FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Procesamiento de señales, fundamentos

Maestría en sistemas embebidos
Universidad de Buenos Aires
MSE 5Co2020

Clase 2 - CIAA<>Python

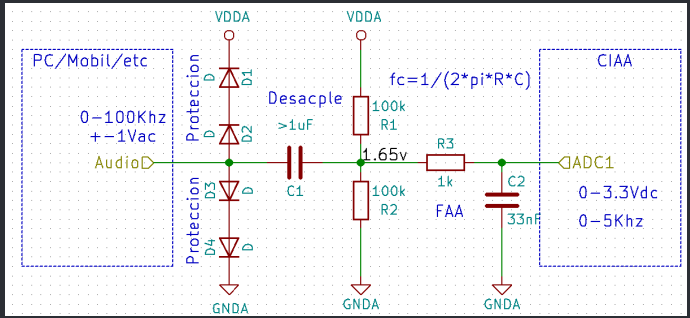
Ing. Pablo Slavkin
slavkin.pablo@gmail.com
wapp:011-62433453

Sampleo

Acondicionamiento de señal

Acondicionar la señal de salida del dispositivo de sonido (en PC ronda ±1V) al rango del ADC del hardware. En el caso de la CIAA sera de 0-3.3V.

Se propone el siguiente circuito, que minimiza los componentes sacrificando calidad y agrega en filtro anti alias de 1er orden.



Sampleo

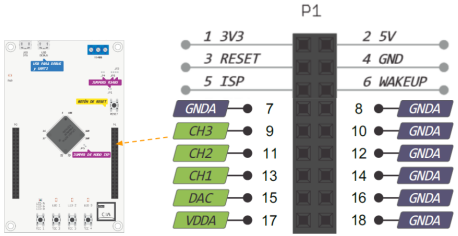
Acondicionamiento de señal

Pinout de la CIAA para conectar el ADC/DAC

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 de audio con Python

simpleaudio lib

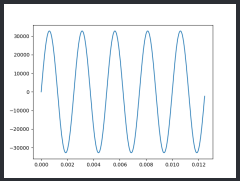
Instalar el modulo simpleaudio para generar sonidos con python

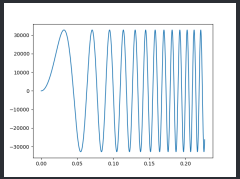
https://simpleaudio.readthedocs.io/en/latest/installation.html Y utilizamos el siguiente código como base:

```
import numpy as np
import scipy.signal as sc
import simpleaudio as sa
import matplotlib.pyplot as plt

f   = 100
fs  = 44100
sec = 20
B   = 5000
t   = np.arange ( 0,sec,1/fs )
note = (2**15-1)*np.sin(2 * np.pi * f * t) #sin
#note = (2**15-1)*np.sin(2 * np.pi * B*t/sec * t)  #
     sweept

audio = note.astype(np.int16)
for i in range(10):
    play_obj = sa.play_buffer(audio, 1, 2, fs)
    play_obj.wait_done()
```





Captura de audio con la CIAA

CIAA->UART->picocom->log.bin

Utilizando picocom https://github.com/npat-efault/picocom o similar se graba en un archivo la salida de la UART para luego procesar como sigue

```
picocom /dev/ttyUSB1 -b 460800 –logfile=log.bin
```

```
#include "sapi.h"

#define LENGTH 512
int16_t adc [ LENGTH ];
uint16_t sample = 0;

int main ( void ) {
   boardConfig       (                          );
   uartConfig        ( UART_USB, 460800         );
   adcConfig         ( ADC_ENABLE               );
   cyclesCounterInit ( EDU_CIAA_NXP_CLOCK_SPEED );
   while(1) {
      cyclesCounterReset();
      uartWriteByteArray ( UART_USB ,(uint8_t* )&adc[sample] ,sizeof(adc[0])
            );
      adc[sample] = ((int16_t )adcRead(CH1)-512);
      if ( ++sample==LENGTH ) { //22.7hz para 512
         sample = 0;
         uartWriteByteArray ( UART_USB ,"header" ,6 );
         gpioToggle         ( LEDR                  );
      }
   while(cyclesCounterRead()< 20400)  //clk 204000000
      ;
   }
}
```

```
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  ................  ........
```

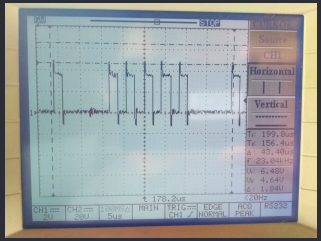
Ancho de banda

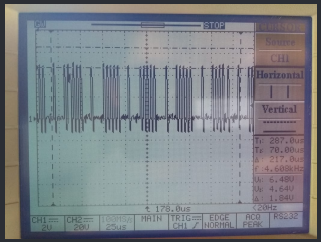
$$USB <> UART_{maxbps} = 460800bps$$

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

$$bits_{muestra} = 16$$

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





Maxima señal muestreable y reconstruible

11520hz

Sampleo

Calculo del filtro antialias 1er orden R-C

$$B = 10kbps$$

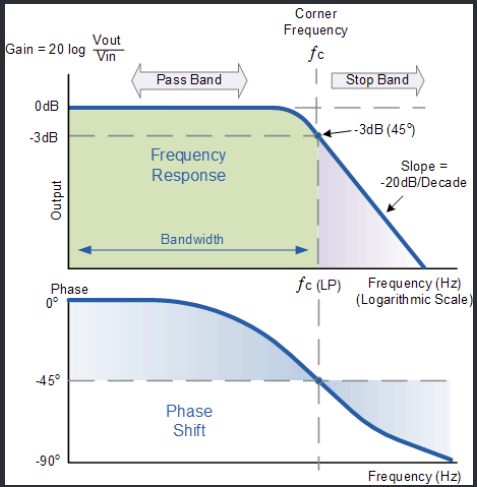
$$f_{corte} = \frac{1}{2 * \pi * R * C}$$

$$R = 1k\Omega$$

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

Maxima señal muestreable y reconstruible

11520hz



Captura de audio con la CIAA

Uart->Python

Lectura de un log y visualización en tiempo real de los datos

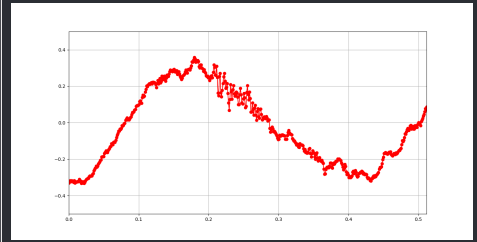
```
import numpy as np
import matplotlib.pyplot as plt
from   matplotlib.animation import
       FuncAnimation
import os

length = 512
fs     = 10000
header = b'header'
fig    = plt.figure (1 )
adcAxe = fig.add_subplot ( 1,1,1 )
time   = np.linspace(0,length/fs,length)
adcLn, = plt.plot ( [],[],'r' )
adcAxe.grid ( True )
adcAxe.set_ylim ( -512 ,512 )
adcAxe.set_xlim ( 0 ,length/fs )

def findHeader(f):
    index = 0
    sync  = False
    while sync==False:
        data=b''
        while len(data) <1:
            data = f.read(1)
        if data[0]==header[index]:
            index+=1
            if index>=len(header):
                sync=True
        else:
            index=0
def readInt4File(f):
    raw=b''
    while len(raw)<2:
        raw += f.read(1)
    return (int.from_bytes(raw[0:2],"
        little",signed=True))

def update(t):
    findHeader ( logFile )
    adc = []
    for chunk in range(length):
        adc.append (readInt4File(logFile))
    adcLn.set_data ( time,adc )
    return adcLn,

logFile=open("log.bin","w+b")
ani=FuncAnimation(fig,update,10,None,blit=
      True,interval=10,repeat=True)
plt.draw()
plt.show()
```



Generación de audio con el DAC de la CIAA

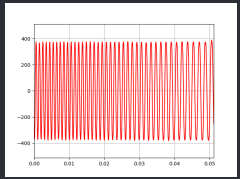
ARM CMSIS-DSP lib *https://www.keil.com/pack/doc/CMSIS/DSP/html/group__sin.html*

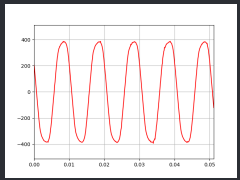
Con arm_sin_f32 se genera un tono y se convierte a analogico con el DAC

```
#include "sapi.h"
#include "arm_math.h"

#define LENGTH 512
#define FS     10000
int16_t  adc[ LENGTH ];
uint16_t sample = 0   ;
uint32_t tick   = 0   ;
uint16_t f      = 100 ;
uint16_t B      = 5000;
uint16_t sweept = 1;
float t         = 0;

int main ( void ) {
   boardConfig ( );
   uartConfig ( UART_USB ,460800 );
   adcConfig ( ADC_ENABLE );
   dacConfig ( DAC_ENABLE );
   cyclesCounterInit ( EDU_CIAA_NXP_CLOCK_SPEED );
   while(1) {
      cyclesCounterReset();
      uartWriteByteArray ( UART_USB ,(uint8_t* )&
            adc[sample] ,sizeof(adc[0]) );
      adc[sample] = adcRead(CH1)-512;
      t=((tick%(sweept*FS))/(float)FS);
      //dacWrite( DAC, 512*arm_sin_f32 (t*B*(t/
            sweept)*2*PI)+512); //sweept
      dacWrite( DAC, 512*arm_sin_f32 (t*f*2*PI)
            +512); //tono
      if ( ++sample==LENGTH ) {
         sample = 0;
         uartWriteByteArray ( UART_USB ,"header" ,6
               );
         gpioToggle ( LEDG );
      }
      tick++;
      while(cyclesCounterRead()<
            EDU_CIAA_NXP_CLOCK_SPEED/FS)  //clk
            204000000
         ;
   }
}
```





Sistemas de números

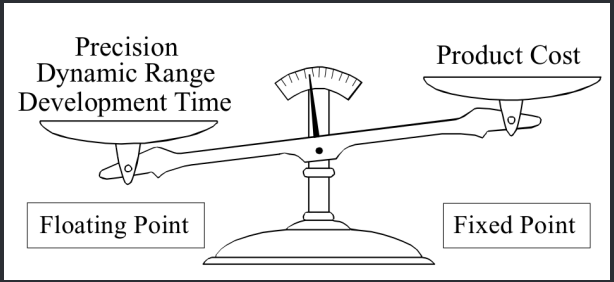
Punto fijo vs punto flotante

Punto fijo:

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

Punto flotante:

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



Sistemas de números

Sistema Q

Qm.n:

- m: cantidad de bits para la parte entera
- n: cantidad de bits para la parte decimal

Q1.15:

1000 0000 0000 0000 = -1

0111 1111 1111 1111 = $1/2 + 1/4 + 1/8 + .. + 1/2^{15} = 0,99$

Q2.14:

1010 0000 0000 0000 = -1.5

0101 0000 0000 0000 = 1.25

Sistemas de números

Sistema Q

Qm.n:

- m: cantidad de bits para la parte entera
- n: cantidad de bits para la parte decimal

Q1.15:

1000 0000 0000 0000 = -1

0111 1111 1111 1111 = $1/2 + 1/4 + 1/8 + .. + 1/2^{15} = 0,99$

Q2.14:

1010 0000 0000 0000 = -1.5

0101 0000 0000 0000 = 1.25