Лабораторная работа №5

Работа с ЦАП

1. ***Теоритическая часть***

ЦАП – это АЦП с точностью до наоборот – он преобразовывает некоторые цифровые данные в их аналоговое представление (читай напряжение). Говоря еще проще - ЦАП позволит нам относительно плавно изменять напряжение на ноге контроллера. Области практического применения: генерация звука, и сигналов произвольной формы. Можно прикрутить к контроллеру SD карточку и сделать wav плеер.

Характеристики:

* Выходное напряжение от 0 до VddA вольт
* Аппаратная генерация шума и сигнала треугольной формы
* Два независимых канала
* Возможность работы в 8-ми и 12-ти разрядном режиме с левым или правым выравниванием

В данной лабораторной работе мы рассмотрим, как сгенерировать звук с помощью ЦАП.

Звук – распространение волн в среде. Характеризуется амплитудой (силой) и спектром частот (характером) колебаний. Существует множество способов генерации данных, которые в дальнейшем можно воспроизвести как звуковые волны: от физического моделирования до фильтрации осцилляторов и даже простого создания звуков с помощью элементарных математических функций.

В цифровом мире звук обычно представляется набором сэмплов: определенное количество раз в секунду (т.н. частота дискретизации) берется значение звукового давления и записывается в массив. Таким образом, получается дискретное представление сплошного сигнала. Естественно, чем больше частота дискретизации (соответственно, меньше время между снятием значений), тем точнее представляется исходный сигнал. Иллюстрация дискретизации сплошного сигнала:

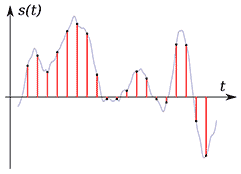


Рис.1 В результате дискретизации аналогового сигнала (синий график), получается набор равноудаленных сэмплов (черные точки)

Для представления синусоиды определенной частоты требуется минимум два сэмпла на период: один представляет волну в минимуме, другой – в максимуме. То есть, фактически, для представления синусоиды частотой N Гц требуется частота сэмплирования (дискретизации) Fs минимум 2\*N Гц. В теории обработки сигналов эта теорема звучит так:

если спектр функции x(t) не содержит частот выше FN Гц, то ее можно восстановить однозначно и без потерь с помощью последовательности точек, разнесенных не более чем на 1/(2\*FN) секунд;

и носит название теоремы Найквиста-Шеннона. Фактически, это объясняет наиболее распространённую частоту сэмплирования – 44100 Гц: максимальная частота звука, улавливаемого человеческим ухом лежит где-то в районе 20 КГц (минимальная частота – 20 Гц). Максимальную частоту, которую можно представить с заданной частотой дискретизации (фактически, равную Fs/2 Гц), называют частотой Найквиста.

Помимо частоты дискретизации другой важной характеристикой сигнала в дискретном мире является глубина дискретизации, так же называемая разрядностью или количеством уровней квантования. Эта величина показывает, сколько ячеек информации выделяется под хранение одного сэмпла. Для стандартного CD аудио используется глубина дискретизации в 16 бит на сэмпл, то есть, например если сигнал приведен к диапазону [-1; 1], то минимальный шаг внутри этого диапазона будет 2 / 65536 ≈ 3 \* 10-5. Понижение глубины дискретизации на существующем сигнале может привнести квантизационный шум.

Для начала стоит определить некоторые понятия, которые часто встречаются в различных статьях, связанных с программированием/редактированием звука.

В первую очередь, необходимо объяснить понятие «звуковой спектр», что это такое, и как с ним работать. **Звуковой спектр** - это состав аккустического сигнала, информация о том, как энергия сигнала распределена по частотам. Чаще всего визуализируется с помощью графика амплитуда-частота.

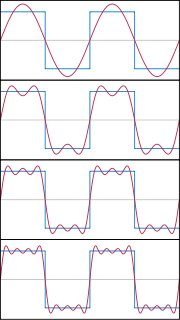


Рис.2 Представление исходной «квадратной» волны суммой более простых синусоидальных волн

Далее, понятие **обертон** (нем. oberton, англ. overtone — «верхний тон») в составе спектра звука означает любой призвук, частота которого выше фундаментальной (самой нижней) частоты звука. Обертоны бывают гармоническими (англ. harmonic) и негармоническими: в первом случае их частота кратна фундаментальной частоте, во втором же – нет. Базовая частота и обертоны формируют то, что называют тембром звука, его специфический «окрас», позволяющий, например, определить, какой музыкальный инструмент звучит.

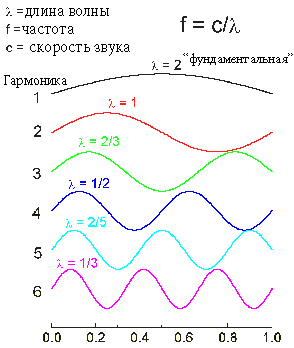
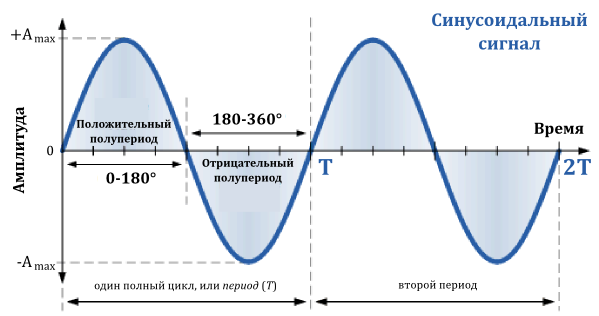


Рис.3 иллюстрация гармонических обертонов

Стоит упомянуть, что слух человека воспринимает музыкальные звуки, отношение частот которых равно двум, как очень похожие, на этом построена «октавная система». То есть частота ноты «до» в определенной октаве ровно в два раза больше частоты ноты «до» соседней, более низкой октавы: Fc2 / Fc1 = 2. Для того, чтобы определить частоту, соответствующую фортепианной клавише, можно воспользоваться следующей формулой (октава на фортепиано равномерно покрыта 12-ю клавишами, n - порядковый номер клавиши на фортепиано, для ноты «ля» субконтроктавы n = 0):

RT softsynth. Частота в зависимости от номера клавиши | Процедурная генерация звука в реальном времени. Основы

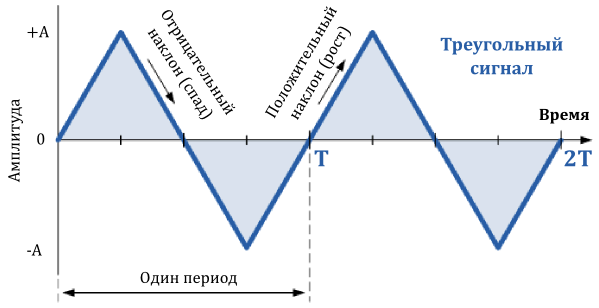
Поскольку звуковые волны – это распространение колебаний в среде, а простейший вид колебаний – синусоидальные.



Период и частота математически являются обратными друг другу величинами. С уменьшением времени периода сигнала, его частота увеличивается и наоборот.

**Треугольные сигналы**

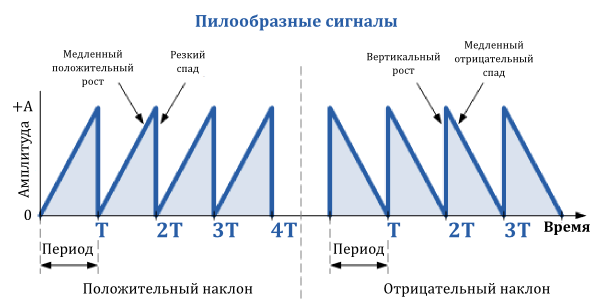
Треугольные сигналы, как правило, это двунаправленные несинусоидальные сигналы, которые колеблются между положительным и отрицательным пиковыми значениями. Треугольный сигнал представляет собой относительно медленно линейно растущее и падающее напряжение с постоянной частотой. Скорость, с которой напряжение изменяет свое направление равна для обоих половинок периода, как показано ниже.



В случае несимметричной треугольной формы сигнала, которую мы можем получить изменением скорости роста и спада на различные величины, мы имеем еще один тип сигнала известный под названием пилообразный сигнал.

**Пилообразный сигнал**

Пилообразный сигнал — это еще один тип периодического сигнала. Как следует из названия, форма такого сигнала напоминает зубья пилы. Пилообразный сигнал может иметь зеркальное отражение самого себя, имея либо медленный рост, но очень крутой спад, или чрезвычайно крутой, почти вертикальный рост и медленный спад.



Чтобы создать звук нам нужно заставить колебаться мембрану динамика с определенной частотой. Каждой ноте соответствует своя частота. Например, ноте ДО 1 октавы, соответствует частота 261 Гц, т.е дрыгая ногой микроконтроллера, подключенной к динамику, со скоростью 261 раз в секунду, мы будем слышать звучание этой ноты. Для тех, кто не силен в музыкальной теории, звук ближе от 1 кГц и выше будет более писклявым, ниже 300 Гц будет басить.

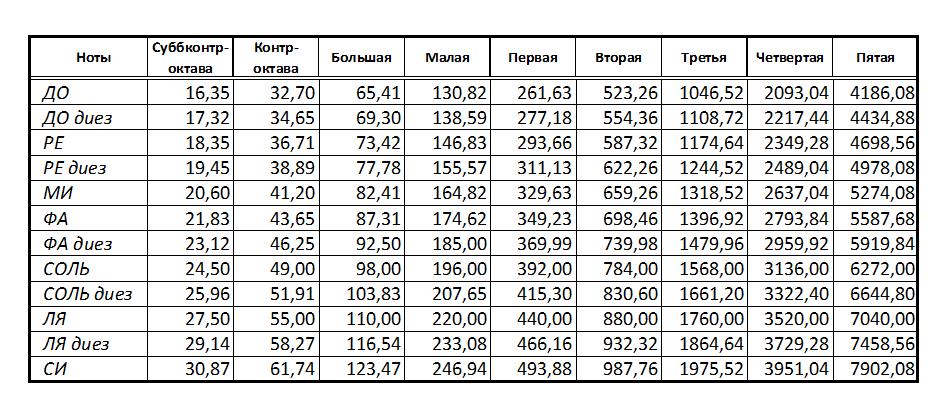


Таблица частот нот



1. ***Практическая часть***

Звуковой сигнал — синусоида, для генерации синусоиды - используется табличный метод. Т.е. он не вычисляется, а значение просто извлекается из массива и выводится в ЦАП всякий раз когда возникает прерывание от таймера.

Во-первых, выберем круглую частоту дискретизации. К примеру, 10кГц.

Чтобы заставить ногу двигаться с такой скоростью нам поможет таймер, работу которого мы рассмотрели в лабораторной работе №3.

Принцип формирования частоты таков: таймер тикает до тех пор, пока его значение не совпадает с заданными. В момент совпадения с заданным значением с текущим значением таймера происходит прерывание, в котором текущее значение инвентируется (вкл/откл),таким образом мы получаем пульсирующий сигнал. Нужно соотнести каждой ноте частоту и по очереди их воспроизводить, т.е создать массив со значениями, которые по очереди будем подставлять в заданное значение.

TIM\_TimeBaseStructure.TIM\_Period = 23; // Это 1 MГц

TIM\_TimeBaseStructure.TIM\_Prescaler = 99; // При делителе 23 и предделителе 99 прерывание каждые 100 мкс (10 кГц)

Теперь берём частоту ноты Ля, например, Субконтроктавы — 27,5 Гц. Ля выбрана потому, что частота этой ноты является целым числом и считается эталоном (440 Гц Первой октавы, обычно камертоны вибрируют с этой частотой), от неё по формуле image считаются остальные 11 нот (с учётом диезов). Частоты нот соседних октав отличаются друг от друга ровно в два раза (подставляем в формулу 12 вместо n и убеждаемся в этом).

Так нам надо рассчитать массив для каждой ноты. Легче всего это сделать в Excel. План расчёта значений с помощью Excel:

1. Несколько октав, в каждой по 12 нот (вместе с диезами).
2. Для каждой ноты расчитывается частота по формуле исходя из базовой (Ля — 27,5 Гц) и количества ступеней до неё (Ля# — это +1 ступень, Си — +2 ступени, Ля следующей октавы — +12 ступеней).
3. Получаем длины периодов в секундах путём деления единицы на частоту, результат делим на 10 000 Гц, округляем и получаем количество элементов в массиве для данной ноты при частоте 10 000 Гц.
4. Высчитываем массив значений синуса конкретной ноты с учётом периода этой ноты.
5. Нормализуем в диапазон от 0 до 4095, округляем.

Значения синуса для нот через каждые 100 микросекунд, от 0 до 4095:

//Нота До

const uint16\_t noteC[64]={

76,2216,2383,2547,2709,2865,3017,3162,3299,3428,3547,3657,3755,3842,3917,

3979,4028,4064,4086,4095,4090,4070,4038,3991,3932,3860,3775,3679,3572,

3455,3328,3193,3050,2900,2744,2584,2419,2253,2085,1917,1749,1584,1422,

1264,1111,965,826,695,574,462,361,271,194,129,76,38,12,1,3,19,49,92,149,

218,300,393,498,613,738,871,1013,1161,1316,1475,1639,1805,1973

};

//Нота До диез

const uint16\_t noteCd[64]={

72,2226,2402,2576,2746,2911,3069,3220,3361,3493,3614,3722,3818,3901,3970,

4024,4063,4087,4095,4088,4065,4027,3974,3907,3825,3730,3622,3503,3372,

3231,3081,2924,2759,2590,2416,2239,2061,1883,1706,1532,1362,1196,1038,

887,744,612,490,381,284,200,130,75,35,10,0,6,27,64,116,182,263,357,464,

583,713,853,1002,1159,1323,1492,1666,1842,2020

};

//Нота Рэ

const uint16\_t noteD[64]={

68,2236,2423,2607,2786,2959,3124,3280,3425,3559,3680,3787,3879,3955,4016,

4059,4086,4095,4087,4061,4018,3959,3883,3792,3686,3566,3432,3288,3132,

2968,2795,2617,2433,2246,2057,1869,1682,1498,1318,1145,980,823,677,543,

421,314,221,143,82,38,10,0,7,32,74,133,207,298,403,523,655,799,954,1118,

1291,1469,1652,1839,2028

};

//Нота Рэ диез

const uint16\_t noteDd[64]={

63,2247,2445,2639,2828,3009,3181,3342,3490,3625,3745,3849,3935,4003,

4053,4084,4095,4087,4059,4012,3946,3862,3761,3643,3510,3363,3204,

3034,2854,2666,2473,2275,2076,1876,1678,1483,1293,1111,938,775,

625,488,366,260,171,100,48,14,0,6,31,76,139,220,319,434,565,710,

868,1037,1215,1402,1594,1792,1991

};

//Нота Ми

const uint16\_t noteE[64]={

60,2259,2469,2673,2872,3061,3239,3405,3556,3691,3809,3907,3986,4044,4080,

4095,4087,4058,4008,3936,3844,3733,3604,3458,3297,3122,2937,2741,2538,

2330,2119,1907,1697,1490,1289,1097,914,744,588,447,324,219,133,69,25,

3,3,24,68,133,218,323,446,586,742,912,1095,1287,1488,1694,1905

};

//Нота Фа

const uint16\_t noteF[64]={

57,2272,2493,2709,2917,3115,3300,3470,3623,3756,3870,3961,4029,4074,

4094,4089,4060,4007,3930,3831,3710,3569,3410,3234,3044,2842,2631,

2413,2190,1965,1742,1522,1309,1104,911,731,567,421,295,189,106,47,

11,0,14,51,113,199,306,434,582,748,929,1123,1329,1543,1763,1987

};

//Нота Фа диез

const uint16\_t noteFd[64]={

54,2285,2519,2747,2966,3172,3362,3536,3689,3820,3927,4008,

4063,4091,4092,4065,4010,3929,3823,3693,3540,3367,3177,

2971,2753,2526,2291,2054,1817,1582,1354,1135,929,738,564,

410,279,171,89,33,4,3,29,83,163,269,399,550,723,912,1118,1336,1563,1797,2034

};

//Нота Соль

const uint16\_t noteG[64]={

51,2299,2547,2787,3016,3230,3426,3602,3754,3880,3978,4047,4086,

4094,4071,4017,3934,3822,3683,3519,3333,3127,2905,2670,2426,2176,

1924,1674,1430,1194,972,766,580,416,276,163,79,25,1,8,47,115,213,

338,490,665,861,1075,1303,1543,1791,2042

};

//Нота Соль диез

const uint16\_t noteGd[64]={

48,2314,2576,2829,3068,3290,3492,3668,3817,3936,4023,4076,

4095,4079,4028,3944,3828,3681,3506,3307,3086,2848,2596,

2335,2069,1802,1540,1286,1045,821,618,440,288,167,78,22,

0,14,61,143,257,401,574,771,991,1228,1479,1739,2005

};

//Нота Ля

const uint16\_t noteA[64]={

45,2330,2606,2872,3123,3353,3558,3734,3878,3987,4059,4092,

4087,4043,3961,3842,3689,3504,3292,3056,2801,2532,2253,

1970,1689,1415,1153,907,684,486,319,184,85,23,0,16,71,

163,292,454,646,865,1107,1366,1639,1919

};

//Нота Ля диез

const uint16\_t noteAd[]={

42,2346,2639,2918,3179,3416,3624,3798,3934,4030,4083,4093,

4059,3982,3864,3707,3514,3290,3039,2767,2480,2183,1883,

1587,1301,1031,782,561,371,218,103,30,1,15,72,172,313,

490,700,940,1203,1484,1777

};

//Нота Си

const uint16\_t noteB[64]={

40,2364,2673,2967,3238,3481,3690,3859,3985,4064,4095,4076,

4009,3894,3736,3536,3301,3036,2747,2441,2126,1809,1498,1199,

922,671,453,274,137,46,3,10,65,168,316,506,732,991,1274,1577,1890

};

// Музыкальная пауза

const uint16\_t noteP[]={

1,2048

};

Теперь функция, возвращающая очередное значение передаваемой ей ноты с поправкой на октаву (указывать надо 1, 2, 4, 8 и т.д.):

uint16\_t mnote(const uint16\_t \*pa, char octave) // Возвращаем значение амплитуды ноты для данного момента времени

{

int ii=0;

do {ii=i;} while (ii!=i);

ii=(octave\*ii)%pa[0];

ii=\*(pa+ii);

return ii;

}

Переменная i увеличивается каждые 100 микросекунд с помощью прерывания:

//обработчик прерываний от таймера

void TIM6\_DAC\_IRQHandler(void) {

TIM\_ClearITPendingBit(TIM6, TIM\_IT\_Update); // сбрасываем флаг

DAC\_SetChannel2Data(DAC\_Align\_12b\_R, sin64[i]); //запихиваем в ЦАП очередной элемент массива

i++;

if (i == 64) i = 0;// если вывели в ЦАП все 64 значения, то начинаем заново

}

В общем, самое время написать мелодию.

uint16\_t \*melody[]={

noteE,noteP,noteG,noteP,

noteA,noteA,noteP,noteE,

noteP,noteG,noteP,noteAd,

noteA,noteA,noteP,noteP,

noteE,noteP,noteG,noteP,

noteA,noteA,noteP,noteG,

noteP,noteE,noteP,noteP,

noteP,noteP,noteP,noteP

};

Чтобы образовать трезвучие (аккорд), добавим ещё две мелодии, с другими нотами:

uint16\_t \*melody2[]={

noteB,noteP,noteD,noteP,

noteE,noteE,noteP,noteB,

noteP,noteD,noteP,noteF,

noteE,noteE,noteP,noteP,

noteB,noteP,noteD,noteP,

noteE,noteE,noteP,noteD,

noteP,noteB,noteP,noteP,

noteP,noteP,noteP,noteP

};

uint16\_t \*melody3[]={

noteGd,noteP,noteB,noteP,

noteCd,noteCd,noteP,noteGd,

noteP,noteB,noteP,noteD,

noteCd,noteCd,noteP,noteP,

noteGd,noteP,noteB,noteP,

noteCd,noteCd,noteP,noteB,

noteP,noteGd,noteP,noteP,

noteP,noteP,noteP,noteP

};

Для одновременного воспроизведения сразу нескольких нот просто складываем значения их текущих амлитуд (из массивов). Но! ЦАП принимает значения от 0 до 4095, и элементы массивов нормализованы в этот же диапазон. Если нам не нужен эффект овердрайва, просто делим получившуюся сумму на количество звучащих нот, результат скармливаем ЦАПу. И так каждые 100 микросекунд…

Пример кода для воспроизведения одного звука

#include "stm32f4xx.h"

#include "stm32f4xx\_rcc.h"

#include "stm32f4xx\_gpio.h"

#include "stm32f4xx\_dac.h"

#include "stm32f4xx\_tim.h"

//массив, элементы которого нужно запихать в DAC чтобы получить синус

const uint16\_t sin32[32]= {

2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056, 4095, 4056,

3939, 3750, 3495, 3185, 2831, 2447, 2047, 1647, 1263, 909,

599, 344, 155, 38, 0, 38, 155, 344, 599, 909, 1263, 1647

};

u8 i = 0;

void TIM6\_DAC\_IRQHandler(void);

int main(void) {

SystemInit();

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE); //включаем порт D

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_DAC, ENABLE); //включаем ЦАП

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM6,ENABLE); //включаем таймер 6

// Настраиваем ногу ЦАПа

GPIO\_InitTypeDef GPIO\_InitStruct;

GPIO\_StructInit(&GPIO\_InitStruct);

GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_5;

GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_AN;

GPIO\_Init(GPIOA, &GPIO\_InitStruct);

// настрройка таймера

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_TimeBaseStructInit(&TIM\_TimeBaseStructure);

TIM\_TimeBaseStructure.TIM\_Period = 500;

TIM\_TimeBaseStructure.TIM\_Prescaler = 0;

TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1;

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIM6, &TIM\_TimeBaseStructure);

TIM\_Cmd(TIM6, ENABLE);

TIM\_ITConfig(TIM6, TIM\_IT\_Update, ENABLE); // включаем прерывания по переполнению

NVIC\_EnableIRQ(TIM6\_DAC\_IRQn); //разрешение TIM6\_DAC\_IRQn прерывания

//включаем DAC

DAC\_Cmd(DAC\_Channel\_2, ENABLE);

while(1){

}

}

//обработчик прерываний от таймера

void TIM6\_DAC\_IRQHandler(void) {

TIM\_ClearITPendingBit(TIM6, TIM\_IT\_Update); // сбрасываем флаг

DAC\_SetChannel2Data(DAC\_Align\_12b\_R, sin32[i]); //запихиваем в ЦАП очередной элемент массива

i++;

if (i == 32) i = 0;// если вывели в ЦАП все 32 значения то начинаем заново

}.