Latency Performance for Real-Time Audio on BeagleBone Black

James William TOPLISS **Victor ZAPPI** **Andrew McPHERSON**
Centre for Digital Music, School of EECS
Queen Mary University of London
Mile End Road
E1 4NS London,
United Kingdom,
j.w.topliss@se10.qmul.ac.uk victor.zappi@qmul.ac.uk a.mcpherson@qmul.ac.uk

Abstract

In this paper we present a set of tests aimed at evaluating the responsiveness of a BeagleBone Black board in real-time interactive audio applications. The default Angstrom Linux distribution was tested without modifying the underlying kernel. Latency measurements and audio quality were compared across the combination of different audio interfaces and audio synthesis models. Data analysis shows that the board is generally characterised by a remarkably high responsiveness; most of the tested configurations are affected by less than 7ms of latency and under-run activity proved to be contained using the correct optimisation techniques.

Keywords

Embedded systems, BeagleBone Black, responsiveness, latency, real-time.

1 Introduction

Research in Music Technology and, in particular, on Digital Musical Instruments (DMIs) is strongly connected to the field of Human-Computer Interaction (HCI). Following the trend of many other disciplines involving HCI, like Ubiquitous Computing [Kranz et al., 2009] and Augmented Reality [Langlotz et al., 2012; Ellsworth and Johnson, 2013], DMI research has recently started capitalising on portable and embedded systems rather than on general purpose architectures. After many years of complete synergy, musical instruments are increasingly abandoning the laptop/desktop computer in favour of onboard audio processing, leaving an important mark in both academia [Berdahl et al., 2013; Oh et al., 2010; Bacławski and Jackowski, 2013] and industry (e.g., Aleph[1], ToneCore DSP[2] and OWL[3]).

This is due to the fact that DMIs require a specific set of design features to provide the user (i.e., a performer, a composer, a casual player) with a musical experience not too far from the one typical of acoustic and electric instruments [Berdahl and Ju, 2011]. This natural comparison with well known "devices", such as piano and guitar, underlines qualities like reconfigurability, independence/autonomy and high responsiveness, which can be assured only on a dedicated system.

As designers and developers of open source novel DMIs, we have decided to explore the promising and evolving world of embedded Linux technologies, focusing as starting point on the concept of *responsiveness*. The work here presented shows the result of a series of tests aimed at measuring the latency of a BeagleBone Black[4] board (BBB), used as the core of a self-contained, open-source musical instrument. Different hardware and software configurations based on the same Linux kernel (v3.8.13) have been analysed under different CPU loads and levels of code optimisation.

This work is part of a larger structured study, whose goal is to assess longevity, usability and reconfigurability of DMIs, compared to the standards of acoustic and electric musical instruments.

2 Related Work

In 2011 Berdahl et al. presented the *Satellite CCRMA* [Berdahl and Ju, 2011], a platform for teaching and practicing interaction design for diverse musical applications, completely based on embedded Linux. It runs on a BeagleBoard[5] coupled with an Arduino Nano[6] and a breadboard, to support the use of sensors and actuators. Two years later, Berdahl et al. upgraded the platform enabling the compatibility with more powerful boards, such as Rasp-

[1] http://monome.org/aleph/
[2] http://line6.com/tcddk/
[3] http://hoxtonowl.com/
[4] http://beagleboard.org/products/beaglebone%20black
[5] http://beagleboard.org/Products/BeagleBoard
[6] http://arduino.cc/en/Main/arduinoBoardNano

berry Pi[7] and BeagleBoard-xM[8], and expanded the range of possible applications including networking capabilities and hardware-accelerated graphics [s[Berdahl et al., 2013]. The project is based on a Fedora distribution with a custom low latency kernel.

A good example of how new generation embedded Linux boards can be used to extend the capabilities of a musical instrument has been recently presented by MacConnell et al. [MacConnell et al., 2013]. This work introduces a BBB-based open framework for autonomous music computing eschewing the use of the laptop on stage. Some important features of embedded systems are here used to provide the instruments designed using the framework with high degrees of autonomy and reconfigurability. Authors include also data regarding the latency of the system running Ubuntu 12.04. Since mainly FX-like processes are addressed, only the audio-throughput time is measured, under different system load configurations. Results vary between 10 to 15 ms, according to the kind of filtering.

The necessity of measuring the responsiveness of computer-based systems is not recent at all, especially in the context of real-time operative systems. In 2002, Abeni et al. used a series of micro-benchmarks to identify major sources of latency in the Linux kernel [Abeni et al., 2002]. They also evaluated its effects on a time-sensitive application, in particular an audio/video player. Moving towards computer-based audio systems, it is worth mentioning the work by Wright et al. [Wright et al., 2004], in which the latency of MacOS, Red Hat Linux (with real-time kernel patches) and Windows XP are compared, both in an audio-throughput configuration and in an event audio-based configuration. The technique used to estimate the event audio latency consists of measuring the time delay between the sound produced by pressing a button on the keyboard and a sinusoidal audio output triggered on the computer by pressing the button itself.

3 System Configuration

The tests presented throughout this work aim at the evaluation of the capabilities of a BBB-based system when used as development platform for DMIs. The chosen configuration includes most of the standard components required to synthesise and control audio in real-time on a self-contained instrument (i.e., without the need of laptops and any additional external devices). Details on each of these components are given in the following subsections.

3.1 Board and OS

The BBB is an embedded Linux board based on a 1GHz ARM Cortex-A8 processor. It is shipped with an embedded Angstrom Linux distribution (v3.8), optimised to run on embedded architectures. This distro is meant to run general purpose applications and it is not specifically audio-oriented. Our first intent was to explore the capabilities of this default board configuration, without introducing any changes in the underlying kernel. We believe this approach could be very useful for the community of embedded developers to have a clear outline of the built-in audio capabilities of the BBB, thus helping choose the right board.

Although belonging to a new generation of compact and fully accessorised boards (e.g., HDMI, uSD card slot), the BBB does not natively provide any audio interfacing. To test the performances of this board in relation to high quality audio synthesis, two commercially available audio interfaces were chosen for comparison; these were both configured for use with the board so as to provide real-time audio output.

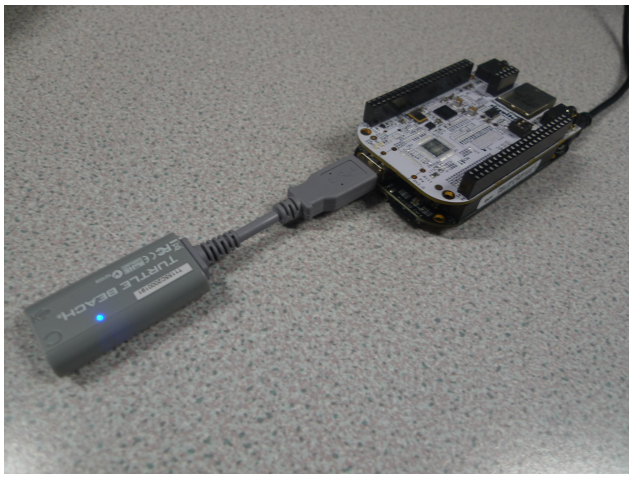

Figure 1: The USB interface and the Audio Cape attached to the BeagleBone Black.

3.2 Audio Interfaces

We tested one USB interface and one BeagleBone expansion "cape" providing audio output.

[7] http://www.raspberrypi.org/

[8] http://beagleboard.org/Products/BeagleBoard-xM

This choice aimed at comparing the two most common solutions used by BBB users for generating audio output. Figure 1 shows both the interfaces attached to the board.

The first interface used is the Turtle Beach Amigo II USB Interface[9]'This device is bus-powered and USB 2.0 class-compliant.. Once attached, this device is automatically recognised as a new hardware interface, meaning that it simply requires being specified as the selected device for audio applications. This interface provides 2 stereo 3.5mm jack receptors, one for input the other for output. Only the latter has been used for our tests.

The second interface used for our tests is the BeagleBone Audio Cape[10]; this device effectively acts as an extension to the BBB, it simply attaches to the top of the board to provide an audio interface. Audio data are exchanged to and from the BBB using an I2S connection. Unlike the USB interface, this device requires some manual configuration to be recognised as a plug-in hardware interface. The on-board HDMI audio virtual cape must be disabled so that the Audio Cape can be loaded by the firmware as the main audio device; this can be easily done by changing the uBoot parameters passed at boot-time. As the USB interface, this cape includes a couple of stereo 3.5mm input/output jack receptors.

3.3 Audio Synthesis

Two different audio backend systems were developed in C++ and cross-compiled to run on the ARM Cortex-A8 processor, one based on ALSA, the other based on JACK. ALSA and JACK implementations are currently adopted by a large number of Linux audio developers.

The audio backend system based on ALSA (Advanced Linux Sound Architecture[11]) essentially comprises an audio engine and a parametric synthesizer. The synthesizer produces frame data; it is connected to the audio engine, which is responsible for collecting and transporting this frame data to the selected output device.

The audio backend system based on JACK (Jack Audio Connection Kit[12]) was similarly designed. A fundamental difference between these two APIs is that JACK uses a client-server model between operating processes and output devices. For this reason, only a synthesizer class was designed to operate as the client process, while a standard JACK server acts as the audio engine for transport. In this configuration the server pulls audio from the client process every time it requires new output data, this is in stark contrast to the ALSA system whereby audio is pushed to the output devices.

Concerning audio synthesis, both the synthesizers implemented in ALSA and JACK generate simple sine waves based on reading from a wavetable. Both systems are configured to provide CD quality audio, (i.e., 16bit resolution, 44.1KHz sample rate) and to run the audio thread at the maximum priority level using a real-time FIFO scheduling.

A parallel control thread was included to manage user input through the keyboard and to have access to the general-purpose in/out pins (GPIO) read/write capabilities of the BBB.

4 Performance Test

The responsiveness and the audio quality of four different specific configurations were tested, combining the use of the 2 audio backends (ALSA an JACK) with the 2 audio devices (USB and Audio Cape). Responsiveness was evaluated considering the latency occurring between the triggering of an audio task producing a waveform and the actual output of the waveform through the audio interface; the assessment of audio quality was connected to the incidence of under-runs.

The performances of each configuration were measured running the audio task in 3 distinct test scenarios. Each of these scenarios (described in the following subsection) involved testing different period and buffer size configurations. As the focus of the test is concerned with very low latency, only the smallest possible period and buffer sizes were examined. In addition, each measurement was repeated enabling 3 different optimisation settings on the C++ cross-compiler (Linaro GCC 4.7 hosted on a x86_64 architecture), using the O1, O2 and O3 flags.

4.1 Test Scenarios

The first scenario involved the generation of a simple monophonic tone. As mentioned in Section 3.2, a simple lookup table was used to generate the frames for this tone.

The second scenario consisted of creating the same monophonic tone as used in the first sce-

[9] www.turtlebeach.com

[10] http://elinux.org/CircuitCo:Audio_Cape_RevA

[11] www.alsa.opensrc.org

[12] www.jackaudio.org

nario, however whilst a `Top` background process was active with a fast refresh rate (passing the command line argument "-d 0.1") (but with standard priority). This scenario allowed for the efficiency of audio synthesis to be observed and measured whilst the system was under heavier load.

The final test scenario was concerned with the generation of a more complex polyphonic tone; this was achieved through the summation of three harmonically related monophonic oscillators. The addition of these two extra tones was considered to be a suitably harder task to synthesise than the simple monophonic tone. It must be noted that no background process was executed during this scenario.

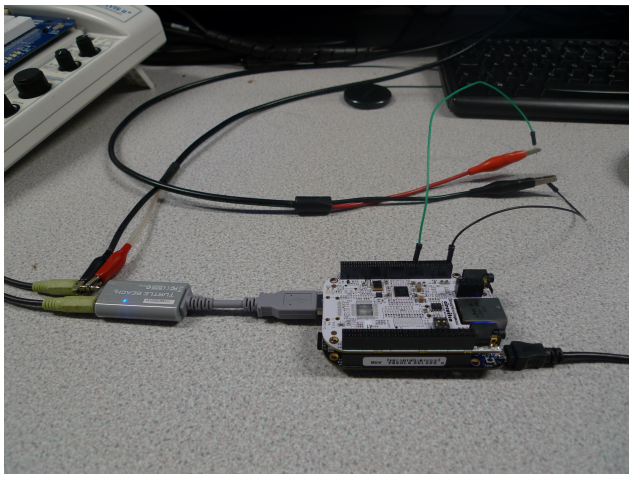

Figure 2: The setup to measure latency when using the USB interface.

4.2 Procedure

The BBB was connected by USB; tests were performed over an ssh connection via BBB's USB network connection. One of the board's GPIOs was attached to the first input channel of an oscilloscope. The audio output was connected to the second channel of the oscilloscope. For the case of the USB interface, the complete setup is shown in Figure 2.

In detail, the test procedure ran as follows. Upon starting one of the executables, the generated system (ALSA or JACK) was programmed to initialise itself but then wait for user input (i.e. a keystroke) before beginning to fill the output buffers with frames. Once the keystroke signal was received across the serial connection, the first task of the system was to drive the GPIO connected to the oscilloscope from low to high. Only immediately after this the audio cycle could begin, outputting the signal into the oscilloscope. The oscilloscope was set to trigger a single display capture on both the channels upon the detection of a rising edge in the GPIO signal. The time distance between the GPIO rising edge in the display and the beginning of the captured audio output hence provided a measurement of the operational latency (Figure 3); each measurement was repeated 5 times [assuming that's right] and an average value calculated.

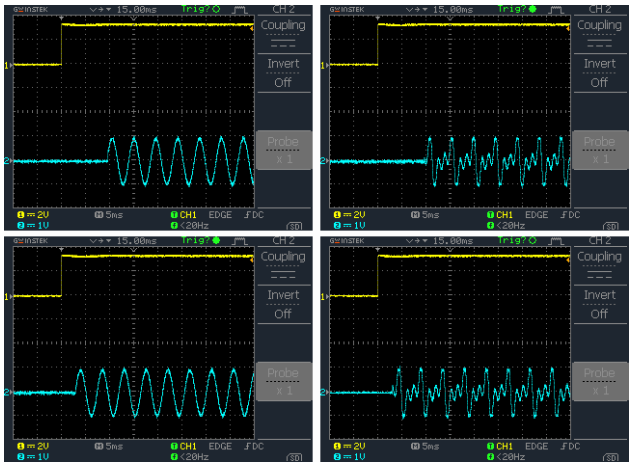

Figure 3: Examples of oscilloscope display capture for scenarios 1 and 3. Latency is measured as the horizontal distance (time gap) between the GPIO signal rising edge (in yellow) and the start of the waveform (in light blue).

It must be noted that the period and buffer sizes chosen were dependent upon both the system type and the target interface; configurations that worked well for one pair did not necessarily run well for another. The six smallest usable configurations were tested for each system and interface pair.

In addition to measuring the output latency, the quality of the output audio was also observed; this observation relied upon noting the frequency of frame dropout (under-run activity) and visible distortion displayed on the oscilloscope (if any).

5 Results

The reported measurements are here presented and discussed, first globally and then analysing more specific cases. Both latency and quality of the output (under-runs) are taken into account and the singular contributions are combined.

5.1 Latency

The latency results were highly consistent across trials: when using the USB audio interface with both ALSA and JACK systems (Fgures 4 and 6) the maximum difference between individual measurements generally only varied by one millisecond, with only few exceptions; this was true regardless of the used optimisation. Measurements concerning the Audio Cape (Figures 5 and 7) were even more consistent than for the USB interface. Across the five measured latencies, measurements only varied by half of a millisecond, without exceptions.

As expected, for all configurations, latency is directly related to buffer and period sizes. No significant, systematic latency differences were noted amongst the three optimisation settings.

However, the choice of monophonic versus polyphonic synthesis and the system load introduce unexpected variations in the latency. These variations may reflect a delay in starting up the ALSA or JACK system, rather than a difference in steady-state latency once the audio rendering is running. Our test procedure toggles a GPIO pin and then immediately begins filling the audio buffers; it is possible that this initial startup produces an additional transient delay compared to reacting to an event once audio is already running. In any case, the difference between test conditions is always less than 1ms.

ALSA - USB Interface

Buffer	Period	O1 Optimisation			O2 Optimisation			O3 Optimisation		
		Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic
		Average Latency (mS)								
128	64	5.6	5.7	5.9	5.2	5.6	5.4	5.6	5.3	5.5
256	64	6.4	6.7	6.4	6.6	6.5	6.4	6.2	6.2	6.6
512	64	7.8	7.9	8.4	7.9	7.9	8.2	8.0	7.8	7.9
256	128	6.3	6.5	6.7	6.5	6.5	6.5	6.5	6.5	6.6
512	128	8.0	8.2	8.4	8.2	8.4	8.1	8.3	7.7	7.8
512	256	9.8	9.7	10.7	8.1	9.9	11.2	9.9	10.7	9.7

Figure 4: ALSA latency measurements for the USB interface.

ALSA - Audio Cape

Buffer	Period	O1 Optimisation			O2 Optimisation			O3 Optimisation		
		Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic
		Average Latency (mS)								
128	8	2.5	2.5	2.5	2.4	2.5	2.5	2.4	2.4	2.5
256	8	3.5	3.5	3.9	3.3	3.5	3.6	3.4	3.4	3.7
512	8	5.6	5.7	6.0	5.7	5.7	5.9	5.5	5.4	5.9
256	16	2.9	2.8	3.0	2.7	2.9	3.0	2.6	2.8	2.9
512	16	4.3	4.5	5.0	4.4	4.2	4.8	4.3	4.5	4.6
512	32	3.5	3.5	4.0	3.5	3.6	4.0	3.5	3.6	4.1

Figure 5: ALSA latency measurements for the Audio Cape.

It can be noted that, on both systems, the Audio Cape allows for smaller buffer and period configurations. Unfortunately, the set of available configurations varies between the 2 systems, making impossible a direct performance comparison. For the only overlapping configuration (buffer 512 - period 32), JACK performed unexpectedly badly, showing a higher latency than configurations based on larger period sizes.

Conversely, USB interface results extend on the same set of configurations for both ALSA and JACK, so that a quantitative comparison is here possible. For the smallest period size (i.e. 64 frames) the JACK system shows better or equal performances, while increasing the size ALSA proved remarkably more responsive. In particular, the last 3 cases listed in Figure 6 shows that JACK's latency is almost the same and quite high, regardless of all the conditions, i.e. buffer/period sizes, test scenario and optimisation level.

JACK - USB Interface

Buffer	Period	O1 Optimisation			O2 Optimisation			O3 Optimisation		
		Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic
		Average Latency (mS)								
128	64	4.5	4.8	4.5	5.0	4.3	4.5	4.5	4.7	4.5
256	64	6.3	7.2	6.7	6.3	7.2	6.7	7.3	6.5	6.5
512	64	6.0	6.2	6.5	6.2	6.5	6.5	6.7	7.0	6.2
256	128	11.8	12.5	12.3	11.8	12.7	10.7	12.7	11.7	11.7
512	128	11.3	11.7	12.0	11.2	11.2	11.0	11.8	11.8	11.3
512	256	11.3	11.3	11.2	11.3	11.3	10.8	11.7	11.8	11.2

Figure 6: JACK latency measurements for the USB interface.

JACK - Audio Cape

Buffer	Period	O1 Optimisation			O2 Optimisation			O3 Optimisation		
		Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic	Monophonic	Mon+bgdproc.	Polyphonic
		Average Latency (mS)								
128	16	3.0	2.8	2.8	3.0	3.0	2.7	2.7	3.0	2.7
128	32	2.2	2.2	2.3	2.2	2.2	2.0	2.5	2.2	2.2
256	32	5.0	5.0	5.2	5.0	5.0	5.0	5.0	5.0	5.0
512	32	11.2	11.0	11.0	11.0	11.0	11.0	11.0	10.8	11.0
256	64	3.8	4.2	4.2	4.3	4.7	4.3	4.0	4.5	4.0
512	64	9.7	9.8	10.0	10.0	10.0	10.0	9.5	10.2	10.7

Figure 7: JACK latency measurements for the Audio Cape.

5.2 Under-runs

The test highlighted a certain incidence of under-runs, whose effects varied according to the chosen configuration. Generally, they occurred in particular when lowest buffer and period sizes were tested. Also the different optimisation settings proved to strongly affect their incidence.

5.2.1 ALSA

Using ALSA on the USB interface, it was observed the occurrence of frame dropout issue only when the buffer and period were set to the minimum values (i.e. respectively 128 and 8 frames). This was true across all the build qual-

ities of the system and for all of the scenarios. During the first scenario (pure tone) and third scenario (polyphonic tone) observed dropouts were not too severe, normally only being exhibited once or twice at the beginning of synthesis. The second scenario seemed to generate significantly higher rates of frame dropout, leading to audible clicks. An interesting observation is that the O2 optimised versions of the system appeared to always exhibit the least amount of under-runs.

In the case of the ALSA system using the Audio Cape, it was observed that frame dropout only occurred whilst using the smallest period size configurations (period size of 8), regardless of the buffer setting. Again this was true across all the build qualities of the system and all of the scenarios. The first and third scenario produced a very small amount of under-run activity, interestingly only for the first period and buffer size configuration tested. Similarly to the USB interface, these observed dropouts were very minor, normally only being exhibited once or twice at the beginning of synthesis. In the second scenario the amount of frame dropout did increase slightly. Interestingly, this time the O3 optimised versions exhibited the best improvements, almost completely preventing all under-runs.

5.2.2 JACK

Since in JACK the stream to the audio device is not managed by the client, under-runs can occur only not he server. In relation to the JACK system using the USB audio interface, most frame dropout issues observed occurred during the first size configurations (buffer 128 - period 16), the second one (buffer 128 - period 32) and the fourth one (buffer 256 - period 32). In regards to the first and third scenarios, the frame dropouts noted for the first period and buffer size configuration were very severe for the O1 and O2 optimisations. The amount of under-run activity for this configuration made it very difficult to gather any measurements for latency; sometimes the JACK server would under-run continuously without the client even being active. The O3 optimisation however did not experience this problem for this configuration; under-runs were noted however were nowhere near as severe. In the case of the second scenario, far less dropouts were observed consistently across all build qualities, a surprising result. Again, O3 optimisation proved to provide the best performance enhancement.

In the case of the JACK system using the Audio Cape, it was noticed that the occurrence of frame dropout appeared more frequently for the first three period and buffer size configurations. The first scenario produced a very small amount of under-run activity, in regards to all the three optimisations; only during the first scenario were any frame dropouts observed. The nature of these under-runs however was different to those previously observed; during the synthesis the server ran smoothly, while under-runs were noted only after the client had been disconnected. It was observed during the second scenario that the amount of frame dropout increased slightly; the type of under-run seen in the first scenario (after the termination of the client) occurred more frequently. This behavior was exhibited in both the first and second period and buffer size configurations for the O1 and O3 optimisations. In the case of the O2 optimisation, this behavior was not observed; instead a severe under-run issue occurred during the second period and buffer size configurations whereby the server immediately began to under-run before the client had even been launched. In relation to the third scenario, similar types of under-run behavior were seen as in the previous two scenarios whereby under-runs occurred after the termination of the client. No frame dropouts were observed for the O2 optimisation for this particular scenario.

6 Conclusion

Throughout this paper we presented a study aimed at evaluating the responsiveness of a Linux embedded system. As part of a larger study on DMIs design, we focused on testing latency and quality of audio output on a BeagleBone Black board running the standard Angstrom distribution with no kernel modifications. Two different audio backend systems were taken into consideration, one based on ALSA, the other on JACK, and measurements using a USB audio interface and an Audio Cape were compared. The test monitored event-to-audio latency and included the monitoring of under-run activity.

Data analysis showed for both ALSA and JACK audio systems remarkably low latency values, especially for small buffer and period size configurations. In particular, the use of the Audio Cape allows for latency values lower than 3ms. In some of the different CPU scenarios taken into consideration the audio stream pre-

sented dropouts and clicks, especially when using small buffer and period size configurations. However, the usage of the different levels of code optimisation available in the chosen compiler (cross-Gcc O1, O2 and O3) completely fixed the audio quality in most of the tested configurations.

No previous works delved into the audio capabilities of the BeagleBone Black while running the default Linux distribution (Angstrom with kernel 3.8). Compared to other distributions, like Ubuntu or Fedora, the usage of Angstrom on the BeagleBone Black proved to support very low latency configurations without the need of a customised kernel. In the context of digital musical instrument design, this feature is remarkable and makes the BeagleBone Black an appealing platform for instrument development.

References

Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, and Jonathan Walpole. 2002. A measurement-based analysis of the real-time performance of linux. In *IEEE Real Time Technology and Applications Symposium*, pages 133–142. IEEE Computer Society.

Krystian Bacławski and Dariusz Jackowski. 2013. Komeda: Framework for interactive algorithmic music on embedded systems. In *Sound and Music Computing Conference*.

Edgar Berdahl and Wendy Ju. 2011. Satellite ccrma: A musical interaction and sound synthesis platform. In *International Conference on New Interfaces for Musical Expression*.

Edgar Berdahl, Spenser Salazar, and Myles Borins. 2013. Embedded networking and hardware-accelerated graphics with satellite ccrma. In *International Conference on New Interfaces for Musical Expression*.

Jeri Ellsworth and Rick Johnson. 2013. Castar - technical illusions. http://technicalillusions.com/?page_id=197.

Matthias Kranz, Albrecht Schmidt, and Paul Holleis. 2009. Embedded interaction - interacting with the internet of things. *IEEE Internet Computing*, 99(1).

Tobias Langlotz, Stefan Mooslechner, Stefanie Zollmann, Claus Degendorfer, Gerhard Reitmayr, and Dieter Schmalstieg. 2012. Sketching up the world: in situ authoring for mobile augmented reality. *Personal and Ubiquitous Computing*, 16(6):623–630.

Duncan MacConnell, Shawn Trail, George Tzanetakis, Peter Driessen, and Wyatt Page. 2013. Reconfigurable autonomous novel guitar effects (range). In *Sound and Music Computing Conference*.

Jieun Oh, Jorge Herrera, Nicholas Bryan, and Ge Wang. 2010. Evolving the mobile phone orchestra. In *International Conference on New Interfaces for Musical Expression*, Sydney, Australia.

Matthew Wright, Ryan J. Cassidy, and Michael F. Zbyszynski. 2004. Audio and gesture latency measurements on linux and osx. In *International Computer Music Conference*, pages 423–429, Miami, FL. International Computer Music Association.