Using the BeagleBoard as hardware to process sound

RAFAEL VEGA
El Software Ha Muerto
Medellín, Colombia
rvega@elsoftwarehamuerto.org

DANIEL GÓMEZ
Grupo Leonardo, Universidad ICESI
Cali, Colombia
dgomez@icesi.edu.co

Abstract

This paper describes the implementation of diverse layers of code to enable an open source embedded hardware device to run audio processing pure data patches. The evolution of this implementation is reviewed describing different approaches taken by the authors in order to find optimal software and hardware settings. Although some problems are detected when running specific patches, the system has a conjunction of features that are relevant for the floss audio community.

Keywords

Beagleboard, Puredata, sound processing, sound synthesis, opensource.

1 Introduction

Although there is a large number of open-source software tools for working with audio, there are few open-hardware alternatives for musical applications. There is a need to explore the possibilities of open hardware in conjunction with open source software to close the gap between musicians, music producers and the hardware used to create their sounds. It is foreseeable that the closer the sound workers get to the tools they use, the more expressiveness and creativity that can be achieved.

In the Latin American context, there has been an evolution of software design for musical and sonic scenarios along with different MIDI or OSC interfaces attached to laptops [5] [6] [12]. This experimental and DIY tools are designed and used in specific concert or installation situations. Due to their low cost and ease of production we have wondered whether these tools could be a solution for musicians that can barely afford a commercial electronic musical instrument. This project is the first step to explore such platform.

This paper describes the diverse approaches taken to build a stable platform for processing and synthesizing audio signals using an open source, low cost, portable computer such as the BeagleBoard. To achieve this goal, diverse software APIs were explored and the Linux stack, libpd [3] and JACK server were chosen for the current implementation. We present the work so far which is a DSP engine that can parse pd patches and run in a BeagleBoard. Although the project should include physical interfaces that allow for changing parameter values at run time, these features have not been implemented at this early stage.

2 Related work

Some open source tools for musicians exist with varying price ranges, programmability and scope. A common approach has been to develop hardware and firmware and release it as GPL or CC, so that users are able to experiment with the assembly, and possible tweaking of hardware and software. This approach is taken by projects such as Meeblip [7] a diy monophonic synthesizer that works with an atmega chip; Shruthi-1 [8] is another open source (GPL firmware and CC schematics) that runs on a custom mother board. The system that is closer to our approach is Satellite CCRMA [1] where a platform is designed to host PureData under a Linux architecture in the BeagleBoard. Their motivation is to support the creation of new instruments and installations guided by low cost, portability and an an existing community of users such as Pure Data community. Their project is aimed to enhance the longevity of new instruments and to become a possible standard platform for further developments. Although the hardware and software used are the same as in our project, our approach differs on the methodology of the implementation described further.

3 Our Work

PureData was chosen as the language for the description of DSP processes for several reasons: It has been greatly adopted by programmers, sound designers, musicians and tinkerers to implement their audio processing algorithms with many of them using it on-stage (running on a laptop computer). It has an easier learning curve than text-based programming languages such as C++ and it's more suited for rapid implementation of algorithms than compiled languages. Finally, there is a large amount of patches already created by it's user community that can be leveraged by anyone for their creations.

On the hardware side, an inexpensive, portable and powerful computer was needed and having an open design would allow for future enhancements or alterations to the board. Some options such as the PandaBoard were considered but the open-ness, price range and community around the BeagleBoard made it the option of choice. The BeagleBoard project was started by a group of Texas Instruments employees in association with Digi-Key with open-source development in mind and is now supported by a very active community and available from a number of international distributors.

The decision was made to write a C++ program called XookyNabox that would parse a PD patch using an available library and one of the available Linux API's to access the input and output audio buffers. Three libraries were considered: PDAnywhere [4] [9], ZenGarden [11] and libpd [10]. PDAnywhere was discarded right away because of the fact that it uses fixed point arithmetic and the lack of active development around it. After taking a quick look at the implementation and API's for ZenGarden and libpd, ZG was chosen because the readability of the code and it's ease for embedding into C++ (and objective-C) projects. This gave way to the first version of XookyNabox.

The lower level, blocking ALSA API was used to interface with the hardware, along with a number of very simple PD patches. It ran but some synchronization issues were found where the sound output for simple oscillators was rendered at different frequencies than expected. The decision was made then to switch the ALSA API with the higher level and callback based API of PortAudio. This approach solved the synchronization issues but it was required that the program ran as a daemon so that it could be launched at startup in the BeagleBoard without user interaction. This became an issue and the the architecture of the program was changed again.

The next version of XookyNabox still used ZenGarden and was implemented as a JACK client. This time it ran as a daemon without issues but once slightly more complicated patches were used, some of the PD blocks did not run in the BeagleBoard. This, and the fact that many vanilla PD blocks were not implemented in ZenGarden, showed the necessity to use a more robust PD implementation. Enter libpd.

The final version of XookyNabox is implemented as a JACK client and instantiates libpd.

4 The approach that worked

A lightweight, minimalistic Linux distribution that was able to run on the BeagleBoard was needed and Angstrom Linux fit the bill [13]. Also, JACK, ALSA, and the JACK devel libraries are available pre-compiled in the default package repositories for Angstrom.

The Linux system was configured to start automatically at runlevel 3 (without a GUI) and JACK was set up to launch at system startup with a sample rate of 48KHz and a buffer size of 256 samples as suggested by the BeagleBoard user community:

```
jackd -d alsa -p 256 -n 4 -P hw:0 -C hw:0 -S -r 48000
&;
```

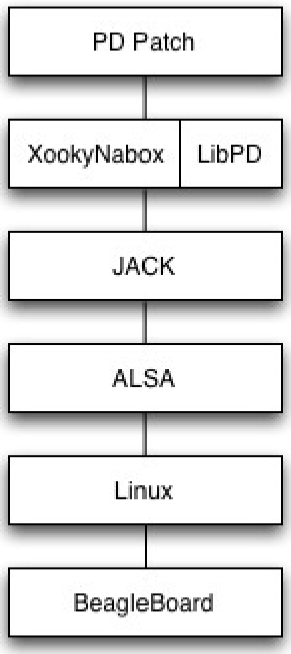

Image 1. Block diagram of the system.

Here's an overview of the implementation of the XookyNabox code. The interesting portion is the process function where mono input buffers from JACK are combined into an interleaved stereo buffer, it is then fed into libpd and the reverse process is applied to libpd's output buffer using a simple technique described in the Audio Programming Book by Boulanger and Lazzarini [2]:

```
//Interleaved io buffers:
float *output =
(float*)malloc(bufferLength*2*sizeof(float)
);
float *input =
(float*)malloc(bufferLength*2*sizeof(float)
);

// ========
// = MAIN =
// ========
int main (int argc, char *argv[]) {
    parseParameters(argc, argv);
    initLibPd();
    initJackAudioIO();

    // Keep the program alive.
    while(1){
        sleep(1);
    }

    return 0;
}

void parseParameters(int argc, char
*argv[]){
    // Show help message, retreive
parameters from argv,
    // make sure the last parameter is
actually a .pd file...
}

// ====================
// = INITIALIZE LIB PD =
// ====================
void initLibPd(){
    // Instantiate libpd, set all relevant
parameters, open the .pd
    //file and pass it to the libd object
}

// ========================
// = INITIALIZE AUDIO I/O =
// ========================
void initJackAudioIO(){
    // Create JACK client, register
callbacks, register
    //io ports and get sample rate from
JACK server.
}

// ======================
// = JACK AUDIO CALLBACK =
// ======================
int process(jack_nframes_t nframes, void
*arg){
    // Get pointers to the input and output
signals
    sample_t *in1 = (sample_t *)
jack_port_get_buffer(portI1, nframes);
    sample_t *in2 = (sample_t *)
jack_port_get_buffer(portI2, nframes);
    sample_t *out1 = (sample_t *)
jack_port_get_buffer(portO1, nframes);
    sample_t *out2 = (sample_t *)
jack_port_get_buffer(portO2, nframes);

    // Jack uses mono ports and pd expects
interleaved
    //stereo buffers.
    for(unsigned int i=0; i<nframes; i++){
        input[i*2] = *in1;
        input[(i*2)+1] = *in2;
        in1++;
        in2++;
    }
    // PD Magic!
    libpd_process_float(input, output);

    for(unsigned int i=0; i<nframes; i++){
        *out1 = output[i*2];
        *out2 = output[(i*2)+1];
        out1++;
        out2++;
    };  return 0;
}
```

It is worth mentioning that an iOS version of XookyNabox was written successfully as a CocoaTouch application that uses libpd and the CoreAudio API's.

5 The patches

The system was tested with basic signal processing and sound synthesis patches constructed exclusively using objects from pd-vanilla. The pd patches were transferred to the SD card that stores the file system for the Linux installation on the Beagle Board. Once the system is powered, the XookyNabox code fetches a file called “patch.pd” and tries to load it. We had successful experiences with some patches, but on the other hand, others did not load at all. The tested patches had no interactivity, they were just single processes that either generated or processed sound in an automatic fashion.

The list of successfully loaded processing patches includes ring, fm and am modulation, filtering and clipping. Successful synthesis patches include additive, subtractive and FM modulation, although envelopes were impossible to create. The problem with controlling the amplitudes over time was a crash of the XookyNabox program that occurred when a patch tried to set the phase of an osc~ object to any specific angle. The same problem occurred when using basic objects for creating envelopes (line, line~, vline~ and delay) so a limit to the testing emerged due to system failures.

The first debugging process was to make a list of objects that, if included, made patches crash, but, at the time of writing this paper, we just got started with checking for the different errors that came form the pdlib when loading the mentioned objects. At this point there is no clarity on the cause of the program crash.

6 Conclussions and future work

At the time of writing the paper, there are still complications in the loading of some pd patches related with specific connections and objects. This complications are below the XookyNabox code and go deeper into libpd in combination with the other JACK, ALSA and Angstrom settings. The actual possibilities of the system are limited and a thorough debugging has to be made.

Paralell to a debugging phase, the development of an interactive electronic bridge to allow communication of electronic sensors (accelerometers, potentiometers, sliders, buttons, etc) to control the patch in real time is a next step in the project.

Although there are some problems to solve, the use of pd patches in a portable, light and relatively low cost computer makes foreseeable a new generation of easily programmable customized audio hardware.

7 Acknowledgements

This paper is the consequence of a long term collaboration within different groups of enthusiasts and academics namely the AudioProgramming group in Medellín, the ACORDE research group, the LEONARDO research group, Julián Brolin Giraldo at the Un-Loquer hackerspace and the invaluable support from Juan Reyes.

References

[1] Edgar Berdahl, and Wendy Ju (2011) “Satellite CCRMA: A Musical Interaction and SoundSynthesis Platform” Proceedings of the New Interfaces for Musical Expression congress 30 May–1 June 2011, Oslo, Norway.

Available online: https://ccrma.stanford.edu/~eberdahl/Papers/NIME2011SatelliteCCRMA.pdf

[2] Richard Boulanger and Victor Lazzarini (2010) “The Audio Programming Book” The MIT Press, 2010.

[3] Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, Hans-Christoph Steiner “Embedding Pure Data with libpd” Pure Data Convention Weimar, Berlin 2011.

[4] Gunter Geiger (2003) PDa: Real Time Signal Processing and Sound Generation on Handheld Devices. Proceedings of the 2003 International Computer Music Conference (San

Francisco), International Computer Music Association, 2003.

[5] Pérez, J., & Jaramillo, J. (2011). MEII: Sistema interactivo para promover la construcción expresiva musical en niños de 4 a 8 años. Revista S&T, 9(17), 55-66. Cali: Universidad Icesi

Web Resources

[6] Juan Reyes, Sonare, https://ccrma.stanford.edu/~juanig/artes/sonare.html

[7] meeblip http://meeblip.noisepages.com

[8] shruthi-1 http://mutable-instruments.net/shruthi1

[9] Gunther Geiger, PDa, http://pd-anywhere.sourceforge.net

[10] Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, Hans-Christoph Steiner, libpd, https://github.com/libpd/libpd

[11] Zen Garden https://github.com/mhroth/ZenGarden

[12] Leonrado Parra, FatChorizo, http://youtu.be/upDvvV0rGuM

[13] Linux Angstrom http://www.angstrom-distribution.org