Making High-Performance Embedded Instruments with Bela and Pure Data

Giulio Moro, Astrid Bin, Robert H. Jack, Christian Heinrichs, Andrew P. McPherson

Centre for Digital Music, Queen Mary University of London, UK

g.moro@qmul.ac.uk, a.bin@qmul.ac.uk, r.h.jack@qmul.ac.uk, c.heinrichs@qmul.ac.uk, a.mcpherson@qmul.ac.uk

Abstract. Bela is an embedded platform for ultra-low latency audio and sensor processing. We present here the hardware and software features of Bela with particular focus on its integration with Pure Data. Sensor inputs on Bela are sampled at audio rate, which opens to the possibility of doing signal processing using Pure Data's audio-rate objects.

Keywords: embedded audio, sonic interaction design, sensors, low latency, musical instrument design.

Introduction

The increasing power and availability of microcontrollers and single-board computers has given rise to many new platforms for creating musical instruments and platforms for interactive audio. Choosing a suitable platform can be a challenge, involving tradeoffs between computing power, hardware connectivity, ease of programming and price.

Many current approaches to designing sounding objects combine two or more devices together, for instance an Arduino which handles analog and digital sensor input communicating via USB-serial with a a computer running the audio processing. Using a self-contained embedded platform in the creation of DMIs and interactive audio systems has several advantages over such a setup.

- **Reliability** Using a single device is less prone to communication errors and it is easier to provide a backup solution for a simpler system.
- **Performance** With a composite setup as the one described above, the serial connection is slow and the throughput is limited. As such, the latency, sampling rate and jitter of the acquired data are all affected negatively, which may in turn affect the expressiveness of the performance. MIDI devices typically perform better than serial ones, while wireless links may be affected by packet loss or channel congestion (McPherson, Jack, and Moro 2016).
- **Reproducibility** It is easier for other people to recreate a device if it does not rely on multiple pieces of software and hardware devices and specific revisions of each of them.
- **Sustainability** Similarly, the developers themselves will find it easier to maintain and develop a system that does not have multiple dependencies, also to the advantage of making software version control easier.

Recent embedded platforms for Digital Musical Instrument creation

Arduino and similar boards are an accessible way of providing low-level connectivity to analog and digital sensors, but the low-powered AVR microcontroller does not allow audio on-board audio processing. The x-OSC board provides analog and digital I/Os over a wireless link (Madgwick and Mitchell 2013).

Two audio-oriented, self-contained platforms based on a 168MHz Cortex M4 microcontroller hit the market in the past few years: the Owl[1] programmable digital effect (Webster, LeNost, and Klang 2014), which surfaced in

[1] http://hoxtonowl.com/

2013, and Axoloti[2], which came out early 2015. The former can be programmed through a C++ API or can run Pure Data patches using the Heavy Audio Tools from Enzien Audio,[3] while the latter provides a custom graphical patcher which includes DSP modules and can be expanded with C++.

Raspberry Pi is arguably the most popular single-board-computer in the world and its latest revision 3 features a quad-core 1.2GHz 64bit CPU. The CCRMA Satellite distribution[4] (Berdahl and Ju 2011) was developed to provide an efficient audio-oriented environment for the Raspberry Pi.

Coala is an audio processing platform based on the BeagleBone Black[5] which was presented in (Piéchaud 2014). The software and hardware architecture of Coala were developed for the specific task of modal control[6], which requires a very tight feedback loop. The platform is therefore optimized for fast sample-by-sample processing in order to minimize round-trip latency.

Bela: an embedded platform for audio and sensor processing

Bela[7] (formerly known as BeagleRT) is a combined hardware and software environment that consists of a BeagleBone Black with an expansion "cape" (McPherson and Zappi 2015a). It was originally developed for the D-Box Hackable Digital Instrument (Zappi and McPherson 2014) which required multiple low-latency hybrid analog-digital feedback loops (McPherson and Zappi 2015b). Bela combines the connectivity of a microcontroller with the processing capability of a single-board computer. The cape provides stereo audio I/O including 1W speaker amplifiers, 8 channels each of 16-bit analog I/O, and 16 digital GPIO pins. Bela is open-source hardware and software. Source code and design materials are publicly available.[8]

The Bela software uses a Debian Linux distribution with the Xenomai[9] real-time kernel extensions. The Programmable Realtime Unit (PRU), a 200MHz microcontroller on the same chip as the BeagleBone Black CPU, transfers audio and sensor data directly to the hardware, bypassing the kernel drivers. The user's Bela code therefore runs at the highest priority of any task on the board, including the Linux kernel itself. This allows audio block sizes as low as 2 samples, resulting in round-trip audio latency of 1ms (or even down to 100us if using the analog inputs and outputs rather than the audio converters) (McPherson, Jack, and Moro 2016).

On Bela, every analog and digital channel is automatically sampled at audio rates, synchronously with the audio clock. The high sampling rate of the analog and digital channels are unique to Bela and their jitter-free alignment with the audio makes it ideal for interactive, intuitive, responsive audio applications.

Compared to Axoloti and Owl, Bela has more processing power, while still providing hard real-time performances, with the added convenience of a full Linux OS and while being minimally affected by system load. It is more general-purpose than Coala which addresses the specific field of real-time control, though Coala is capable of even lower latencies than Bela. On a Raspberry Pi running CCRMA Satellite, despite the high processing power available on board, audio depends on the standard Linux audio drivers, so that low-latency processing is difficult because of the presence of other processes on the board, which may cause underruns at small audio blocksizes even when the CPU load is low on average. The BeagleBone Black CPU is less powerful overall but the Xenomai extensions used in the Bela software allow reliable and consistent performance with sub-millisecond latency. Additionally, the number of I/Os available in Bela is greater than those on commonly available Raspberry Pi hats.

Bela, providing a large number of I/Os for audio and sensors, power output for loudspeakers, and providing enough processing power to satisfy most needs, entirely fulfills the requirements of a self-contained device, which can be embedded in a stand-alone Digital Musical Instrument or sounding object.

Pure Data on Bela

Pure Data[10] (Pd) is a popular open source graphical programming language widely used by musicians and sound designers alike, which allows for quick prototyping of sound and sensor mappings. Pd patches are usually run

[2] http://www.axoloti.com/ [3] http://enzienaudio.com/ [4] https://ccrma.stanford.edu/~eberdahl/Satellite/
[5] http://beagleboard.org/black [6] http://instrum.ircam.fr/smartinstruments/ [7] http://bela.io [8] http://bela.io/code/
[9] http://xenomai.org/ [10] http://puredata.info

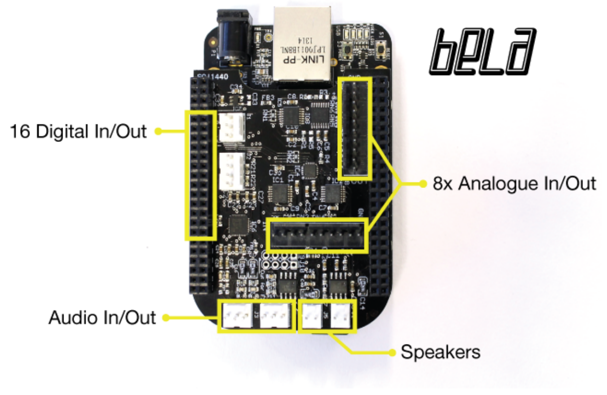


Figure 1: The Bela cape

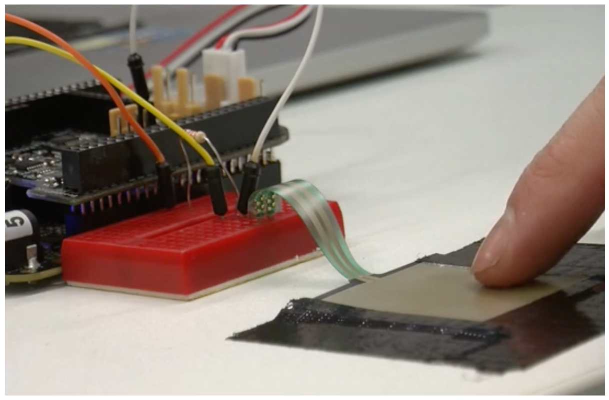


Figure 2: Using a force sensitive resistor with Bela

within Pd itself, or using the shared library *libpd* [11]. The messaging architecture and the audio engine of Pd was not designed to be fast and computationally efficient which can lead performance penalties on platforms with limited computational power.

Heavy Audio Tools The Heavy Audio Tools from Enzien Audio use Pd as a front-end to generate optimised C code. By analyzing the graph of connections between objects in the Pd code, Heavy is capable of producing high-performance vectorized C code which can outperform *libpd*, making it particularly well suited for embedded devices and, more generally, hardware with limited computational power. Heavy is a proprietary, cloud-based service and the generated code is licensed under the MIT non-commercial license.

The C code produced by Heavy is well-suited to be integrated in a Xenomai environment, as memory is allocated on the stack, thus avoiding system calls during execution. An automated script takes care of uploading the Pd patch to Heavy's server, collect the generated C code and compile it on the Bela board. The entire process generally takes less than one minute and most of the time is spent compiling the C code on the BeagleBone Black.

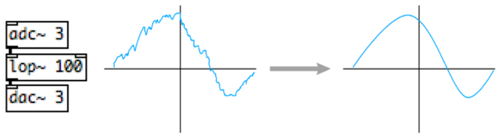
libpd Minimal modifications were required to port *libpd* for Bela, these included allowing blocksizes as small as 8 samples per block and removing socket and disk I/O from the audio thread. Additionally, the calls to the pthread functions were wrapped into Xenomai functions. The resulting shared library can be linked to a Bela program and `libpd_process_float()` is then invoked from within Bela's audio callback.

Deploying a Pd patch using *libpd* is virtually instantaneous as it does not require compiling. As soon as the patch is saved on the BeagleBone's filesystem, the Bela program can be restarted and it will load the updated patch. An added advantage of using *libpd* is that it is easier to port Pd externals when their source code is available. The same precautions listed above should be taken for new externals in order to make sure that new objects do not introduce Xenomai mode switches in the audio thread [12].

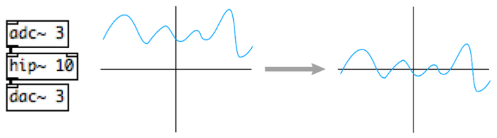
Performance comparison Running an example patch containing a generative audio composition, Heavy code compiled with the *clang*[13] compiler uses 26% of the CPU. The same Heavy code, compiled with *gcc*[14], occupies 43% of the CPU cycles. Running the patch using *libpd* uses 53% of the CPU.

Traditionally, the highest-performance platforms have also placed the most technical demands on the programmer. For many years, custom DSP boards offered the best balance of hard real-time performance and high processing power, but they were generally programmed in low-level languages using custom development environments. High-level music programming languages often come with significant processing overhead. Running Pure Data on Bela,

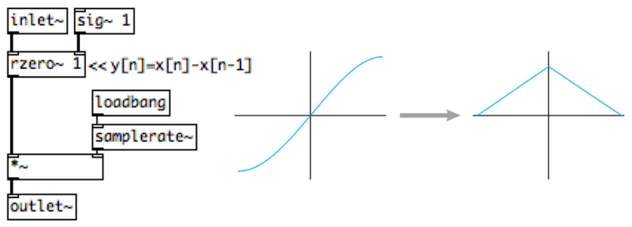
[11] http://libpd.cc/ [12] https://xenomai.org/2014/08/porting-a-linux-application-to-xenomai-dual-kernel/
[13] http://clang.llvm.org/ [14] https://gcc.gnu.org/



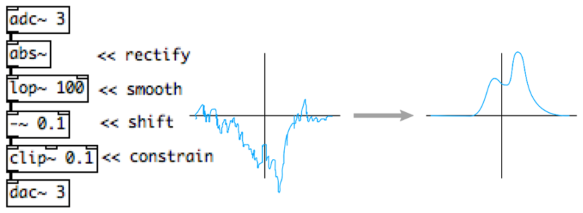
(a) Smoothing



(b) Re-centering



(c) Differentiating



(d) Thresholding

Figure 3: Using Pure Data objects to process sensor data

especially through the Heavy Audio Tools, provides a convenient graphical environment with minimal sacrifice in performance compared to programming in C++.

Sensor processing in PureData

Many interactive systems take approaches where sensors are sampled at low and non-constant rates and the most recent frame of sensor data is used to modulate a particular sonic parameter. But in actual fact, the meaning of sensor data is often deeper, in its behaviour over time or its frequency content. A high sampling rate yields a very high bandwidth of interaction which captures subtle details that might be lost at lower sample rates. Though all the same techniques could be implemented at control rate, audio-rate sensor data can help reorient the designer's thinking to become more aware of these possibilities. When using Bela with Pd, this allows to conveniently process sensor signals using audio-rate objects. Some examples include:

- **Smoothing** Some sensors are inherently noisy, for instance a potentiometer may generate high-frequency noise when it is actuated, or an infra-red optical sensor may be subject to transient perturbations from other emitting sources. The noise in the sensor readings may leak into the audio signal, depending on the signal flow. An easy approach to remove high-frequency noise is to apply a low-pass filter with an appropriate cut-off frequency, as in 3a.
- **Re-centering** Readings from accelerometers and other sensors have inherent DC-offsets which may be undesirable for certain applications. A quick way of removing them which does not require calibration is using an high-pass filter with an appropriate cut-off frequency, as in Figure 3b.
- **Differentiating** Some sound-generator parameters are better controlled using the velocity of a sensor reading, rather than with the raw reading. A high-pass filter with a cut-off frequency of 0, properly rescaled can used for this purpose, as in Figure 3c.
- **Thresholding** A more complicated example in Figure 3d shows how to combine full-wave rectification, smoothing, DC shift and constrain to threshold a signal.

Conclusion

There are several tradeoffs involved in different digital musical instrument design tools: processing power, latency, connectivity, sensor bandwidth, ease of programming and accessibility. With any of the programming environments, Bela brings together the connectivity and CPU power of an embedded Linux computer with the low latency and precise synchronisation of a microcontroller and brings a high-bandwidth dimension to sensor processing. Using either the Heavy or the *libpd* environments, Bela is also suitable for rapid prototyping using the widely-used Pure Data graphical programming language, with full access to both audio and sensors.

Acknowledgements. This work is supported by grants EP/K032046/1, EP/K009559/1 and EP/L019981/1 from the UK Engineering and Physical Sciences Research Council and the Queen Mary Centre for Public Engagement.

References

Berdahl, Edgar, and Wendy Ju. 2011. "Satellite CCRMA: A Musical Interaction and Sound Synthesis Platform." *Proceedings of the International Conference on New Interfaces for Musical Expression*. Oslo, Norway, 173–178.

Madgwick, Sebastian, and Thomas J Mitchell. 2013. "x-OSC: A versatile wireless I/O device for creative/music applications." *SMC Sound and Music Computing Conference*. Stockholm, Sweden: KTH Royal Institute of Technology.

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

McPherson, Andrew, and Victor Zappi. 2015b. "Exposing the Scaffolding of Digital Instruments with Hardware-Software Feedback Loops." *Proceedings of the International Conference on New Interfaces for Musical Expression*. Baton Rouge, Louisiana, USA: Louisiana State University, 162–167.

McPherson, Andrew P., Robert H. Jack, and Giulio Moro. 2016. "Action-Sound Latency: Are Our Tools Fast Enough?" *Proceedings of the International Conference on New Interfaces for Musical Expression*. Brisbane, Australis.

Piéchaud, Robert. 2014. "A Lightweight C++ Real-Time Active Control Framework." *16th Real Time Linux Workshop, October 12 to 13, 2014 at the CCD Congress Center Dusseldorf collocated with LinuxCon Europe in Dusseldorf, Germany*.

Webster, Thomas, Guillaume LeNost, and Martin Klang. 2014. "The OWL programmable stage effects pedal: Revising the concept of the on-stage computer for live music performance." *Proceedings of the International Conference on New Interfaces for Musical Expression*. London, United Kingdom: Goldsmiths, University of London, 621–624.

Zappi, V., and A. McPherson. 2014. "Design and Use of a Hackable Digital Instrument." *Proceedings of the International Conference on Live Interfaces*. Lisbon, Portugal, 208–219.