1. Show us a cool thing you've built/coded up :) I know you're already submitted a portfolio, but this one should be something that you're proud of and want to explain why. For reference a single image and a three sentence description explaining why it's cool / unusual is perfect.

Figure 1: Homam 64GB camera

Links:

Company site: https://zorachka.com/homam

Amazon link: https://www.amazon.sg/dp/B08FRF4W2N

One of the projects I'm really proud of is the Homam 64GB smart camera. I joined a startup developing this camera at the very beginning of the project as a firmware developer and eventually developed three versions of the firmware for the three PCBs (each equipped with unique set of peripherals) the camera consists of. During the firmware development, I learned a lot about software/hardware co-design, firmware development processes, and debugging approaches and tools, including oscilloscopes and bus analyzers. I also wrote and modified several

Linux Kernel drivers in parallel with firmware development, including a custom GPIO driver. I took part in the camera certification process for Wi-Fi Alliance compliance — in particular, I optimized several Wi-Fi driver code paths to improve performance. Eventually, I also ported the user space software from Apple HomeKit v2 to v3. I worked on this project for two years and gained a wide range of embedded and firmware development skills.

I am proud of it not only because I gained solid technical skills, but also because I truly felt the magic of creating a new product and having a sense of ownership. I am also proud of the camera because we successfully delivered the product to the market.

2. What's your favourite niche peripheral / feature of a microcontroller?

I used to work extensively with the PIC16Fxxxx family of microcontrollers. Apart from standard peripherals like ADC/DAC, GPIO, and PWM, it includes an interesting feature called the Configurable Logic Cell (CLC). It's like having a tiny programmable logic block inside the microcontroller, where you can wire internal signals—timers, interrupts, pins—through simple AND/OR/XOR gates without consuming firmware cycles.

Although I have never used it in a production project, I really like the concept because it allows you to solve timing-critical problems without adding external hardware. It feels like a "hidden FPGA block" inside a low-cost chip, and using it effectively can provide elegant, hardware-level solutions to tricky problems.

3.Have you ever hacked together a firmware "fix" you knew was kinda sketchy but it kept the project alive? What was it?

This comes from my experience on the Homam 64GB camera project. One of the PCBs had a sound output and a pair of microphones. We discovered that the microphones would randomly turn on and interfere with the sound output, producing an almost ultrasonic, very quiet noise. None of our early customers noticed it, but in a quiet room we could reproduce the problem.

Our investigation showed that two bits in the SoC registers controlling the microphones could occasionally flip. To keep the product stable, I wrote a quick firmware fix in one of the ASoC drivers: a workqueue that periodically reset those bits. While workqueues don't hurt Linux kernel performance much, it was still an ugly hack. But the hack kept the system stable until the hardware team traced it back to a bad schematic (likely EMI noise or something similar) and fixed it properly. Once that was resolved, we removed the workqueue.

4. You get a PCB that resets randomly once every few hours. What's your go-to first couple of tests?

Random PCB resets can stem from many causes, broadly falling into two categories: hardware malfunctions and software issues. In my answer, I'll assume that higher-level software layers (such as the Linux kernel) have already been validated, and I'll focus on how I would approach HW/FW debugging. With no

prior knowledge of the particular PCB design, I'll outline a general methodology for tackling this class of problems. The key to solving such issues is increasing visibility into both hardware and firmware. - Hardware visibility: I would connect oscilloscope channels to critical circuitry points where failures are most likely (e.g., MCU and PMIC power pins) and set up triggers to capture voltage drops. - Software visibility: I would enable all available hardware reset flags (power-on reset, brown-out reset, watchdog reset, software reset, etc.) and log these states into persistent storage such as EEPROM or NVRAM upon reboot. This way, I can determine the root cause of the reset. I would also instrument the code to log key events (e.g., ISR entry/exit, function entry/exit, subsystem activity). Stack canaries and periodic watermark checks are also useful. A global fault handler should be configured to capture PC, SP, and the most recent stack frames. To store this information, I'd allocate a circular buffer and periodically flush it to NVRAM. Finally, I would dedicate one MCU pin as a "heartbeat" signal (5–10 Hz) and attach an oscilloscope probe to trigger when the signal halts.

These above steps alone can reveal more than 90% of failure causes. If these initial steps don't expose the issue, I would proceed with deeper strategies:

- Isolation testing: Run minimal firmware with all subsystems disabled. If resets still occur, the issue is highly likely hardware-related.
- Stress testing: Push the system with burst I/O or heavy loads. If resets occur more frequently, it points to hardware instability (though firmware faults under stress must also be checked by analyzing reset flags).
- Binary search of subsystems: If isolation shows no errors, progressively enable subsystems. For example, enable half of them and run tests. If resets occur, narrow further by halving again. If not, enable the other half and repeat. With two boards, this process can be parallelized to save time.

If the root cause still isn't clear, I would extend the investigation to environmental conditions — heatmaps, vibration, humidity, and other external factors can reveal intermittent issues.

5. Which of the skill categories mentioned in the role description apply to you the most? Feel free to pick multiple, just put your most comfortable competency first. (If your background is unusual, explain in a few words what skills & background you bring!)

- Should have experience writing low-level drivers for a number of various chips.
- Excellent debugging skills in complex settings
- Be able to form hypothesis and run experiments about the behaviour of the system
- Experience programming, shipping, and maintaining hardware in a real-world setting.
- Must be excited by rapid iteration more than trying to perfectly plan ahead of time