Anatomy of Contemporary Smartphone Hardware

by

Harald Welte <laforge@gnumonks.org>

Introduction

Who is speaking to you?

- an independent Free Software developer, consultant and trainer
- 13 years experience using/deploying and developing for Linux on server and workstation
- 10 years professional experience doing Linux system + kernel level development
- strong focus on network security and embedded
- expert in Free and Open Source Software (FOSS) copyright and licensing
- digital board-level hardware design, esp. embedded systems
- active developer and contributor to many FOSS projects
- thus, a techie, who will therefore not have fancy animated slides ;)

Introduction

My involvement with mobile phones

- 2003/2004: gpl-violations.org / Motorola A780
- 2004: Started OpenEZX for A780 (now E680, A1200, E6, ...)
- 2006: Bought my first GSM BTS
- 06/2006-11/2007: Lead System Architect at Openmoko, Inc.
- 10/2008: Started the 'gnufiish' project
- 12/2008: Running my own GSM test network (see talk tomorrow morning!)

Introduction

What is a Smartphone?

- No clear definition on terminology
- Many technical people differentiate
  - Feature Phone: Single-CPU phone
    - Single CPU + Single OS for GSM + UI
  - Smartphone: Dual-CPU phone
    - First CPU core for the actual network protocol
    - Second CPU for the UI + Applications

Smartphone hardware

Major Components (AP side)

- Application Processor (System-on-a-Chip)
  - Samsung / Marvell / Ti / Freescale
- Flash (typically SLC or MLC NAND)
  - connects to SoC internal NAND controller
- RAM (mobileSDRAM / mobileDDR)
  - connects to SoC internal SDRAM controller
- Power Management Unit (PMU / PMIC)
  - connects via I2C or SPI
- Audio Codec
  - connects via I2C + PCM
- Bluetooth
  - connects via UART or SPI
- WiFi
  - connects via SDIO or SPI

Smartphone hardware

Major Components (BP side)

- DSP
  - RF Baseband Signal Processing
  - Voice Signal Processing
- CPU (typically ARM7)
  - GSM protocol Stack (Layer 2, Layer 3)
  - AT Command Interpreter
  - Typically LCM + Keypad Matrix
    - not used, just for feature phone
- RF PA (Power Amplifier)
- Antenna Switch (MEMS SPST)
- DAC + ADC
  - Voice and Baseband DAC + ADC

Smartphone hardware

AP / BP hardware interface

- 2G (GSM Voice/SMS/CSD + GPRS)
  - typically connects via (high-speed) UART
  - sometimes USB
  - UART speeds still sufficient
- 3G (UMTS) / 3.5G (HSDPA/HSUPA)
  - shared memory interface
  - SPI or USB
- USB by itself is not sufficient
  - doesn't allow for wake-up by BP

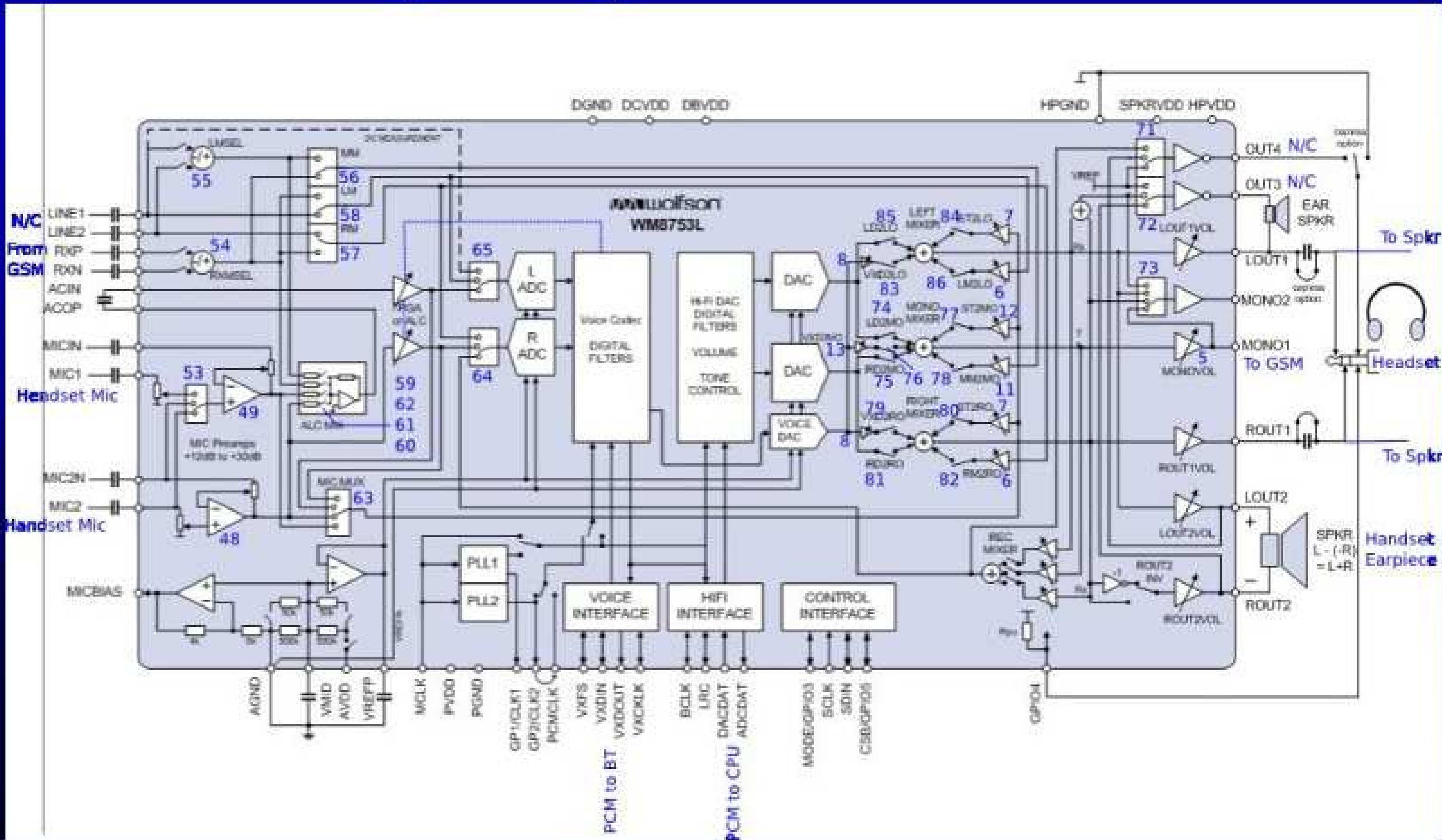
Smartphone hardware

Audio interface

- Typically at least three analog outputs
  - one handset ear speaker
  - one ringtone speaker
  - headphone/earphone/headset
- Typically at least two analog inputs
  - built-in microphone
  - headphone/earphone/headset
- GSM Modem interface
  - analog at line-level (for featurephone bb)
  - digital (PCM) in some cases
- At least two PCM busses
  - one between SoC and Audio Codec
  - one between Bluetooth and Audio Codec

Smartphone hardware

Audio routing on Openmoko GTA01/GTA02

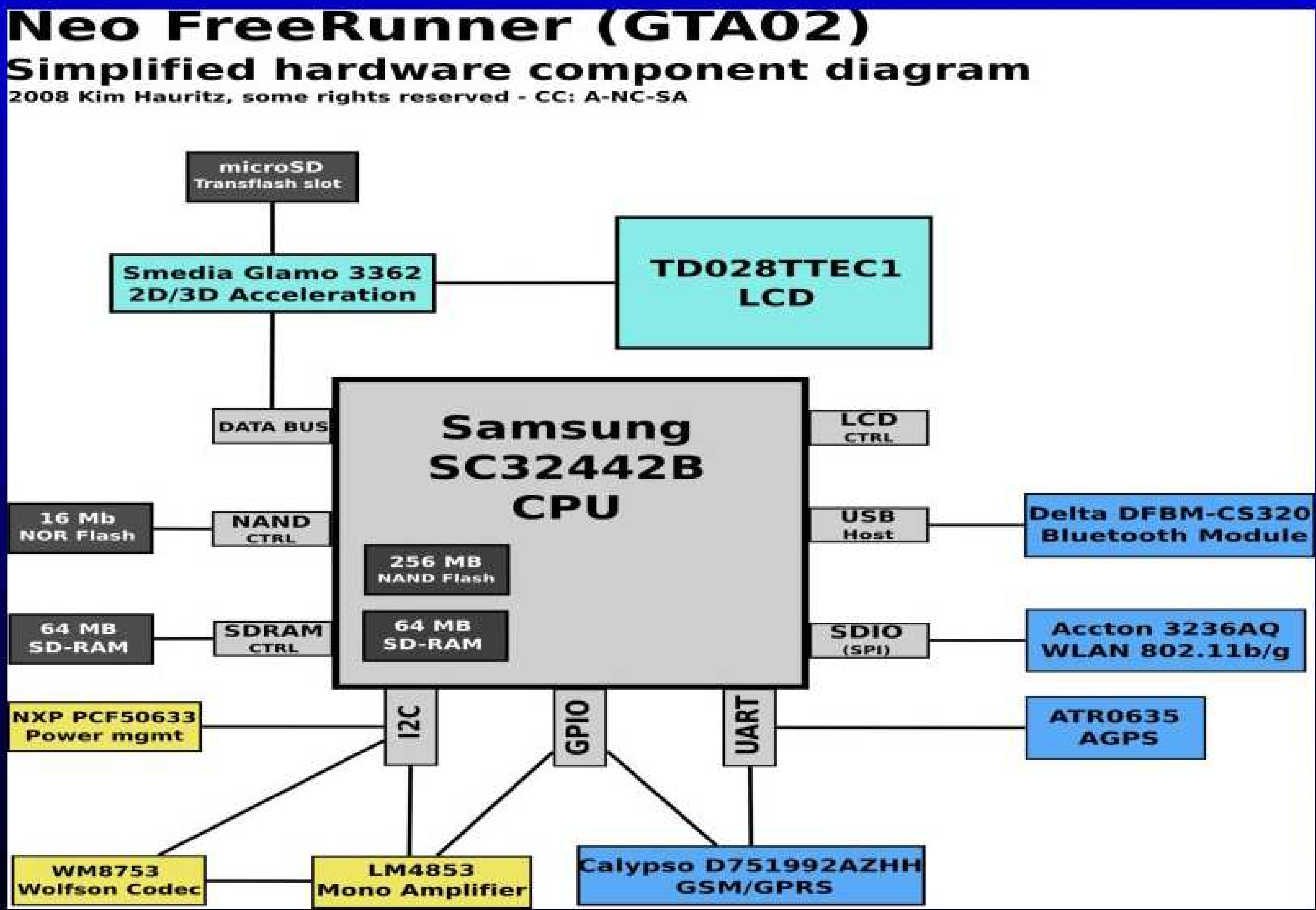


Openmoko hardware

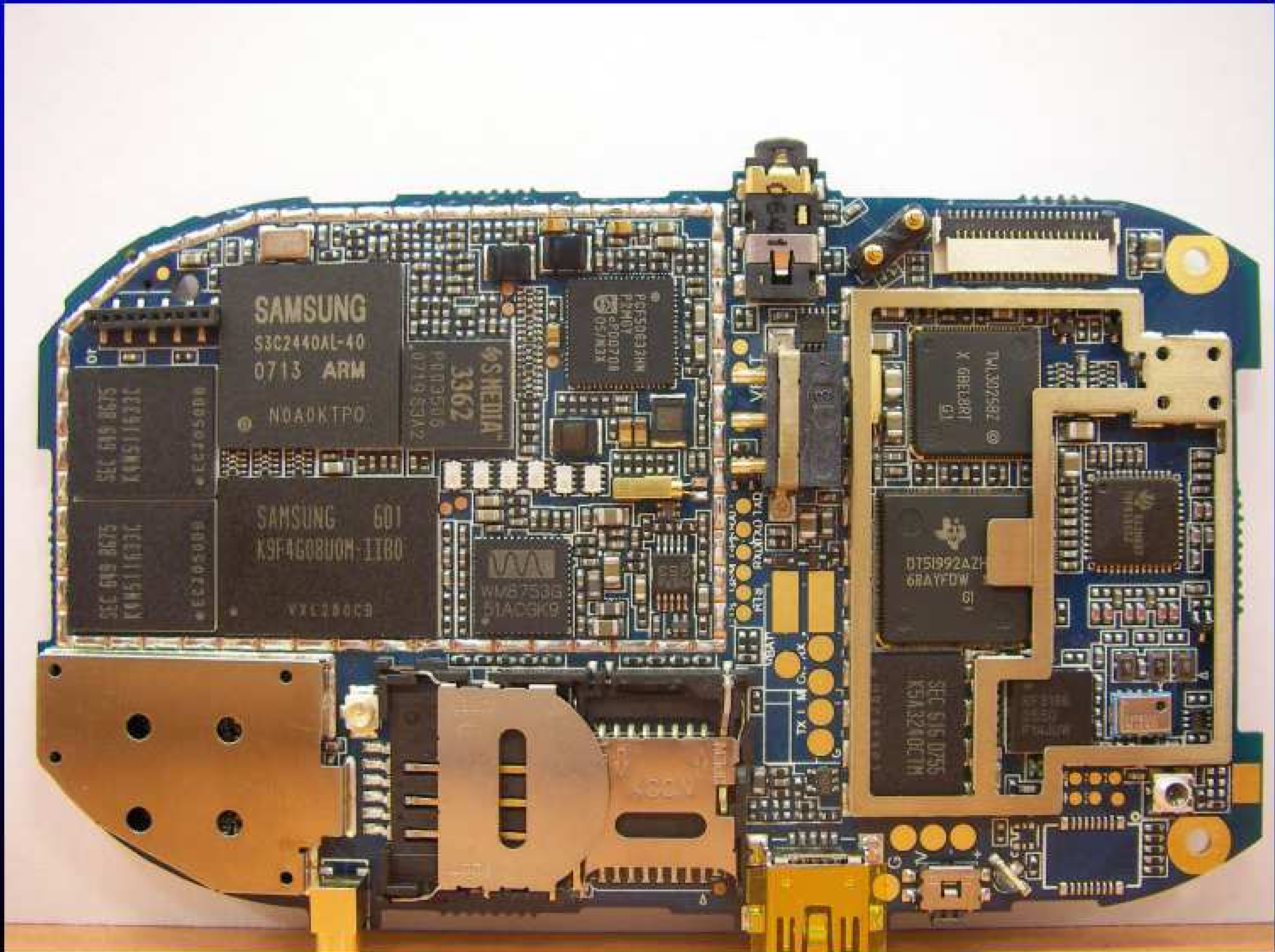
Openmoko hardware

- GTA01 (Neo1973)
- GTA02 (FreeRunner)
- Interesting to study, since schematics are public
  - only the GSM baseband side has been removed

Openmoko hardware



Openmoko hardware



Motorola EZX hardwware

Motorola EZX hardwware

- Generation 1:
  - Motorola A760, A768, A780, E680
  - Hardware mostly known, schematics leaked
- Generation 2:
  - Motorola A910, A1200, Rokr E6, A1600
  - Hardware mostly known, schematics partially leaked
- Generation 3:
  - Rokr E8, Rizr Z6, Razr2 V8, i876, U9, A1800
  - Very little knowledge about hardwrae, custom SoC

Motorola EZX hardwware

EZ Gen1

- SoC: PXA27x
- PMU: Motorola PCAP
  - interface: SPI
- BP: Neptune LTE
  - interface: USB + gpio handshake

Motorola EZX hardwware

EZ Gen3

- SoC: Custom Freescale
- BP: Custom Freescale
- A lot is unknown

Community based projects

Linux mobile phone community ports

- The vendor ships WM or other OS, community replaces it
- xda-developers.com community
  - mostly focused on HTC devices
  - way too little developers fro too many devices
  - hardware product cycles getting shorter / faster
  - many new devices based on completely undocumented chipsets

Linux-friendly hardware

The E-TEN glofiish device family

- various devices with different parameters
  - screen full-VGA or QVGA
  - EDGE-only, UMTS or HSDPA
  - keyboard or no keyboard
  - GPS or no GPS
  - Wifi or no Wifi
- application processor is always the same (S3C2442)

Linux-friendly hardware

I went through this process

- I found the E-TEN glofiish devices
- They are very similar to Openmoko
  - Samsung S3C2442 SoC MCP with NAND+SDRAM
  - TD028TTEC1 full-VGA LCM
- Other hardware parts reasonably supported/known
  - Marvell 8686/libertas WiFi (SPI attached)
  - SiRF GPS (UART attached)
  - CSR Bluetooth (UART attached)
- Only some unknown parts
  - CPLD for power management and kbd matrix
  - Ericsson GSM Modem (AT commandset documented!)
  - Cameras (I don't really care)

Project gnufiish

Project 'gnufiish'

- Port Linux to the E-TEN glofiish devices
- Initially to the M800 and X800
- Almost all glofiish have very similar hardware
- Openmoko merges all my patches in their kernel!
- Official inclusion to Openmoko distribution

Project gnufiish

gnufiish Status

- Kernel (2.6.24/2.6.27) booted on _first attempt_

- Working
  - I2C host controller
  - I2C communication to CPLD and FM Radio
  - USB Device mode (Ethernet gadget)
  - Touchscreen input
  - LCM Framebuffer
  - LCM Backlight control
  - GPS and Bluetooth power control
  - GPIO buttons
- In the works
  - Audio Codec driver (50% done)
  - GSM Modem (SPI) driver (80% done)
  - M800 Keyboard + Capsense driver (25% done)
  - SPI glue to libertas WiFi driver (70% done)

HOWTO

How was this done?

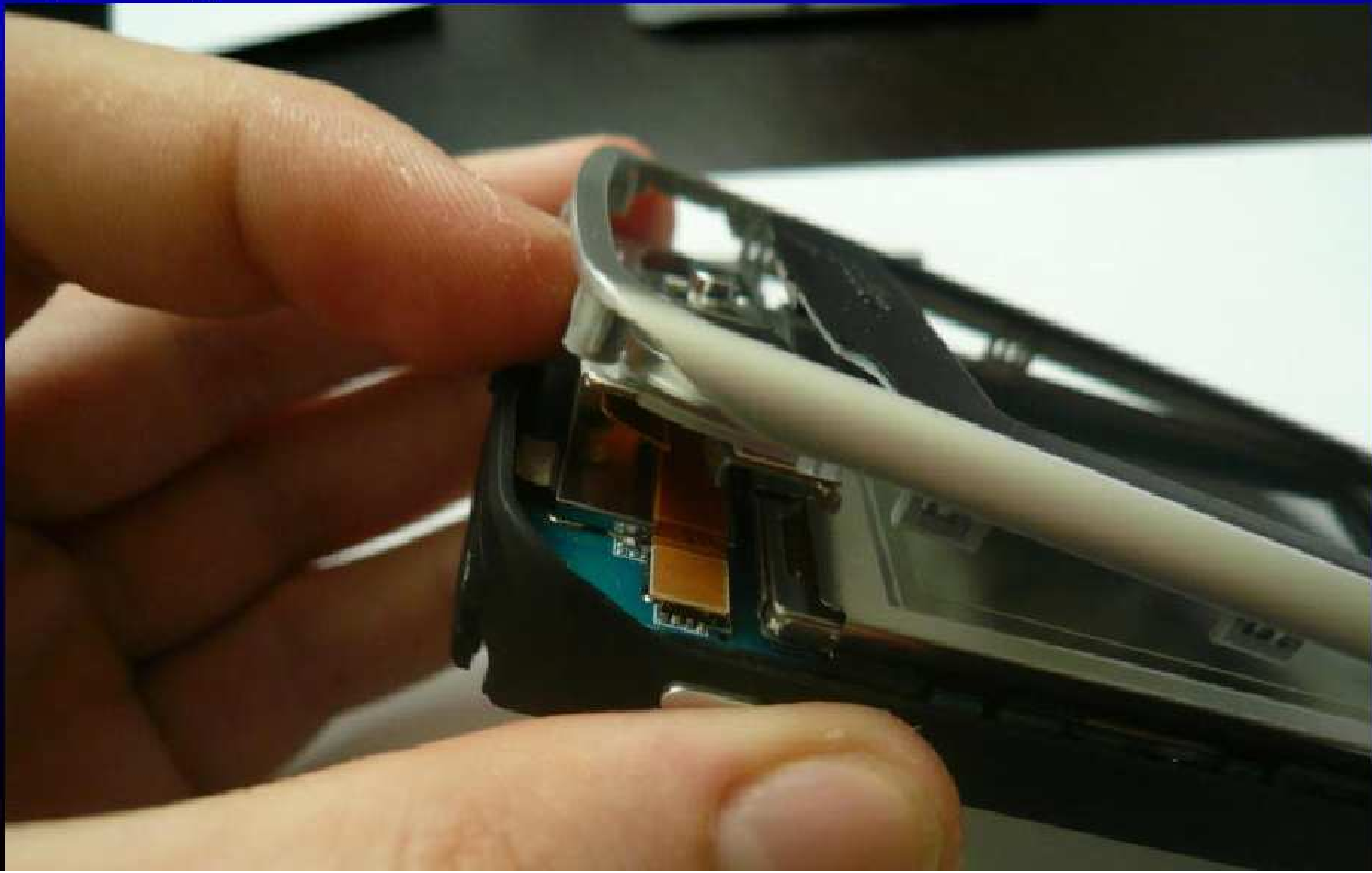
- Various reverse engineering techniques
  - Take actual board apart, note major components
  - Use HaRET (hardwar reverse engineering tool)
  - Find + use JTAG testpads
  - Find + use serial console
  - Disassemble WinMobile drivers

Take hardware apart

Opening the case and void your warranty

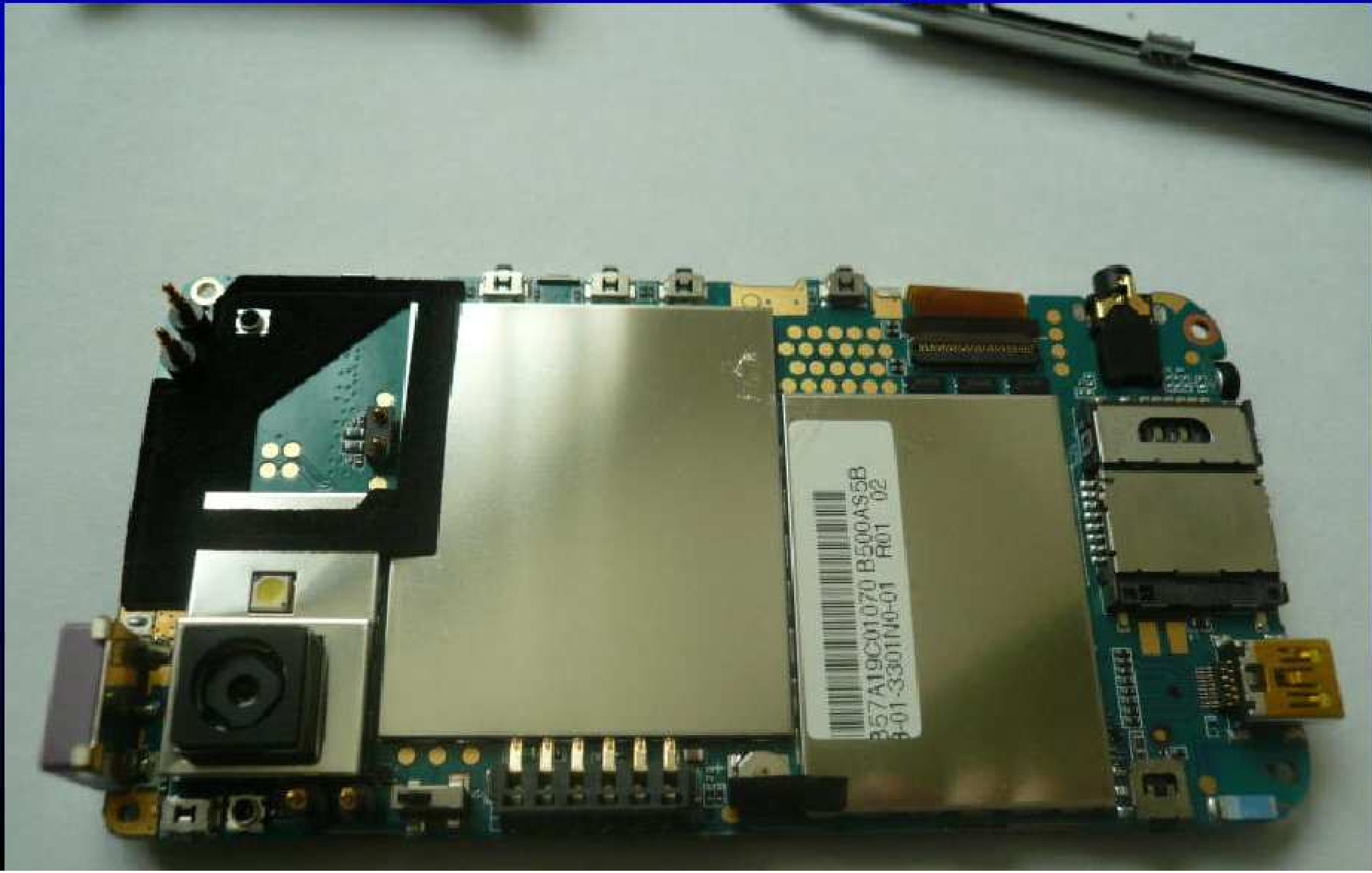
Take hardware apart

Opening the case



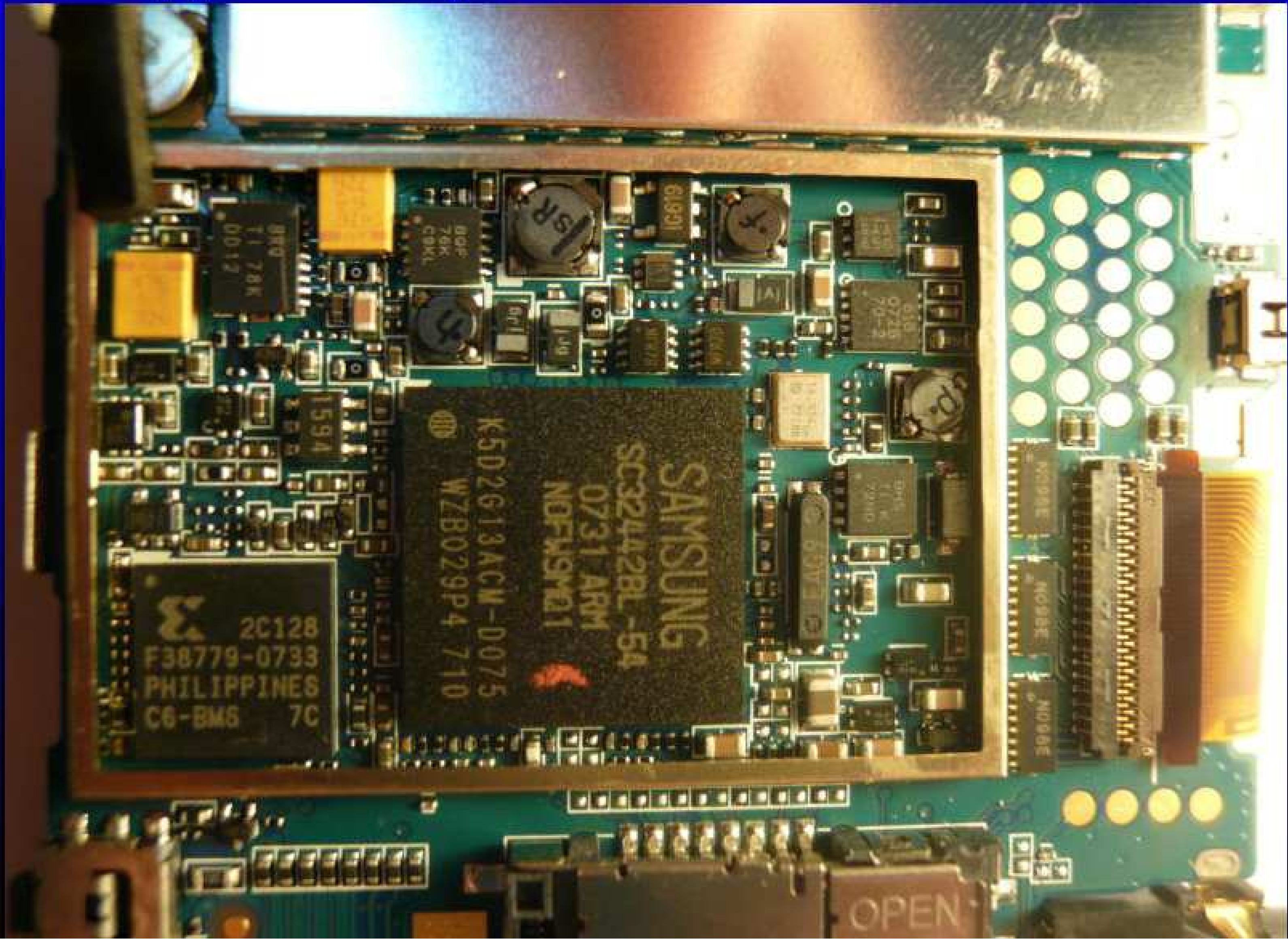
Take hardware apart

The Mainboard with all its shielding covers



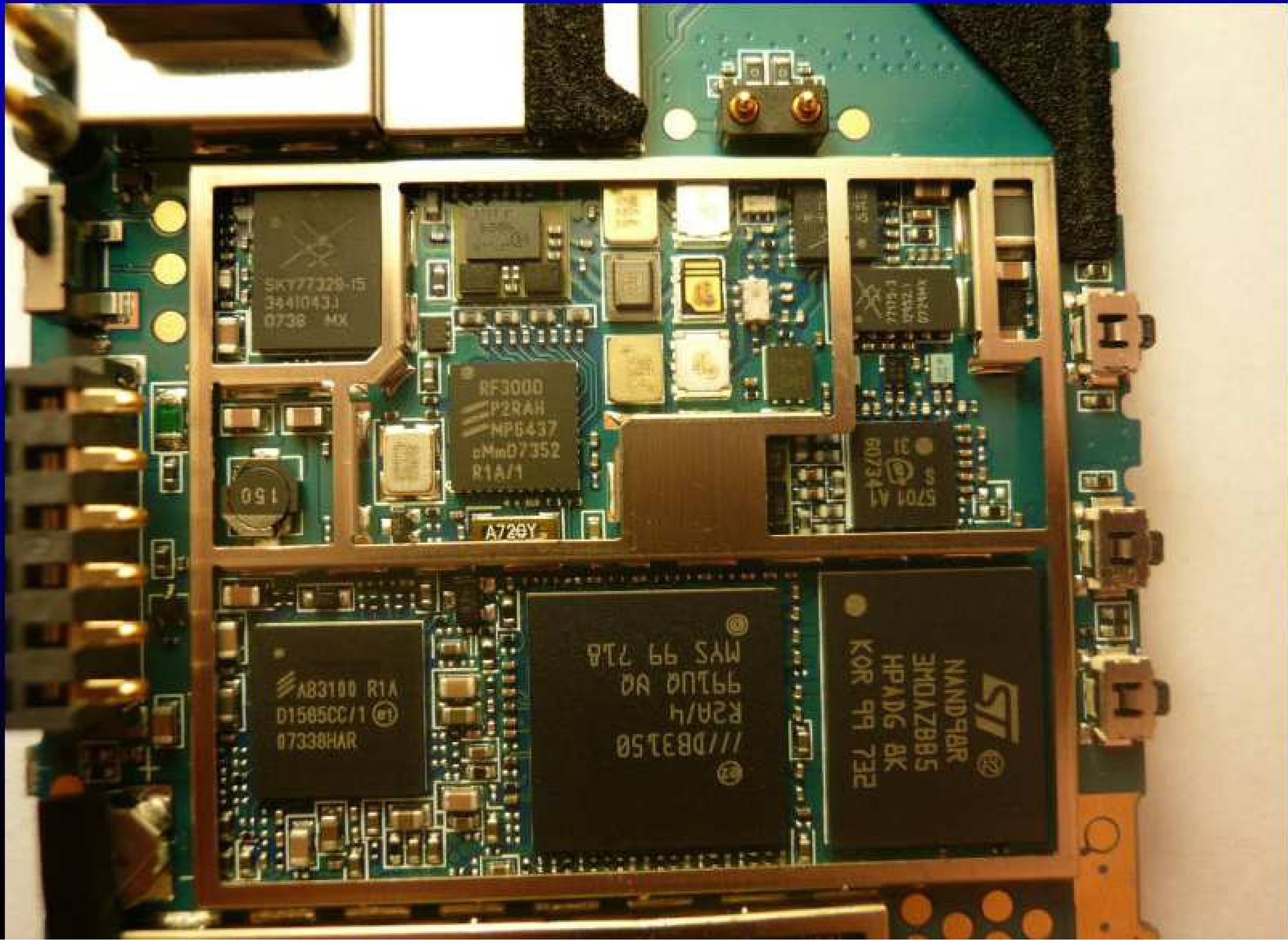
Take hardware apart

The application processor section



Take hardware apart

The HSDPA modem section



Take hardware apart

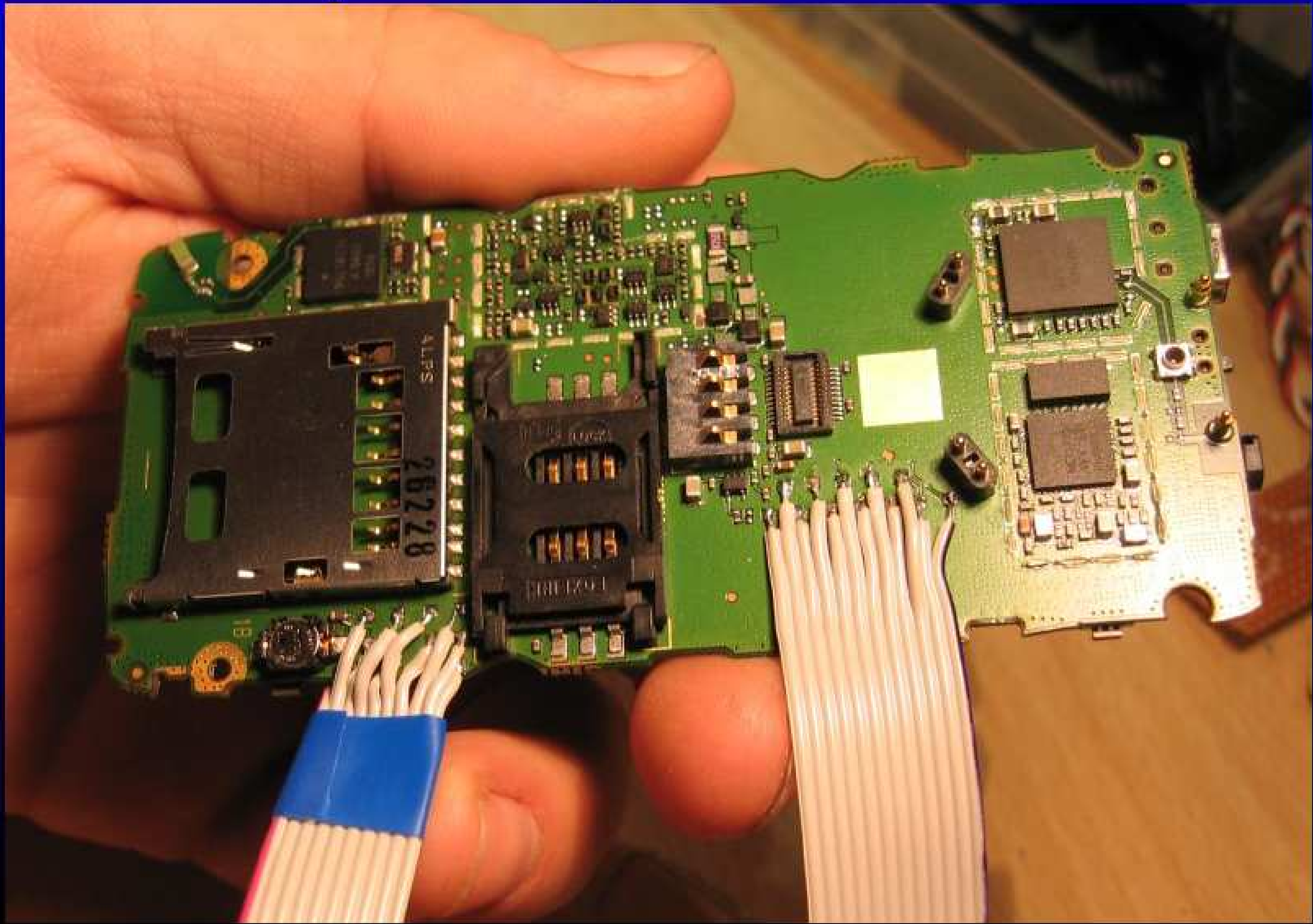
The backside

JTAG pins

- JTAG is a very useful interface
  - boundary scan (EXTEST + INTEST)
  - ARM Integrated Debug Macrocell
- Find + use JTAG testpads
  - look for suspicious testpads on PCB
  - tracing PCB traces impossible at 8-layer PCB
  - trial + error
  - sometimes you might find schematics ;)

JTAG pins

Find + use JTAG testpads

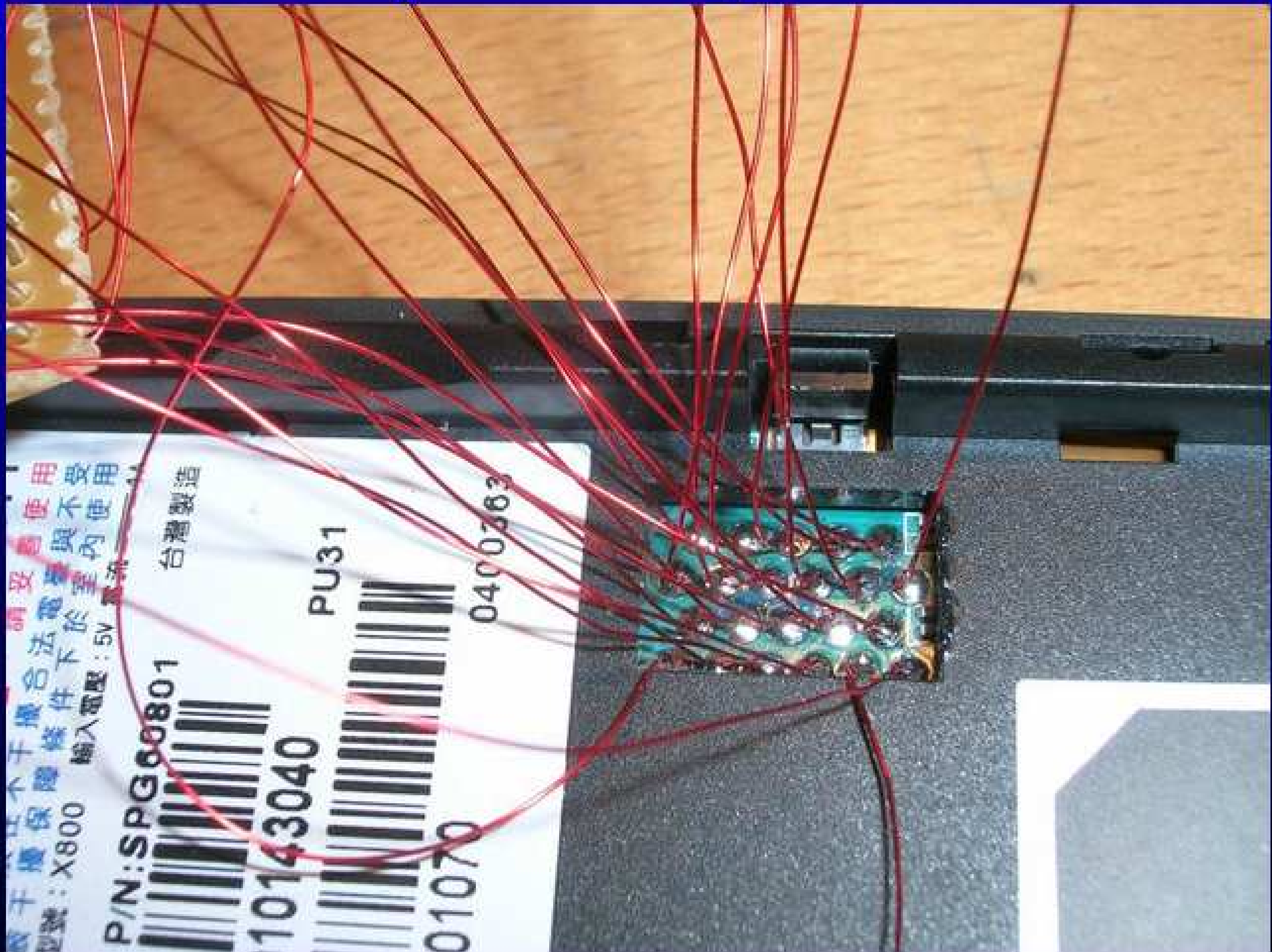


JTAG pins

- Find + use JTAG testpads
  - JTAG is basically a long shift register
  - Input, Output, Clock (TDI, TDO, TCK)
  - Therefore, you can try to shift data in and check if/where it comes out
  - Automatized JTAG search by project "jtagfinder" by Hunz (German CCC member)

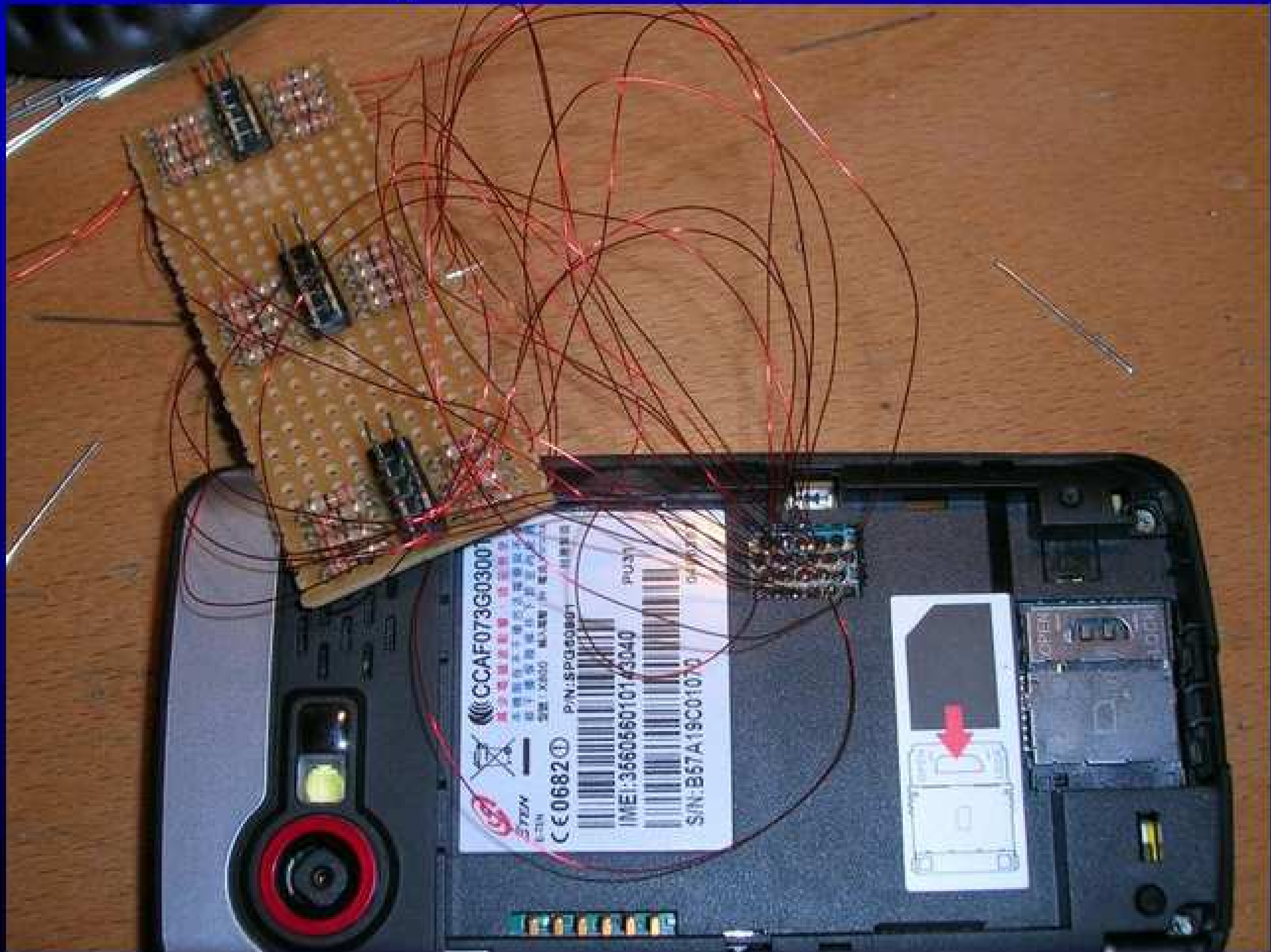
JTAG pins

Find + use JTAG testpads



JTAG pins

Find + use JTAG testpads



JTAG pins

Find + use JTAG testpads

JTAG pins

Find + use JTAG testpads

JTAG pins

Found JTAG pins

- Chain 1
  - Samsung S3C2442 Application Processor
  - Has standard ARM JTAG ICE
- Chain 2
  - CPLD programming interface
- Remaining work
  - find the nTRST and nSRST pins

Serial console

How to find the serial console

- Just run some code that you think writes to it
- Use a Scope to find typical patterns of a serial port
- I haven't actually done (or needed) this on the glofiish yet, but on many other devices
- RxD pin is harder to find, just trial+error usually works as soon as you have some interactive prompt that echo's the characters you write
- Don't forget to add level shifter from 3.3/5V to RS232 levels

What's HaRET

What is HaRET

- a Windows executable program for any WinCE based OS
- offers a control interface on a TCP port
- connect to it using haretconsole (python script) on Linux PC
- supports a number of popular ARM based SoC (PXA, S3C, MSM)
- features include
  - GPIO state and tracing
  - MMIO read/write
  - virtual/physical memory mapping
  - IRQ tracing (by redirecting IRQ vectors)
  - load Linux into ram and boot it from within WinCE

Using HaRET

Using HaRET

- run the program on the target device
- connect to it using haretconsole over USB-Ethernet
- read GPIO configuration
  - Create GPIO funciton map based on SoC data sheet
- watch for GPIO changes
  - remove the signal from the noise
  - exclude unitneresting and frequently changing GPIOs
- watch for GPIO changes while performing certain events
  - press every button and check
  - start/stop peripherals
  - insert/eject SD card

Using HaRET

Using HARET

- watch for IRQ changes/events
  - e.g. you see DMA3 interrupts while talking to the GSM
  - read MMIO config of DMA controller to determine user: SPI
  - read SPI controller configuration + DMA controller configuration
  - find RAM address of data buffers read/written by DMA
- haretconsole writes logfiles
  - you can start to annotate the logfiles
- of course, all of this could be done using JTAG, too.
  - but with HaRET, you mostly don't need it!!!

Disassembling WinCE drivers

Disassmbling WinCE drivers

- is the obvious thing to do, right?
- is actually not all that easy, since
  - WinCE doesn't allow you to read the DLLs
    - not via ActiveSync neither WinCE filesystem API's
  - Apparently, they are pre-linked and not real files anymore
- luckily, there are tools in the 'ROM cooking' scene
  - hundreds of different tools, almost all need Windows PC
  - therefore, not useful to me
- conclusion: Need to understand the ROM image format

Disassembling WinCE ROM files

Disassembling WinCE ROM files

- 'datextract' to extract different portions like OS image
- 'x520.pl' to remove spare NAND OOB sectors from image and get a file
- split resulting image in bootsplash, cabarchive and disk image
- 'xx1.pl' to split cabarchive into CAB files
- 'partextract' to split disk image in partitions
- 'SRPX2XIP.exe' (wine) to decompress XPRS compressed partition0+1
- 'dumpxip.pl' to dump/recreate files in partition0 and 1
- 'ImgfsToDump.exe' to dump/recreate files in

Disassembling WinCE Drivers

Disassembling WinCE Drivers

- Now we finally have the re-created DLL's with the drivers
- Use your favourite debugger/disassembler to take them apart
- I'm a big fan of IDA (Interactive Disassembler)
  - The only proprietary software that I license+use in 15 years
  - There's actually a Linux x86 version
  - Was even using it with qemu on my Powerbook some years back

Disassembling WinCE Drivers

Important drivers

- pwrbtn.dll: the power button ?!?
- spkphn.dll: high-level device management
- i2c.dll: S3C24xx I2C controller driver
- spi.dll: The GSM Modem SPI driver
- Sergsm.dll: S3C24xx UART driver, NOT for GSM
- SerialCSR.dll: CSR Bluetooth driver
- fm_si4700.dll: The FM Radio (I2C)
- battdrvr.dll: Battery device (I2C)
- keypad.dll: Keypad+Keyboard+Capsense (I2C)
- GSPI8686.dll: Marvell WiFi driver (SPI)

Disassembling WinCE Drivers

Disassembling WinCE drivers

- Is typically hard, they're completely stripped
- Windows drivers are very data-driven, not many symbols/functions
- However, debug statements left by developers are always helpful
- After some time you get used to it
- You know your hardware and the IO register bases
  - take it from there, look at register configuration
- What I've learned about WinCE driver development
  - … would be an entirely separate talk
- MSDN luckily has full API documentation

WinCE Registry

WinCE has a registry, too

- I never really understood what this registry is all about, but it doesn't matter ;)
- You can use 'synce-registry' to dump it to Linux
- Contains important information about
  - how drivers are interconnected
  - various configuration parameters of drivers

Links

- http://wiki.openmoko.org/
- http://wiki.openezx.org/Glofiish_X800
- http://git.openezx.org/?p=gnufiish.git
- http://eten-users.eu/
- http://wiki.xda-developers.com/

Thanks

Thanks to

- The OpenEZX team that continues the project
- Openmoko, Inc. for trying to create more open phones
- Hunz for his jtagfinder
- xda-developers.org for all their work on WinCE tools
- eten-users.eu for the various ETEN related ROM cooking projects
- Willem Jan Hengeveld (itsme) for his M700 ROM tools
- Samsung, for having 100% open source driver for their SoC's
- Ericsson, for publishing the full AT command set for their modems