Лабораторная работа №6

Использование АЦП в среде MBed

Теоретическая часть

Микроконтроллер общается с внешним миром посредством портов ввода/вывода. В общем случае он может “воспринимать” только цифровые сигналы – логический ноль или логическую единицу. Например, для микроконтроллера при напряжении питания 3,3 В логический ноль – это напряжение от 0 до 1,3 В, а логическая единица – от 1,8 до 3,3 В. Довольно часто возникает потребность измерять напряжения, которые могут принимать любое значение в диапазоне от 0 до напряжения питания. Для этих целей в составе микроконтроллеров есть аналого-цифровой преобразователь (АЦП).

Не вдаваясь в подробности внутреннего устройства АЦП, представим его в виде черного ящика. На вход АЦП подается непрерывный аналоговый сигнал, а на выходе получается последовательность цифровых значений. АЦП имеет много характеристик, но в качестве основных можно назвать разрешающую способность, абсолютную точность, предельную частоту дискретизации и диапазон входных напряжений.

Разрешающая способность (разрешение) – это способность АЦП различать два значения входного сигнала. Определяется как величина обратная максимальному числу кодовых комбинаций на выходе АЦП. У STM32 АЦП 12-ти разрядный. Максимальное число кодовых комбинаций будет равно 2^{12} = 4096. Разрешающая способность равна 1/4096 от всей шкалы допустимых входных напряжений.

Для работы АЦП необходим источник опорного напряжения (ИОН). Для него это эталон, по отношению к которому он измеряет входные сигналы.

Напряжение питания в нашей схеме 3,3 В, тогда 1/4096 от всей шкалы это 3,3 * 1/4096 = 0,0008 В или примерно 1 мВ. С таким шагом (это называется шаг квантования) АЦП будет измерять входное напряжение. Если два ближайших значения сигнала на входе АЦП будут отличаться между собой на величину < 1 мВ, АЦП воспримет их как одинаковые. На практике разрешающая способность АЦП ограничена его шумами.

Абсолютная точность – отклонение реального преобразования от идеального. Это составной результат нескольких погрешностей АЦП. Выражается в количестве младших значащих разрядов (LSB - least significant bit) АЦП. Для STM32 абсолютная погрешность АЦП = ±2LSB. Для нашего примера абсолютная точность будет равна 2^2 * 0.8 мВ = 3,2 мВ.

Предельная частота дискретизации определяет быстродействие АЦП и измеряется в герцах или количестве выборок в секунду (SPS – samples per second).

Теорема Котельникова (теорема Найквиста-Шеннона, теорема о выборке) гласит, что аналоговый сигнал имеющий ограниченный спектр, может быть восстановлен однозначно и без потерь по своим дискретным отсчётам, если частота выборки (дискретизации) превышает максимальную частоту спектра сигнала более чем в 2 раза. Выражаясь по-простому - если вам нужно оцифровать аналоговый сигнал с полосой спектра 0 - 7 КГц, то в идеальном случае частота дискретизации должна быть > удвоенной максимальной частоты спектра этого сигнала, то есть > 14 КГц. На практике все намного сложнее. Перед входом АЦП всегда ставят НЧ фильтр, чтобы ограничить спектр сигнала, а частота дискретизации выбирается еще более высокой.

Диапазон входных напряжений – это минимальное и максимальное значение напряжения, которое можно подавать на вход АЦП. Для нашего случая – это 0 – Vcc.

Если сравнить АЦП с линейкой, то минимальное и максимальное значение шкалы - диапазон входных напряжений, количество делений линейки – число кодовых комбинаций (число уровней квантования), расстояние между двумя соседними делениями - шаг квантования, а шаг квантования деленный на диапазон входных напряжений – разрешающая способность.

Пример использования АЦП

Допустим, у нас имеется в наличии фоторезистор – элемент меняющий свое сопротивление в

зависимости от уровня освещенности. И мы хотим организовать систему, в которой искусственное освещение будет автоматически включаться если естественный уровень света слишком низкий.

Для начала необходимо подключить фоторезистор к плате. Проще всего это сделать в виде делителя напряжения: одна нога датчика подключается к напряжению питания, а вторая через понижающий резистор к земле. Номинал постоянного резистора необходимо подбирать исходя из условий работы схемы (для более точной работы на ярком солнце нужно брать номинал пониже, а для определения уровней темноты – повыше), мы возьмем номинал 10 кОм. Зная зависимость сопротивления фоторезистора от освещенности при помощи закона Ома можно составить следующую таблицу:

Освещение окружения…	Освещение окружения (лк)	Сопротивление фоторезистора(Ω)	LDR + R (Ω)	Сила тока, проходящая через LDR +R	Напряжение, проходящее через R
Безлунная ночь	**0.1 лк**	600KΩ	610 KΩ	0,01 мА	0.05 В
Лунная ночь	**1 лк**	70 KΩ	80 KΩ	0.04 мА	0.41 В
Темная комната	**10 лк**	10 KΩ	20 KΩ	0.17 мА	1.65 В
Облачный день / Ярко освещённая комната	**100 лк**	1.5 KΩ	11.5 KΩ	0.29 мА	2.87 В
Солнечный день	**1000 лк**	300 Ω	10.03 KΩ	0.32 мА	3.20 В

Из нее видно, какие уровни напряжения будут при том или ином уровне освещенности. Теперь перейдем к коду.

Использование АЦП в MBed

Для работы с АЦП в MBed используется класс AnalogIn аналогичный рассмотренным ранее DigitalIn и DigitalOut.

Объявления аналогового входа:

```
AnalogIn Light(A0) ;
```

Light – имя нашей переменной,

A0 – название ножки к которой будет подключен фоторезистор

Для считывания значений может использоваться два метода:

```
float AnalogIn read()
```

Он возвращает действительное число в диапазоне [0.0, 1.0]. При этом 0 – это напряжение 0 В, а 1 – это уровень опорного напряжения – 3.3 В

```
uint16_t AnalogIn read_u16()
```

Возвращает целочисленное значение в диапазоне [0, 0xFFFF]

Включать освещение мы будем, если естественный уровень упадет ниже 10 люкс. Согласно таблице это соответствует 1,65 В, теперь это требуется перевести в нужные попугаи. Если 1 – это напряжение 3.3 В, то напряжение 1,65 будет равно 0.5

Напишем программу используя эти знания.

```cpp
#include "mbed.h"

//Объявляем аналоговый вход для определения уровня освещенности
AnalogIn Light(A0);

//Цифровой выход для включения лампочки
DigitalOut led(LED1);

//Пороговое значение, ниже которого будет включаться лампа
const float MIN_LIGHT = 0.5f;

int main()
{

    while(1) {

        //считываем текущее значение освещенности
        float currentLight = Light.read();

        //если она ниже порога
        if(currentLight < MIN_LIGHT) {
            //включаем свет
            led = 1;
        else {
            //иначе - выключаем
            led = 0;
        }

        wait(1.0);
    }
}
```

Данная программа в бесконечном цикле опрашивает значение текущей освещенности и управляет лампочкой.

Однако, в ней есть довольно неприятная ошибка. Как было сказано выше, АЦП обладают ограниченным уровнем точности измерений. Это означает, что при стабильном уровне входного сигнала, на него будут накладываться внутренние шумы АЦП, каждое измерение может отличаться от предыдущего на некоторую дельту в большую или меньшую сторону. Таким образом, если уровень освещенности медленно подойдет к указанному граничному уровню, то наши измерения могут давать следующие результаты:

№ измерения	значение
1	0.52
2	0.50
3	0.51
4	0.50
5	0.49
6	0.51
7	0.49
8	0.50
9	0.48
10	0.51

При этом, каждый раз когда измеренное значение будет переходит через пороговое 0.5, наша программа будет включать-выключать лампочку. Такое поведение недопустимо. Необходимо создать **гистерезисную систему управления.**

Гистерезис — свойство систем, мгновенный отклик которых на приложенные к ним воздействия зависит в том числе и от их текущего состояния, а поведение системы на интервале времени во многом определяется её предысторией.

В нашем случае, наша система обладает двумя устойчивыми состояниями

- Светло, лампочка выключена
- Темно, лампочка включена

Необходимо производить переход из одного состояния в другое с определенной дельтой (которая должна быть больше чем шум АЦП).

```
#include "mbed.h"

//Объявляем аналоговый вход для определения уровня освещенности
AnalogIn Light(A0);

//Цифровой выход для включения лампочки
DigitalOut lamp(LED1);

//Пороговое значение, ниже которого будет включаться лампа
const float MIN_LIGHT = 0.5f;

//дельта гистерезиса
const float hystDelta = 0.1f;

//Перечисляем возможные состояния системы
enum state_t {
    LAMP_OFF,
    LAMP_ON
};

//переменная которая хранит текущее состояние системы
state_t state = LAMP_OFF;

int main()
{
    while(1) {

        //считываем текущее значение освещенности
        float currentLight = Light.read();

        //Если мы находимся в состоянии лампа выключена
        if(state == LAMP_OFF) {
            //то включаем ее только если уровень освещения упал
            //до уровня 0.49
            if(currentLight < (MIN_LIGHT - hystDelta)) {
                lamp = 1;
                //переходим в противоположное состояние
                state = LAMP_ON;
            }
        }
        //если мы находимся в состоянии лампа включена
        else {
            //то выключаем ее только если уровень освещения вырос
            //до уровня 0.51
            if(currentLight > (MIN_LIGHT + hystDelta)) {
                lamp = 0;
                state = LAMP_OFF;
            }
        }

        wait(1.0);
    }
}

```

Варьируя величину hystDelta – можно подстраивать систему под конкретные ситуации, чем выше значение, тем больший уровень шумов будет сглаживаться, но одновременно с этим будет снижаться отзывчивость системы управления.

Другим способом избавления от шумов измерений будет использование цифрового фильтра (см. https://ru.wikipedia.org/wiki/Цифровой_фильтр), но оставим этот вопрос курсу МОТС.

Аналогичным образом можно строить системы управления с чувствительными элементами построенными на других принципах (ИК датчики, датчики магнитного поля и тд.).