How to make hardware that is unfriendly to coreboot

Ron Minnich
Google

also: Trammel Hudson, Patrick Georgi, Timothy Pearson, Martin Roth, Stefan Reinauer, ...

Overview

- The “standard” PC is not the standard any more
- Assumptions that used to apply no longer do
- Vendors decide they don’t want some hardware
- Or design it in in a dumb way
- Or violate standards
- Or don’t tell you it’s there
- Or do it in a way that only firmware/BIOS/whatever know about
- Or install backdoors that can’t be turned off that are full of zero-days/exploits
- In this talk I’ll go over some examples in hopes that new hardware designers can avoid them

The wall of shame

- No serial ports
- Non-standard baud rates (really happened!)
- More than one serial port, and the one that defaults to enabled is not the one that’s hooked up
- Same for debug ports (eg USB debug) that exist but aren’t exposed (but drive the webcam or similar nonsense)
- FLASH too small
- Complex audio paths without documentation
  - That require 5 MiB of firmware to play a beep
- ME, PSP, other auxiliary processors that control the “C”PU
- Multiple different devices marketed as the same name

Wall of shame (does it ever end?)

- Closed Graphics
- Closed Embedded Controllers
- "Verified Boot" modes that lock out any unsigned firmware (a.k.a. “TiVo-ised” hardware), versus "measured boot" that allows user modification.
- Obfuscated / closed source ILOM / BMC solutions
- GPIO pin wiring that changes for no good reason
  - And requires overdependence on ACPI
- Hardware on the zero page
- Breaking port 0x80 (POST)

Serial port

- Vendors keep telling us it’s not there
  - Pushed “USB debug port” at us
  - After about 10 years ...
- EHCI (USB2) debug port hardware costs $100
- xHCI (USB3) debug requires RAM (optionally, ie never, provided by the controller)
- So, here’s a question: what’s worst thing to debug
  - RAM startup
- What does xHCI debug require to work
  - RAM
- So, is the xHCI debug there for you when you really, really need it?
  - no

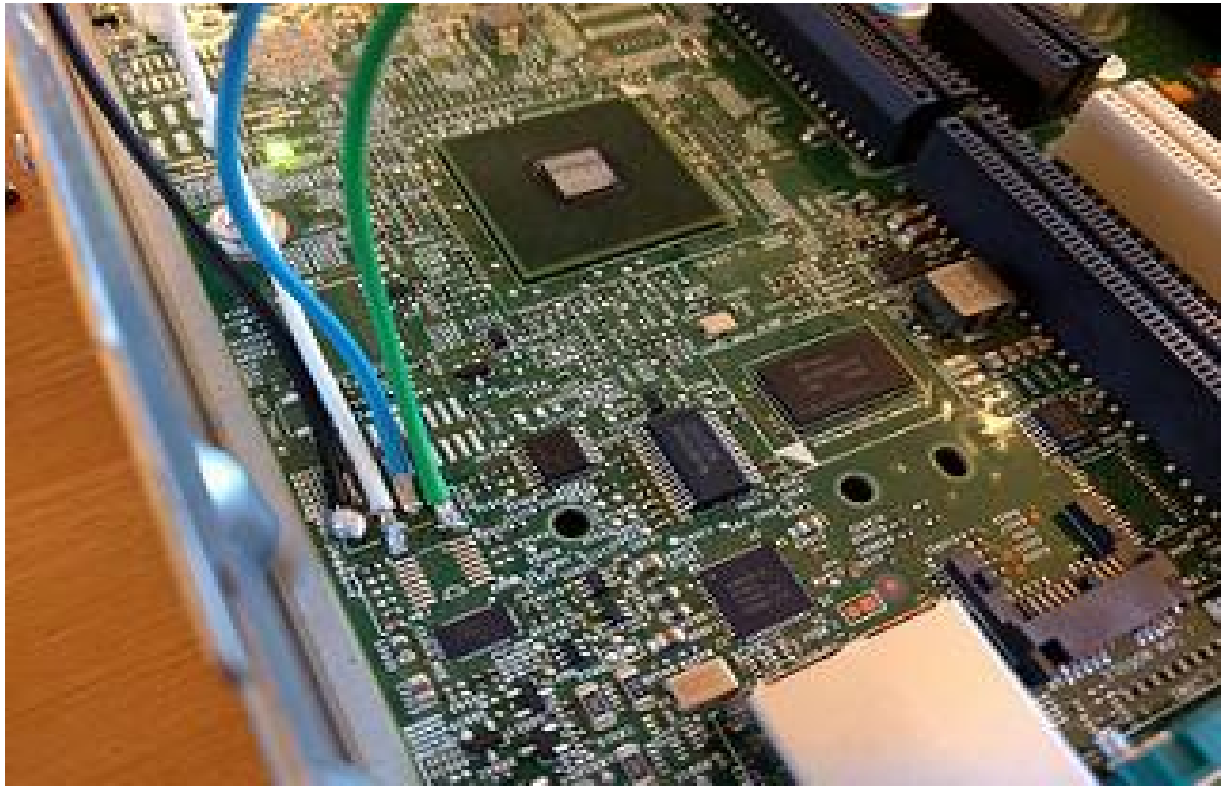
Serial port

- USB debug port does not coexist with normal USB operation
  - So how do you test the USB port when you need it for serial output?
- Way too complex
  - Requires, again, lots of things to be working
  - Old school serial port Outb %al, %dx is hard to beat
- Somehow, Universal Serial Bus forgot about “S” meaning “Serial”
- But is the serial port really “gone”?

Is the serial port really gone?

- To the best of my knowledge, no
- One way or another it's always there, on some version of ICHx, or superio, or something
- Because it's basically impossible to do bringup/test without it
  - See previously slide about how much usb serial debug sucks
- So, if you're designing, just make sure there's a pad on there which people can get to
- Input is good, but we can manage without it
- But we *must* have output
- Serial ports are the single best debugging tool we've had
- Yes, even more than JTAG

We’re willing to work for it! (thanks Trammel Hudson)



If one port is good, is two better?

- Usually not
- One board we worked on had two serial ports
- And the one that came up defaulting to “on” was not the connected one
- The connected port was a real mess to enable, requiring PCI config cycles
- Simple rule: the port you wire up should default to working at power on reset
- It’s fine to have more than one
- Just make sure that if only one is wired, it’s the one the defaults to on
- MiniPCI serial is another worst case so please don’t count on that

Standard baud rates

- This should be obvious, but
- If you are tempted to create parts with non-standard baud rates
- Don’t
- Yes, this happened
- No, it’s not a good idea
- Initial vendor response was “working as intended”
- Then “Won’t fix”
- Then “hang on, we’re fixing it”
- Because people needed it
- There’s no excuse for it

Small flash

- Linuxbios became coreboot when flash parts went from 512k to 256k
- The name change came 8 years later, but …
- The forcing function was the change in flash size
- People are once again looking at Linux in flash
- Advice: *Make it easy*
- Make flash parts 16 MiB and we have room to grow
- Kernel about 2M, initramfs 2-6M, leaves room for kernel, backup, 2 copies of coreboot

Crazy complex audio paths

- Chromebooks among other things need to play a tone
- This should not take megabytes of flash space
- If you are going to drop one of these onto a board….
- Consider dropping enough simple hardware so we can do pwm in software

ME, PSP, and friends

- The ME has been a matter of concern for years
- And we were right to be concerned
- It's been broken for *ten years* in a way that left us all vulnerable
  - https://semiaccurate.com/2017/05/01/remote-security-exploit-2008-intel-platforms/
- Chip vendors: learn from that mistake; don't repeat it
  - Even though you keep repeating it :-), see, e.g., WEP
- If you want to create security infrastructure, *go to the community first*
- Maybe it could be good, and minimal, and work
  - ME is none of these things
- If you make our security depend on you, open review process is *mandatory*
- Trusted boot, not secure boot, should be the rule

Multiple different devices marketed with same name

- Simple case: two laptops, same part number
- Utterly different motherboards inside
- We realize this is SOP in the PC world
- But it should stop
- Product names should define a unique implementation

Graphics

- Make graphics controllable from firmware
- We've done measurements in coreboot
- It turns out to be a good place to do graphics
  - Better than kernel performance
  - Better than binary blob quality and openness
- The native graphics init in coreboot has worked for five years now
- It's not been a technical issue for all that time
- What's holding up chipset vendors?
  - You know who you are
  - And we all know why you're doing it

How fast it can be (sorry, Docs broke, won’t view)

How fast it can be

- Left: Linux
  - Which must be very general
- The VGA ROM blob is far worse
- Right: coreboot
  - Specialized to the platform
- We can skip lots of probing and delays, and hence run faster, with coreboot
- In one case we reduced 7.7 seconds to 2.7 seconds
- This is possible if graphics information is available
- But vendors still want to use slow, inflexible, buggy blobs
- From an old email: “... we just got told we need to get NDA docs to fix the GPL DRIVER IN THE KERNEL.”

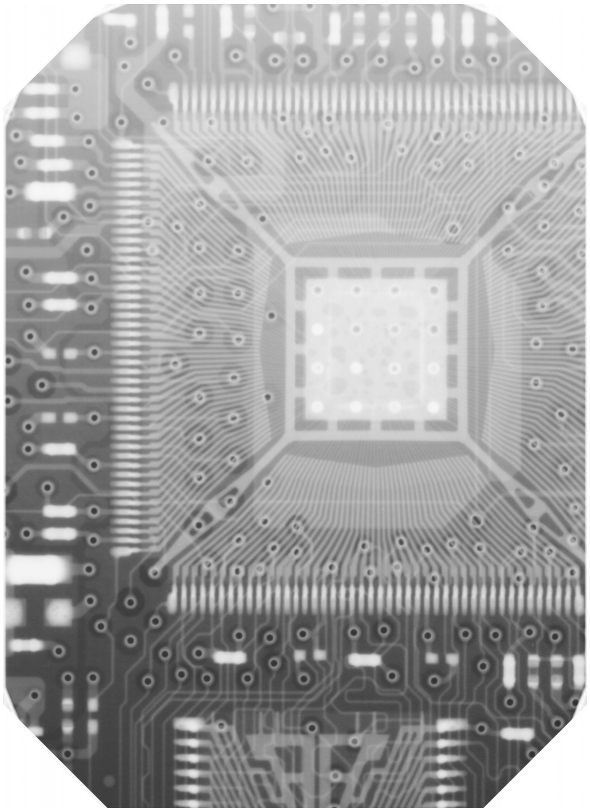
Embedded Controllers

- We are long past the time that these can be 8 bits
- The closed nature of them has been a problem for some OEMs
- And you don’t need to write your own
- Google has industrial-strength code available for free
  - 20M Chromebooks can’t be wrong
- There’s also an OpenBMC effort (Tim Pearson!)
- Same rules apply: security-critical functions should be open and reviewed

GPIO

- This problem really became apparent with Opteron
- Certain GPIOs were needed for memory operation (RST, CKE, etc.)
- AMD had no recommendation of which we know
- So of course, every vendor chose a *different* set
- And, worse, changed it for each new board
- And, worse, sometimes changed it for *revisions of a board type*
- How did we know which GPIO to use?
- Sometimes, the vendor told us
- Other times, well ...

Don’t make us go there



- Image courtesy http://uvicrec.blogspot.com/2015/08/xy-ray-x-ray-scanner.html
- Yes, people really do this
- First time I heard of it, a friend at SGI took a board from IBM to a veterinary clinic

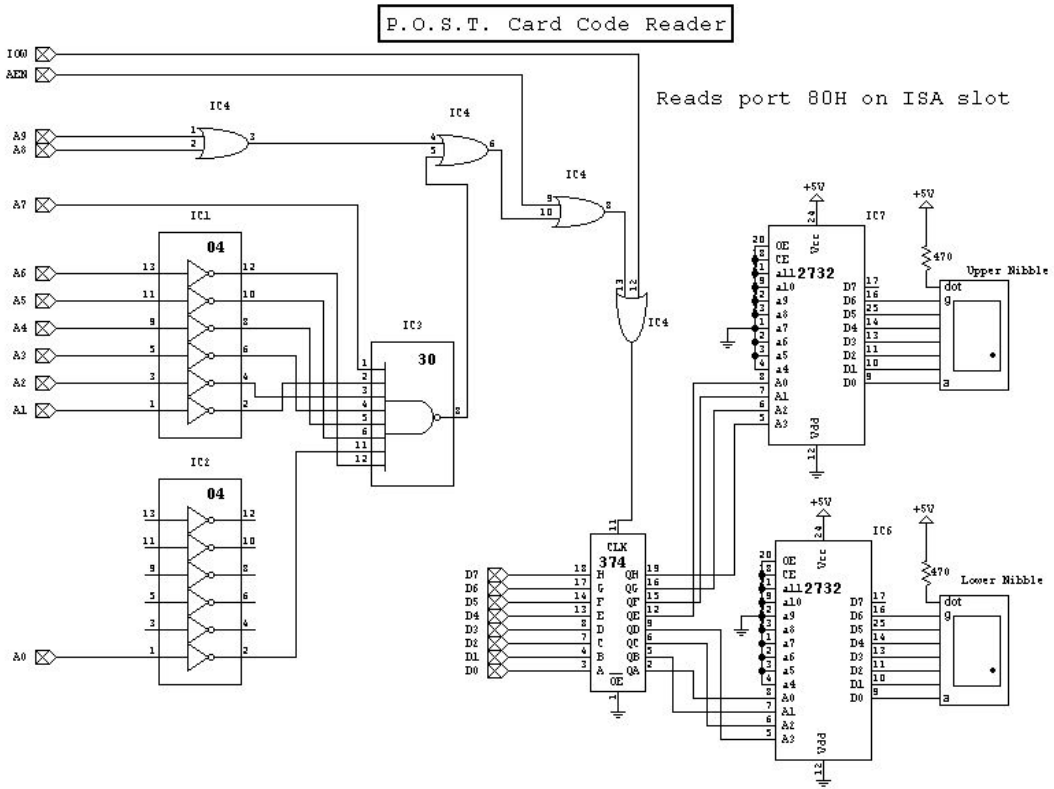
GPIOs

- Chipset vendors: if you have required GPIOs, provided guidance
- Board vendors: if you have required GPIOs, at least make sure they are the same as much as possible, at the least for the same board type!
- Because it’s not like we can’t figure it out anyway
  - Schematics of (at minimum) low speed board regions help increase trust in hardware
  - Information contained in board netlist generally not able to be considered “trade secret”
    - Can be extracted from physical board via trivial processes
    - Withholding schematics just slows down devs w/o meaningfully impacting bad actors
  - Additional information beyond mere schematic data normally discovered during netlist RE
    - May not be good for vendor’s image -- faulty routing, poor design, questionable quality
    - Netlist RE of a physical board ***cannot*** be legally stopped, but...
    - ...released schematics make routing level and lower flaws less likely to be discovered

Hardware on “page zero”

- I.e. hardware that requires that NULL pointers be valid
- Please don’t do this
- It makes setting up paging to catch NULL pointer usage *almost* impossible
- Workaround
- P = mallocalign(4096, 4096)
- Zp = map(p, NULL);
- Now we can use ZP to address the NULL page
- But let’s just try to avoid this altogether, eh?

Don’t break POST (just saw another note today!)



- Here’s a classic ISA POST card design (thanks http://bbright.tripod.com/information/postcard.htm)
- On PCI, POST cards have to work whether present or not
- How’s this work? Via IO errors:
  - POST never responds
  - Hardware sees timeouts and acts accordingly
- outb to 0x80 is an error ...
- Coreboot does outbs
- Don’t make your chipset lock up if there is an IO error!
  - Yes, this happened
  - Forced us to add config option

Summary

- If you follow a few simple rules, coreboot ports are pretty easy
- If you don’t, they can really be miserable
  - Which is why, 5 years in, there’s a board we still don’t want to finish a port for
  - No, can’t tell you more
- And if you really get clever, you might even make a port impossible
- There’s an upside to following these rules
- They can greatly reduce the cost of board design and bringup
- They can greatly ease all firmware ports, not just coreboot ports
- They can greatly ease problem resolution
- And the people on the list can give you really good advice, so be sure to ask!

extra

More cool stuff

- Power decouple SPI from board
- SPI header
- LPC header? (dead, and security hole?) [for TPM?]
- JTAG? [pads, not header]
- Case closed debugging that is open and in all new chromebooks
  - See chromeos wiki, chrome EC codebase
  - On USB-C port you get uarts, SPI,
  - Just needs modified EC code
  - Found in chrome EC code but does not require full chrome EC part or all its software
- “Golden connector” as on thinkpad

BMC / ILOM (Tim Pearson)

- Traditionally closed source (AMI)
  - Riddled with security holes
  - Spawned an entire generation of system architectures assuming BMC has *no* security at all!
- Excellent alternative (OpenBMC) already exists
  - Used in production (ships installed from vendor) on IBM OpenPOWER systems
  - x86 versions also “in the wild”
  - Most BMC designs use ASpeed hardware; OpenBMC ready to use on many ASpeed devices
- BMC needs to have secure boot ability
  - BMC can be significant attack vector due to network access and privileged position on board
  - ASpeed does not integrated secure boot hardware into their devices
    - This is a good thing! Vendor signing here would render BMC a useless security risk
  - Use a secure boot solution from the list of options on the ME/PSP avoidance slide
    - FlexVer, OTP boot key, TPM, etc.
- BMC must not have undisclosed access to any part of the system!

How to avoid ME, PSP, and friends (Tim Pearson)

- Many decentralized options available for a truly secure boot
  - FlexVer™ auditable measured boot (Raptor Engineering)
  - OTP boot signing keys (Qualcomm QorIQ, other embedded systems)
  - TPM-based measured boot (Google, IBM)
- Pick one or more and use them!
- Caveats
  - Adding any of these technologies to an ME or PSP enabled platform does *not* mitigate threat
  - Listed technologies are only useful on platforms that do not contain an ME, PSP, or equivalent
    - Adding them won't necessarily harm security, but won't help security either
- OEMs must pressure x86 vendors to offer securable CPUs without ME, PSP
  - Otherwise, x86 will be abandoned for secure platforms with high value data

BMC / ILOM (from Tim)

- BMC *must not* have undisclosed access to any part of the system!
  - Undocumented NCSII links to one or more system Ethernet ports borders on negligent
  - BMC should not be able to alter the main system Flash ROM while machine power is on!
    - SMI exploits possible
    - FlexVer™ / TPM can mitigate somewhat, TPM is less able to do so
    - BMC secure boot can help mitigate this attack vector
- Weak BMC access mechanisms should be disabled by default
  - Username/password for network-based effective local root access? Not a good idea…
    - SSH disables this by default for a reason!
  - Consider Kerberos or other cryptographic authentication solution for BMC access

Why the ME, PSP, and friends must be avoided (from Tim)

- Don’t repeat the mistakes of the centralised SSL authentication system!
  - Also known as “Crack once, hack everywhere”...
    - Centralized systems ***always*** end up being hacked, often to devastating effect
      - Remember “Code Red”? “Blaster”? Now extend to the physical hardware…
      - Monocultures, especially those under strict vendor control, remain a severe hazard
  - Legal framework for data breach liability still in flux (and varying with jurisdiction)
    - Data breaches are being taken more seriously year after year (GDPR, etc.)
    - Does hardware vendor have control of platform? If so, does this equal legal liability?
    - Hardware vendor may end up paying large fines for data breaches….
    - ...or platform vendor may select alternate hardware if platform vendor is legally liable
    - Standard “platform no longer supported” caveat *may not protect* from legal liability
      - Liability of vendor retaining full control w/ signed firmware remains largely untested
- Decentralization makes hacking any particular target much harder
  - Brute force no longer viable option -- cost / benefit ratio much lower outside of select targets
  - “Dragnet”-type hacking and surveillance of private systems no longer viable