Sensor Library for Arduino

David A. Mellis

May 18, 2010

1 Introduction

Sensors convert physical phenomenon into electrical signals that can be processed by computers, allowing for a wide range of interactions between electronic devices and people or the environment. One popular platform for prototyping or implementing devices that can interact in new or unusual ways is Arduino, a combination of open-source hardware and software. Arduino includes a range of development boards based on the 8-bit AVR microcontrollers from Atmel, a set of C/C++ libraries for use in microcontroller programs, and a cross-platform development environment for writing these programs. The most common of these boards, Arduino Duemilanove, includes an ATmega328, a 16 MHz crystal, and a USB-to-serial chip that allows the board to be directly connected to a computer with a USB cable.

Although there are many people using Arduino for reading and processing sensor data, the platform itself is not particularly optimized for the purpose. The API for the analog-to-digital convertor, for example, is limited to a pair of functions which expose little of the ATmega328's capabilities. While it's possible to write low-level code which directly controls the ADC, this requires reading and understanding the microcontroller's datasheet. For someone who just wants slightly more precise control over the acquisition of their sensor data, this is a significant hurdle. It points to the need for a set of APIs which provide better control over the ADC and related microcontroller functionality in an accessible fashion.

This report documents the development of a sensor library for Arduino, which provides capabilities for fast, accurate, low-power and other types of data acquisition and processing. The following section provides an overview of the library's capabilities. The three sections which follow document various use cases enabled by the library and provide preliminary analysis of its performance.

2 Library Overview

The Arduino sensor library provides better control over a number of the microcontroller peripherals as well as higher level functions specific to reading

ADC	Timer 1	Timer 2
begin();	setPrescaleFactor();	setPrescaleFactor()
isRunning();	setMode();	setMode()
setPrescaleFactor();	read();	read()
setMUX();	write();	write()
setReference();	writeCompareA();	writeCompareA()
setAutoTriggerSource();	writeCompareB();	writeCompareB()
autoTrigger();	attachCompareAInterrupt();	attachCompareAInterrupt()
noAutoTrigger();	attachCompareBInterrupt();	attachCompareBInterrupt()
attachInterrupt();	detachCompareAInterrupt();	detachCompareAInterrupt()
detachInterrupt();	detachCompareBInterrupt();	detachCompareBInterrupt()
read();		attachOverflowInterrupt()
		detachOverflowInterrupt()

Table 1: A summary of the sensor library's low-level API.

sensors. In particular, it exposes low-level configuration options for the analog-to-digital convertor and two of the hardware timers. This allows, for example, for specification of the divisor applied to the clocks of these peripherals (relative to the system clock), for the automatic triggering of ADC readings by various conditions, for attaching handlers to different interrupts, etc.

The higher-level commands include a sleep function that puts the microcontroller into a low power state for a given number of milliseconds. The interval is controlled through configuration of a hardware timer using the library's lower-level APIs. There's also a routine for configuring automatic sampling of the ADC at an interval specified in hertz. This, too, uses the low-level APIs to configure a timer. Finally, there's a function which uses the fixed 1.1V ADC source to determine the supply voltage to the microcontroller. These routines provide high-level functionality to the user and demonstrate the power and flexibility of the lower-level APIs which enable them. There are many other specific behaviors which can be generated using the hardware configurations allowed by the library which would otherwise be accessible only by reading the datasheet.

3 Use-Case: Fast Sampling

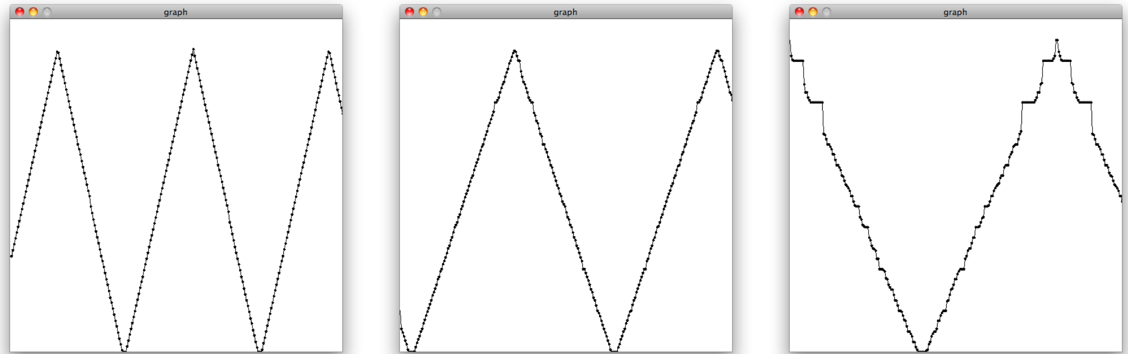
By default, the Arduino core libraries use the slowest available clock speed for the ADC: 1/128 of the system clock (125 KHz for the standard system clock of 16 MHz). This provides accurate and precise samples but a maximum sample rate of 10 KHz (one sample takes 13 ADC clock cycles). By allowing for the adjustment of the ADC clock prescale factor (its divisor relative to the system clock), the Arduino sensor library lets the user select the appropriate balance of sample rate and precision for their application. The prescale factors available are 128, 64, 32, 16, 8, 4, and 2, providing sample rates of approximately 10 KHz, 19 KHz, 38 KHz, 77 KHz, 153 KHz, 308 KHz, and 615 KHz, respectively.

The following code provides an example of the use of the library to set the

ADC prescale factor, take a number of samples, and record the time elapsed:

```
unsigned long start = millis(), end;
ADC.setPrescaleFactor(ADC.PRESCALE16);
for (int i = 0; i < NUM; i++) samples[i] = analogRead(0);
end = millis();
```

Here are graphs of data collected using this code (the follow program listing appears in the appendices). They show 512 10-bit samples from a 5V, 1 KHz triangle wave using ADC prescale factors of 8, 4, and 2, respectively:



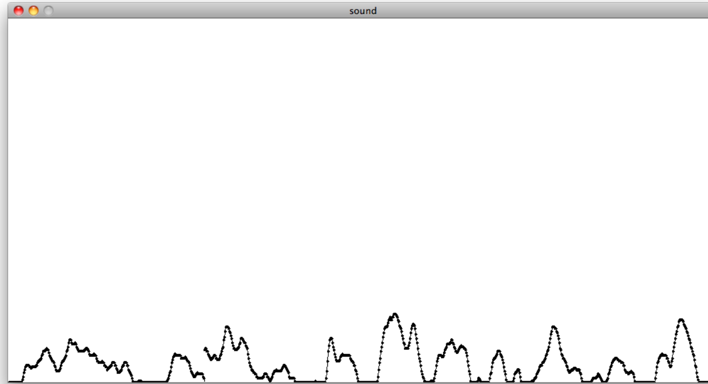
You can see some inaccuracy in the last graph, but the others are relatively clean. This is surprising considering that all of them are well outside the range recommended for precision sampling by the microcontroller datasheet.

4 Use-Case: Precise Sample Rate

In some cases, it's desireable to precisely control the sample rate of the ADC. For example, audio is frequency sampled at 44.1 KHz. The sensor library provides a function to automatically sample the ADC at a specified rate. For example, the function call:

```
autoSample(0, 44100, sample);
```

configures the library to invoke the `sample()` function at 44.1 KHz with the ADC reading from channel zero. Here's a visualization of sensors collected from an electret microphone using this functionality.



5 Use-Case: Low-Power

To support applications which require periodic sampling and low power consumption, the sensor library includes a sleep function. This function places the ATmega328 into extended standby mode for a specified duration (which can range from 1 millisecond to a few seconds). An internal timer interrupt triggers the wake from sleep.

The library includes an example which illustrates the use of the sleep function and was used to test its effect on the microcontroller's power consumption. It takes an ADC reading every second and sends it over the serial port. The following excerpt shows the simplicity of the code required:

```
void loop()
{
  sleep(1000);
  Serial.println(analogRead(0));
}
```

Power consumption was tested with both the standard Arduino delay function and the new sleep function supplying the one-second intervals. With a standard delay, the ATmega328 itself drew approximately 18 mA of current (running at 5V on an 8 MHz external resonator). With a sleep instead of a delay, this decreased to about 1 mA. On the Arduino board (with a regulated 5V power supply and an external 16 MHz crystal), the delay program drew about 48 mA. The sleep program drew about 18 mA.

6 Conclusions and Next Steps

The use cases described above provide some initial evidence of the utility of the sensor library. In partiular, its low-level APIs seem to do a good job of exposing

the power and flexibility of the capabilities of the microcontroller. The higher-level functions offer a few initial examples of the types of applications that the library enables. The ATmega328 has been designed to provide for a number of common uses and the sensor library makes them accessible to a wider audience.

The next steps for the library are its publication and wider dissemination. This include documenting the API and explaining the behavior of the microcontroller peripherals it exposes. It could also include creating more complex or compelling demostrations of the interface possibilities offered by more capable sensor processing.

A Audio Circuit

Electret microphone circuit used for audio sampling: