The Bootlin ~~War stories~~ Debugging Chronicles

The Bootlin Team

Embedded Recipes 2026

~~whoami~~whoarewe

- Engineering company specialized in **Embedded Linux** and **Zephyr**
- **Engineering** services and **training** services
  - Strong upstream focus
  - Freely available, open-source training materials
  - New **Embedded Linux Security** training course
- Contributors to
  - Linux kernel, U-Boot, Yocto Project, Buildroot, and more
- Authors/maintainers of
  - `elixir.bootlin.com`
  - Snagboot
  - sbom-cve-check
- Proud to be a **Chef Sponsor** of Embedded Recipes
- Team of **28**
  - We're **hiring**!

This talk

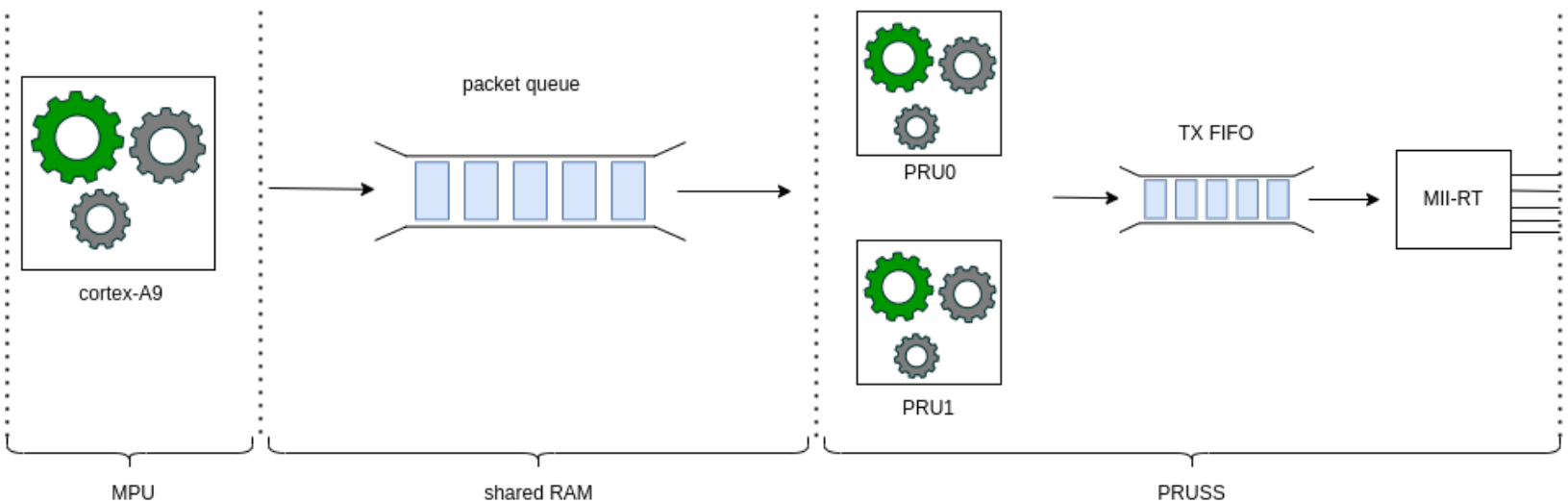
- A selection from the Bootlin **Debugging Chronicles**
  - We have 6 chronicles in the slides
  - But we'll stop when we run out of time
  - Remaining chronicles (and more!) might be covered in a follow-up edition of this talk :)
- What broke, why it broke, how we tracked it down, and what it taught us

Debugging Chronicle 1: Blame it on the firmware

Presented-by: Romain Gantois <romain.gantois@bootlin.com>

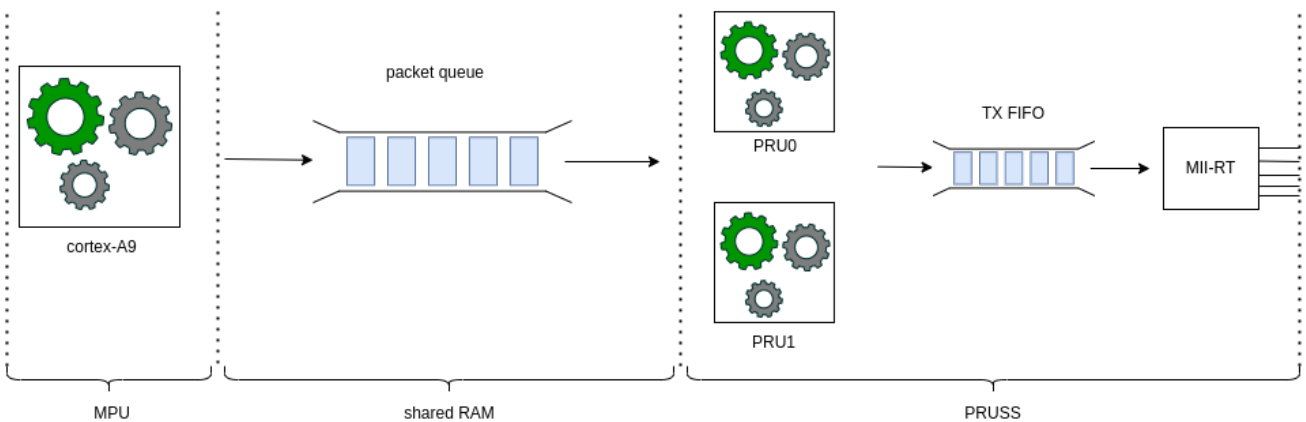
Initial problem statement

- Customer had a product based on an AM437x SoM
- Malformed packets were observed on a PRU Ethernet interface



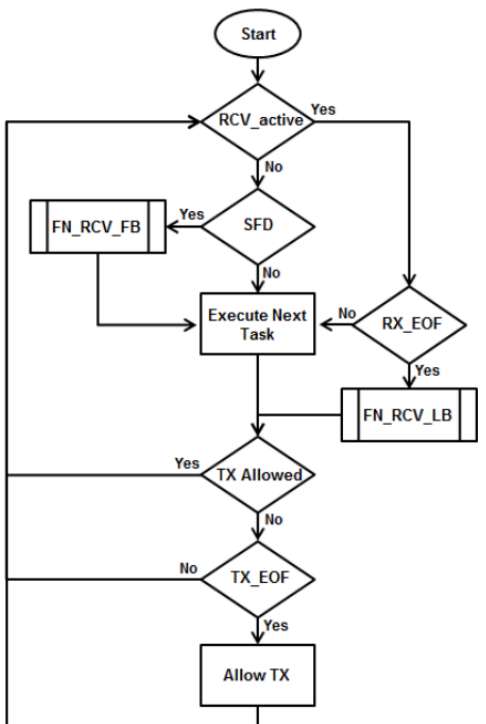
Initial investigation

- Reproduced the issue and characterized it further
- As it turns out, packets under a certain size were truncated and resent multiple times
- Initial troubleshooting ruled out a bug in the Ethernet driver



PRU investigation

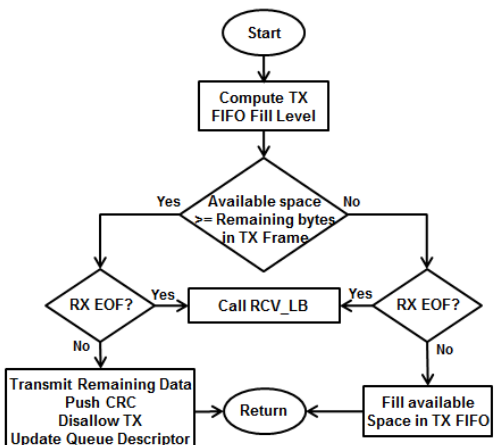
- Found sources of the PRU firmware
- Rebuilt a firmware image from source
- Set up tracing methods



credits: Texas Instruments, ICSS DUAL EMAC FIRMWARE DESIGN GUIDE

PRU investigation

- Checked control flow in PRU firmware
- Found some strange TX FIFO overlow events



credits: Texas Instruments, ICSS DUAL EMAC FIRMWARE DESIGN GUIDE

- Turns out there was a bug in the code computing the TX FIFO fill level

FIFO fill level computation

1. read IEP counter which indicates the number of elapsed clock cycles.
2. subtract SOF (Start Of Frame) previous counter value to the current counter value to get the cycles elapsed since the start of the transmission.
3. divide by 80 to obtain number of packets sent by MII_RT hardware (the clock rate is 200MHz, the counter increment value is 5, and the bitrate is 100Mbps) are sent each clock cycle).
4. subtract this value from the number of bytes pushed to the FIFO to get the current FIFO fill level.
5. check if the result is larger than 64, which would indicate an overflow.

Final diagnosis

- The timestamp from the IEP wasn't written to the expected register before computing the fill level
- In the case of a rev. 1 PRU IP in EMAC mode, the SOF counter value was never stored, which meant that the computed value of elapsed time was wrong, and subsequently the computed FIFO fill level was incorrect. This explains why there were FIFO fill errors for each transmitted packet. The smaller packets were not hindered by this, probably because they were completely sent anyway by the time the FIFO level check took place.

Proposed fix

```
--- emac_MII_Xmt.asm
+++ emac_MII_Xmt.asm
@@ -915,6 +915,13 @@ XMT_LB_100Mbps_MODE:
     ;Detect FIFO overflow and reset
     .if $defined("ICSS_REV1")
+
+    .if $defined("PRU0")
+        LBCO      &TEMP_REG_3, IEP_CONST, CAP_RISE_TX_SOF_PORT1_OFFSET, 4
+    .else
+        LBCO      &TEMP_REG_3, IEP_CONST, CAP_RISE_TX_SOF_PORT2_OFFSET, 4
+    .endif
+
     M_XMT_FILL_LEVEL_CALC_ICSS_REV1         ;calculates Fill level and
resets in case of underflow/overflow
     .endif
```

Conclusions and followup

- Informed customer and TI PRU firmware team of our findings
- Concerted with TI for final fix
- TI integrated fix into PRU firmware sources for AM437x

Debugging Chronicle 2: A stupid hack bites you back!

Presented-by: Thomas Petazzoni <thomas.petazzoni@bootlin.com>
Debugged-with: Hervé Codina <herve.codina@bootlin.com>
Debugged-with: Alexis Lothoré <alexis.lothore@bootlin.com>

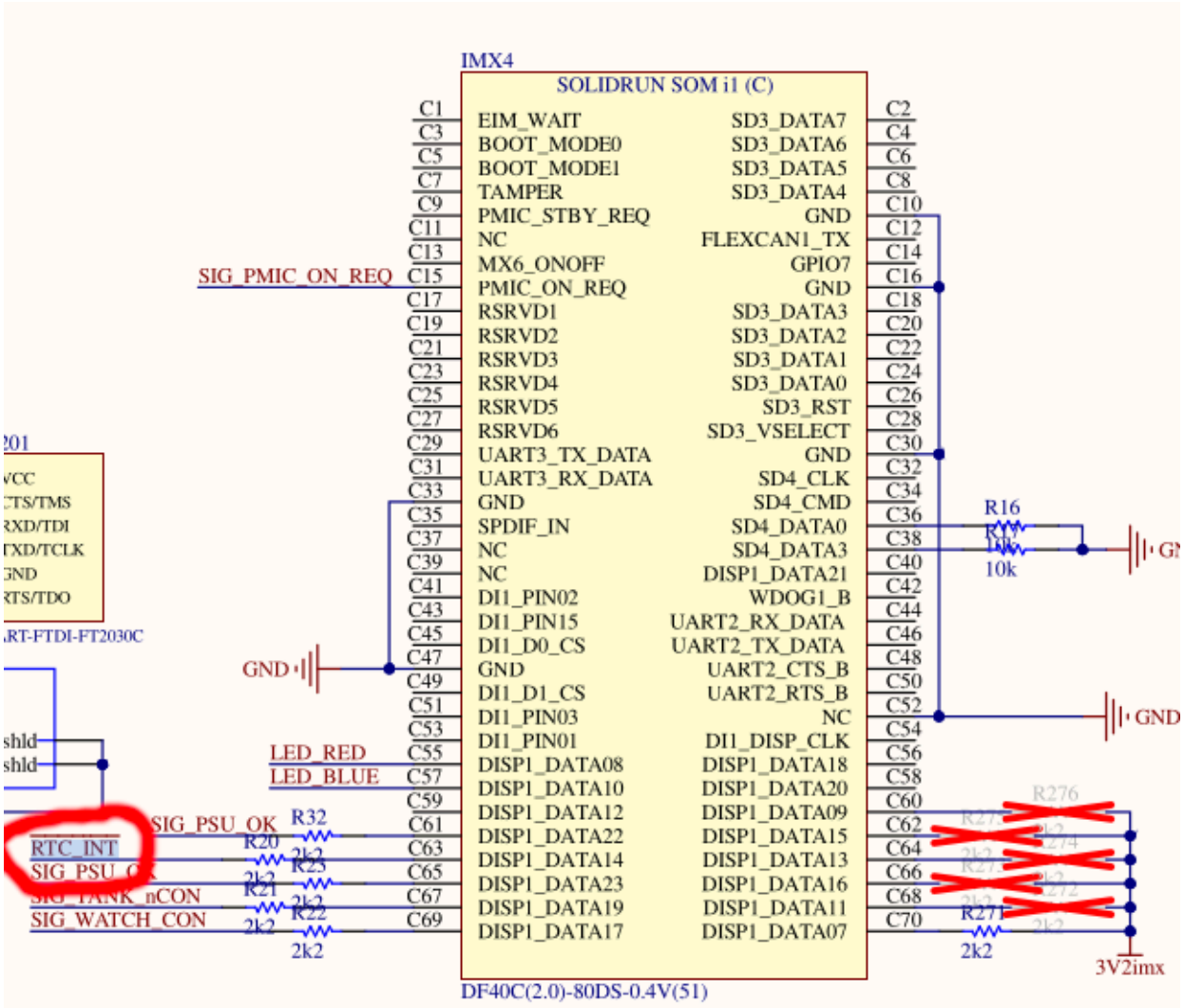
Initial problem statement

- Customer has a product based on an i.MX6 SoM
- Customer claims *"our SoM vendor has been shipping new SoMs, on which the Ethernet PHY is different from previous SoMs and the new SoMs are ≈2x slower"*
- Us: WTF
- Customer: *"in the end, we found old SoM that also exhibit the same ≈2x performance hit"*
- Us: ah, this is already a bit less crazy
- Customer sends us "fast" and "slow" SoMs, and we begin the investigation

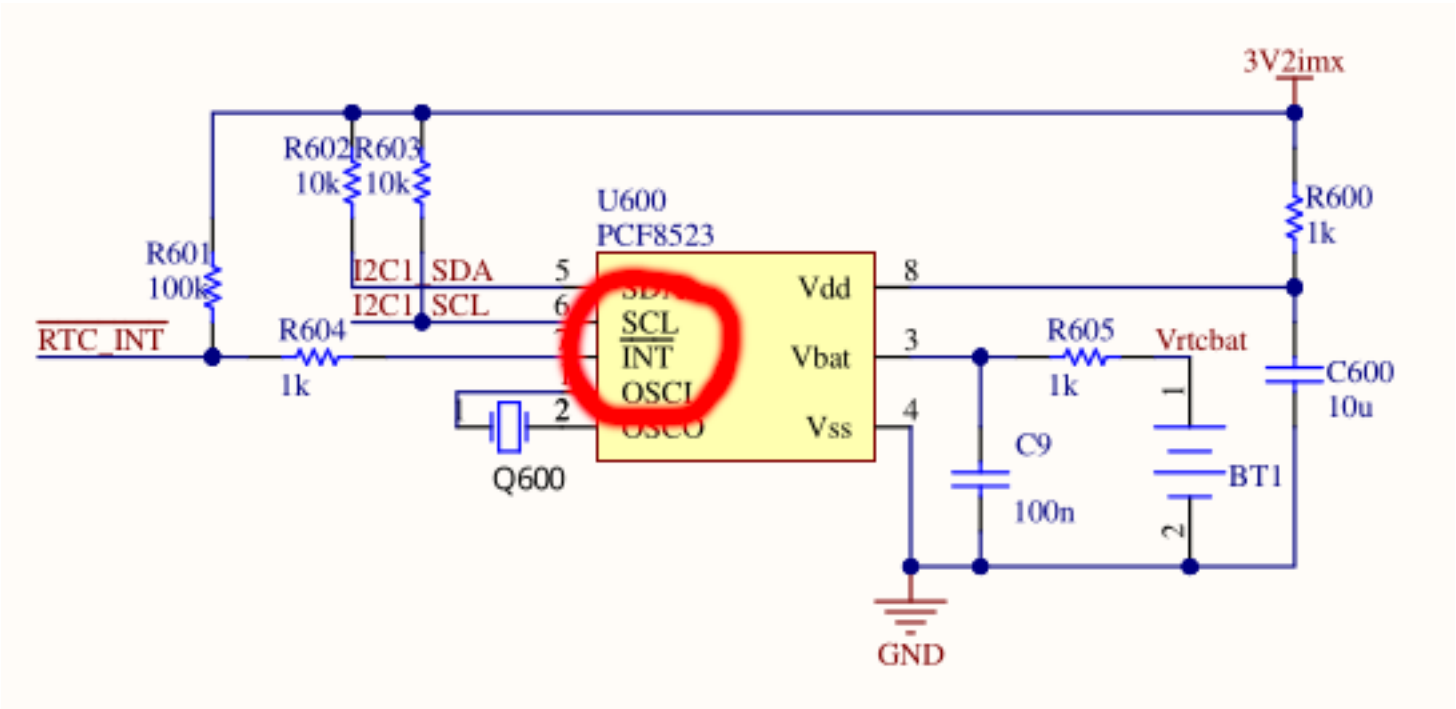
Initial investigation

- A very simple CPU-bound application is indeed ≈2x slower
- What else is using the CPU?
  - `top` doesn't show anything suspicious: no other process is obviously hogging the CPU
- If nothing is hogging the CPU, is the CPU running at a lower frequency?
  - `cpufreq` seems to be configured / working correctly
  - Switching to the `performance` governor doesn't make any difference
- So there must be something else hogging the CPU
  - Interrupts?
  - `watch -n1 cat /proc/interrupts`
  - And indeed, one interrupt line has **10000s** of interrupts per second on the "slow" SoMs, while the "fast" SoMs are not affected
  - This interrupt line is a GPIO interrupt

OK, schematics now!

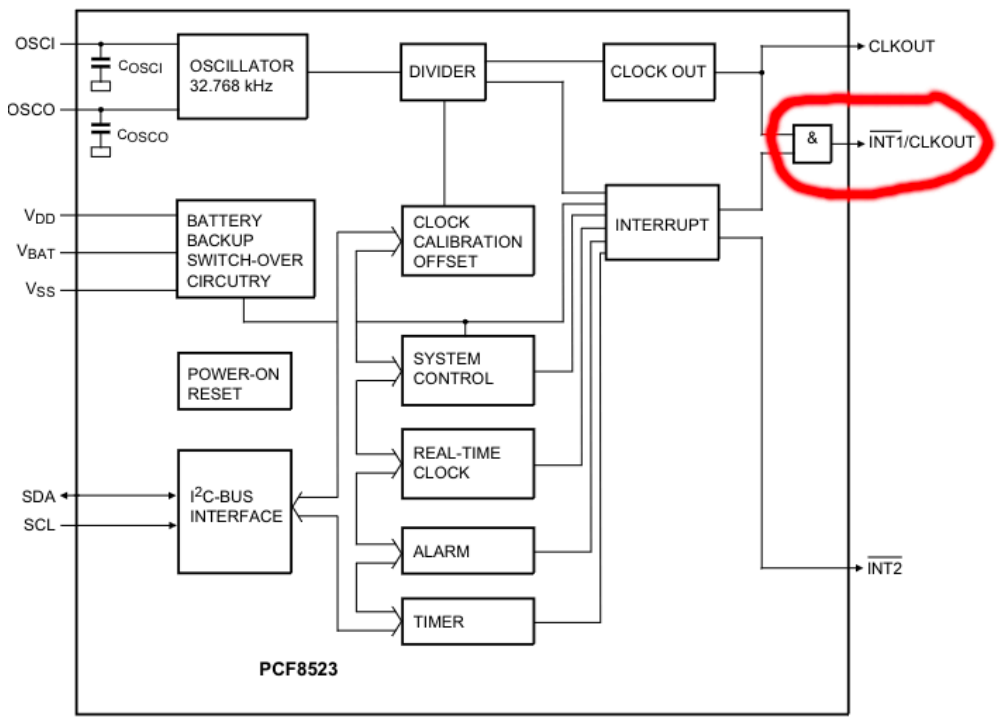


OK, schematics now!



RTC interrupt

- So the RTC is hammering us with **10000s** of interrupts per second
- How does that even make sense?
- RTC datasheet to the rescue



RTC interrupt or clkout ?

- So, perhaps we're not getting hammered by interrupts but by the clock generator?

After reset, the following mode is entered:

- 32.768 kHz CLKOUT active
- 24 hour mode is selected
- Register Offset is set logic 0
- No alarms set
- Timers disabled
- No interrupts enabled
- Battery switch-over is disabled
- Battery low detection is disabled
- 7 pF of internal oscillator capacitor selected

- Ah, ah, out of reset, `CLKOUT` is **active**, at 32,768 Hz, nicely matches the number of interrupts we're seeing!
- But then why are some SoMs getting hammered by this clock signal and not others?

drivers/rtc/rtc-pcf8523.c

```
static int pcf8523_probe(struct i2c_client *client)
{
        [...]
        if (client->irq > 0) {
                [...]
                err = regmap_write(pcf8523->regmap, PCF8523_TMR_CLKOUT_CTRL, 0x38);
                if (err < 0)
                        return err;
                [...]
}
```

```
$ bindump 0x38
0000 0000 0000 0000 0000 0000 0011 1000
--28 --24 --20 --16 --12 ---8 ---4 ---0
```

Back to the datasheet

8.9.1.1 Register Tmr_CLKOUT_ctrl and clock output

Table 34. Tmr_CLKOUT_ctrl - timer and CLKOUT control register (address 0Fh) bit description

Bit	Symbol	Value	Description
7	TAM	0[1]	permanent active interrupt for timer A and for the second interrupt timer
		1	pulsed interrupt for timer A and the second interrupt timer
6	TBM	0[1]	permanent active interrupt for timer B
		1	pulsed interrupt for timer B
5 to 3	COF[2:0]	see Table 35	CLKOUT frequency selection
2 to 1	TAC[1:0]	00[1] to 11	timer A is disabled
		01	timer A is configured as countdown timer if CTAIE (register Control_2) is set logic 1, the interrupt is activated when the countdown timed out
		10	timer A is configured as watchdog timer if WTAIE (register Control_2) is set logic 1, the interrupt is activated when timed out
0	TBC	0[1]	timer B is disabled
		1	timer B is enabled if CTBIE (register Control_2) is set logic 1, the interrupt is activated when the countdown timed out

Table 35. CLKOUT frequency selection

COF[2:0]	CLKOUT frequency (Hz)	Typi
000[2]	32768	60 :
001	16384	50 :
010	8192	50 :
011	4096	50 :
100	1024	50 :
101	32	50 :
110	1	50 :
111	CLKOUT disabled (high-Z)	

So, what's going on?

- The driver has logic to disable the `CLKOUT` signal when the interrupt is used
  - Prior to enabling the interrupt
- OK, so let's look at the *Device Tree*

```
gpio-keys {
        compatible = "gpio-keys";

        pcf8523 {
                label = "RTC alarm";
                gpios = <&gpio2 19 GPIO_ACTIVE_LOW>;
                linux,code = <143>;
                wakeup-source;
        };
};

&i2c1 {
        pcf8523: rtc@68 {
                compatible = "nxp,pcf8523";
                reg = <0x68>;
        };
}
```

So what's going on?

- Customer is not using the `interrupts` property of the `rtc` node, but a separate `gpio-keys`
- Why?
- After asking
  - *"Because we needed to configure it as a wake-up source and the PCF8523 driver didn't support that"*
- But of course, this means the RTC driver does not disable `CLKOUT`
  - While `gpio-keys` enables the GPIO as an interrupt signal
  - Flooding us of interrupts
- OK, so now we understand why we are flooded on the *slow* SoMs, but not why we also have *fast* SoMs

Why do we have *fast* and *slow* SoMs?

- Registers on the RTC are battery-backed!
- So if one day you boot a system that disables `CLKOUT`, it remains disabled forever
  - Even if you boot a new kernel that doesn't explicitly disable it, like what happens with our current DT
- So the story went like this:
  - The customer initially used Bootlin's original Device Tree that was properly describing the RTC interrupt
  - Deployed this on many units, things worked fine
  - Then on their own wanted to use the RTC interrupt as a wake-up source and changed the DT to use `gpio-keys`
  - Worked perfectly fine on existing units since the RTC battery preserved the `CLKOUT` disabled situation
  - But caused all newly flashed units to be "slow"
- Confirmed with: removing the RTC battery from a "fast" unit would make it a "slow" unit, as `CLKOUT` was no longer disabled

Conclusion

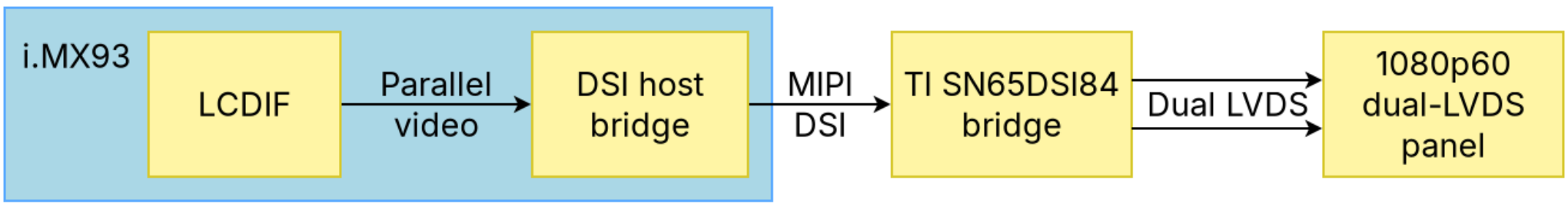
- When playing with RTCs, do some tests from a well known RTC state
  - Remove the battery, to make sure the RTC is really reset
- Don't do hacks, they come bite you later!

Debugging Chronicle 3: Just give me pixels

Presented-by: Luca Ceresoli <luca.ceresoli@bootlin.com>

Initial problem statement

- Customer has a product based on i.MX93
- All peripherals working, on the NXP 6.6-based kernel
- Except the LCD panel

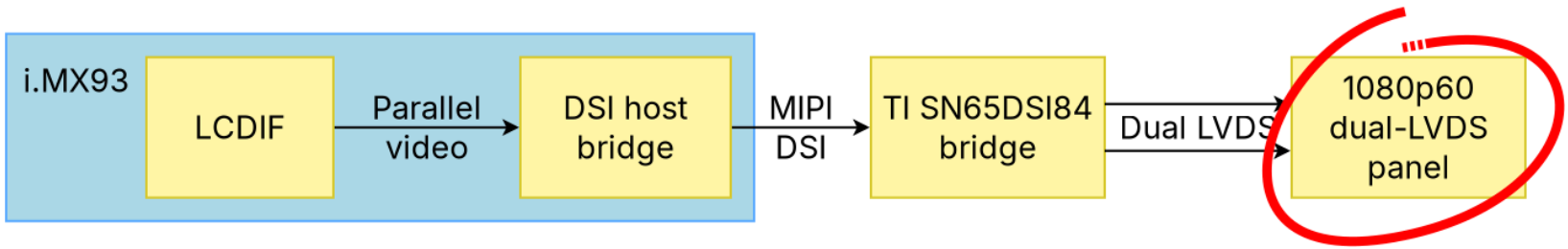


- The same panel works on a different product, based on x86-64

A look at the code

- Upstream is missing the DSI host → cannot be used easily
- The TI SN65DSI84 driver has fixes upstream after 6.6
  - Not easy to apply
- The customer kernel is:
  - NXP fork of 6.6 (2 years old)
  - plus a huge patch by the customer with hacks from their attempts at enabling the panel
- Split the huge patch in individual changes

The panel

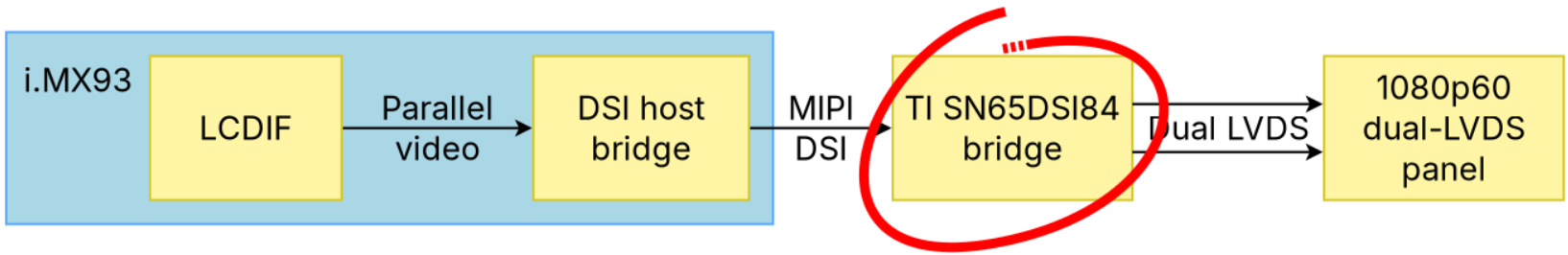


- Two *different* timings
  - In the datasheet
  - In an EDID file accompanying the panel
    - This is what the x86-64 product uses
- Tried both, still no image
- Format unclear (JEIDA or VESA?)
  - Irrelevant until there's a picture, postponed

The usual suspects

- Check clock configuration
- Check signals with an oscilloscope
  - No test points
  - Had to scratch solder mask to put probes on vias
  - DSI data looked OK
  - Could not check DSI clock, too fast for my scope
  - LVDS lines looked reasonable
- Check pinmux (none)
- Check resets
- Check power management
  - Disable all PM features

SN65DSI84 test pattern



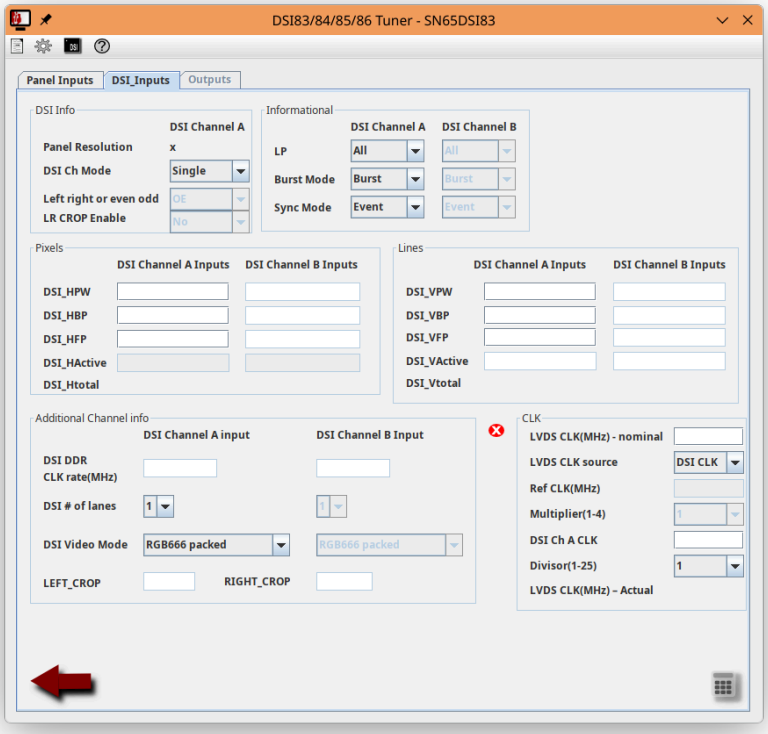
- The SN65DSI84 can generate a test pattern
- Not implemented by the driver, but just one bit to set!
- Still a black screen

DSI-Tuner 1/3

- Tool provided by TI
- https://www.ti.com/tool/DSI-TUNER
- Computes register values given all the parameters (clocks, panel timings etc)

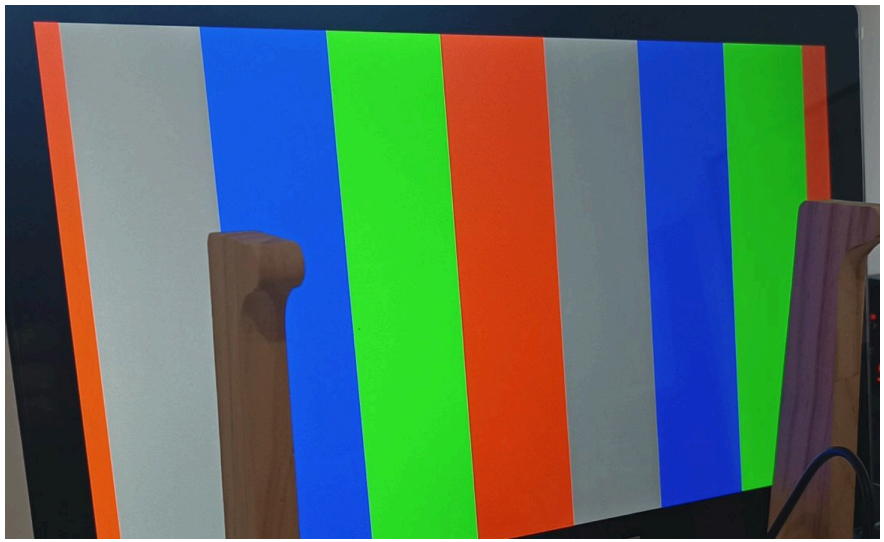
DSI-Tuner 2/3

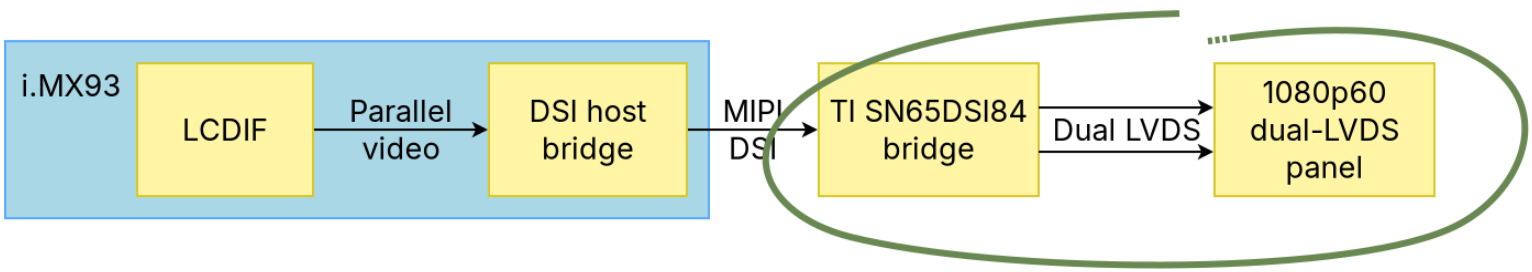
- DSI-Tuner is written in Java (portable!)
  - But it is provided as a .MSI installer for Windows
  - And the installer needs a specific and old Java version
  - Can be downloaded from `sun.com`
  - Which is a dead link
- Solution (all on Linux):
  - Extract the .MSI with 7zip
  - It contains a JAR file!
  - `java -jar 'DSI Tuner.jar'`



DSI-Tuner 3/3

- DSI-Tuner has lots of parameters, not all obvious
- But after some attempts...
- Test pattern working!



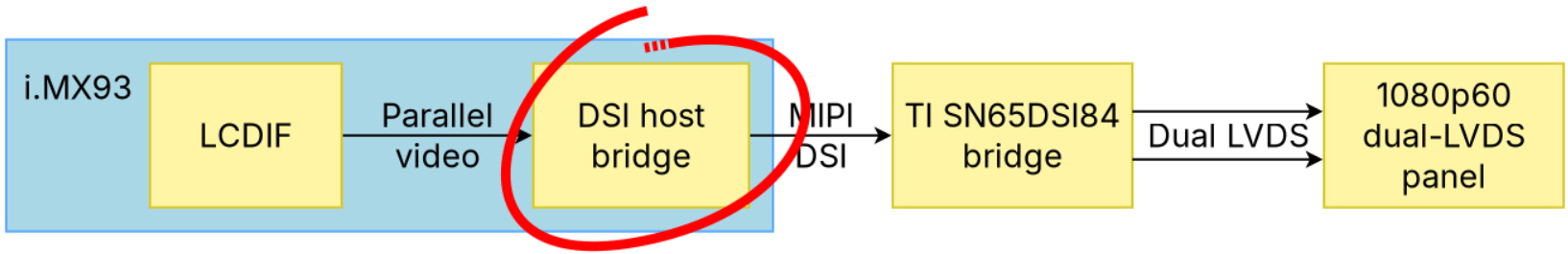


- But disabling test pattern, still a black screen ☹

SN65DSI84 driver fixes and improvements

- DSI-Tuner allowed to find bugs in the driver
  - Some registers need to be halved for dual-LVDS
    - Nobody noticed before?
    - Perhaps many other panels tolerate incorrect timings
  - Some need to be divided *only* to generate the test pattern
  - `CHA_DSI_CLK_RANGE` was slightly incorrect (no visible effect on my panel)
- Upstreaming
  - Fixes merged in kernel v7.0
  - Test pattern queued for v7.2

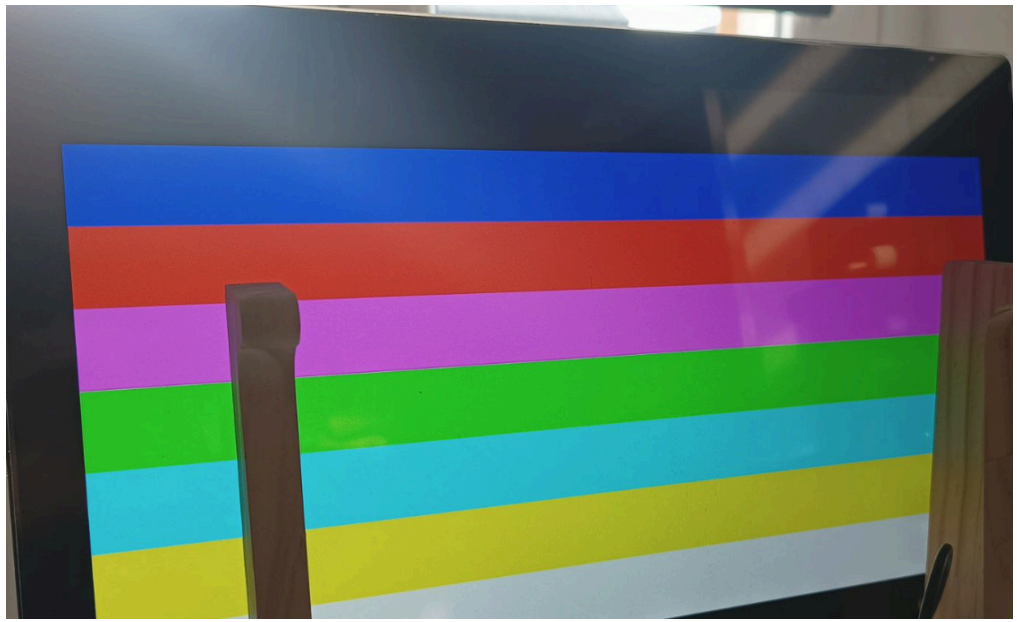
i.MX93 MIPI DSI host bridge 1/2

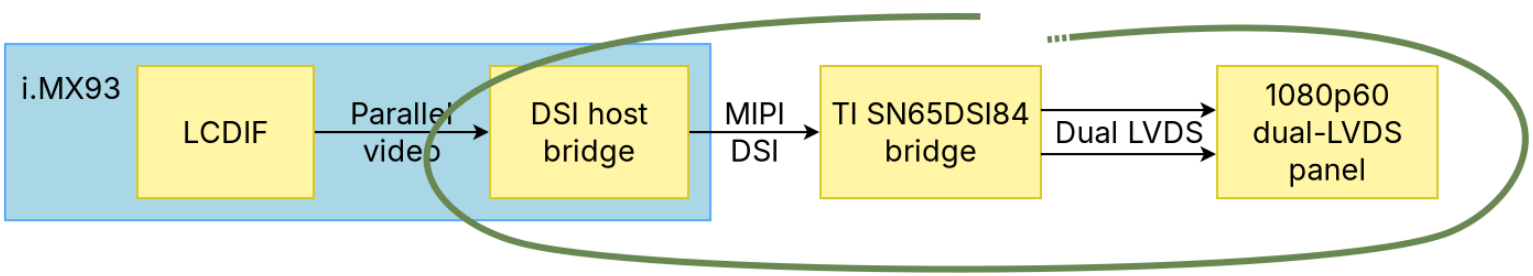


- Read docs
- Check all registers
- Enable test pattern (already implemented by driver)
- Check all the clocks (again)
  - All pixel-related clocks have correct values with good margin
  - All data busses have a clock fast enough for the needed bandwidth, with large margin
- The “DPI Payload Write Error” bit goes high
  - Suggests the DSI host is unable to buffer enough data from the LCDIF → underrun
  - Tried increasing the AXI bus clock, but to no avail

i.MX93 MIPI DSI host bridge 2/2

- Check all the clocks (again[2])
- Many desperate attempts later...
- Remove `assigned-clock-rates` from the board device tree
- DSI host test pattern appeared!





After Joy...

- But at the next reboot...

  ```
  [   21.778214] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
  ```

  Kernel hangs during boot, $>$ 50% of the times

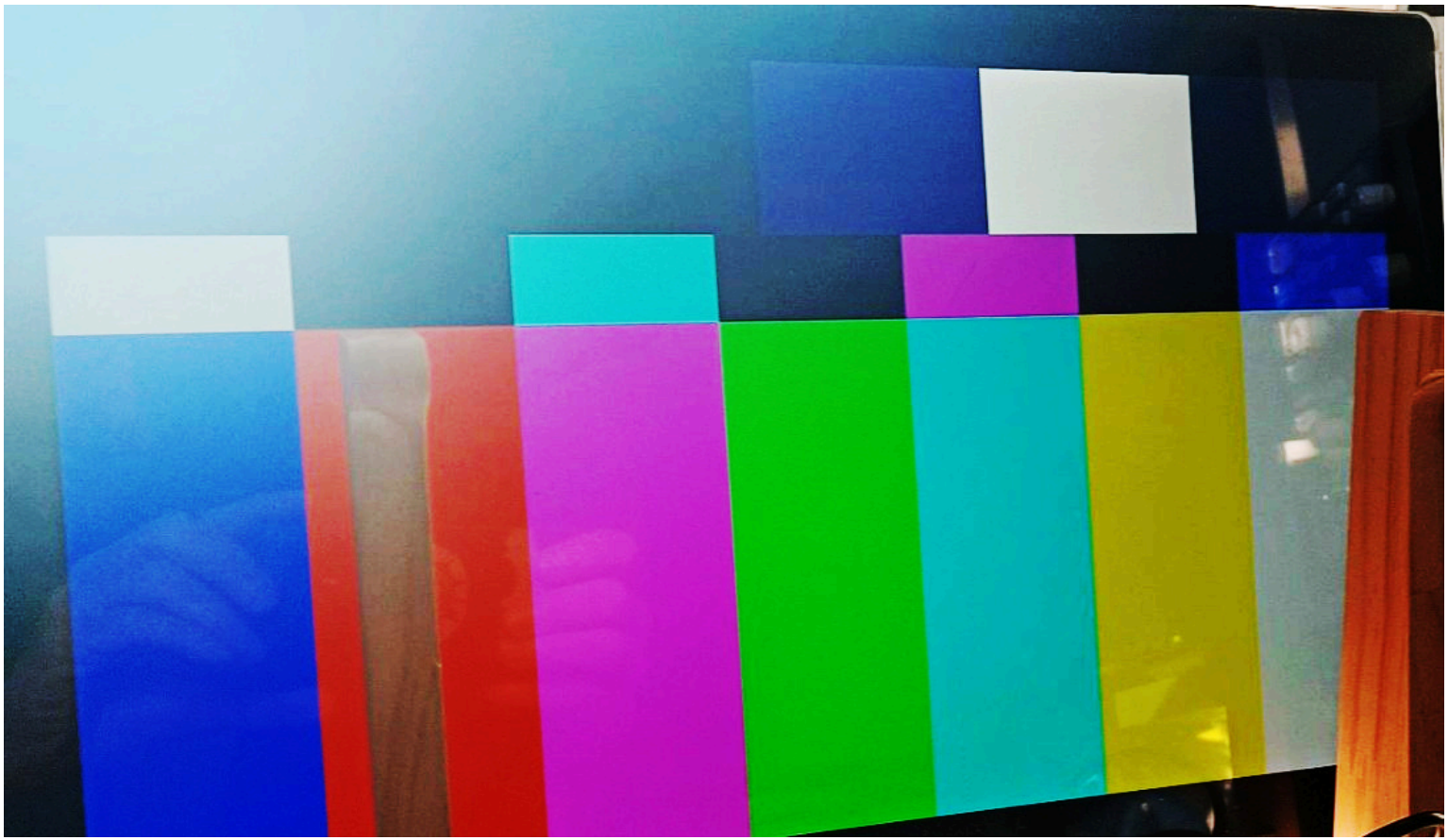
Back to clocks (*it's always the clocks*)

- Take a deep breath...
- Examine the removed `assigned-clock-rates` one by one

  `assigned-clock-rates = <437100000>, <142380000>, <400000000>, <133333333>;`
  - `nic_media` clock increased from 400 MHz to 800 MHz
    - But it causes the kernel hang! → change reverted
  - Pixel clock got *very slightly* more accurate
    - This *is* actually relevant! → change kept

And finally...

- Disabling all test patterns...



Time for cleanups

- Remove lots of temporary commits...
  - Logging
  - Desperate man's `msleeps()`
  - Various hacks tried along the way
  - Each removed as an individual commit (*)
- At the end: screen is black, again 😯
- Bisected thanks to (*), found the relevant commit

The last piece

- The last change needed to fix the panel is one of the many changes in the initial patch from the customer
- Disable DSI burst mode

```
@@ -665,7 +665,7 @@ static int sn65dsi83_host_attach(struct sn65dsi83 *ctx)
-       dsi->mode_flags = MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST |
+       dsi->mode_flags = MIPI_DSI_MODE_VIDEO |
```

What's wrong with burst mode?

- MIPI DSI burst mode allows to reduce power consumption
  - Data is sent in bursts, to leave "idle" time with clock disabled
- But "The SN65DSI84 *supports burst video mode* and non-burst video mode"
  - Source: SN65DSI84 datasheet
- So, why disabling burst mode fixes the panel?
  - I'm in contact with TI to clarify
  - Meanwhile another user sent a patch disabling burst mode (+ another change) to fix the same issue on a different panel:
    - `drm: bridge: ti-sn65dsi83: Fix DSI mode flags for stable LVDS output`
    - Currently under discussion
- Reason still unclear. Non-burst mode is just "safer"?

Takeaways

- Video pipelines are complex, each component is a SPoF, limited observability
- Having a known-working setup (no matter how ugly) is gold
- Read documentation
- Don't trust documentation
- Vendor tools can be useful
- Desperate? Some guesswork helps, but don't rely just on it!
- Baby steps: don't be too ambitious, get a small bit working at a time
- Suggested strategy (rule of thumb):
  1. Get the basics right: clocks, pinmuxes and resets
  2. Start with the tail: panel timings, the go back one component a time
  3. Use test patterns to enable the last leg; then go backwards, one bridge at a time
- Do small commits and keep them

Debugging Chronicle 4: When Fresh DRAM Breaks Old Assumptions

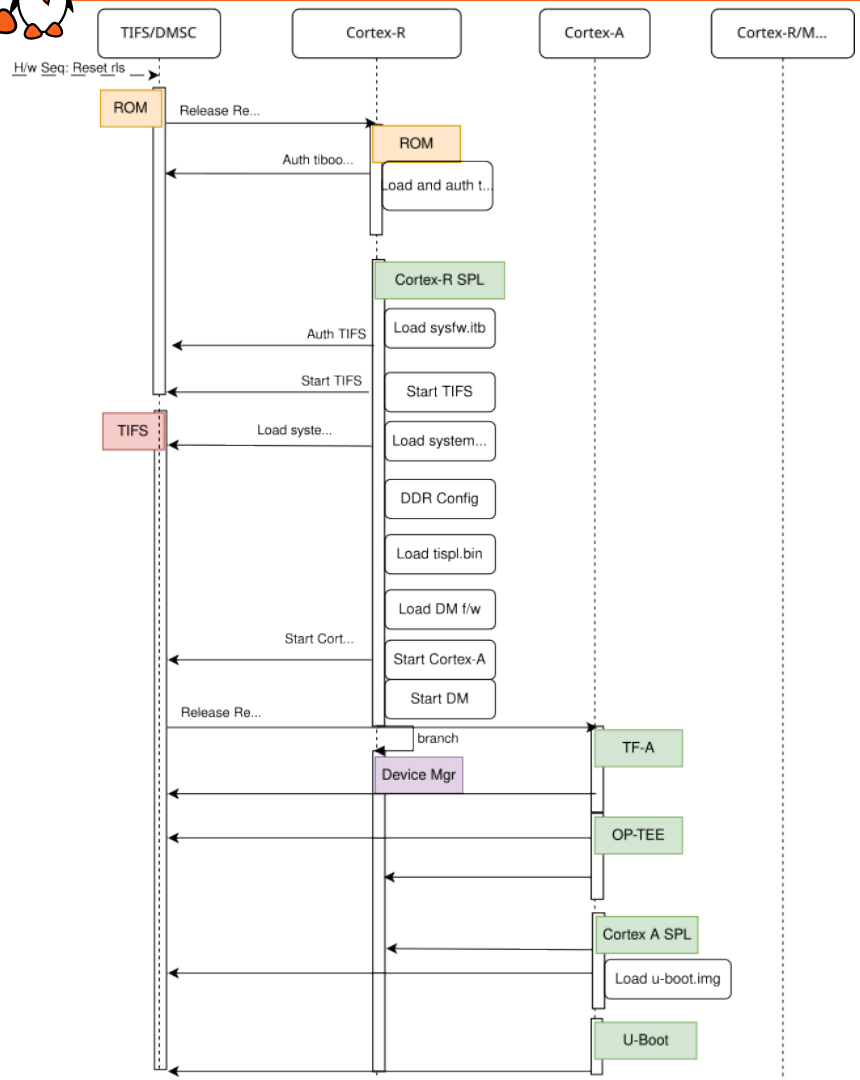
```
Debugged-by: Gregory CLEMENT <gregory.clement@bootlin.com>
Debugged-with: Thomas Richard <thomas.richard@bootlin.com>
Presented-by: Thomas Richard <thomas.richard@bootlin.com>
```

Initial problem statement

- Customer has a product based on TI J7200 SoC.
- We implemented suspend-to-ram few years ago.
- Already deployed, several versions of the product are supported.
- RAM prices explosion because of AI.
- Customer found a 2nd source for DRAM chips.
- Goal is to confirm that suspend-to-ram is still working with new chips.
- It `should` work, the enter and exit retention sequences `should` not be memory dependent.
- New configuration could be needed but it will not impact the retention sequences.

The board boots but crashes very early during resume 😯

J7200: Boot, Suspend and Resume flows



- Suspend-to-ram: SoC is fully off.
- TF-A is running in SRAM, it shall be `restored` during resume.
- OP-TEE is running in DRAM, no need to restore it.
- TF-A and DM images are saved in DRAM by R5-SPL to speed-up resume (LPM memory region).

Suspend path:

- Linux
- TF-A (saves its context in a dedicated memory region)
- DM sets the DRAM in self-refresh and configures PMIC to switch to S2R state.

Resume path:

- Similar to boot flow until DRAM configuration
- R5-SPL asks PMIC if it's a cold boot or a resume
- Exit DRAM from retention
- R5-SPL loads DM and TF-A **(from DRAM)** and start them.
- TF-A restores its context and goes to its resume path.

Initial investigation

- Look at datasheets.
- Run kernel memtest during the boot.
- Review retention sequences.
- Enable debug logs in enter retention sequence.
- Enable debug logs in R5-SPL.

```
Starting ATF on ARM64 core...
```

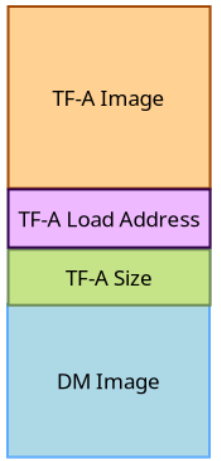
- Add some debug logs in TF-A.
- But nothing, A72 core never starts.
- Something is wrong in TF-A loading or A72 core starting.

Investigating TF-A Loading Issue 1/3

- Load TF-A image in SRAM.

  $$\text{TF-A Image} = \text{TF-A Firmware} + \text{Certificate}$$

- In resume path TF-A image is loaded from DRAM (LPM memory region) instead of boot media (FIT image).
- Call `ti_secure_image_post_process()`, so TIFS will:
  - Authenticate the image
  - Enable firewall on the SRAM.
  - Set the processor configuration flags



LPM memory region

```
debug("%s: Authenticating image: addr=%lx, size=%ld, os=%s\n", __func__,
      fit_image_info[IMAGE_ID_ATF].image_start,
      fit_image_info[IMAGE_ID_ATF].image_len,
      image_os_match[IMAGE_ID_ATF]);

ti_secure_image_post_process(&image_addr,
                             (size_t *)&fit_image_info[IMAGE_ID_ATF].image_len);
```

Investigating TF-A Loading Issue 2/3

- Enable debug in `ti_secure_image_post_process()`.
- Issue found: Image size is equal to 0.

```
void ti_secure_image_post_process(void **p_image, size_t *p_size)
{
        struct ti_sci_handle *ti_sci = get_ti_sci_handle();
        struct ti_sci_proc_ops *proc_ops = &ti_sci->ops.proc_ops;
        u64 image_addr;
        u32 image_size;
        int ret;

        image_size = *p_size;
        if (!image_size) {
                debug("%s: Image size is %d\n", __func__, image_size);
                return;
        }
        ...
```

- Why? The `p_size` pointer doesn't point to TF-A image size, but at the end of the LPM carveout (unused memory).
- New memory chips are initialized to `0x00000000`.
- On other boards memory chips are initialized to `0xffffffff`.
- Size is just used to invalidate cache lines, so `0xffffffff` works!!

Investigating TF-A Loading Issue 3/3

- We fixed the pointer issue.
- Suspend-to-ram is working now
- But only one time.

```
Authentication failed!
```

- Reason: `ti_secure_image_post_process()` returns a new size using `p_size` pointer which now points to TF-A size.
- A modified (and now wrong) TF-A size is used during the 2nd resume sequence to load TF-A image in SRAM.
- TF-A Image is partially loaded ➡ Authentication fails
- Solution is to use intermediate variable.

Two software bugs were hidden, an unrelated hardware modification just revealed them.

Debugging Chronicle 5: Four Instructions Too Many

Presented-by: Théo Lebrun <theo.lebrun@bootlin.com>
Debugged-with: Grégory Clement <gregory.clement@bootlin.com>
Debugged-with: Théo Lebrun <theo.lebrun@bootlin.com>

Booting?

- One morning, waking up to a stalled Mobileye EyeQ5 (mips64r6el) board:

```
Waiting for core 1 to start... STAT_CONF=0x333003
Waiting for core 1 to start... STAT_CONF=0x333003
Waiting for core 1 to start... STAT_CONF=0x333003
... same message every second ...
```

- Code hadn't changed, it might have been caused by a config change.
  Let's *bisect* from `crashing_defconfig` to `upstream_defconfig`.
- Landed on `CONFIG_DYNAMIC_FTRACE=y` as root cause.
  NOT `CONFIG_DEBUG_KERNEL` or `CONFIG_FTRACE` or `CONFIG_FUNCTION_TRACER`.
- *Lesson: changing your `.config` before leaving the office is a bad gift to future-you.*

Some leads?

Here we have a few leads to follow:

- Toolchain issue? We have access to a MIPS Technologies LLC *Codescape* GCC toolchain and we faced odd bugs in the past with upstream.
  - Quickly tested 4 toolchains (2 MIPS versions & 2 upstream) ⇒ unrelated.
- Investigate the impact of `CONFIG_DYNAMIC_FTRACE=y` to understand why?
  - Digging into the code that hot-patches kernel executable might be harmful to one's inner peace.
- Bisect? I used dynamic ftrace for performance measurements in the past.
  - Start by find a working kernel version (good written notes helped).
  - Then bisect from OK v6.15 to NOK v6.16 in ~14 steps.
  - Landed on commit `76c43eb507bc` ("MIPS: SMP: Implement parallel CPU bring up for EyeQ", Grégory Clement, 2025-04-13, +25/-0).
  - *Lesson: keep your colleagues nearby.*

Parallel boot?

- Said colleague indicated kernel parameters to control CPU boot flow.
  - `nosmt` OK
  - `cpuhp.parallel=false` OK
  - `maxcpus=0` OK
  - `maxcpus=1` NOK
- Bug is reproducible and prerequisites are clear. The kernel needs parallel CPU boot active & dynamic ftrace support built-in for a proper stall.
- With some more logs, we noticed the wrong CPU was booting! The sequence was CPU0→CPU3 rather than CPU0→CPU2.

CPU	Cluster	Core	VPE
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	0	2	0
5	0	2	1
6	0	3	0
7	0	3	1

Mobileye EyeQ5 topology

Parallel boot?

```
static bool __init cpuhp_bringup_cpus_parallel(unsigned int ncpus)
{
   const struct cpumask *mask = cpu_present_mask;

   if (cpuhp_smt_aware()) {                                  // We are thread-aware.
      const struct cpumask *pmask = cpuhp_get_primary_thread_mask();
      static struct cpumask tmp_mask __initdata;
      cpumask_and(&tmp_mask, mask, pmask);
      cpuhp_bringup_mask(&tmp_mask, ncpus, CPUHP_BP_KICK_AP); // Bring up a first batch of
      cpuhp_bringup_mask(&tmp_mask, ncpus, CPUHP_ONLINE);     // CPUs: primary threads.
      ncpus -= num_online_cpus();
      if (!ncpus)
         return true;
      cpumask_andnot(&tmp_mask, mask, pmask);
      mask = &tmp_mask;
   }
   cpuhp_bringup_mask(mask, ncpus, CPUHP_BP_KICK_AP);         // Bring up remaining CPUs.
   cpuhp_bringup_mask(mask, ncpus, CPUHP_ONLINE);
   return true;
}
```

Memory corruption?

- In `cpuhp_bringup_cpus_parallel()`, primary thread mask is `3-5` (non-sensical).
- At boot, it is `0,2,4,6` (much more reasonable).
- Someone, somewhere is overwriting our primary thread mask.

- Discovered by littering code with `printk("... %*pbl\n", ...)`.
- *Lesson: printk debugging for-the-win.*

Memory corruption?

- What is overwritten?

```
// arch/mips/include/asm/topology.h
extern struct cpumask __cpu_primary_thread_mask;
#define cpu_primary_thread_mask ((const struct cpumask *)&__cpu_primary_thread_mask)
// arch/mips/kernel/smp.c; variable appeared in commit
// 76c43eb507bc ("MIPS: SMP: Implement parallel CPU bring up for EyeQ")
struct cpumask __cpu_primary_thread_mask __read_mostly;
```

- Who could it be? Hum, ftrace maybe?

```
void __init ftrace_dyn_arch_init(void)
{                                                    // before: pmask == {0, 2, 4, 6}
   /* Encode the instructions when booting */
   ftrace_dyn_arch_init_insns();
   /* Remove "b ftrace_stub" to ensure ftrace_caller() is executed */
   ftrace_modify_code(MCOUNT_ADDR, INSN_NOP);
}                                                    // after:  pmask == {3, 4, 5}
```

ftrace assembly generation?

```
static inline void ftrace_dyn_arch_init_insns(void)
{
    u32 *buf;
    unsigned int v1;

    /* la v1, _mcount */
    v1 = 3;
    buf = (u32 *)&insn_la_mcount[0]; // <-- 2 * sizeof(u32)
    UASM_i_LA(&buf, v1, MCOUNT_ADDR);

    /* jal (ftrace_caller + 8), jump over the first two instruction */
    buf = (u32 *)&insn_jal_ftrace_caller;
    uasm_i_jal(&buf, (FTRACE_ADDR + 8) & JUMP_RANGE_MASK);

#ifdef CONFIG_FUNCTION_GRAPH_TRACER
    /* j ftrace_graph_caller */
    buf = (u32 *)&insn_j_ftrace_graph_caller;
    uasm_i_j(&buf, (unsigned long)ftrace_graph_caller & JUMP_RANGE_MASK);
#endif
}
```

ftrace assembly generation?

```
void UASM_i_LA_mostly(u32 **buf,
                unsigned int rs, long addr)
{
    if (!uasm_in_compat_space_p(addr)) {
        uasm_i_lui(buf, rs,
                   uasm_rel_highest(addr));
        if (uasm_rel_higher(addr))
            uasm_i_daddiu(buf, rs, rs,
                uasm_rel_higher(addr));
        if (uasm_rel_hi(addr)) {
            uasm_i_dsll(buf, rs, rs, 16);
            uasm_i_daddiu(buf, rs, rs,
                          uasm_rel_hi(addr));
            uasm_i_dsll(buf, rs, rs, 16);
        } else
            uasm_i_dsll32(buf, rs, rs, 0);
    } else
        uasm_i_lui(buf, rs,
                   uasm_rel_hi(addr));
}
```

```
void UASM_i_LA(u32 **buf,
               unsigned int rs,
               long addr)
{
    UASM_i_LA_mostly(buf, rs, addr);
    if (uasm_rel_lo(addr)) {
        if (!uasm_in_compat_space_p(addr))
            uasm_i_daddiu(buf, rs, rs,
                          uasm_rel_lo(addr));
        else
            uasm_i_addiu(buf, rs, rs,
                         uasm_rel_lo(addr));
    }
}
UASM_EXPORT_SYMBOL(UASM_i_LA);
```

- *Lesson: when writing to a buffer, please take/put the buffer size.*

Upstream fix?

```
⟩ git show -U0 --format= 36dac9a3dda1
diff --git a/arch/mips/kernel/ftrace.c b/arch/mips/kernel/ftrace.c
index f39e85fd58fa..b15615b28569 100644
--- a/arch/mips/kernel/ftrace.c
+++ b/arch/mips/kernel/ftrace.c
@@ -57,4 +57,14 @@ static inline void ftrace_dyn_arch_init_insns(void)
-       /* la v1, _mcount */
-       v1 = 3;
-       buf = (u32 *)&insn_la_mcount[0];
-       UASM_i_LA(&buf, v1, MCOUNT_ADDR);
+       /* If we are not in compat space, the number of generated
+        * instructions will exceed the maximum expected limit of 2.
+        * To prevent buffer overflow, we avoid generating them.
+        * insn_la_mcount will not be used later in ftrace_make_call.
+        */
+       if (uasm_in_compat_space_p(MCOUNT_ADDR)) {
+               /* la v1, _mcount */
+               v1 = 3;
+               buf = (u32 *)&insn_la_mcount[0];
+               UASM_i_LA(&buf, v1, MCOUNT_ADDR);
```

Upstream fix?

```
+       } else {
+               pr_warn("ftrace: mcount address beyond 32 bits is not supported (%lX)\n",
+                       MCOUNT_ADDR);
+       }
@@ -191,0 +202,7 @@ int ftrace_make_call(struct dyn_ftrace *rec, unsigned long addr)
+       /* When the code to patch does not belong to the kernel code
+        * space, we must use insn_la_mcount. However, if MCOUNT_ADDR
+        * is not in compat space, insn_la_mcount is not usable.
+        */
+       if (!core_kernel_text(ip) && !uasm_in_compat_space_p(MCOUNT_ADDR))
+               return -EFAULT;
+
```

- Next step: insert trampolines everywhere for ftrace outside kernel code space (think modules).

Debugging Chronicle 6: When Sine Waves Betray You

Presented-by: Alexandre Belloni <alexandre.belloni@bootlin.com>

Initial problem statement

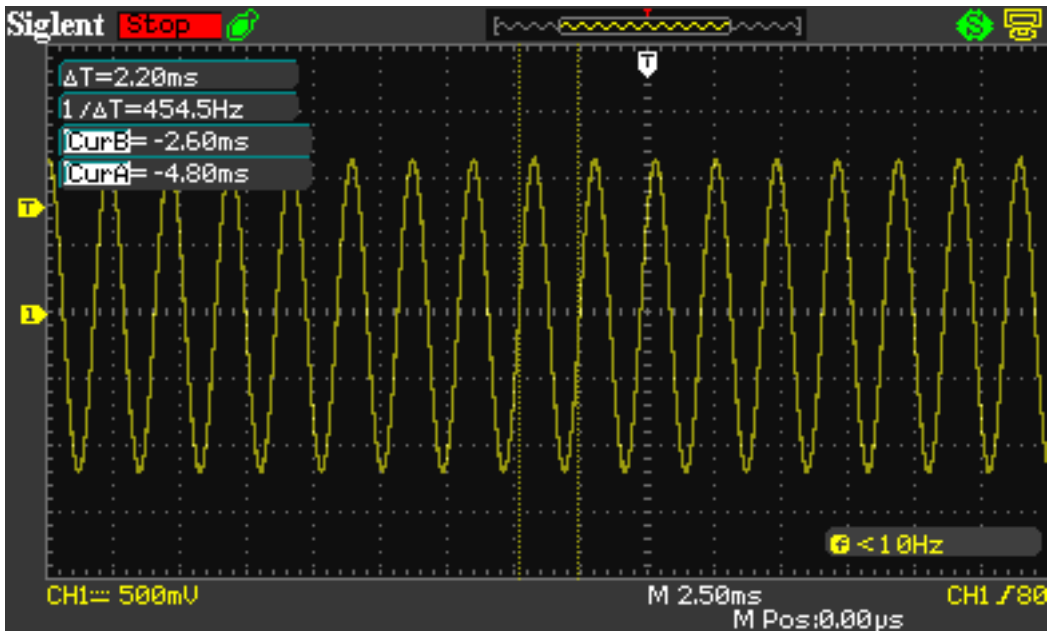
- Customer makes i.MX7 SoM based PA systems
- The audio card is composed of a PCM3168A connected to SAI2 on the SoC
- The vendor kernel was 4.1.15, we target upstream v4.9
- Adding the card description in device-tree made audio work
- DONE!

The issue

- Testing was done as usual, using `speaker-test` as this allows generating audio samples without shipping audio files on the rootfs.
- My usual commands are:
  - `speaker-test -t sine -f 440 -c 2`
  - `speaker-test -t sine -f 1000 -c 1`
- This generates pure sine wave tones. And when testing, it is possible to output a tone on the left channel and another tone on the right channel.

The issue

- The analog output was perfectly fine:

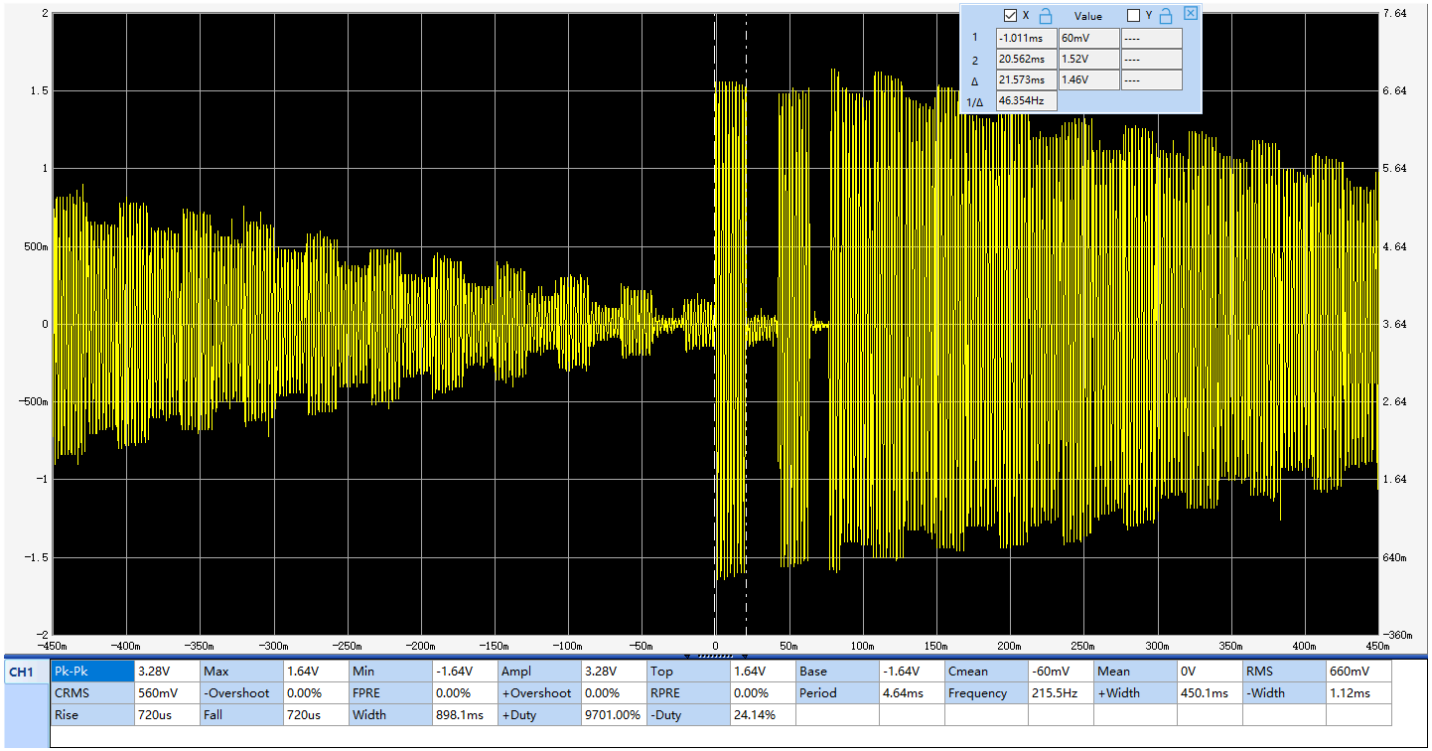


The issue

- Testing with an actual sound file works but sounds weird
- The format of the audio file samples is exactly the same as what `speaker-test` is generating
- The PCM bus data looks fine and the sine wave tests suggest it is correct.
- Tweaking the PCM3168A analog control is not making the audio sound better
- There was an SGTL5000 codec on the SoM, testing with this codec showed the same results
- As DMA is used, to get samples from userspace to the SAI FIFO, it is not possible to inspect the data going into the FIFO.

Investigation

- To get more visibility on the FIFO content, an audio file has been crafted: a fading out 440 sine wave.



Investigation

- Here it is, the issue is that the periods (20ms of samples here) are not played in order.
- This is definitely observable on the PCM bus
- But the file and the userspace buffers are correct
- $\Longrightarrow$ DMA is the culprit
- But the transfers are happening regularly, they have the correct size and run for the proper duration.

Investigation

- Multiple SDMA firmwares were tested without any luck.
- Backporting board support on 4.1.35 fixed the issue but this was not satisfying
- The culprit was 5881826ded79 (“dmaengine: imx-sdma - update the residue calculation for cyclic channels”) that appeared in v4.9
- Final fix is 85f57752b33c (“dmaengine: imx-sdma - correct the dma transfer residue calculation”) in v4.10

Key Takeaways

- When testing with `speaker-test`, use a non easily divisible frequency for the tone, e.g. 441 Hz, 444 Hz or 999 Hz so it doesn't repeat itself exactly on period boundaries. This will exhibit gaps or jumps in the wave when DMA is not working properly.
- Issues are not always caused by clocks or pinctrl!

Thanks for your attention!
Enjoy the social event!

The Bootlin Team