Smart Data Collection and Processing on Arduino

Justin A. Shetty[1], Anna C. Gilbert[1,2]

1. University of Michigan, Ann Arbor 2. Herman H. Goldstine Collegiate Professor, Department of Mathematics

Abstract

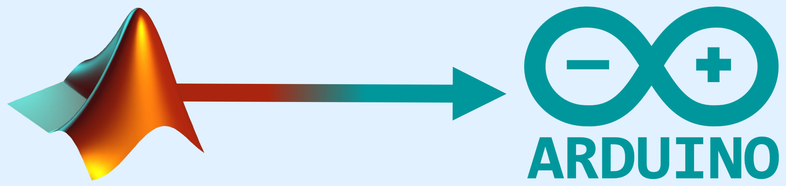
Currently there is a lack of fast, accurate, memory-efficient data collection and storage methods on the Arduino platform. The goal of this research is to implement an existing sublinearly-efficient MATLAB data sampling algorithm developed by Gilbert, Strauss, and Tropp on the Arduino platform. Preliminary work involved developing an understanding of the existing MATLAB algorithm while subsequent efforts involved the implementation of the systems' components on the Arduino. Testing included comparing the detected frequency-coefficient pairs produced by the Arduino implementation with those of the MATLAB implementation. Results revealed the extent to which this sublinear data collection/storage algorithm retains its effectiveness on the Arduino. Further work can include implementation of the algorithm in other languages.

Figure 1: RAM capacity of a typical x86 system versus that of an Arduino Mega

Introduction

The Cooley-Tukey Fast Fourier Transform (FFT) algorithm provides users with a means of characterizing signals. However, on low-resource platforms, even the highly-efficient FFT, which is $O(n\,logn)$, still proves to be a computational and memory bottleneck. This obstacle is especially pertinent to Arduino microcontrollers, where computational power and RAM are exceptionally limited. In response to the need for more efficient means of signal characterization, Iwer, Gilbert, and Strauss developed the AAFFT [1], an algorithm that has been shown to be sublinearly-efficient in characterizing time-sparse signal samples and faster than the FFTW 3.1 for sample sets larger than 10^6 [1]. However, this implementation is limited to its MATLAB environment, which necessitates full x86 architecture system. To facilitate usage of such an algorithm in possible Internet of Things applications, this research pursues the implementation of the AAFFT algorithm in C++ for use on Atmel microcontroller based devices, e.g. Arduinos. RAM capacity differences are evident in Figure 1.

[1] A. Gilbert, M. Iwen, and M. Strauss, "Empirical evaluation of a sub-linear time sparse DFT algorithm", *Communications in Mathematical Sciences*, vol. 5, no. 4, pp. 981-998, 2007.



Methods

A divide-and-conquer, bottom-up approach was taken to handle the "translation" of the system. As opposed to the MATLAB implementation, the Arduino system was structured as a traditional library with all functions in a single file for ease of use. The "lowest level" parts of the algorithm, i.e. functions that do not rely on many or any other functions from the system, were written first so that rudimentary unit testing could be done to ensure some level of basic functionality. Then, the "higher level" functions, those which utilize other AAFFT-specific functions, could be implemented.

During development, concern arose as to whether the Arduino's built-in random number generator was random enough for a proper implementation of the AAFFT, which relies on randomly selected sampling points in time. To ensure proper functionality, the "randomness" of the Arduino's random number generation was found to be satisfactory based on the data visible in Figure 2.

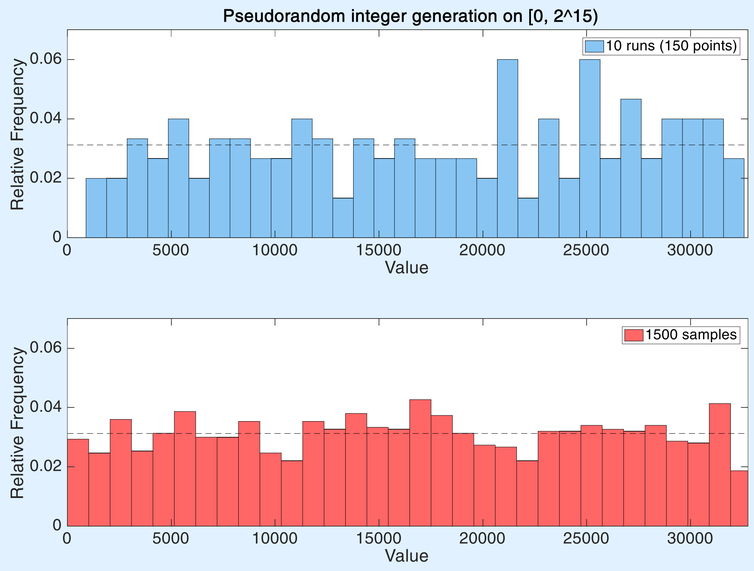


Figure 2: Distribution of pseudorandom integer samples. *Top:* 150 samples, *Bottom:* 1500 samples

Results

Despite the Arduino Mega's substantially lower RAM capacity, which limited the types of data structures that could be used, the Arduino implementation of the AAFFT can now successfully identify the frequencies and coefficients of software-generated signals. Unfortunately, the RAM limitation does limit number of "passes" that can be taken on the signal, meaning that as the number of tones in a signal there are, the less likely the algorithm is to accurately or precisely detect the tones and their corresponding coefficients.

The complete and latest Arduino implementation can be found on GitHub at the following address:

https://github.com/justinshetty/**AAFFT-Arduino**

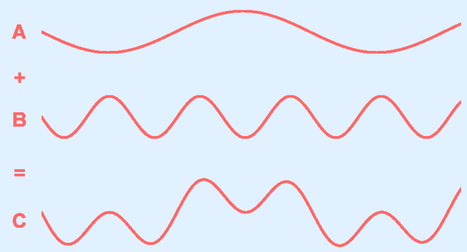


Figure 3: Illustration of a complex signal represented in time as a sum of single-tone sine waves

Graphic Source: http://www.bbc.co.uk/blogs/researchanddevelopment/2010/11/mozilla-audio-data-api.shtml

Conclusions

Implementation of the AAFFT on the Arduino brings the computational and memory efficiency of the MATLAB counterpart to the world of small-scale computing and internet integration. Now with a way to efficiently collect, store, and in turn transmit data, low bandwidth internet connections and devices with little computational and storage resources can be used in applications involving heavy data collection.

Future work could involve development of a Python implementation for use on devices like the Raspberry Pi that are neither based on the Arduino nor are capable of running heavy software like MATLAB. A Python implementation would also be useful for users without access to a MATLAB license.