ECE 471 – Embedded Systems
Lecture 28

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

17 November 2017

Announcements

- Project status report due Monday. Not long:
  - A one-line statement of your project idea
  - A summary of the project you've made so far
  - List any parts you need that you don't have yet
  - List if you're willing to present early (Monday of Wednesday vs Friday) (there will be some bonus for presenting early)
- Midterm will hopefully be back Monday
- Website issue was resolved

Supercomputing Review

- Back from SC'17
- Significant ARM presence
- Talk about machines in Top500 `top500.org` and why it might be good to use embedded processors
- Talk about the Pi cluster paper
  - Pi Clusters, bunch at SC this year
  - Our big difference, power measurement and LED displays
  - Some others with displays now

- Work was done with undergrad. There are opportunities for undergrad research, CUGR and REU.
- Look into updating to Pi3, Linpack crashes it.

Wii Nunchuck

- Fairly easy to interface
- Put onto i2c bus. Device 0x52
- Send handshake to initialize. Use longer one (0xf0/0x55/0xfb/0x00) not the simpler one you might find(0x40/0x00). This works on generic nunchucks and possibly also disables encryption of results.
- To get values, send 0x00, usleep a certain amount, and read 6 bytes. This includes joy-x, joy-x, accelerometer

x/y/z and c and z button data. More info can be found online.
byte0 = joy-x, byte1 = joy-y, byte2 = top8 acc x, byte3 = top8 acc y, byte4 = top8 acc z, byte 5 is bottom 2 z,y,x then button c and z (inverted?)

Linux and Keyboard

- Old ps/2 keyboard just a matrix of keys, controlled by a small embedded processor.
  Communication via a serial bus. Returns "keycodes" when keypress and release and a few others.
- Many modern keyboards are USB, which requires full USB stack. To get around needing this overhead (for BIOS etc) support bit-bang mode. OS usually has abstraction layer that supports USB keyboards same as old-style

- Linux assumes “CANONICAL” input mode, i.e. like a teletype. One line at a time, blocking input, wait until enter pressed.
- You can set non-CANONICAL mode, async input, and VMIN of 1 to get reasonable input for a game. Arrow keys are reported as escape sequences (ESCAPE-[-A for up, for example).
- Even lower-level you can access “RAW” mode which gives raw keycode events, etc.
- There are libraries like ncurses that abstract this a bit. Also GUI and game libraries (SDL).

Faking Linux Input Events

- How to insert input events into Linux, i.e. have a software program fake keyboard/mouse/joystick events.
- Linux supports a "uinput" kernel driver that lets you create user input.
- There is some info on a library that makes this easier here: `http://tjjr.fi/sw/libsuinput/`
- It has examples for keyboard and mouse. Joystick should be possible but there's no sample code provided.
- Python wrappers seem to exist too.

System Busses

- Older busses often exposed CPU pins directly to connector: Apple II, S-100, ISA
- This was not sustainable, if only because number of CPU pins grew rapidly. Also speed issues.

Parallel vs Serial Busses

- Originally most busses were Parallel. More bits at a time means higher bandwidth. IDE, Parallel Port, 32-bit PCI, 64-bit PCI
- Problems with parallel: keeping signals in sync. As busses go faster, skew comes into things. Wire length matters. Power issues with driving wide busses.
- Newer busses are serial: SATA, PCIe, USB, Firewire, etc. Also advantage of having fewer wires to route.
- HPC users grumble about speed of PCIe

USB Bus

- USB 1.0 – 1996 – Low Speed 1.5Mbit/s (keyboard, etc), Full Speed 12Mbit/s (disk)
- USB 1.1 –
- USB 2.0 – 2000 – High Speed 470MBit/s
- 2-5m cables
- 4 pins. 5V, GND, D+, D-. Differential signaling (subtract). More resistant to noise.

- Micro connectors have extra pin for on-the-go (says if A end or B end gnd vs v+)
- Unit load, 100ma. Can negotiate up to 500ma (more USB 3.0)
- Up to 127 devices (by using hubs)
- Enumeration, vendor and device

USB Bus

- USB 3.0 – 2008 – SuperSpeed 5GBit/s (though hard to hit that) Full Duplex (earlier half duplex)
- USB 3.1 – 2014 – SuperSpeed+ 10Gbit/s
- Backwards compatible, has 5 extra pins next to standard micro with GND, SSTX+/- and SSRX+/- (full duplex)
- USB-C – 2014
  24-pin: 4 power/ground pairs, two differential non-super-speed pairs, four pairs of high-speed data bus, two

sideband pins, two pins for cable orientation

cables can be USB2, USB3, USB3.1, up to 5A(20V=100W) but 3A more common

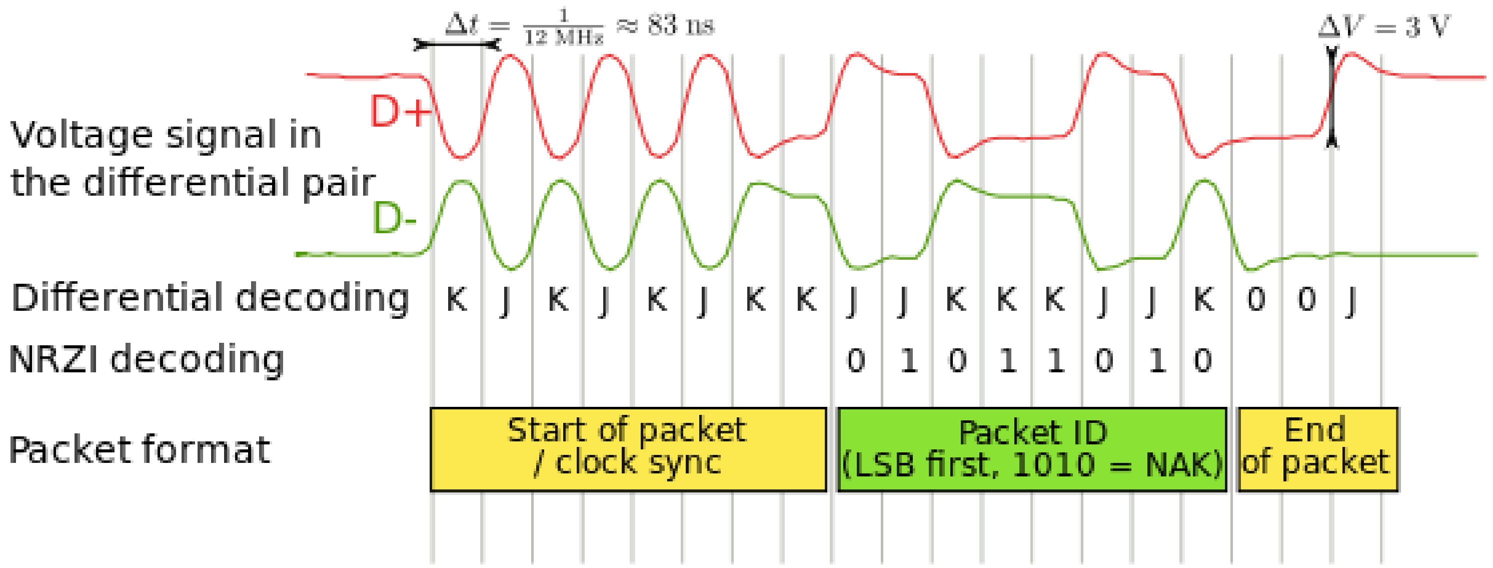
wrong pullup can cause cable that damages hardware

USB Signaling

- Differential signaling, twisted pair, 90Ohm impedance
- Low+Full = 0V low, 3.3V high, not terminated
- High = 0V low, 400mV high, terminated with resistor
- SuperSpeed, separate lines, but original lines used to config
- Host, 15k pulldown pulls data lines to 0 (nothing connected, SE0)

- USB device pulls line high with 1.5k which overpowers pulldown. Full bandwidth D+ high, low bandwidth D- high
- J and K states.
- NRZI line coding – 0 signaled by J to K (switching state). 1 signaled by leaving as is
- Bit stuffing – after six consecutive 1s must include 0
- starts with 8 bit synch – 00000001 which is KJKJKJKK. Data then sent. End marked by 00J.

- Reset by 10ms SE0
- Highspeed uses "chirping" to negotiate speeds, during reset chirps J and K
- SuperSpeed uses 8b/10b encoding (limits bandwidth), CRC, other features
- SuperSpeed+ uses 128b/132b encoding
- Example from Wikipedia CC0:


$\Delta t = \frac{1}{12\ \mathrm{MHz}} \approx 83\ \mathrm{ns}$
$\Delta V = 3\ \mathrm{V}$
D+
D-
Voltage signal in the differential pair
Differential decoding
K J K J K J K K J J K K K J J K 0 0 J
NRZI decoding
0 1 0 1 1 0 1 0
Packet format
Start of packet / clock sync
Packet ID (LSB first, 1010 = NAK)
End of packet