Background

I intend to build a modular, programmable DSP platform, suitable for implementing and experimenting with electric guitar effects. I aim for the system to be as flexible as possible through programmability: rather than building a handful of specific effects in hardware, I want a system that makes it relatively easy to create new effects by writing code in a high-level language, and which allows the user to chain effects together on-the-fly. This project appeals to me because I like music, audio engineering, and programming language theory, among other things. As proof of concept, I will use the system to implement several effects, described below.

Hardware

The core component of this project is the PSoC, as I will rely on its relatively powerful 32-bit ARM processor to handle the calculations involved for DSP. I will not be designing circuits for specific effects, but I will need hardware to get the audio signal into and out of the PSoC. As I want fairly high-resolution audio, I will use an ADC other than the ADC0820, which offers 8 bits. Conveniently, the PSoC offers a Delta Sigma ADC capable of 16-bit sampling at audio rates. Unfortunately, the PSoC 5LP offers only an 8-bit DAC, but I anticipate I can obtain a higher resolution (12 bits) through dithering or combining multiple 8-bit DACs. Outside of the PSoC, I anticipate using op-amps such as the LM358 and LM386 to scale the input and output appropriately.

Beyond audio data, I'll need a way for the user to interact with this system and modify the effects in real-time. For this, I will wire up a bank of generically useful components, such as potentiometers and switches, which DSPs programs can read to shape their behavior. I'll need another ADC to read potentiometer values; 8-bits will suffice. To avoid needing one ADC per potentiometer, I'll use a multiplexer. For switches, I can simply use ports on the PSoC. Additionally, the user should be able to switch effects (programs) easily, which I will enable via the keypad and 74C922 from prior labs.

Special Components

Though not strictly necessary, some nice rotary potentiometers would do wonders for the user interface. Also, a few audio connectors would come in handy: a female 1/4" mono connector (for guitar input), a female 1/8" stereo connector (for headphone output) and, less critically, a female XLR connector (for microphone input).

For scanning through potentiometers, I could use two digital multiplexers (one is included in the lab kit) to control power and ground on each potentiometer, or a single analog multiplexer (such as the 74HC4051, or the PSoC component).

As a stretch goal, I would like to connect the system to a touchscreen display, allowing an additional means for both input and output. For this, the Amulet could work, though a smaller, more responsive display might be preferable, such as a ~3" TFT LCD touchscreen (e.g. the MI0283QT-11, with the ILI934 driver chip). For another stretch goal (portability), I would need a battery and power circuitry suitable for running the PSoC, powering a display, and driving headphones, as well as some kind of enclosure.

Software

As the hardware is intended to be useful for a variety of audio processing effects and tasks, the software is really the crux of the project. My chief concern is building infrastructure that makes it easy to describe small, self-contained audio processing programs or "patches," which can then be chained together in interesting ways. To this end, I will create a system that enables writing these programs in a higher-level language than assembly. I suspect that a general-purpose interpreted language might present performance issues, so more likely option is a (possibly domain-specific) compiled language, which is used to generate efficient machine code to carry out the described signal processing operation. Regardless of the language, all user programs will implement a consistent interface: samples in, samples out. I will program the PSoC in C to support chaining these user programs together, such that the output of one program acts as the input of another. One realization of this would choosing C as the patch language, in which case each user program could be expressed as a C function that modifies an input array, and the signal chain could be represented as a linked list (or tree, for increased flexibility) of function pointers. Other possibilities are discussed in the next section.

The basic operation of the software will be maintaining the current DSP configuration, updating it upon user input, scanning through analog inputs (potentiometers & switches), and of course running the audio through the configured signal chain.

Project Management

My project goals have a few tiers of risk. The core functionality is a system that can take in audio, process it in a user-defined way, and output audio. Thus, the most essential elements are getting audio in to and out of the PSoC at a satisfactory resolution and rate, with relatively low latency (~12 ms at most), and to provide some means of programming the processor. At minimum, the system will be programmable writing new processing functions in C, which can be chained and routed arbitrarily at runtime. Also essential is a rudimentary user interface, which will consist of at minimum a bank of potentiometers and switches and keypad. And a few effects are essential for demonstrating the system's functionality, so I will at minimum create three: a delay (echo), distortion, and boost.

The next level of risk consists of adding additional options for input and output, creating more complex demonstration effects, and allowing higher-level languages for describing user programs. For I/O, I would like to add support for inputs other than an electric guitar, such as a dynamic microphone (with XLR output), 3.5mm jack ('aux in'), and perhaps Bluetooth. Bluetooth would also be a useful output target, assuming I can get the latency sufficiently low. For additional effects, I would like to build a 'looper' that records and replays audio, allowing for overdubbing in real-time; other options include an octave-shifter, reverb, and wah effect. As for higher-level languages, this could take the form of supporting other compiled languages that can support C as a target, such as the audio processing language Faust.

The highest level of risk involves building new subsystems. I would love to get the system working with an LCD touchscreen to make the user interface as flexible as the DSP, and to allow for the possibility of interesting sonic visualizations, but this would require a significant amount of work on top of the core functionality. Additionally, it would be interesting but risky to add support for interpreted languages, such as Python (via MicroPython), Lua (eLua), Javascript (Espruino), or Scheme. It is unclear if these would be performant enough for DSP on the PSoC's

ARM core, but increasing the chunk size (e.g. operating on 512 samples at a time rather than 1) might suffice to compensate at the cost of added latency. An extremely cool but perhaps unrealistic application would be allowing the user to edit DSP programs in real-time via a keyboard.

Timetable

Here I describe my goals for each week starting on the given date.

- 4/15: Successfully get audio into and out of the PSoC, using a function generator as test input.

- 4/22: Hard-code a basic effect and test it using a guitar as input and headphones as output. Wire up potentiometer/switch bank and write code to scan through it.

- 4/29: Write software to support user programs and route audio through them, allowing for customization of the signal chain via keypad.

- 5/6: Write a small collection of effects in C. Investigate possibility of wiring up and controlling a touchscreen display, such as the Amulet or a smaller LCD touchscreen. Investigate possibility of writing effects in higher-level languages that compile to C.

- 5/13: Write more effects, in C or another compiled language. Investigate possibility of running device off of a battery. Investigate possibility of writing effects in higher-level languages that are interpreted on the PSoC.

Figure 1: Hardware block diagram

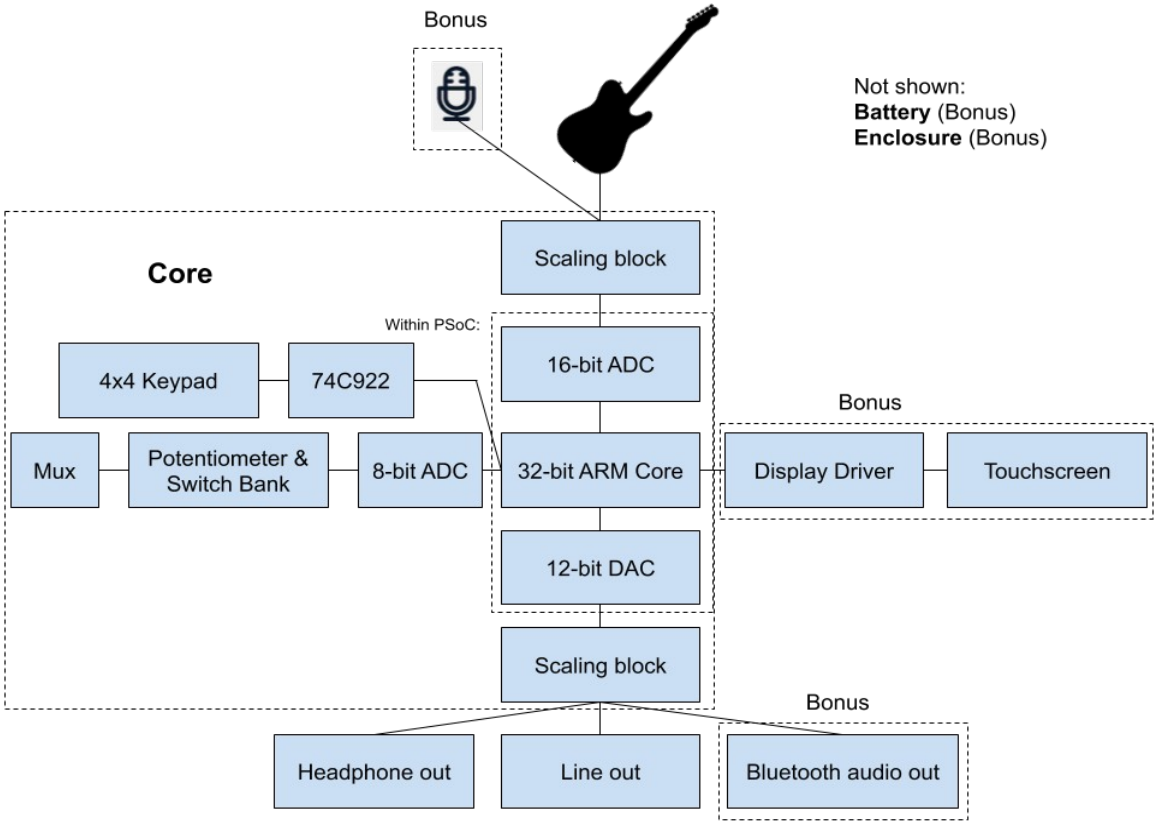

Figure 2: Software overview