FFADO: firewire audio for Linux

http://www.ffado.org

Linux.conf.au 2009

Jonathan Woithe

1 Talk outline

- The FFADO project
- The firewire bus
- Why audio over firewire?
- Audio-specific firewire issues
- Manufacturer support
- Protocol analysis techniques
- Using FFADO
- Usability/integration issues
- Current status
- Future plans
- Helping FFADO
- Acknowledgements

2 The FFADO project

- FFADO: Free Firewire Audio Drivers for Linux
- Originally known as “FreeBob”, in reference to the “BeBob” platform used for some devices
- Renamed FFADO to be more vendor neutral after Freebob 1.0 release to reflect wider variety of devices being supported
- Hense the “first” FFADO release will be v2.0

3 The firewire bus

3.1 Properties

- High speed bidirectional serial bus
- Speeds of 400 Mbps and 800 Mbps in use today
- Up to 63 devices per bus
- The bus implements a global address space
- Each device is assigned a unique address range on the bus
- Often abstracted as a register model – each device has a collection of individually addressable registers

3.2 Packet types

- Asynchronous (“async”): addressed to a specific bus address – that is, a particular address/register on a given device. No timing guarantees. Used for device configuration and control.
- Isosynchronous (“iso”): broadcast packets, sent on one of 63 “channels”. Guaranteed bandwidth and delivery timing. Used for audio data streaming.

3.3 Iso packet management

- A second is divided into 3072 “iso cycles”
- Each node can transmit one iso packet per iso cycle
- Timing within an iso cycle specified by a “cycle offset” (0-7999)
- An “iso clock” runs at 24.576 MHz (3072×8000) synchronously across the bus
- iso clock drives the “iso counter” to count iso clock ticks for 128 seconds before wrapping around
- Device providing iso clock master (the “Isosynchronous Resource Manager”, or IRM) is negotiated by the bus
- A node must request its iso bandwidth requirements from the IRM. Can be denied if bus’s bandwidth already fully allocated.

4 Why audio over firewire?

- High bandwidth (400 / 800 Mbps, faster variants in development) – good for 20+ channels of audio at 24 bit / 48 kHz.
- Bus bandwidth almost entirely accessible to devices
- Devices have known guaranteed bandwidth – suits streaming applications
- Slow devices can't block faster ones

5 Audio-specific firewire issues

5.1 Synchonisation

- Need sample-accurate synchronisation
- No direct access to audio DAC/ADC clocks
- Iso clock is synchronous, but devices don't lock audio clock to this for various reasons (eg: 44.1 kHz not easily derived from 24.576 MHz, audio device not necessary the IRM)
- Devices timestamp outgoing samples with the iso counter at the time of the respective sampling
- PC synthesises audio clock (relative to iso clock) using these timestamps and a Delay-Locked Loop (DLL)
- Synthesised audio clock used to timestamp samples sent to the device for playback a short time into the future

5.2 Transport latencies

- Need to compensate for buffer and transport delays through firewire stack and the device electronics
- FFADO's DLL takes care of this

5.3 Audio clock accuracy

- Some devices are very picky about the accuracy of the inferred audio clock
- Requires the use of a DLL for clock synthesis — anything simpler has insufficient accuracy when running the clock forward for transmitted samples
- Maintaining DLL is costly in terms of CPU resources — we want to address this

5.4 Firewire interface chipsets

- Some PC firewire chipsets have bugs/issues which prevent reliable operation with audio devices (and streaming devices in general). This is not Linux specific, but can be an annoying trap for newcomers.
- Generally anything from Texas Instruments (TI) is good
- Older NEC and Via chipsets were buggy
- Ricoh chipsets are patchy - some work well, others don't
- Recent VIA chipsets are reportedly OK

6 Manufacturer support

6.1 Ideal situation

- All firewire audio devices are class-compliant and adhere to the published standards for AV devices
- One common driver interfaces to all such devices

6.2 The reality

- There are published standards. But ...
- Most manufacturers use vendor-specific extensions for key aspects of device control
- Some manufacturers think they can do better than the standard and use their own

6.3 Why we need vendor support

- Only the vendors know the details of their own vendor-specific extensions
- Only the vendors know the details of their own protocols
- FFADO is receiving support from some device vendors: Echo, ESI, Focusrite, Terratec (now Musonic), Mackie
- Platform vendor support from BridgeCo and TCAT

6.4 Uncooperative vendors: our options

- Forget them. Not good PR — alienates people moving to Linux with significant prior investment in devices from these manufacturers
- Support as best we can (protocol analysis) but discourage new purchases of such hardware

7 Protocol analysis techniques

7.1 Sniffing firewire traffic

- Put second PC on firewire bus to watch traffic generated by vendor-supported system. Can see iso packets, so audio data format can be deduced. But ...
- "Standard" OHCI firewire cards can't see async packets addressed to other nodes (by-design hardware limitation)
- Can use purpose-designed firewire analysis hardware. Expensive.
- Alternatives:
  - use card based on TI's PCILynx chip and "nosy" Linux software. These cards are hard to find. IOI technology still sells a PCI version (IOI-1394TT) at "reasonable" cost (approx US$100 plus shipping).
  - use older Apple Power Macs – blue-and-white G3 or original "Yikes" G4 (with PCI graphics adapter) – which used PCILynx-based firewire solution. Use with Apple's Firebug software.
- Latest nosy release (0.3) is old and doesn't work with recent kernels. Use recent git snapshot instead.

7.2 Sniffing – another way

- Use bushound (`http://www.perisoft.net/bushound/`) on OSes supported by audio hardware vendor
- Sniffs bus traffic via a driver sitting very low in the driver stack
- Rather expensive (US$700)
- Free binary version does exist and can be used in our context (despite a small capture buffer), but it runs on only one OS version

7.3 Sniffing via Qemu

- If firewire passthrough were added to Qemu this could be used instead of bushound
- No one is working on this and I don't have time (using a PCILynx card is quicker and still rather cost effective)
- It would be great if this could be made to work

7.4 Protocol analysis

- Capture async packets sent in response to configuration changes
- Deduce protocol based on packet contents
- Some protocols are easier than others. For example: some devices provide unrelated functionality from a given register depending on whether it was read or written.

7.5 Indirect analysis

- Use if no traffic capture is possible
- Functions by comparing device registers before and after an operation
- Works well for devices whose registers are closely aligned to configuration
- Not so good on devices which rely on write sequences or side-effects
- Advantage: no “special” hardware required, more accessible
- FFADO trunk includes a tool to facilitate this method (scan-devreg)

8 Using FFADO

- For details, refer to slides from the FFADO tutorial given during the Multimedia miniconf on Tuesday
- FFADO provides a JACK backend driver
- Userspace ALSA driver is planned
- Require user access to realtime scheduling (via PAM or set_rlimits)
- User running jackd/FFADO needs read-write access to `/dev/raw1394`
- USB activity can be fatal to FFADO. RT-patched kernel can improve this.

9 Usability/integration issues

- RT scheduling configuration: could be easier
- Managing transistion from current firewire kernel stack (“ieee1394”) to the new stack (“firewire”, aka “juju”)
  - Will possibly be abstracted within libraw1394
- sensible udev rules for firewire device nodes
  - One device node per bus model of “current” firewire stack
  - One device node per device model of “new” firewire stack
- scheduling issues need addressing (especially in relation to IRQs)
  - Most promising approach is threaded IRQ handlers in the kernel
  - Otherwise it may prove necessary to put a FFADO component into the kernel to get required scheduling determinancy

10 Current status

- Currently in beta testing for “initial release” (will be FFADO 2.0)
- Release date: early 2009
- FFADO 2.0 will support:
  - Interfaces based on DM1000 chip: Focusrite Sapphire, Edirol FA-101 & FA-66
  - Some Echo Audiofire devices
  - Some MOTU devices (Traveler, 828Mk2, 896HD)

11 Future plans

- Support more devices
- Encourage more vendor support
- Refine device mixer interfaces
- Lower CPU consumption
- Improve device synchronisation stability and recovery

12 Helping FFADO

- Purchase devices from FFADO-friendly vendors and tell them their support of FFADO/Linux is why you're purchasing their interface
- Avoid devices from hostile vendors and tell them why you're not buying their devices
- Download beta releases or subversion snapshots and test them
- Donate/lend devices to FFADO developers

13 Acknowledgements

- Fellow primary developers (Daniel Wagner, Pieter Palmers)
- Companies actively supporting FFADO: Echo, ESI, Focusrite, Terratec (now Musonic), Mackie
- Our users, for continued testing and bug reports
- The trademarks of companies referred to thoughout this presentation are acknowledged

14 Links

- FFADO project: `http://www.ffado.org`
- JACK: `http://www.jackaudio.org`
- Set_rlimits:
  `http://www.physics.adelaide.edu.au/~jwoithe/set_rlimits-1.3.0.tgz`
- Nosy: `http://bitplanet.net/nosy/`
- IOI technology: `http://www.ioi.com.tw/`
- Bushound: `http://www.perisoft.net/bushound/`

Contacting me:

- jwoithe@physics.adelaide.edu.au