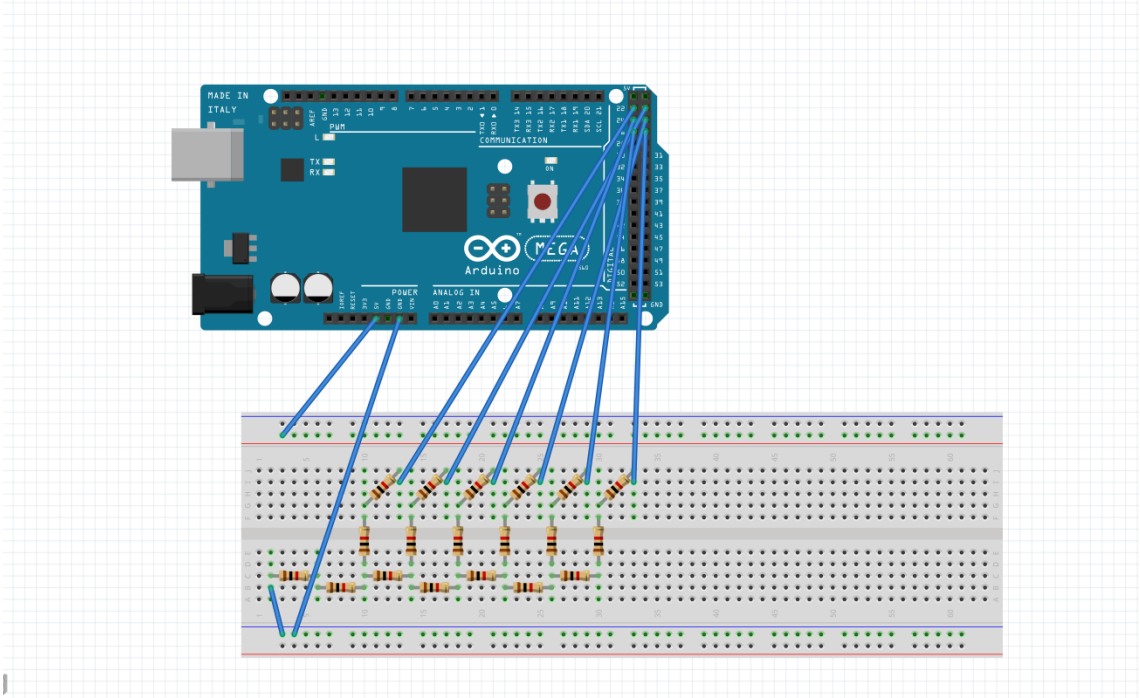
Processamento Digital de Sinal

Trabalho Prático 1



Grupo 4:

43861 – Francisco Chicharro

43864 – Joao Monteiro

43874 – Joao Florentino

Docente: Paulo Marques

08/11/2019

Índice

Índice Figuras

Objetivos

Este trabalho tem como objetivos:

- Implementar a interface analógica para o Arduino Mega. A reconstrução de sinais será suportada no conversor D/A com estrutura R2R com 6 bit.
- Implementar um programa que realize o “echo” dos sinais analógicos, com ganho unitário. Testar o sistema e verificar os limites em termos de: frequência e amplitudes máximas e mínimas para os sinais de entrada e de saída; frequência de amostragem; níveis de quantificação.
- Projetar um gerador de sinais com forma de onda arbitrária e frequência e amplitude programáveis, verificar o funcionamento e testar os limites do gerador.
- Projetar e implementar o código que sintetiza um sistema FIR genérico e que utilize interrupções para estabelecer o tempo entre amostras, alterando o conversor R-2R de forma a suportar 8 bit.
- Projetar um sistema que realizasse o disfarce da voz humana, permitindo a sua inteligibilidade mas impedindo o reconhecimento do individuo.

Introdução

Entende-se por norma que um **sinal** é uma sequência de estados em um sistema de comunicação que codifica uma mensagem. Um sinal pode ser de tempo contínuo ou de tempo discreto, digitais ou analógicos.

Sinais de **tempo discreto** são sequências de valores, normalmente definidos em instantes de tempo periódicos. Sinais de **tempo contínuo** possuem o seu estado definido em qualquer instante de tempo.

Um **sinal digital** é uma sequência discreta (descontínua) no tempo e em amplitude. Isso significa que um sinal digital só é definido para determinados instantes de tempo, e que o conjunto de valores que pode assumir é finito. Um **sinal analógico** é um sinal contínuo cuja variação em relação ao tempo é a representação proporcional de outra variável temporal

Num sistema de comunicações o transmissor recebe uma mensagem e codifica-a em um sinal, que é transportado pelo sistema de comunicações até ao recetor, que o decodifica em uma mensagem. Do outro lado o recetor recebe um sinal, sendo este decodificado. É gerado assim uma mensagem que se espera ser igual á originalmente transmitida.

O processamento de sinais consiste na análise e/ou modificação de sinais utilizando

Exercício I

Implemente a interface analógica, descrita na aula teórica, para o Arduino Mega. A reconstrução de sinais será suportada no conversor D/A com estrutura R2R com 6 bit.

O esquema do trabalho encontra-se representado na figura 1, o gerador de sinais dá origem ao sinal que é enviado diretamente para o Arduino, este apresenta internamente um conversor Analógico-Digital que converte as tensões recebidas em valores binários, lidos pelo porto A0.

Int amostra = analogRead(A0);

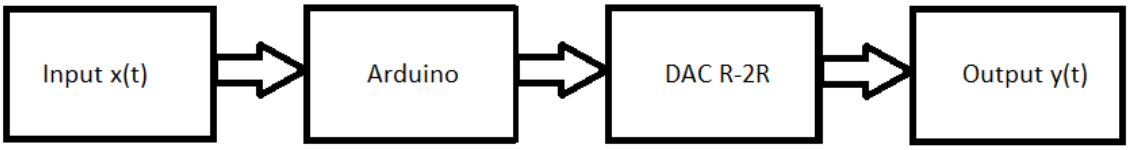


Figura 1– Esquema de Montagem

Para criar o conversor analógico-digital (figura 2 e3) a partir do sistema R-2R foram necessárias 20 (18 resistências para o circuito R-2R e 2 resistências para o circuito do condensador) resistências de 1 kΩ para um total de 6 bits. Visto que não existem resistências de exatamente 2 kΩ, foi necessário colocar em série o dobro das resistências necessárias.

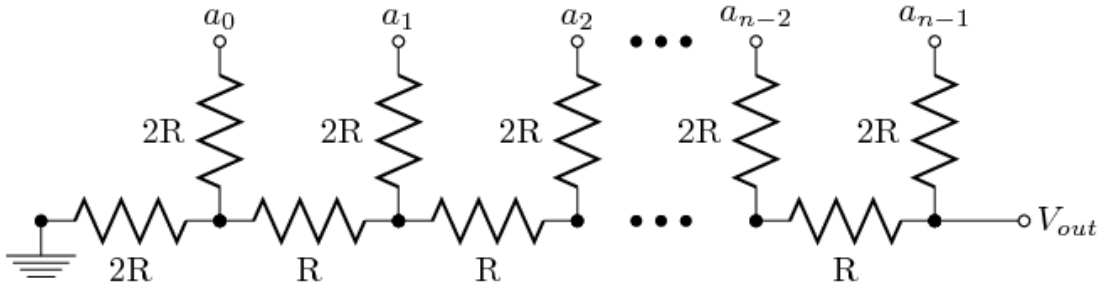


Figura 2- Representação do DAC

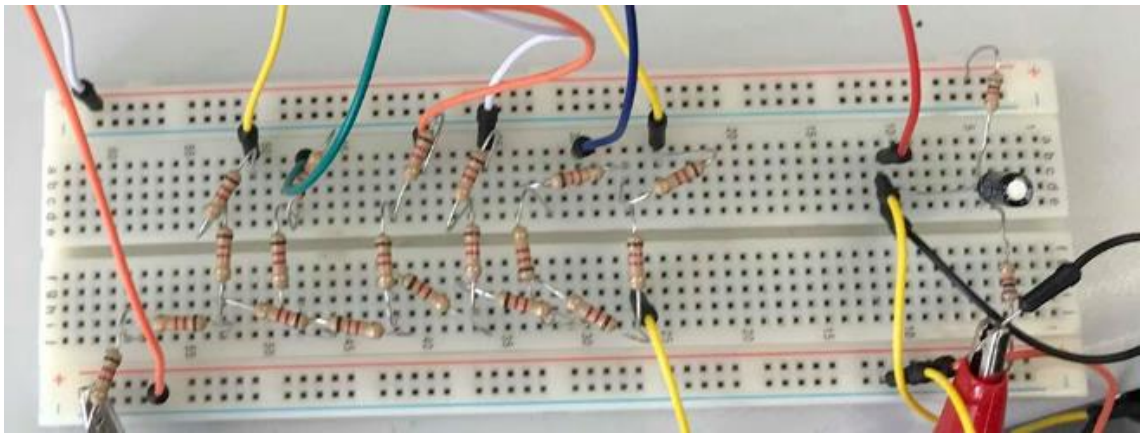


Figura 3- DAC implementado com circuito R-2R

Na figura 3 pode-se observar a ligação entre o Arduino e o DAC a partir dos pinos 22-28, funcionando estes como Output do Arduino.

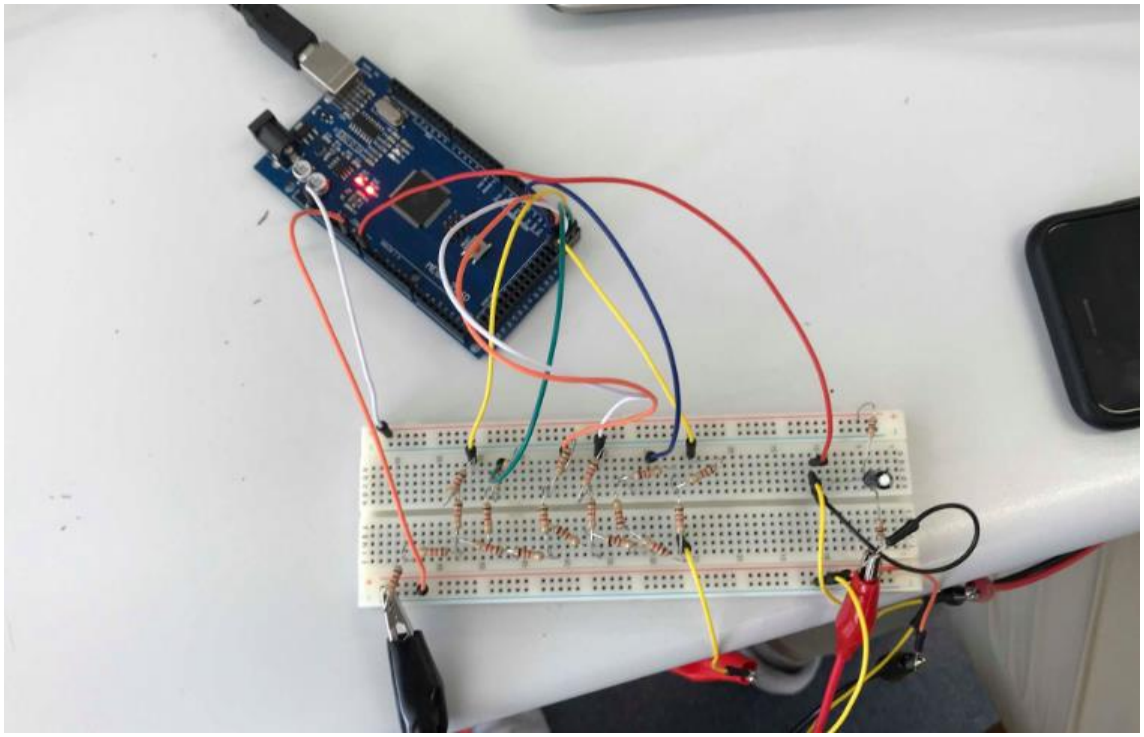


Figura 4- Circuito Arduino– R-2R

Este circuito permitiu-nos ter 6 bits de representação dos valores de tenção. Quanto maior o número de bits, mais próximo da realidade estará o sinal, visto que os valores de output do Arduino são discretos ao contrário dos valores de tensão de entrada que compõe um sinal contínuo.

Para representar a correta montagem do circuito Arduino – R-2R, utilizámos o programa "Gerador de sinal em rampa" disponibilizado nos slides da cadeira:

Exemplo: Gerador de sinal em rampa ("força bruta")

```
int rampa = 0;

void setup(){
    ...
 // Colocar pinos PORTA como
    output
 for(int i=22; i<30;i++){
  pinMode(i,OUTPUT);
    ...
 }
}
```

```
void loop(){
    ...
    PORTA = rampa++;
    if(rampa>=64){
          rampa=0;
    }
    ...
    delayMicroseconds(100);
}
```

Arduino Mega Porto A = A0...A7 -> Pinos digitais 22...29

Figura 5- Programa "Gerador de sinal em rampa"

Observando o output do sistema, conseguimos observar a "rampa" desejada, com os números de níveis de quantificação desejados:

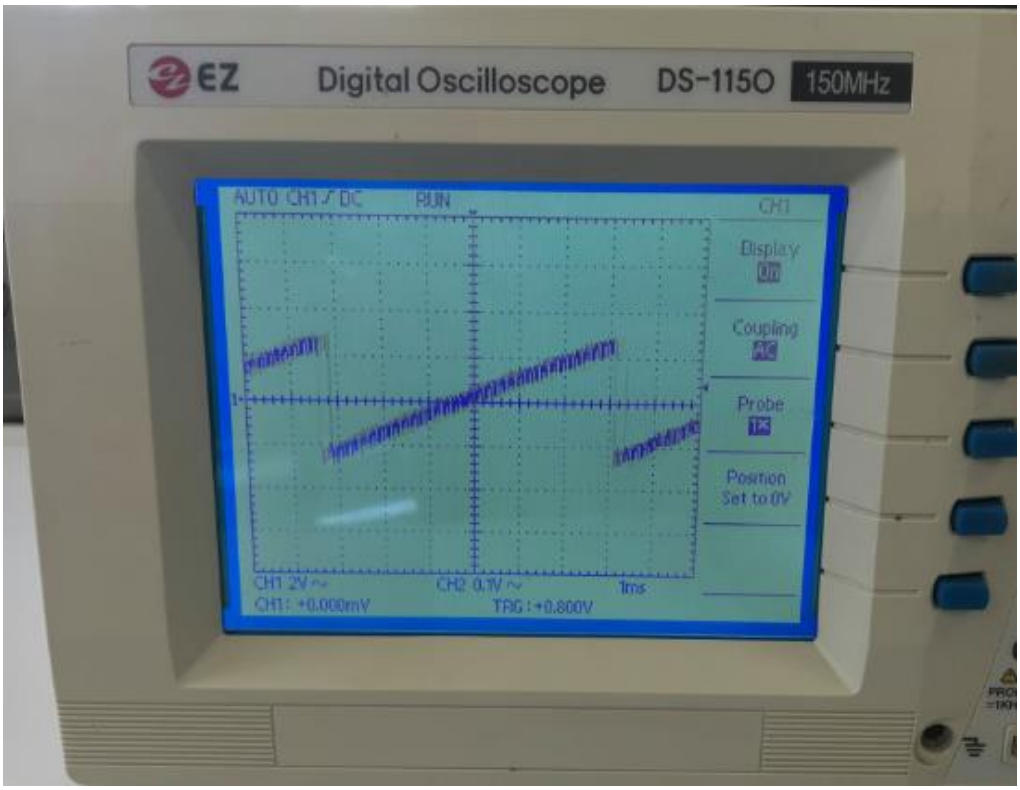


Figura 6 - Representação da "rampa" no osciloscópio

Exercício II

Implemente um programa que realize o "echo" dos sinais analógicos, com ganho unitário. Verificar os limites do sistema em termos de: frequência e amplitudes máximas e mínimas para os sinais de entrada e saída, frequência de amostragem, níveis de quantificação.

Valores de frequência e amplitude:

Para testar os valores de frequência máximos e mínimos, fizemos variar este valor no gerador, observando no osciloscópio o comportamento tanto do sinal de entrada como o sinal de saída:

Sinal de entrada: fmin = 200 hz e fmax = 2k hz
Sinal de saída: fmin = 180hz e fmax = 600 hz

Estes valores foram retirados observando as frequências onde o sinal deixava de respeitar o sinal original, alterando a sua forma original. Uma vez encontrada a frequência máxima conseguimos encontrar a largura de banda do sinal.

Para a amplitude, fizemos o mesmo processo, mas agora variando a amplitude no gerador:

Sinal de entrada: Amin = 0.2V e Amax = 5v
Sinal de saída: Amin = 0.3V e Amax = 2.5V

A amplitude máxima corresponde ao máximo valor de amplitude que o Arduíno possui, ou seja, 5V.
No entanto, esperávamos um valor de amplitude superior no sinal de saída, mas o circuito do condensador fez atenuação e filtragem no sinal, baixando assim o seu valor de amplitude.

Outro motivo para a amplitude de saída sobre a sua queda para metade é o erro no primeiro bit, causando um shift errado no circuito (provavelmente alguma falha

no circuito uma vez que não temos exatamente resistências com o dobro do valor da anterior).

Frequência de amostragem

O valor de frequência de amostragem deve respeitar o ritmo de Nyquist, ou seja, deve ser superior a 2 vezes o valor máximo da frequência do sinal

No caso do sinal de entrada: Famostragem > 2 * 2Kz = 4kHz

No caso do sinal de saída: Famostragem > 2* 600Hz = 1.2 kHz

Níveis de quantificação

O número de níveis de quantificação corresponde a 2^n onde n é o número de bits que o nosso sistema possui que são 6, tendo assim 32 níveis de quantificação

Exercício III

Projete um gerador de sinais com forma de onda arbitrária e frequência e amplitude programáveis, verifique o funcionamento e teste os limites do gerador.

Neste exercício não utilizámos equipamento externo (gerador, osciloscópio) mas sim o próprio Arduíno que tem capacidade de gerar sinais modelando frequências e amplitudes. Baseámo-nos nos sinais mais conhecidos (Sinal sinusoidal, sinal retangular, sinal triangular e rampa).
Estes sinais foram gerados em MATLAB com objetivo de explorar o potencial que o Arduíno tem no que toca ao processamento de sinal.

Foram criados 4 arrays distintos cada um contendo 64 valores que representam amostras dos sinais correspondentes. Estas amostras apresentam valores entre 0-255 visto que o Arduíno tem 8 bits para fazer a sua representação em valores de tensão.

```
%%Rampa
x2 = round(A * Rampa(2*pi*f0*t) + A);
subplot(4,1,2);
plot (t,x2)
```

```
function [ x ] = Rampa( t )
%RAMPA Summary of this function goes here
%   Detailed explanation goes here
x = sawtooth (t);
end
```

Figura 7- Excerto de código Matlab para sinal de rampa

Gerados todos os sinais em MATLAB, colocámos todos os valores no array correspondente a cada sinal:

```
int sinaux[] = {126,126,125,123,121,118,115,111,107,102,97,92,86,80,74,68,61,55,49,43,37,
32,26,21,17,13,9,6,4,2,1,0,0,1,2,4,6,9,13,17,21,26,
32,37,43,49,55,61,68,74,80,86,92,97,102,107,111,115,118,121,123,125,126, 126};

int ramp[]= {0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,
42,   44,   46,   48,   50,   52,   54,   56,   58,   60,   62,   64,   66,   68,   70,   72,   74,   76,   78,   80,   82,
84,   86,   88,   90,   92,   94,   96,   98,   100,  102,  104,  106,  108,  110,  112,  114,  116,  118,  120,  122,  124,
0};

int retang[]= {126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,
126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   126,   0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0, 126};

int triang[] = {0,    4,    8,    12,   16,   20,   24,   28,   32,   36,   40,   44,   48,   52,   56,   60,   64,   68,   72,   76,   80,
84,   88,   92,   96,   100,  104,  108,  112,  116,  120,  124,  124,  120,  116,  112,  108,  104,  100,  96,   92,   88,
84,   80,   76,   72,   68,   64,   60,   56,   52,   48,   44,   40,   36,   32,   28,   24,   20,   16,   12,   8,    4,
0};
```

De seguida observou-se os sinais gerados pelo Arduíno no osciloscópio sendo que estes apresentam falhas entre “steps” por serem discretos:

Sinal Sinusoidal

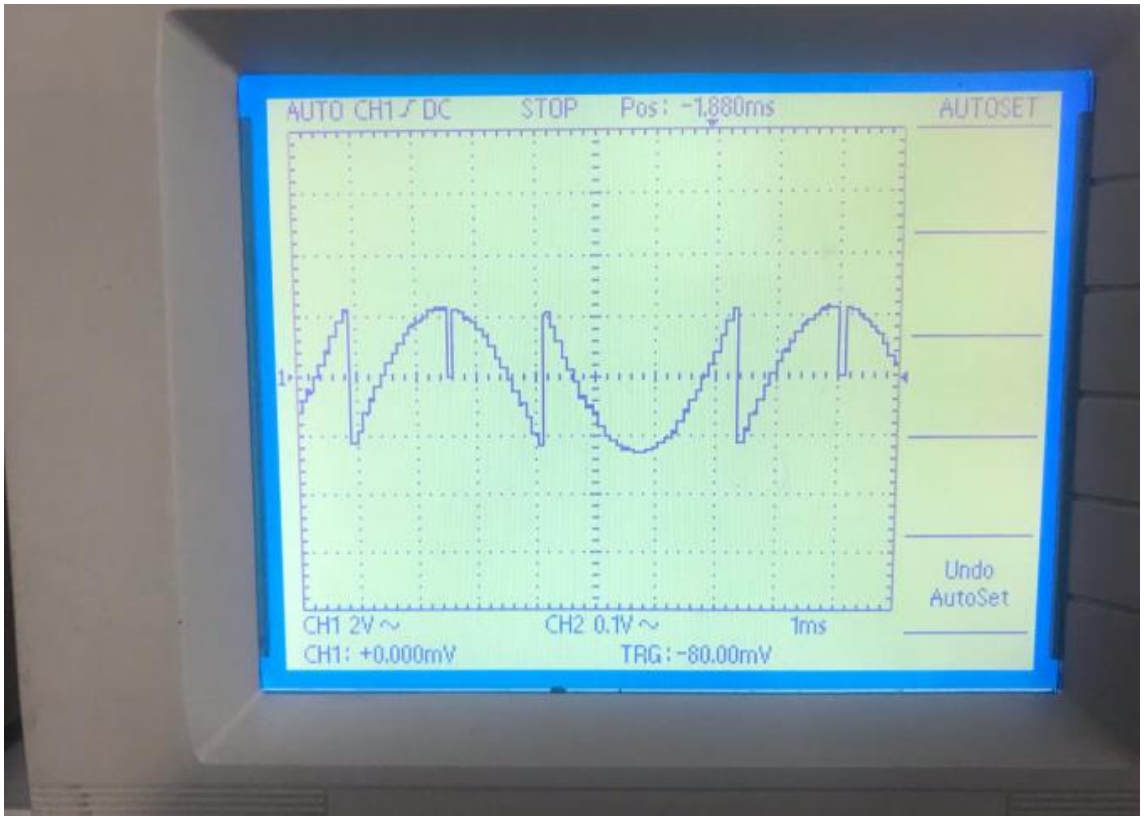


Figura 8- Sinal Sinusoidal

Sinal Retangular

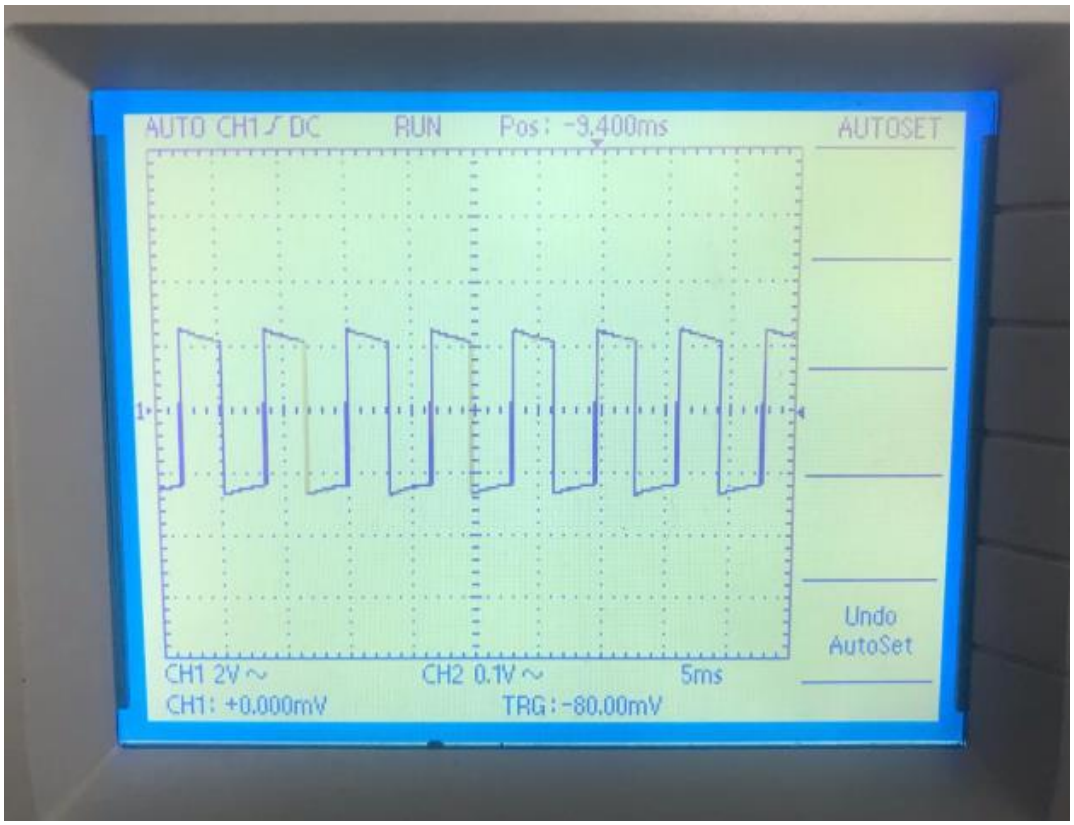


Figura 9- Sinal Retangular

Sinal Rampa

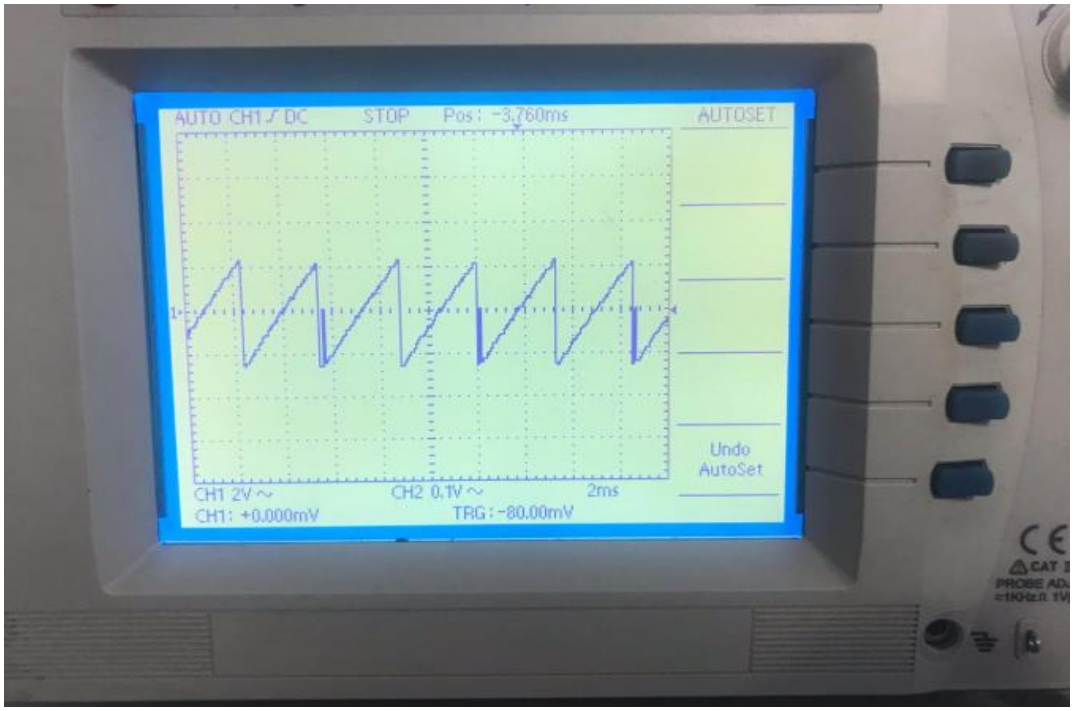


Figura 10-Sinal Rampa

Sinal Triangular

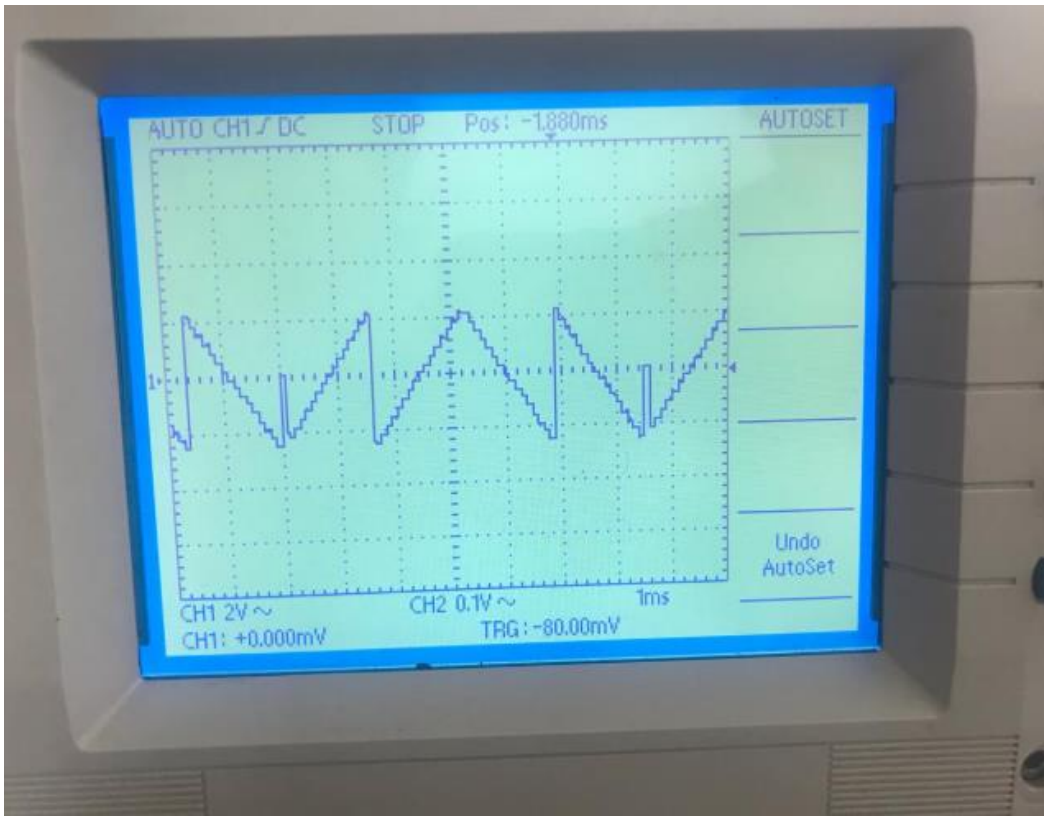


Figura 11- Sinal Triangular

Exercício IV

4.1 Passa baixo

No laboratório anotamos os valores chave onde a amplitude do sinal era completamente nula e só depois é que comparamos com os dados teóricos, tirando partido do MATLAB.

A partir da análise dos gráficos do osciloscópio concluímos que a função se trata de um filtro passa baixo. Verificámos que as baixas frequências até certo ponto tinham o máximo de amplitude até se irem aproximando da frequência de corte, na qual se anulavam por completo.

Alterando o valor de N (para as funções levava ao aparecimento de um maior número de zonas de corte, nos quais as funções tomavam valor nulo)

Concluímos que a frequência de corte ocorria aproximadamente de Fs/N em Fs/N para uma frequência de amostragem de 5kHz.

A partir das figuras abaixo e dos valores de frequência presentes no gerador concluímos que a frequência de corte se encontra em Fs/N para os diferentes valores de N. Assim sendo após este valor as frequências passam a ser de novo baixas, deixando assim “passar” o sinal.

Não conseguimos obter exatamente o zero para esta frequência de corte uma vez que tivemos de utilizar 3 resistências de 1kHz (diferentes das que utilizámos no circuito R-2R de 1.2kHz) pelo que assim a atenuação afetou a frequência de corte.

Para N=2:

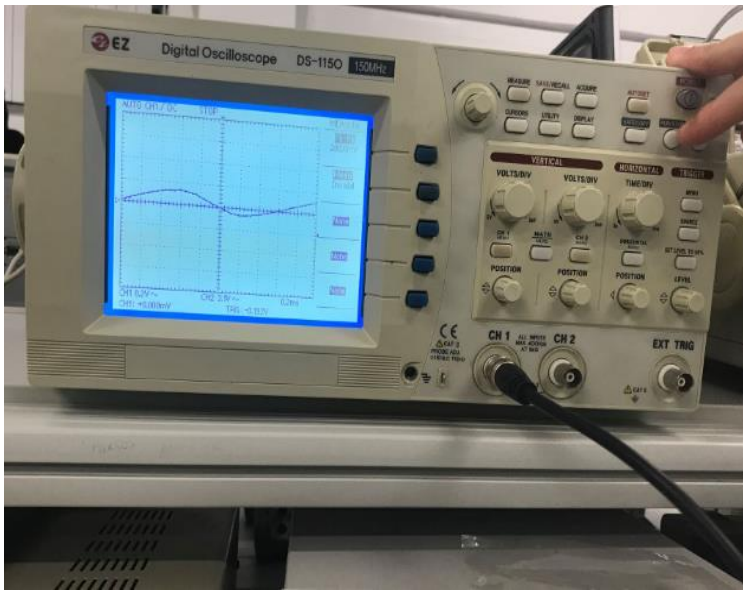


Figura 12: Zeros em N=2

Figura13: Freqência de corte para Fs = 5kHz e N = 2

Para N=4

Alterando o valor de N, a nova frequência de corte será diferente, situando-se agora em Fs/4:

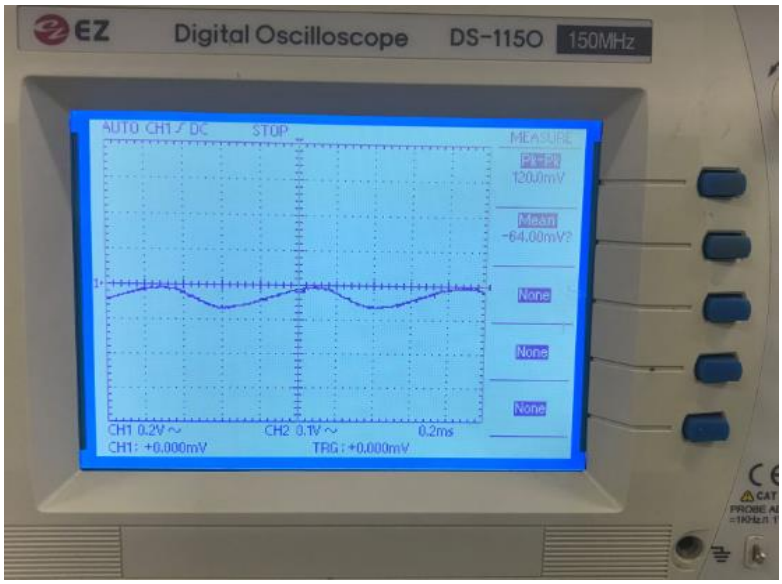


Figura 14: Zeros para N=4

Figura 15: Frequência de corte para Fs=5kHz N=4

Exercício V

Na implementação do código respetivo a este exercício encontrámos uma forma de disfarçar a voz humana, conseguindo colocá-la mais grave ou aguda, o seguinte gráfico mostra a ideia base a utilizar, sendo que é necessário dividir as amostras a metade do seu período:

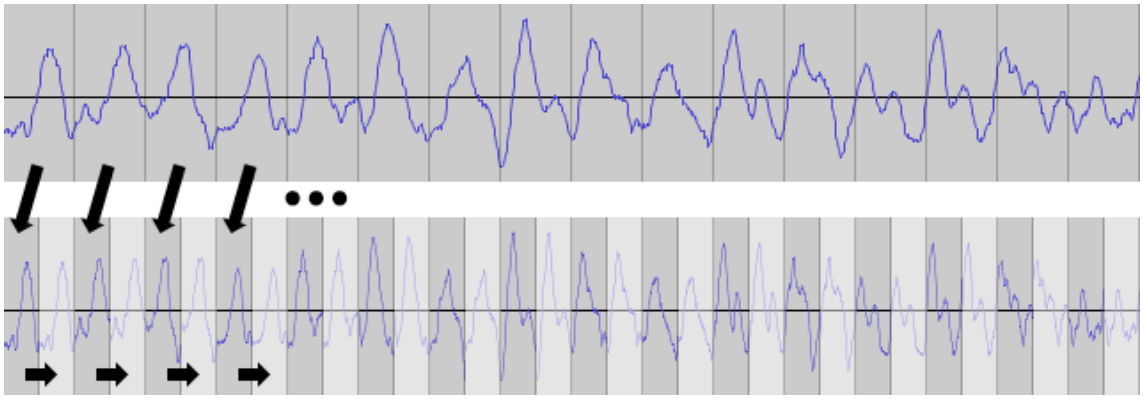


Figura 16- Disfarce da Voz

Pegando em amostras de áudio, comprimindo-as até metade do seu tamanho e de seguida adicionando uma réplica sua ao sinal final de que o Arduíno irá retornar conseguimos disfarçar a voz da pessoa que originou o sinal inicial.

Utilizámos o código da função do exercício 4 e adaptámos a este exercício sendo que as ideias são parecidas: Criação de um buffer onde utilizámos dois índices: Um para a leitura das amostras e outro para a escrita das mesmas, fazendo avançar estes índices consoante a frequência desejada para obter um sinal mais grave ou agudo

O N vai influenciar o número de amostras que vamos ter na saída, quanto maior for o N mais espaçadas estarão as amostras havendo assim uma diminuição da frequência face ao sinal original.

Inversamente, o período vai aumentar face ao sinal de entrada n vezes o valor do N.

Conseguimos também perceber que o sinal será mais grave quanto maior for o N e mais agudo se menor o N.

Figura 17:Sinal de saída para N=2

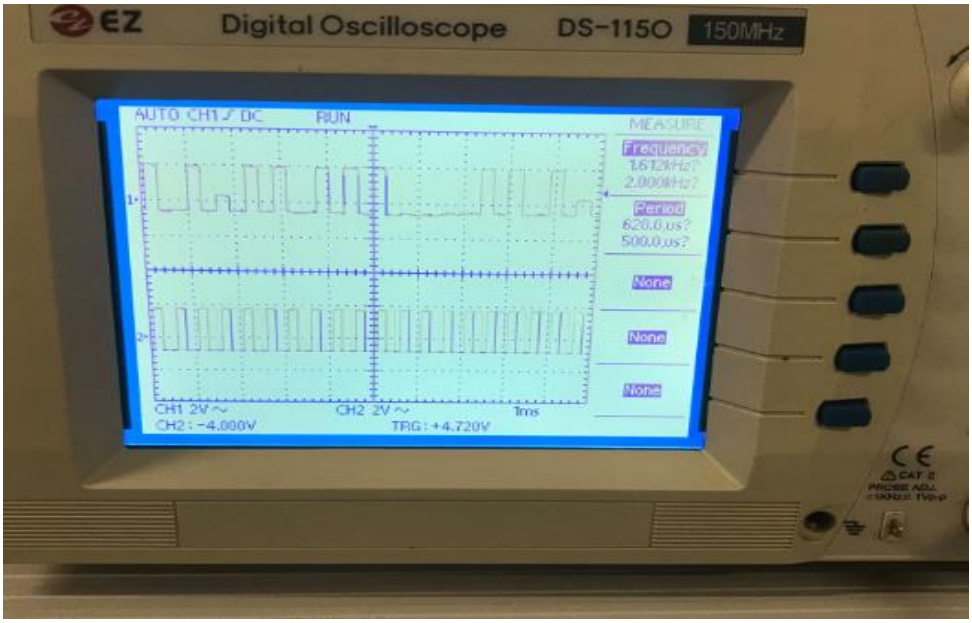


Figura 18: Sinal de saída para N=4

Conclusão

No geral, nos exercícios iniciais vimos como é que a discretização do sinal afetava a sua representação no visor do osciloscópio, tendo um comportamento distinto do sinal contínuo. O sinal discreto possuía “espaços” entre valores de tensão, isto porque não apresentava bits suficientes para representar todos os valores de tensão possíveis.

No exercício 4 verificámos que o código utilizado tem um peso significativo sobre o processamento das amostras o que levou a alguma discrepância nos valores que efetivamente obtivemos para as frequências de corte

A partir dos equipamentos presentes no laboratório conseguimos chegar à conclusão de que a função do exercício 4 se tratava de um filtro passa baixo.

Por fim, o mecanismo utilizado para implementar o disfarce de voz permitiu-nos ter uma melhor perceção do “poder” do Arduíno no que toca a processamento de sinal, sendo capaz de responder na perfeição a algo que é muito utilizado na vida real.

Bibliografia

(Slides de PDS no Moodle) – Introdução Arduino - Utilização em PDS