Diseño Basado en Microprocesadores

Tema 2. Microcontroladores

- 2.1. Introducción a los microcontroladores
- 2.2. Entradas/Salidas Digitales
- 2.3. Temporizadores
- 2.4. Excepciones
- 2.5. Conversión Analógica/Digital
- 2.6. Comunicación serie RS232C
- 2.7. Teclado, conversión D/A y sonido
- 2.8. Interfaz I2C

2.7. Teclado, conversión D/A y sonido

2.7. Teclado, conversión D/A y sonido

2.7.1. Teclado matricial 4 x 4

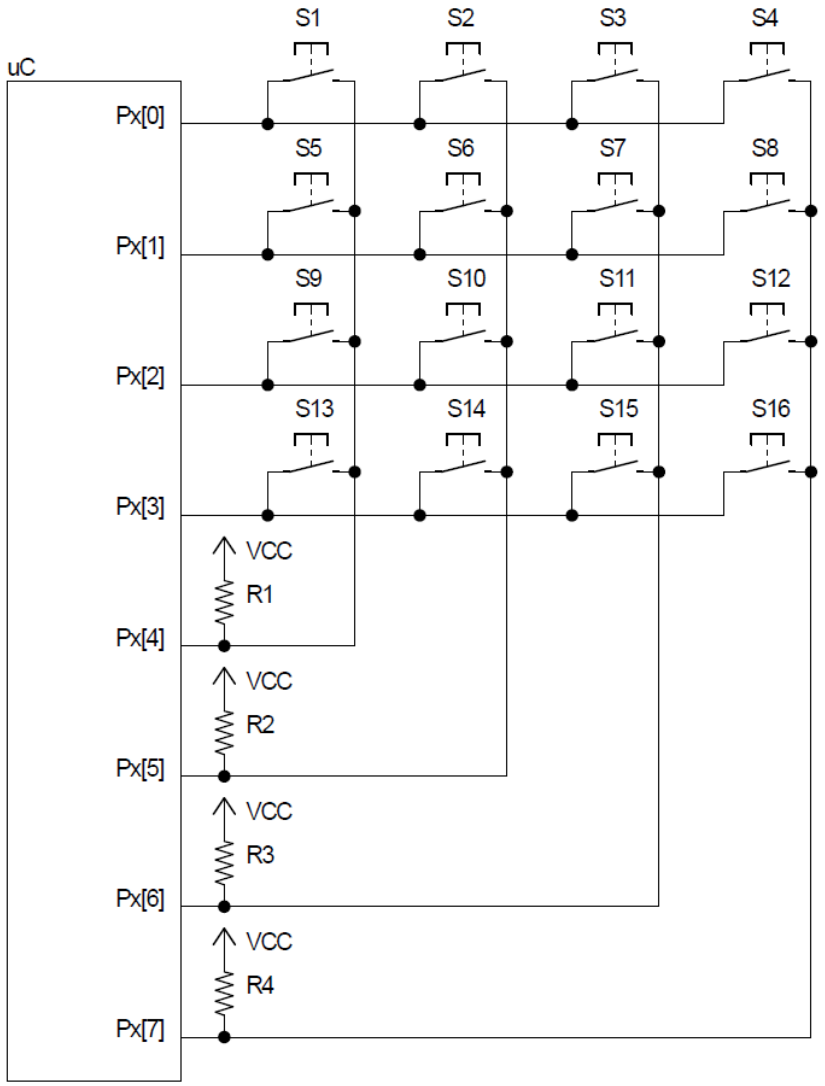
Teclado

- Conjunto de botones
- Normalmente, conectamos cada pulsador/interruptor a un pin
- Si hay muchos pulsadores/interruptores esto puede consumir muchos pines
- Podría necesitarse un µC más grande y caro sólo para tener suficientes pines

Solución: Teclado matricial

- Conectar los pulsadores/interruptores en matriz
- Complica un poco el software, pero el ahorro en hardware compensa

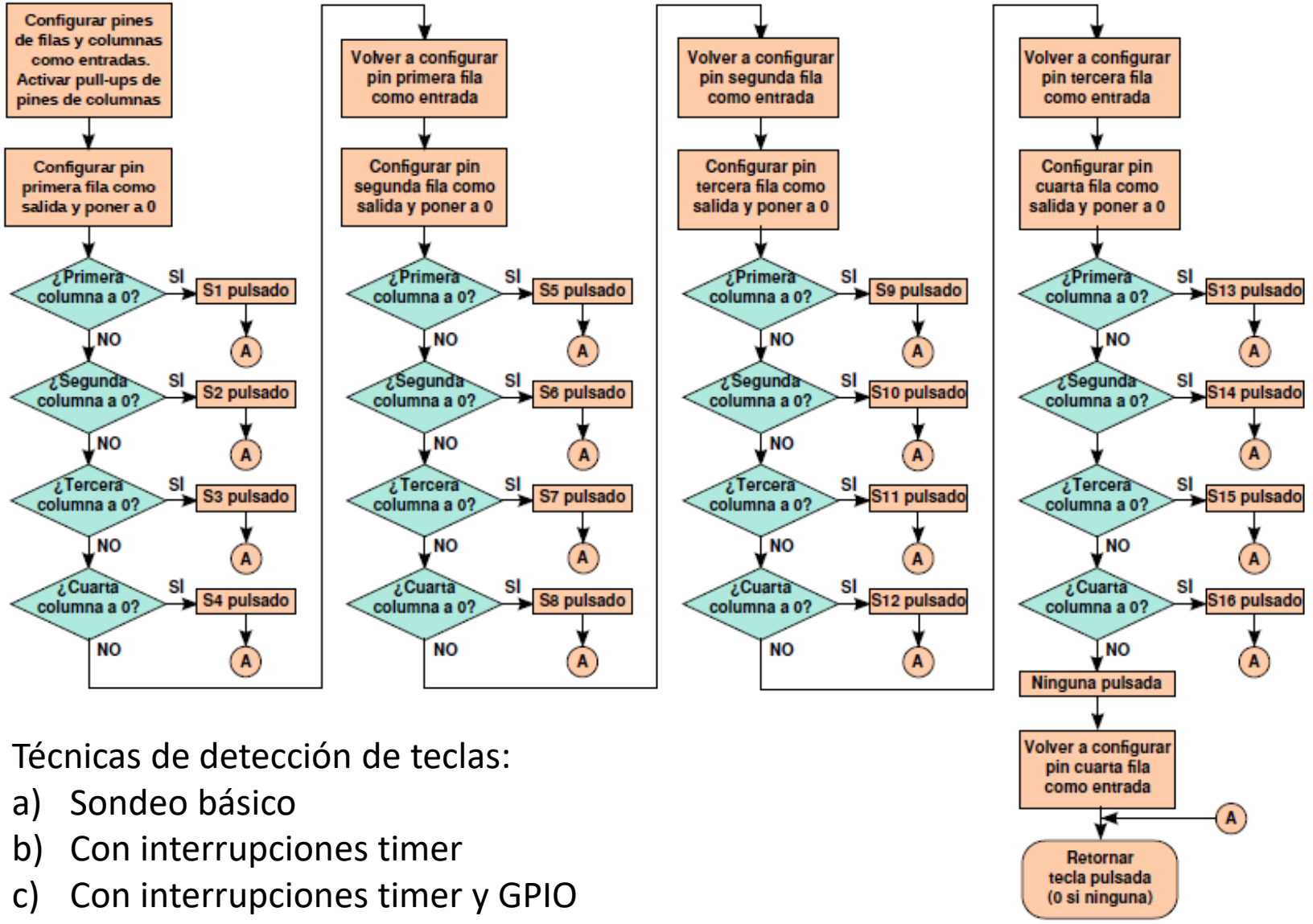
2.7.2. Esquema de conexiones



- Si el microcontrolador tiene la posibilidad de activar pull-ups internos, pueden ahorrarse las resistencias externas R1 ... R4 para las columnas
- Las filas no llevan resistencias

Teclado	Color cable	Puerto-bit	Pin conector
Fila 0	Negro	P0[5]	J5-16
Fila 1	Blaco	P0[7]	J5-17
Fila 2	Gris	P0[9]	J5-18
Fila 3	Morado	P0[21]	J5-22
Columna 0	Azul	P2[0]	J4-5
Columna 1	Verde	P2[1]	J4-7
Columna 2	Amarillo	P2[14]	J5-45
Columna3	Naranja	P2[19]	J5-46

2.7.3. Proceso de barrido



Técnicas de detección de teclas:

a) Sondeo básico
b) Con interrupciones timer
c) Con interrupciones timer y GPIO

2.7.4. Fichero teclado4x4.h (1)

```
/* Símbolos para referirse a los números de los puertos a los que está conectado el teclado */

#define TEC4X4_PUERTO_FILAS          LPC_GPIO0
#define TEC4X4_PUERTO_COLUMNAS       LPC_GPIO2

/* Símbolos para referirse a las líneas de puerto a las que están conectadas las distintas filas
 * y columnas del teclado */

#define TEC4X4_PIN_FILA_0      5u
#define TEC4X4_PIN_FILA_1      7u
#define TEC4X4_PIN_FILA_2      9u
#define TEC4X4_PIN_FILA_3      21u

#define TEC4X4_PIN_COLUMNA_0   0u
#define TEC4X4_PIN_COLUMNA_1   1u
#define TEC4X4_PIN_COLUMNA_2   14u
#define TEC4X4_PIN_COLUMNA_3   19u
```

Fichero teclado4x4.h (2)

```
/* Máscaras de selección de pines de filas y columnas */
#define TEC4X4_MASCARA_FILA_0      (1u << (TEC4X4_PIN_FILA_0))
#define TEC4X4_MASCARA_FILA_1      (1u << (TEC4X4_PIN_FILA_1))
#define TEC4X4_MASCARA_FILA_2      (1u << (TEC4X4_PIN_FILA_2))
#define TEC4X4_MASCARA_FILA_3      (1u << (TEC4X4_PIN_FILA_3))
#define TEC4X4_MASCARA_TODAS_LAS_FILAS      (TEC4X4_MASCARA_FILA_0 | \
                                    TEC4X4_MASCARA_FILA_1 | \
                                    TEC4X4_MASCARA_FILA_2 | \
                                    TEC4X4_MASCARA_FILA_3)

#define TEC4X4_MASCARA_COLUMNA_0   (1u << (TEC4X4_PIN_COLUMNA_0))
#define TEC4X4_MASCARA_COLUMNA_1   (1u << (TEC4X4_PIN_COLUMNA_1))
#define TEC4X4_MASCARA_COLUMNA_2   (1u << (TEC4X4_PIN_COLUMNA_2))
#define TEC4X4_MASCARA_COLUMNA_3   (1u << (TEC4X4_PIN_COLUMNA_3))
#define TEC4X4_MASCARA_TODAS_LAS_COLUMNAS    (TEC4X4_MASCARA_COLUMNA_0 | \
                                     TEC4X4_MASCARA_COLUMNA_1 | \
                                     TEC4X4_MASCARA_COLUMNA_2 | \
                                     TEC4X4_MASCARA_COLUMNA_3)
```

Fichero teclado4x4.h (3)

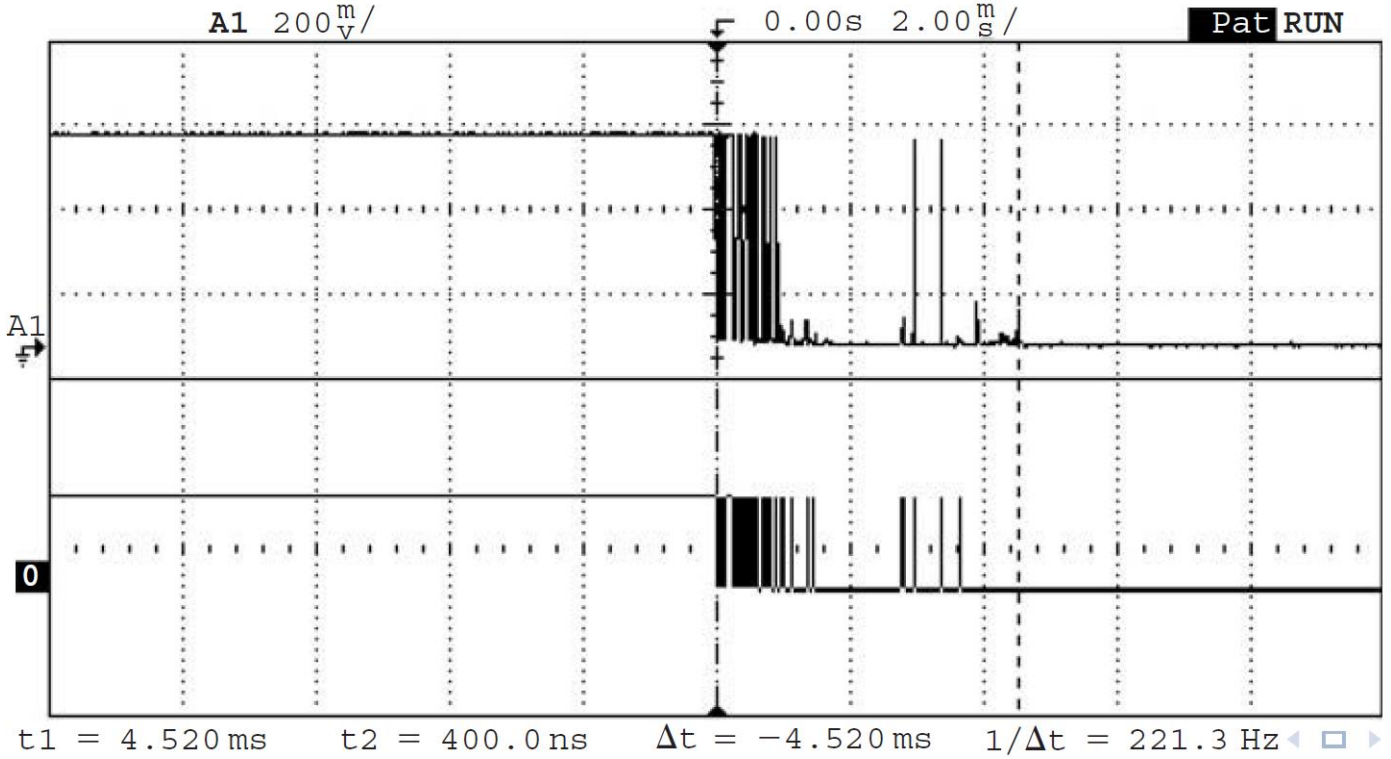
```
/* Las funciones de teclado retornan 0 para indicar ninguna tecla pulsada */
#define TEC4X4_NINGUNA_PULSADA      0

/* Timer usado por el teclado */
#define TEC4X4_TIMER     LPC_TIM3

/* Tiempo en microsegundos que el teclado debe permanecer en un determinado
 * estado para considerarse estable y no en situación de rebotes */
#define TEC4X4_TIEMPO_FILTRO_REBOTES_US 20000u

/* Tiempo en microsegundos entre la activación de una línea y la lectura de
 * otra (versión básica sin timer ni interrupción) */
#define TEC4X4_TIEMPO_ESTABILIZACION_US 10u

/*===== Constantes para los dos tipos de teclados matriciales ==================*/
#define TEC4X4_CONEX_ARRIBA 0
#define TEC4X4_CONEX_ABAJO 1
```

2.7.5. Biblioteca funciones teclado4x4_sondeo_basico

```
void tec4x4_inicializar(void);
char tec4x4_leer(void);
char tec4x4_esperar_estable(void);
char tec4x4_esperar_pulsacion_con_rebotes(void);
char tec4x4_esperar_pulsacion(void);
void tec4x4_vaciar_buffer(void);
uint32_t tec4x4_caracteres_en_buffer(void);
void tec4x4_leer_cadena(char *ptr_buffer, uint32_t
tamano_buffer);
void tec4x4_leer_cadena_numeros(char *ptr_buffer,
uint32_t tamano_buffer);
```

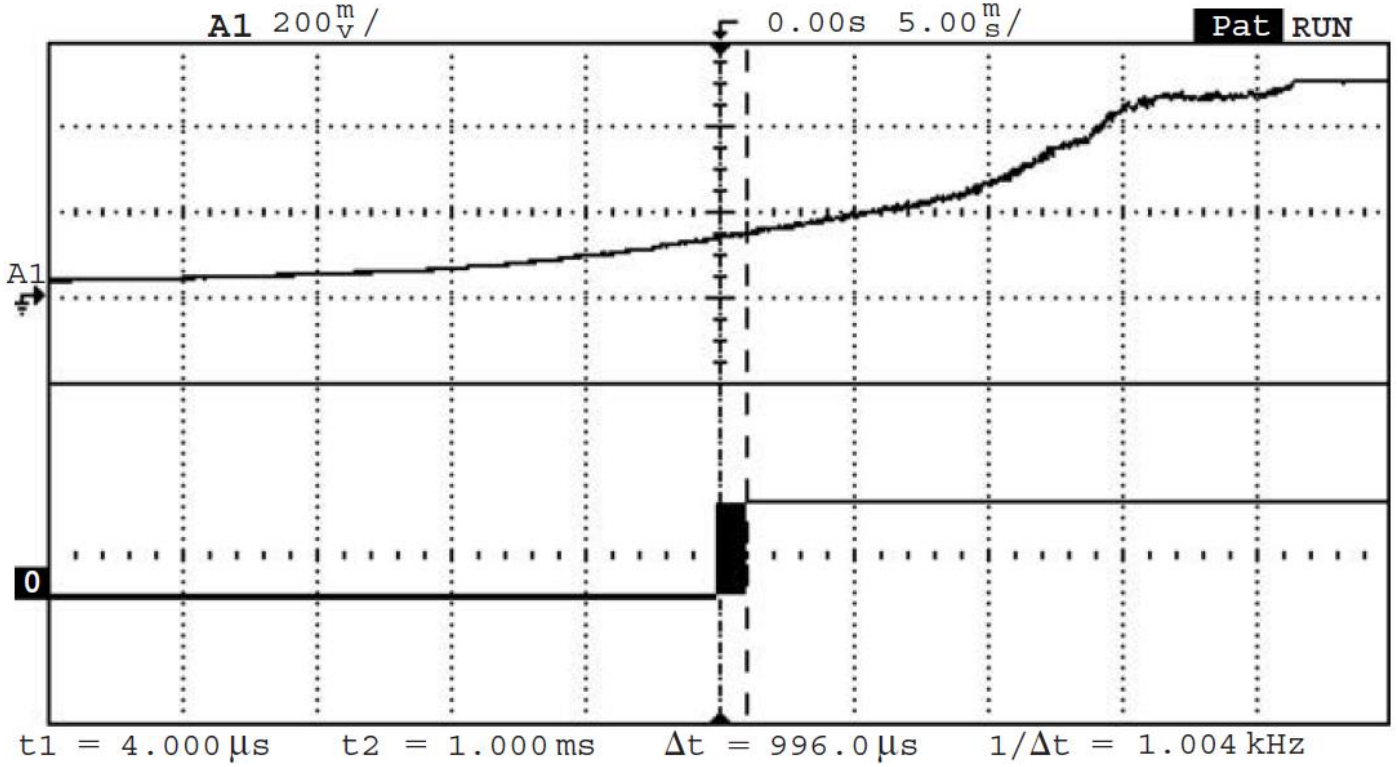
2.7.6. Rebotes en pulsadores e interruptores (1)

- Al activar o desactivar un interruptor o pulsador los contactos no se cierran o abren limpiamente
- Al cerrarse, los contactos rebotan, abriéndose y cerrándose varias veces antes de cerrarse finalmente
- Al abrirse, también se producen aperturas y cierres intermitentes antes de la apertura definitiva



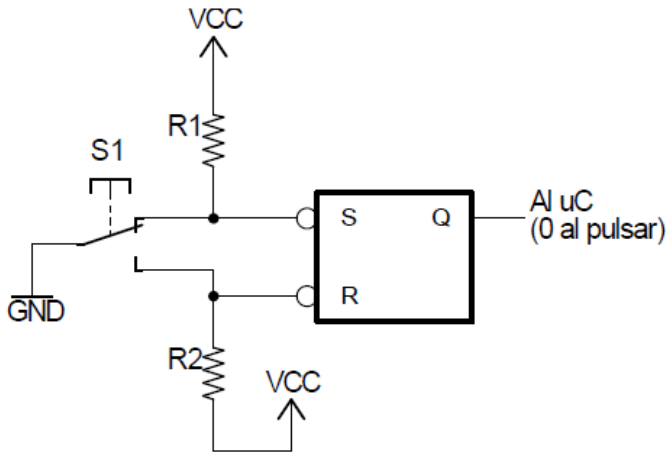
2.7.6. Rebotes en pulsadores e interruptores (2)

- Algunos pulsadores de goma conductora no tienen muchos rebotes, pero sí un cambio gradual de resistencia
- La señal pasa mucho tiempo en la zona de incertidumbre de la entrada digital



Por hardware

Interruptor de doble acción y biestable RS



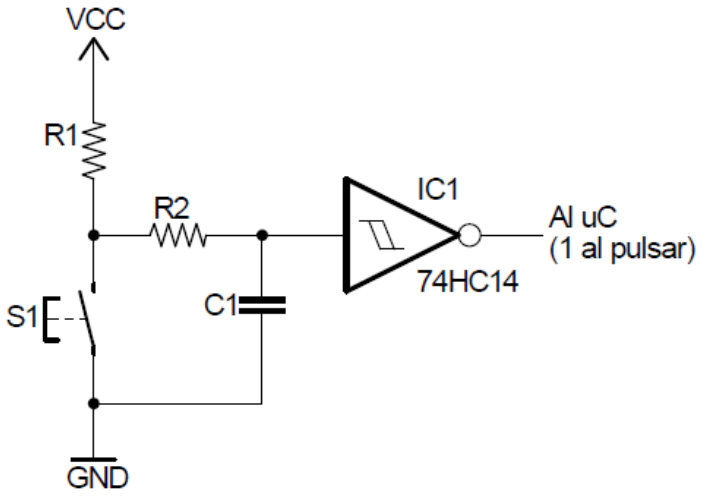
Ventajas

- Es el sistema antirrebotes más efectivo

Desventajas

- Los interruptores/ pulsadores de doble acción son más caros
- Se necesita añadir un integrado para el biestable o puertas

Filtro RC con Inversor trigger Schmitt



Ventajas

- Buena eliminación de rebotes
- Algunos µC, como el LPC4088, tienen pines con histéresis así que sólo es necesario un condensador adicional

Desventaja

- Sigue necesitando hardware adicional

Por software: Sondeo básico (1)

```
char tec4x4_esperar_estable(void){
    /* La variable estado_actual indicará cómo se encuentra el teclado en un instante dado.
     * La variable estado_ultimo_cambio indicará a qué estado cambió el teclado la vez anterior
     * que se detectó un cambio */

    char estado_actual;
    char estado_ultimo_cambio;

    /* Las dos variables se inicializan con el estado actual del teclado */

    estado_actual = estado_ultimo_cambio = tec4x4_leer();

    /* La cuenta en microsegundos del timer usado por el teclado se pone a 0.
     * (El timer ya está contando en microsegundos) */

    timer_poner_contador_a_0(TEC4X4_TIMER);
```

Por software: Sondeo básico (2)

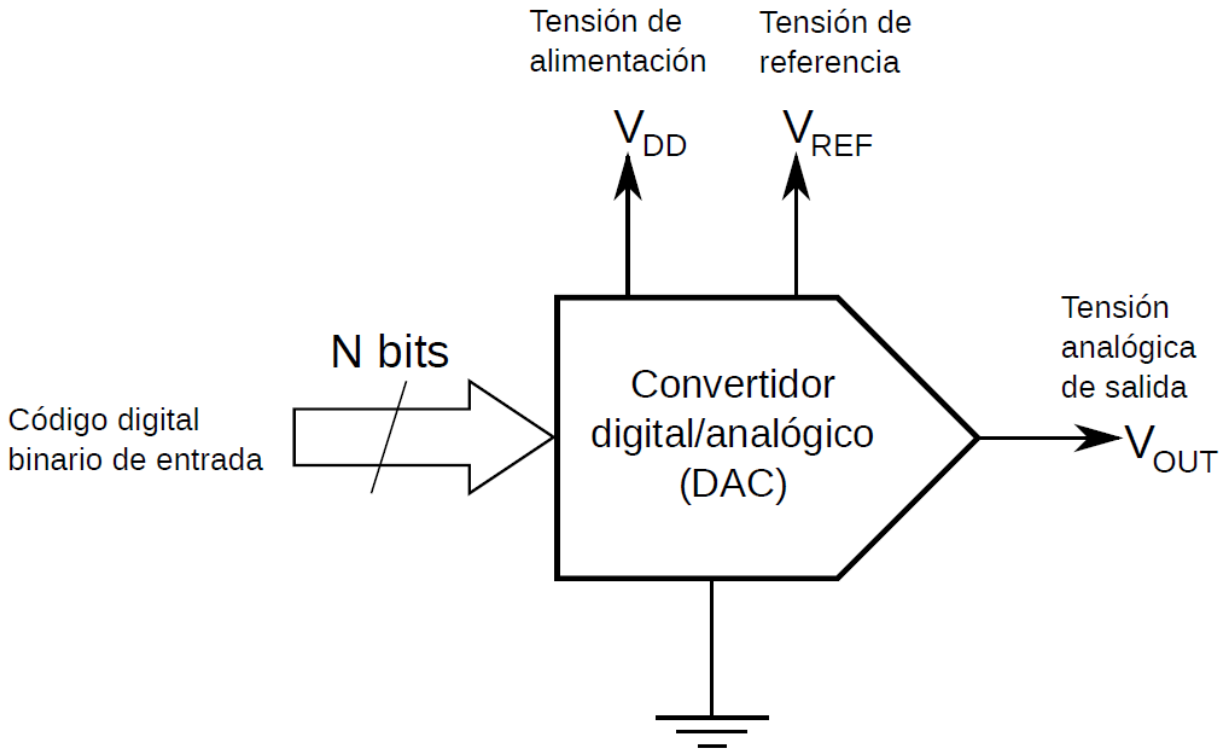
```
/* Mientras el contador en microsegundos no alcance TEC4X4_TIEMPO_FILTRO_REBOTES_US,
 * es decir, mientras el teclado no permanezca estable TEC4X4_TIEMPO_FILTRO_REBOTES_US ...  */
while (timer_leer(TEC4X4_TIMER) < TEC4X4_TIEMPO_FILTRO_REBOTES_US){
        /* Leer el estado del teclado en este instante */
        estado_actual = tec4x4_leer();
        /* Si el estado actual no coincide con el registrado tras el cambio anterior, el teclado no
         * está permaneciendo estable. Entonces, se pone a 0 el contador de tiempo estable
         * y estado_ultimo_cambio actualiza al estado actual */
        if (estado_actual != estado_ultimo_cambio)
        {   timer_poner_contador_a_0(TEC4X4_TIMER);
            estado_ultimo_cambio = estado_actual;
        }
    }

/* Cuando el teclado permanece sin cambios durante TEC4X4_TIEMPO_FILTRO_REBOTES_US
 * termina el bucle anterior y se sale de la función indicando el estado estable alcanzado  */
    return estado_actual;
}
```

Por software: Sondeo básico (3)

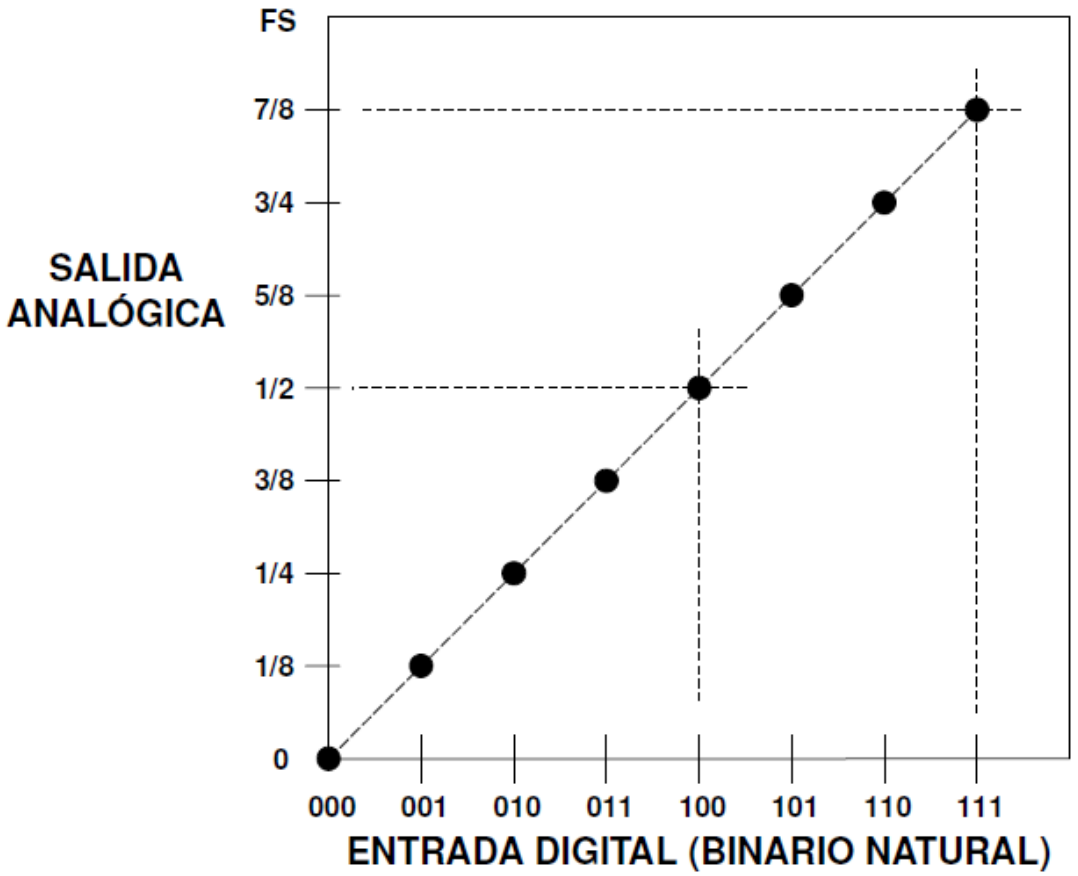
```
char tec4x4_esperar_pulsacion(void){
    char tecla;

    /* Esperar mientras se detecte una pulsación  */
    while (tec4x4_esperar_estable() != TEC4X4_NINGUNA_PULSADA);

    /* Luego, esperar mientras no se detecte una pulsación */
    do{
        tecla = tec4x4_esperar_estable();

    }while (tecla == TEC4X4_NINGUNA_PULSADA);

    /* Retornar la pulsación detectada */
    return tecla;
}
```

2.7.7. Convertidor digital/analógico (DAC) (1)

- Un convertidor digital/analógico, o DAC, permite transformar un código digital binario de entrada en una tensión analógica de salida
- Símbolo de un DAC unipolar (la tensión sólo puede ser >=0)



2.7.7. Convertidor digital/analógico (DAC) (2)



Curva de transferencia de un DAC unipolar de 3 bits

- La tensión de referencia determina la tensión de fondo de escala (FS)
- Tensión 1 LSB:
  - Cambio que se produce en la salida cuando cambia el LSB del código de entrada
  - También se denomina cuanto (Q) del convertidor

$$V_{LSB} = Q = \frac{V_{FS}}{2^N}$$

2.7.8. Características DAC del LPC4088

- Convertidor de cadena de resistencias
- 10 bits de resolución
- 1 canal de salida a través del pin P0[26]/AD0[3]/DAC_OUT/U3_RXD
- Rango de salida analógica de 0 a VREFP (VREFP: tensión de referencia aplicada externamente)
- Tasa de conversión de hasta 1 MHz
- Posibilidad de DMA
- Puede apagarse para reducir el consumo

2.7.9. Registros del DAC (1)

Nombre	Descripción	Tipo	Reset	Offset
CR	Registro del convertidor DA. Este registro contiene el valor digital a ser convertido en analógico y un bit de control de alimentación.	R/W	0x00	0x000
CTRL	Registro de control DAC. Este registro controla el acceso DMA y las operaciones del timer.	R/W	0x00	0x004
CNTVAL	Registro del valor del contador DAC. Este registro contiene el valor de recarga para el acceso DMA/interrupción del timer del DAC.	R/W	0x00	0x008

2.7.9. Registros del DAC. Registro de control CR (2)

Table 686: D/A Converter Register (CR - address 0x4008 C000) bit description

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved. Read value is undefined, only zero should be written.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the DAC_OUT pin (with respect to V_{SSA}) is VALUE × V_{REFP}/1024.	0
16	BIAS		Settling time The settling times noted in the description of the BIAS bit are valid for a capacitance load on the DAC_OUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time. One or more graphs of load impedance vs. settling time will be included in the final data sheet.	0
		0	The settling time of the DAC is 1 µs max, and the maximum current is 700 µA. This allows a maximum update rate of 1 MHz.	
		1	The settling time of the DAC is 2.5 µs and the maximum current is 350 µA. This allows a maximum update rate of 400 kHz.	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

2.7.10. Inicialización del DAC

1. Seleccionar función DAC_OUT en el pin P0[26]/AD0[3]/DAC_OUT/U3_RXD

```
void dac_inicializar(void){
  LPC_IOCON->P0_26 = (1u << 16) | 2;
   /* FUNC = 2 => función DAC_OUT
    * ADMODE = 0 => selecciona modo analógico
    * DACEN = 1 => bit 16 a 1 => activa el DAC
    */
}
```

2.7.11. Conversiones controladas por software

1. En el registro CR del DAC:
   - Colocar el dato digital a convertir (10 bits) en los bits 6 a 15
   - Poner el bit 16 a 0 o a 1 según el tiempo de establecimiento requerido

```
void dac_convertir(uint16_t dato_a_convertir){
    ASSERT(dato_a_convertir < 1023,
    "dato_a_convertir debe estar entre 0 y 1023");
    LPC_DAC->CR = dato_a_convertir << 6;
}
```

2.7.12. Tensión de salida en función del dato digital de entrada

$$V_{DACOUT} = \frac{VALUE_{DAC}}{2^N} V_{REFP}$$

donde:

VREFP tensión de ref. aplicada a la patilla VREFP: 3.3V

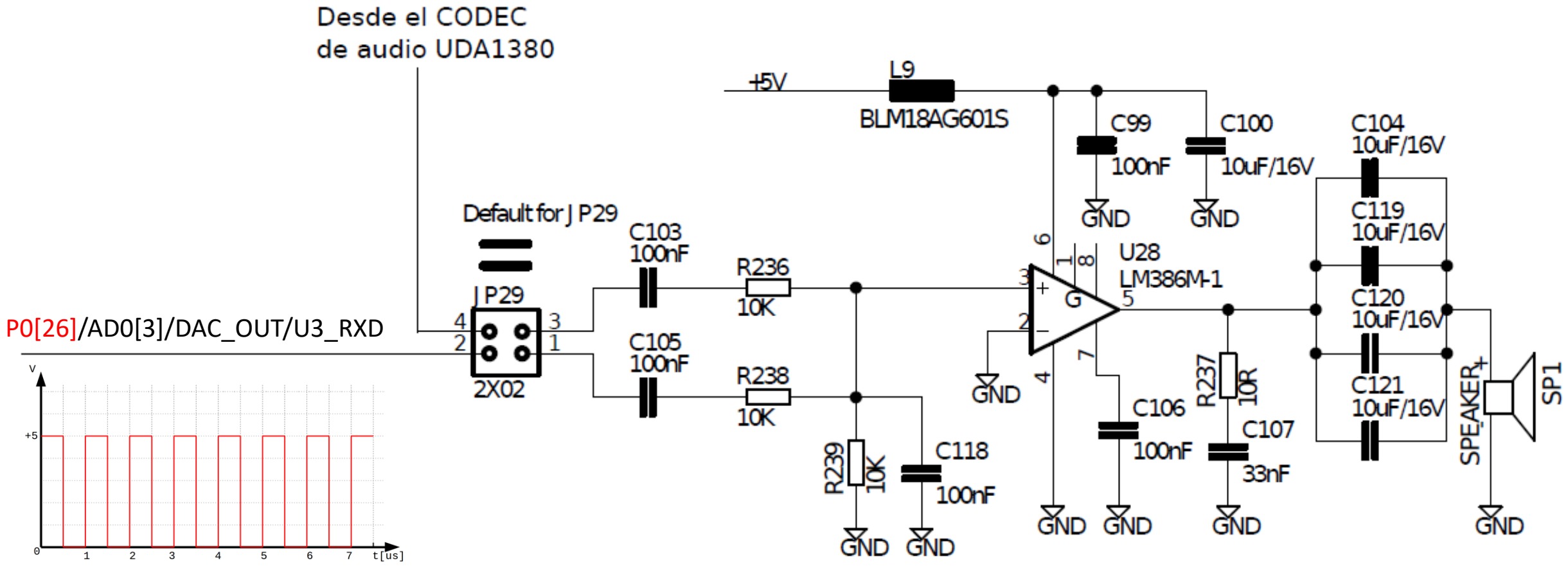
N número de bits de resolución del DAC: 10

VALUEDAC dato digital escrito en el registro CR del DAC

Sustituyendo los valores:

$$V_{DACOUT} = \frac{3.3V\ VALUE_{DAC}}{1024} (V)$$

2.7.13. Generación de sonido en el LPC4088. Esquema



Graves
20 Hz – 50 Hz

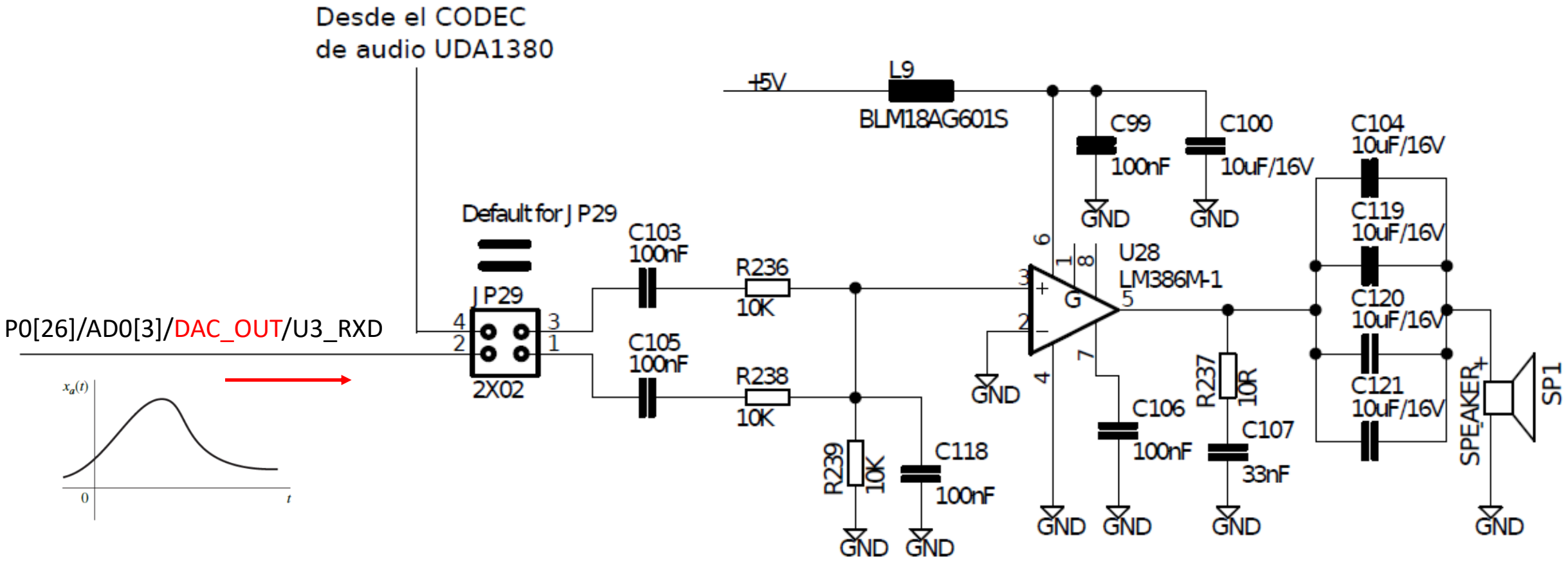
a

Agudos
16 kHz - 20 kHz

2.7.14. Generación de sonido en el LPC4088. Función sonido

```
void sonido_emitir_pitido(uint32_t frecuencia, uint32_t duracion_ms){
    uint32_t T_2_us;
    uint32_t N_periodos;
    uint32_t i;

    T_2_us = 500000/frecuencia;
    N_periodos = duracion_ms*frecuencia/1000;
    LPC_GPIO0->DIR |= 1u << 26;

    timer_inicializar(TIMER1);
    timer_iniciar_ciclos_us(TIMER1, T_2_us);

    for (i = 0; i < N_periodos; i++){
        LPC_GPIO0->SET = 1u << 26;
        timer_esperar_fin_ciclo(TIMER1);
        LPC_GPIO0->CLR = 1u << 26;
        timer_esperar_fin_ciclo(TIMER1);
    }
}
```

2.7.15. Ejemplo de aplicación: generación de sonido con el DAC



2.7.16. Sonido con DAC usando un timer para obtener la frec. de muestreo (1)

main

```
extern const sonido_t sonido_de_prueba;
int main(void){
  /* Inicializar la reproducción de sonidos. */
  sonido_inicializar();

  /* Reproducir el sonido repetidamente. */
  while (TRUE){
    sonido_reproducir(&sonido_de_prueba);

    /* Esperar a que termine la reproducción del sonido.*/
    while (sonido_en_reproduccion()) {}
  }
}
```

sonido_de_prueba.c

sonido_con_dac_timer_e_interrupcion.c

```
sonido_inicializar();

sonido_reproducir(&sonido_de_prueba);

sonido_en_reproduccion();
```

TIMER0

NVIC

dac_lpc40xx.c

```
dac_inicializar(void);
dac_convertir(uint16_t);
```

Sonido de timbre de teléfono:

- Frecuencia de muestreo: 8 kHz
- 29970 muestras
- Duración: 29970/8000 = 3.75 s

Período TIMER0:
T=1/F = 1/8000Hz = 0,000125s = 125 µs

2.7.16. Sonido con DAC usando un timer para obtener la frec. de muestreo (2)

```
/*****************************************************************************
 * \file    sonido_con_dac_timer_e_interrupcion.c
 * \brief   Funciones de reproducción de sonido usando el DAC del LPC40xx.
 *          La frecuencia de muestreo es de 8000 Hz y se consigue mediante
 *          interrupciones periódicas del timer 0.
 */

#include <LPC407x_8x_177x_8x.h>
#include "sonido.h"
#include "dac_lpc40xx.h"
#include "timer_lpc40xx.h"
#include "tipos.h"
#include "error.h"

/* Variables globales estáticas usadas por la función de interrupción
 * del timer 0 que lleva a cabo la reproducción de sonido. */
static const int16_t *ptr_muestras = NULL;
static int32_t contador_muestras = 0;
static bool_t reproduciendo = FALSE;
```

2.7.16. Sonido con DAC usando un timer para obtener la frec. de muestreo (3)

```
/*****************************************************************************
 * \brief   Inicializar el sistema de reproducción de sonido. Debe ser llamada
 *          antes del primer uso de la función sonido_reproducir. */
void sonido_inicializar(void){
  /* Inicializar las variables globales del módulo. */
  ptr_muestras = NULL;
  contador_muestras = 0;
  reproduciendo = FALSE;
  /* Inicializar el DAC. */
  dac_inicializar();
  /* Habilitar las interrupciones del timer 0.*/
  NVIC_ClearPendingIRQ(TIMER0_IRQn);
  NVIC_EnableIRQ(TIMER0_IRQn);
  __enable_irq();
  /* Preparar timer0 para generar interrupciones cada 125us PERO SIN PONERLO EN MARCHA AUN*/
  timer_inicializar(TIMER0);
  LPC_TIM0->TCR = 0;
  LPC_TIM0->PC = 0;
  LPC_TIM0->TC = 0;
  LPC_TIM0->PR = PeripheralClock/1000000 - 1;
  LPC_TIM0->MR0 = 125 - 1;
  LPC_TIM0->MCR = 3;
}
```

2.7.16. Sonido con DAC usando un timer para obtener la frec. de muestreo (4)

```
/*****************************************************************************
 * \brief       Iniciar la reproducción de un sonido.
 * \param[in]   snd     Puntero a una estructura de tipo sonido_t que indica el
 *                      sonido a reproducir. */
void sonido_reproducir(const sonido_t *snd){
  /* Comprobar que snd y snd->muestras son válidos. */
    ASSERT(snd != NULL, "Puntero snd nulo.");
    ASSERT(snd->muestras != NULL, "Puntero snd->muestras nulo.");

  /* Inicializar el puntero ptr_muestras a partir de snd->muestras
   * y contador_muestras a partir del número de muestras.
   * La variable reproduciendo se pone a TRUE para indicar que se está
   * reproduciendo un sonido. */
  ptr_muestras = snd->muestras;
  contador_muestras = snd->numero_muestras;
  reproduciendo = TRUE;

  /* Poner en marcha el timer. */
    LPC_TIM0->TCR = 1;
}
```

2.7.16. Sonido con DAC usando un timer para obtener la frec. de muestreo (5)

```
/**********************************************************************
 * \brief       Manejador de interrupción del timer 0. */
void TIMER0_IRQHandler(void){
  /* Borrar los flag de petición de interrupción. */
  LPC_TIM0->IR = 1;
  NVIC_ClearPendingIRQ(TIMER0_IRQn);
  if (ptr_muestras == NULL) return;

  /* Enviar al DAC la siguiente muestra de sonido y avanzar el
   * puntero a la siguiente y decrementar el contador de muestras. */
  LPC_DAC->CR = (*ptr_muestras + 512) << 6; //las muestras están entre -512 y +512
  ptr_muestras++;
  contador_muestras--;
  /* Si se llegó al final del sonido, invalidar el puntero ptr_muestras,
   * parar el timer y poner FALSE la variable "reproduciendo". */
  if (contador_muestras == 0){
    ptr_muestras = NULL;
    LPC_TIM0->TCR = 0;
    reproduciendo = FALSE;
  }
}
```

2.7.17. Sonido con Códec de audio UDA1380 (1)

- Un códec de audio es un dispositivo capaz de codificar o decodificar un flujo digital de audio
- La velocidad máxima de muestreo que podemos utilizar con el códec UDA1380 es de 44.1 kHz y 16 bits que equivaldría a la calidad de un CD de audio
- Formatos de sonido:
  - WAV (Waveform audio file format) es un formato de audio digital con o sin compresión de datos desarrollado por Microsoft e IBM
  - MP3 es un formato de compresión de audio digital que usa un algoritmo con pérdida para conseguir un menor tamaño de archivo
- Bibliotecas para MP3:
  - i2c_lpc40xx.c
  - uda1380.c y i2s_lpc40xx.c
  - salida_con_audio_uda1380.c
  - libmad (conjunto de funciones para MP3)
  - reproductor_mp3.c

UM10562 Cap. 23: LPC408x/407x I2S interface
Data Sheet UDA1380

2.7.17. Sonido con Códec de audio UDA1380 (2)