13

Цифровая обработка сигналов

Плата Arduino способна выполнять простую обработку сигналов. В этой главе обсуждаются разные способы такой обработки, от фильтрации сигнала, поступающего на аналоговый вход, с применением программного обеспечения вместо внешних электронных устройств до вычисления относительной величины различных частотных сигналов с применением быстрого преобразования Фурье.

Введение в цифровую обработку сигналов

Принимая информацию с датчика, вы фактически измеряете сигнал. Сигналы принято отображать в виде линии (обычно извилистой), идущей слева направо с течением времени. Именно так отображаются электрические сигналы на экране осциллографа. Ось *y* отражает *амплитуду* сигнала (его силу), а ось *x* — время. На рис. 13.1 изображен сигнал, соответствующий воспроизведению музыкального фрагмента длительностью 1/4 секунды, который был захвачен с помощью осциллографа.

Вы можете заметить некоторую цикличность сигнала. Скорость повторения циклов называют *частотой*. Она измеряется в герцах (Гц). Сигнал с частотой 1 Гц повторяет цикл со скоростью 1 раз в секунду, с частотой 10 Гц — 10 раз в секунду. Взгляните на сигнал в левой половине рис. 13.1 — один цикл сигнала длится примерно 0,6 квадрата координатной

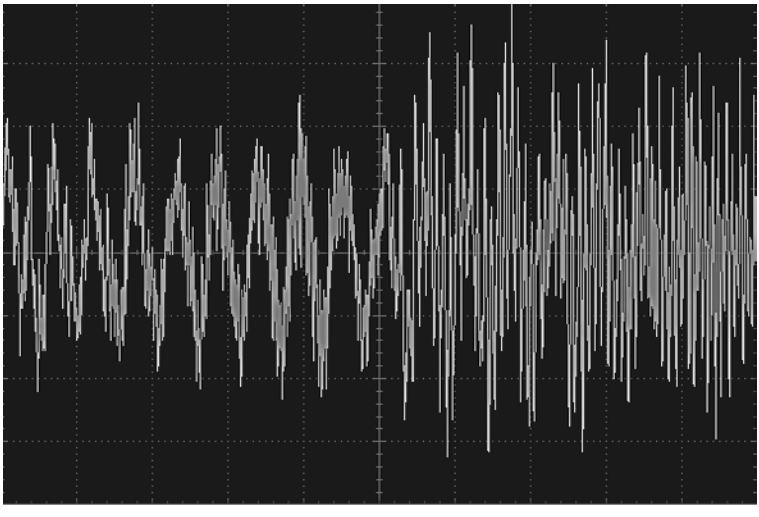

Рис. 13.1. Сигнал от источника музыки

сетки. Так как при настройке осциллографа размер стороны квадрата по оси *x* был выбран равным 25 мс, частота этой части сигнала составляет $1/(0{,}6 \times 0{,}025) = 67$ Гц. Если увеличить масштаб, выбрав более короткий промежуток времени, можно будет увидеть множество других частотных составляющих звука, подмешивающихся к основному сигналу. Если сигнал не является чистой синусоидой (как показано далее на рис. 13.5), он всегда будет включать в себя целый спектр гармоник.

Сигнал, изображенный на рис. 13.1, можно попытаться захватить с использованием одного из аналоговых входов на плате Arduino. Этот прием называется *оцифровкой*, потому что в ходе его реализации аналоговый сигнал переводится в цифровую форму. Чтобы получить довольно точную копию оригинального сигнала, замеры нужно выполнять очень быстро.

Суть цифровой обработки сигналов (Digital Signal Processing, DSP) заключается в том, чтобы оцифровать сигнал с помощью аналого-цифрового преобразователя (АЦП), выполнить некоторые манипуляции с ним и затем сгенерировать выходной аналоговый сигнал с помощью цифроаналогового преобразователя (ЦАП). Самое современное звуковое оборудование, MP3-плееры и сотовые телефоны используют цифровую обработку сигналов для частотной коррекции, управляя относительной мощностью высоких и низких частот при воспроизведении музыкальных произведений. Однако иногда не требуется выводить измененную версию входного сигнала, когда

цифровая обработка нужна, только чтобы убрать из сигнала нежелательные помехи и тем самым получить от датчика более точное значение.

В общем случае платы Arduino — не самые лучшие устройства для цифровой обработки сигналов. Они не способны захватывать аналоговый ввод с высокой скоростью, а их аналоговые выходы ограничены возможностями технологии широтно-импульсной модуляции (ШИМ). Исключение составляет модель Arduino Due, которая имеет несколько АЦП, быстрый процессор и два истинных ЦАП. То есть модель Due обладает достаточными аппаратными возможностями для того, чтобы ее можно было использовать для оцифровки звукового стереосигнала и выполнения каких-то манипуляций с ним.

Усреднение замеров

При чтении сигнала с датчика часто обнаруживается, что лучших результатов можно добиться, выполняя усреднение по нескольким замерам. Одно из возможных решений этой задачи — использование циклического буфера (рис. 13.2).

При использовании циклического буфера каждый новый замер сохраняется в ячейке буфера с текущим индексом, после чего индекс увеличивается на единицу. Когда будет заполнена последняя ячейка, значение индекса обнулится, и новые замеры начнут записываться поверх старых. Следуя

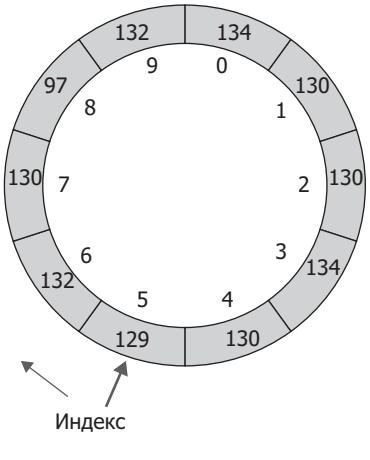

Рис. 13.2. Циклический буфер

этой методике, вы всегда будете иметь N последних замеров, где N — это размер буфера.

Следующий фрагмент реализует циклический буфер:

```
// sketch_13_01_averaging

const int samplePin = A1;

const int bufferSize = 10;
int buffer[bufferSize];
int index;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int reading = analogRead(samplePin);
  addReading(reading);
  Serial.println(average());
  delay(1000);
}

void addReading(int reading)
{
  buffer[index] = reading;
  index++;
  if (index >= bufferSize) index = 0;
}

int average()
{
  long sum = 0;
  for (int i = 0; i < bufferSize; i++)
  {
    sum += buffer[i];
  }
  return (int)(sum / bufferSize);
}
```

Это решение дает неверный усредненный результат, пока буфер не заполнится. На практике это не должно быть большой проблемой, так как

не составит труда заполнить буфер замерами перед тем, как начать запрашивать усредненные замеры.

Обратите внимание на то, что переменная `sum` для хранения суммы в функции `average` объявлена с типом `long`. Это особенно важно, если используется емкий буфер и есть вероятность, что сумма превысит максимальное положительное значение `int`, которое немногим больше 32 000. Отметьте также, что она безопасно может возвращать результат усреднения в виде значения `int`, потому что среднее значение будет находиться в диапазоне значений отдельных замеров.

Введение в фильтрацию

Как говорилось в разделе «Введение в цифровую обработку сигналов», любой сигнал обычно состоит из целого спектра гармоник. Иногда бывает желательно исключить некоторые из этих гармоник, и тогда следует использовать прием фильтрации.

Наиболее распространенный способ фильтрации в Arduino — *низкочастотная фильтрация*. Представьте, что у вас имеется датчик освещенности и вы пытаетесь определить общий уровень освещенности и поминутную динамику ее изменения, например, чтобы определить момент, когда станет достаточно темно, чтобы включить освещение. Но вам нужно устранить высокочастотные изменения освещенности, вызванные такими событиями, как быстрое перемещение вблизи датчика объектов, заслоняющих свет, или засветка датчика искусственными источниками света, которые в действительности мерцают с частотой напряжения питания (50 Гц, если вы живете в России). Если вас интересует только медленно изменяющаяся часть сигнала, то вам нужен низкочастотный фильтр. И наоборот, если требуется откликаться на скоротечные события и игнорировать более протяженные тенденции, используйте *высокочастотный фильтр*.

Вернемся к проблеме искажений, вызываемых частотой переменного тока в электросети. Если, к примеру, вам нужно исключить только паразитную гармонику с частотой 50 Гц и оставить гармоники с частотами выше и ниже этого значения, тогда простое отсечение низкочастотных гармоник не даст желаемого результата. Для решения этой задачи следует использовать *полосовой фильтр*, который удалит только гармонику с частотой 50 Гц или, что более вероятно, все гармоники с частотами от 49 до 51 Гц.

Простой низкочастотный фильтр

Часто в циклическом буфере нет никакой необходимости, если требуется всего лишь сгладить сигнал. Такое сглаживание можно рассматривать как низкочастотную фильтрацию, которая отсекает высокочастотные составляющие сигнала и оставляет только общую динамику. Подобного рода фильтрация часто используется при работе, например, с датчиками поворота, чувствительными к высокочастотным изменениям, которые могут не интересовать вас, или когда вам достаточно знать, на какой угол повернуто устройство.

Простой и эффективный способ решения этой задачи заключается в сохранении некоторого скользящего среднего по нескольким замерам. Скользящее среднее вычисляется как пропорция между текущим скользящим средним значением и значением нового замера:

$$\text{Сглаженное значение}_n = (\text{Коэффициент} \times \text{Сглаженное значение}_n - 1) + \\ + ((1 - \text{Коэффициент}) \times \text{Замер}_n).$$

Коэффициент — это константа между 0 и 1. Чем выше значение коэффициента, тем сильнее эффект сглаживания.

Такое определение выглядит сложнее, чем есть на самом деле, поэтому взгляните на следующий код, чтобы убедиться, насколько этот прием прост в реализации:

```
// sketch_13_02_simple_smoothing
const int samplePin = A1;
const float alpha = 0.9;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  static float smoothedValue = 0.0;
  float newReading = (float)analogRead(samplePin);
  smoothedValue = (alpha * smoothedValue) +
    ((1 – alpha) * newReading);
  Serial.print(newReading); Serial.print(",");
  Serial.println(smoothedValue);
  delay(1000);
}
```

Скопировав результаты сглаживания из окна монитора последовательного порта и вставив их в электронную таблицу, можно построить график, чтобы увидеть, насколько хорошо выполняется сглаживание. На рис. 13.3 показан результат работы предыдущего скетча в плате, к аналоговому входу A1 которой подключен некоторый источник переменного сигнала.

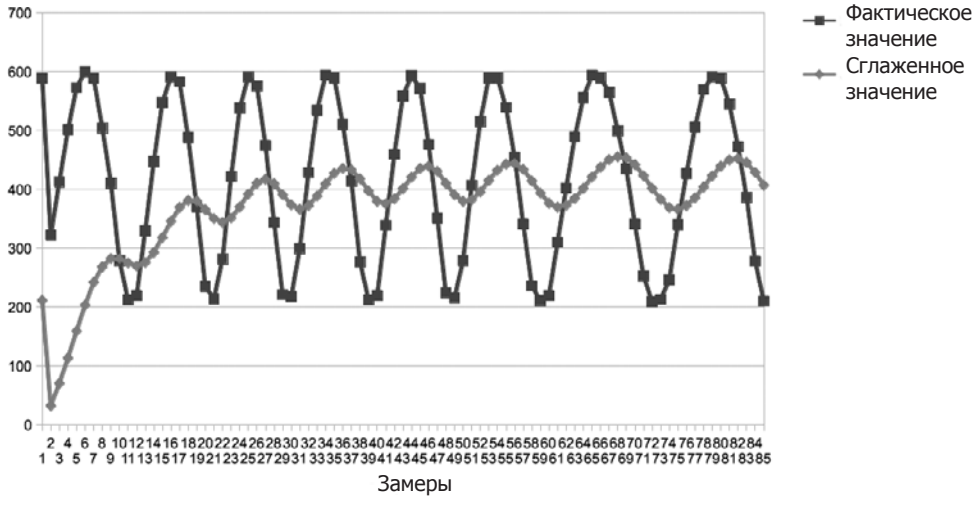

Рис. 13.3. График изменения сглаженных значений

Как видите, для выхода на нормальный уровень сглаживания требуется некоторое время. Если увеличить значение *коэффициента*, например, до 0,95, сглаживание получится еще более сильным. Построение графиков на основе данных, скопированных из окна монитора последовательного порта, — отличный способ проверить, насколько результат сглаживания соответствует вашим потребностям.

Цифровая обработка сигналов в Arduino Uno

На рис. 13.4 изображена схема подключения источника сигнала звуковой частоты к контакту A0 на плате Arduino и приемника выходного ШИМ-сигнала (10 кГц), генерируемого платой. В качестве генератора сигнала я использовал приложение на смартфоне и подключил выход для наушников на телефоне к плате Arduino.