Linux on Quick-Turnaround Projects at Ball: No, We Aren’t Putting Linux in Canning Jars

Sam Povilus

Embedded Software Engineer

Tell'em what you're gonna tell'em

- Why am I here today
- Overview of Ball Aerospace (cans are interesting but in a totally different way)
- Quick overview of my Linux use and career
- Overview of the phased array antenna
- Overview of our hardware
- Why I wanted to use Linux
- Linux lessons
- General embedded development
- Advantages

Why am I here?

Bigger picture for Linux developers

- Don’t use Linux their whole careers
- What helped me (what your customers care about)

Bigger picture for Embedded Linux users

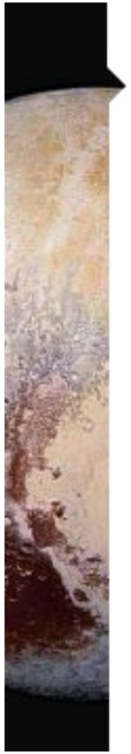
- You might be taking a risk
  - The risk pays off
- You’ll do more reading and less writing
- Things will come together

FROM JARS

TO STARS

Some know us for the glass jars that bear our name.
Some know us for **metal packaging for food, beverage and consumer goods**.
Some know us for ground-breaking **aerospace and defense solutions**.
All our customers know we are there to help them succeed – no matter the challenge or mission.

Enabling Firsts. Driving Innovation.

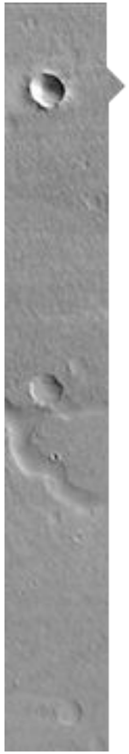


(2015) Ralph camera: first instrument to return high-resolution colored images of Pluto

(2011) Kepler: first spacecraft to discover an exoplanet in the habitable zone

(2010) Deep Impact: first spacecraft to intercept a comet

(2007) Orbital Express: first spacecraft designed for on-orbit, autonomous servicing operations



(2006) HiRISE: first instrument to return high-resolution images of Mars' surface from an orbiting platform

(2000) Tactical High Energy Laser: first anti-missile defense system using a high-energy chemical laser

(1989) Cosmic Background Explorer (COBE): first instrument to provide definitive evidence of the "Big Bang"

(1987) Solar Backscatter Ultraviolet Radiometer/2: first instrument to provide confirmation of the Antarctic ozone hole

(1983) Infrared Astronomical Satellite (IRAS): first instrument to provide all-sky imaging of the universe in the infrared from space

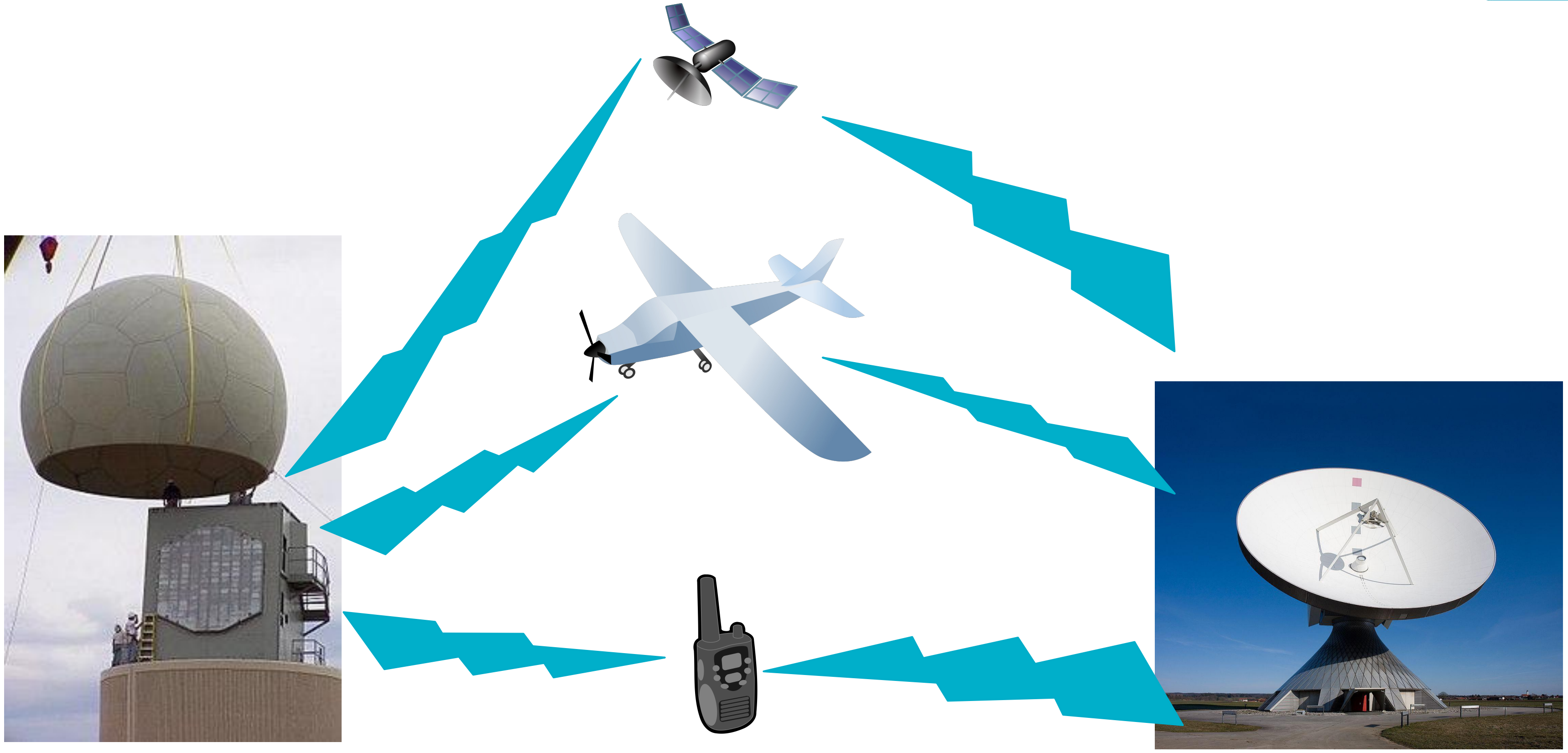
(1962) Orbiting Solar Observatory (OSO): first solar observation satellite

My career

In 6 Years I have worked on 9 programs:

- Spacecraft instrument running on a SPARC
- Spacecraft Simulator
- Algorithms translation
- Star tracker running on PPC
- Camera system for the Navy x86
- Aircraft/Space station based LIDAR for NASA x86
- Camera system for the Air Force x86
- Phased array (this presentation) ARM
- Camera simulator

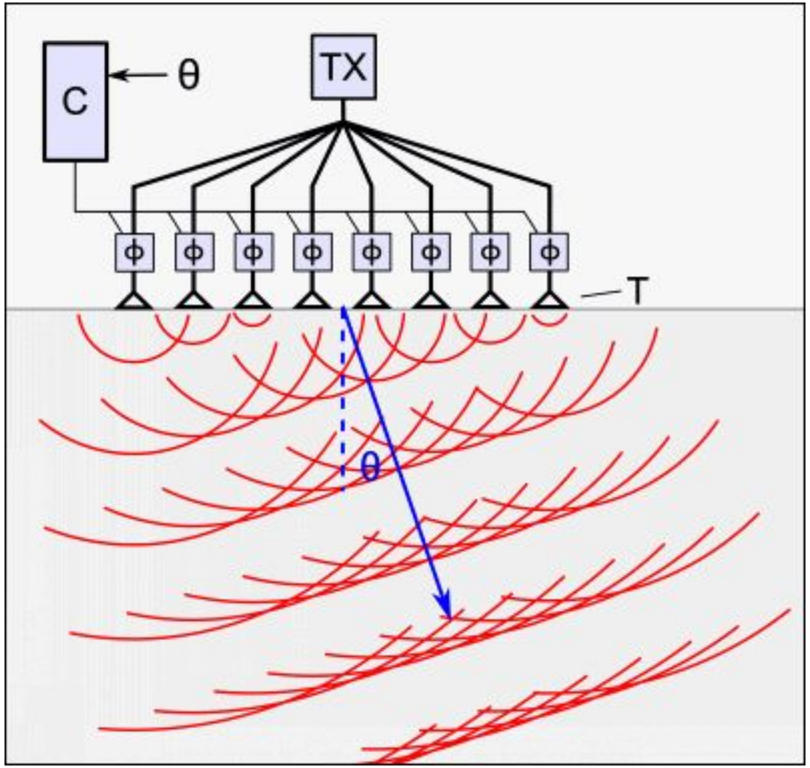
Why a phased array



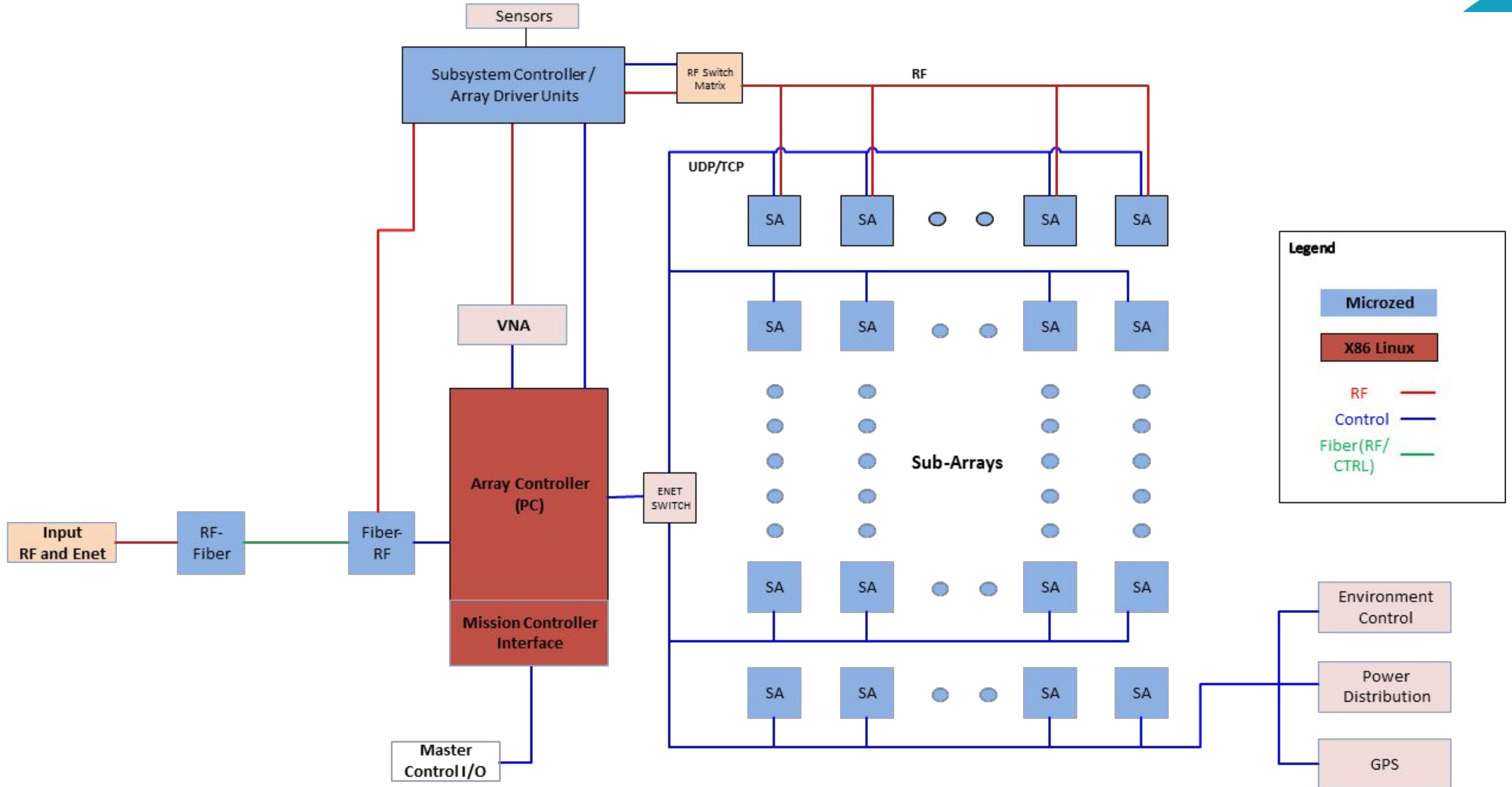
Phased array

Lots of tiny antennas working together to make a big antenna.

- RF stays analog through the whole process in our case
- RF goes through amplifiers, attenuators, phase shifters, and time delays that are controlled by digital signals.
- Control computers need to produce the digital signals not at the RF frequency but the control frequency.
- Commanding rate in the tens of Hz



Linux controllers

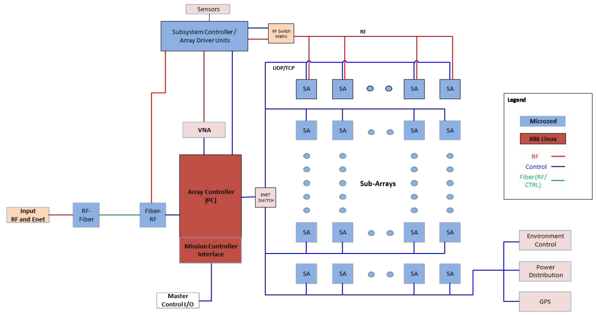


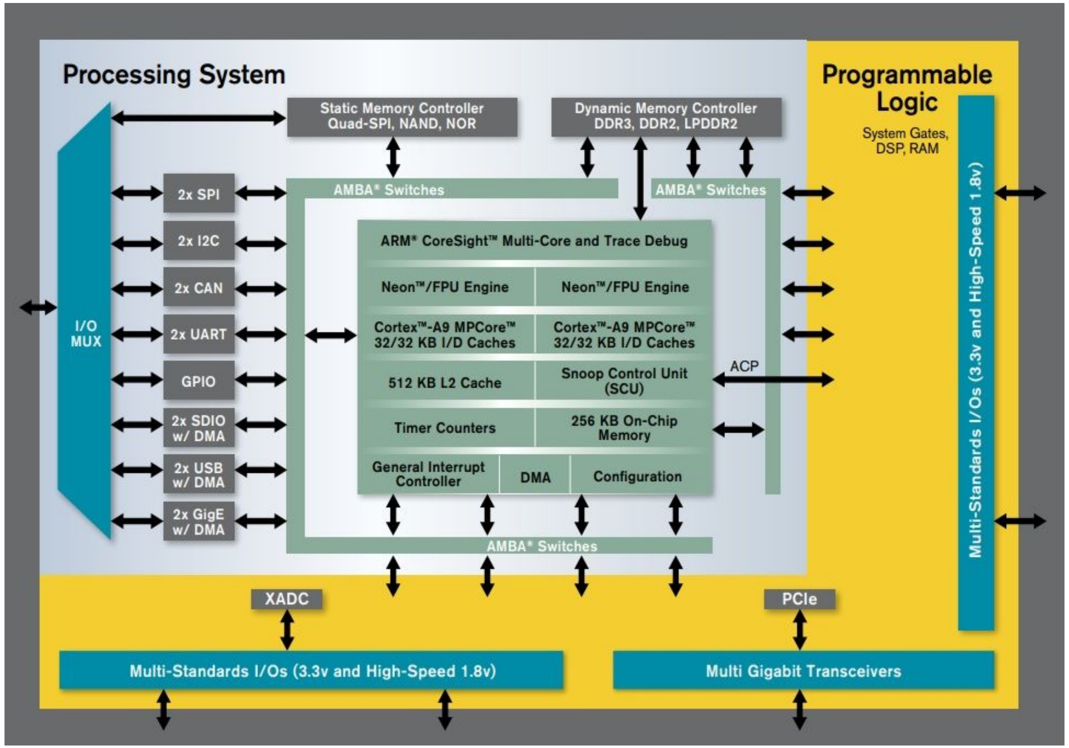
What is an FPGA and why do we use them

- Are re configurable silicon
- Can be
  - Controllers
  - Memory
  - Custom processors
- are re-configurable all the way to run-time
- allow hardware to be swapped and added
- Provide expand ability for future versions
- Can allow for different hardware to be connected to the same board

Zynq overview

- Dual ARM cores
- Lots of IO both “hard” and “soft”
- Lots of available soft controllers



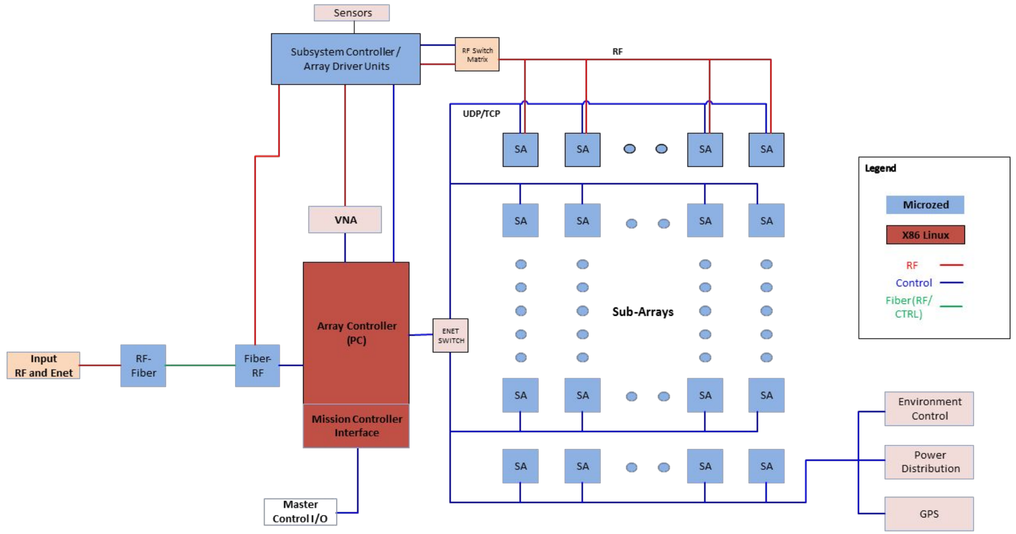

Processing System
Programmable Logic
System Gates, DSP, RAM
Static Memory Controller Quad-SPI, NAND, NOR
Dynamic Memory Controller DDR3, DDR2, LPDDR2
AMBA® Switches
AMBA® Switches
I/O MUX
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO w/ DMA
2x USB w/ DMA
2x GigE w/ DMA
ARM® CoreSight™ Multi-Core and Trace Debug
Neon™/FPU Engine
Neon™/FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
ACP
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
AMBA® Switches
XADC
PCIe
Multi-Standards I/Os (3.3v and High-Speed 1.8v)
Multi Gigabit Transceivers
Multi-Standards I/Os (3.3v and High-Speed 1.8v)

Linux hardware

RF-Fiber converters 2	Array driver ~5	Subarray ~50	Array Controller
▪ 1 I2C ▪ GPIO ▪ Analog	▪ 4 SPI ▪ 2 spi flash ▪ 2 I2C ▪ Analog ▪ GPIO	▪ 8 SPI busses ▪ 4 SPI flashes ▪ 5 I2C busses ▪ Analog ▪ GPIO	▪ Ethernet in and out

What I needed to do

- Receive commands and send telemetry to Array Controller
- Control RF chain
  - Tens of Hz
- Store calibration on daughterboards
- Automate parts of calibration
- Protect hardware from operator (or more likely engineer) stupidity



Why I chose Linux

- I wanted to
- I have experience with it
- I've spent a lot of time walking between buildings
- Others have experience with it
- I didn't want to write drivers (bad embedded engineer)
- Someone had to
- I was starting from scratch

Ball concerns with Linux

Speed

Support

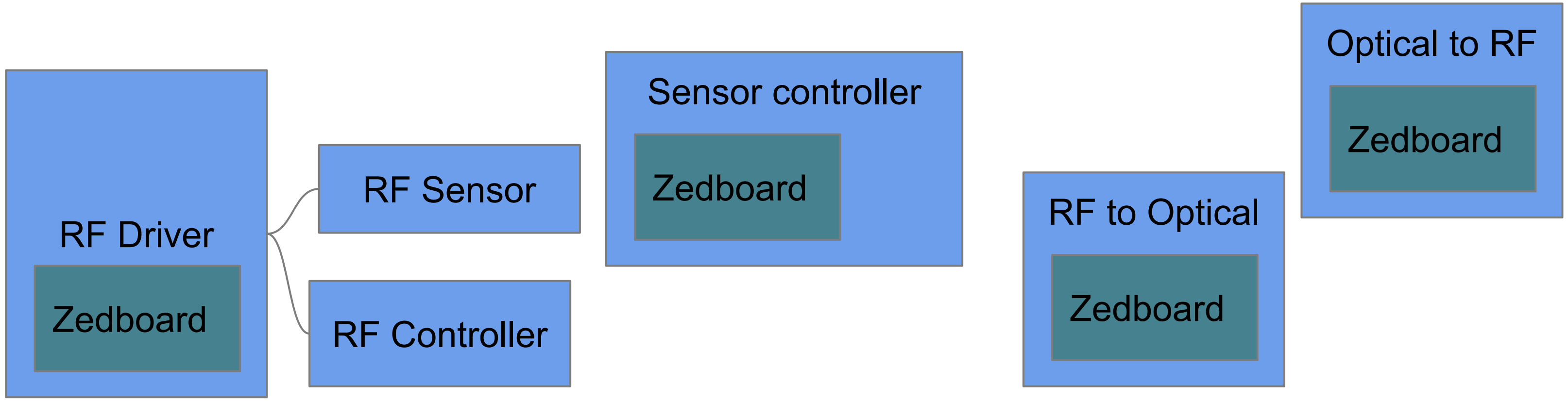
Ball’s Reluctance to Accept Linux

- Linux is still considered risky
  - Don’t bring a Lotus to a Cadillac show

- Experienced developers
  - We aren’t as constrained by hardware anymore
  - Old habits
  - Past experiences
  - Better knowledge of history

Device tree + FPGA is fantastic

- Allows multiple daughterboards to be used by one software image
- Device tree overlays would make it even more powerful
- Device tree parameter standardization is hard
- What “supported” by the device tree means is not clear



Don’t trust auto-generated files.

```
/*
 * CAUTION: This file is automatically generated by PetaLinux SDK.
 * DO NOT modify this file
 */
&qspi {
	#address-cells = <1>;
	#size-cells = <0>;
	flash0: flash@0 {
		compatible = "micron,n25q128";
		reg = <0x0>;
		#address-cells = <1>;
		#size-cells = <1>;
		spi-max-frequency = <50000000>;
		partition@0x00000000 {
			label = "bootenv";
			reg = <0x00000000 0x00020000>;
		};
		partition@0x00020000 {
			label = "jffs2";
			reg = <0x00020000 0x00fe0000>;
		};
	};
};
```

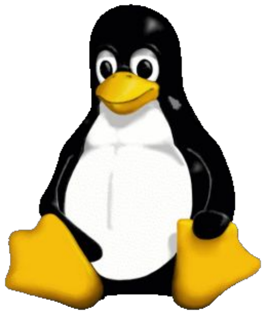
```
/*
 * CAUTION: This file is automatically generated by PetaLinux SDK.
 * DO NOT modify this file
 */
&qspi {
	#address-cells = <1>;
	#size-cells = <0>;
	flash0: flash@0 {
		compatible = "micron,n25q128a13";
		reg = <0x0>;
		#address-cells = <1>;
		#size-cells = <1>;
		spi-max-frequency = <50000000>;
		partition@0x00000000 {
			label = "bootenv";
			reg = <0x00000000 0x00020000>;
		};
		partition@0x00020000 {
			label = "jffs2";
			reg = <0x00020000 0x00fe0000>;
		};
	};
};
```

Linux doesn’t know your SPI chain

- When you have long wires your bits can get screwed up
- You need to be able to prove to the hardware engineers that it’s a hardware problem
- If you have a return value don’t ignore it

(Linux) developers will argue about everything

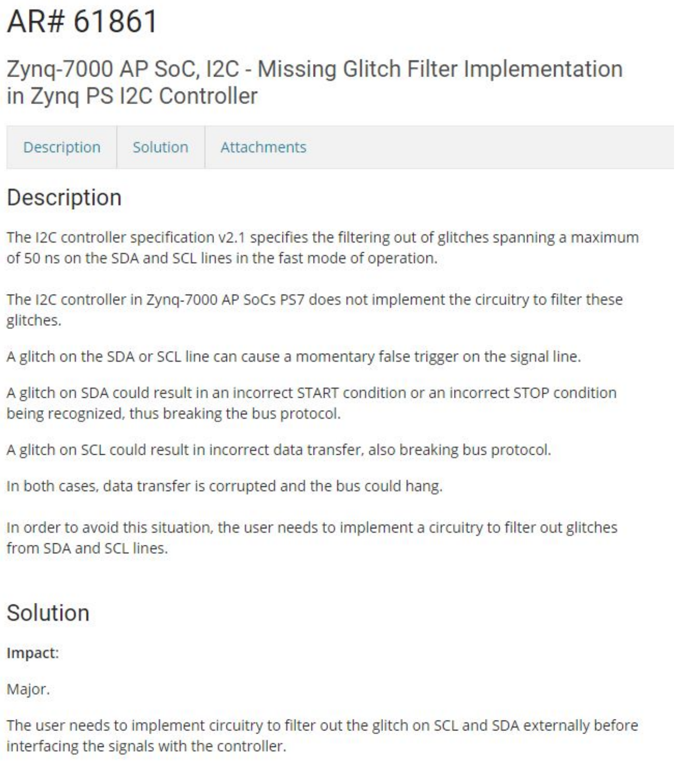
- The line between Linux and the device tree isn't obvious
- Get used to reading mailing list archives
- You will vehemently agree with people on one side of an argument and disagree with the other a few people had years ago



!=

devicetree
.org

It's not always about Linux: You will find hardware bugs



AR# 61861

Zynq-7000 AP SoC, I2C - Missing Glitch Filter Implementation in Zynq PS I2C Controller

Description | Solution | Attachments

Description

The I2C controller specification v2.1 specifies the filtering out of glitches spanning a maximum of 50 ns on the SDA and SCL lines in the fast mode of operation.

The I2C controller in Zynq-7000 AP SoCs PS7 does not implement the circuitry to filter these glitches.

A glitch on the SDA or SCL line can cause a momentary false trigger on the signal line.

A glitch on SDA could result in an incorrect START condition or an incorrect STOP condition being recognized, thus breaking the bus protocol.

A glitch on SCL could result in incorrect data transfer, also breaking bus protocol.

In both cases, data transfer is corrupted and the bus could hang.

In order to avoid this situation, the user needs to implement a circuitry to filter out glitches from SDA and SCL lines.

Solution

Impact:

Major.

The user needs to implement circuitry to filter out the glitch on SCL and SDA externally before interfacing the signals with the controller.

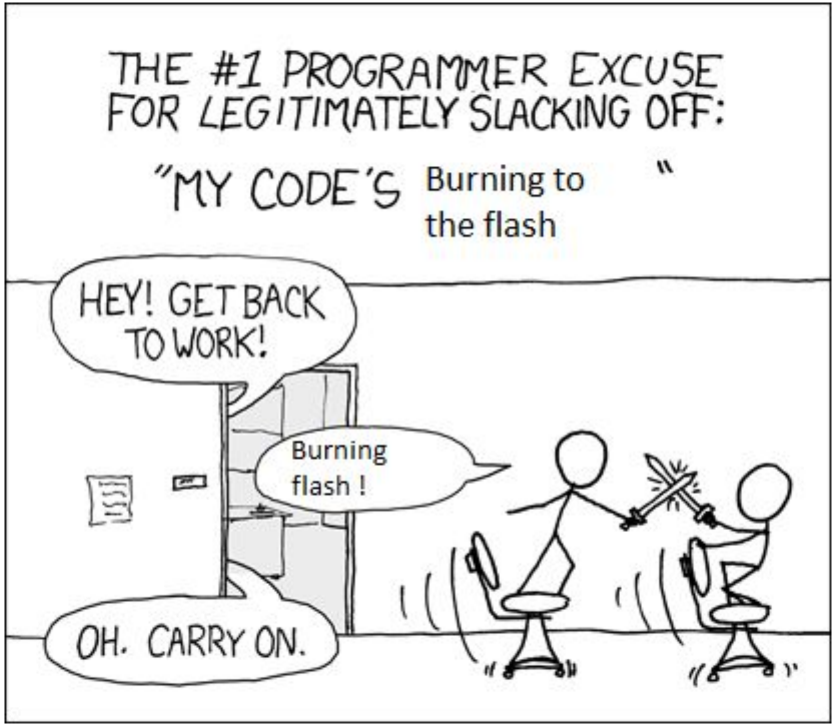
- They will bite you after hardware is done
- You will have to prove 100% without a doubt it's hardware and not software (software is a lot cheaper than hardware to fix)

Be nice to your techs

- Automate everything they have to do
- I gave them a 17 step process, cut down from 30 steps and it was still painful.

Find the fastest way to burn flash

- You will do it a lot
- You will get distracted
- SD cards are nice but not particularly vibration resistant


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S Burning to the flash"
HEY! GET BACK TO WORK!
Burning flash !
OH. CARRY ON.

Based on a comic from XKCD.com

Insert yourself into the hardware design process

- Hardware engineers are busy
- Reprints of RF boards are expensive
- Check their work anywhere you care about
- Get them to pick devices with drivers already in mainline
- For network devices make sure they chose ones that don’t require ActiveX

Hardware disappears

- When you are working with experimental hardware any number of teams might take the first few boards don’t expect to have them to yourself
- Beg, plead, and threaten for a dedicated software test bench

Testability

- Familiarity for test team
- Some tests can be run on commodity hardware
- Access to Python, Perl, or Bash makes automated testing all that much easier

Keep a log of everything you do

- You will forget commands
- You will repeat trials
- You will have to write a user's guide
- You will have to show other developers what you have done

Advantages

- I2C hardware easy to move from “hard” to soft when we found a bug in the “hard” I2C device
- SSH was invaluable
- FPGAs allow for hardware commonality
  - FPGAs + Device tree = pure magic
- GPIO from user space is magic
  - The classic embedded “hello world” is trivial
- I didn’t have to write a SPI flash driver
- Other boot options (this is U-Boot)
- Expandability
- Built in tools for upgrades
- Ability to patch
- Modularity built in
- Broad support channels

Thank you

Questions?

kernel.development@povil.us