Turning a photo frame into a display for embedded devices

Some glue code between the framebuffer and `libusb`

Peter Meerwald

pmeerw@pmeerw.net

June 2011

Agenda

- Motivation
- A closer look
- Sniffing Windows' USB traffic
- Replaying USB traffic with `libusb`
- A service to push the framebuffer to the monitor

Motivation

- certain Samsung photo frames have a ’mini monitor’ feature, i.e. use the device to extend your Windows desktop

- e.g. SFP-107H: 10,2" (1024 × 600), 1 GB internal flash + SD card slot, USB 2.0 host + USB 2.0 peripheral, 6 W
- note: this is not a DisplayLink device which has Linux support `http://libdlo.freedesktop.org`
- so: have some fun and maybe it can be useful to output my beagleboard’s framebuffer...

A closer look...

- there is no GPL/LGPL software on it (I asked Samsung and checked the firmware image for signatures)
- SoC seems to be developped by Magic Pixel[1]
  - probably MP600, MIPS based
  - datasheet and some strange source code discovered after the fact
- Grace Woo[2] and I independently reversed the USB protocol (partly)
- basically JPEG images matching the display size are transferred repeatedly

[1] http://www.magicpixel.com.tw/

[2] http://web.media.mit.edu/~gracewoo/stuff/picframe/

Sniffing Windows' USB traffic

- install VirtualBox + extension pack (for USB 2.0 support), <u>not</u> the OSE version
- install Windows XP and device driver in VirtualBox
- use `usbmon` debugging driver to watch USB traffic going through the Linux kernel
  - `/sys/kernel/debug/usb/usbmon/*` exposes USB busses
  - just `cat` *X*`u` to watch USB traffic (*X* is the bus no.) when the thing is active in Windows

USB traffic analysis

- control and bulk transfers:
  - many different, frequent control messages
  - and large, periodic bulk transfers
- bulk transfers have JFIF magic, hmm...
  - starting at fixed offset (12 bytes) in the message
  - first check: try to JPEG-decode stripped message
  - second check: try to replay USB traffic

Replaying USB traffic with libusb

- either in Python, e.g. pyusb[3], or in C, e.g. libusb[4]

```
struct usb_device *dev = find_dev();
usb_dev_handle *udev = usb_open(dev);

// prepend magic header to JPEG data
char hdr[12] = {0xa5, 0x5a, 0x18, 0x4,
    0xff, 0xff, 0xff, 0xff /* filesize */, 0x48, 0, 0, 0};

// write it out chunk by chunk
unsigned char buf[URB_BUF];
usb_bulk_write(udev, endpoint, buf, URB_BUF, 1000 /* timeout */);

// periodically poll device status to keep it in monitor mode
unsigned char buf[STAT_BUF];
usb_control_msg(udev, USB_TYPE_VENDOR | USB_ENDPOINT_IN,
    0x6, 0, 0, buf, STAT_BUF, 1000 /* timeout */);
```

[3] http://pyusb.sourceforge.net

[4] http://www.libusb.org

A service to push the framebuffer to the monitor

- check for monitor device presence
- capture screen content from framebuffer (`/dev/fb`*X*)
    - just `open()`, `ioctl(FBIOGET_VSCREENINFO)`, `read()`
- convert pixel format: usually RGBA to RGB
- encode to JPEG using `libjpeg`[5]
- write JPEG data via `libusb`
- poll device status to stay in monitor mode

[5] http://www.ijg.org/

More ideas

- photo frame switches to a different mode when a button is pressed – need to handle
- use `libjpeg-turbo`[6] for performance
  - has SSE2 support and ARM NEON support soon
  - ~ 40 % CPU at 2 fps on beagleboard and Intel Atom with `libjpeg`
  - ~ 10 % CPU at 2 fps on Intel Atom with `libjpeg-turbo`

[6] http://libjpeg-turbo.virtualgl.org/

Summary

- I'm such a coward – did not open the device ☺
- should work for similar Samsung SPF models
- USB traffic only partly understood
  - brightness control?, many unknown control messages
- code is at `https://pmeerw.net/hg-emb/minimon`