Designing Hardware Systems: Audio Engineering Case Study

By Daniel Pasternak, ECE '16

All hardware systems are designed for specific applications. Hardware engineers consider the various requirements and constraints of these applications to create systems that is perform better. In audio applications such as the Rust Team's senior design capstone project, system design choices were made to create a real time machine that interfaces with analog and digital devices.

Introduction

If you have ever purchased a computer, you may have compared many different machines based on their specifications. For example, if you were planning to play games, you might purchase a computer with a Graphics Processing Unit (GPU). Alternatively, if you wanted a computer that was lightweight and had long battery life, you might purchase a computer with a special mobile Central Processing Unit (CPU). In both cases, hardware engineers that designed these computers made several system architecture decisions to create machines that are better for their particular uses.

Since computers are everywhere in the music world, there are a huge number of specialized hardware systems, each engineered for a very specific use. For example, computers are present in synthesizers that create sound, audio effects processors which in turn create special effects or correct pitch and in controllers for digital instruments, such as the Yamaha Disklavier (a modern take on the player piano). In each of these systems, hardware engineers made specific architecture decisions, each with tradeoffs, to address specific constraints. This note explores a few of these constraints the Rust Team faced during our senior design project, which focused on a device that captures performances on an acoustic piano and digitizes them.

Background: What Does A Personal Computer Do?

Personal computers are designed to run a wide range of programs efficiently, and to be relatively easy to program. For applications that are user driven, such as word processing or web browsing, these computers are generally effective. However, for applications requiring large amounts of specific types of computation, they are less effective. For example, rendering graphics in games requires a large amount of vector math. Typical CPUs do this very slowly, since their design requires hundreds or thousands of operations to do this. A GPU is designed for this kind of math, and may be able to do it in a fraction of the time. On a range of benchmarks with this type of math, GPUs averaged 2.5x faster than GPUs (Lee et al., 2010).

What's Special About Music Processors?

In many audio processing applications, a device is operating in *real-time*. For example, in the audio effects processor, there is a constant input of audio from a musician's instrument, and a constant output of modified instrument sounds. Since the input is never ending, the processor never gets a break.

This introduces the first major design constraint – The processor must act at least as fast as the stream of input data. If it did not, queuing would occur, and there would be latency, or lag between when the input signal arrives and the output leaves, and the system would no longer be real time.

Several techniques can be used to create a fast enough system. One technique is to use a dedicated device – A microprocessor running only one service (such as the music effects processor) will be faster than one that has to contend with multiple things running, such as a program running on a PC with a Windows OS. The other is to design hardware that can perform the

operation in question very quickly. Since a music processor is doing the same thing repeatedly, it only needs to be able to do that one thing very fast. Hence, special hardware should be selected for that step.

Example – Music Synthesis

The Input to a music synthesizer might be a Musical Instrument Device Interface (MIDI) signal from a device like a keyboard. The MIDI signal will tell the computer what note and volume level to create. So how is the sound actually created?

Since sound is a wave, it can be characterized either by a wave through time, or by its frequency. Musicians do this by referring to each note by its pitch. For example, the middle C on a piano has a fundamental frequency of about 261 Hz. Then, in addition to the fundamentals, there are harmonics: other frequencies that result in the unique sound between different instruments.

Therefore, to synthesize a sound, a whole bunch of different frequencies in various volumes can be selected. Then an operation known as the *inverse Fourier Transform* can be used to transform these frequencies into an audio signal (the time domain) (Burk, n.d.). The FFT requires many multiply and add operations on a long vector of data. This is something that a traditional CPU performs very poorly. Specialized hardware (such as digital signal processors) will be faster in this case.

Senior Capstone Project: The Real Time Acoustic Piano to MIDI Converter

The Rust Team's senior capstone project focused on a device that attaches to a piano frame, and in real time, transcribes any notes played on that piano to a MIDI Stream. The result is that any acoustic piano can be used either as a digital keyboard, or as in an input into other Digital Music systems such as composition software, learning software, or effects software.

In our project, the input to the system is a wave that is generated by magnetic pickups (Figure 1). It is roughly equivalent to the shape of the sound wave that you hear when a note is played. The output is a digital signal – a MIDI stream with information about the notes played and their volumes. Like many other music devices – ours is a real-time device. Ultimately, the two operations our device performs are picking out *note velocity* (loudness) and detecting *note pitch*. Here are the hardware decisions we made for both of these operations.

Figure 1: The Rust Team utilized magnetic pickups to convert vibrations of a piano string to an electrical signal. Shown here is a proof of concept version of our device using a guitar pickup to sense a piano note.

Note Velocity

Note velocity corresponds to how fast a key is pressed. In the case of our system, it is the first derivative of amplitude of the string's vibration. Note velocity tells you how loud the note will be at the moment it is played, similar to how the maximum height of a balls trajectory can be calculated from its initial speed. Since our system measures displacement, in order to get velocity, it is necessary to differentiate the input signal. This can be done either in software, or in hardware. By opting for cheap, specialized hardware, we free up the microprocessor to focus on other work, resulting in a faster system.

Note Pitch

There are several approaches to calculating note pitch. One approach would be the reverse of the sound synthesis process – performing the Fourier Transform and picking out the fundamental frequency. However, this approach is very computationally expensive, especially for the number of notes on a piano. The

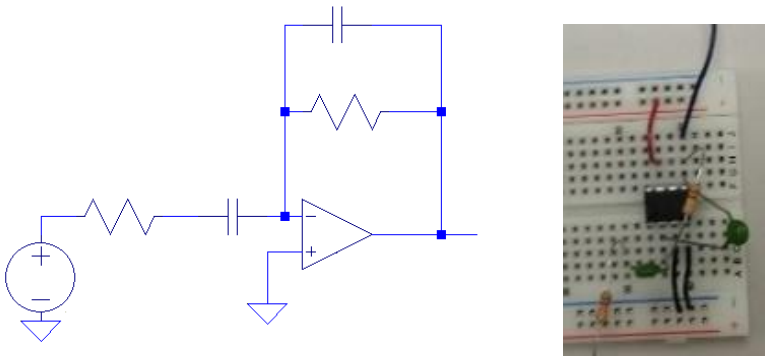

Figure 2: An analog differentiator can be crafted for less than a dollar using a resistor, an op-amp, and a capacitor. At left is a circuit of a simple op-amp differentiator. At right is a physical implementation.

other approach is to create a parallel system – many copies of the system working simultaneously (Figure 3). In order to achieve the speed we needed for our system, we opted for the highly parallel architecture, with one copy of the sensor for each string. The system parallelism follows up to the microprocessor (we used an Arduino). The system consists of several Arduinos each running only our dedicated software – with no other software to slow things down. Each needs only read from its group of sensors, breaking the job into many small parts that can be done simultaneously, reducing latency.

Conclusion

By designing a hardware system properly, it is possible to can obtain a substantial improvement in performance for relatively low cost. Just as adding a GPU to a personal computer frees up the CPU and

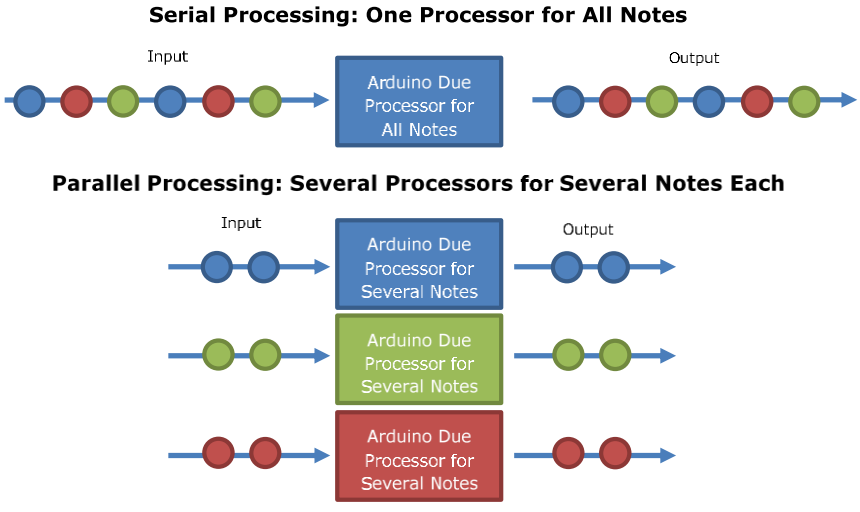

Figure 3: Each piano key is independent from the others, so they can be handled in parallel. Multiple processors are used to process different sets of keys (a parallel system).

improves graphics performance, adding special chips to other machines can increase performance. In our senior design project, the Rust Team architected a system using parallelism, dedicated hardware, and specialized circuits in order to achieve the real-time, very low latency performance required by musicians.

References

Burk, P., Polansky, L., Repetto, D., Roberts, M., & Rockmore, D. (n.d.). *The Synthesis of Sound by Computer*. Retrieved from http://music.columbia.edu/cmc/MusicAndComputers/

Lee, V. W., Hammarlund, P., Singhal, R., Dubey, P., Kim, C., Chhugani, J., Chennupaty, S. (2010). Debunking the 100X GPU vs. CPU myth. *ACM SIGARCH Computer Architecture News - ISCA '10*, 38(3), 451. doi: 10.1145/1816038.1816021