

# Manual de Usuario para el Grabador de Campo

Sabrina Benas, Franco Lucio Tambosco

Laboratorio 6 y 7

Departamento de Física, FCEyN, UBA

2019

## CONTACTO:

Sabrina Benas - [sabrinabenas@gmail.com](mailto:sabrinabenas@gmail.com)

Franco Lucio Tambosco - [franco.tambosco@gmail.com](mailto:franco.tambosco@gmail.com)

# Índice

<b>1. Introducción</b>	<b>2</b>
1.1. Circuito del grabador . . . . .	2
1.2. Algunas especificaciones de los componentes: . . . . .	5
1.2.1. Reguladores de Tensión y Baterías . . . . .	5
1.2.2. TLC2722 . . . . .	5
1.2.3. Tarjeta SD . . . . .	5
1.3. Placa Final del Grabador de campo . . . . .	6
1.4. Diseño de la placa . . . . .	7
<b>2. Especificaciones</b>	<b>8</b>
2.1. Tabla de especificaciones . . . . .	8
2.2. Medición: Tiempo de uso . . . . .	8
2.3. Medición: Rango de uso . . . . .	9
2.4. Medición: SD recomendadas . . . . .	10
2.5. Medición: Audios varios . . . . .	11
<b>3. Guía de Uso</b>	<b>13</b>
<b>4. Software</b>	<b>14</b>
4.1. Microcontrolador MSP430 . . . . .	14
4.1.1. Algunas limitaciones . . . . .	15
4.1.2. Entorno IAR: . . . . .	15
4.2. El programa . . . . .	17
4.2.1. ¿Cómo habilitamos los puertos? . . . . .	21
4.2.2. Botón . . . . .	22
4.2.3. ADC . . . . .	23
4.2.4. TIMER . . . . .	23
4.2.5. Interrupción del timer . . . . .	24
4.3. Comunicación con la SD . . . . .	24
<b>5. Lectura de la tarjeta SD</b>	<b>26</b>
5.1. Lectura de SD y pasaje a decimal . . . . .	26
5.1.1. El programa... . . . .	26
5.2. Reconstrucción del Audio, Python . . . . .	29
5.2.1. Audio total... . . . .	29
5.2.2. Audio entrecortado... . . . .	31

# 1. Introducción

El monitoreo automático de las poblaciones de animales salvajes tiene dos pilares: la construcción de sensores capaces de registrar y almacenar los registros, y el software que identifica, a partir de las señales, la presencia de individuos de las especies monitoreadas. En el caso de realizar un monitoreo por medio de las vocalizaciones, un elemento clave es contar con grabadores lo suficientemente económicos para afrontar las dificultades de establecer una red en el campo (riesgo de daño por causas ambientales, vandalismo, etc). En el mercado, los grabadores típicamente están preparados para una interacción versátil con el usuario, y para cubrir un rango amplio de frecuencias, sin embargo su elevado precio los hace incompatibles con las tareas de monitoreo en el campo.

Es por ello que procedió al diseño y construcción de un grabador de campo de bajo costo, el cual de ahora en adelante se denominará grabador **TamBe**. Este puede recibir señales y grabar las mismas en una tarjeta microSD.

## 1.1. Circuito del grabador

En la Figura 2 se presenta el circuito del grabador TamBe, en el cual se detallan las diferentes secciones del mismo.

En primer lugar la alimentación es a través de una batería (LiPo 300 mA/Hora) la cual se encuentra conectada a dos reguladores de tensión (modelo LP2985 de *Texas Instruments*). El primero *Regulador de Tensión microSD* se encarga de entregar un voltaje de 3,3 V a la tarjeta microSD, mientras que el *Regulador de Tensión* se encarga de entregar 3,3 V a un micrófono pastilla (*electret conde*), un amplificador y el microcontrolador MSP430. Cada regulador de tensión se encuentra en una configuración recomendada por el fabricante <sup>1</sup>. Una vez alimentado el sistema, se encuentra la sección *Micrófono y Amplificación*, en esta se encuentra el micrófono pastilla conectado a un amplificador TLC2722 en una configuración no inversora. En la entrada de la señal se encuentra un filtro de 20 Hz (rojo punteado) que se encargan de filtrar el ruido de línea DC. A la salida del amplificador, se encuentra un filtro 13 KHz, con el fin de evitar *aliasing*. En el sistema de ganancia del amplificador se encuentra una resistencia variable, la cual se encarga de permitir un aumento o disminución de la ganancia en caso de ser necesario. En la Figura 1, se muestra una simulación de la ganancia en función de la frecuencia de la sección *Micrófono y Amplificación*. Se observa que el rango de funcionamiento óptimo es entre 1kHz a 10kHz, lo cual se encuentra entre el rango de canto aviar.

---

<sup>1</sup><http://www.ti.com/lit/ds/symlink/lp2985.pdf>

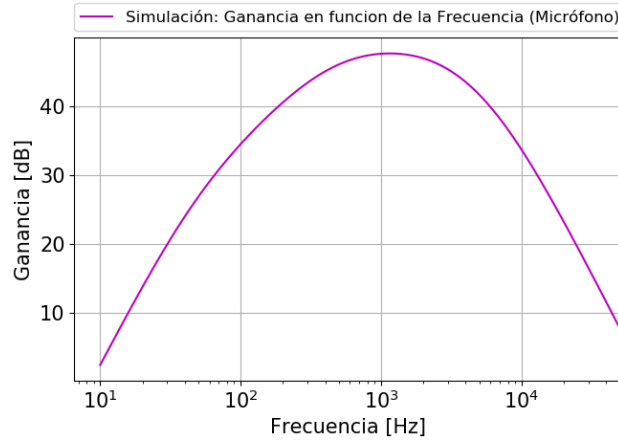


Figura 1: Simulación de la ganancia del sector *Micrófono y Amplificación* en función de la frecuencia

Luego, esta sección se encuentra conectada a la sección del *Microcontrolador*, la cual contiene el MSP430G2553, un LED y dos capacitores en paralelo (uno cerámico y otro polarizado) con la función de desacoplar su alimentación (violeta punteado). De esta sección se desprenden 3 secciones: el *Botón* (azul), *Buzzer* (celeste) y la *microSD* (verde). En esta última sección, se encuentra un filtro 8 Hz en la alimentación de la microSD (verde punteado) y un capacitores para el desacople en paralelo, igual que en la Sección *Microcontrolador*. Este filtro es de suma importancia debido a que la tarjeta microSD consume más corriente cuando escribe. Este consumo produce una frecuencia parásita en los audios. Este filtro logra separar en buena medida la alimentación de la SD del resto del sistema junto con los dos reguladores de tensión.



## 1.2. Algunas especificaciones de los componentes:

### 1.2.1. Reguladores de Tensión y Baterías

Debido que para el grabado de sonido era necesaria la utilización de una batería económica y que pueda entregar pulsos de corriente, se decidió utilizar baterías de Litio, del tipo polimérico (LiPo). Estas, son utilizadas en juguetes de radio y drones, por lo que están diseñadas para soportar descargas rápidas tanto pulsadas como continuas. En este trabajo se utilizó una batería Hyperion G3 CX - 1S 3.7V 600 mAh, con un peso de 4 g y un costo de 4.5 dolares.

Dado el voltaje de la batería, se decidió utilizar el regulador de tensión integrado LP-2985-N de *Texas Instruments*, que aparte de cumplir la función de mantener una tensión de alimentación constante de 3.3V (Valor necesario para el correcto funcionamiento del microcontrolador), este posee un bajo *drop-out*, es decir, para su funcionamiento consume 0.3 V. Este tipo de reguladores de tensión se denominan de *Ultra low drop-out* y tiene un valor de aproximadamente 4 USD, siendo la única opción disponible comercialmente en Argentina.

### 1.2.2. TLC2722

El amplificador operacional TLC2722 tiene la ventaja de ser *rail to rail*, es decir, la salida puede alcanzar voltajes muy cercanos a los de la alimentación y por lo tanto se puede utilizar el rango de tensión bajo el cual operan mas eficientemente. El ancho de banda es de 2.2 MHz y el consumo típico es de 2.2 mA

### 1.2.3. Tarjeta SD

Se propuso utilizar, como medio de almacenamiento, una tarjeta microSD comercial. Estas tarjetas utilizan el protocolo SPI (*Serial Port Interface*) y a la hora de la escritura de datos en estas habrá que tener en cuenta que están divididas en sectores de a 512 bytes (1 sector es igual a 512 bytes) y que los datos pueden guardarse de a bytes (8 bits).

### 1.3. Placa Final del Grabador de campo

En la siguiente Figura 3 se encuentra en diseño del PCB del grabador TamBe. Cabe destacar que todos los componentes que no tengan pista, están conectados a través de tierra. Para una mejor visualización no se colocó la masa de tierra que conectaría estos últimos. Por otro lado, se nota que hay pistas de dos colores: rojas y azules. Esto es porque la impresión será doble faz, es decir, para una placa de cobre habrá pistas por debajo y por prolijidad unas pocas por arriba junto con las componentes. Las pintas azules se pueden ver en la Figura 5 y las componentes y las pistas rojas en la Figura 4.

Cabe destacar que en letras Rojas dice Reloj Externo. Esto es si para en un futuro se quisiera utilizar un reloj externo y no el del microcontrolador.

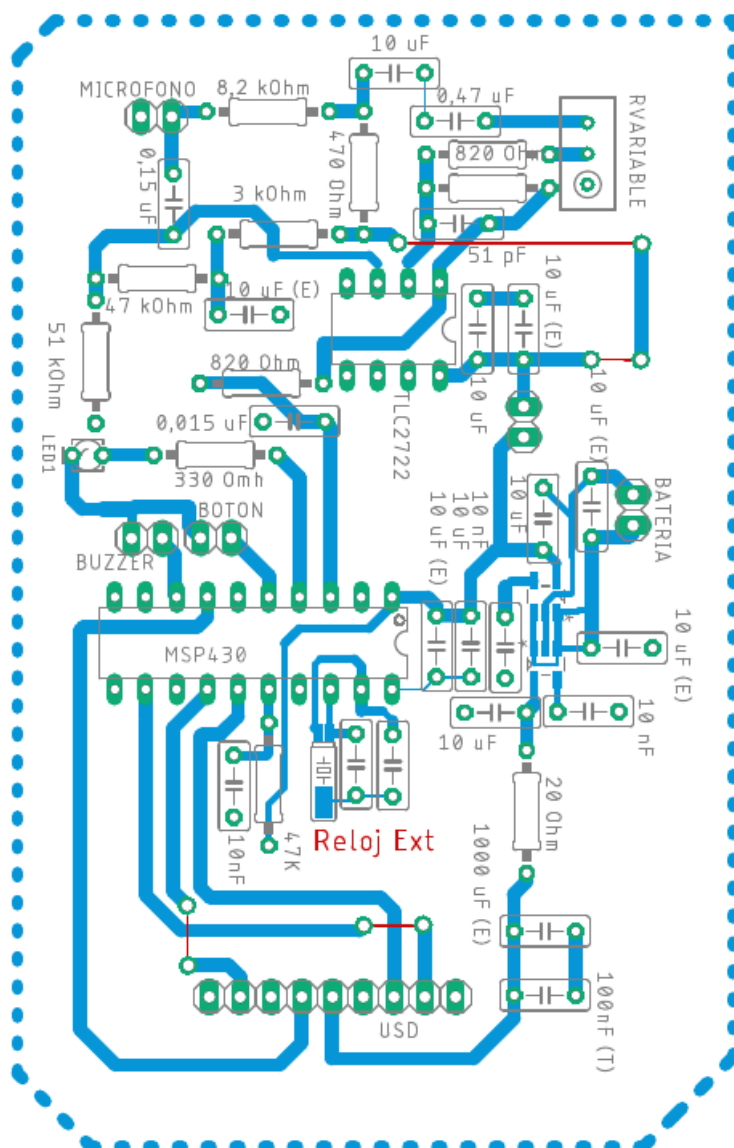


Figura 3: Diseño del PCB del grabador.



## 1.4. Diseño de la placa

En la Figura 4 puede observarse el diseño de la placa del grabador TamBe. Es posible diferenciar de esta manera los diferentes sectores y componentes descritos en la Figura 2, pero ordenados de forma que cada sección quede lo más compacta. La placa tiene un tamaño 9x6 cm. En la Figura 6 puede observarse la placa de la sección *Boton*, la cual se conecta como se indica en la Figura 2 a través de cables. La placa del botón contiene pines dobles, lo cual permite que pueda ser usada en dos dispositivos a la vez.

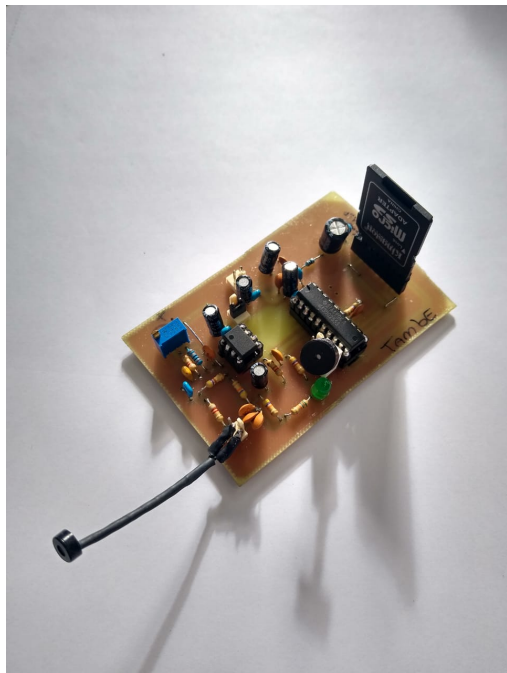


Figura 4: Foto de la cara superior del grabador

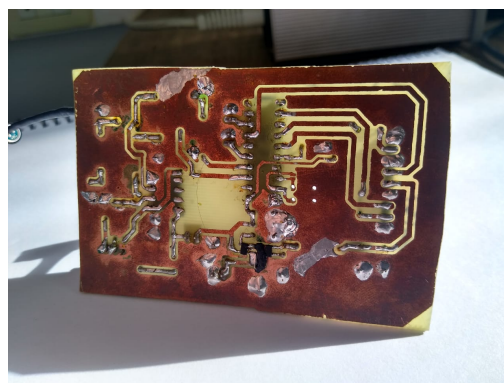


Figura 5: Foto de la cara inferior del grabador

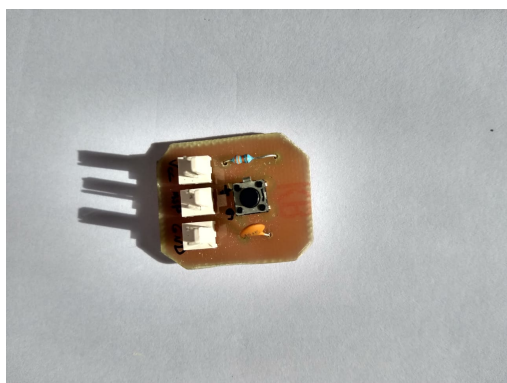


Figura 6: Foto de la cara superior de la placa del botón



Figura 7: Foto de la cara inferior de la placa del botón

## 2. Especificaciones

### 2.1. Tabla de especificaciones

En esta sección se muestran las especificaciones del grabador TamBe. En primer lugar se muestra una tabla para su visualización rápida, mientras que posteriormente se muestran las mediciones que llevaron a dichos resultados

Descripción	Valor
Frecuencia de Adquisición *	22 kHz
Buffer *	207 bytes
Distancia Máx. de la fuente sonora	$(11 \pm 1)$ m
Duración de grabación continua	13 horas
Micrcontrolador	MSP430G2553
Amplificador	TLC2722
Regladores de Tensión	LP2985

\*Este valor puede ser cambiado mediante manipulación del código

### 2.2. Medición: Tiempo de uso

Mediante la utilización de una placa DAC, se procedió a conectar una entrada a la batería y otra a un regulador de tensión para medir el voltaje de ambas componentes. Es así que se comenzó a grabar y no se perturbó la grabación hasta que la batería se agotase. Des esta forma, se obtuvo la Figura 8. Aquí se observa que el tiempo de duración del grabador es de 13 horas, después de las cuales el regulador de tensión deja de entregar 3,3 V, por lo que las mediciones pierden fidelidad.

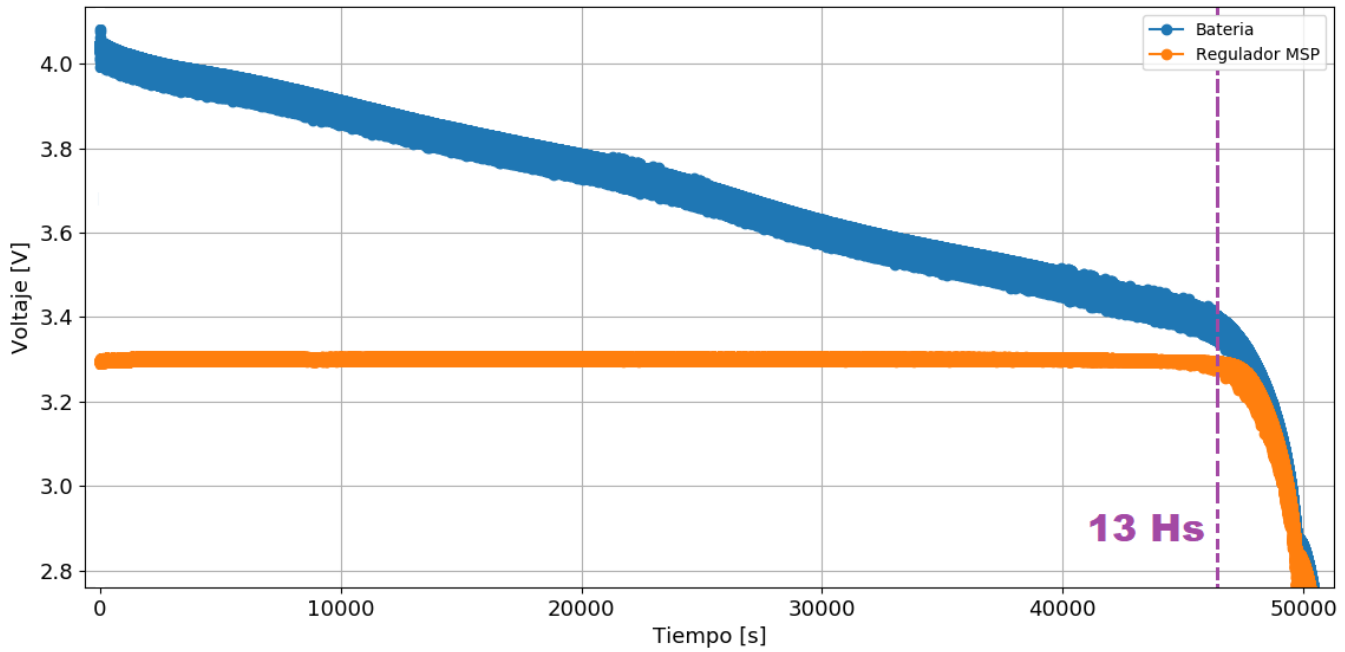


Figura 8: Curvas de Tensión de la batería y el regulador del microcontrolador

### 2.3. Medición: Rango de uso

Con el objetivo de determinar el rango de distancia de grabación del sistema, se procedió a realizar una medición enviando, mediante un parlante, una señal sinusoidal de frecuencia: 3 kHz, variando la distancia a la que se encontraba la fuente. La frecuencia fue electa debido a que se encuentra dentro del rango de frecuencias del canto avar. La medición se realizo al aire libre, ya que su aplicación es para ambientes con ruido de fondo. De esta forma se llego a la Figura 9.

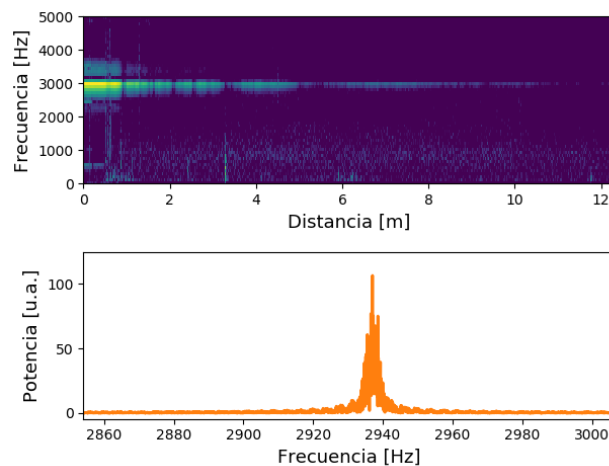


Figura 9: Espectrograma de frecuencias en función de la distancia.

Es posible así observar que un correcto análisis de la señal puede hacerse hasta  $(11 \pm 1)m$  en el exterior. Se utilizó un decibelímetro para observar la potencia sonora de la señal a 1,5 metros del grabador, resultando

en 60 dB. Esto coincide con la misma medición, pero con el canto de un canario.

## 2.4. Medición: SD recomendadas

Las tarjetas microSD tienen una latencia de escritura, debido a protocolo interno de la misma. Este tiempo no es periódico y no está bien documentado por el fabricante. Con el objetivo de observar estas diferencias, se procedió a enviar una señal triangular, de 2 kHz con un  $V_{pp} = 1,3V$ , con un generador de funciones al ADC10, y a recolectar los datos en diferentes tarjetas microSD. Es así que se analizaron 2 tarjetas de diferente clase y marca: una de clase 4 y otra de clase 10. En las Figura 10 puede observarse las diferentes señales recolectadas de ambas tarjetas y los datos perdidos para una tarjeta de clase 4. Mientras que en la Figura 11 los errores de una tarjeta de clase 10.

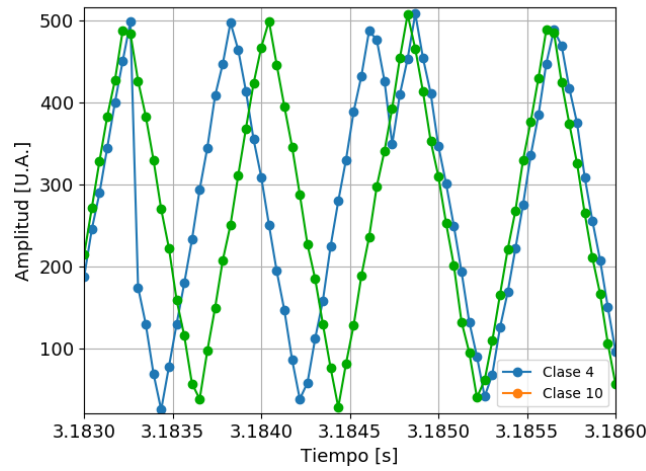


Figura 10: Señal triangular para dos tarjetas SD. Aquí se muestran la pérdida de puntos para una tarjeta sd de clase 4.

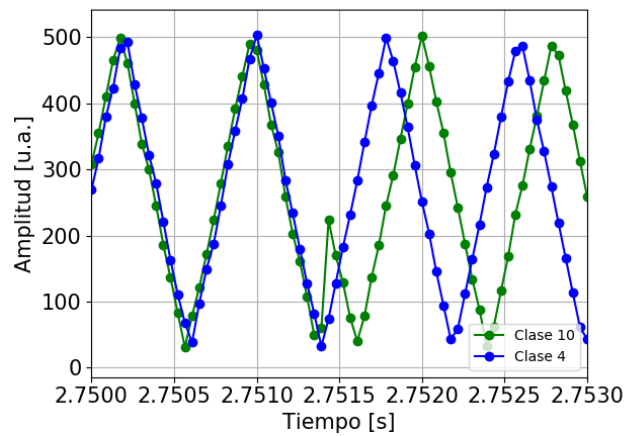


Figura 11: Señal triangular para dos tarjetas SD. Aquí se muestran la pérdida de puntos para una tarjeta sd de clase 10.

Se observó que para la misma señal en 15 segundos, la tarjeta de clase 10 tiene errores que aparecen

dos veces, mientras que para la de clase 4 aparecen 6 veces en ese tiempo. Esto se condice con la diferencia de tasa media de escritura para cada una de ellas. Es así que puede observarse la superioridad de la tarjeta microSD de clase 10 en comparación a las de clase 4, al tener una perdida mucho menor de datos. Es importante aclarar que para el análisis de audio esta cantidad de puntos no presentan un problema, pero en el marco del proyecto de telemetría dificulta la sincronización entre varios equipos.

## 2.5. Medición: Audios varios

En la Figura 12 puede observarse el canto de un diamante mandarín del Laboratorio. El grabador se encontraba posicionado a un metro del pájaro. En la figura puede observarse la amplitud en función del tiempo con su respectivo espectrograma. Es posible observar la presencia de la frecuencia fundamental cerca de 1 kHz y los primeros cinco armónicos del canto, estas mediciones coinciden con el rango de canto de estas aves.

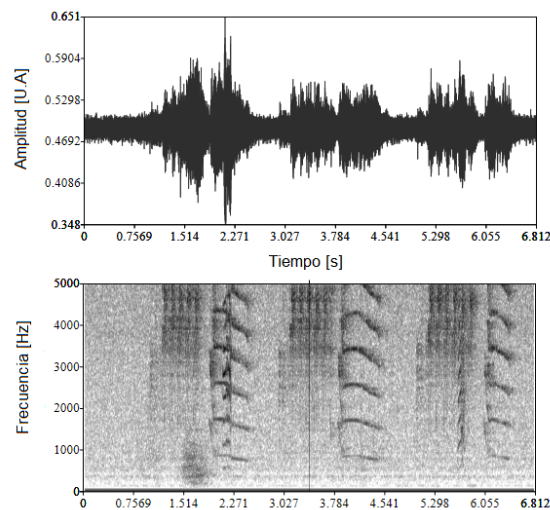


Figura 12: Amplitud y frecuencia en función del tiempo para el canto de un diamante mandarín.

Además se realizaron mediciones sobre la voz humana, como puede observarse en la Figura 13:

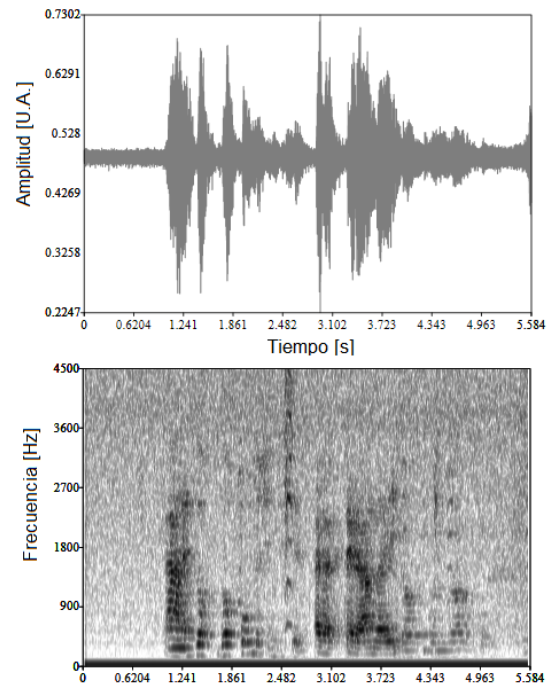


Figura 13: Amplitud y frecuencia en función del tiempo para el habla humana.

### 3. Guía de Uso

A continuación se detallan los pasos para el grabado de un audio con el grabador TamBe:

- Paso 1: Posicionar el grabador a menos de 11 metros del objetivo. Antes de comenzar a grabar asegurarse que se encuentre insertada una microSD, es preferible que esta se encuentre formateada previamente (se han observado mejores resultados en los audios resultantes). En el caso que se quiera tener una referencia de tiempo al usar más de un grabador, insertar el *buzzer*, en caso contrario se recomienda no utilizarlo ya que esto podría producir ruidos molestos en la grabación.
- Paso 2: Conectar la batería al equipo (asegurarse que esta se encuentre cargada, si es así la duración se la grabación podrá ser de hasta 13 horas).
- Paso 3: Tocar el botón una vez (no varias veces ni mantenerlo apretado) y observar el LED. En caso que este comience a titilar en un corto tiempo, esto significa que no se ha producido la comunicación entre el microcontrolador y la tarjeta sd, por lo que es necesario desconectar la batería, asegurarse de que la microSD se encuentre bien conectada y luego volver al Paso 2. En caso contrario, la LED titilará cada 11 segundos aproximadamente (esto corresponde a 800 sectores de la microSD), lo cual indica que el equipo se encuentra grabando.
- Paso 4: Dejar el tiempo deseado, se recomienda no mover mucho el grabador ya que puede producirse un corte en la comunicación y arruinar la medición. Verificar cada tanto que la LED continúe titilando.
- Paso 5: Una vez que se desee terminar la medición, desconectar la batería previo a quitar la microSD. No se tiene que volver a tocar el botón. En caso de que la medición se haga hasta que se descargue la batería, el procedimiento es el mismo. Una vez que se tenga la microSD en mano, introducirla en la computadora y seguir los pasos descriptos en la sección *Lectura de la tarjeta SD*

## 4. Software

Todos los programas se encuentran en el git:

<https://github.com/Sabrineitor/Laboratorio-6-7/tree/master>

### 4.1. Microcontrolador MSP430

el microcontrolador se encarga de guardar una señal de audio en una tarjeta microSD. Estas tarjetas utilizan el protocolo SPI (*Serial Port Interface*) y a la hora de la escritura de datos en estas habrá que tener en cuenta que están divididas en sectores de a 512 bytes (1 sector es igual a 512 bytes) y que los datos pueden guardarse de a bytes (8 bits). Entonces, es necesario configurar el microprocesador para el protocolo de comunicación con la tarjeta, establecer la comunicación y finalmente preparar y montarse sobre un sector de la tarjeta para poder escribirlo. Para esto, se utilizaron las siguientes librerías: *mmc.h* que tiene definidas las funciones de la tarjeta y *hal\_spi.h*, las funciones del protocolo de la comunicación. (obs: no son librerías como python, dependiendo el autor pueden variar drásticamente.)

También, es necesario configurar el ADC10 (conversor analógico digital de 10 bits) y el *timer* del microcontrolador para tener mediciones en intervalos definidos. Es decir, tener una conversión cada vez que el *timer* marca. Cabe destacar que la tarjeta solo puede guardar datos de 8 bits, por ende, como el ADC10 envía 10 bits, estos se guardan en dos paquetes, uno de 8 y otro de 2 bits!

Para escribir datos en una tarjeta microSD es necesario primero abrir un sector, montarse sobre él, guardar los datos y luego cerrarlo. Estas operaciones referentes a la tarjeta pueden llevar más tiempo que el tiempo entre mediciones. Entonces, para evitar que la transmisión de datos ocurriera después de la siguiente adquisición se decidió utilizar un buffer circular. Durante las operaciones de la tarjeta, este guarda los datos hasta que puedan ser enviados y debe ser vaciado ante el cambio del sector. Es importante destacar que la frecuencia de adquisición debe ser menor a la de transmisión, sino los datos comenzarían a sobre-escribirse.

En la Figura 14, es posible observar el diagrama de flujo del código utilizado para adquirir una señal de audio, digitalizarla y guardarla en la tarjeta microSD. En primer lugar se encuentra la etapa de configuración: se configura el reloj digital interno del microcontrolador que define una velocidad de procesamiento a 16MHz, la comunicación SPI para la tarjeta sd, la inicialización de la misma y luego la configuración del ADC10 (en un rango de 0 a 2.5 V) y el *timer*. Aquí se habilitan las interrupciones (ver Figura 15). Estas son bloques de código que se ejecutan en momentos específicos. En este caso, cuando el *timer* llega a el valor especificado, se desencadena, habilitando la realización de una medición. A su vez se utilizan para establecer la frecuencia de muestreo y asegurar su estabilidad. En ella se guardan los datos a enviar en un buffer interno del microprocesador y luego se envían cuando la operación se pueda realizar.



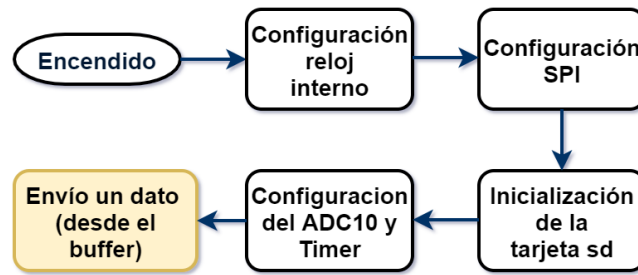


Figura 14: Diagrama de flujo del código utilizado en el MSP430G2553 para la grabación de la señal en una tarjeta microSD

También se le agregaron funciones para una visualización rápida del correcto funcionamiento del grabador (titileo de una led), un botón para inicializar el grabado y un buzzer para tener una referencia sonora cada un minuto. Los tiempos de titileo o cuando suena el buzzer se pueden cambiar a gusto del consumidor.



Figura 15: Interrupción

#### 4.1.1. Algunas limitaciones

Una de las limitaciones de esta tecnología es la capacidad de almacenamiento de los procesadores que es de tan solo 512 bytes de RAM. Debido a esto, el tiempo entre la adquisición y grabado, es decir, el tiempo que el dato está en el buffer, no es muy grande (i.e. podría ser sobre-escrito) y esto depende de la frecuencia de muestro. Por otro lado existen dos limitaciones por parte de las tarjetas microSD: el tiempo de preparación de la memoria del orden del milisegundo y la capacidad de garantizar una tasa mínima de escritura. En tarjetas comerciales de clase 4 la tasa media mínima de escritura es de 4MB/s mientras que para las de clase 10, 10MB/s, según fabricante. Para garantizar esta tasa media, la tarjeta contiene una programación interna mediante la cual prepara varios centenares de sectores de memoria para la escritura. Estos procesos llevan más tiempo que el cambio de sector, no son revelados por el fabricante y pueden variar de tarjeta a tarjeta. Se observó que con una tarjeta de clase 10 se pierde menos que con una de clase 4.

#### 4.1.2. Entorno IAR:

Para subir/quemar el programa en el msp para que haga lo que queremos, esta el **entorno iar**. El entorno es pago, sin embargo existen versiones gratuitas.

Hago el ejemplo:

1. creo o abro un Workspace
- file— >New — > Workspace

file -> open -> Wokspace

(esto es para crear un entorno de trabajo)

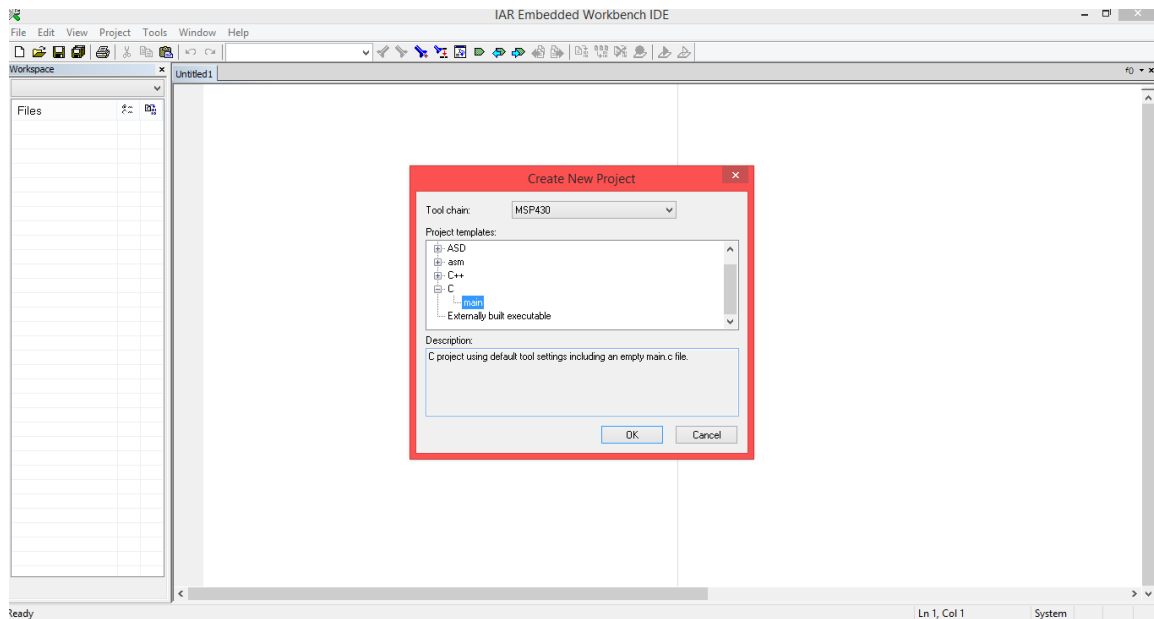
2. creo un proyecto:

project -> create new project

En esa ventana hay que elegir:

tool chain: msp430

C -> main (acá va a ir el programa que querramos subir)

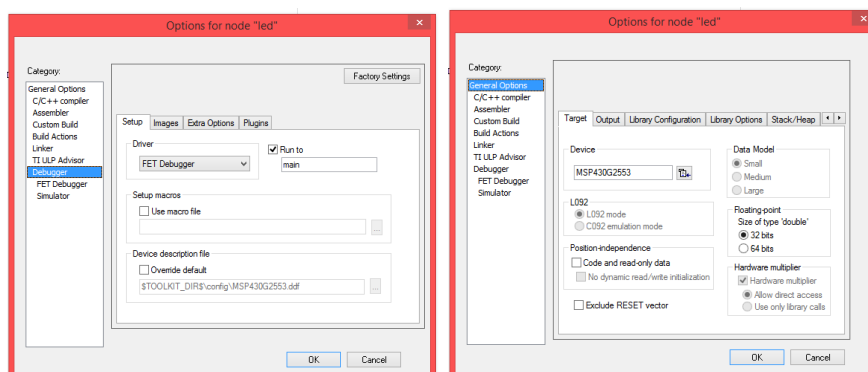


3. Preparo para descargar un programa al msp:

project -> options -> general options ->

A) target -> device -> msp430g2553 (o el que se use)

B) -> Debugger -> set up -> en driver FET-Debugger.



4. Para subirlo:

project -> Download -> eraser memory

project -> Download -> Download active application

**OBS:** puede aparecer una ventana que diga que tiene errores en el fet, se puede ignorar. Los que no hay que ignorar son los que aparecen en el dialogo de abajo del entorno. esos son porque hay problemas de código y significa que no se subió.

En el git si se baja la carpeta completa, no hace falta crear el workspace! Abrir el Iar, open workspace ir a donde está la carpeta y tocar el archivo Labo67-Sabri-Fran-MSP.eww y este ya abre el main (si esto no funciona crear el workspace).

## 4.2. El programa

Adjuntamos el programa comentado + la explicación de los registros:

```
1 // Librerias a incluir
2 #include "io430.h"
3 #include "intrinsics.h"
4 #include "bsp.c"
5 #include "mmc.c"
6
7 //Declaracion de variables
8 unsigned char status = 1;
9 unsigned char timeout = 0;
10
11 //Buffer de escritura
12 short int buffer_in = 0; // dejarlo como int para que no haga cambio de dato
13 short int buffer_out = 0;
14
15 // Definicion de Macros
16 #define LED_WRITE P1OUT_bit.P2
17 #define Boton BIT3
18 #define buffer_size 207 //180
19
20 struct audioStr {
21     volatile unsigned char msb[buffer_size]; //stores one 8 bit audio output ←
        sample
22     volatile unsigned char lsb[buffer_size]; //guarda los 2 bits perdidos (10←
        bit ADC -> 8bit audio)
23     unsigned char Rec; //estamos en modo grabar
24     unsigned char bufferBytes; //number of audio bytes available
25     unsigned char Button; //variable boton
26 } audio;
27
28 struct flashstr {
```

```

29 unsigned short currentByte;
30 unsigned long currentSector;
31 //Dejarlo en long, no se debe poner en int
32 // Una tarjeta de 8 Gb tiene 15 759 360 sectors
33 // 4 GB segun WinHex: 7 864 320
34 // Con unsigned int de 0 to 65535
35 } flashDisk;
36
37 unsigned char sdWrite(unsigned char data) {
38 //this function writes one byte to the SD card
39 //it takes care of switching to the next 512 byte sector
40 //the first sector to write to must already be mounted
41 //the flaskdisk variables must be setup for the location to start reading
42
43 unsigned char status;
44 mmcWriteByte(data); //write audio sample to the sector
45 flashDisk.currentByte++; //increment the byte counter
46
47 if(flashDisk.currentByte==512+1) { // Cambio de sector, LED y buzzer
48     mmcWriteUnmount(flashDisk.currentByte); //close the sector
49     flashDisk.currentByte=1; //reset byte counter to 1
50     flashDisk.currentSector++;
51     status=mmcWriteMount(flashDisk.currentSector); // Se monta por sector!
52
53     //LED
54     if(flashDisk.currentSector % 800 == 0) {P1OUT |= BIT2;} //LED (←
55     on)
56     if((flashDisk.currentSector - 20) % 800 == 0) {P1OUT &= ~BIT2;} //LED (←
57     off)
58
59     //BUZZER
60     if(flashDisk.currentSector % 5000 == 0) {P2OUT|= BIT0;} //Buzzer (on)
61     if((flashDisk.currentSector - 20) % 5000 == 0) {P2OUT &= ~BIT0;} //←
62     Buzzer (off)
63 }
64 return status;
65 }
66
67 //Funcion para el boton
68 void Button(){
69     P1DIR &= ~Boton; // poner P1.3 como entrada

```

```

67  P1REN |= Boton;    // habilito pull up/down p1.3
68  P1OUT |= Boton;    // lo pongo como pull up
69  P1IE |= Boton;     // Habilito la interrupcion
70  P1IES = Boton;     // Activa por frente de caida
71  P1IFG &= ~Boton;   // Por si acaso se limpia la bandera
72 }
73
74 //config ADC
75 void ADC10(){
76     ADC10CTL0 &= ~ENC; //(apagar antes de configurar)
77     ADC10CTL0 = SREF_1 + REF2_5V + REFON + ADC10SHT_3 + ADC10ON + ADC10IE;
78     ADC10AEO |= BIT1;
79 }
80
81 //config timer
82 void Timer(){
83     CCTLO = CCIE;
84     TAOCTL = TASSEL_2 + ID_0 + MC_1;
85     TACCR0 = 84; // TACCR0 + 1 = 42.5 us -> 23.5 kHz
86     TACCR1 = 29 ;
87 }
88
89 void Rec(){
90     mmcReadUnmount(flashDisk.currentSector);
91     audio.bufferBytes=0;
92     audio.Rec=0;
93     flashDisk.currentByte=1;
94     status=mmcWriteMount(flashDisk.currentSector); //mount the first sector so←
           it is ready to write, now all writes go through sdWrite()
95 }
96
97 int main(void){
98     WDTCTL = WDTPW + WDTHOLD; //linea que siempre tiene que estar para que no ←
           se resetee todo el tiempo
99     //llamo a estas lib, aca def reloj interno
100     MainClockInit();
101     SMCLKInit();
102
103     P2DIR_bit.P0 = 1; // Buzzer suena una vez
104     __delay_cycles(100000);
105     P2OUT_bit.P0 ^= 1;

```

```

106
107         P1DIR_bit.P2 = 1; // Led escritura, encendido hasta que se oprime
108
109 while (status != 0) { //Inicializacion de la SD
110     status = disk_initialize();
111     timeout++;
112     LED_WRITE ^= 1;
113     __delay_cycles(500000);
114
115     //si no se comunica con la sd, titila rapido:
116     if (timeout == 150){
117         LED_WRITE ^= 1;
118         __delay_cycles(25000);
119         LED_WRITE ^= 1;
120         __delay_cycles(25000);
121     }
122 }
123 LED_WRITE ^= 1;
124
125 ADC10(); // Seteo del ADC
126 Rec(); // Seteo del Sector
127 flashDisk.currentSector=0; //empiezo en cero
128 Timer(); // Seteo del Timer
129 Button(); // Seteo del Boton
130
131 __enable_interrupt(); // Habilitacion de las interrupciones
132
133 while(1){
134     if(audio.Rec){ // Si estamos en modo grabar (i.e. tocamos el boton)
135         if(audio.bufferBytes>0){ //There is a byte to be written in the ↵
            record buffer
136             buffer_out = buffer_in-audio.bufferBytes+1;
137             if(buffer_out < 0) buffer_out += buffer_size;
138             status=sdWrite(audio.msb[buffer_out]); //write the byte, get the ↵
                result code in status
139             status=sdWrite(audio.lsb[buffer_out]); //escribo el segundo byte, con↵
                los digitos menos significativos
140             audio.bufferBytes--; //subtract one from the buffer counter
141         }
142     }
143 }

```

```

144 }
145
146 // Para las interrupciones, vectorizacion para el compilador
147
148 #pragma vector = PORT1_VECTOR // Start del sistema: Boton (P1.3, BIT3)
149 __interrupt void PORT1(void) {
150     if ((P1IFG & Boton) != 0) { // chequea la interrupcion
151         if (audio.Rec == 0) {audio.Rec = 1;} //si tocamos el boton esto variable ←
            es uno y entra al while del main y empieza grabar
152     P1IFG &= ~Boton; // clears P1.3 interrupt flag
153 }
154 }
155
156 #pragma vector=TIMER0_A0_VECTOR // Interrupcion activacion de conversion del ←
    ADC10 (Cuando el timer llega a TACCR0)
157 __interrupt void interrupt_timer(void) {
158     ADC10CTL0 &= ~ENC;
159     ADC10CTL1 &= ~0xFFFF;
160     ADC10CTL1= CONSEQ_0 + INCH_1;
161     ADC10CTL0 |= ENC + ADC10SC;
162 }
163
164 #pragma vector=ADC10_VECTOR //Interrupcion ADC10 (ADC10 completa una ←
    medicion)
165 __interrupt void ADC10_ISR(void) {
166     audio.bufferBytes++;
167     buffer_in = buffer_in + 1;
168     if (buffer_in == buffer_size) buffer_in = 0;
169     while (ADC10CTL1 & BUSY);
170     // El ADC10 guarda la medicion de 10 bits en ADC10MEM
171     audio.msb[buffer_in]=(ADC10MEM >>2); //convert audio to 8 bit sample.... ←
        >> binary shift -> mueve los bits 2 lugares
172     audio.lsb[buffer_in]=ADC10MEM & 0x03;
173 }

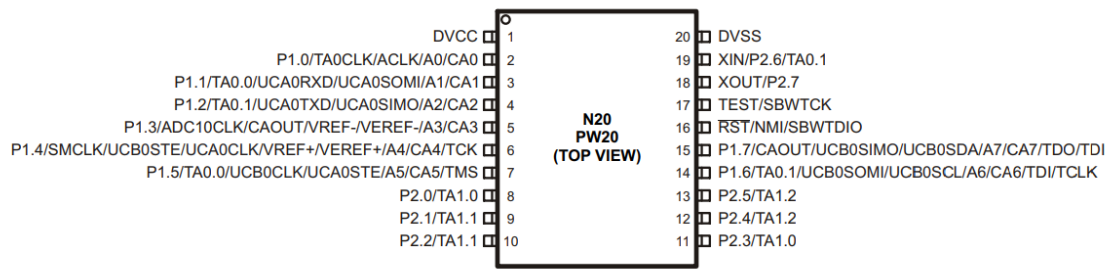
```

#### 4.2.1. ¿Cómo habilitamos los puertos?

La pregunta significa: ¿cómo hacemos que la patita 1 del msp sea por ejemplo una LED?

Antes de eso, hay que tener en cuenta que el msp se divide en dos sectores: tiene 8 patitas que se llamas P1.0 ; P1.1 ; P1.2 .... y otras 8 que son P2.0 ; P2.1 .... como se muestra en la figura 16.

## Device Pinout, MSP430G2x13 and MSP430G2x53, 20-Pin Devices, TSSOP and PDIP



NOTE: ADC10 is available on MSP430G2x53 devices only.

NOTE: The pulldown resistors of port P3 should be enabled by setting P3REN.x = 1.

Figura 16: MSP430

Hay varias formas de habilitar un puerto: una es llamar una variable marco LED\_WRITE y la definimos como P1OUT.bit.P2 – > esto significa las patitas p1 las ponemos como una salida P1OUT y de todas las p1 elegimos la P1.2.

Otra forma es definir la variable Boton como BIT3 esta significa que la patita P1.3 es una 'salida' que la llamamos Boton porque ahi vamos a poner el botón.

Algunos símbolos:

| = operator does a bitwise OR

&= Bitwise AND assignment operator

&= ~ does a bit clear

### 4.2.2. Botón

En la función Button esta la configuración para que la patita P1.3 sea el botón:

Registros	Función
PxDIR	le digo la dirección del pin (BIT0, BIT1...)
PxREN	activa o desactiva la resistencia pullup / pulldown correspondiente al pin
PxIE	habilita las interrupciones!
PxOUT	defino el pin x como salida
PxIES	Activa por frente de caída
PxIFG	Limpia la bandera de la interrupción

Para mas información ir al capitulo 8 del manual del MSP. <sup>2</sup>

<sup>2</sup><https://www.ti.com/lit/ug/slau144j/slau144j.pdf>



### 4.2.3. ADC

Registros	Función
SREF_1	Selección de referencia, al poner uno estamos diciendo: $VR+ = VREF+$ and $VR- = VSS$
REF2.5V	Reference-generator voltage (en este caso es 2.5 V) (va de 0 a 2.5 V)
REFON	Generador de referencia ON
ADC10SHT_3	sample and hold ('cuanto tiempo está el dato en el cap interno') Al poner tres estamos diciendo 64 ADC10CLKs
ADC10ON	Prende al adc
ADC10IE	habilita las interrupciones
ADC10AE0	aca le decimos que patita es

Para más información ir al capítulo 22.

### 4.2.4. TIMER

Registros	Función
CCTL0 = CCIE	para habilitar la interrupción
TA0CTL =	ponemos las config al timer A
TASSEL_2	Timer_A clock source select * con 2 especificamos el SMCLK
ID_0	Input divider. These bits select the divider for the input clock.  0 es division por 1
MC_1	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power 1 es Up mode: the timer counts up to TACCR0
TACCR0 = VALOR	FRECUENCIA DE ADQUISICIÓN!!**

\*\* la frec de adqui se calcula haciendo  $1000000 / (\text{valor} + 1)$  con el reloj interno del msp a 8MHz. Como nosotros lo tenemos a 16MHz es hacer  $2 \times 1000000 / (\text{valor} + 1)$  (VER 12.2.3.1)

\* A grandes rasgos, el msp tiene un DCO (oscilador interno) que se controla por software. A partir de este se config otros: el master clock (MCLK), el sub-main clock (SMCLK), entre otros. El tic del reloj define la vel de operación. Hay procesos que requieren más tiempos, otros menos y a veces puede ser útil tener operaciones a distintas velocidades. Ejemplo: si tenemos una señal que varía lento el adc10 no necesita operar rápido, entonces podemos usar un reloj más lento como el SMCLK. Y usamos el reloj más rápido MCLK para ejecutar el código en si.

En la librería bsp.h definimos la frecuencia del MCLK y la configuración del SMCLK! Pongo igual la tabla de los registros importantes del MCLK:

Registros	Función
BCSCTL1 = CALBC1_16MHZ	Use 16Mhz cal data for DCO
DCOCTL = CALDCO_16MHZ	Use 16Mhz cal data for DCO

y del SMCLk lo configuramos para que el DCO sea su source clock y divisor /1 (ver bsp.h)  
Para mas info ver cap 12.

#### 4.2.5. Interrupción del timer

La primera linea de las interrupciones es propia de la sintaxis de la interrupción.

Registros	Función
ADC10CTL0 &= ~ENC;	Apaga las config de este registro
ADC10CTL1 &= ~0 xFFFF;	Otra forma de decir que apague las config de este registro
CONSEQ_0	Conversion sequence mode select 0 – > Single-channel-single-conversion
INCH_1	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions 1 es A1 (BIT1)
ENC + ADC10SC	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.

### 4.3. Comunicación con la SD

Utiliza SPI. Esto está programado en las librerías hal spi. Si bien no es necesario un gran conocimiento sobre este tipo de comunicación, se puede encontrar información en el Capítulo 16 del manual del MSP.

SD Pin	miniSD Pin	microSD Pin	Name	I/O	Logic	Description
1	1	2	nCS	I	PP	SPI Card Select [CS] (Negative Logic)
2	2	3	DI	I	PP	SPI Serial Data In [MISO]
3	3		VSS	S	S	Ground
4	4	4	VDD	S	S	Power
5	5	5	CLK	I	PP	SPI Serial Clock [SCLK]
6	6	6	VSS	S	S	Ground
7	7	7	DO	O	PP	SPI Serial Data Out [MOSI]
8	8	8	NC nIRQ	- O	- OD	Unused (memory cards) Interrupt (SDIO cards) (Negative Logic)
9	9	1	NC	-	-	Unused
	10		NC	-	-	Reserved
	11		NC	-	-	Reserved

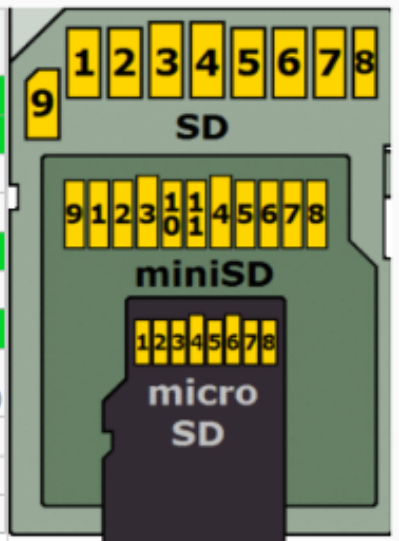


Figura 17: puertos sd

Las líneas importantes de esto son: CS, MOSI, MISO, SCKL, MISO, Ground y Power.

## 5. Lectura de la tarjeta SD

### 5.1. Lectura de SD y pasaje a decimal

Para correr un programa en C en nuestras computadoras con Windows tuvimos que descargar MinGW (compilador de C). Para activar esto en la terminal de la compu, seguimos los pasos del mismo programa. Es decir, con la terminal vamos a donde esté la carpeta, ingresamos y corremos los siguientes comandos:

```
set_distro_paths.bat
open_distro_window.bat
```

(**OBS1** : estos pasos se repiten cada vez que se quiere correr un programa en C, i.e. cuando se cierra y se vuelve a abrir la terminal)

Luego, vamos a la carpeta donde está el archivo ReadSd.c y corremos los siguientes comandos:

```
gcc ReadSd.c
gcc -o read ReadSd.c
```

Con realizar esto una única vez es suficiente. Veremos que en la carpeta donde está ReadSd.c creó el ejecutable **read**. (**OBS:** cada vez que se cambia el programa ReadSd.c hay que ejecutar devuelta estas dos últimas líneas!!)

#### 5.1.1. El programa...

El programa lee una tarjeta SD sin formato y guarda dos archivos, uno con los datos en hexadecimal (hex.dat) y otro en decimal (dec.dat). El que vamos a usar después es el dec.dat

**OBS!!:** la línea 27 puede cambiar dependiendo de la compu (unidad E, unidad D, F, etc.)

**obs2:** Notar que en la línea 7 hay un 3000. Esas son la cantidad de sectores que vamos a leer cada vez que se ejecute el read! Si queremos cambiarlo no olvidar la **OBS1**. Notar que hay que cambiar en todos los lugares que diga 3000.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 //int cant_sectores = 4000;
7 #define BUFFER_SIZE 512*3000 // Bytes a leer
8 // Datos = bytes/2
9 int i;
10 int can; //Para diferenciar si se grabo 1 canal o 2
11 char aux_sector[10];
12 // Read from one sector
13 int main(int argc, char *argv[]){
14     char *letras; // Numero de medicion
```

```

15  letras = argv[1];
16
17  can = 1;          // Cantidad de canales
18  FILE *volume;     // Origen
19  FILE *out_hex;    // Datos en hex
20  FILE *out_dec;    // Datos en dec
21  int k = 0;
22  long long sector;
23  char buf[BUFFER_SIZE] = {0};
24
25  sprintf(aux_sector, "%s", letras);
26  sector = atol(aux_sector);      // Numero de sector inicial
27  volume = fopen("\\\\.\\D:", "r");
28  //volume = fopen("ejemplo1.IMA", "r");
29  //volume = fopen("output.raw", "r");
30  setbuf(volume, NULL);          // Disable buffering
31  if(!volume){
32      printf("Cant open Drive\n");
33      return 1;
34  }
35  if(fseek(volume, sector*3000*512, SEEK_SET) != 0){ // Sector-aligned ↵
36      printf("Can't move to sector\n");
37      return 2;
38  }
39  // read what is in sector and put in buf
40  // Lee BUFFER_SIZE cantidad de datos y los guarda en buf
41  fread(buf, sizeof(*buf), BUFFER_SIZE, volume);
42  fclose(volume);
43
44  // A partir de aca es la conversion y guardado de datos
45  out_hex = fopen("hex.dat", "w");
46  int lsb = 0; //El primer byte es msb
47
48  // Guardo dos bits concatenados por fila: msb+lsb
49  for(k=0; k<BUFFER_SIZE; k++){
50      if(lsb){
51          fprintf(out_hex, "%x\n", buf[k]);
52          lsb = 0;
53      }else{
54          fprintf(out_hex, "%x", buf[k]);

```

```

55         lsb = 1;
56     }
57 }
58 fclose(out_hex);
59
60 char aux[8] = {0};
61 char aux_msb[2] = {0};
62 long int valor_msb, valor_lsb, valor;
63 char * pEnd;
64 out_hex = fopen("hex.dat", "r");
65 out_dec = fopen("dec.dat", "w");
66 int j = 1;
67 for(k=1; k<(BUFFER_SIZE)/2+1; k++){
68     fscanf(out_hex, "%s", aux);
69     int len = strlen(aux);
70     if(len>2){ //medio tecnico, hay una diferencia en como escribe el ←
        hexadecimal que lee si es mayor o menor a 512
71         char *aux_2 = &aux[len-3]; //Me quedo con los 3 digitos que ←
            importan, al ppio pone muchas f's
72         strncpy(aux_msb, aux_2, 2); //Los dos primeros son los msb
73         aux_msb[2] = '\0';
74     }else{
75         char *aux_2 = &aux[len-2]; //Me quedo con los 2 digitos que ←
            importan
76         strncpy(aux_msb, aux_2, 1); //Los dos primeros son los msb
77         //aux_msb[2] = '\0';
78         aux_msb[1] = '\0';
79     }
80     char *aux_lsb = &aux[len-1]; //El ultimo digito es el lsb
81     valor_msb = pow(2,2)*strtol(aux_msb, &pEnd, 16); //Paso de hex a ←
        decimal y hago bitshift
82     valor_lsb = strtol(aux_lsb, &pEnd, 16); //Paso de hex a decimal
83     valor = valor_msb+valor_lsb;
84     if(can == 1){ // Si 1 canal escribo 1 columna
85         fprintf(out_dec, "%ld\n", valor);
86     }else{ // Si dos canales los escribo en columnas ←
        distintas
87         if(j){
88             fprintf(out_dec, "%ld\t", valor);
89             j = 0;
90         }else{

```

```

91         fprintf(out_dec, "%ld\n", valor);
92         j = 1;
93     }
94 }
95 }
96 fclose(out_hex);
97 fclose(out_dec);
98 return 0;
99 }

```

## 5.2. Reconstrucción del Audio, Python

### 5.2.1. Audio total...

Ahora procedemos a realizar la reconstrucción de audio. Para ello utilizamos el código de python *Leo\_muchos\_datos.py*. El código se ejecuta en la terminal correspondiente (o **Spyder**) y se genera un archivo *.wav* que contendrá al audio. Si se descomenta, hace el gráfico de la amplitud en función del tiempo.

Algunas aclaraciones:

- el programa tiene que estar en la misma carpeta que el ReadSd.c (porque utiliza el ejecutable read!)
- Dentro del for esta la función call que es la que llama al ejecutable read j veces
- la variable cantidad es la cantidad de veces (j) que va a llamar al ejecutable read (Esto lo hacemos porque las compus no nos dejan leer mas de 4000 sectores de una. Entonces tenemos que hacerlo por partes!)
- La variable ttot es el tiempo que queremos tener de audio (en segundos).
- La variable es tmed se modifica en caso de que se cambie la cantidad de sectores en el programa ReadSd.c (deben coincidir en **ambos** programas!!)
- dependiendo de cuanto tiempo sea ttot, el programa puede demorar unos minutos en compilar (se puede optimizar)
- Por las, el dec.dat final que está en la carpeta corresponde a la última llamada del read. Osea, corresponde a los últimos 3000 sectores leídos.

```

1 import matplotlib.pyplot as plt
2 from scipy.io import wavfile as wav
3 from scipy import signal
4 import numpy as np

```

```

5 import glob
6 from subprocess import call
7 from io import StringIO
8 from matplotlib import pyplot
9 import sys
10
11 frec_adq = 23000
12
13 programa = 'read' # Programa compilado a partir de readSd.c
14
15 ttot = 400 #en segundos
16 tmed = 3000.0 * 256 /frec_adq # Duracion de cada medicion: cantidad de ←
    sectores medidos por el tamano del sector
17 #tiene que coincidir el numero de sectores en ambos programas!!!
18
19 #print(tmed) con 3000 sect es 34 seg aprox
20
21 cantidad = int(ttot/tmed) # Cantidad de iteraciones
22 print('cant de iteraciones ' + str(cantidad))
23
24 Datos = [] #es vector de vectores
25
26 for j in range(cantidad):
27     # Ejecuta el programa de lectura de 1
28     call(["./" + programa, str(int(j))]) #Agrega el argumento en el main de ←
        ReadSd.c
29     dat_file = glob.glob('dec.dat')[0]
30     datos1 = np.loadtxt('dec.dat')
31     Datos.append(datos1)
32
33
34 datos = []
35 for i in range(len(Datos)):
36     for j in range(len(Datos[i])):
37         datos.append(Datos[i][j])
38
39 print('cant de datos leidos ' + str(len(datos)))
40
41 sonido_normalizado = datos/np.max(np.abs(datos))
42 sonido_normalizado = sonido_normalizado * 32767
43 sonido_normalizado = sonido_normalizado.astype('int16')

```



```

44 wav.write('audio.wav',int(frec_adq),sonido_normalizado)
45
46
47 ##### Paso datos a txt para analisis (esto mucho no sirve porque np.loadtxt ←
    no me deja leer muchos datos)
48 #np.savetxt('todos_los_datos.txt', datos, delimiter = '\t')
49
50 ##### Grafico
51 """
52 tiempo = np.arange(0, len(datos)*1/frec_adq, 1/frec_adq)
53
54 plt.plot( tiempo,datos, 'o-')
55 plt.tick_params(labelsize=13)
56 plt.legend()
57 plt.xlabel('Tiempo [s]', size = 13)
58 #ax1.set_ylabel('Intensidad [s.u.]')
59 plt.ylabel('Amplitud', size = 13)
60 plt.grid(True)
61 plt.show()
62 """

```

### 5.2.2. Audio entrecortado...

Si reconstruimos un audio de mucho tiempo (más de 10 minutos), el .wav puede ser muy pesado. Una posible solución es recortar el audio total cada x minutos y analizarlos por separado. El código que usamos en este caso se llama *leo\_muchos\_sectores\_audioCortado.py*. Aca se tienen que definir las variables **ttot**, **tmed**, **tdiv**, **min\_recorte**.

Aclaraciones:

- el programa tiene que estar en la misma carpeta que el ReadSd.c
- Dentro del for esta la función call que es la que llama al ejecutable read j veces
- la variable cantidad es la cantidad de veces (j) que va a llamar al ejecutable read (Esto lo hacemos porque las compus no nos dejan leer mas de 4000 sectores de una. Entonces tenemos que hacerlo por partes!)
- La variable ttot es el tiempo que queremos tener de audio (en segundos).
- **tdiv** cantidad de audios finales !!!
- **min\_recorte** tiempo que quiero cada audio (en minutos)

- La variable es tmed se modifica en caso de que se cambie la cantidad de sectores en el programa ReadSd.c (deben coincidir en **ambos** programas!!)
- dependiendo de cuanto tiempo sea ttot, el programa puede demorar unos minutos en compilar (se puede optimizar)