Real time digital audio processing with Arduino

André J. Bianchi  Marcelo Queiroz
ajb@ime.usp.br    mqz@ime.usp.br

Departament of Computer Science
Institute of Mathematics and Statistics
University of São Paulo

SMC 2013, July 30th - August 3rd
Real time digital signal processing

Digital audio signal processing includes:

▶ Acquiring samples.
▶ Processing.
▶ Outputting results.

Real time restriction:

▶ Block processing: $N$ samples.
▶ Sampling frequency: $R$ Hz.
▶ DSP cycle period: $T_{DSP} = \frac{N}{R}$ s.
Real time DSP with Arduino

http://interface.khm.de/index.php/lab/experiments/arduino-realtime-audio-processing/
Atmel AVR microcontroller (ATmega328P)

Microcontroller’s characteristics:

- CPU: ALU and registers (16 MHz - 8 bits).
- Memory: Flash (32 KB), SRAM (2 KB) e EEPROM (1 KB).
- Digital I/O ports:
  - Audio input: analog to digital converter.
  - Audio output: counters capable of doing PWM.
Arduino performance for real time digital audio processing

Questions:

- What is the maximum number of operations feasible in real-time?
- Which implementation details make a difference?
- What is the quality of the resulting audio signal?

DSP algorithms implemented:

- Additive synthesis.
- Time-domain convolution.
- FFT.
Audio input: analog to digital converter

Arduino ADC maximum conversion frequencies

- Advertised ADC frequency
- Measured ADC frequency

Graph showing the relationship between prescaler value and frequency (KHz). The graph includes two lines, one for advertised ADC frequency and another for measured ADC frequency.
Pulse Width Modulation

original signal → PWM output → counter overflow

1
1
0
1
0
1
1
0
Audio output: Pulse Width Modulation

8-bit counter frequencies for different prescaler values:

<table>
<thead>
<tr>
<th>prescaler</th>
<th>( f_{\text{incr}} ) (KHz)</th>
<th>( f_{\text{overflow}} ) (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16000</td>
<td>62500</td>
</tr>
<tr>
<td>8</td>
<td>2000</td>
<td>7812</td>
</tr>
<tr>
<td>32</td>
<td>500</td>
<td>1953</td>
</tr>
<tr>
<td>64</td>
<td>250</td>
<td>976</td>
</tr>
<tr>
<td>128</td>
<td>125</td>
<td>488</td>
</tr>
<tr>
<td>256</td>
<td>62.5</td>
<td>244</td>
</tr>
<tr>
<td>1024</td>
<td>15.625</td>
<td>61</td>
</tr>
</tbody>
</table>

PWM overflow interrupt allow for periodically triggering:

- ADC conversion.
- Signal manipulation.
- PWM mechanism value set.
Additive synthesis

\[ f_1 r_1, f_2 r_2, f_3 r_3, \ldots, f_K r_K \]

\[ y(n) \]

Additive Synthesis on Arduino (loop)

Additive Synthesis on Arduino (inline)
Additive synthesis

Example

Sum of harmonics with $f_0=200$ Hz:

$$y[n] = \sum_{k=1}^{\infty} \sin \left( 2\pi k 200 \frac{n}{R} \right).$$
Time-domain convolution

\[ x[n] \xrightarrow{z^{-1}} \sum b_0 \xrightarrow{z^{-1}} \sum b_i \xrightarrow{z^{-1}} \sum b_N \rightarrow y[n] \]

---

**Time-domain convolution on Arduino (mult/div)**

- bl. size 32
- rt per. 32
- bl. size 64
- rt per. 64
- bl. size 128
- rt per. 128
- bl. size 256
- rt per. 256

**Time-domain convolution on Arduino (bit-shifting)**

- bl. size 32
- rt per. 32
- bl. size 64
- rt per. 64
- bl. size 128
- rt per. 128
- bl. size 256
- rt per. 256
Time-domain convolution

Example: moving average

Moving Average Frequency Response

- order 2
- order 4
- order 8
- order 16
Fast Fourier Transform

A signal in the time domain can be converted to its counterpart in the frequency domain by means of Fourier Transform (FT). The signal must be sampled at discrete time by an A/D converter before it can be analyzed by a computer. Discrete Fourier Transform (DFT) can be used to convert the discrete signal (discrete in time) in the time domain to its counterpart (discrete in frequency) in the frequency domain. DFT can be computed efficiently in practice using a Fast Fourier Transform (FFT) algorithm, which is generally $N/\log(N)-1$ times faster than DFT, where $N$ is called DFT or FFT size, which is the number of data points used in the computation. To achieve maximum efficiency of computation in FFT, $N$ is generally constrained to an integer power of two, e.g. 1024, 2048, 4096, 8192, etc..

![Diagram](https://example.com/diagram.png)

1) continuous signal in time domain
2) N points in time domain
3) N points in frequency domain containing both negative and positive frequency parts
4) N/2+1 points in amplitude/power spectrum

The above figure illustrates the aforementioned process. An N-point time record $[x(t_0), x(t_1), \ldots, x(t_{N-1})]$ will generate N points $[X(-f_{N/2}), \ldots, X(f_0), \ldots, X(f_{N/2})]$ in the frequency domain containing both negative and positive frequency parts. The positive and negative frequency parts can be combined to produce $N/2+1$ points $[X(f_0), X(f_1), \ldots, X(f_{N/2})]$ at real frequencies in the amplitude/power spectrum. These points are located at frequencies: $0, \Delta f \times 1, \ldots, \Delta f \times N/2$, where $\Delta f = f_s/N$, where $f_s$ is the sampling frequency. The highest frequency measurable is $f_s/2$ and is called Nyquist frequency.

An important principle in digital signal processing is the "Nyquist-Shannon Sampling Theorem" which states that an analog signal that has been sampled can be perfectly reconstructed from the samples if the sampling frequency is greater than twice the highest frequency in the original signal.

There are three possible issues inherent in DFT or FFT, which may result in errors if no proper precautions are taken. They are aliasing, leakage, and picket fence effect.

2. Aliasing

Aliasing occurs when a signal is sampled at less than twice of the highest frequency present in the signal. It causes frequency components that are higher than half of the sampling frequency to be incorrectly represented as lower frequency components. This can lead to incorrect interpretations of the signal's frequency content.

www.virtins.com                    D1002                                       3                              Copyright © 2009 Virtins Technology
Fast Fourier Transform

FFT on Arduino (at 1953 Hz)

Maximum frequency for block size 256:

- Mean calculation time \( \approx 428.15 \, \mu s \) per sample.
- Maximum frequency \( \approx 2.335 \, Hz \).
- PWM prescaler value 32 \( \Rightarrow R = 1.953 \, Hz \).
Conclusions

- Many implementation details make a difference:
  - Types used (byte, unsigned long, int, float, etc).
  - Type of operations: integer (multiplication, division, sum) and bitwise.
  - Presence of loops.
  - Use of variables and vectors.

- Families of algorithms can be found to make it feasible to use the Arduino in real time audio processing.
Thank you for your attention!

Contact:

- Email: \{ajb, mqz\}@ime.usp.br
- This presentation: http://www.ime.usp.br/~ajb/
- CM at IME: http://compmus.ime.usp.br/

Attribution of figures taken from wikipedia:

- PWM: Zurecs (zureks@gmail.com).
- Additive synthesis: Chrisjonson.
- FFT: Virens.