Action-Sound Latency: Are Our Tools Fast Enough?

Andrew P. McPherson
Centre for Digital Music
School of EECS
Queen Mary U. of London
a.mcpherson@qmul.ac.uk

Robert H. Jack
Centre for Digital Music
School of EECS
Queen Mary U. of London
r.h.jack@qmul.ac.uk

Giulio Moro
Centre for Digital Music
School of EECS
Queen Mary U. of London
g.moro@qmul.ac.uk

ABSTRACT

The importance of low and consistent latency in interactive music systems is well-established. So how do commonly-used tools for creating digital musical instruments and other tangible interfaces perform in terms of latency from user action to sound output? This paper examines several common configurations where a microcontroller (e.g. Arduino) or wireless device communicates with computer-based sound generator (e.g. Max/MSP, Pd). We find that, perhaps surprisingly, almost none of the tested configurations meet generally-accepted guidelines for latency and jitter. To address this limitation, the paper presents a new embedded platform, Bela, which is capable of complex audio and sensor processing at submillisecond latency.

Author Keywords

Latency; jitter; timing accuracy; embodied interaction; musical interaction; embedded hardware.

ACM Classification

H.5.1 [Multimedia Information Systems] Evaluation / methodology, H.5.2 [User Interfaces] Benchmarking, H.5.5 [Sound and Music Computing] Systems

1. INTRODUCTION

In his theory of embodied music cognition [11], Marc Leman holds that the musical instrument becomes a natural extension of the player's body. Through extended practice, the instrument becomes "transparent" to the player who can then focus directly on the music. More broadly, the idea that technological artefacts can become extensions of the body has deep roots in philosophy [15] and support from experimental psychology (e.g. [14]). However, in the domain of digital interactive systems, there is still much to learn about how the design of the tool itself affects the user's incorporation of it into the body schema.

Latency is a fundamental issue affecting digital systems. The delay between a user's action and the corresponding reaction (be it auditory, visual or tactile) can present problems both obvious and subtle. This paper examines the state of play in 2016, measuring whether platforms commonly used to create interactive music systems meet generally-accepted guidelines for latency and jitter (variation in latency).

Where previous papers have focused on latency in a single component (e.g. audio throughput latency [21, 20] and roundtrip network latency [17]), this paper measures the latency of complete platforms commonly employed to create digital musical instruments. Within these platforms we test the most reductive possible setup: producing an audio "click" in response to a change on a single digital or analog pin. This provides a minimum latency measurement for any tool created on that platform; naturally, the designer's application may introduce additional latency through filter group delay, block-based processing or other buffering.

In addition to latency tests on commonly-used platforms, this paper also presents Bela [16], a new hard real-time audio processing platform based on the BeagleBone Black. Bela is capable of synchronous audio and sensor processing with submillisecond latencies and jitter under 50μs.

2. HOW FAST IS "FAST ENOUGH"?

In 2002, Wessel and Wright suggested that digital musical instruments should aim for latency less than 10ms [22]. This figure is perhaps the most common one still used in the community, though Lago and Kon [10] point out that the threshold is highly dependent on musical context and spatial positioning. Percussive interactions are likely to be the most sensitive to latency: humans tapping a steady beat exhibit variations around 4ms [19]. An asynchrony of 6ms in a steady pulse is detectable by listeners [6].

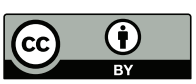
On the other hand, continuous gestural interaction may be less sensitive to latency: 20-30ms was the minimum detectable by theremin performers [13]. In a study of audio-tactile latency, Dahl and Bresin [3] found that in a system with latency, musicians execute their gestures ahead of the beat to align the sound with a metronome, and that they can maintain synchronisation this way up to 55ms latency.

Outside the musical domain, Kaaresoja et al. [9] examine touchscreen buttons, suggesting that latency should be lowest for the tactile channel (5-50ms) followed by audio (20-70ms) and finally visuals (30-85ms). Deber et al. [4] provide a survey of latency in touch interfaces, and show that the just-noticeable difference from touch to visual output is lower for dragging (11ms) than for tapping (69ms).

The analyses in this paper use Wessel's 10ms standard as a point of reference, though we suspect that the ideal threshold may be much lower for percussive musical interactions.

2.1 Jitter

In addition to constant latency, variation in timing (jitter) can present a problem for interactive systems. While users can compensate for static latency [3], unpredictable variation cannot be corrected.



NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.
.

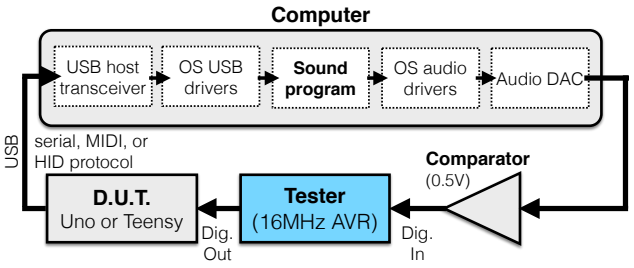


Figure 1: Signal flow of the latency testing setup, showing an Arduino attached to a computer as the device under test. Tests are initiated by the Tester and measure the total reaction time of the DUT and computer.

Measurements of human sensorimotor synchronisation offer clues to the amount of acceptable jitter. Repp and Su [18] provide a detailed review. Notably, the standard deviation of asynchrony is lower for highly trained musicians than for nonmusicians. Fujii et al. [7] find that professional drummers can achieve a mean synchronisation error of 2ms for a metronome at 1000ms and 500ms, and 1ms for a metronome at 300ms, with standard deviations of 10-16ms. Of course, these numbers do not tell us the maximum acceptable amount of jitter, but it stands to reason that any system with jitter less than 1ms would be sufficient for even the most stringent scenarios.

2.2 Previous Device Measurement Work

Latency measurement in sound systems typically focuses on audio input to audio output. In 2010, Wang et al. [21] measured desktop audio latency, finding results ranging from just over 3ms to 70ms depending on the operating system and driver. In 2014, Topliss et al. [20] found latencies as low as 2.5ms on BeagleBone Black. The lowest latencies typically require small hardware buffer sizes which are prone to dropouts as system load increases.

Mitchell et al. [17] in 2014 measured roundtrip network latency for OSC over WiFi networks using the x-OSC board [12], identifying a number of factors affecting performance including network settings, RF interference and network load. They found a mean of 5.3ms under ideal conditions in an RF-controlled environment, rising to 8.1ms under load, with jitter in the range of 1-5ms.

Measuring latency from a sensor input to an audio output is a further challenge. The sensor data often arrives asynchronously to the audio clock, and multiple communication busses and software drivers may separate the streams. Unlike pure audio throughput, the asynchrony of these streams can potentially result in a large amount of jitter.

3. LATENCY EXPERIMENTS

The primary source of latency lies not in the processing speed of the microcontroller but in the communication link, which can be affected by limited bandwidth, OS drivers not optimised for latency, delays and jitter in scheduling within a program. All of these are added to the inherent audio latency of the computer. We examine the effect of all of these factors in this section.

Code for the tester and the devices under test, along with raw data of the results, is available online.[1] The link also contains further tests which do not fit in this paper.

[1] `http://isophonics.net/latency-measurements`. Code is available under a Creative Commons BY-SA 3.0 license.

3.1 Measurement Jig

A latency tester was created using an Arduino Uno. In place of the normal Arduino library calls, low-level GPIO register access and hardware timers were used with interrupts disabled for maximum timing accuracy. Sample results were validated using a digital oscilloscope, showing consistent timing within $\pm 1\mu$s.

Figure 1 shows how the tester attaches to the device under test (DUT). A GPIO output from the tester connects to an input on the DUT. The DUT connects by USB to a computer running audio software. The audio output of the computer runs to a comparator at 0.5V whose digital output goes back to the tester.

3.2 Test Procedure

The measurement procedure for a single test involving a microcontroller and computer is as follows:

1. Tester toggles GPIO pin from low to high;
2. Upon measuring the change, DUT sends a message to the computer via USB;
3. When the computer receives the message, it generates a click at the audio output, consisting of an immediate jump from 0 to 1 followed by a 20ms ramp back to 0;
4. The click produces a low-to-high transition on the comparator which is measured by the tester. The tester records the time from toggling the pin to receiving this response.

A similar procedure applies when the DUT is a wireless link; when the DUT is an embedded computer, it may perform both functions 2 and 3 internally. A test set consists of around 1000 consecutive measurements by the tester, separated with a 250ms delay between tests. The tester sends latency measurements in microseconds via a separate USB-serial link, which is logged to file on the computer.

From 1000+ trials, mean latency can be calculated. To measure jitter, we first discard the upper and lower 2.5% of measurements (to eliminate spurious or transient events) then subtract the shortest from the longest latency.

3.3 Configurations Tested

Our tests were divided into three categories: microcontroller-computer links; single-board computers; and wireless communication links.

3.3.1 Microcontroller to Computer

We tested two microcontrollers: Arduino Uno and Teensy 2.0. Both use a 16MHz AVR. The Uno uses a hardware serial port connected to a separate USB-to-serial converter. The Teensy has native on-chip USB support. For the Uno, we tested serial bitrates of 9600bps and 115200bps. Bitrate is much higher (over 1Mbps), and not adjustable, on the native USB-serial of the Teensy.

On the Teensy, we further tested it in three different USB modes: serial, MIDI and mouse (click button on trigger). This comparison examined the effects of OS drivers and software support on latency.

Sound was generated from the built-in audio device of a MacBook Pro (mid 2014) with a 2.7GHz Core i7 and 16GB of RAM, running MacOS X 10.10.5. Two programs were tested: Max/MSP 6.1.9 and Pd-extended 0.43.4. Max/MSP uses the native CoreAudio drivers; Pd was configured to use the portaudio library [1]. Various audio buffer sizes were compared on Max/MSP; Pd was used at its lowest buffer size of 64 samples.

3.3.2 Single-Board Computers

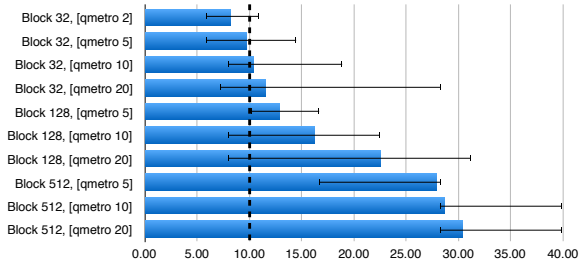


Figure 2: Latency of Arduino Uno connected to Max/MSP for various audio block sizes and serial polling intervals. Dashed line indicates 10ms target; error bars indicate jitter.

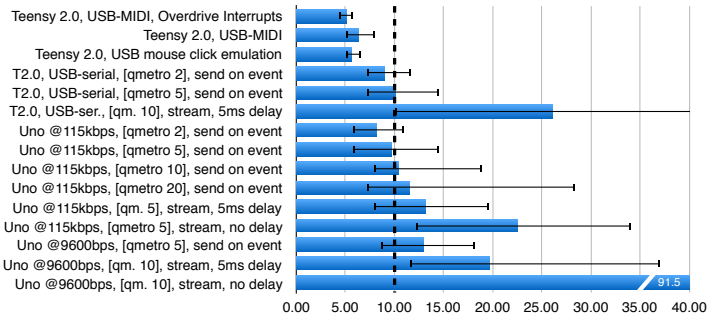


Figure 3: Latency performance of Arduino ("Uno") and Teensy ("T2.0") to Max/MSP; audio block size 32.

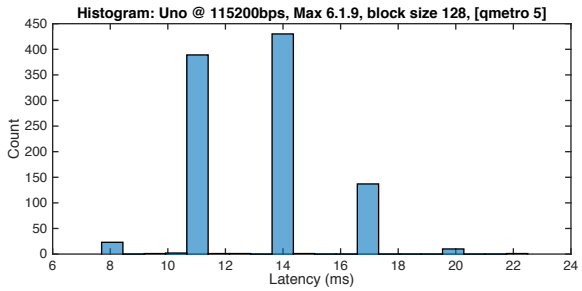


Figure 4: Example latency histogram, showing jitter at intervals of the audio block size.

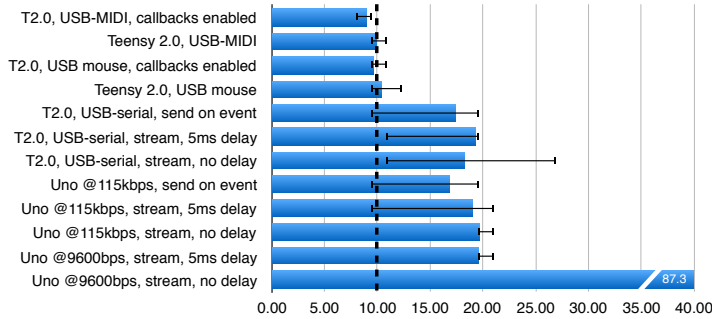


Figure 5: Latency performance of Arduino/Teensy to Pd using portaudio on Mac; audio block size 64 samples.

We tested a Raspberry Pi 2 running Pd-extended 0.43.4 on Satellite CCRMA [2]. We examined GPIO input both natively on the Pi and via an Arduino Uno attached by USB. We also compared the performance of the built-in PWM audio output, the IQaudIO Pi-DAC+ audio board which connects via I2S, and an external USB audio interface (Focusrite Scarlett 8i6). Pd was run with a buffer size of 64 samples with the minimum delay setting that did not lead to dropouts.

We also compared Satellite CCRMA on Pi 2 against the standard Raspbian (Jessie) image running on both Pi 2 and Pi 3. Pd was used with similar settings in each case.

Finally, we tested the Bela [16] single-board computer platform, as discussed separately in Section 5.

3.3.3 Wireless Links

We examined three wireless communication devices: the x-OSC WiFi module [12], a pair of Xbee Series 2 wireless modules, and a Bluetooth Low-Energy (BLE) device (BLEMini from Red Bear Labs). x-OSC testing was performed on a 21-inch iMac (midi 2010) with a 3.2GHz Core i3 and 4GB of RAM, running MacOS X 10.10 and Max/MSP 7.1.0. Several wireless settings were tried, discussed in Section 4.

In the Xbee setup, the tester was attached to one Xbee module acting as a router. The router sent messages to the second Xbee which acted as a coordinator. The router was set to the highest possible sampling rate of 20Hz (50ms intervals). The coordinator Xbee was connected to a second Arduino Uno which parsed the Xbee native output and sent a 1 or 0 over the USB-serial link at 9600bps to Max/MSP running on the iMac.

The BLEmini was tested on the same MacBook Pro described in Section 3.3.1 using the computer's built-in BLE support. Because MacOS X only natively supports BLE keyboards and mice, which was not possible to configure on the BLEmini, a simple helper application (derived from the SimpleControls example by Red Bear Labs) using Apple's IOBluetooth framework received BLE messages from the device and sent an OSC message to Max/MSP running on the same computer.

4. RESULTS

In this section, we examine the latency performance of each set of devices. We compare different settings and configurations of each device and offer recommendations to designers using these devices to create DMIs.

4.1 Microcontroller to Computer

Figure 2 shows latency and jitter for an Arduino Uno connected to Max/MSP for varying audio block sizes. As expected, latency increases with larger audio buffer sizes.

Figure 3 focuses on the smallest block size for Max/MSP (32 samples) and compares different communication protocols. In USB-serial mode, the most common communication protocol for Arduino, none of the configurations consistently met Wessel's 10ms latency target, even on the highest bitrates and the smallest audio buffer sizes. The fastest settings on Max/MSP produced a mean latency of 8.2ms, but jitter resulted in >10ms latency 12% of the time. The serial object in Max/MSP requires polling, and this configuration relied on a 500Hz timer which is unlikely to perform well under load. This suggests that 10ms latency is unattainable in practical situations.

The best case of Teensy 2.0 USB-serial showed 0.7ms more latency (8.9ms) than the Uno, a surprising result given its higher bandwidth, which is perhaps due to driver variations. Using 9600bps on Arduino (still the default in many code examples) adds a further 3.2ms latency in the fastest case tested, explainable by the time it takes to transmit 3 bytes (character + CR + LF) at this bitrate.

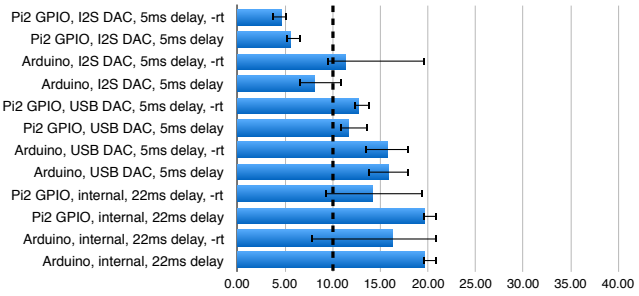


Figure 6: Latency of Pd on Raspberry Pi 2 for internal audio and USB and I2S sound cards, triggered from internal or Arduino GPIO; audio block size 64.

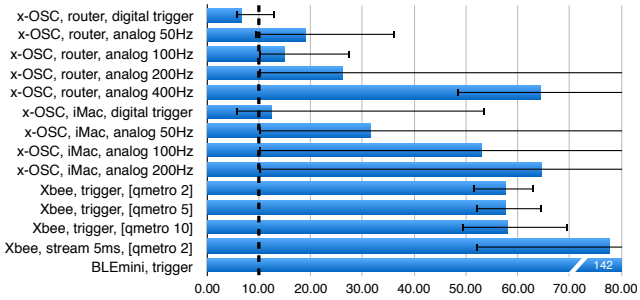


Figure 7: Latency of x-OSC (WiFi), Xbee and BLEmini (Bluetooth LE) wireless boards connected to Max/MSP at a block size of 32 samples.

Using MIDI or mouse events instead of serial resulted in a substantial improvement in performance. The best performance was attained in Max/MSP with a block size of 32, all scheduler overdrives enabled, and a MIDI trigger, resulting in 5.1ms mean latency. Without overdrive settings, mean latency was 6.4ms for MIDI or 5.7ms for mouse clicks.

Latency in Pd (Figure 5) was consistently worse than Max/MSP. The lowest serial latency was 16.8ms (Uno at 115200bps), with Teensy serial 0.5ms slower. Best performance was obtained with MIDI triggers and the "callbacks enabled" option selected in the Pd audio settings, resulting in 9.0ms average latency (9.5ms maximum in any trial).

The difference between Pd and Max/MSP is not explainable by audio buffer sizes alone (64 in Pd vs. 32 in Max). It suggests that another audio pipelining stage exists within Pd, or perhaps in the audio libraries it uses. Pd can run on either portaudio or JACK. We tested a similar setting on JACK (Arduino Uno at 115200bps) and found similar performance (18.7ms latency in JACK vs. 16.8ms for portaudio, with 10ms jitter).

Jitter was above 1ms in every configuration tested. MIDI on Max/MSP exhibited the best performance at 1.5ms jitter, where the serial configurations all showed jitter of 5ms or higher (and sometimes over 20ms). Jitter results for Pd were similar, though two of the 9600bps serial configurations showed high latency (21ms) with low jitter (1.5ms). Histograms of each individual test result (e.g. Figure 4) show that jitter tends to be at intervals of the hardware audio block size, meaning that using smaller audio blocks improves both raw latency and jitter.

4.1.1 Recommendations

The latency performance of the USB-serial interface is at least sporadically over 10ms in every situation tested, but USB-MIDI and HID fared much better. The difference probably lies in OS MIDI drivers being optimised for latency performance. Wherever possible, DMI designers using a microcontroller would be best placed to implement a native USB-MIDI device. If the MIDI specification proves limiting, data could be transmitted by Sysex messages.

If an Arduino Uno is the only available choice, then setting the serial baud rate to 115200bps and reducing the `[qmetro]` polling interval in Max/MSP improves latency performance, even when there is relatively little data being transmitted.

Generally speaking, latency performance is better when data is transmitted only on a trigger, rather than in a continuous stream. Of course, this limits the ability to simultaneously transmit other continuous data streams across the same channel.

Max/MSP shows better latency than Pd. However, other factors including open-source status often drive the decision to choose one tool over another. In either tool, the minimum usable audio buffer size should be chosen.

4.2 Raspberry Pi

Figure 6 shows the results with Pd on Raspberry Pi 2 using Satellite CCRMA. Unlike Pd on the Mac, a nonzero value for Pd's Delay setting was needed for glitch-free operation. The minimum reliable values were chosen for these tests.

Excellent performance can be obtained using the I2S DAC and the internal GPIO of the Rasbperry Pi. Running Pd with the `-rt` flag, the mean latency is 4.6ms (or 5.5ms without the flag). Jitter is 1.5ms in both cases. Neither of the other two audio devices fared well: using the internal PWM sound, a higher Delay value was needed and the best latency performance was 14.2ms with 10ms of jitter. Using the Focusrite USB audio interface, performance of 11.7ms could be achieved with 3.0ms jitter.

The internal GPIO of the Rasberry Pi performed substantially better than the Arduino Uno connected by USB. Only one of the Arduino configurations (I2S DAC, no `-rt` flag) achieved mean latency of under 10ms (8.0ms in this case), and even then, latency was over 10ms 7% of the time. The poor performance of USB for either audio or triggering might be explained by the 1ms frame interval in the full-speed USB protocol used by most devices or by the performance of the OS drivers.

Separately, we tested the default Raspbian Jessie image running on both a Pi 2 and the newer Pi 3. With the best settings (I2S DAC, internal GPIO, `-rt`), the Pi 2 achieved mean latency of 6.2ms (2.9ms jitter) and the Pi 3 achieved 7.7ms latency (4.4ms jitter). This compares with 4.6ms mean (1.5ms jitter) on Satellite CCRMA, demonstrating that Satellite CCRMA does offer noticeably improved performance. At the time of publication, Satellite CCRMA was not available for Pi 3, however it appears that even with the same software, the Pi 3 offers no latency advantage.

4.2.1 Recommendations

Excellent latency performance can be achieved on Raspberry Pi 2 provided an I2S audio codec is used alongside the onboard GPIO, a result which aligns with findings on the BeagleBone Black [20]. Involving USB anywhere in the chain, either for sensor gathering or for audio, significantly degrades performance. In fact, audio performance seems to depend heavily on the particular model of USB audio interface: we earlier tested a Behringer U-Control UCA222 interface and found over 45ms of latency in every case.

Unfortunately, I2S audio boards tend to be more expensive than the cheapest USB audio interfaces, and they occupy the Pi's expansion header. USB audio interfaces still

produce superior results to the onboard PWM audio both in terms of latency and quality. Here again, internal GPIO should be used if possible.

The Raspberry Pi 3 offers processing power advantages over the earlier Pi 2, but it does not improve latency performance. Therefore if Pi 2 is adequate for a given sound synthesis application, no performance benefit should be expected from upgrading to Pi 3.

4.3 Wireless Links

Figure 7 examines several configurations of the x-OSC WiFi board. Three wireless networks were tested: the x-OSC ad hoc network, the iMac shared network, and a consumer Sagecom WiFi router. ([17] recommends the use of an external router.) We found significant packet loss with the x-OSC ad hoc network, so Figure 7 only shows the other two cases. Lost packets resulted in occasional missed triggers in some of the iMac WiFi tests; no missed triggers were observed on the external router. Missed triggers were excluded from the calculations but may have a practical effect on a DMI. All tests were in unicast mode.

The x-OSC board was tested in two modes: digital trigger, where a message is sent only when the digital pin changes values; and analog sampling, where regular samples are taken of an analog input. Analog sampling was tested at 50Hz, 100Hz, 200Hz and 400Hz on the external router. The 400Hz case was excluded from the iMac WiFi because of high packet loss. On both WiFi networks, the best performance was obtained with the digital trigger input: mean latency 6.7ms on the router (95% range 5.7ms to 13.0ms), 12.3ms on the iMac WiFi (range 5.7ms to 57ms).

It might be expected that a faster analog sampling rate would reduce the latency, but the opposite effect was observed. For the iMac WiFi, mean latency ranged from 31ms at 50Hz to 64ms at 200Hz. On the external router, 100Hz sampling produced the best performance, with 14.9ms mean latency (range 10 to 27ms). This is likely due to the network saturation from transmitting high-frequency OSC packets.

Jitter patterns on the x-OSC tend to show a long-tailed distribution, with most tests at a consistent baseline latency, but some tests showing latencies of 50ms or more. Whether this is due to packet loss, delays in WiFi transmission, or limitations of the `[udpreceive]` object in Max/MSP merits further testing.

Figure 7 also shows the results for the Xbee and BLEmini boards. The best-case Xbee performance is 57ms with 11ms jitter, perhaps related to its internal 20Hz sample rate. The BLEmini showed a highly consistent latency of 139ms for nearly all tests, though 1% of tests measured 334ms. It is possible this results from a low-frequency data polling process inside MacOS X.

4.3.1 Recommendations

The x-OSC produces by far the best latency of the wireless options we tested; Xbee and computer-based BLE appear unusable for most DMIs. At its best, x-OSC performance rivals USB MIDI; however, by the nature of WiFi, sporadic packets may be delayed or even lost. These effects are reduced, but not eliminated, by using an external router rather than the built-in network of either the x-OSC or the computer. Though the best latency in our trials was obtained using digital triggers, x-OSC (and WiFi generally) may be better suited for transmitting continuous gestural interaction rather than percussive events which are by nature unforgiving to sporadic delays.

BLE is commonly used to connect to smartphones, with applications including Bluetooth MIDI. We have not tested these configurations but their increasing use in commercial music devices suggests that the performance will be significantly better than we observed for computer-based BLE.

5. BELA: BEAGLEBONE LOW-LATENCY AUDIO

Bela (`http://bela.io`, [16]) is a hard real-time, ultra-low-latency audio and sensor platform for the BeagleBone Black (BBB) single-board computer. The BBB features a 1GHz ARM Cortex-A8 processor with a diverse set of on-board peripherals. Bela uses a custom hardware expansion board ("cape") for the BBB (Figure 8) which contains stereo audio I/O with integrated speaker amplifiers, plus 8 channels each of 16-bit ADC and DAC.

The Bela software environment is based on a Xenomai Linux kernel. Audio processing runs in a Xenomai real-time task which can preempt the kernel itself, allowing buffer sizes as small as 2 audio samples at 44.1kHz (46μs) without underruns, regardless of other system load. Rather than using the ALSA Linux audio drivers, hardware I/O bypasses the kernel using the Programmable Realtime Unit (PRU), a 200MHz on-chip microcontroller which shares memory and peripheral access with the CPU. For further details on the design, see [16].

Bela has been used for a number of digital musical instruments and interactive audio systems, including the D-Box hackable instrument [23], an instrument featuring dynamic tactile feedback [8], and an active vibration control system

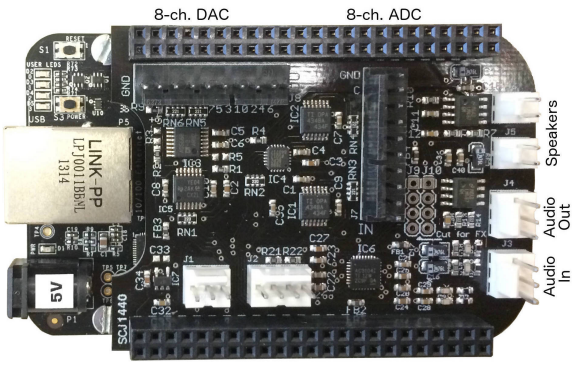


Figure 8: Bela hardware: BeagleBone Black with a custom expansion board for audio and DC-coupled analog I/O.

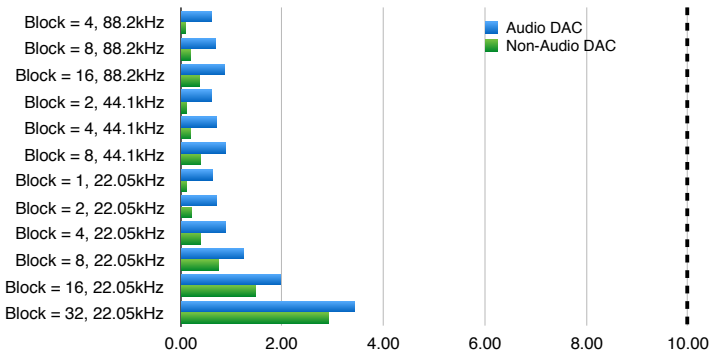


Figure 9: Latency performance of Bela using analog sensor input and generating audio either via sigma-delta ("audio") DAC or AD5668 16-bit ("non-audio") DAC. No error bars as jitter is always less than 0.025ms.

for a musical instrument bridge [5].

Figure 9 shows the latency and jitter performance of Bela triggered by an analog (sensor) input. Using the audio (sigma-delta) DAC, consistent latency around 600μs can be achieved. The dominant latency here is the FIR filter within the sigma-delta DAC itself. Using the analog output instead (which uses an AD5668 string DAC) reduces the latency by 500μs, though this arrangement requires external analog filtering for the best audio quality. Latencies of just over 100μs can be reliably achieved with this DAC.

Because digital, analog and audio I/O is handled synchronously within Bela, jitter is no more than a single sampling period. In practice, every configuration of Bela measured a jitter of 23μs or less, even for larger block sizes.

6. CONCLUSION AND NEXT STEPS

Our results show that over a decade after Wessel and Wright's article [22] set out a 10ms standard for digital musical instruments, many platforms still fall short of this target. Even under a reductive best-case scenario, a microcontroller connected to a computer by USB fails to meet the 10ms action-to-sound latency standard when communication uses a serial protocol, which is a common way of using these tools. This result holds true whether the device has a hardware serial port (Arduino Uno) or native USB Communication Device Class support (Teensy 2.0).

Where the MIDI protocol is used with the smallest audio buffer size, latency can be as low as 5.1ms on Max/MSP, but jitter remains above 1ms. These numbers suggest that the precision of these digital tools in any configuration may still be lower than the precision of skilled human percussionists (cf. [7]) which in turn suggests that these platforms could become the limiting factor in demanding musical situations.

We plan further investigations into the effects of latency for audio and tactile cues, aiming to see how user experience changes when latency and jitter are introduced. We are focusing specifically on the performer experience rather than that of the audience, for whom the speed of sound means that audio cues normally arrive later than visual cues. An ongoing line of enquiry is whether latency erodes the sense of a tool's transparency [11] even before the user becomes consciously aware of a delay.

This paper also evaluated Bela, a platform that eliminates asynchrony between sensors and audio by replacing both microcontroller and computer with a high-performance embedded board running hard real time data processing. With latency under 1ms and jitter around 20μs, Bela meets even the most stringent specification for interactive systems.

7. ACKNOWLEDGMENTS

This work was supported by EPSRC under grants EP/K032046/1 (Hackable Instruments), EP/K009559/1 (Centre for Digital Music Platform Grant) and EP/L019981/1 (Fusing Semantic and Audio Technologies for Intelligent Music Production and Consumption).

8. REFERENCES

[1] R. Bencina and P. Burk. Portaudio–an open source cross platform audio API. In *Proc. ICMC*, 2001.

[2] E. Berdahl and W. Ju. Satellite CCRMA: A musical interaction and sound synthesis platform. In *Proc. New Interfaces for Musical Expression*, 2011.

[3] S. Dahl and R. Bresin. Is the player more influenced by the auditory than the tactile feedback from the instrument? In *Proc. DAFx*, 2001.

[4] J. Deber, R. Jota, C. Forlines, and D. Wigdor. How much faster is fast enough?: User perception of latency & latency improvements in direct and indirect touch. In *Proc. CHI*, 2015.

[5] L. B. Donovan and A. P. McPherson. Active control of a string instrument bridge using the posicast technique. In *Audio Engineering Society Convention 138*, 2015.

[6] A. Friberg and J. Sundberg. Time discrimination in a monotonic, isochronous sequence. *The Journal of the Acoustical Society of America*, 98(5):2524–2531, 1995.

[7] S. Fujii, M. Hirashima, K. Kudo, T. Ohtsuki, Y. Nakamura, and S. Oda. Synchronization error of drum kit playing with a metronome at different tempi by professional drummers. *Music Perception*, 28(5):491–503, 2011.

[8] R. H. Jack, T. Stockman, and A. McPherson. Pitch selection on a digital musical instrument with dynamic tactile feedback. In *Proc. TEI*, 2016.

[9] T. Kaaresoja, S. Brewster, and V. Lantz. Towards the temporally perfect virtual button: touch-feedback simultaneity and perceived quality in mobile touchscreen press interactions. *ACM Transactions on Applied Perception*, 11(2):9, 2014.

[10] N. P. Lago and F. Kon. The quest for low latency. In *Proc. ICMC*, 2004.

[11] M. Leman. *Embodied music cognition and mediation technology*. MIT Press, Cambridge, MA, 2008.

[12] S. Madgwick and T. J. Mitchell. x-OSC: A versatile wireless I/O device for creative/music applications. In *Proc. SMC*, 2013.

[13] T. Mäki-Patola and P. Hämäläinen. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *Proc. International Computer Music Conference*, 2004.

[14] A. Maravita and A. Iriki. Tools for the body (schema). *Trends in cognitive sciences*, 8(2):79–86, 2004.

[15] M. McLuhan. *Understanding media: The extensions of man*. 1964.

[16] A. P. McPherson and V. Zappi. An environment for submillisecond-latency audio and sensor processing on BeagleBone Black. In *Audio Engineering Society Convention 138*, 2015.

[17] T. Mitchell, S. Madgwick, S. Rankine, G. S. Hilton, A. Freed, and A. R. Nix. Making the most of Wi-Fi: Optimisations for robust wireless live music performance. In *Proc. NIME*, pages 251–256, 2014.

[18] B. H. Repp and Y.-H. Su. Sensorimotor synchronization: a review of recent research (2006–2012). *Psychonomic Bulletin & Review*, 20(3):403–452, 2013.

[19] D. Rubine and P. McAvinney. Programmable finger-tracking instrument controllers. *Computer music journal*, pages 26–41, 1990.

[20] J. Topliss, V. Zappi, and A. P. McPherson. Latency performance for real-time audio on BeagleBone Black. In *Proc. Linux Audio Conference*, 2014.

[21] Y. Wang, R. Stables, and J. Reiss. Audio latency measurement for desktop operating systems with onboard soundcards. In *Audio Engineering Society Convention 128*, 2010.

[22] D. Wessel and M. Wright. Problems and prospects for intimate musical control of computers. *Computer Music Journal*, 26(3):11–22, 2002.

[23] V. Zappi and A. McPherson. Design and use of a hackable digital instrument. In *Proc. Live Interfaces*, 2014.