How I survived to a SoC with a terrible Linux BSP

Working with jurassic vendor kernels, missing pieces and buggy code

Luca Ceresoli
luca@lucaceresoli.net
`http://lucaceresoli.net`

FOSDEM 2017

About me

- Open source enthusiast
  - Contributor to Buildroot and a few other projects
- Embedded Linux engineer
  - Develop real products on custom hardware
  - Kernel, bootloader, drivers
  - Integration, build system

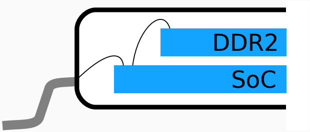
Introduction

Typical embedded Linux system

- A physical product
  - based on an ad-hoc electronic board
  - Built around a System-on-Chip (SoC)

The System on Chip

- Nuvoton N32926
  - Cheap
  - ARM926EJ-S @ 240 MHz
  - Peripherals: H.264 en/decoder, Ethernet MAC, USB, CMOS sensor interface, video out, LCD controller, sound, ...
  - 64 MB DDR2 *on package*
  - LQFP package
- Source: `https://www.nuvoton.com/hq/products/microprocessors/arm9-mpus/n3292-h.264-codec-series/n32926u1dn`



The ideal BSP

- BSP = Board Support Package
- The ideal BSP
  - Mainline kernel
  - Mainline U-Boot or Barebox
  - Good hardware documentation
- Why?
  - All standard, open-source components
    - Well known quality
    - Community and commercial support
    - Maintained (bugfixes!)
  - Reuse components
    - Infrastructure from other products
    - Any open source package (including those on your PC)

The Quest

The quest

- Documentation
- Linux kernel
- Toolchain
- Booting
- Tools
- Customer support

Documentation

Public documentation

- Website: https://www.nuvoton.com/hq/products/microprocessors/arm9-mpus/n3292-h.264-codec-series/n32926u1dn
- An 8-page datasheet (mostly a list of features)

Documentation for customers

- Only under NDA

Accessible documentation

- A “low-cost” devkit is available from chinese online stores
- Contains a DVD-ROM with a subset of the BSP for customers
  - Documentation and software
  - Contains the N3292x Design Guide
    - SoC peripherals (registers)

Linux kernel

Vendor kernel VS mainline kernel

Base kernel: Linux 2.6.35.4 (2010)

2.6.35.4 → 2.6.35.14
(latest stable)

- 11 months
- 1382 bugfix commits
- Merged with minimal conflicts

2.6.35.14 → 4.9
(latest mainline)

- A countless number of bugfixes, performance improvements, new features
- Security
- Device Tree

Vendor kernel additions

- Provided as patches:
    - `w55fa92-kernel-2.6.35-000.patch` (3.6 MB)
    - `w55fa92-kernel-2.6.35-001.patch` (1.4 MB)
    - `w55fa92-kernel-2.6.35-002.patch` (0.4 MB)
    - `do_kernel_patch.sh`
- Total: 170.000 lines changed

Vendor kernel issues

1. Bugs
2. Missing features
3. Code quality

Bugs

Examples:

- Sound Processing Unit ALSA driver
  - `arecord myfile.wav` $\rightarrow$ kernel crash
    - `NULL` pointer dereference
- H.264 decoder driver
  - Works with sample streams
  - Kernel crash on streaming packet loss
    - Several `NULL` pointer dereferences

Missing features

Examples:

- GPIO
  - Basic functionality is implemented
  - No interrupt handling
- Power Management
  - Implemented with a proprietary API
  - Also implemented the Linux standard way, but incomplete and not working

Code quality

- Average quality of additions: generally bad
- Trivial metric: $+521$ lines starting with `#if 0`
- A few examples follow

Code quality: driver model

`drivers/video/w55fa92_fb.c`:

```
#ifdef  CONFIG_GIANTPLUS_GPM1006D0_320X240
#include "w55fa92_GIANTPLUS_GPM1006D0.c"
#endif

#ifdef  CONFIG_TOPPLY_320X240
#include "w55fa92_TOPPLY_320x240.c"
#endif

/* ...5 more displays... */

#if 0
  #ifdef  CONFIG_SHARP_LQ035Q1DH02_320X240
  #include "w55fa92_Sharp_LQ035Q1DH02.c"
  #endif

  #ifdef  CONFIG_WINTEK_WMF3324_320X240
  #include "w55fa92_Wintek_WMF3324.c"
  #endif

  /* ...5 more displays... */
#endif
```

Code quality: H.264 codec memory allocation

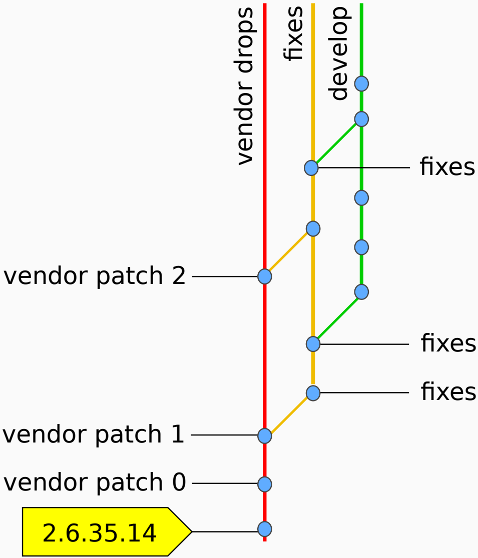
drivers/misc/codec264/favc_module.c:

```
unsigned int get_avc_buffer_size(void)
{
  /* ...~90 lines... */
  return TOTAL_VDE_BUF_SIZE;
}
EXPORT_SYMBOL(get_avc_buffer_size);
```

From arch/arm/mm/mmu.c:

```
extern unsigned int get_avc_buffer_size(void);
void __init reserve_node_zero(pg_data_t *pgdat)
{
  /* ... */
  buffer_size = get_avc_buffer_size();
  printk("AVC Buffer Size: 0x%x\n",buffer_size);
  w55fa92_vde_v = alloc_bootmem_low_pages (buffer_size);
  /* ... */
}
```

Versioning


vendor drops
fixes
develop
fixes
vendor patch 2
fixes
fixes
vendor patch 1
vendor patch 0
2.6.35.14

Toolchain

Vendor toolchain

- The BSP provides a toolchain.
  - Why?
- What's inside
  - gcc 4.2.1 (July 2007)
    - No C++11 support
    - gcc 4.2.x got fixes until 4.2.4 (May 2008)
  - uClibc 0.9.29 (2007)
    - What if I need glibc or musl?
    - Bugfixes and improvements in later versions?
  - A few other libraries (libcurl, libpng ...)
  - A hand-crafted script to install it at a hard-coded location

Toolchain selection

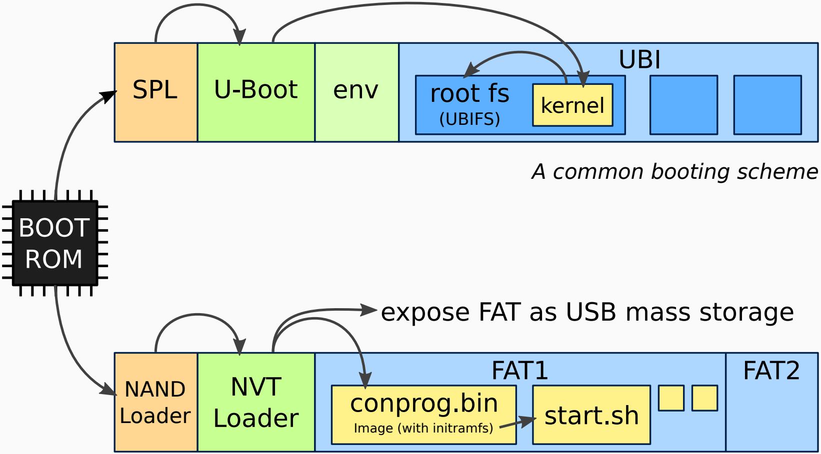
- Don't use the provided toolchain
- You could use a pre-built toolchain
  - If it has been built with kernel headers $<=$ 2.6.35
  - So it's probably quite old itself
- Build you own
  - crosstool-NG, Buildroot, Openembedded...

Booting

Bootloaders in the BSP

- No U-Boot
- No Barebox
- Some proprietary bootloaders
  - Sources provided
  - Not open source

Vendor booting scheme (NAND)



A common booting scheme

Vendor (demo) booting scheme

Vendor booting scheme pros

- Easy deployment of demos provided by vendor
  1. Press a button during boot
  2. Mount mass storage on PC
  3. Replace files

Vendor booting scheme issues /1

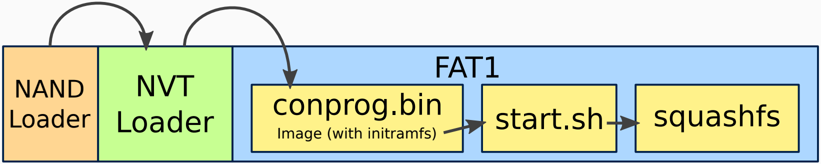
- FAT
  - Unreliable on power loss
  - It just cannot contain a UNIX-like rootfs (no users, groups, permissions, symlinks...)
- NAND FTL
  - FAT-on-NAND emulation (with FTL) is in a binary module
  - NVT Loader cannot mount UBIFS
- No provision for redundancy: one copy of each component

Vendor booting scheme issues /2

- Root filesystem is an initramfs
  - Changes are volatile
  - Limited size, everything in RAM
  - Persistent changes stay in FAT
- Nobody passes cmdline to kernel
  - it must be hard-coded in the kernel (`CONFIG_CMDLINE`)
- NFS booting
  - Needs cmdline parameters → must rebuild and reflash the kernel
- Cannot load kernel via TFTP

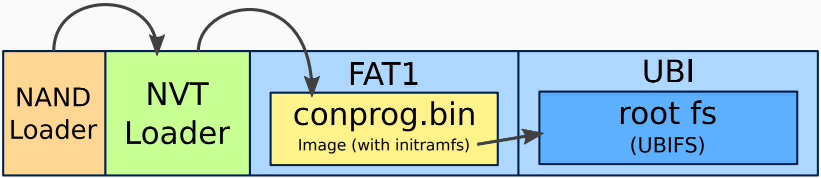
Alternative booting options?

Option 1: add a SquashFS layer on top of FAT



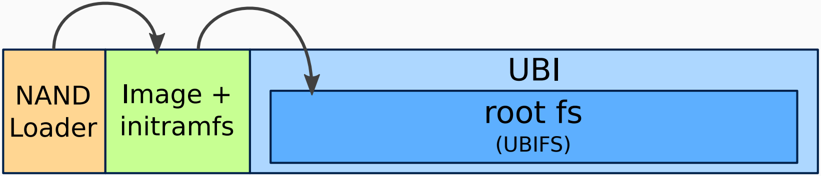
- Keep the existing structure untouched
- Remove FAT space constraint and RAM usage
- Still read-only
  - ext2 or any other rw filesystem over FAT over NAND is crazy
- The device cannot atomically upgrade itself

Option 2: jump from FAT to UBIFS



- UBI and UBIFS are designed for NAND! (efficient, reliable, quite scalable)
- Tweaks needed
  - Change the initramfs `/init` to mount UBIFS and `switch_root`
  - Tweak NVT Loader not to use all space for FAT
- USB mass storage can only update kernel
- FAT area atrophied, NVT Loader almost useless

Option 3: skip NVT Loader



- NAND Loader loads kernel `Image` to address `0` and jump there
- No more NVT Loader and FAT
  - Less code, less bugs, faster boot, more free space
- Kernel still on bare NAND and without cmdline
- Safe kernel upgrade?
  - kexec

Option 4: Port U-Boot

- Port U-Boot or Barebox to the SoC
  - Maybe keeping the vendor NAND Loader (SPL)
- Unleashes all the known advantages
  - Environment, boot-time scripting, prompt, cmdline, TFTP boot...
  - Redundancy for all/most components on bare NAND
- Time to market?

Tools

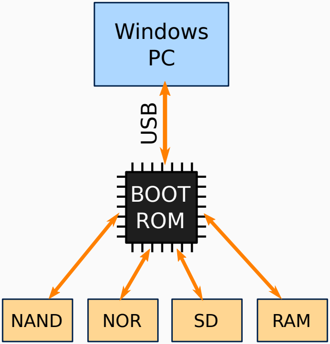
Tools

- Which tools do I need?
  - Ideally, none
- Flashing an empty memory is different
  - Some vendors have proprietary tools

Flashing tools

- Tool provided to write memory
- Quite flexible
  - Can write NAND, SPI, SD, SDRAM (and execute)
  - Over USB
- For Windows only
- Proprietary
- GUI, not scriptable
- Protocol to Boot ROM not documented

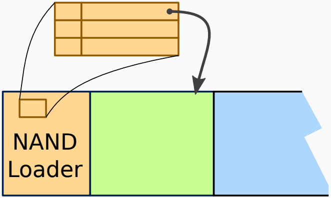
→ You're locked to it



NAND partition table

- Proprietary partition table in the NAND Loader area
- The proprietary tool writes only this format
- Not a bad idea
  - but standard tools work differently

→ You can't get rid of the table



Customer support

Customer support

- Standard, mainline code
    - Plenty of options for community and commercial support
- Proprietary code with sources
    - Vendor support
    - Read the code
- Proprietary, binary software (and hardware issues)
    - One choice: vendor support
    - Still acceptable if support is good
        - But don't bet it will be

Customer support issues

- The engineer who knows the answer is hidden by reseller sales dept., FAE, customer support department
- Responsiveness
- Timezone issues

Customer support — a bad example

A real conversation (short form)

Me The proprietary tool doesn't work

CS Works on my PC, see screenshot

Me Not on mine; can it log errors so you can diagnose it?

CS Adding logging would not be practical

Concluding remarks

The result

Comparison with a well-supported SoC

- The product works
  - Final quality is lower
  - The hardware would allow to do better
- Extra time spent
  - Sometimes we supported ourselves
  - Look for stuff in the BSP
  - Fix bugs
  - Reinvent booting

What can I do to improve things?

What can I do to make a better world?

- As an embedded Linux engineer
    - Assess potential problem early while evaluating a SoC
        - Especially booting and hardware support
- As a hobbyist or a hacker
    - Pick boards with good mainline support, or...
    - Improve existing support and mainline it

What can vendors do to ship better BSPs?

- Happy engineer = good product = more sales
- Don't reinvent the wheel
- Write good docs, no NDA, no registration
  - Including yout Boot ROM protocol
    - And let people write the tools they want
- Push your code to mainline
  (or outsource this to a specialized company)
  - Expensive, but rewarding
  - Somebody else will update, fix, improve and support it
- Leverage the community
  - Let your engineers use mailing-lists, IRC etc
  - Make cheap, hacker-friendly boards

Thank you for your attention

Questions?

luca@lucaceresoli.net

http://lucaceresoli.net

Extra Slides

Using an old gcc — an example

A C++ program using libconfuse 3.0

```
#include <confuse.h>
//...
cfg_opt_t opts[] =
{
  CFG_STR("my-param", "defval", CFGF_NONE),
  CFG_END()
};
```

With gcc <= 4.8 fails building due to designated initializers not being implemented:

```
error: expected primary-expression before '.' token
```

Kernel code quality — example 1

Changes to `Makefile`:

```
-ARCH?= $(SUBARCH)
-CROSS_COMPILE?=
-CROSS_COMPILE?= $(CONFIG_CROSS_COMPILE:''%''=%)
+#ARCH?= $(SUBARCH)
+ARCH= arm
+#CROSS_COMPILE?=
+#CROSS_COMPILE?= $(CONFIG_CROSS_COMPILE:''%''=%)
+CROSS_COMPILE= arm-linux-
```

- Prevents using toolchains with a different prefix
- Any advantage?

Kernel code quality — example 2

Changes to `arch/arm/boot/Makefile`:

```
 $(obj)/Image: vmlinux FORCE
        $(call if_changed,objcopy)
        @echo '  Kernel: $@ is ready'

+ifeq ($(CONFIG_ARCH_W55FA92),y)
+        cp $@   ../image/conprog.bin
+endif
```

- `../image/` does not make sense in any buildsystem

Kernel code quality — example 3

`sound/soc/w55fa92/w55fa92_spu.c`:

```
if (nChannels ==1)
{
    DrvSPU_EnableInt(_u8Channel0, DRVSPU_ENDADDRESS_INT);
    DrvSPU_EnableInt(_u8Channel0, DRVSPU_THADDRESS_INT);
}
else
{    /* just open one channel interrupt */
    DrvSPU_EnableInt(_u8Channel0, DRVSPU_ENDADDRESS_INT);
    DrvSPU_EnableInt(_u8Channel0, DRVSPU_THADDRESS_INT);
}
```

- Find the differences between the *then* and the *else* branch

Kernel code quality — example 4

`sound/soc/w55fa92/w55fa92_spu.c`:

```
static int DrvSPU_EnableInt(u32 u32Channel, u32 u32InterruptFlag)
{
  if ( (u32Channel >=eDRVSPU_CHANNEL_0) && (u32Channel <=eDRVSPU_CHANNEL_31) )
  {
    /* ... */
    if (u32InterruptFlag & DRVSPU_USER_INT)
    {
      AUDIO_WRITE(REG_SPU_CH_EVENT, AUDIO_READ(REG_SPU_CH_EVENT) | EV_USR_EN);
    }
    if (u32InterruptFlag & DRVSPU_SILENT_INT)
    {
      AUDIO_WRITE(REG_SPU_CH_EVENT, AUDIO_READ(REG_SPU_CH_EVENT) | EV_SLN_EN);
    }
    /* ...a few more times... */

    /* ... */
    return E_SUCCESS;
  }
  else
    return E_DRVSPU_WRONG_CHANNEL;
}
```

Kernel code quality — example 5

arch/arm/mach-w55fa92/include/mach/w55fa92_gpio.h:

```
static inline int w55fa92_gpio_configure(int group, int num) {
  /* ... */
    case GPIO_GROUP_B:
      if(num <= 7)
        writel(readl(REG_GPBFUN0) &~ (0xF << (num<<2)), REG_GPBFUN0);
      else
        writel(readl(REG_GPBFUN1) &~ (0xF << ((num%8)<<2)), REG_GPBFUN1);
      break;

    case GPIO_GROUP_C:
      if(num <= 7)
        writel(readl(REG_GPCFUN0) &~ (0xF << (num<<2)), REG_GPCFUN0);
      else
        writel(readl(REG_GPCFUN1) &~ (0xF << ((num%8)<<2)), REG_GPCFUN1);
      break;
  /* ...similarly fo other GPIO ports... */
}
```

- A little refactoring would help