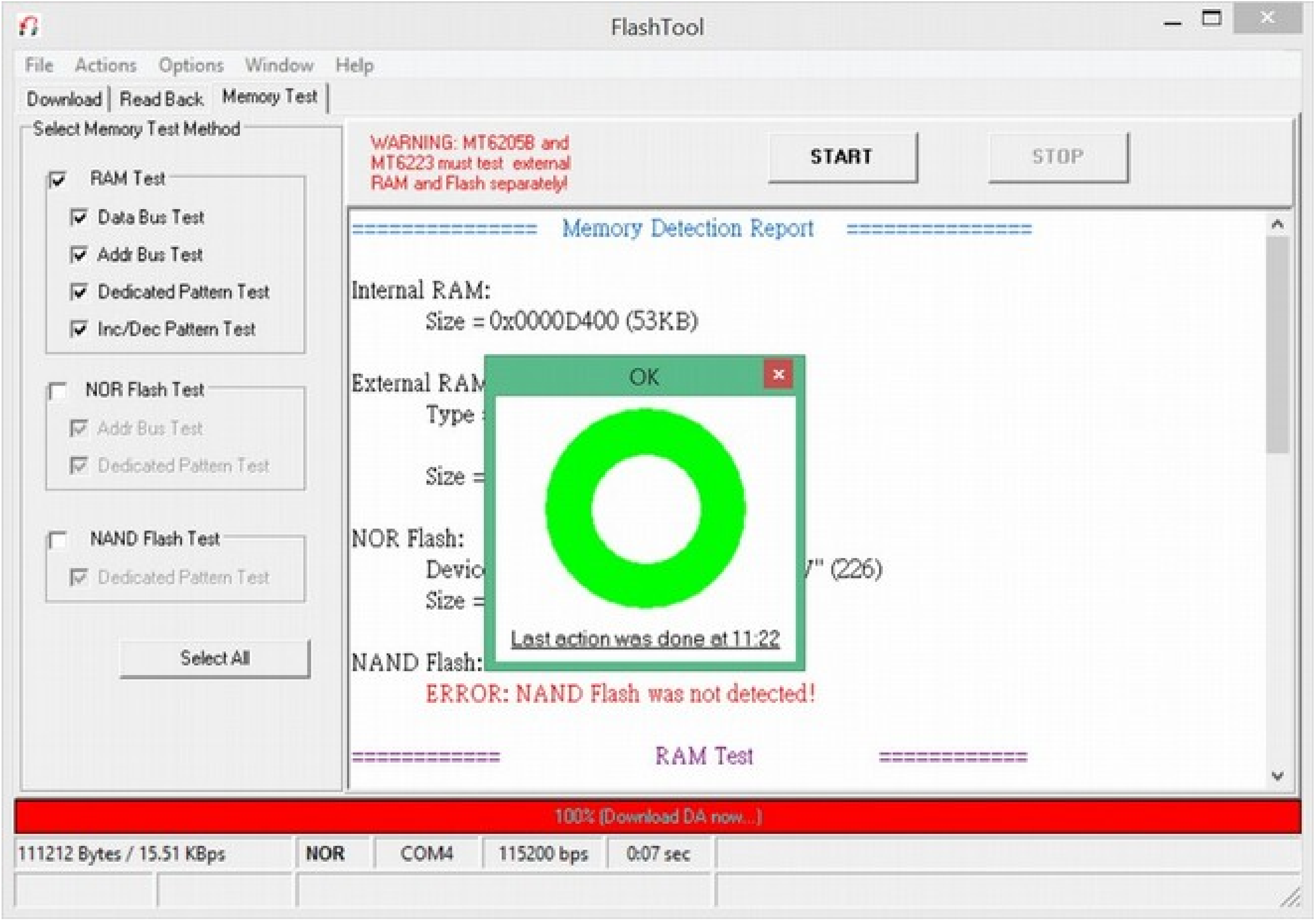
NuttX port

- 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



WARNING: MT6205B and MT6223 must test external RAM and Flash separately!

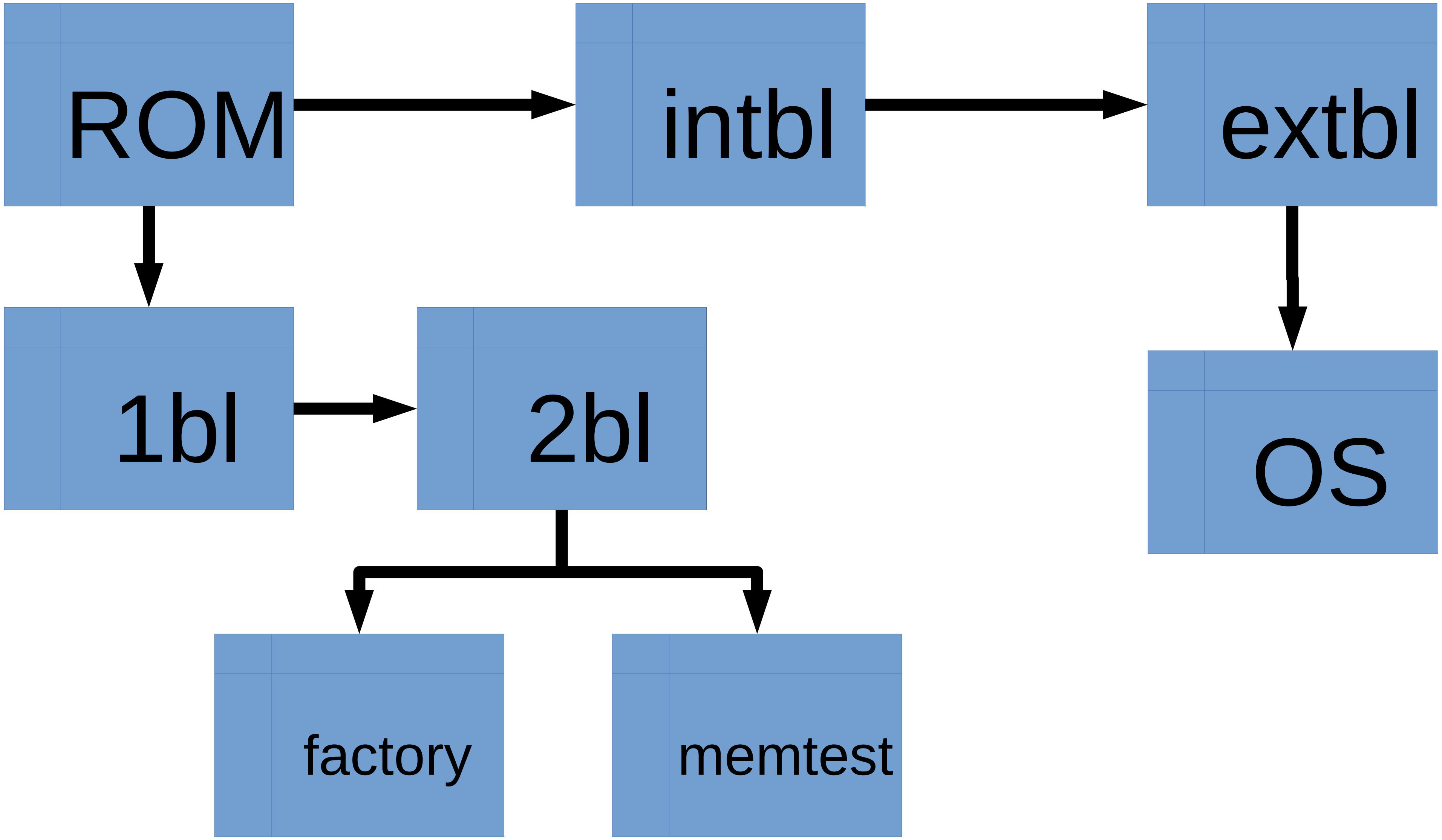
```
===============  Memory Detection Report  ===============

Internal RAM:
        Size = 0x0000D400 (53KB)

External RAM
        Type :

        Size =

NOR Flash:
        Devic                              " (226)
        Size =

NAND Flash:
        ERROR: NAND Flash was not detected!

============          RAM Test          ============
```

Last action was done at 11:22

100% (Download DA now...)

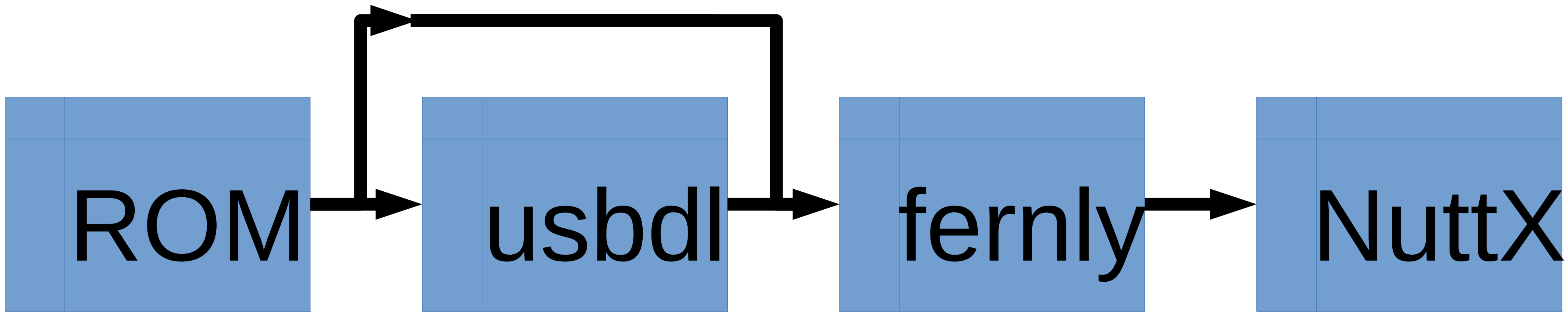
111212 Bytes / 15.51 KBps | NOR | COM4 | 115200 bps | 0:07 sec

Boot - Mediatek



fernvale-usb-loader

- Open-licensed
- Writes to /dev/ttyUSB0



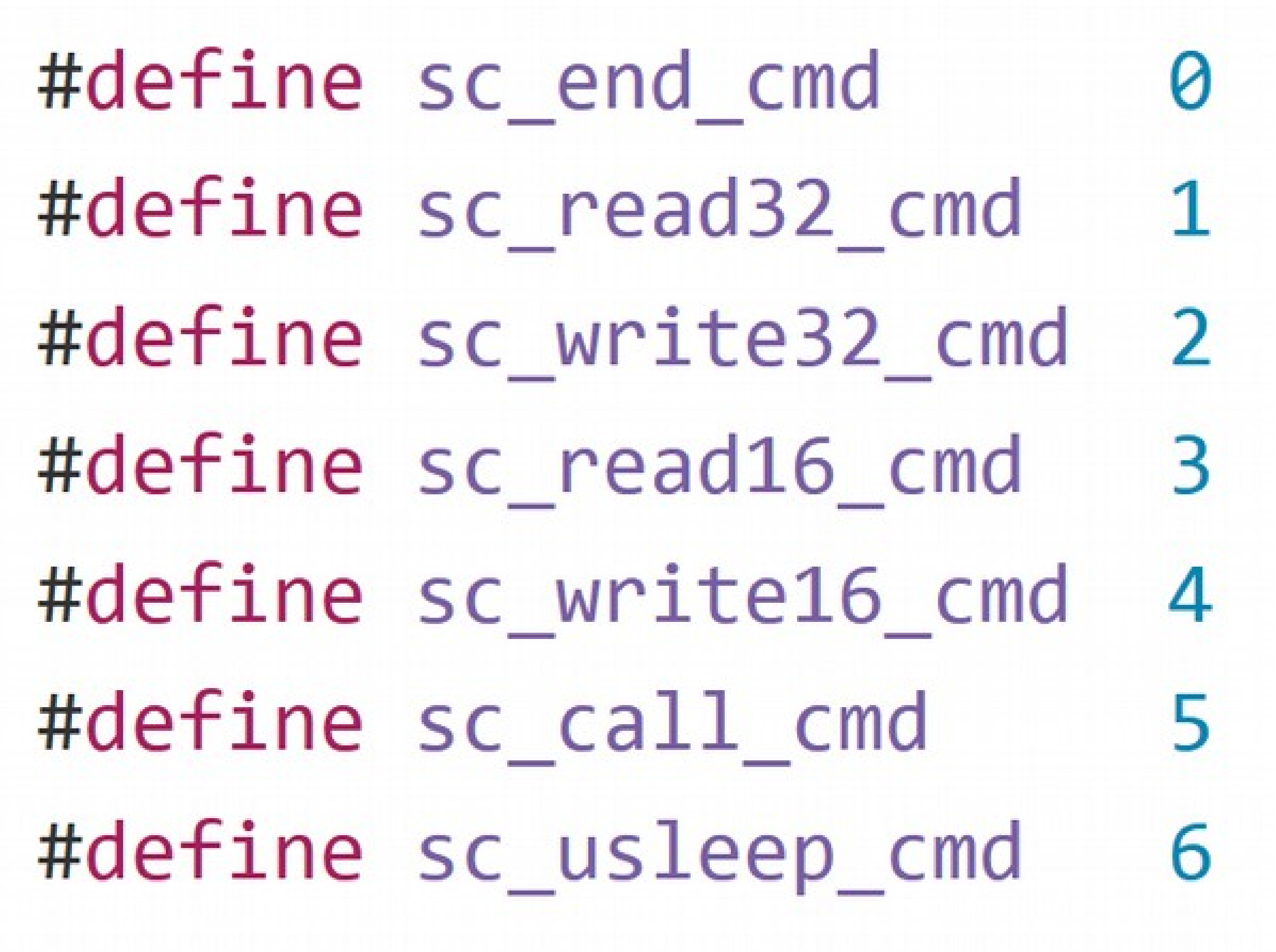
Towards an “open” boot

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

Scriptic

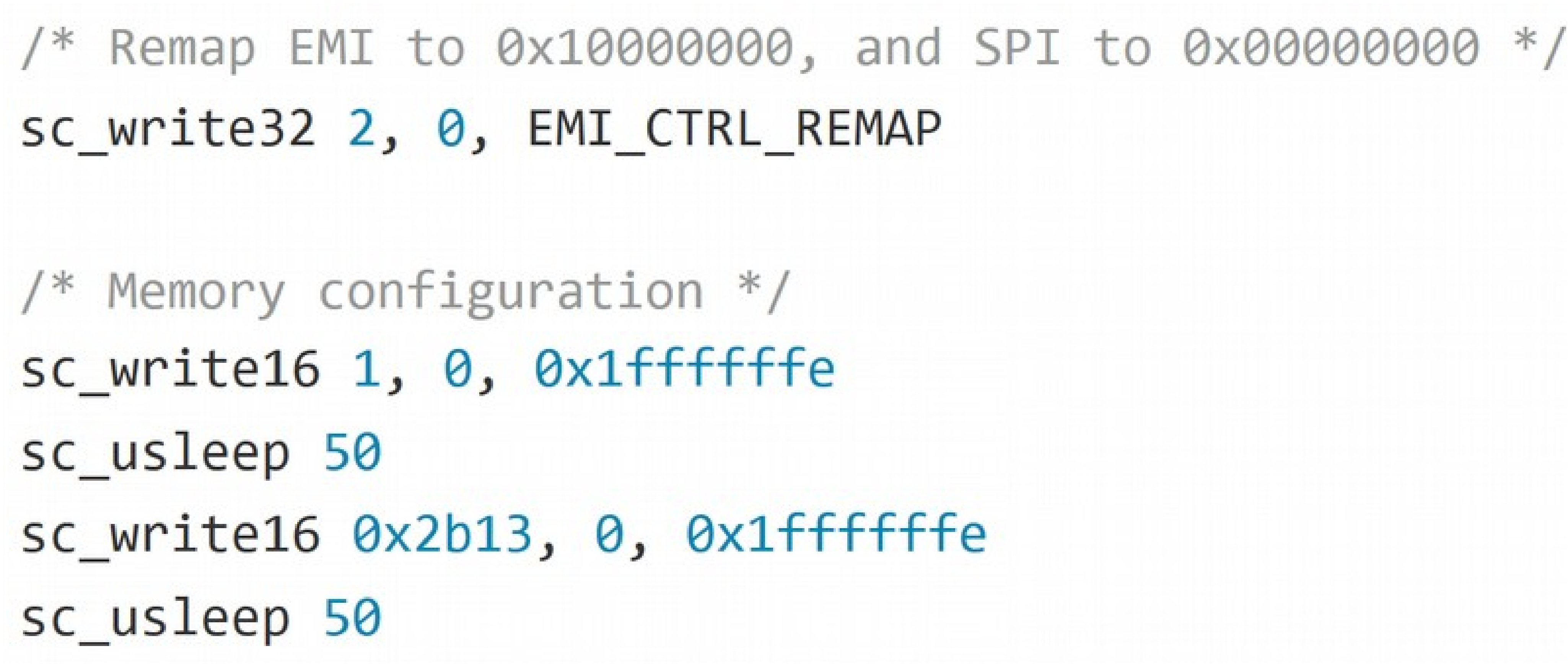
- Simple command language
  - Very similar problem to SoC boot scripts
- Can distill facts down into scripts
- Scripts are not Turing-complete
  - Can call C functions
  - E.g. PSRAM calibration
- Implemented as assembler macros

Scriptic - Commands



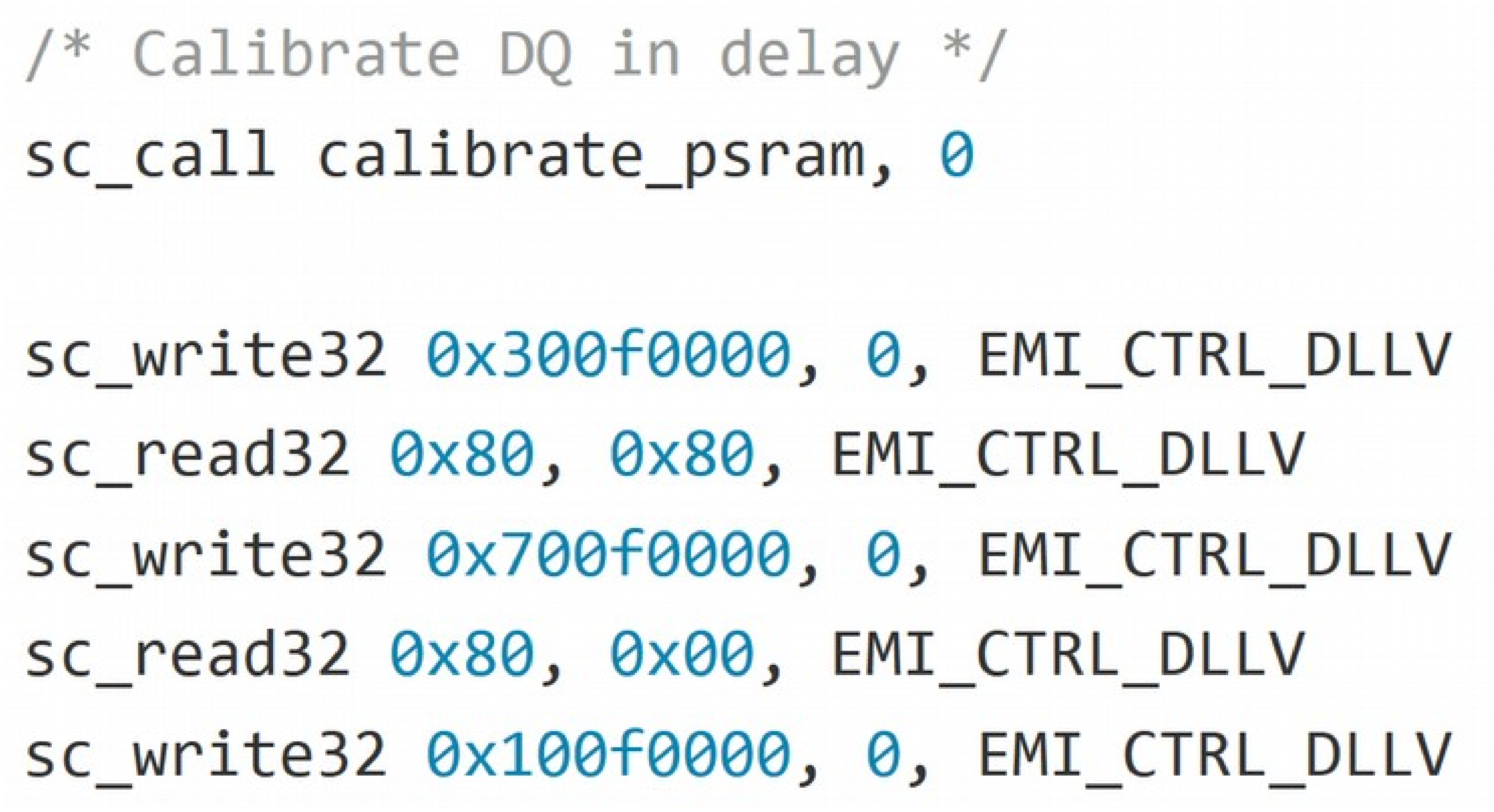
```
#define sc_end_cmd        0
#define sc_read32_cmd     1
#define sc_write32_cmd    2
#define sc_read16_cmd     3
#define sc_write16_cmd    4
#define sc_call_cmd       5
#define sc_usleep_cmd     6
```

Scriptic - Basics



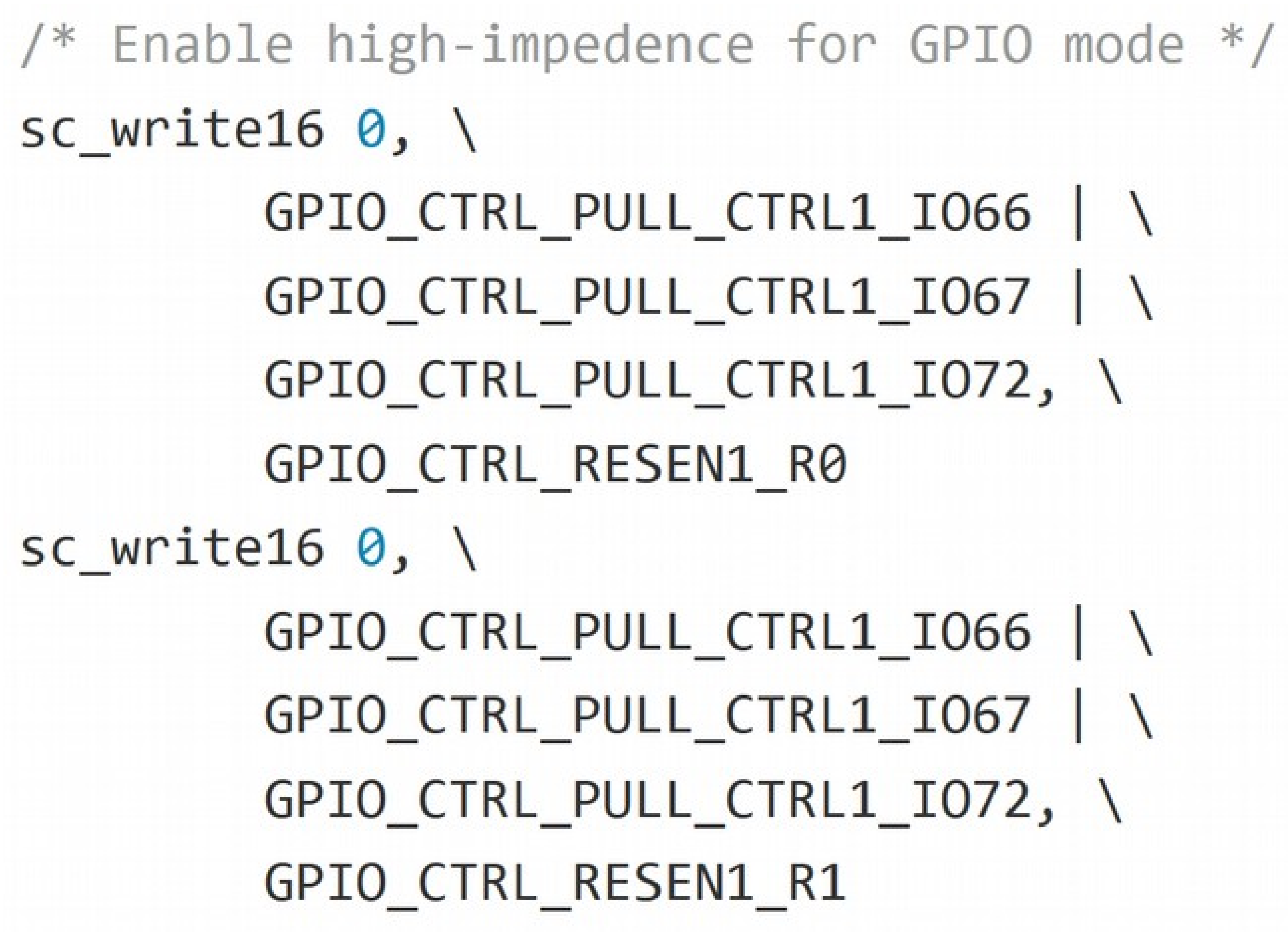
```
/* Remap EMI to 0x10000000, and SPI to 0x00000000 */
sc_write32 2, 0, EMI_CTRL_REMAP

/* Memory configuration */
sc_write16 1, 0, 0x1ffffffe
sc_usleep 50
sc_write16 0x2b13, 0, 0x1ffffffe
sc_usleep 50
```

Scriptic – Functions



```
/* 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
```

Scriptic - Masks



```
/* Enable high-impedence for GPIO mode */
sc_write16 0, \
        GPIO_CTRL_PULL_CTRL1_IO66 | \
        GPIO_CTRL_PULL_CTRL1_IO67 | \
        GPIO_CTRL_PULL_CTRL1_IO72, \
        GPIO_CTRL_RESEN1_R0
sc_write16 0, \
        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