

Procesamiento de señales. Fundamentos

Clase 2 – CIAA<>Python

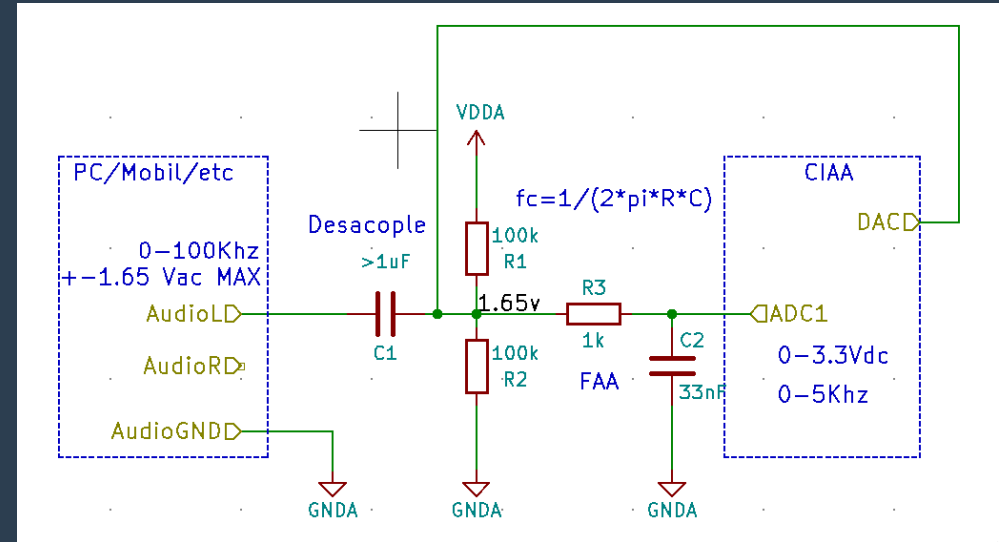
- Bring-up del circuito adquisidor
- Generación de señales con Python
- Generación de señales con DAC
- Adquisición y transmisión
- Captura y visualización en Python
- Ancho de banda del canal
- Calculo del FAA
- Números Q y Float
- Utilidades CMSIS-DSP

```
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 .....
```

Hardware

Acondicionamiento - Consideraciones

- Atención de no sobrepasar $\pm 1.65\text{V}$ en la señal de entrada.
- Se puede usar el canal AudioR para conectar un parlante de monitoreo
- Se puede agregar un jumper para deshabilitar el C2 y validar el comportamiento del FAA
- Se puede agregar por comodidad un jumper al DAC



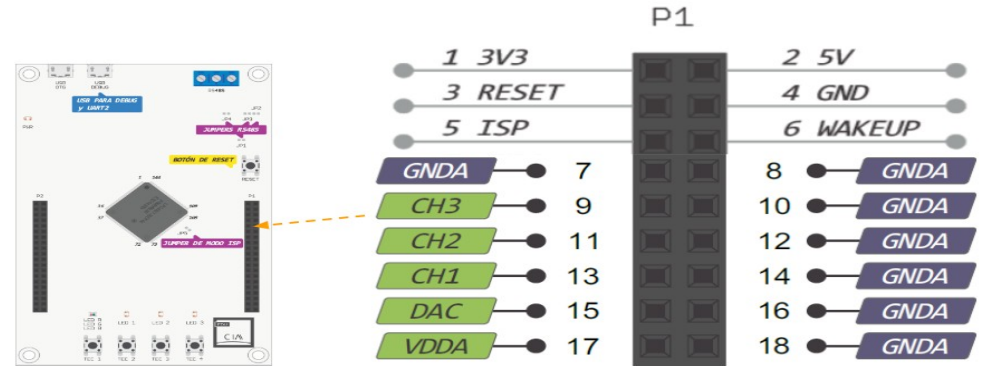
Conexión a CIAA

- Estudiar la hoja de datos para validar los límites del ADC y del DAC
- Analizar Fmax y rangos

CIAA ADC y DAC en la EDU-CIAA-NXP

Mapeo de ADC y DAC en la biblioteca sAPI:

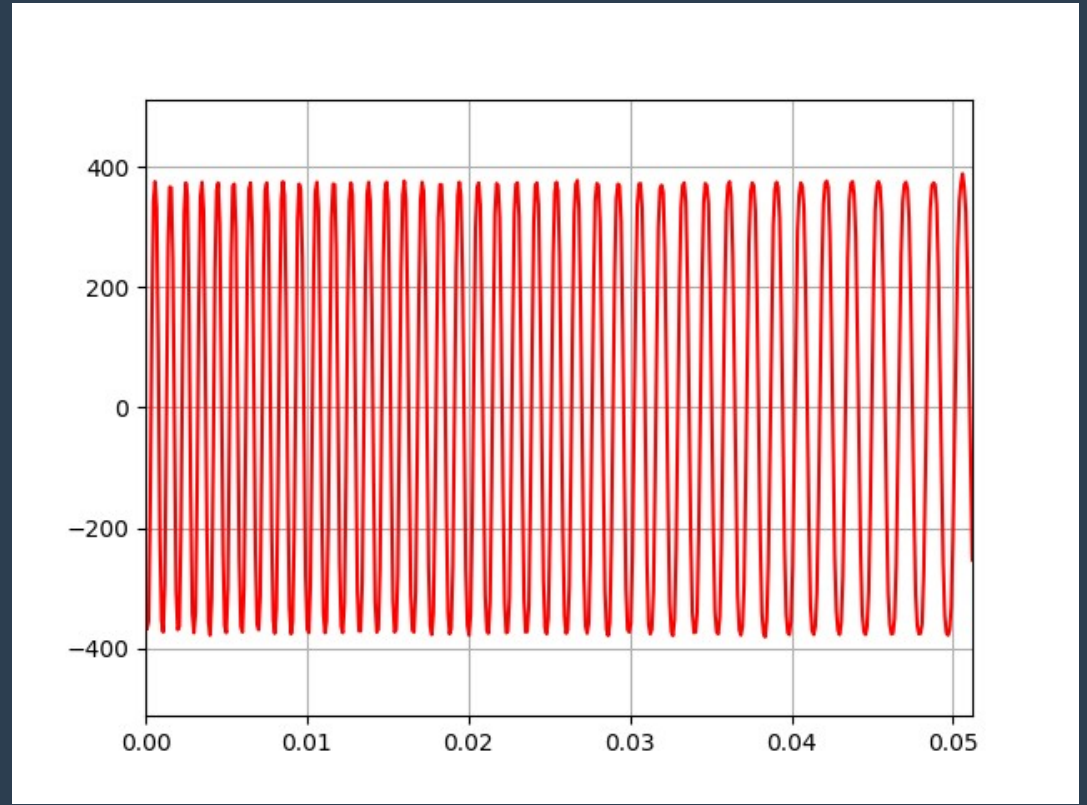
- 3 entradas analógicas nombradas CH1, CH2 y CH3 (ADC).
- 1 salida analógica nombrada DAC.



Generación

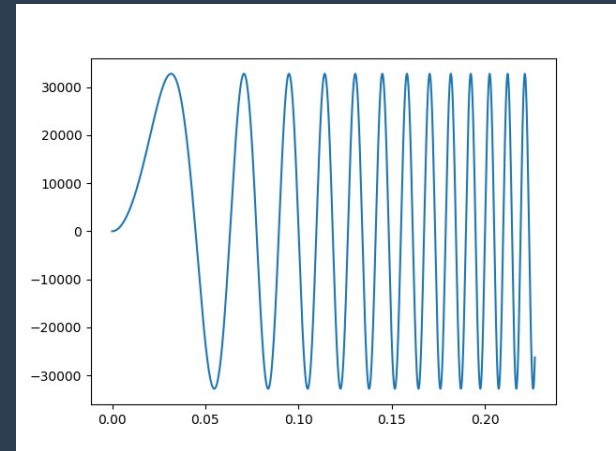
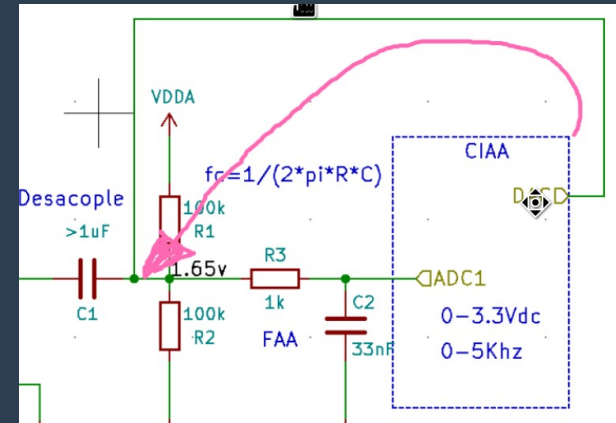
Generación de sonido con Python

- 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, 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/psf2/psf.c`



Visualización

Superloop de adquisición

- Se propone capturar con el ADC
- Enviar los datos por la UART
- Visualizar con matplotlib
- Se envía un header y luego los samples uno a uno en int16_t
- Ver código: psf.c

```
struct header_struct {  
    char mark[8];  
    uint32_t id;  
    uint16_t length;  
    uint16_t fs ;  
} header={"*header*",0,256,20000};
```

```
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 .....
```

0x1234 (dos bytes por sample)

header

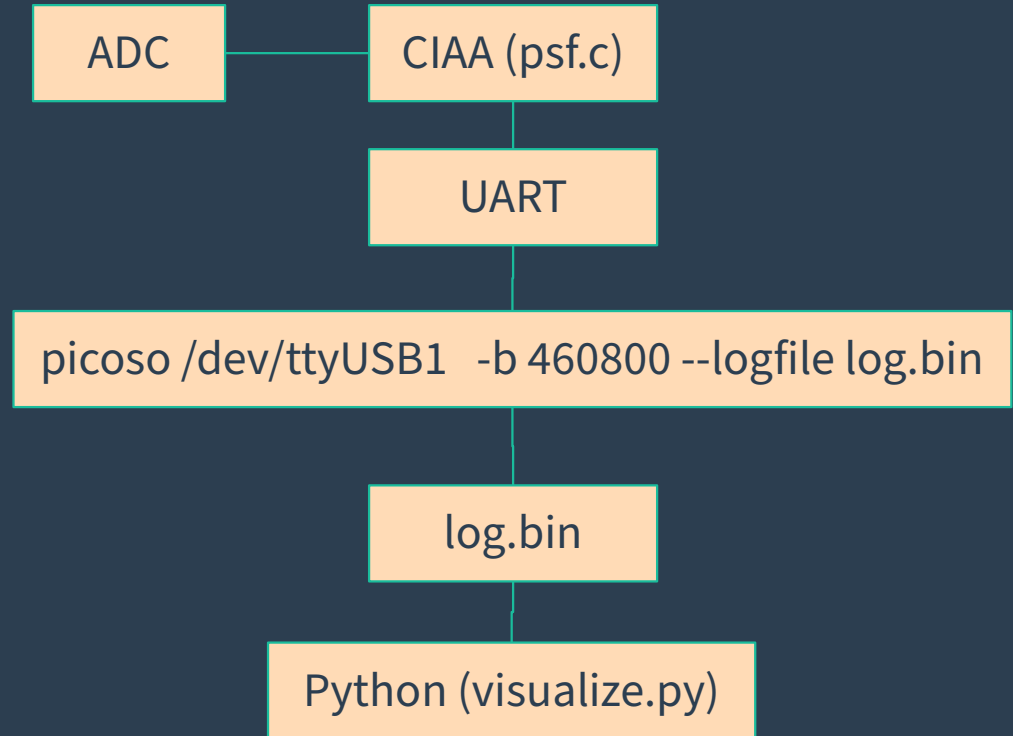
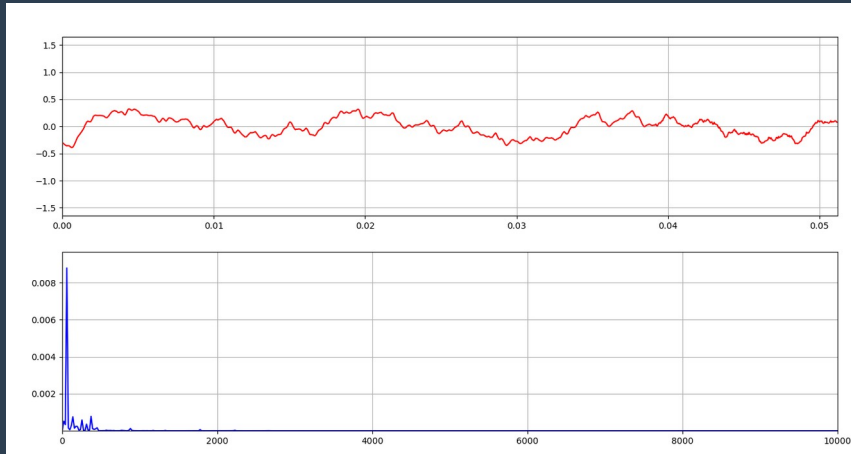
sample

sample

sample

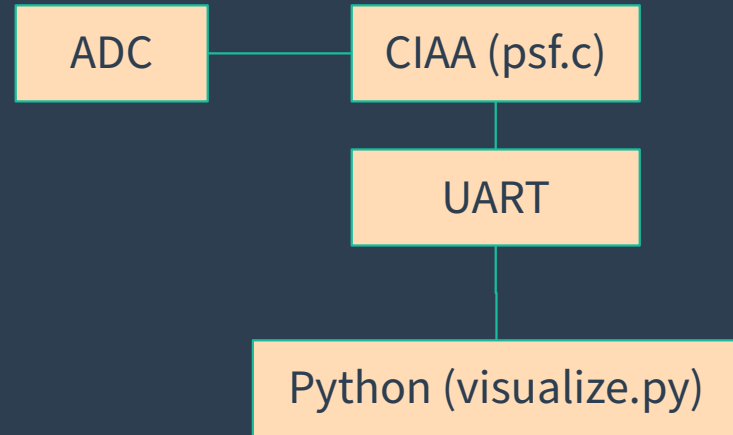
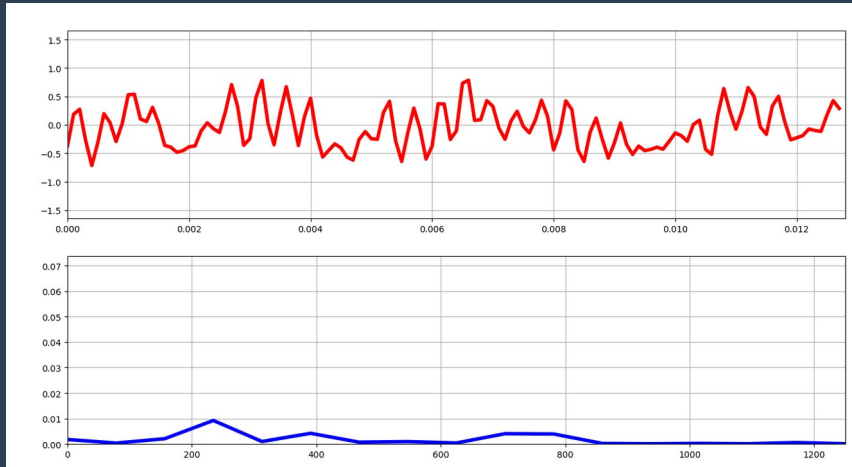
Envío por UART y grabación a archivo

- Se propone utilizar picoso v3.1 que permite reenviar los datos a un archivo
- Luego se lee el archivo desde Python
- Se grafica con matplotlib
- Tiene la ventaja de poder volver a procesar los datos offline.
- En Linux demostró no perder ningún byte



Envío por UART y recepción con pyserial

- Se propone leer directamente los datos utilizando la biblioteca pyserial.
- Se procesan los datos y se grafican con matplotlib
- En ciertos casos se encontró que agrega bytes en cero. Se mitiga el problema con flush, pero se pierden tramas.



Ancho de banda máximo por UART@460800bps

- Se calcula el ancho de banda máximo para transmitir datos desde la CIAA a la PC. Se determino empíricamente que el baudrate máximo que soporta la CIAA es de 460800bps limitado por el conversor USB.
- En función de esa tasa se podrá predecir cual sera la capacidad del sistema de visualización en tiempo real, sin perdida de tramas.
- Sin embargo es posible capturar datos a mayor velocidad si se utiliza otro método de visualización, se acepta el descarte de tramas, compresión, menos bits por muestra o almacenamiento

$$USB \leftrightarrow UART_{maxbps} = 460800bps$$

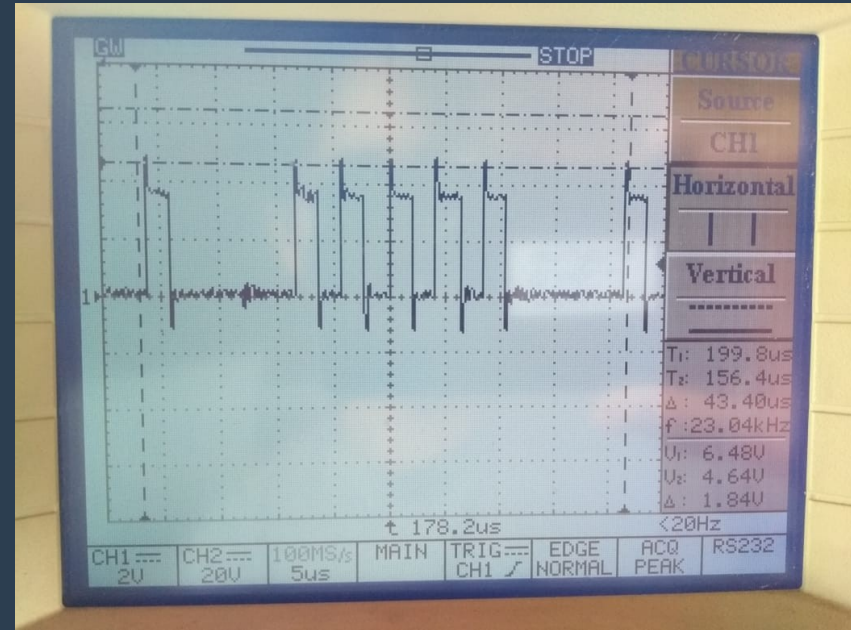
$$Eficacia = \frac{10b}{8b} = 0,8$$

$$bits_{muestra} = 16$$

$$Tasa_{efectiva} = \frac{460800_{bps} * 0,8}{16} = 23040$$

Máxima señal muestreable y reconstruible

11520hz



Calculo del FAA @Fs=20K y B=10K

- Considerando $F_s=20\text{KHz}$ entonces se espera un ancho de banda máximo de 10KHz
- Se muestra el calculo de un filtro RC de 1er orden para mitigar el efecto de aliasing mayores a 10K .
- Sin embargo, dado que la pendiente de atenuación del filtro no es muy abrupta, se puede optar por aumentar la F_s o reducir B para minimizar el efecto del aliasing.
- Otra alternativa es aumentar el orden del filtro

Calculo del filtro antialias 1er orden R-C

$$B = 10\text{kbps}$$

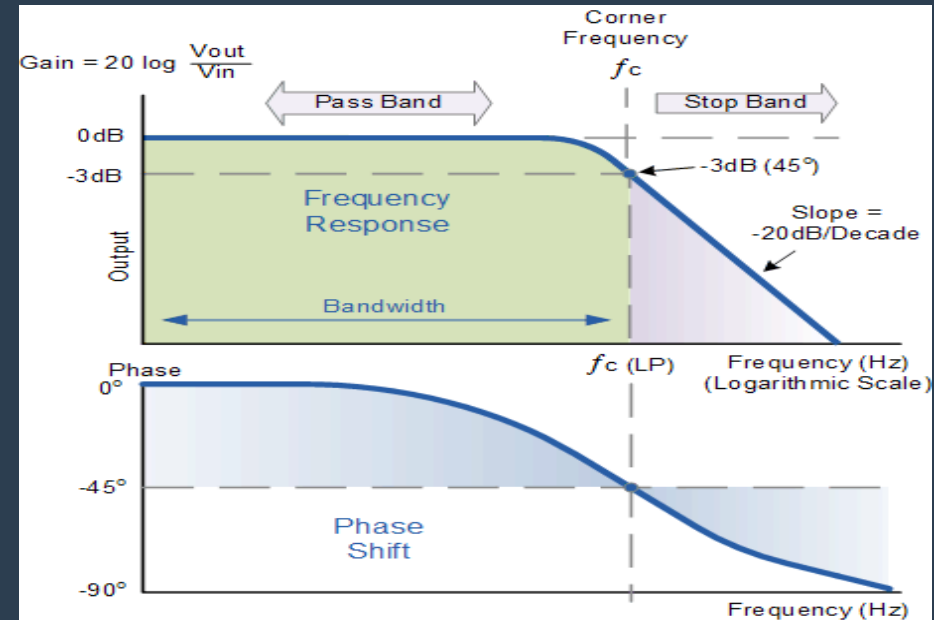
$$f_{\text{corte}} = \frac{1}{2 * \pi * R * C}$$

$$R = 1\text{k}\Omega$$

$$C = \frac{1}{f_{\text{corte}} * R * 2 * \pi} \approx 15\text{nF}$$

Máxima señal muestreable y reconstruible

11520hz

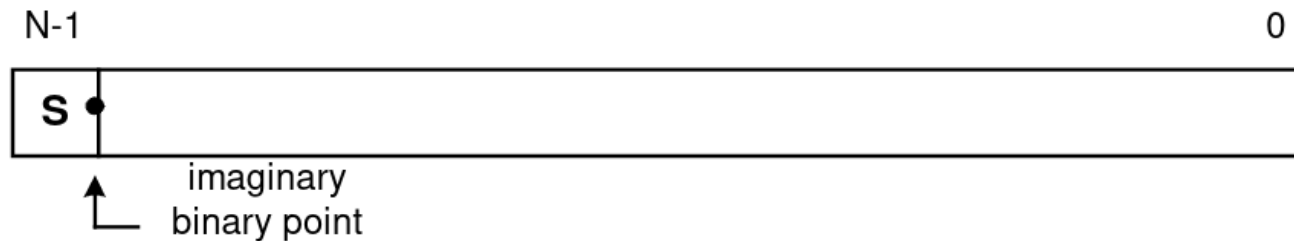


Números Q

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 número de punto fijo o Q1.15 como se ve en la figura

$$x' = -b_{N-1}2^0 + b_{N-2}2^{-1} + \dots + b_12^{N-2} + b_02^{N-1}$$



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

- 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

```
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
```

```
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):

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

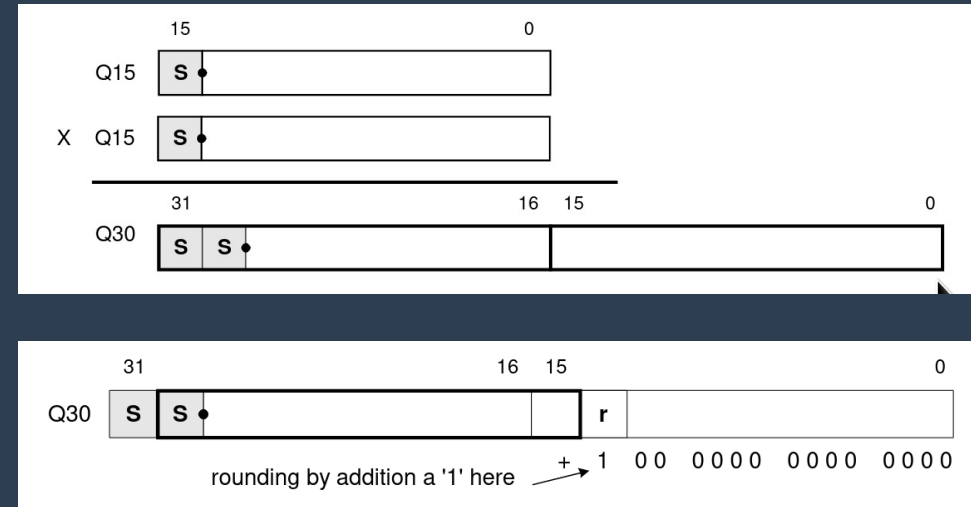
Fixed Point Number (input1):

Q format of fixed point (input2):

Floating Point Number (Output):

Multiplicación de números Q

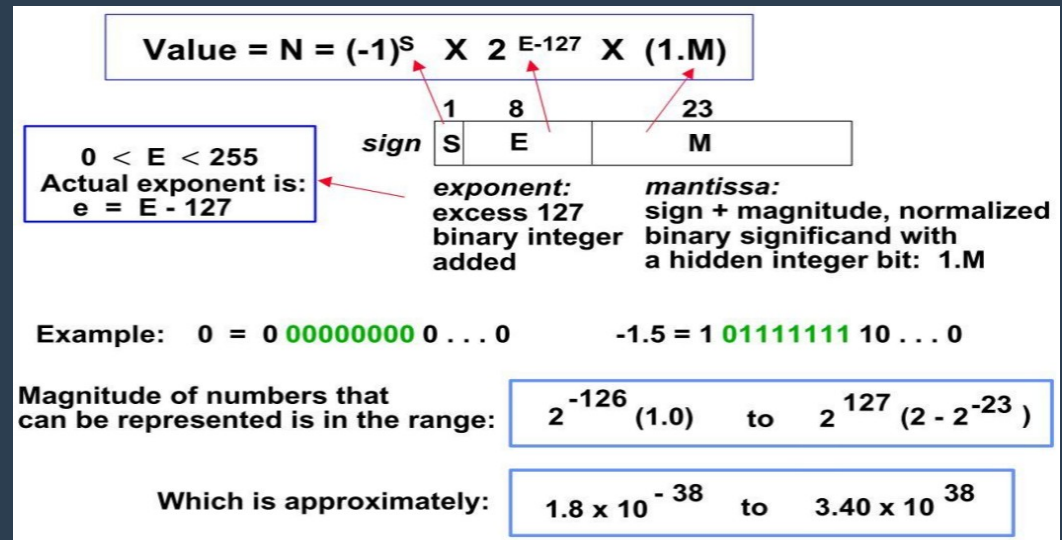
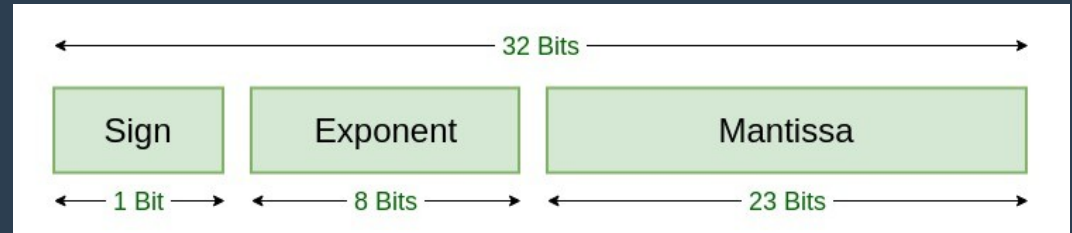
- Notar que la multiplicación de dos números Q1.15 genera un resultado Q2.30.
- En general la multiplicación de 2 números de n bits requieren $2 \times n$ bits para su representación.
- Se debería optar por alguna política de redondeo, truncamiento o cambio de tipo de numeración



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]

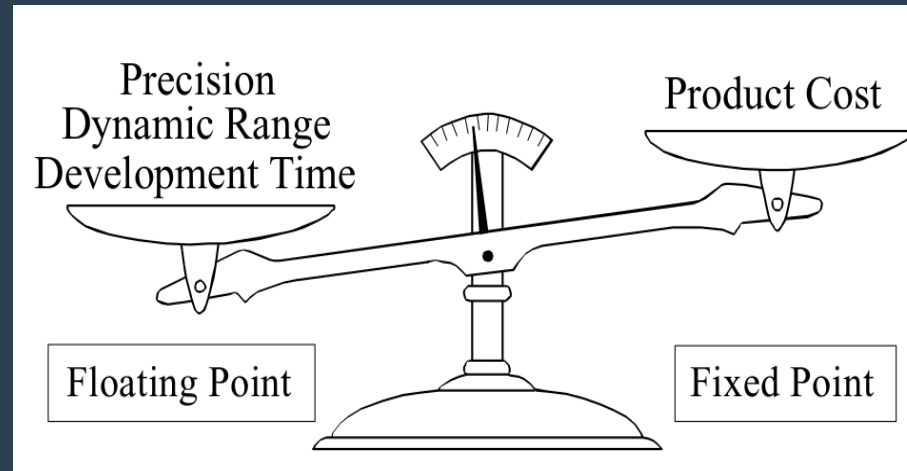
Comparativa Q vs Float

- Float

- Cantidad de patrones de bits= 4,294,967,296
- Gap entre números variable
- Rango dinámico $\pm 3,4e1038$, $\pm 1,2e10-38$
- Gap 10 millones de veces mas chico que el numero

- Q32

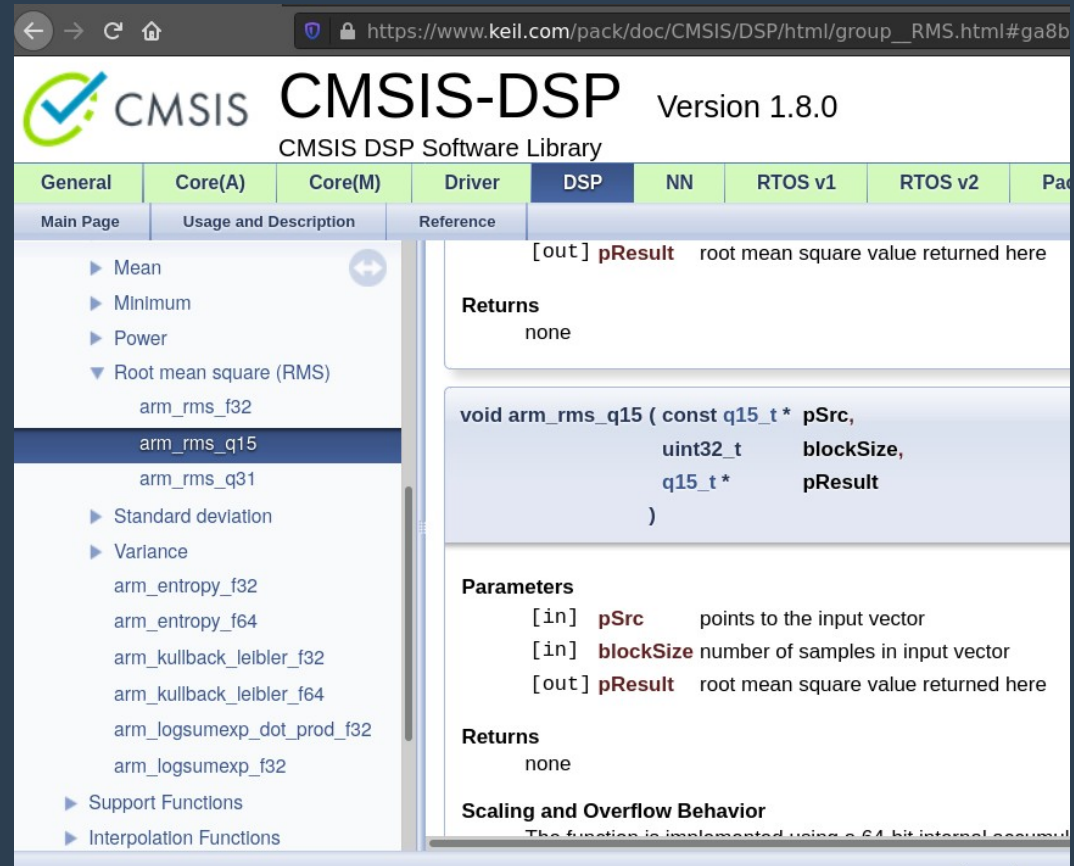
- Cantidad de patrones de bits= 65536
- Gap entre números constante
- Rango dinámico 32767, -32768
- Gap 10 mil veces mas chico que el numero



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 calculo del máximo, mínimo y rms de una señal
- Ver código
 - `ciaa/psf3/psf.c`
- Se grafican los datos, ver código
 - `ciaa/psf3/visualize.py`

