

Procesamiento de señales. Fundamentos

Reconstrucción

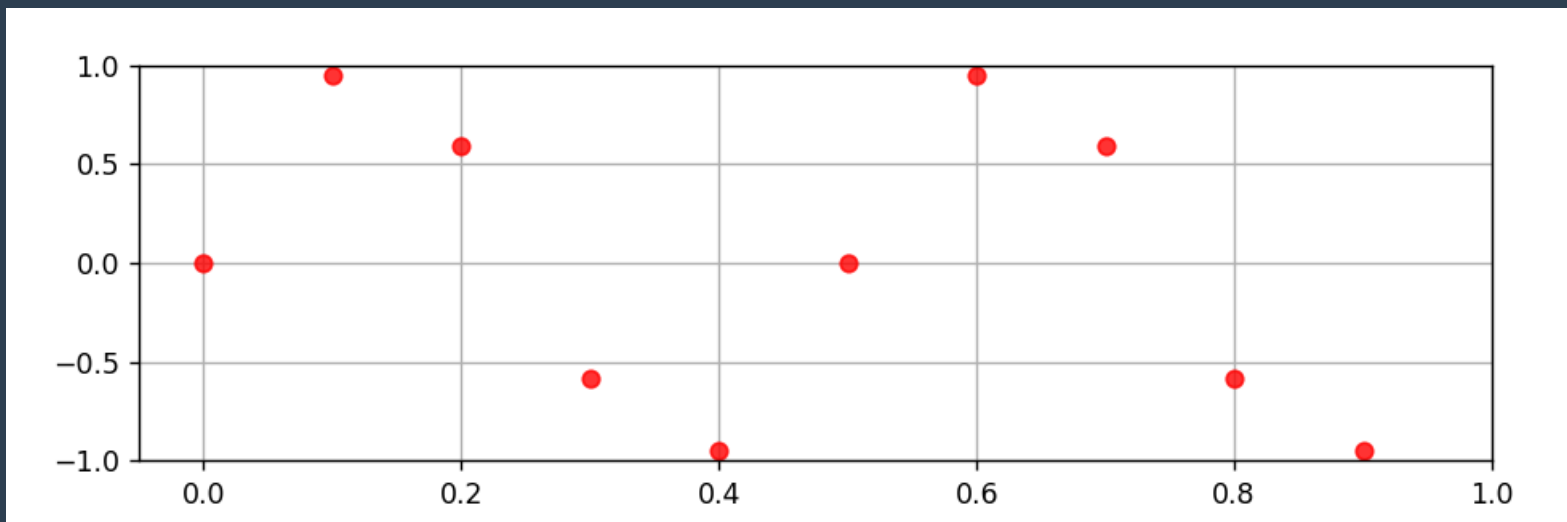
- Interpolación de Shannon
- Efecto ZOH
- Generación y reproducción de audio con Python y pulseaudio
- Números Q y Float
- Utilidades CMSIS-DSP

```
0000 c0ff 0300 80ff 0100 c0ff 0400 4000 a.....@.
0500 0000 faff c0ff 0200 c0ff 0600 0000 .....
0200 c0ff 1100 0000 ffff 80ff 0200 c0ff .....
0000 6865 6164 6572 2000 0000 f2ff 0000 .....header.....
fcff 0000 0600 0000 0000 c0ff f6ff 0000 .....
fcff 4000 f6ff 0000 0d00 4000 feff 0000 .....@.....@.
0400 c0ff 0200 0000 feff 0000 0800 c0ff .....
ffff 0000 0600 c0ff ffff 0000 fdff 0000 .....
0000 6865 6164 6572 2000 0000 ecff c0ff .....header.....
e8ff 0000 0100 c0ff 0300 c0ff edff c0ff .....
0c00 0000 f0ff 0000 0100 c0ff 0500 c0ff .....
0200 c0ff 0500 c0ff f6ff 0000 feff 0000 ..@.....
0600 4000 0900 0000 0a00 c0ff f8ff 0000 .....@.....@.....
0000 6865 6164 6572 2000 80ff f4ff 0000 .....header.....
0200 c0ff f9ff c0ff 0400 c0ff 0500 0000 .....
fcff 0000 f2ff 0000 0300 0000 0100 c0ff .....
f5ff c000 fcff 4000 0200 0000 fdff 0000 .....@.....
ffff 0000 f9ff 0000 0400 0000 f7ff 0000 .....
0000 6865 6164 6572 2000 c0ff e0ff 0000 .....header.....
fcff 0000 0a00 c0ff ffff 0000 f1ff c0ff .....
```

Reconstrucción

Filtro reconstructor (FAA) - Pasa bajos

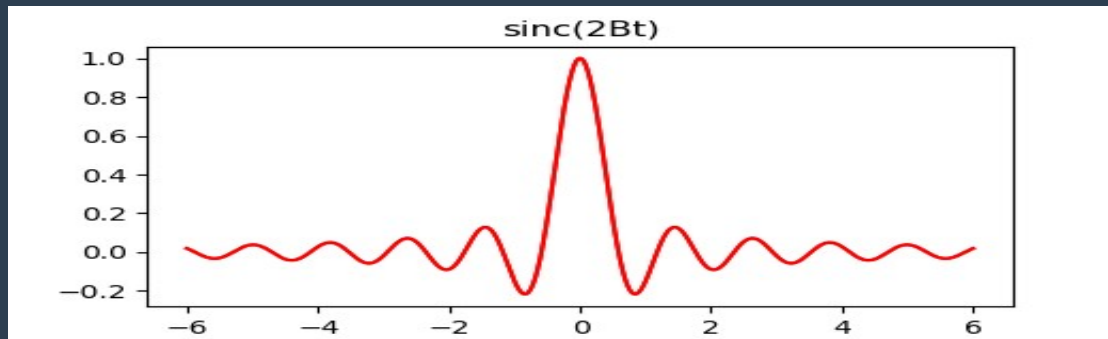
- Consiste en obtener una señal continua a partir de una discreta
- Si es continua => el bloque reconstructor deberá ser analógico
- Habrá que rellenar/interpoliar los infinitos puntos intermedios
- Cuantos puntos como mínimo serian necesarios para una interpolación correcta?
- Cual seria la función de interpolación optima?



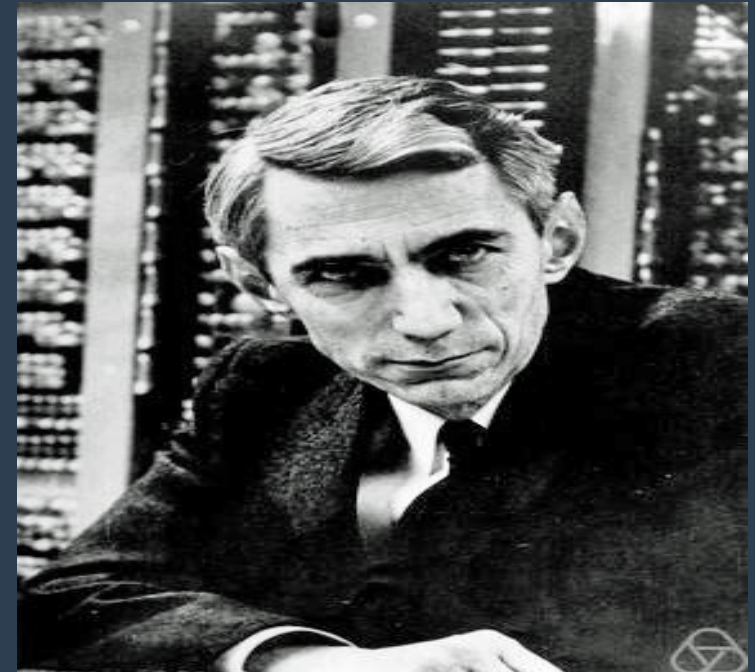
Teorema del sampleo - Shannon-Nyquist

- La reconstrucción exacta de una señal **periódica** continua en banda base a partir de sus muestras, es **matemáticamente** posible si la señal está **limitada** en banda y la tasa de muestreo es **superior al doble** de su ancho de banda

$$x(t) = \sum_{n=-\infty}^{\infty} x_n \frac{\sin \pi(2Bt - n)}{\pi(2Bt - n)}.$$

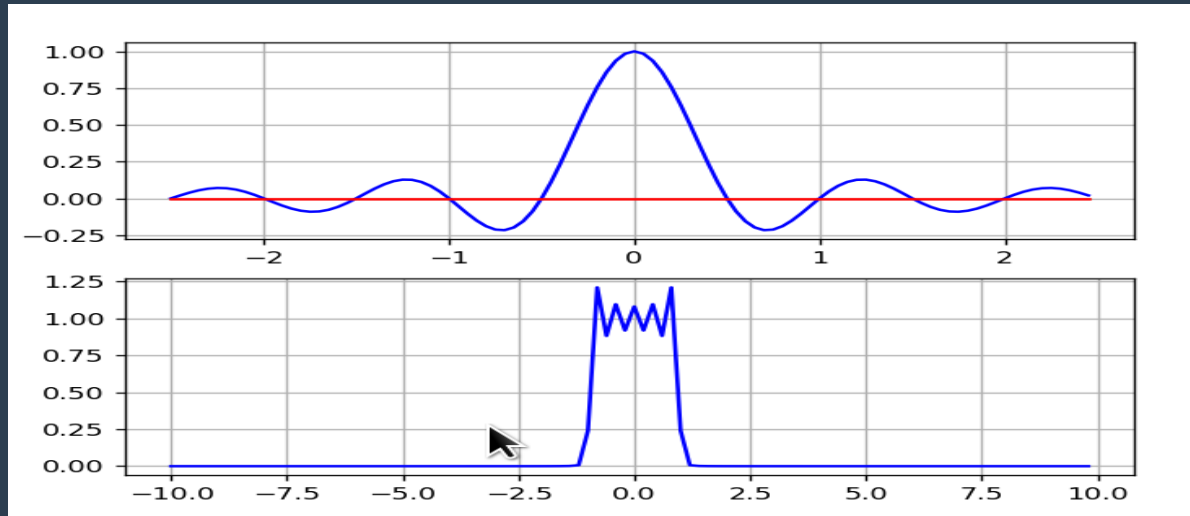


- Ver código `sinc.py`



Teorema del sampleo - Sinc Function

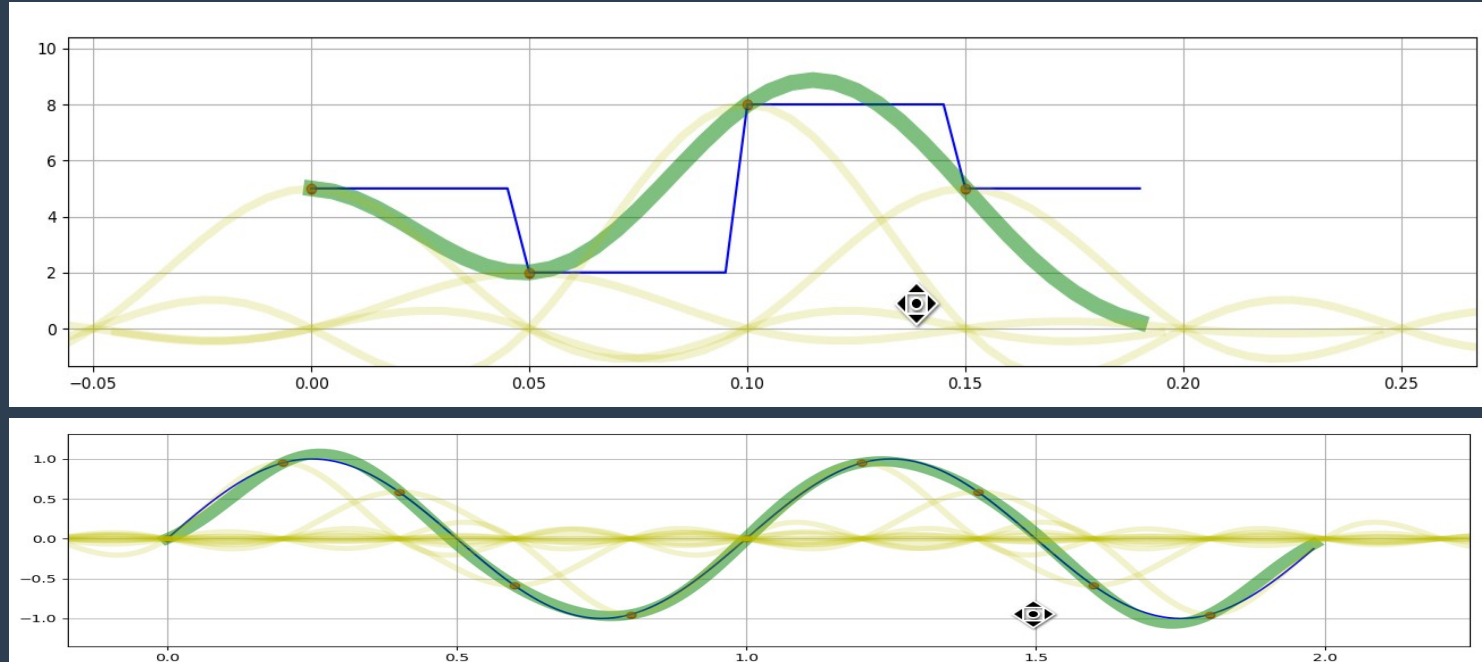
- Notar que los instantes de cruce por cero coinciden con los momentos de sampleo
- Notar que vale 1 justo en $t=0$
- Esta cualidad hace que para el instante de sampleo en 0, mantenga el sample intacto, y entre sample y sample se sumen un proporcional de todos los samples anteriores y posteriores
- Notar que la transformada de la sinc es un rectángulo. Se puede inferir que si la respuesta en frecuencia de un filtro es un rectángulo, estaríamos reconstruyendo la señal de manera óptima



Ver código: `sinc.py`

Teorema del sampleo - Python

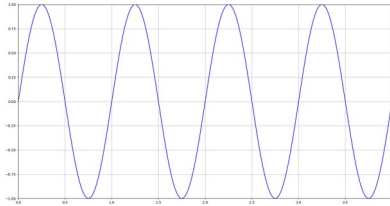
- Reconstrucción utilizando el promedio de funciones Sinc centradas en cada muestra. Teorema de Shannon
- Probar diferentes escenarios cambiando la cantidad de samples



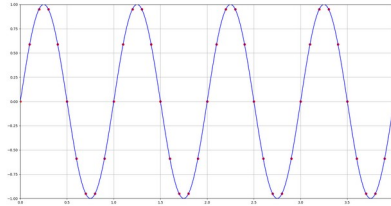
Ver código:
[reconstruccion.py](#)

Efecto de aliasing en Python

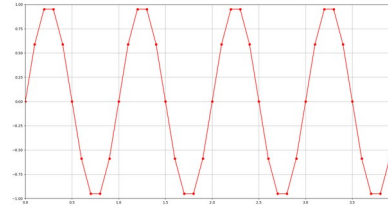
Azul: 1hz $f_s = \infty$



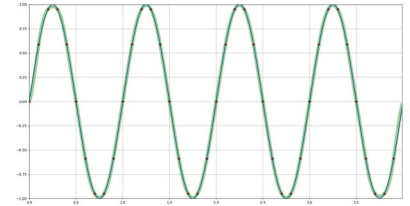
Azul: 1hz $f_s = \infty$
Rojo: samples $f_s = 10$



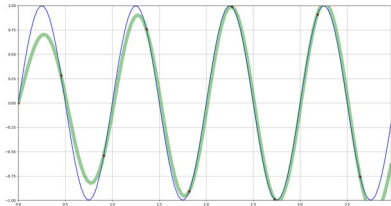
Reconstrucción con
lineas



Reconstrucción con
Shannon



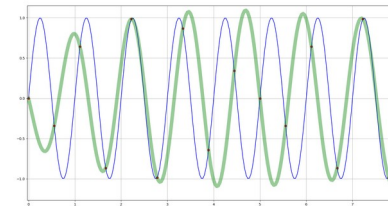
Rojo: samples $f_s = 2.2$
Verde Shannon



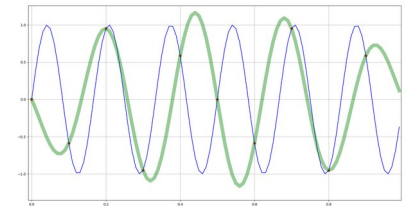
Rojo: samples $f_s = 2.0$
Verde: Shannon



Rojo: samples $f_s = 1.8$
Verde: alias $f = -0.8$



Azul: $f = 6$ $f_s = 10$
Verde: $f = -4$



Se puede distinguir una señal de 1Hz sampleada a $f_s = 1.8$ de una señal de -0.8Hz sampleada a la misma f_s ? Porque? Como se resuelve?

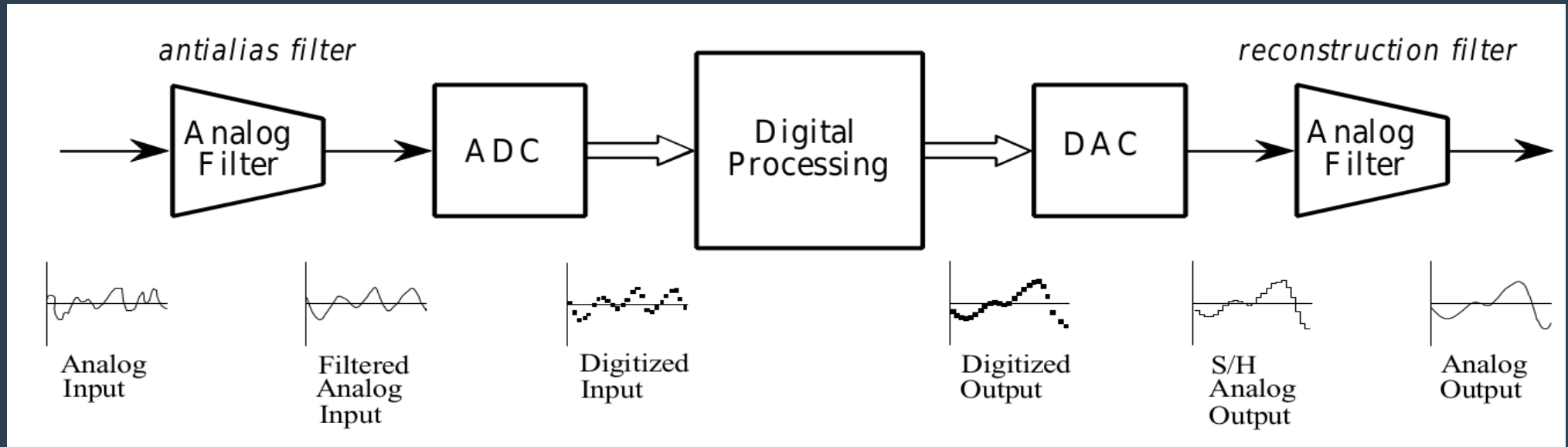
Auxiliar $f = -0.8$ $f_s = 10$



Ver código:
alias_interpolacion.py

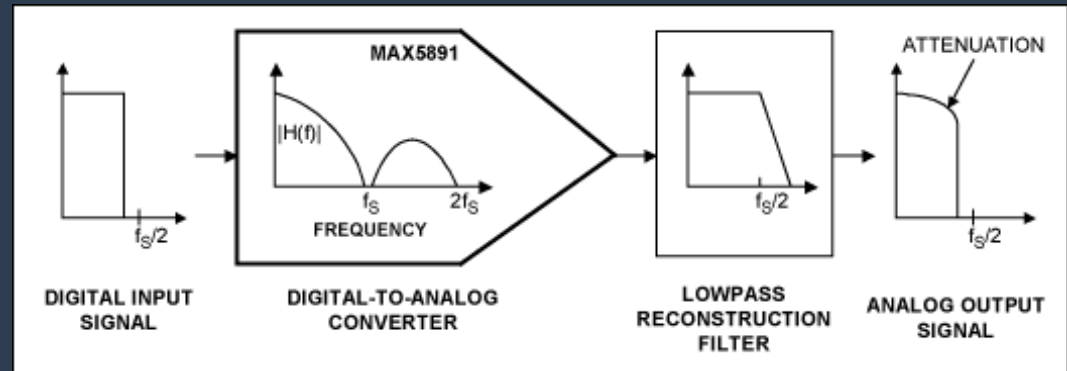
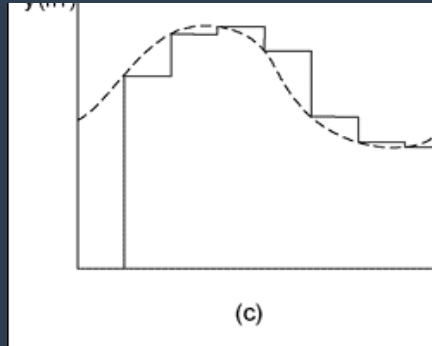
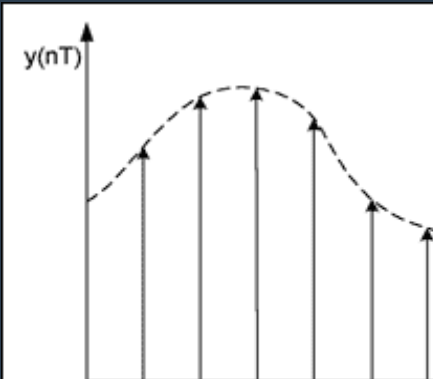
Digitización - Secuencia completa

- Secuencia de digitización, procesamiento y reconstrucción.
- Se agrega el bloque (filtro) anti alias.
- Se agrega el bloque (filtro) de reconstrucción
- Falta algo mas??



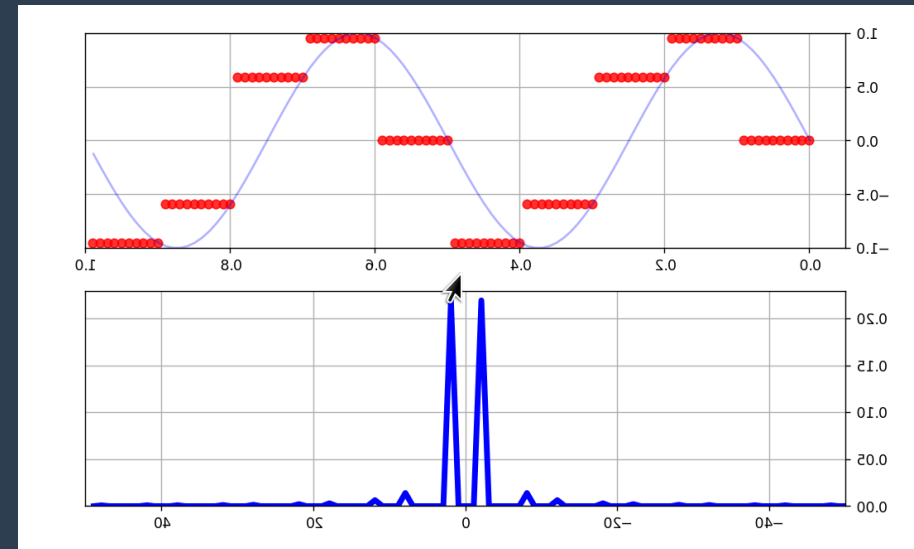
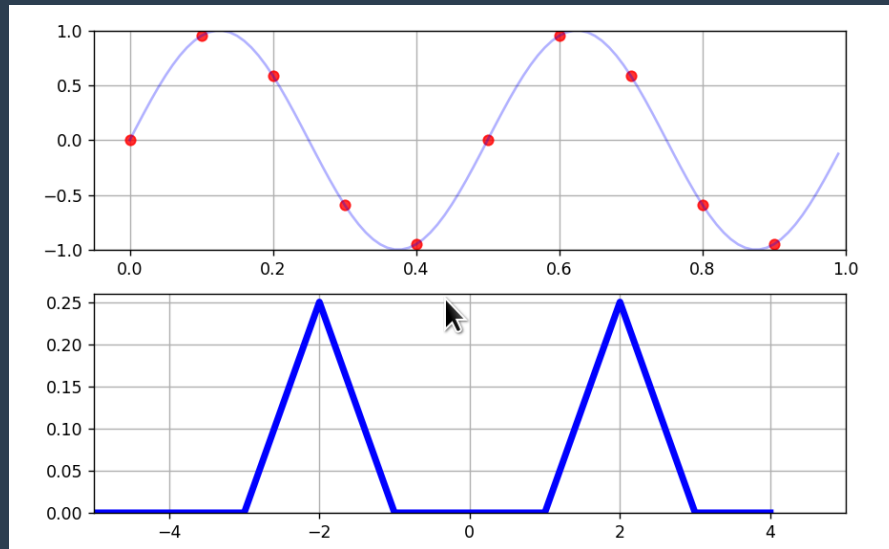
Reconstrucción - Respuesta del DAC - ZOH

- En el análisis teórico la salida del DAC debería ser un pulso de duración 0 para cada muestra.
- En general el DAC no genera pulsos de corta duración sino que mantienen un valor constante por un lapso de tiempo igual a $1/f_s$
- Esta diferencia se traduce en una atenuación de la salida para frecuencias cercanas a $f_s/2$



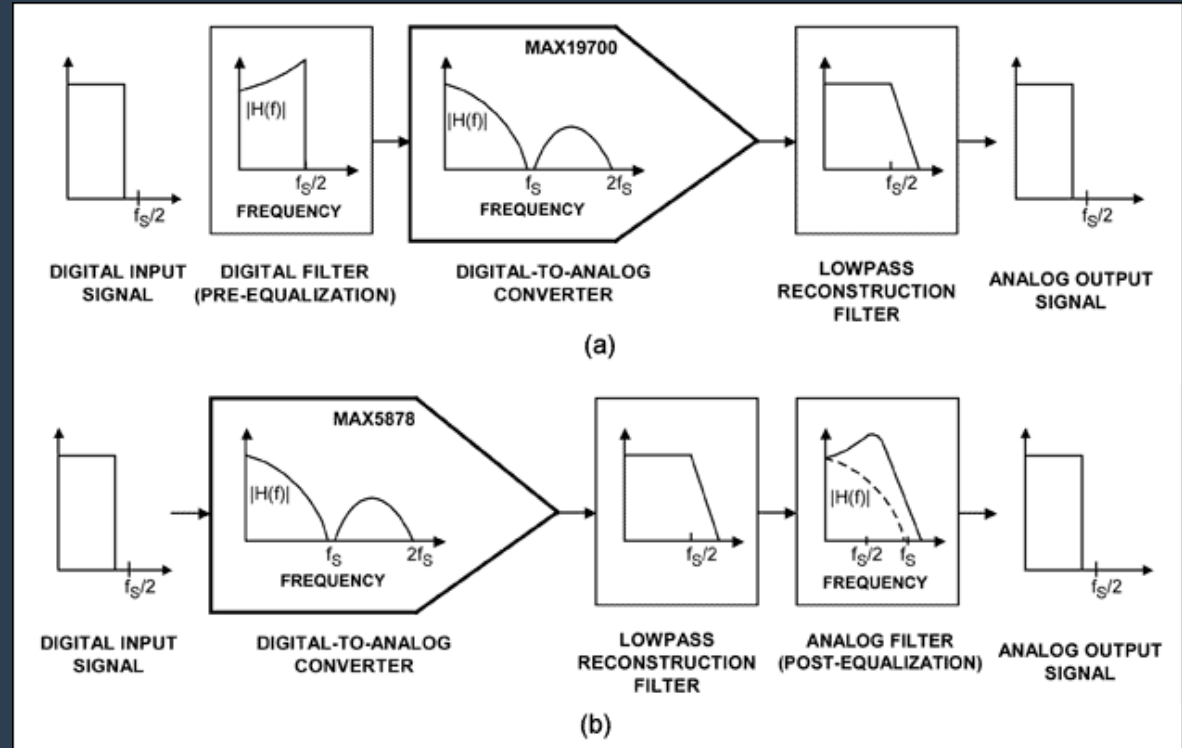
Reconstrucción - Respuesta del DAC - ZOH

- Se puede ver que el espectro de la señal con ZOH tiene componentes de mayor frecuencia que lo esperado
- Por otro lado la energía de la componente principal se ve disminuida con respecto al grafico de la izquierda
- Significa que para reconstruir la salida de un ZOH no alcanza con un filtro reconstructor optimo, habra que amplificar las componentes de frecuencia cercanas a f_s



Reconstrucción - Mitigacion efecto ZOH

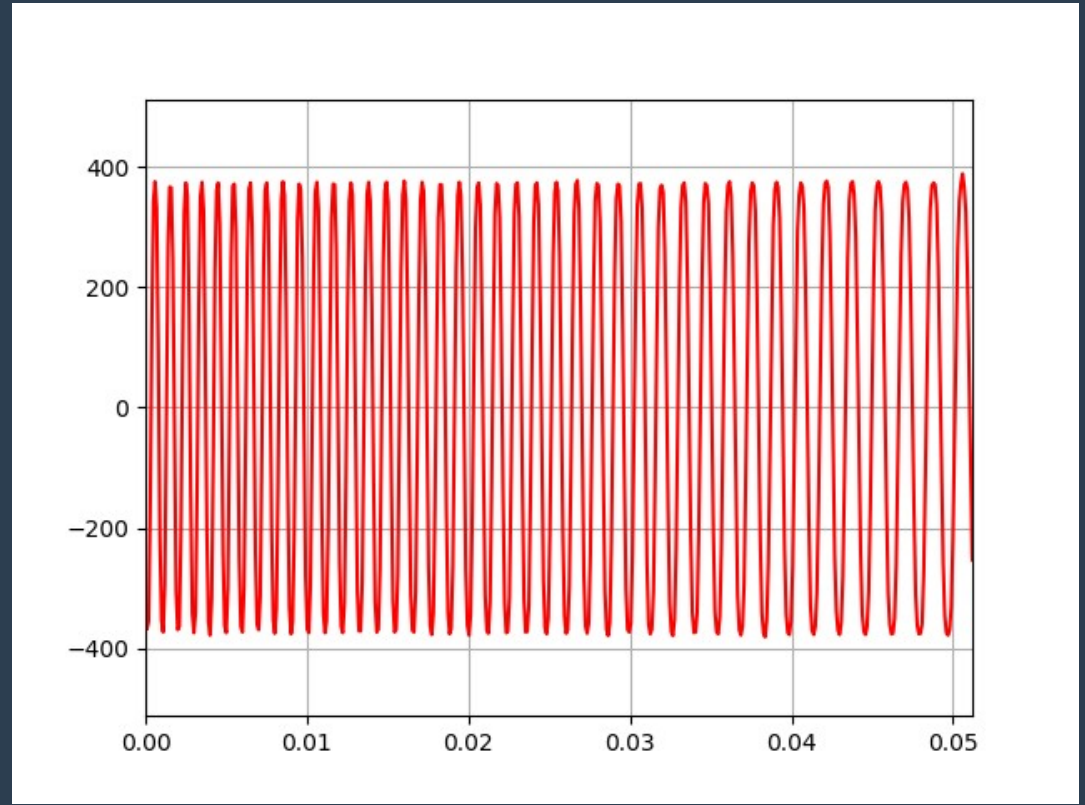
- Se puede mitigar el efecto de la atenuación con diferentes técnicas
 - Pre-equalization
 - Post-equalization
 - Interpolación agregando datos en el DAC
 - Aumentando F_s



Generación

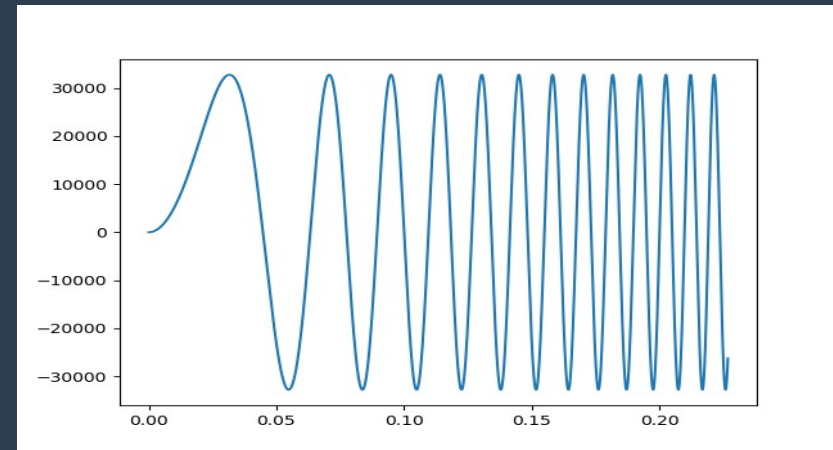
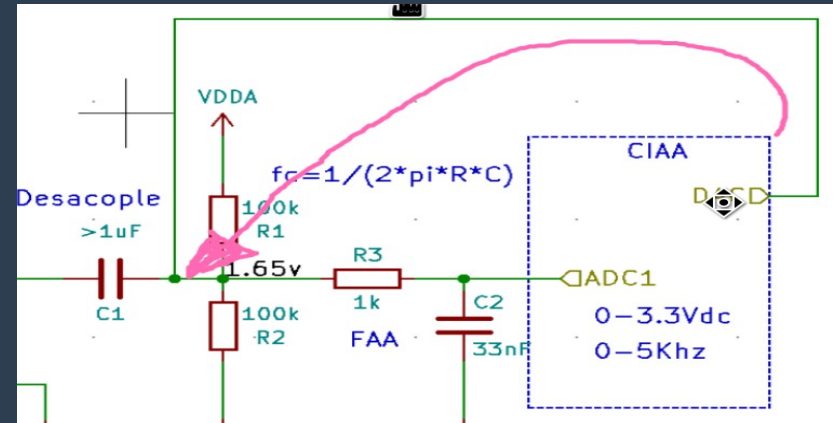
Generación de sonidos con Python

- Se puede utilizar el modulo pulseaudio para generar señales de prueba
- Install:
<https://pypi.org/project/simpleaudio/>
- Ejemplos en
2_clase/simpleaudio_tutorial.pdf
- Se muestran ejemplos de generación de senoidales, cuadradas, sweeps etc.
- También se puede reproducir directamente audio desde el ordenador para su análisis
- Ver código: audio_gen.py



Generación de sonido con CIAA

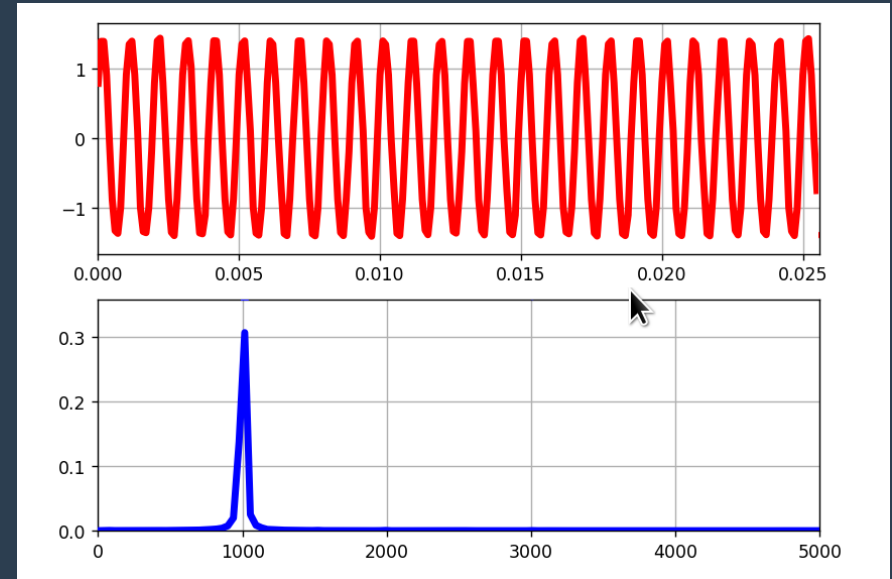
- Se puede utilizar el loop que conecta la salida del DAC con la entrada del ADC
- Se pueden utilizar las primitivas de CMSIS-DSP para sintetizar señales
- Se continua utilizando el ADC para samplear y enviar por UART



Ver código: [ciaa/psf1/psf.c](#)

Generación de sonido con CIAA

- Se puede generar una senial senoidal y a partir de alli un tono, un acorde o una sweepts
- Al tiempo que se genera, tambien se adquiere, esto es particularmente util para calibrar el ADC y validar los datos



```
dacWrite( DAC, D0m(t)); // acorde  
dacWrite( DAC, 512*arm_sin_f32 (t*B/2*(t/sweep)*2*PI)+512); // sweep  
dacWrite( DAC, 512*arm_sin_f32 (t*tone*2*PI)+512); // tono
```

Ver código: [ciaa/psf1/psf.c](https://github.com/ciaa/psf1/psf.c)

Reproducción

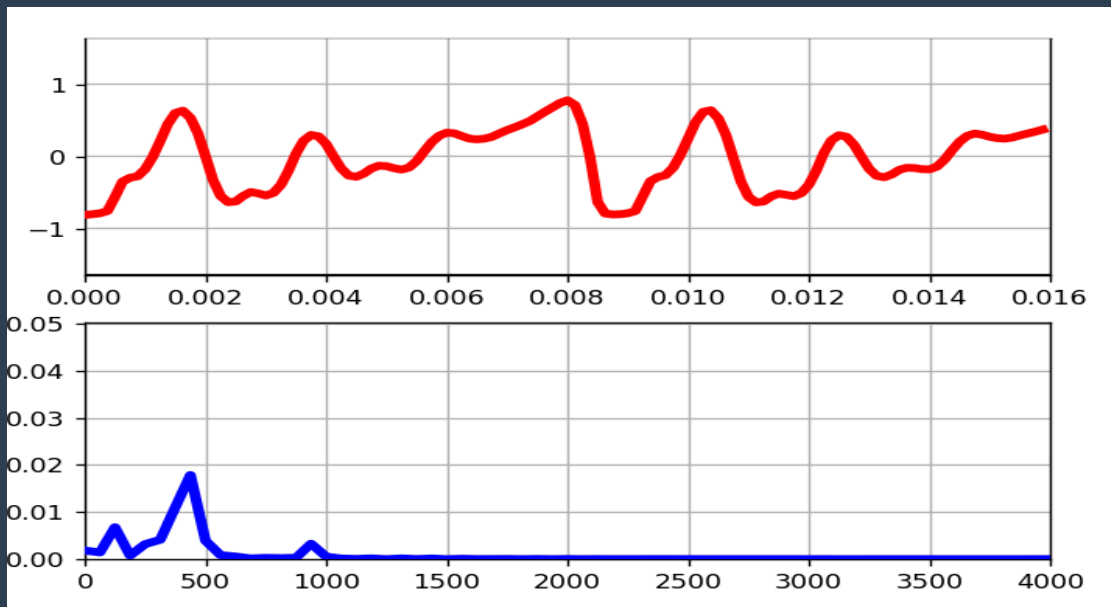
Proyecto repetidor de sonido | Lo-Ro-Lo

- Se pueden utilizar los datos capturados con la ciao y reproducirlos con simpleaudio
- En la mayoría de las tarjetas de audio convencionales solo soportan 8k, 22050hz y 44100hz como fs. En el caso de la ciao, el mas cercano posible es 8k
- Se reproducen chunks de datos mientras se captura el resto

Ver códigos:
[ciaa/psf3/src/psf.c](#)
[ciaa/psf3/visualize.py](#)

```
rec=np.concatenate((rec,((adc/1.65)*2**((15-1)).astype(np.int16)))  
return adcLn, fftLn
```

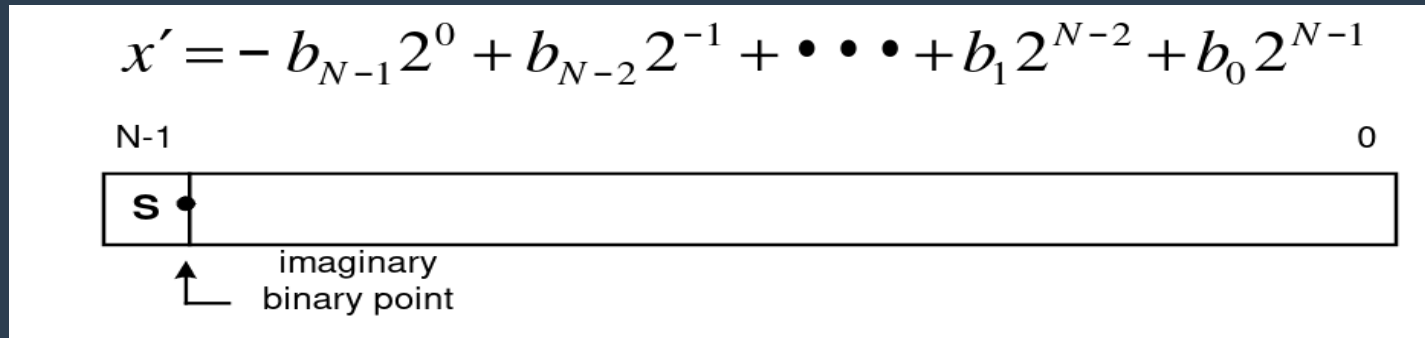
```
play_obj = sa.play_buffer(rec, 1, 2, 8000)  
print('playing')
```



Números Q

Sistema de números Q

- Porque no almacenar los números en formato entero complemento a 2?
- Se puede, pero en general es conveniente darle sentido a los datos, y para ello se necesita escalar los datos para que representen los parámetros físicos
- Por lo general es muy usual usar valores entre -1 y 1, dado que es fácil de interpretar
- En ese caso se puede imaginar un punto fraccional en el bit 15.
- Esta interpretación sería un numero de punto fijo o Q1.15 como se ve en la figura



Sistema de números Q

- Qm.n:

- m: cantidad de bits para la parte entera
- n: cantidad de bits para la parte decimal
- Los rangos para signados $[-(2^{m-1}), 2^{m-1} - 2^{-n}]$
- Los rangos para sin signo $[0, 2^m - 2^{-n}]$
- Resolución constante $1/2^n$

Sistema de números Q

- Tabla de ejemplos Q1.2 y Q2.1 signado (S) y no signado (U)

UQ3.0	UQ2.1	UQ1.2
011 = 3	01.1 = $1+1/2= 1.5$	0.11 = $0+1/2+1/4= 0.75$
010 = 2	01.0 = $1+0/2= 1.0$	0.10 = $0+1/2+0/4= 0.5$
001 = 1	00.1 = $0+1/2= 0.5$	0.01 = $0+0/2+1/4= 0.25$
000 = 0	00.0 = $0+0/2= 0.0$	0.00 = $0+0/2+0/4= 0.0$
111 = 7	11.1 = $3+1/2= 3.5$	1.11 = $1+1/2+1/4= 1.75$
110 = 6	11.0 = $3+0/2= 3.0$	1.10 = $1+1/2+0/4= 1.5$
101 = 5	10.1 = $2+1/2= 2.5$	1.01 = $1+0/2+1/4= 1.25$
100 = 4	10.0 = $2+0/2= 2.0$	1.00 = $1+0/2+0/4= 1.0$

SQ3.0	SQ2.1	SQ1.2
011 = +3	01.1 = $1+1/2=+1.5$	0.11 = $0+1/2+1/4=+0.75$
010 = +2	01.0 = $1+0/2=+1.0$	0.10 = $0+1/2+0/4=+0.5$
001 = +1	00.1 = $0+1/2=+0.5$	0.01 = $0+0/2+1/4=+0.25$
000 = +0	00.0 = $0+0/2=+0.0$	0.00 = $0+0/2+0/4=+0.0$
111 = -1	11.1 = $-1+1/2=-0.5$	1.11 = $-1+1/2+1/4=-0.25$
110 = -2	11.0 = $-1+0/2=-1.0$	1.10 = $-1+1/2+0/4=-0.5$
101 = -3	10.1 = $-2+1/2=-1.5$	1.01 = $-1+0/2+1/4=-0.75$
100 = -4	10.0 = $-2+0/2=-2.0$	1.00 = $-1+0/2+0/4=-1.0$

Sistema de números Q en Python

```
20 from fxpmath import Fxp
19 import numpy as np
18
17 M = 2
16 N = 2
15 SIGNED = True
14
13 if(SIGNED):
12     MIN = -2**(M-1)
11     MAX = 2**(M-1)-1/2**N
10 else:
9     MIN = 0
8     MAX = 2**(M)-1/2**N
7
6 n=np.arange(MIN,MAX+1/(2**N),1/(2**N))
5
4 Q = Fxp(n, signed = SIGNED, n_word = M+N, n_frac = N,rounding = "trunc")
3
2 for i in range(len(n)):
1     print("decimal: {0:.5f} \tbinary: {1:} \thex: {2:}"
21         .format(n[i], Fxp.bin(Q)[i], Fxp.hex(Q)[i]))
1
```

Ver código:
numeros_Q.py

- Se propone el uso de la biblioteca fxpmath para convertir de float a Q
- Se comparte un generador de números Q genérico
- Soporta signados y no signados de cualquier combinacion de m,n

```
Press ENTER or type command to continue
decimal: -1.00000    binary: 100    hex: 0x4
decimal: -0.75000    binary: 101    hex: 0x5
decimal: -0.50000    binary: 110    hex: 0x6
decimal: -0.25000    binary: 111    hex: 0x7
decimal: 0.00000     binary: 000    hex: 0x0
decimal: 0.25000     binary: 001    hex: 0x1
decimal: 0.50000     binary: 010    hex: 0x2
decimal: 0.75000     binary: 011    hex: 0x3
```

```
Press ENTER or type command to continue
```

Números Q – Conversor en línea Float->Q

- <https://chummersone.github.io/qformat.html>
- <https://www.rfwireless-world.com/calculators/floating-vs-fixed-point-converter.html>

Q-Format Settings

Choose your conversion options.

Word size in bits ($m+n$):

Fractional bits (n):

Signed?: ☒

Integer format:

Maximum float value:

Minimum float value:

Convert

Type an integer or float in the boxes below.
Click or tab away to update.

Integer (hexadecimal)

Float:

Fixed-point value:

Representation error:

Representation error (dB):

Floating Point Number (input1):

Q format of fixed point (input2):

CALCULATE

Fixed Point Number (Output):

EXAMPLE:

INPUTS: Floating Point Number = 1.5 ; Q format = 8

OUTPUT: Fixed Point Number = 384

Fixed point to floating point converter

Fixed Point Number (input1):

Q format of fixed point (input2):

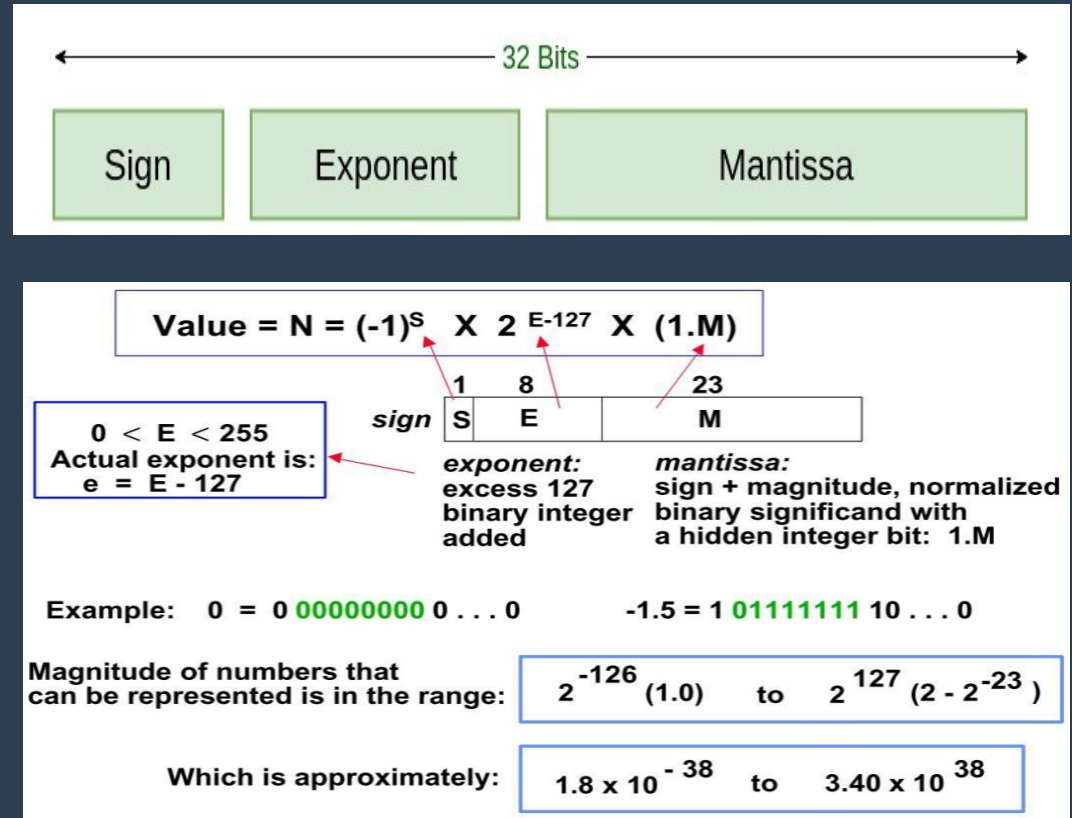
CALCULATE

Floating Point Number (Output):

Números Float32

Números Float32 IDIEE 754

- Se propone el uso de la biblioteca fxpmath para convertir de float a Q
- Se entrega un generador de números Q genérico
- Ver código: números_Q.py



Números Float32 IDEE 754

- Convertidor en linea:
- <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

[illegible]

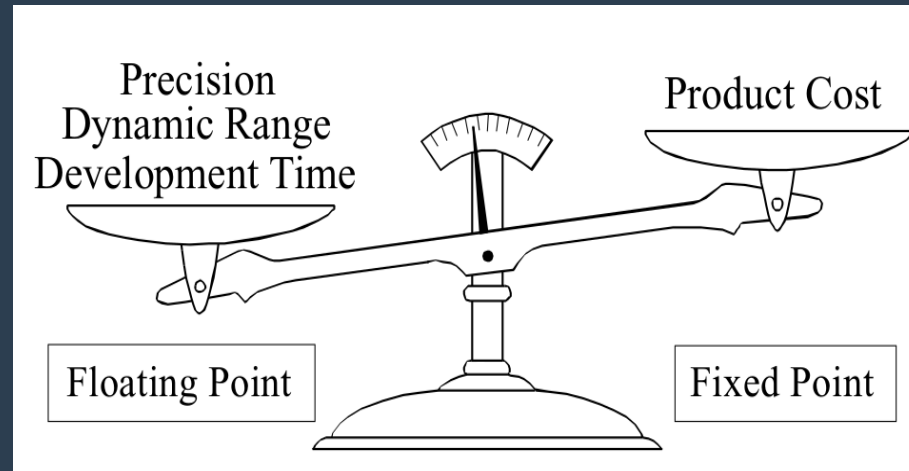
Comparativa Q vs Float

- Float

- Cantidad de patrones de bits = 4,294,967,296
- Gap entre números variable
- Rango dinámico $\pm 3,4 \times 10^{38}$, $\pm 1,2 \times 10^{-38}$
- Gap 10 millones de veces mas chico que el numero

- SQ1,31

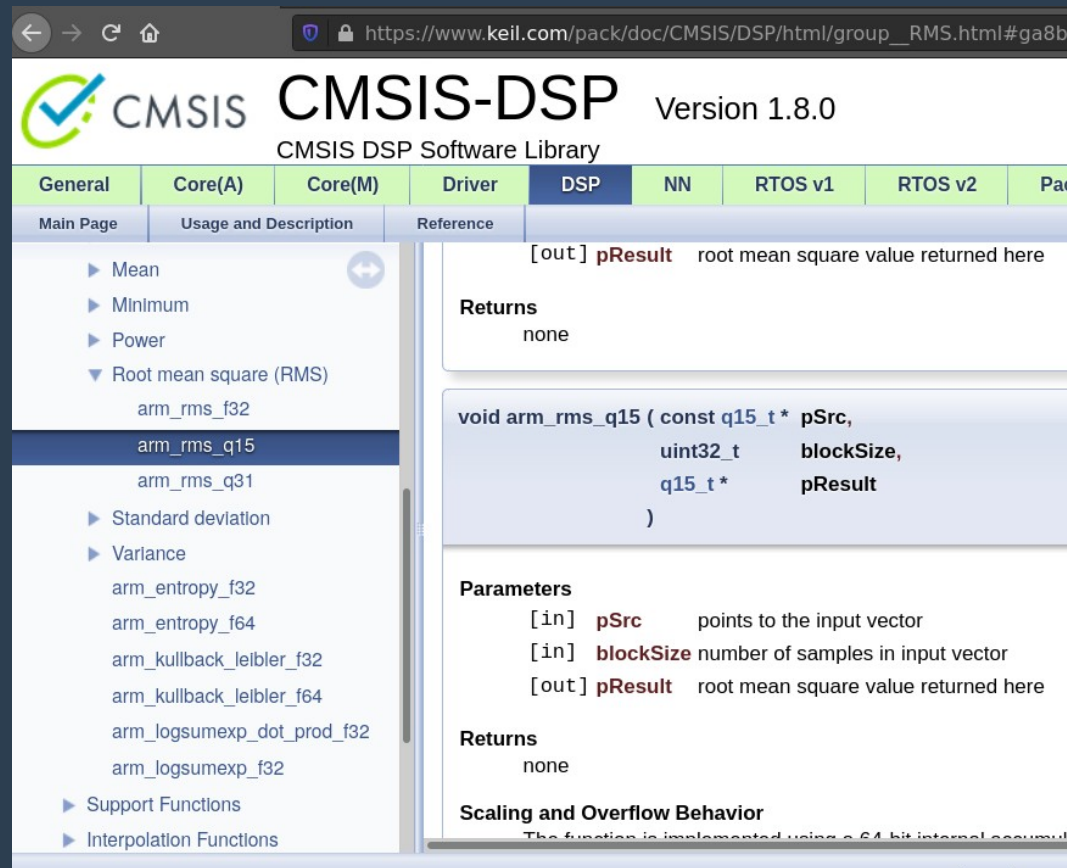
- Cantidad de patrones de bits = 4 294 967 295
- Gap entre números constante = 4.65661×10^{-10}
- Rango dinámico 1(no inclusive) a -1 (inclusive)
- Gap 4mil millones de veces mas chico que el numero mas grande



CMSIS-DSP

CMSIS-DSP – max, min , rms con números Q

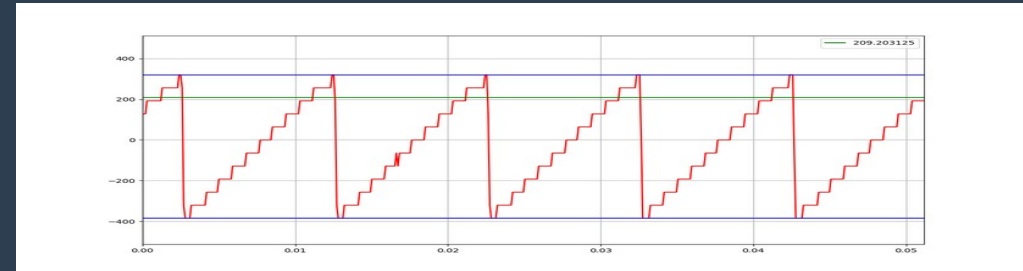
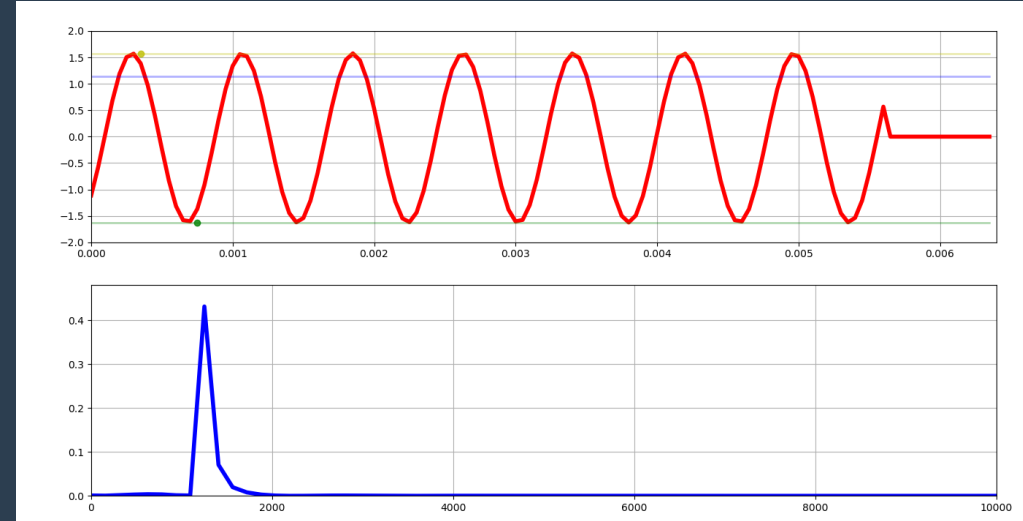
- Investigar la biblioteca CMSIS-DSP
- Implementar y probar algunas de sus funciones utilizando números Q
- https://arm-software.github.io/CMSIS_5/Core/html/index.html



The screenshot shows the CMSIS-DSP documentation page for the `arm_rms_q15` function. The page is titled "CMSIS-DSP Version 1.8.0" and "CMSIS DSP Software Library". The navigation tabs include General, Core(A), Core(M), Driver, DSP, NN, RTOS v1, RTOS v2, and Pa. The left sidebar lists various functions, with "Root mean square (RMS)" expanded, showing `arm_rms_f32`, `arm_rms_q15` (selected), and `arm_rms_q31`. The main content area for `arm_rms_q15` shows the function signature: `void arm_rms_q15 (const q15_t * pSrc, uint32_t blockSize, q15_t * pResult)`. It also includes the "Returns" section (none) and the "Parameters" section: `[in] pSrc` points to the input vector, `[in] blockSize` number of samples in input vector, and `[out] pResult` root mean square value returned here. The "Scaling and Overflow Behavior" section is partially visible at the bottom.

CMSIS-DSP – max, min , rms con números Q

- Se propone como ejemplo de uso de la biblioteca CMSIS-DSP y el formato de numeración Q1.15 el cálculo del máximo, mínimo y rms de una señal
- Ver código
 - `ciaa/psf2/psf.c`
- Se grafican los datos, ver código
 - `ciaa/psf2/visualize.py`



Sistema de números Q con CMSIS

```
3 uint16_t printQ15(q15_t n, char *buf)
4 {
5     int i;
6     float ans=(n&0x8000)?-1:0;
7     for(i=1;i<16;i++)
8     {
9         if(n&(0x8000>>i)){
10             ans+=1.0/(1U<<i);
11         }
12     }
13     return sprintf(buf,"q15: %i float:%.20f\r\n",n,ans);
14 }
15
16 q15_t printSqrtQ15(q15_t n, char *buf)
17 {
18     q15_t b;
19     arm_sqrt_q15(n,&b);
20     return printQ15(b,buf);
21 }
```

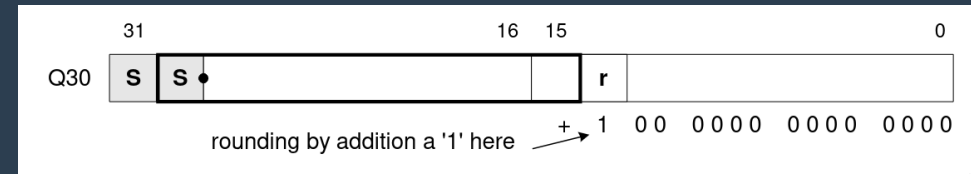
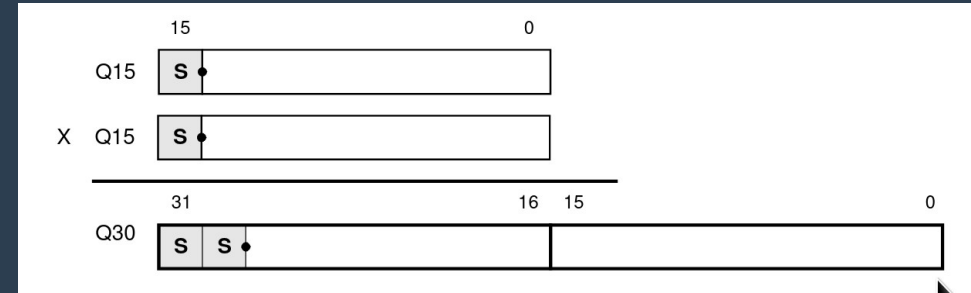
```
q15: -30637 float:-0.93496704101562500000
q15: 23172 float:0.70715332031250000000
q15: -30636 float:-0.93493652343750000000
q15: 23172 float:0.70715332031250000000
q15: -30635 float:-0.93490600585937500000
q15: 23172 float:0.70715332031250000000
q15: -30634 float:-0.93487548828125000000
q15: 23172 float:0.70715332031250000000
q15: -30633 float:-0.93484497070312500000
```

- Se propone el uso de la biblioteca CMSIS de ARM para trabajar con números Q
- Los mas populares son Q15, Q32 y Q64
- Se muestra como ejemplo una funcion para imprimir números Q como float
- Se muestra un ejemplo de como calcular la raiz cuadrada con CMSIS en Q15

Ver código:
[ciaa/psf4/src/psf.c](https://github.com/ARM-software/ciaa/psf4/src/psf.c)

Multiplicación de números Q

- Notar que la multiplicación de dos números Q1.15 genera un resultado Q2.30.
- En general la multiplicación de 2 números de n bits requieren $2*n$ bits para su representación
- Se deberá optar por alguna política de redondeo, truncamiento o cambio de tipo de numeración



```
4 q15_t multiQ15(q15_t a,q15_t b)
5 {
6     q31_t ans;
7     ans=a*b;
8     ans<<=1;
9     return ans>>16;
10 }
```