Recent Work on OSC Timetags at CNMAT

Andy Schmeder
Adrian Freed

http://cnmat.berkeley.edu/research

What is OSC?

OSI Layer	OSI Layer #	Example
Application	7	Interface hardware, audio synthesizer, programming language
Presentation	6	OSC, XML
Transport	5, 4, 3	UDP/IP, TCP/IP +SLIP, Serial+SLIP
Hardware	2, 1	Ethernet, USB, Serial

In a nutshell

OSC Message

/mixer/slider/3 f 0.25

vs. MIDI...

0x01 0x00 0x03 0x32

vs. XML...

```
<?xml version="1.0"?>
  <message xmlns:xs="http://w3c.org/XMLSchema">
    <mixer>
      <slider />
      <slider />
      <slider type="xs:float">0.25</slider>
    </mixer>
  </message>
```

OSC Bundles

OSC Bundle format: a collection of concurrent messages and a timestamp

```
#bundle    2008-03-07 17:30:00.2646 Z  [

  /mixer/slider/1   f   0.0
  /mixer/slider/2   f   0.9
  /mixer/slider/3   f   0.25
  ...
  /mixer/slider/8   f   0.0

]
```

Brief History

- 1998: First use of OSC (CAST Synth, etc)
- 2002: OSC 1.0 publish (Wright, Freed)
- 2004: OSC Conference: “Timetag support is a big problem” (Freed)
- 2008: micro-OSC (Schmeder, Freed)

OSC Timestamp Format

- NTP format: 64-bit fixed point
  32-bit uint, #seconds since Jan 1 1900
  32-bit uint, fractions of a sec

Leap Second Problem

1997-06-30 23:59:59.7 UTC -> 867715199.7 xntpd
1997-06-30 23:59:59.8 UTC -> 867715199.8 xntpd
1997-06-30 23:59:59.9 UTC -> 867715199.9 xntpd
1997-06-30 23:59:60.0 UTC -> 867715200.0 xntpd
1997-06-30 23:59:60.1 UTC -> 867715200.1 xntpd
1997-06-30 23:59:60.2 UTC -> 867715200.2 xntpd
1997-06-30 23:59:60.3 UTC -> 867715200.3 xntpd
1997-06-30 23:59:60.4 UTC -> 867715200.4 xntpd
1997-06-30 23:59:60.5 UTC -> 867715200.5 xntpd
1997-06-30 23:59:60.6 UTC -> 867715200.6 xntpd
1997-06-30 23:59:60.7 UTC -> 867715200.7 xntpd
1997-06-30 23:59:60.8 UTC -> 867715200.8 xntpd
1997-06-30 23:59:60.9 UTC -> 867715200.9 xntpd
1997-07-01 00:00:00.0 UTC -> 867715200.0 xntpd
1997-07-01 00:00:00.1 UTC -> 867715200.1 xntpd
1997-07-01 00:00:00.2 UTC -> 867715200.2 xntpd

May occur once every 6-months, depending on Earth rotation rate (cannot be predicted)

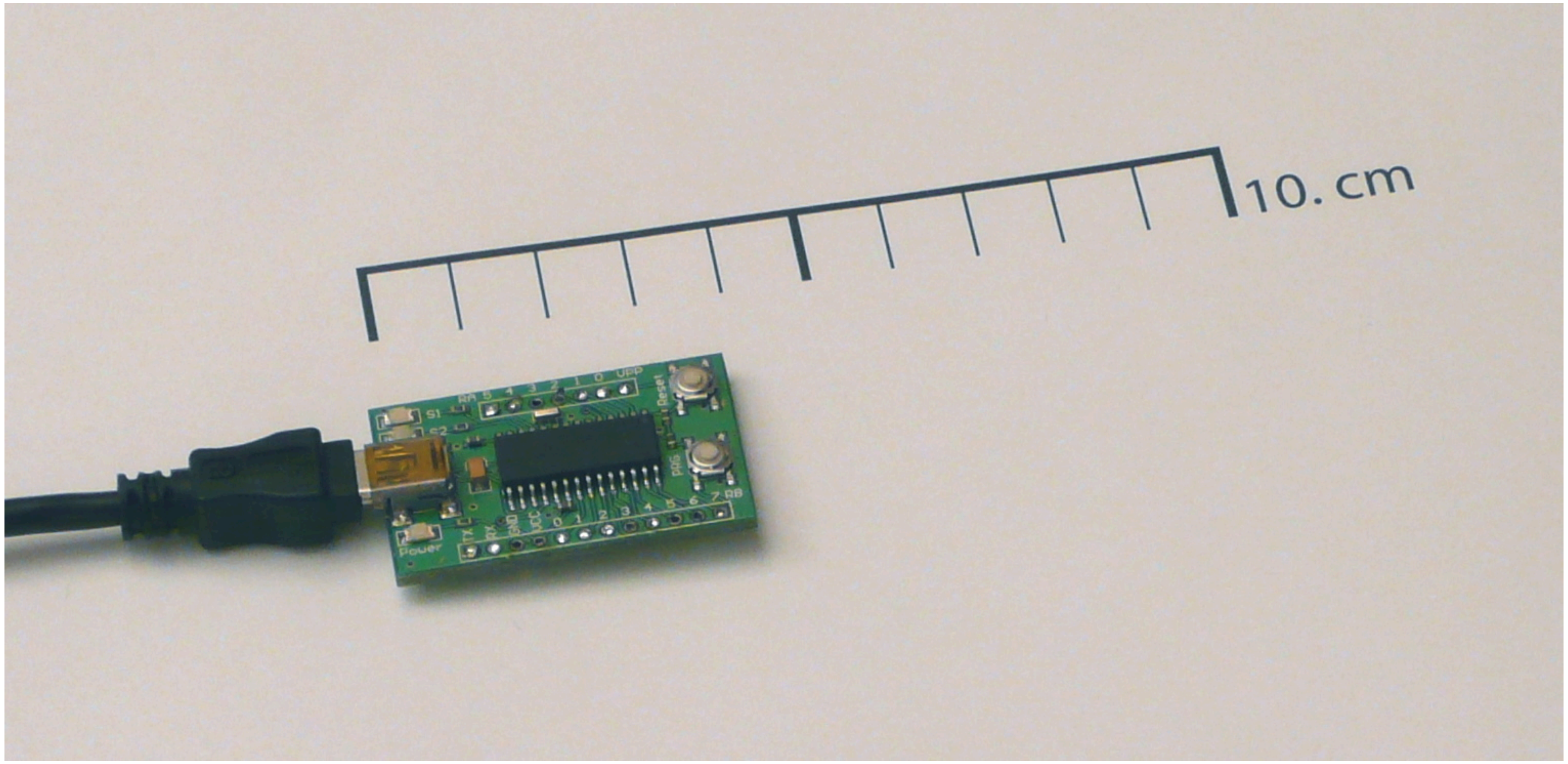
TAI64 Format is Better

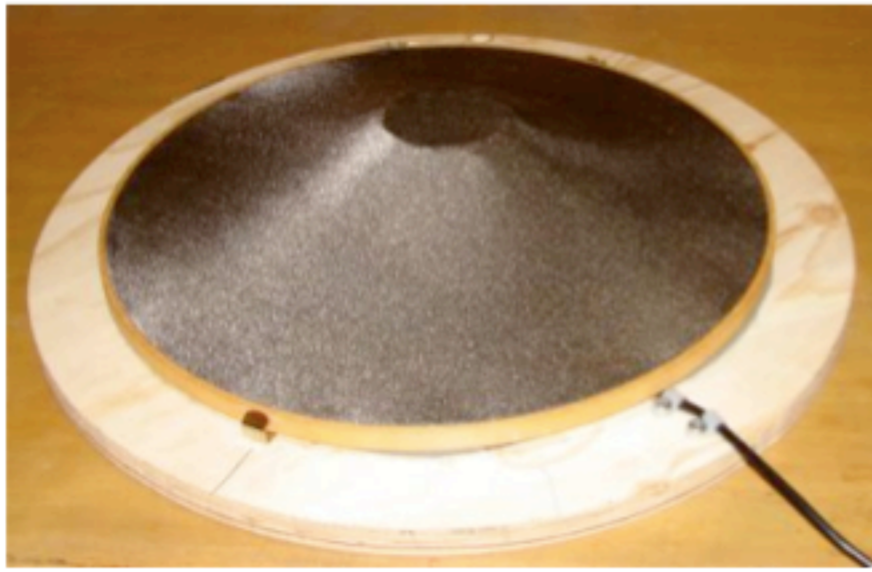
- International Atomic Time
- Strictly monotonic (no leap seconds)
- 64-bit uint #seconds from epoch
- 32-bit uint #nano-seconds (TAI64N)
- 32-bit uint #atto-seconds (TAI64NA)
- libtai (DJ Bernstein)

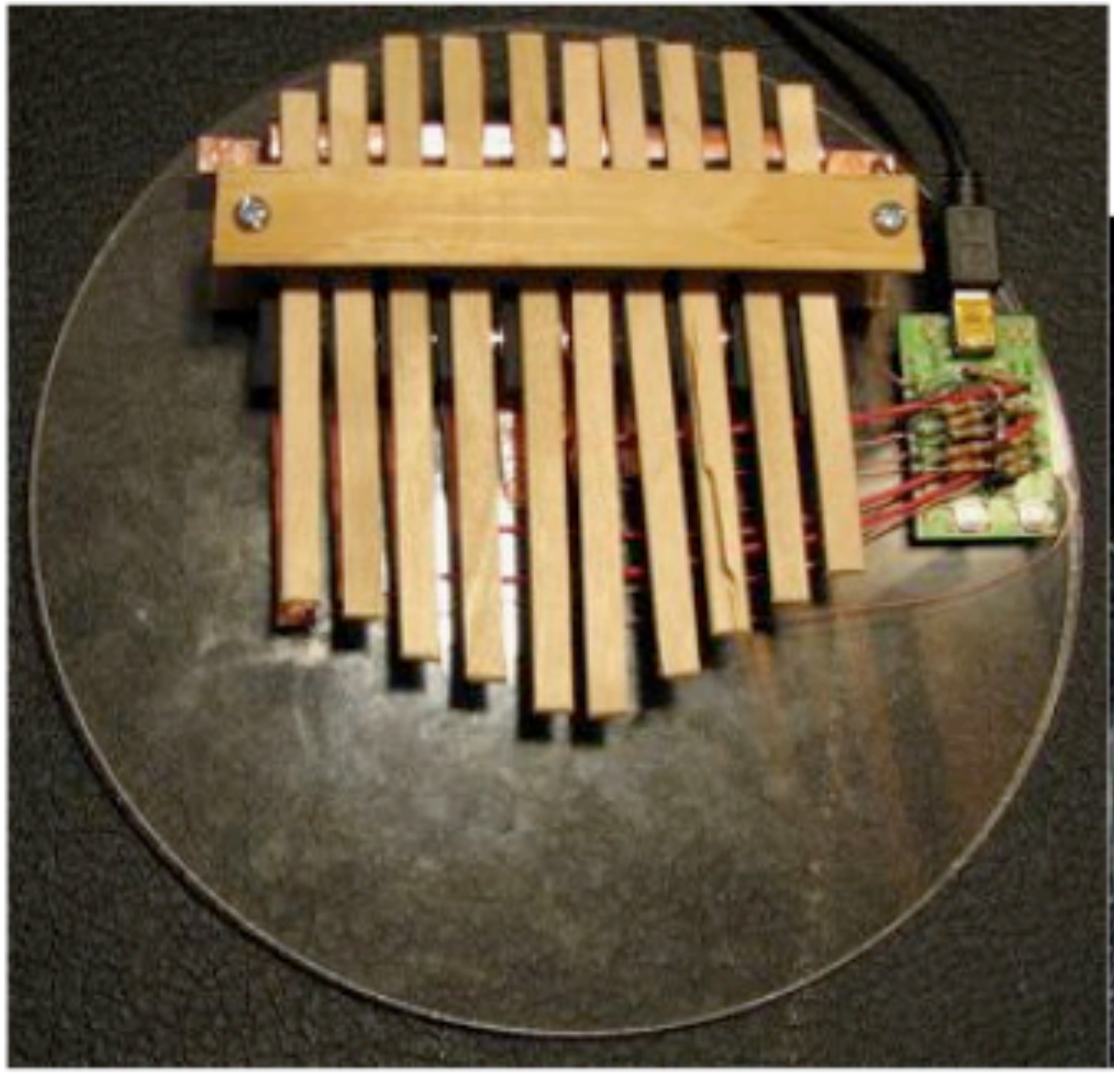
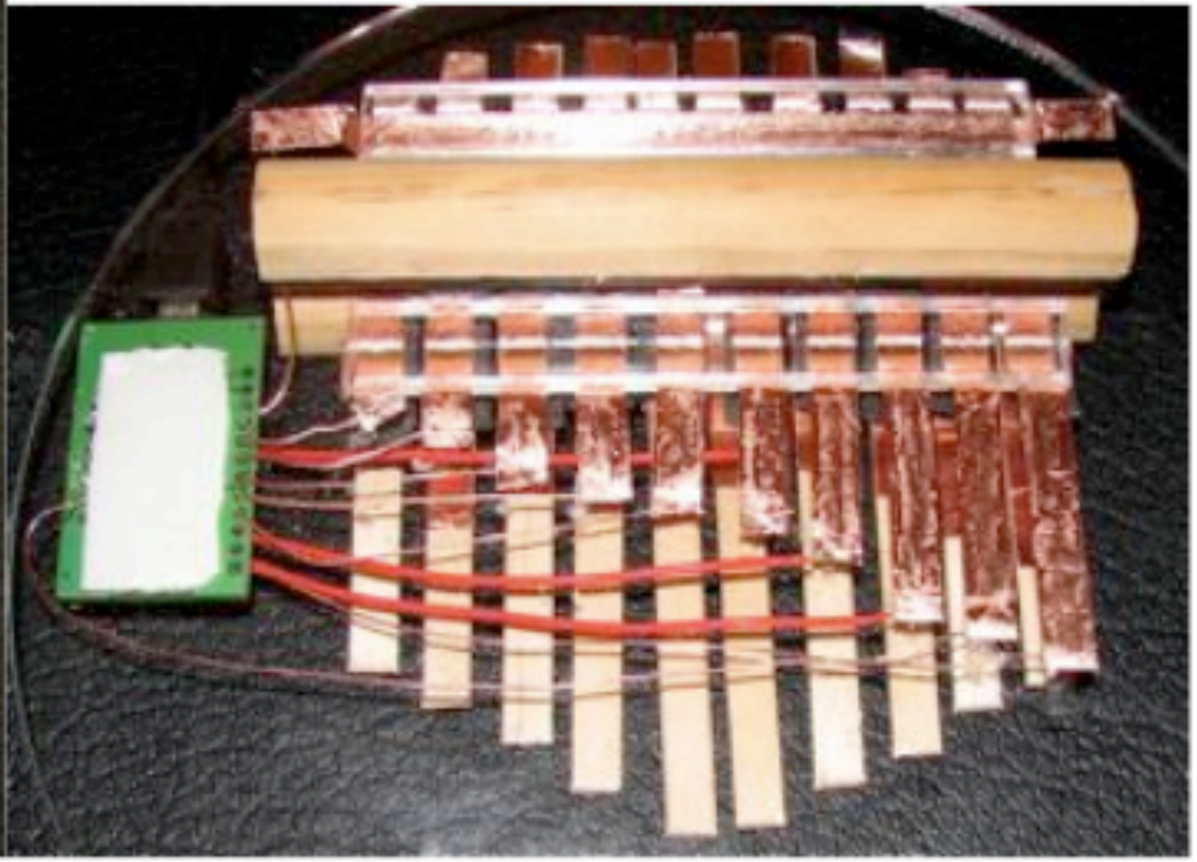
micro-OSC

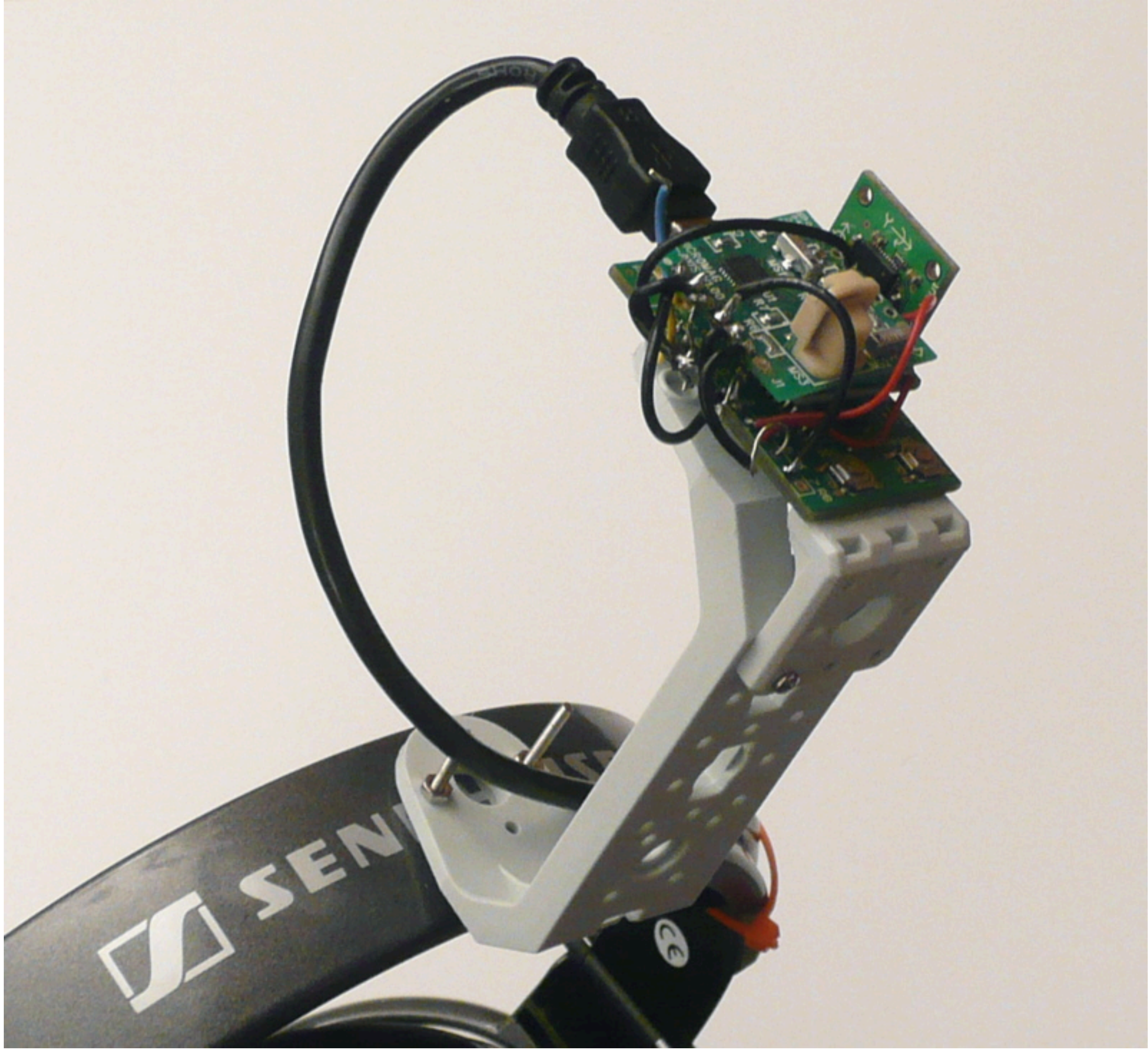
- “OSC in the microcontroller”
- A platform for experimentation with new physical interfaces at CNMAT
- Consideration of requirements for musical gestures
- Replace tedious error-prone programming with on-the-fly messaging
- Low cost for rapid prototyping ($25)

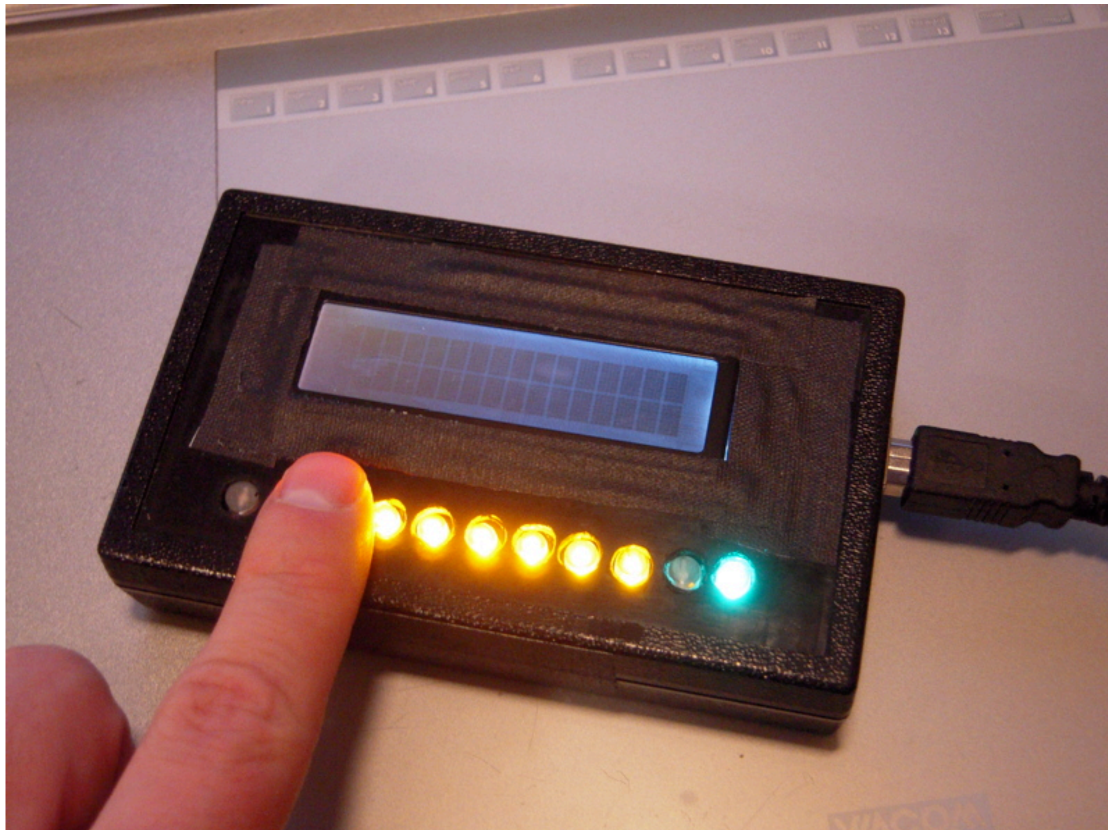
micro-OSC Hardware

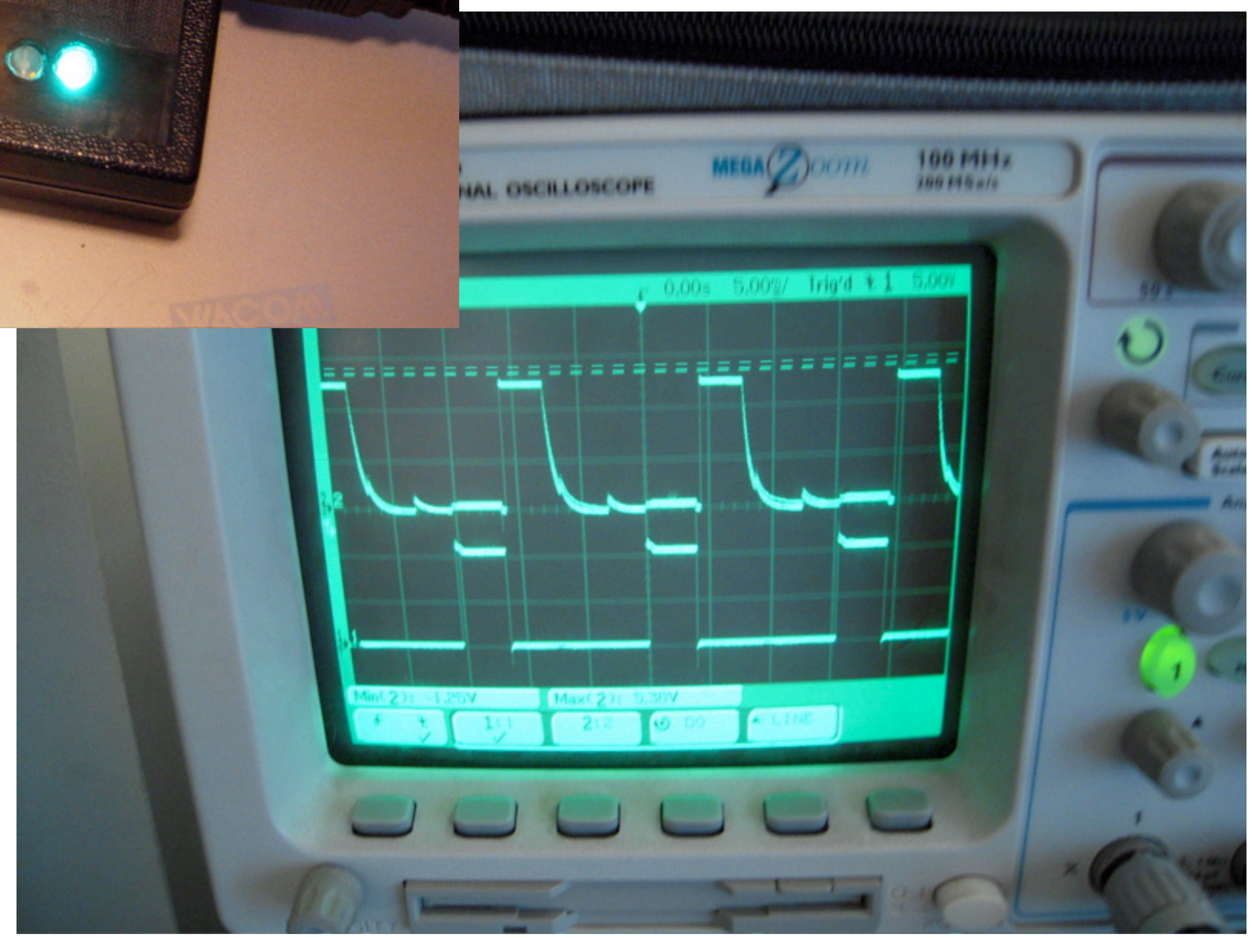


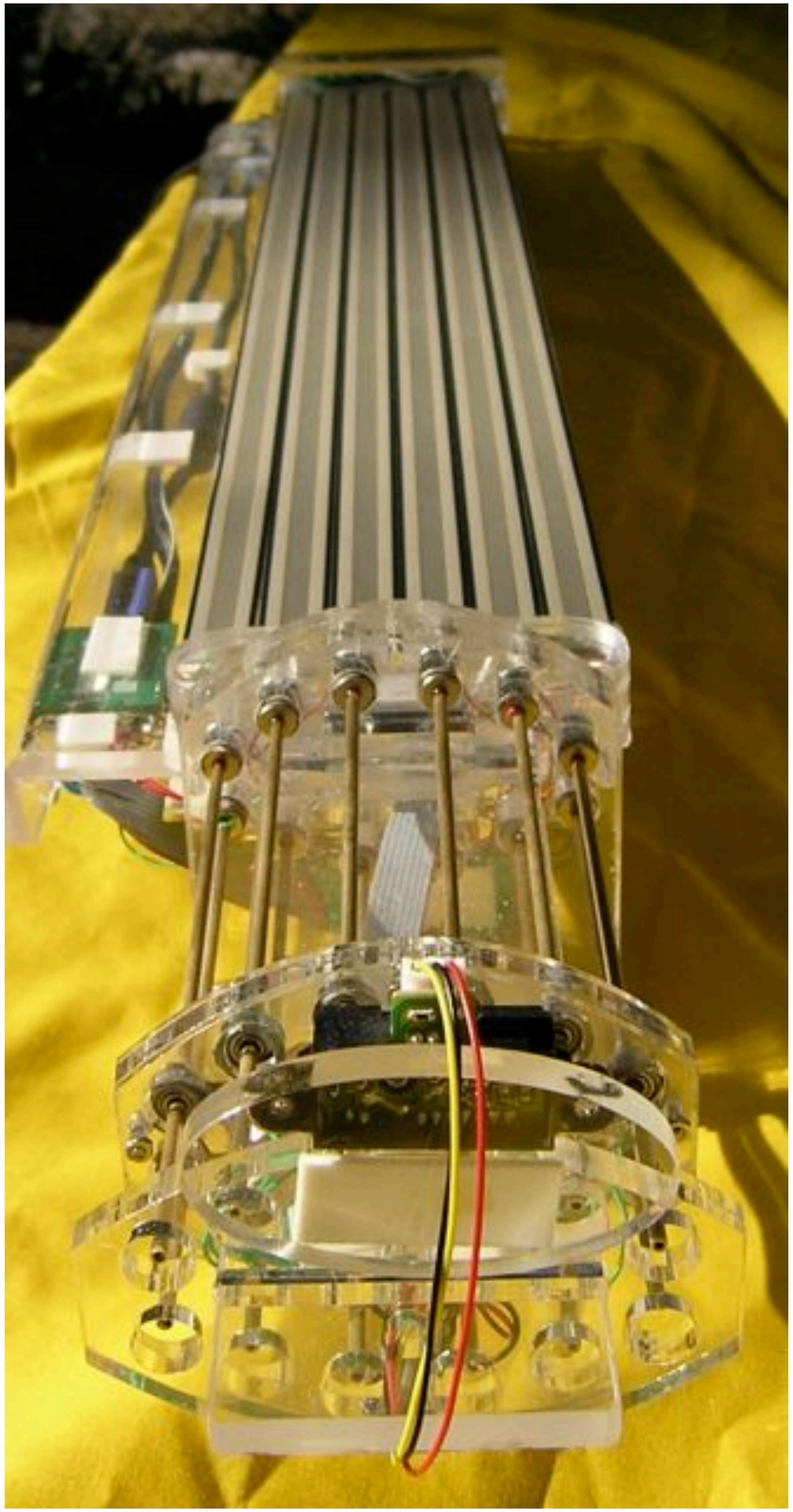


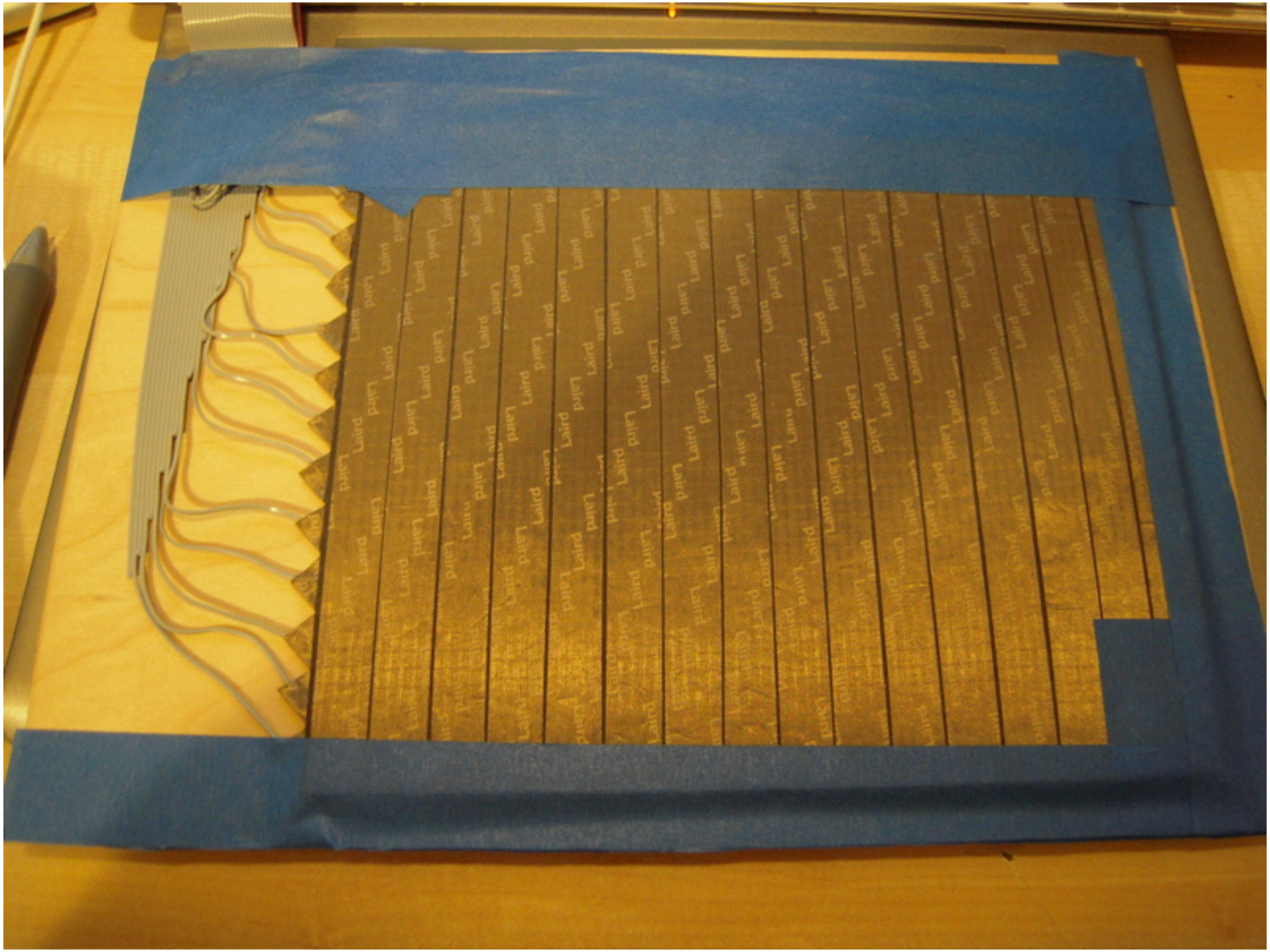








OSCILLOSCOPE
100 MHz



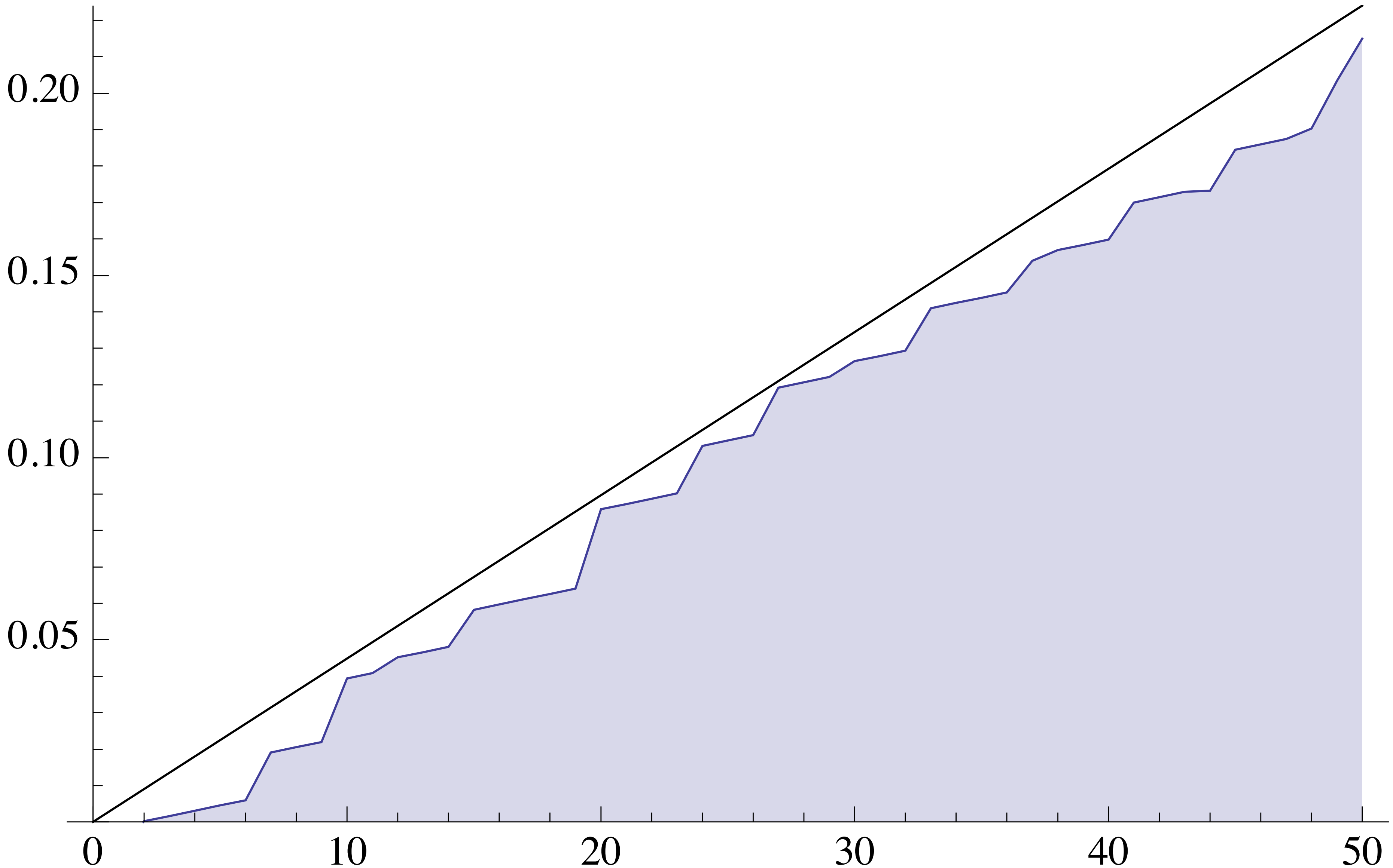

Laird

Musical Gestures...

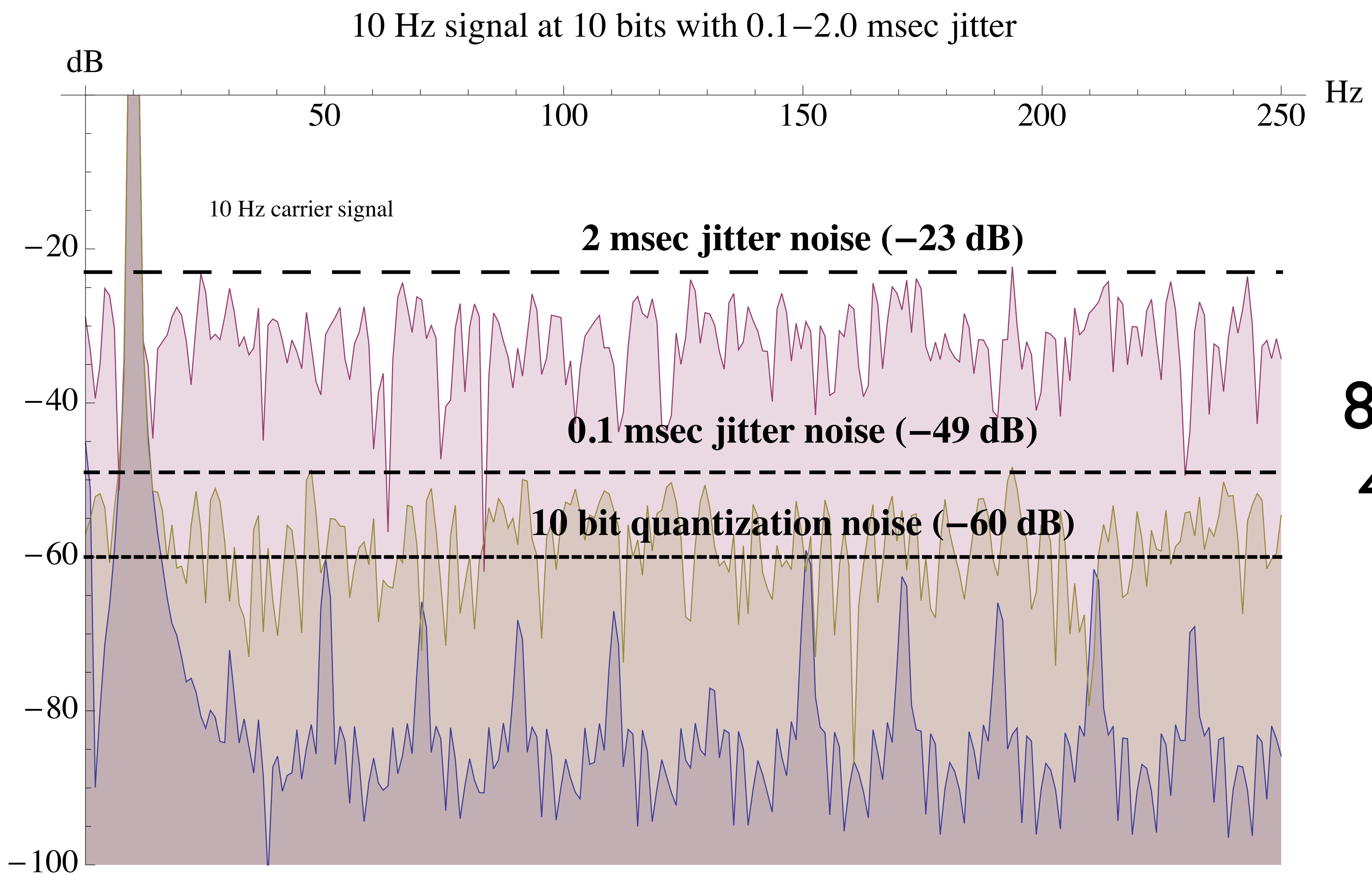
- High temporal precision
  1-20 msec relative event onset precision,
  bandlimit of 50-1000hz
  depends on training, etc
- Wide dynamic range
  pppp - *ffff* is about 8 bits on a linear scale.

Signal Quality Issues

- If gesture stream is inadequately sampled it can have audible consequences, e.g. "zipper noise"
- If latency is too large it can affect performance of rhythmic patterns
- If the entire system is not sufficiently responsive, it can inhibit virtuosity, e.g. "boring to play"



Random delay of 10 +/- 5 msec of input event arrivals observed on a typical operating system



8bits = 48 dB

Spectrum of noise on a 10Hz carrier signal, simulated jitter

	0.01 msec	0.1 msec	1. msec	2. msec	4. msec
0.5 Hz	100.806	80.942	60.5853	54.4588	48.2834
1. Hz	89.4672	69.2973	49.5129	**42.7719**	**37.1899**
2 Hz	83.5256	64.1865	**44.4936**	**37.811**	**32.166**
4 Hz	77.8606	58.3905	**38.2024**	**32.4498**	**25.4497**
8 Hz	72.3401	52.0053	**31.2989**	**25.7653**	**20.1786**
16 Hz	66.1133	**45.8497**	**25.8291**	**19.7408**	**14.3312**
32 Hz	60.2471	**39.6844**	**19.7202**	**13.546**	**8.26448**
64 Hz	53.9285	**33.8882**	**13.9203**	**7.90135**	**1.7457**

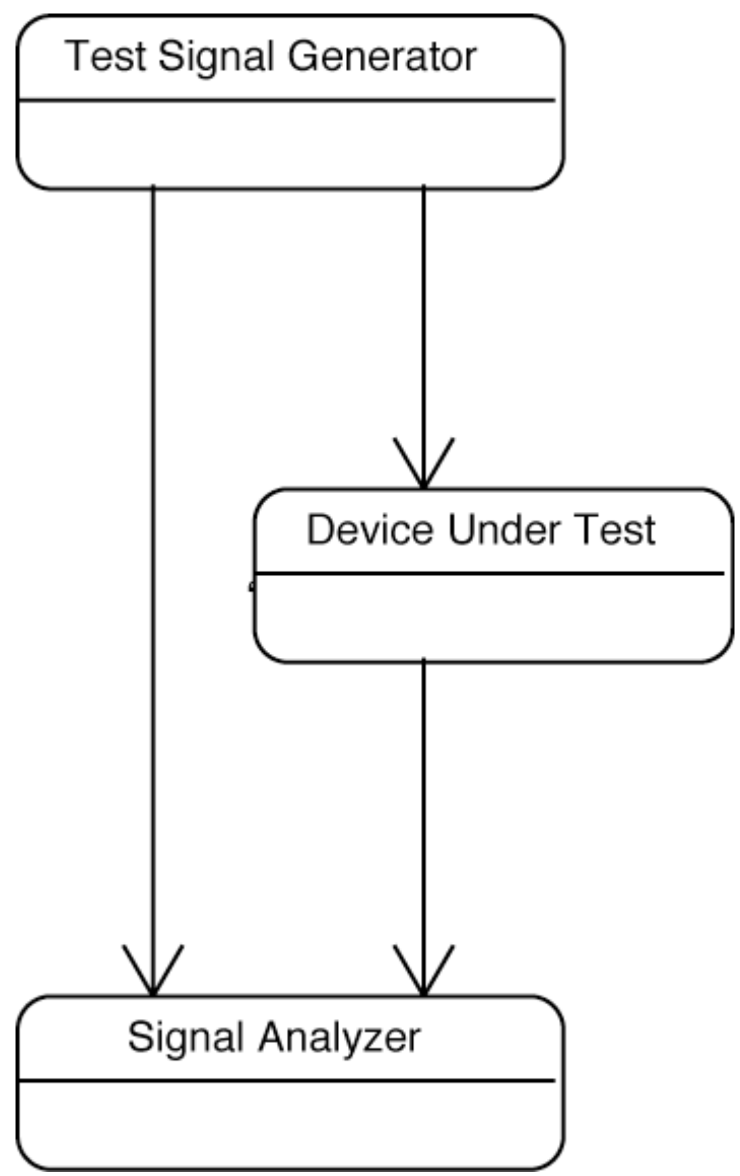
Channel headroom as a function of bandwidth (carrier frequency) vs jitter noise (std deviation of transport delay)

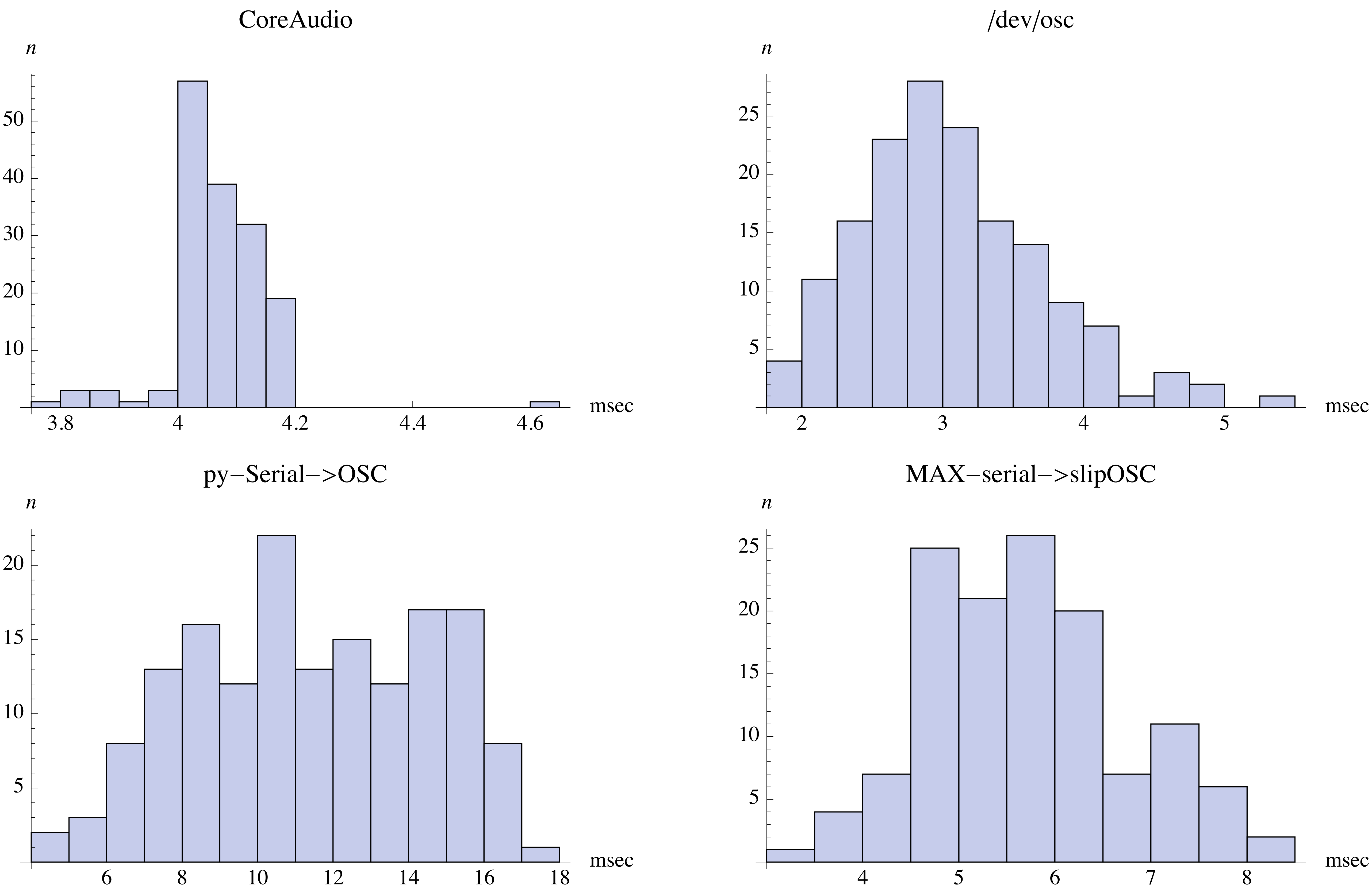
BOLD = <8 bits headroom

Why Jitter Gets Ignored

- The problem goes away at DC
- Irrelevant to pointing tasks (mouse)
- Irrelevant (mostly) to synchronously sampled systems (audio)

Timing measurements with micro-OSC...




CoreAudio
n
10
20
30
40
50
3.8
4
4.2
4.4
4.6
msec
/dev/osc
n
5
10
15
20
25
2
3
4
5
msec
py−Serial−>OSC
n
5
10
15
20
6
8
10
12
14
16
18
msec
MAX−serial−>slipOSC
n
5
10
15
20
25
4
5
6
7
8
msec

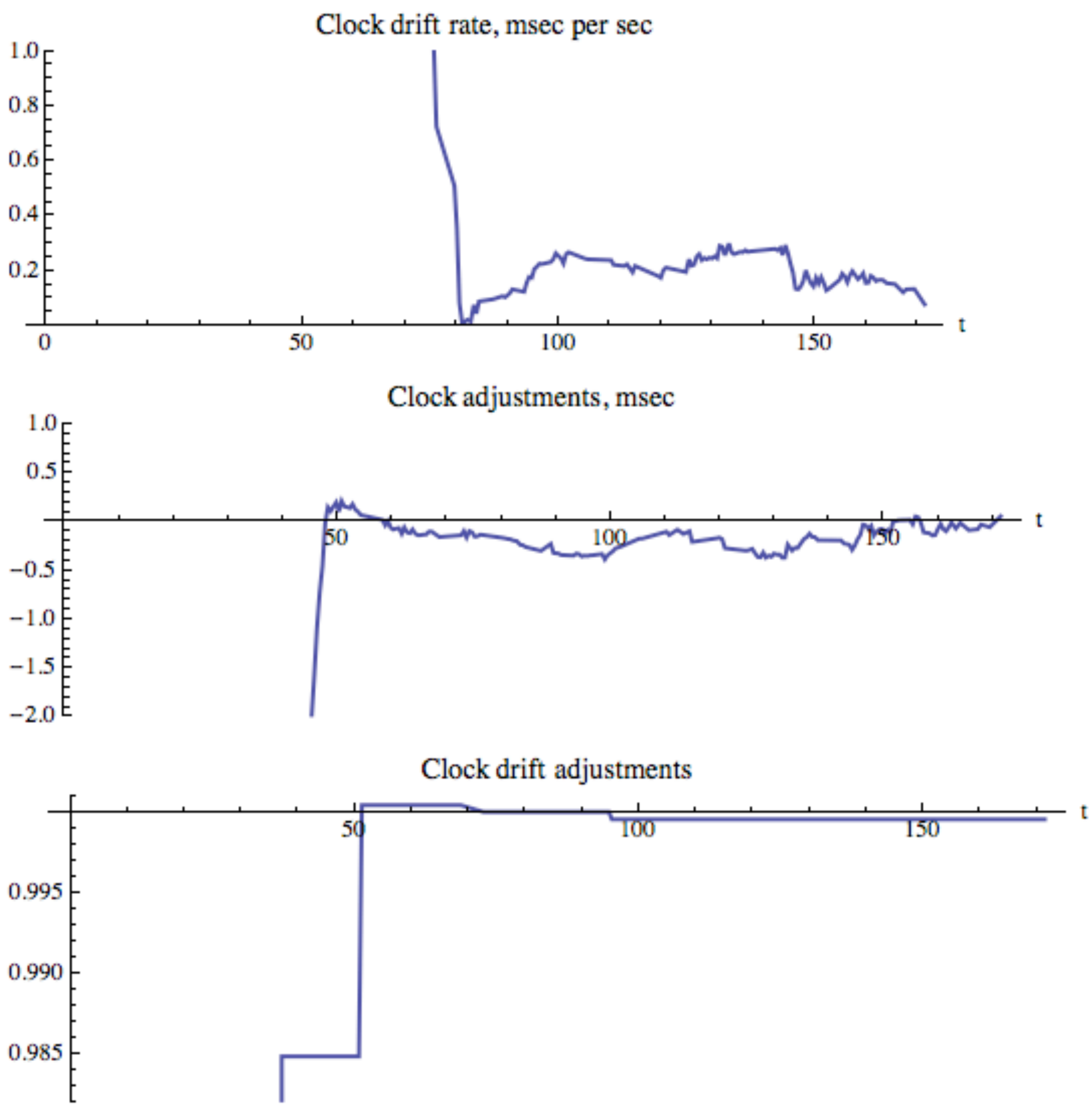
Jitter Attenuation: Step #1) Simple Clock Sync

- Synchronize clock on micro-OSC to host gettimeofday() - 1/2 min(round-trip-delay)
- Timestamp when all data acquisition occurred on the microcontroller
- Re-schedule messages to 1/2 max(round-trip-delay). Trade larger delay for zero jitter.

Clock Sync (Cristian)

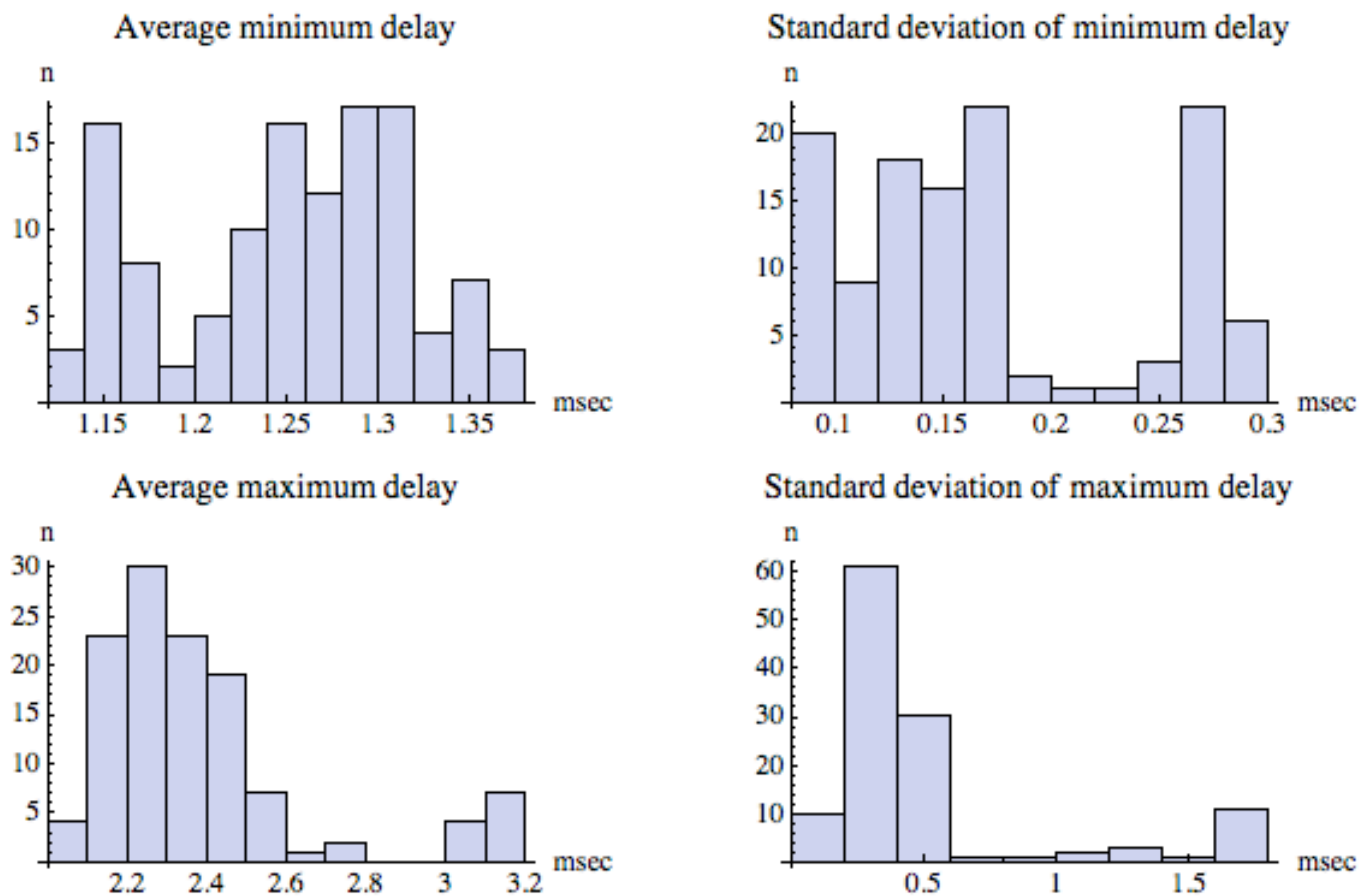
- /time/now -- get the time
- /time/set -- set time (not accurately)
- /time/inc -- add/subtract increments to the clock
  /time/dec
- /time/scale -- adjust scaling coefficient for internal time to external time (drift-rate correction)

Sync Trace

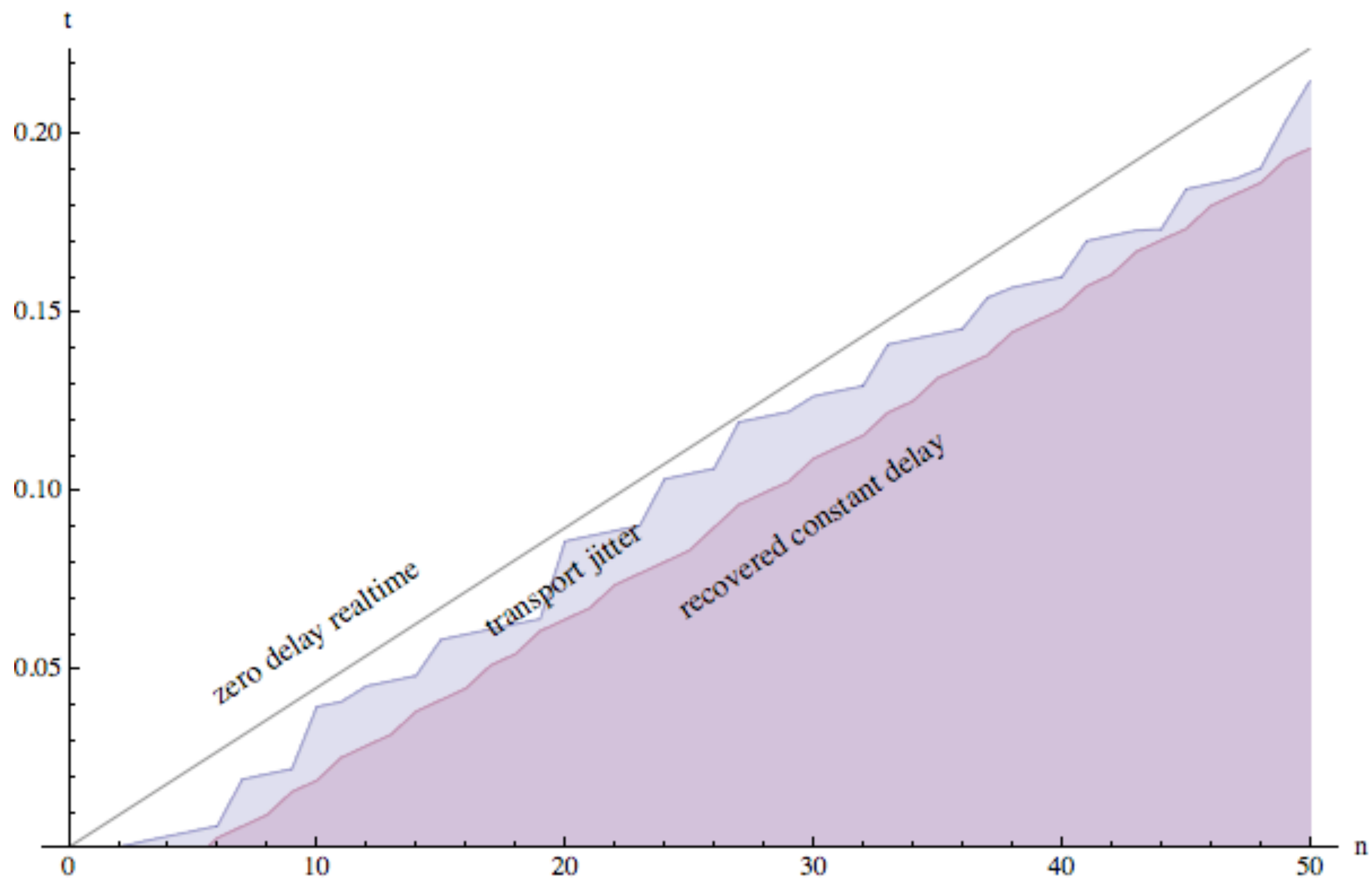


Sync accuracy within within 0.1 msec in 3 minutes between MaxMSP and micro-OSC

1/2 Round Trip Delay Stats



Recovered Timing



Other Uses of Timestamps in micro-OSC

- Clean and accurate way to deal with interrupt-on-change hardware feature
- Flexibility: can't always anticipate how long things will take
- Better instrumentation: easy to do time quality measurement without external testing jig

Synchronization Paradigms

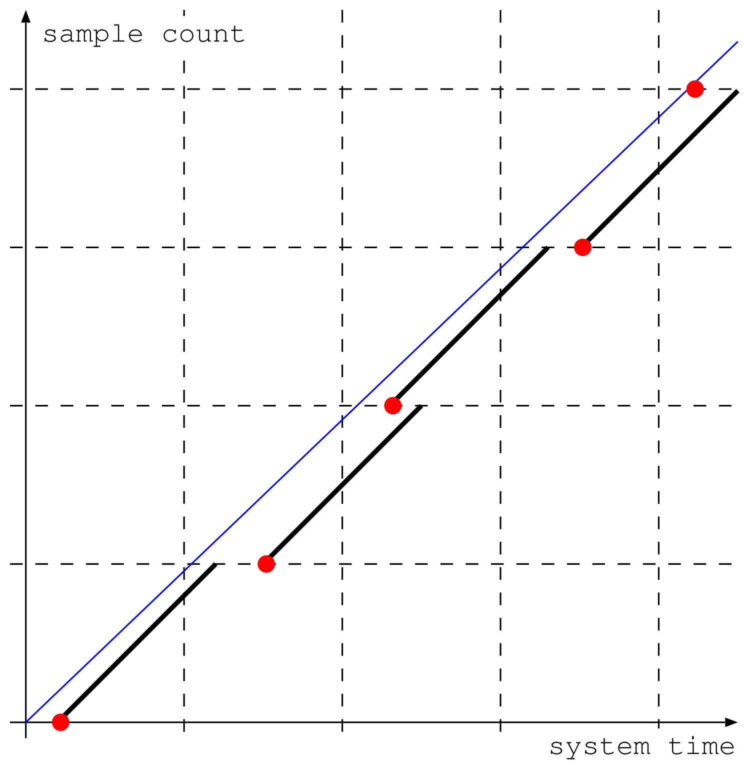
- Forward Sync (Brandt, Dannenberg): Sender anticipates transport delay and timestamps control events for future delivery
- Backward Sync (micro-OSC): Receiver measures transport delay and reschedules for future delivery

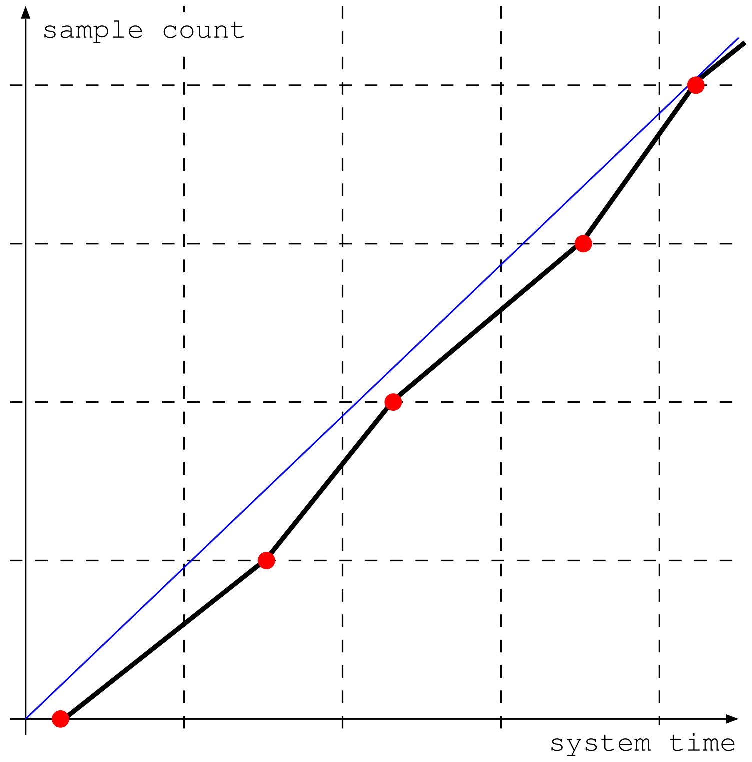
OSC Timestamp Semantic Ambiguity

- Original conception was only the forward sync model
- Turns out in most cases the sender doesn’t know the appropriate delay. Oops!
- No method exists in OSC Bundles to indicate which type of sync is expected.

Jitter Recovery with Sample-Synchronous Accuracy

- Measure delay between audio sample-block callbacks, remove jitter with a 2nd order critically damped IIR filter
- Results: Jitter drops from ~2msec to 10usec





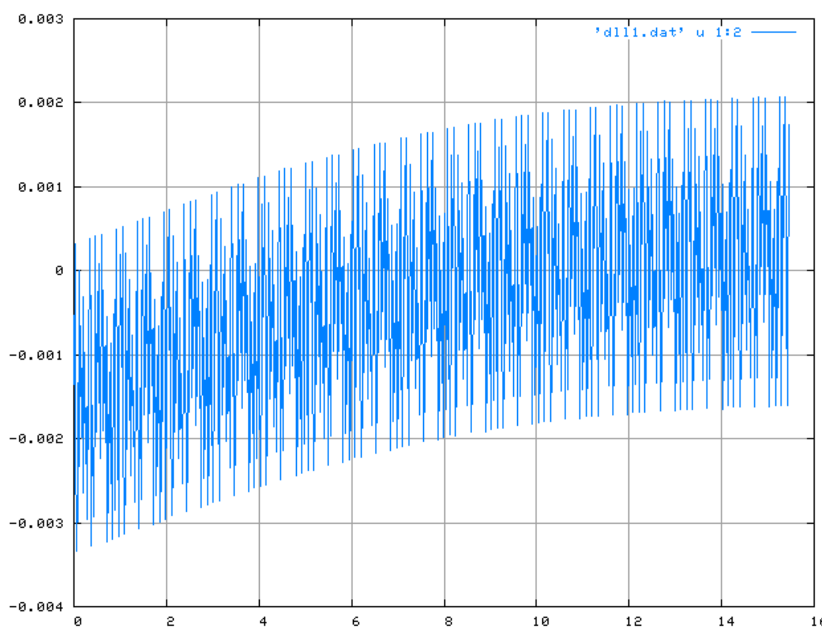


Figure 5: Jitter with USB audio card

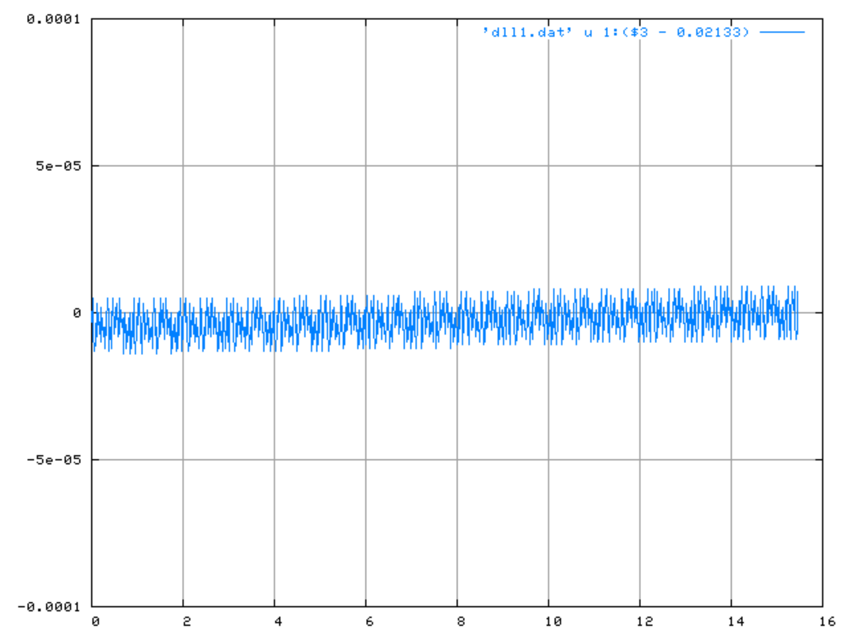


Figure 6: Remaining jitter with DLL filter

- “Using a DLL to Filter Time” (F Adriaensen)

realtime objects in MaxMSP

- Interface between synchronous (audio) and asynchronous (event) computation without loss of timing information
- Globally phase-synchronized oscillators

realtime object library

- realtime.onehz~
  falling edge at start of every real second
- realtime.phasor~
  globally in-phase oscillator
- realtime.edge~
  outputs timestamped events on signal edge detection
- realtime.sig~
  output signal from timestamped events

OSC Timestamps in Databases

- Time-base queries
- Database and file system queries can be treated as transports with large jitter
- Timestamps in the past can be rewritten to current or future on warped scales to implement playback, scrubbing (variable rate playback)
- OSCStreamDB (Schmeder 2009)

Summary

- Gesture signal quality is important for music and audio applications
- Jitter recovery is possible today with micro-second accuracy, provided:
  - Use timestamps everywhere
  - Ensure timestamps are monotonic
- Recording/playback works fine with absolute timestamps (no need to use relative time encodings)

Unsolved Problems

- Choice, encoding, use of sync paradigms (forward, backward).
- Practical issues of dealing with time in programming languages (syntax, semantics).
  - There is no “end-to-end” support for time in current environments
    - ...odot library (Freed, MacCallum)