Audio Engineering Society

Convention Paper

Presented at the 138th Convention
2015 May 7–10 Warsaw, Poland

An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black

Andrew P. McPherson[1] and Victor Zappi[2]

[1] *Centre for Digital Music, School of Electronic Engineering and Computer Science, Queen Mary University of London, UK*

[2] *Media and Graphics Interdisciplinary Centre, University of British Columbia, Vancouver, BC, Canada*

Correspondence should be addressed to Andrew P. McPherson (a.mcpherson@qmul.ac.uk)

ABSTRACT

This paper presents a new environment for ultra-low-latency processing of audio and sensor data on embedded hardware. The platform, which is targeted at digital musical instruments and audio effects, is based on the low-cost BeagleBone Black single-board computer. A custom expansion board features stereo audio and 8 channels each of 16-bit ADC and 16-bit DAC for sensors and actuators. In contrast to typical embedded Linux approaches, the platform uses the Xenomai real-time kernel extensions to achieve latency as low as 80 microseconds, making the platform suitable for the most demanding of low-latency audio tasks. The paper presents the hardware, software, evaluation and applications of the system.

1. INTRODUCTION

This paper presents BeagleRT, a new environment for ultra-low-latency processing of audio and sensor data on embedded hardware. Low-cost self-contained processing is valuable for designers of digital musical instruments and real-time audio effects. Using a laptop for performance is not always practical, especially for devices which need to move with a performer: cables can be restrictive and wireless communication links can be unreliable. Moreover, general-purpose operating systems impose a minimum audio latency below which dropouts are likely to occur. In some cases, including live in-ear monitoring, even a few milliseconds latency is detectable by the performer [9]; in other cases, including feedback control systems, microsecond latencies

are needed for mathematical stability [3].

Real-time audio systems can be divided into *hard* and *soft* real-time categories according to whether the timing is guaranteed by design. Audio on general-purpose computers is soft real-time: audio calculations usually hit their deadlines, but system load can cause buffer underruns and, therefore, gaps in the output. Hard real-time, which guarantees that every deadline will be met, is achievable with single-function microcontroller and DSP systems. The system in this paper achieves hard real-time performance using a commodity single-board ARM computer and a custom software environment based on the Xenomai Linux kernel extensions.[1]

1.1. Latency of Audio Systems

Audio latency on general-purpose operating systems can vary wildly; in 2010, Wang et al. [14] found results ranging from just over 3ms on Linux and Mac OS X to over 70ms on certain configurations of Windows. Mobile and embedded platforms are similarly variable, with latency as low as 2.5ms for certain settings of the Linux ALSA sound architecture on BeagleBone Black [12] to hundreds of milliseconds on some versions of the Android OS (as of 2012) [8].

Latency comes from several sources: buffering by the OS and drivers, group delay of interpolation and decimation filters within sigma-delta audio converters [13], and group delay introduced by the end-user audio processing itself (especially where block-based calculations are used). Of these, buffering is the factor most affected by the design of the operating system. Minimum latency measurements can mask performance limitations: while simple audio code might be able to run with small buffers, more demanding code might require a large buffer size and hence a long latency to avoid frequent buffer underruns.

1.2. Embedded Musical Instrument Platforms

Many platforms for creating self-contained musical instruments have been developed in the past few years. These can be divided into high-performance microcontrollers and embedded computers running general-purpose operating systems. Recent microcontroller instrument platforms include CUI32Stem [11], the OWL effects pedal [15], the SPINE toolkit [7] and the Mozzi[2] audio library for Arduino. Other commonly used boards include mbed (mbed.org), Teensy 3.1 (pjrc.com), STM32F4Discovery (st.com) and Arduino Due (arduino.cc).

Other instrument platforms are based on embedded Linux/Unix computers. Mobile phones and tablets are widely used for audio. Satellite CCRMA [1, 2] is a popular and well-supported platform using Raspberry Pi (or BeagleBoard XM) connected to an Arduino microcontroller; sound is generated by audio programs such as Pd and ChucK. Sonic Pi[3] is a live-coding environment for Raspberry Pi intended for classroom use. The original BeagleBone running Pd has also been used for real-time audio [10].

In general, microcontroller platforms offer easy connections to hardware sensors and predictable timing, but have limited computing power. Embedded computers benefit from the ability to use familiar software tools (Pd, SuperCollider, ChucK, etc.) and from the resources of a general-purpose OS, including file I/O and networking. On the other hand, general-purpose operating systems are optimised to balance many simultaneous processes, and may not guarantee audio performance under load.

Since many mobile devices and embedded computers do not provide easy hardware connections for sensors, Arduino and similar microcontrollers are often connected by a serial port. This creates a bottleneck which limits sensor bandwidth (typically to 115.2kbps or less). As a result, sensor data is often sampled infrequently or at low bit resolution. Serial or USB timing uncertainties also contribute to jitter between sensor and audio samples. Each of these effects can reduce the sensitivity of the instrument.

1.3. Goals

BeagleRT aims to combine the best aspects of embedded Linux systems and dedicated microcontrollers for real-time audio. The specific goals are:

[1] In this audio context, the term *hard real-time* is used not in the strictest sense that a single missed deadline is a catastrophic event certified never to occur, but in the looser sense that deadline performance depends only on the content of the audio code and not on any external system factors, and that dropouts should be essentially zero with suitable code. The term *firm real-time* is also sometimes used here.

[2] http://sensorium.github.io/Mozzi/

[3] http://sonic-pi.net

1. Simultaneous stereo audio and multichannel sensor data capture
2. Ultra-low latency, less than 1ms round trip
3. High sensor bandwidth with no bottleneck between sensor and audio processing
4. Jitter-free synchronisation of audio and sensor data
5. Robust performance under load; no buffer underruns due to unrelated processes
6. Lightweight C-based API
7. Self-contained platform suitable for inclusion inside a digital musical instrument

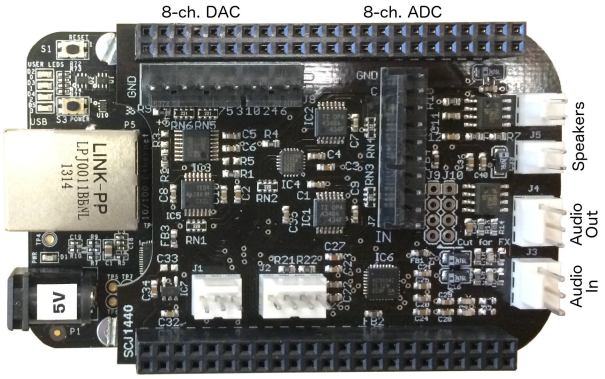

Fig. 1: BeagleRT cape: expansion board containing stereo audio in/out, 8-channel ADC and DAC, stereo speaker amplifiers. BeagleBone Black seen underneath at left.

2. HARDWARE

BeagleRT is based on the BeagleBone Black[4] single-board computer, which contains a 1GHz ARM Cortex-A8 processor with NEON vector floating-point unit, 512MB of RAM and 4GB of onboard storage. The BeagleBone also includes two Programmable Realtime Units (PRUs), 200MHz microcontrollers with access to the same memory and peripherals as the CPU. The PRUs are specifically designed for real-time, timing-sensitive tasks, with most instructions executing in a single 5ns cycle.

Figure 1 shows a custom hardware expansion board ("BeagleRT cape") which provides stereo audio input and output, plus 8 channels each of 16-bit ADC and 16-bit DAC for sensors and actuators. The board also contains onboard stereo 1.1W speaker amplifiers for making self-contained instruments.

2.1. Audio

The audio portion of the cape derives from the schematic of the open-source BeagleBone Audio Cape, revision B[5]. It uses a TLV320AIC3104 codec from Texas Instruments; the codec is capable of up to 96kHz operation though BeagleRT uses it in 44.1kHz mode. The codec includes an onboard headphone amplifier as well as a line output, both of which are accessible on the cape. Like nearly all audio codecs, the TLV320AIC3104 uses sigma-delta modulation, and the internal decimation and interpolation filters introduce 17 and 21 samples of latency, respectively (together, approximately 860μs at 44.1kHz).

2.2. Sensor/Actuator ADC and DAC

Sensor and actuator signals are provided by an AD7699 8-channel ADC and an AD5668 8-channel DAC from Analog Devices. Both ADC and DAC (hereafter termed the *sensor ADC and DAC*) are DC-coupled; the ADC inputs are buffered with op amps, and the DAC outputs can drive up to 30mA, making these signals suitable for a variety of sensor applications.

The ADC uses an SAR-type converter which adds only 2μs sampling latency. Typical settling time for the DAC output is 2.5μs. Therefore, applications requiring near-zero latency are better suited to these parts than the audio codec. However, any antialiasing filters must be implemented externally in analog.

The sensor ADC and DAC share a 24MHz SPI bus; the bus speed sets the upper limit on sample rate. In total, BeagleRT achieves 176 ksps input and output (2.8 Mbps each direction) when synchronised to the audio clock, with selectable configurations of 2, 4 or 8 channels (see Section 3.1). If the ADC and DAC are free running without synchronisation to audio, roughly 50% higher sample rate can be achieved.

[4] http://beagleboard.org/black
[5] http://elinux.org/CircuitCo:Audio_Cape_RevB

3. SOFTWARE

BeagleRT is based on Linux with the Xenomai[6] real-time kernel extensions. In 2010, Brown and Martin [4] found that Xenomai is the best-performing of the hard real-time Linux environments. On BeagleRT, audio processing runs as a Xenomai task with higher priority than the kernel itself, ensuring audio is unaffected by Linux system load.

Running audio at higher priority than the Linux kernel means that kernel hardware drivers cannot be used. We developed a custom driver for the audio codec and the sensor ADC/DAC. The driver uses the BeagleBone PRU to effectively act as a sophisticated DMA (Direct Memory Access) controller. The PRU shuttles data between the hardware and a memory buffer; the Xenomai audio task then processes data from this buffer (Figure 2).

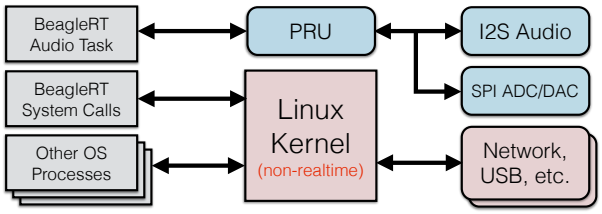

Fig. 2: Operation of the BeagleRT software. The audio task runs under Xenomai, bypassing the kernel using a custom PRU-based hardware driver.

3.1. Sample Rates and Formats

Audio is sampled at 44.1kHz. The PRU also samples each of the 8 sensor ADC and DAC channels at 22.05kHz, a much higher sample rate than typically found in digital musical instruments. This allows capturing subtle details like audio-rate vibrations or detailed temporal profiles within sensor signals. It also means that sensor data is immediately available to the programmer, with no need to request and wait for readings.

Buffer sizes as small as 2 audio samples (= 1 sensor ADC/DAC sample) are supported. This compares favourably with audio buffer sizes of 32 samples or more on typical general-purpose operating systems [14]. Alternative sensor ADC and DAC formats are available: instead of sampling 8 channels at 22.05kHz each, 4 channels can be sampled at 44.1kHz, or 2 channels at 88.2kHz.

3.2. API

BeagleRT is written in C++, but the API for working with audio and sensor data is standard C. Full code is available through the Sound Software project [5].[7] In an arrangement similar to common audio plug-in APIs, the programmer writes a callback function which is called by the BeagleRT system every time a buffer of new samples is required. The callback function provides input and output buffers for both audio and sensor data. For convenience, all data is in `float` format, normalised -1 to 1 for audio, 0 to 1 for sensor ADC/DAC data.

In addition to the audio callback, the API provides initialisation and cleanup functions in which the programmer can allocate and free resources. A simple wrapper is provided to create and manage other Xenomai real-time tasks; the programmer can specify the priority of these tasks, which will always be higher than the Linux OS but lower than the audio rendering task. For example, large block-based calculations might be delegated to a lower-priority task if the audio buffer size is very small, since the entire calculation might not fit in one audio period.

Finally, all of the resources of the standard Linux OS are available at normal (non-realtime) priorities. Xenomai will transparently switch a task from real-time to non-realtime mode whenever an OS call is made. BeagleRT thus offers the performance advantages of a dedicated microcontroller system with the broader feature set of a general-purpose OS.

4. PERFORMANCE AND APPLICATIONS

4.1. Latency

The theoretical round-trip latency of either audio or sensor data is given by twice the buffer length. For example, a simple passthrough program running with a buffer size of 8 audio samples at 44.1kHz will produce 16 samples (0.36ms) of latency from input to output. The TLV320AIC3104 audio codec adds a further 17 samples latency at the input for the decimation filter and 21 samples at the output for the

[6] http://xenomai.org

[7] https://code.soundsoftware.ac.uk/projects/beaglert

Latency: Audio In to Audio Out			
Buffer Size	Predicted (buffer only)	Predicted (+ codec)	Measured
64	2.90ms	3.76ms	3.84ms
32	1.45ms	2.31ms	2.38ms
16	0.73ms	1.59ms	1.66ms
8	0.36ms	1.22ms	1.30ms
4	0.18ms	1.04ms	1.11ms
2	0.09ms	0.95ms	**1.02ms**

Table 1: Audio latency performance of BeagleRT environment under different buffer sizes, f_s = 44.1kHz. Predicted values given without and with 860μs group delay internal to codec.

Latency: Sensor ADC ch. 0 to DAC ch. 0		
Buffer Size	Predicted	Measured
32	2.90ms	2.92ms
16	1.45ms	1.48ms
8	0.73ms	0.76ms
4	0.36ms	0.38ms
2	0.18ms	0.21ms
1	0.091ms	**0.12ms**

Table 2: Sensor ADC/DAC latency performance of BeagleRT environment under different buffer sizes, f_s = 22.05kHz.

interpolation filter (0.86ms in total). The conversion latency for the sensor/actuator ADC and DAC are negligible, about 5μs total.

Actual latency was measured with an oscilloscope and signal generator. In each test, a 50Hz square wave was applied to the input, and the software was configured to pass input to output unchanged. The latency was calculated by measuring the difference in time between edges of the input and output. For tests with the sensor ADC, the edge of the square wave could drift with respect to the ADC sampling period; this artifact is a form of aliasing from the unfiltered test signal. Latency measurements involving this ADC would drift by up to one sampling period (46μs). The mean value is reported for these tests.

Results for audio input/output are reported in Table 1, and sensor results are reported in Table 2. In both cases, the results conform closely to predictions. Minimum audio latency is just over 1ms, mostly due to the sigma-delta codec, while the minimum sensor latency is $120 \pm 23\mu$s.

A hybrid scenario was tested passing audio input to sensor/actuator DAC and, conversely, sensor ADC to audio output (Table 3). The measured latency is approximately halfway between audio and sensor scenarios, with the audio DAC showing greater internal latency than the audio ADC. This result conforms to expectations from the datasheet.

Finally, the effect of different sensor and actuator channels was tested (Table 4). These channels are sampled in a round-robin fashion via the SPI bus, so channel 7 will be sampled almost 7/8 of a period later than channel 0. The lowest latency is obtained by passing ADC channel 7 (the last to be sampled) to DAC channel 0 (the first to be sampled), with this arrangement measuring $80 \pm 23\mu$s.

All of these results compare favourably with 3ms+ latency using the Linux ALSA drivers on the same hardware [12]. 1ms audio latency meets even the most demanding of live monitoring applications [9].

Latency: ADC to Audio Out; Audio In to DAC		
Audio Buffer	ADC to Audio	Audio to DAC
16	1.25ms	1.14ms
8	0.89ms	0.80ms
4	0.71ms	0.62ms
2	**0.62ms**	**0.53ms**

Table 3: Measured latency performance of BeagleRT environment for signals going from sensor ADC to audio output and from audio input to sensor DAC, $f_{s,aud}$ = 44.1kHz, $f_{s,sens}$ = 22.05kHz.

4.2. Performance

Informal testing suggests that audio performance is independent of system load, though the reverse is not true: smaller buffer sizes and more complex audio calculations reduce the computing resources available to the Linux OS. This is expected from the Xenomai implementation. BeagleRT is intended for single-purpose embedded devices, so a penalty on non-realtime system tasks is generally acceptable.

As an approximate metric for overall audio performance, a wavetable oscillator bank was implemented using ARM NEON vector floating point instruc-

Latency: Effect of ADC/DAC Channels		
ADC Channel	DAC Channel	Measured
0	7	0.158ms
0	4	0.144ms
0	0	0.120ms
4	0	0.097ms
7	0	**0.080ms**

Table 4: Latency performance of BeagleRT environment for different combinations of sensor ADC and DAC channels, 1 sensor frame per buffer, f_s = 22.05kHz.

tions.[8] Table 5 shows the results wih a 1024-point wavetable (wavetable size did not have a significant impact on performance). Performance is similar for buffers of 8 samples or larger, plateauing at 740 oscillators above 32 samples. The smallest buffer size of 2 samples shows a performance reduction of about 25%, due to the overhead of frequent task switching.

By comparison, earlier testing using ALSA and C code with compiler optimisation showed a maximum of 74 oscillators with a buffer size of 512 samples.[9]

NEON Oscillator Bank Performance	
Audio Buffer Size	Max # Oscillators
32	740
16	724
8	700
4	648
2	552

Table 5: Maximum number of oscillators before underruns occurred using a 1024-point wavetable oscillator bank.

4.3. Applications

BeagleRT forms the basis for the D-Box hackable musical instrument [16] (Figure 3). The D-Box is a self-contained instrument: a 15cm wooden cube containing two capacitive touch sensors, a pressure sensor, two piezo pickups, a 10cm speaker and a rechargeable battery along with the BeagleBone Black and BeagleRT cape.

[8]Code included in the BeagleRT repository: https://code.soundsoftware.ac.uk/projects/beaglert

[9]J. Topliss, unpublished MEng thesis, Queen Mary University of London

Uniquely, the behaviour of the D-Box can be subverted by rewiring analog circuits on an internal breadboard. Unlike modular synthesisers, the instrument's behaviour is determined by feedback loops between software and circuits, creating unusual results when changed. The feedback loops, which are implemented using the sensor ADC and DAC, are only possible because of BeagleRT's low and predictable latency.

BeagleRT can also be used for active feedback control applications; [6] uses it to apply feedback and feedforward control to a string instrument bridge. Using the sensor ADC and DAC on the same channels, active feedback control up to 3.1kHz bandwidth can be obtained with 45° phase margin.

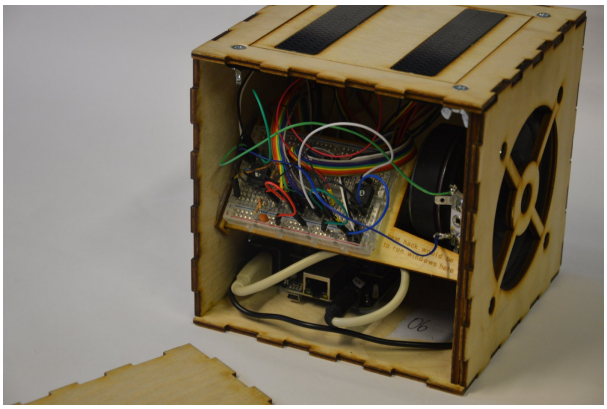

Fig. 3: The D-Box hackable instrument, which uses BeagleRT. The breadboard contains signals from the 8 sensor ADC and DAC channels.

5. CONCLUSION

BeagleRT is a new hardware and software framework for ultra-low-latency audio and sensor data processing which combines the resources of an embedded Linux system with the performance and timing guarantees typically reserved for dedicated DSP chips and microcontrollers. Audio and sensor data are sampled synchronously without the bottleneck imposed by some hybrid microcontroller-plus-computer systems. With hardware buffers as small as 2 audio samples, latencies as small as 1ms for the audio codec and 80μs for the sensor ADC/DAC can be achieved.

Various extensions to the project are possible, in-

cluding synchronous sampling of digital I/O pins, incorporation of real-time MIDI or I2C, and improvements to the communication between PRU (Programmable Realtime Unit) and Xenomai to reduce CPU overhead. The BeagleBone Black is an ideal host on account of its powerful PRUs and numerous GPIO pins, but the same principles could also be applied to other embedded systems.

6. ACKNOWLEDGMENTS

This work was supported by grants EP/K032046/1 and EP/K009559/1 from the UK Engineering and Physical Sciences Research Council.

7. REFERENCES

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

[2] E. Berdahl, S. Salazar, and M. Borins. Embedded networking and hardware-accelerated graphics with satellite ccrma. In *Proc. New Interfaces for Musical Expression*, 2013.

[3] E. Berdahl, H.-C. Steiner, and C. Oldham. Practical hardware and algorithms for creating haptic musical instruments. In *Proc. New Interfaces for Musical Expression*, 2008.

[4] J. H. Brown and B. Martin. How fast is fast enough? choosing between Xenomai and Linux for real-time applications. In *Proc. Real-Time Linux Workshop*, 2010.

[5] C. Cannam, L. A. Figueira, and M. D. Plumbley. Sound software: Towards software reuse in audio and music research. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, pages 2745–2748, 2012.

[6] 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.

[7] A. Hadjakos and S. Waloschek. SPINE: a TUI toolkit and physical computing hybrid. In *Proc. New Interfaces for Musical Expression*, 2014.

[8] V. Lazzarini, S. Yi, and J. Timoney. Digital audio effects on mobile platforms. 2012.

[9] M. Lester and J. Boley. The effects of latency on live sound monitoring. In *Audio Engineering Society Convention 123*, 2007.

[10] D. MacConnell, S. Trail, G. Tzanetakis, P. Driessen, and W. Page. Reconfigurable autonomous novel guitar effects (RANGE). In *Proc. Sound and Music Computing*, 2013.

[11] D. Overholt. Musical interaction design with the CUI32Stem: Wireless options and the GROVE system for prototyping new interfaces. In *Proc. New Interfaces for Musical Expression*, 2012.

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

[13] Y. Wang. Latency measurements of audio sigma delta analog to digital and digital to analog converts. In *Audio Engineering Society Convention 131*, 2011.

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

[15] T. Webster, G. LeNost, and M. Klang. The OWL programmable stage effects pedal: Revising the concept of the onstage computer for live music performance. In *Proc. New Interfaces for Musical Expression*, 2014.

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