



Laboratorio 9:

## **E/S por DMA y bus IIS**

reproducción/grabación de sonido con un Audio Codec

Programación de sistemas y dispositivos

**José Manuel Mendías Cuadros**

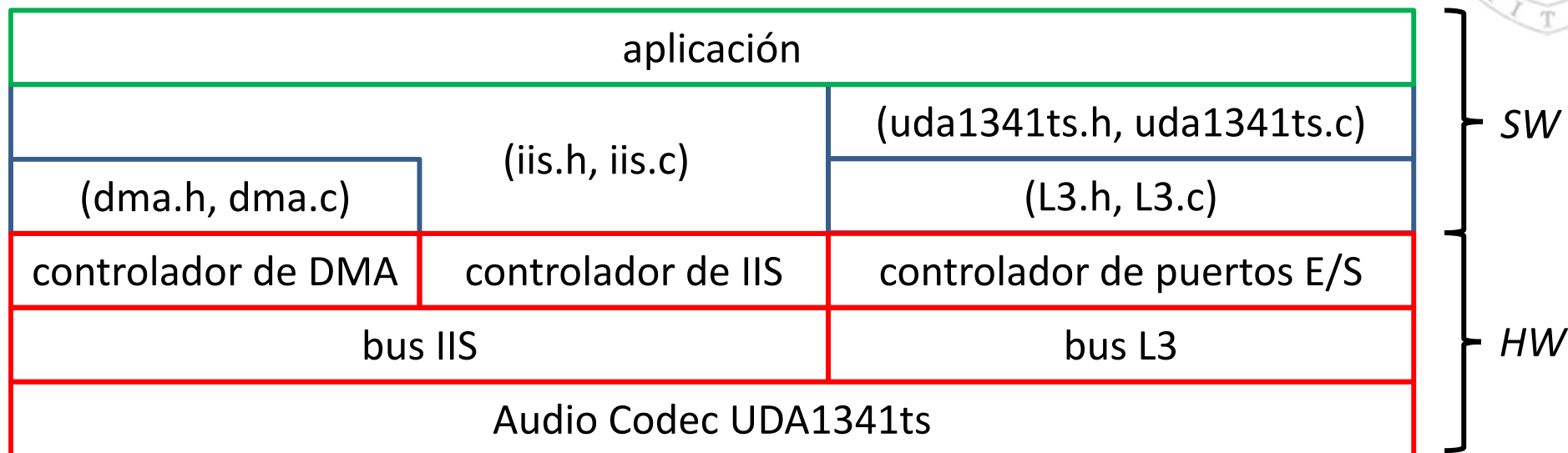
*Dpto. Arquitectura de Computadores y Automática  
Universidad Complutense de Madrid*

# Presentación

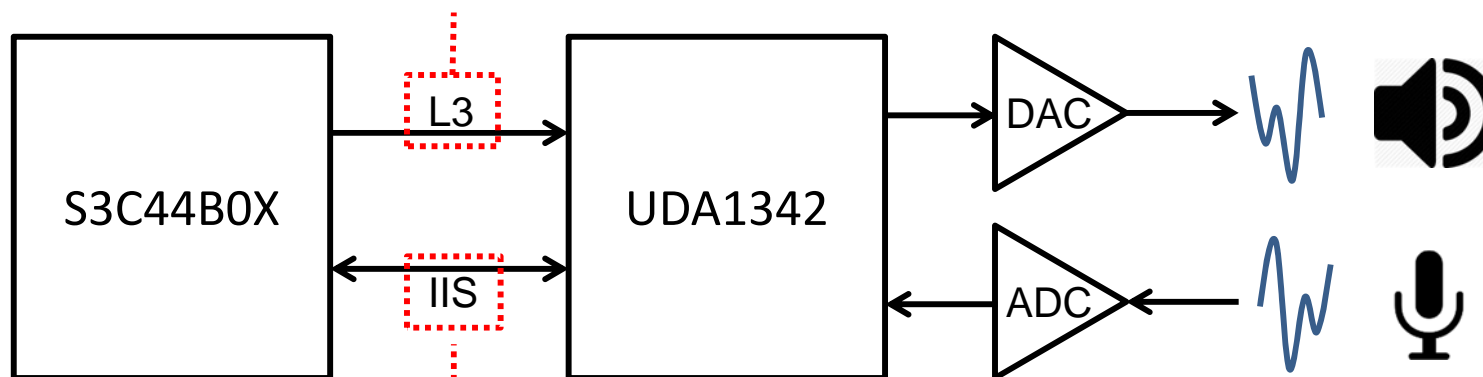


- Desarrollar 4 capas de firmware para grabación/reproducción de sonido muestreado usando un audio codec conectado a buses L3 e IIS:
  - Una para el envío de bytes por pooling por bus L3:
    - Envío: `L3_putByte`
    - Inicialización: `L3_init`
  - Otra para el control de operación de un audio codec conectado a un bus L3:
    - Envío de comandos: `uda1341ts_mute` / `uda1341ts_on` / `uda1341ts_off` / `uda1341ts_setvol`
    - Recuperación de estado: `uda1341ts_status` / `uda1341ts_getvol`
    - Inicialización: `uda1341ts_init`
  - Otra para el envío/recepción de muestras de audio por bus IIS:
    - Envío/recepción de 1 muestra por pooling: `iis_putSample` / `iis_getSample`
    - Envío/recepción de n muestras por polling/DMA: `iis_playwav` / `iis_recwav`
    - Control del estado de DMA: `iis_pausewav` / `iis_continuewav`
    - Recuperación del estado de DMA: `iis_statuswav`
    - Inicialización: `iis_init`
  - Otra para inicialización de DMA:
    - Inicialización: `bdma0_init` / `bdma0_open` / `bdma0_close`

# Presentación



Para envío de información de control



Para el envío y/o recepción de sonido muestreado

# Sonido muestreado



## ■ Frecuencia de muestreo ( $f_s$ )

- 8.000 Hz    Teléfono
- 16.000 Hz    Voz IP
- 32.000 Hz    mini DV
- 44.100 Hz    audio CD
- 48.000 Hz    DVD
- 96.000 Hz    HD-DVD

## ■ Número de canales:

- 1 (mono) / 2 (stéreo)

## ■ Representación de la muestra:

- Enteros (C2): 8...32 bits/muestra

Trabajaremos con sonido muestreado:

- **$f_s = 16\text{KHz}$ , estéreo, 16b**
- tasa transferencia = 500 Kbps
- 1s sonido = 62.5 KB

## ■ Tasa de transferencia (b/s)

- $f_s \times (\text{num. canales}) \times (\text{anchura de datos})$

## ■ Tamaño del sonido (B)

- $(\text{duración}) \times f_s \times (\text{num. canales}) \times (\text{anchura de datos}) / 8$

# Bus IIS

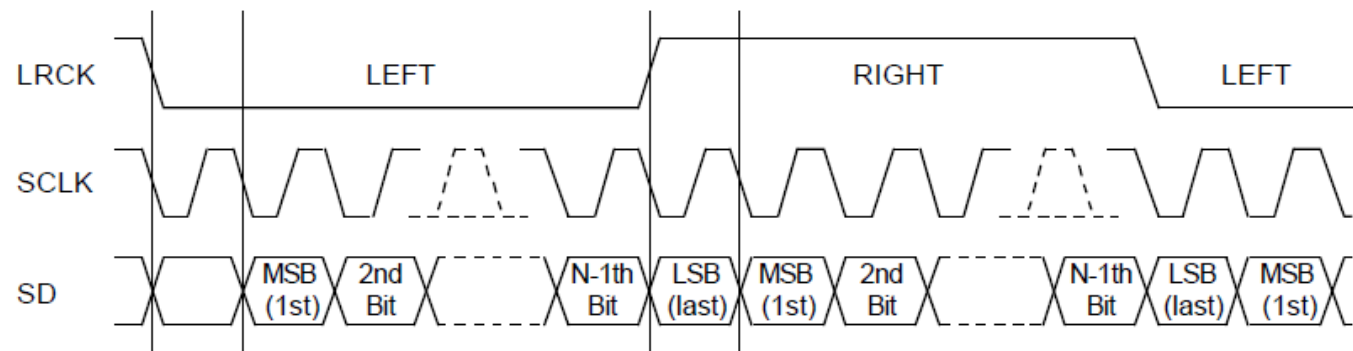


- **IIS** (Integrated Interchip Sound) es un **bus serie síncrono** para la transmisión de audio estéreo entre dispositivos digitales.
  - Tiene 2 líneas unidireccionales de datos serie: **IISDI** (entrada) y **IISDO** (salida).
    - Permite el envío/recepción simultánea de datos de sonido.
    - Otros datos (p.e. control) deben transmitirse por líneas aparte.
  - Tiene 3 líneas de reloj: **SCLK** (transmisión de datos serie), **LRCLK** (selección de canal izquierdo/derecho), **CODECLK** (reloj principal)
  - Todos los datos transmitidos están codificados en C2 (MSB first) con un número de bits no definido (depende del emisor/receptor).
    - Los datos serán truncados o extendidos con 0 según convenga en el emisor/receptor.
- **Protocolo:**
  - La **transmisión de datos es continua**
  - El maestro genera las señales de reloj determinando así implícitamente  $f_s$
  - Emisor y receptor transmiten las señales de datos de audio alternando muestras del canal izquierdo y derecho.

# Bus IIS



- Su funcionamiento se controla por medio de 3 señales de reloj:
  - **CODECLK**: determina la frecuencia de funcionamiento del controlador
  - **LRCK**: indica la muestra (izquierda o derecha) que se está transmitiendo
  - **SCLK**: sincroniza la transferencia serie de cada uno de los bits de la muestra.



IIS-BUS FORMAT (N=8 or 16)

- Sus frecuencias **se definen siempre relativas a la fs**:
  - **LRCK**:  $f_s$
  - **SCLK**:  $(\text{bits/muestra}) \times (\text{número de canales}) \times f_s$
  - **CODECLK**:  $256 f_s / 384 f_s / 512 f_s$

Trabajaremos con frecuencias:

- LRCK =  $f_s$
- SCLK =  $32 f_s$
- CODECLK =  $256 f_s$



# Controlador de IIS

## configuración

- **Formato de transmisión:** 16b compatible con IIS
  - $IISMOD[3] = 1$  16b
  - $IISMOD[4] = 0$  IIS compatible
- **Frecuencia de SCLK:** 32fs
  - $IISMOD[1:0] = 1$
- **Frecuencia de CODECLK:** 256fs
  - $IISMOD[2] = 0$
- **Frecuencia de muestreo:**  $f_s = 16$  KHz (calidad Voz IP)
  - $n = 64 \text{ MHz} / (256 \times 16 \text{ KHz}) = 15,6$
  - $IISPSR[7:4]/IISPSR[3:0] = 7$  division factor = 16
- **Polaridad de LRCLK:** 0 para canal izquierdo
  - $IISMOD[5] = 0$
- **Generación de LRCLK en inactividad:** sí
  - $IISCON[3] = 0$  en transmisión
  - $IISCON[2] = 0$  en recepción

# Controlador de IIS

## operación por pooling (i)



- El controlador puede configurarse para:
  - Enviar y recibir simultáneamente (o no) muestras por pooling
    - Idóneo para efectos de audio en tiempo real
  - Solo enviar muestras por DMA
    - Idóneo para reproducción de sonido
  - Solo recibir muestras por DMA
    - Idóneo para grabación de sonido
- Cada muestra son 2 datos
  - Correspondientes a los canales izquierdo y derecho que se van alternando.
- Para **recibir una muestra** por pooling:
  - Esperar a que haya al menos 2 datos en la FIFO Rx (mientras  $\text{IISFCON}[3:0] < 2$ )
  - Leer 2 veces de IISFIF
- Para **enviar una muestra** por pooling:
  - Esperar a que haya al menos 2 huecos en la FIFO Tx (mientras  $\text{IISFCON}[7:4] > 6$ )
  - Escribir 2 veces en IISFIF





# Controlador de IIS

## operación por pooling (ii)

- **Modo:** maestro en modo transmisión y recepción
  - IISMOD[8] = 0 master mode
  - IISMOD[7:6] = 3 modo transmisión y recepción
  - IISCON[1] = 1 prescaler habilitado
  - IISCON[0] = 1 interfaz IIS habilitado
- **FIFO transmisión:** habilitada y accedida por pooling
  - IISFCON[9] = 1 Tx FIFO enable
  - IISFCON[11] = 0 normal access mode
  - IISCON[5] = 0 Tx DMA request disable
- **FIFO recepción:** habilitada y accedida por pooling
  - IISFCON[8] = 1 Rx FIFO enable
  - IISFCON[10] = 0 normal access mode
  - IISCON[4] = 0 Rx DMA request disable

# Driver de controlador de IIS

## iis.h



```
#ifndef __IIS_H__
#define __IIS_H__

#include <common_types.h>

#define IIS_DMA      (1)
#define IIS_POLLING  (2) } Nemotécnicos para identificar el modo de envío/recepción de muestras

void iis_init( uint8 mode );

inline void iis_putSample( int16 ch0, int16 ch1 );
inline void iis_getSample( int16 *ch0, int16 *ch1 );
void iis_play( int16 *buffer, uint32 length, uint8 loop );
void iis_rec( int16 *buffer, uint32 length );
void iis_pause( void );
void iis_continue( void );
uint8 iis_status( void );
void iis_playwawFile( uint8 *fileAddr );

#endif
```

# Driver de controlador de IIS

**iis.c**



```
static uint8 iomode;
```

```
void iis_init( uint8 mode )
```

```
{
```

```
    iomode = mode;
```

```
    if( mode == IIS_POLLING )
```

```
    {
```

```
        IISPSR = ...;
```

```
        IISMOD = ...;
```

```
        IISFCON = ...;
```

```
        IISCON = ...;
```

```
    }
```

```
    ...
```

```
}
```

*Configura el controlador de IIS para operar por pooling*

# Driver de controlador de IIS

## iis.c



```
inline void iis_putSample( int16 ch0, int16 ch1 )
{
    while( ... );
    IISFIF = ch0;
    IISFIF = ch1;
}

void iis_play( int16 *buffer, uint32 length, uint8 loop )
{
    uint32 i;
    int16 ch1, ch2;

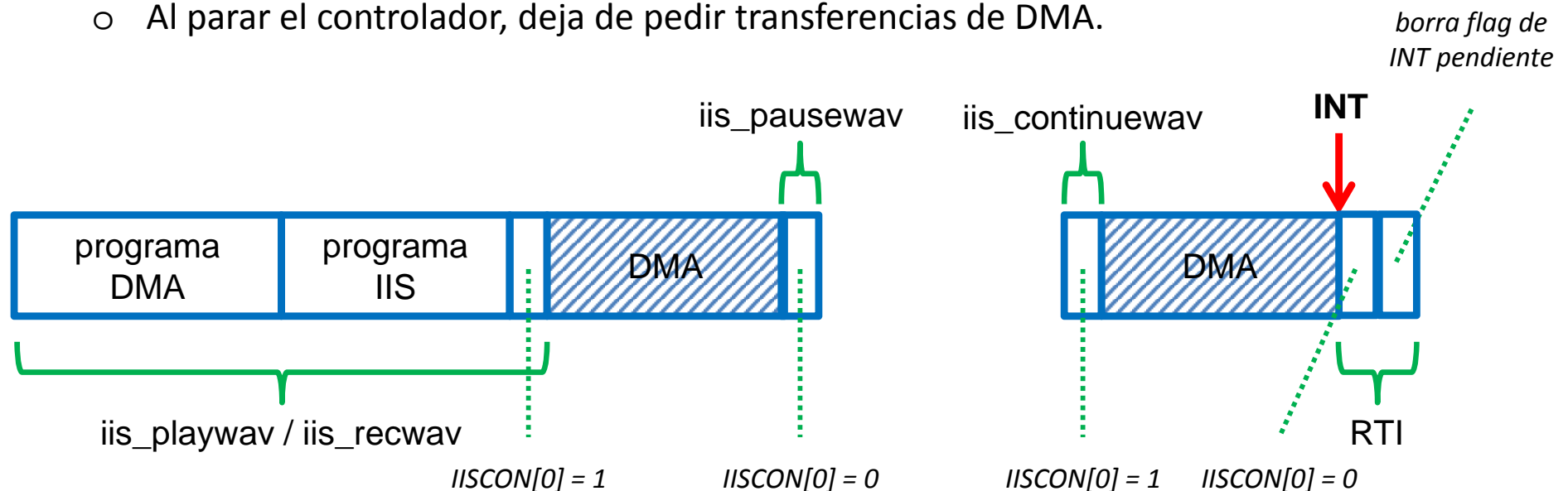
    if( iomode == IIS_POLLING )
        for( i=0; i<length/2; )
        {
            ch1 = buffer[i++];
            ch2 = buffer[i++];
            iis_putSample( ch1, ch2 );
        }
    ...
}
```



# Controlador de IIS

## operación por DMA (i)

- Para **enviar/recibir muestras** por DMA
  1. Configurar el controlador de DMA con las características de la transmisión.
  2. Configurar el controlador de IIS para que comience a solicitar peticiones de DMA
  3. Cuando la transferencia termina, el controlador de DMA interrumpe
  4. La RTI de DMA, debe parar al controlador de IIS para que deje de hacer peticiones
- Además, para **parar/reanudar la transmisión** de muestras por DMA
  - $IISCON[0] = 0/1$
  - Al parar el controlador, deja de pedir transferencias de DMA.





# Controlador de IIS

## operación por DMA: solo recepción

- **Modo:** maestro en modo recepción
  - IISMOD[8] = 0 master mode
  - IISMOD[7:6] = 1 modo recepción
  - IISCON[1] = 1 prescaler habilitado
  - IISCON[0] = 1 interfaz IIS habilitado
- **FIFO transmisión:** deshabilitada
  - IISFCON[9] = 0 Tx FIFO deshabilitada
  - IISFCON[11] = 0 normal access mode
  - IISCON[5] = 0 Tx DMA request deshabilitada
- **FIFO recepción:** habilitada y accedida por DMA
  - IISFCON[8] = 1 Rx FIFO habilitada
  - IISFCON[10] = 1 DMA access mode
  - IISCON[4] = 1 Rx DMA request habilitada



# Controlador de IIS

## operación por DMA: solo transmisión

- **Modo:** maestro en modo transmisión
  - IISMOD[8] = 0 master mode
  - IISMOD[7:6] = 2 modo transmisión
  - IISCON[1] = 1 prescaler habilitado
  - IISCON[0] = 1 interfaz IIS habilitado
- **FIFO transmisión:** habilitada y accedida por DMA
  - IISFCON[9] = 1 Tx FIFO habilitada
  - IISFCON[11] = 1 DMA access mode
  - IISCON[5] = 1 Tx DMA request habilitada
- **FIFO recepción:** deshabilitada
  - IISFCON[8] = 0 Rx FIFO deshabilitada
  - IISFCON[10] = 0 normal access mode
  - IISCON[4] = 0 Rx DMA request deshabilitada



# Controlador de IIS

## operación por DMA: en reposo

- **Modo:** maestro en modo recepción
  - IISMOD[8] = 0 master mode
  - IISMOD[7:6] = 0 no transfer
  - IISCON[1] = 0 prescaler deshabilitado
  - IISCON[0] = 0 interfaz IIS deshabilitado
- **FIFO transmisión:** deshabilitada
  - IISFCON[9] = 0 Tx FIFO deshabilitada
  - IISFCON[11] = 0 normal access mode
  - IISCON[5] = 0 Tx DMA request deshabilitada
- **FIFO recepción:** deshabilitada
  - IISFCON[8] = 0 Rx FIFO deshabilitada
  - IISFCON[10] = 0 normal access mode
  - IISCON[4] = 0 Rx DMA request deshabilitada



# Controlador DMA

## configuración



- Solo el BDMA0 permite transferencias por DMA entre memoria y el controlador de bus IIS.
  - BDICNT0[31:30] = 1 DMA source seleccion: IIS
  - BDCON0[3:2] = 0 DMA request enable
  - BDCON0[1:0] = 0 no command
  - BCICNT0[20] = 1 DMA enable
- Tamaño de datos: 16b
  - BDISRC0[31:30] = 1 half word
- Opciones de DMA:
  - BCICNT0[29:28] = 0 handshake mode
  - BCICNT0[27:26] = 1 unit transfer mode
  - BCICNT0[25:24] = 0 no on-the-fly mode
- Generación de interrupción: al terminar la transferencia DMA
  - BCICNT0[23:22] = 3



# Controlador DMA

## operación por DMA: solo transmisión

- **Sentido de la transferencia:** memoria -> IIS
  - $BDIDES0[31:30] = 1$  from external memory to internal peripheral
- **Dirección origen:**
  - $BDISRC0[29:28] = 1$  post-incrementada
  - $BDISRC0[27:0] = DIR$  dirección del buffer
- **Dirección destino:**
  - $BDIDES0[29:28] = 3$  fija
  - $BDIDES0[27:0] = 0x1D18010$  dirección de IISFIF
- **Número de bytes a transferir:**
  - $BCICNT0[19:0] = LENGTH$  tamaño del buffer
- **Autoreload:** según se defina
  - $BCICNT0[21] = X$



# Controlador DMA

## operación por DMA: solo recepción

- Sentido de la transferencia: IIS -> memoria
  - BDIDES0[31:30] = 2 from internal peripheral to memory
- Dirección origen:
  - BDISRC0[29:28] = 3 fija
  - BDISRC0[27:0] = 0x1D18010 dirección de IISFIF
- Dirección destino:
  - BDIDES0[29:28] = 1 post-incrementada
  - BDIDES0[27:0] = DIR dirección del buffer
- Número de bytes a transferir:
  - BCICNT0[19:0] = LENGTH tamaño del buffer
- Autoreload: desactivado
  - BCICNT0[21] = 0

# Driver de controlador de IIS

## iis.c



```
static void isr_bdma0( void ) __attribute__ ((interrupt ("IRQ")));

static uint8 flag;
static uint8 iomode;

void iis_init( uint8 mode )
{
    iomode = mode;

    ...
    if( mode == IIS_DMA )
    {
        IISPSR = ...;
        IISMOD = ...;
        IISFCON = 0;
        IISCON = 0;
        bdma0_init();
        bdma0_open( isr_bdma0 );
        flag = OFF;
    }
}
```

Configura el controlador de IIS para operar por DMA en modo reposo

# Driver de controlador de IIS

**iis.c**



```
void iis_rec( int16 *buffer, uint32 length )
{
    ...
    if( iomode == IIS_DMA )
    {
        while( flag != OFF );
        BDISRC0 = (1 << 30) | (3 << 28) | (uint32) &IISFIF;
        BDIDES0 = (2 << 30) | (1 << 28) | (uint32) buffer;
        BDCON0  = 0;
        BDICNT0 = (1 << 30) | (1 << 26) | (3 << 22) | (1 << 20) | length;
        IISMOD  = ...;
        IISFCON = ...;
        IISCON  = ...;
        flag = ON;
    }
}

static void isr_bdma0( void )
{
    IISCON &= ~1;
    flag = OFF;
    I_ISPC = BIT_BDMA0;
}
```

Configura el controlador de IIS para operar por DMA en modo solo recepción

# Driver de controlador de DMA

## `dma.h`



```
#ifndef __DMA_H__
#define __DMA_H__

#include <common_types.h>

void bdma0_init( void );
void bdma0_open( void (*isr)(void) );
void bdma0_close( void );

#endif
```

# Driver de controlador de DMA

**dma.c**



```
#include <s3c44b0x.h>
#include <s3cev40.h>
#include <dma.h>

extern void isr_BDMA0_dummy( void );

void bdma0_init( void )
{
    BDCON0 = 0;
    BDISRC0 = 0;
    BDIDES0 = 0;
    BDICNT0 = 0;
}

void bdma0_open( void (*isr)(void) )
{
    pISR_BDMA0 = ...; ..... instala la ISR argumento en la tabla virtual de vectores de IRQ
    I_ISPC      = ...; ..... borra flag de interrupción pendiente por fin de BDMA0
    INTMSK      &= ...; ..... desenmascara globalmente interrupciones e interrupciones por fin de BDMA0
}

void bdma0_close( void )
{
    INTMSK      |= ...; ..... enmascara interrupciones por fin de BDMA0
    pISR_BDMA1  = ...; ..... instala isr_BDMA0_dummy en la tabla virtual de vectores de interrupción
}
```

# Ficheros WAV



- Un fichero **WAV convencional**:
  - Dispone de una **cabecera** con información diversa
  - Las muestras se almacenan sin compresión alternando canales
    - En el chunk identificado como "data"
    - El número de bytes ocupado por las muestras se indica en los 4B siguientes al identificador
- Para **reproducir** en el audio codec un fichero WAV
  - Deberá estar precargado en memoria
  - Tendrá estar muestreado a 16KHz en estéreo y 16 bits por muestras
  - Deberá enviarse por bus IIS al audio codec:
    - Ignorando la cabecera
    - Al ser un formato no comprimido las muestras podrán transmitirse directamente
- Para **cargar un WAV en memoria**
  - Desde la consola del depurador teclear el comando

```
restore <fichero> binary <dir_memoria>
```





# Programa principal

## efectos en dominio temporal: echo & fade

```

void iis_playwawFile( uint8 *fileAddr )
{
    uint32 size;

    while( !( fileAddr[0] == 'd' &&
               fileAddr[1] == 'a' &&
               fileAddr[2] == 't' &&
               fileAddr[3] == 'a' ) ) } Busca el identificador del "chunk"
        fileAddr++;
    fileAddr += 4; ..... Salta el identificador del "chunk"

    size = (uint32) fileAddr[0];
    size += (uint32) fileAddr[1] << 8;
    size += (uint32) fileAddr[2] << 16;
    size += (uint32) fileAddr[3] << 24; } Extrae el número de bytes que ocupan las muestras
                                         (el fichero está ordenación little endian)
    fileAddr += 4; ..... Salta la indicación de número de bytes

    iis_play( (int16 *)fileAddr, size, OFF );
}

```

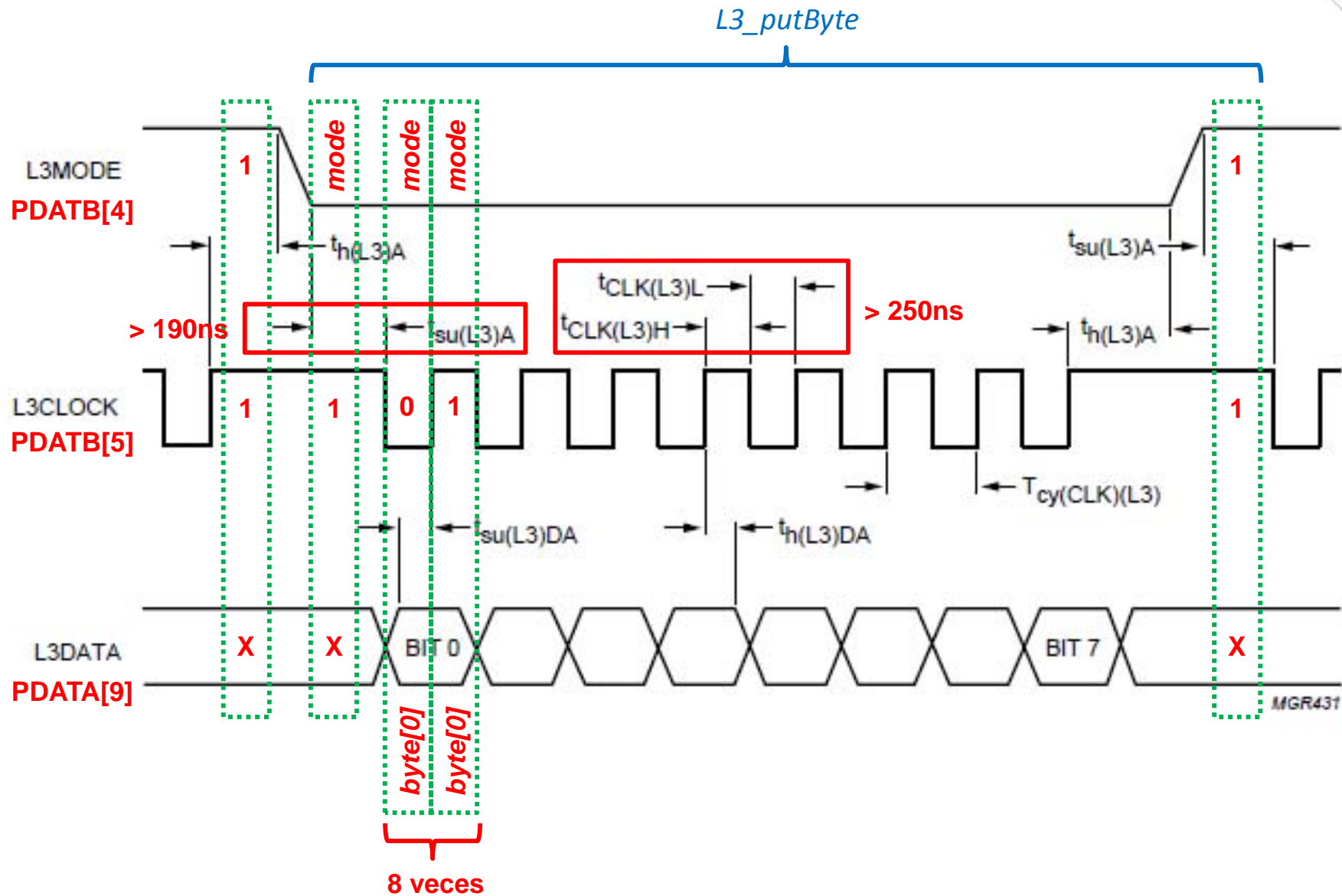
# Bus L3



- El interfaz L3 es un bus serie síncrono
  - 1 línea bidireccional de datos: L3DATA
    - Todos los datos transmitidos son de 8 bits (LSB first)
  - 2 líneas de entrada: L3MODE (modo) y L3CLOCK (reloj).
    - Si L3MODE = 0 (address mode), por L3DATA se envían las direcciones del dispositivo (6b) y del registro destino (2b).
    - Si L3MODE = 1 (data mode), por L3DATA se envían/reciben datos.
  
- El SoC no incluye un controlador de bus L3
  - Las líneas están conectadas directamente a puertos de E/S configurados como salidas:
    - L3DATA = PDATA[9]
    - L3MODE = PDATB[4]
    - L3CLOCK = PDATB[5]
  - Las formas de onda deben generarse por SW respetando ligaduras de tiempo establecidas en los cronogramas
  - Dado que el puerto B está compartido con los leds:
    - Cada vez que escribamos en PDATB para generar las formas de onda de L3MODE y L3DATA hay que sobrescribir el estado de los leds para que no cambien.

# Bus L3

## cronograma (i)



# Driver de bus L3

## 13.h



```

#ifndef __L3_H__
#define __L3_H__ ..... Indica el modo a transmitir: L3CLOCK=1 y L3MODE=mode

#include <common_types.h>

#define L3_ADDR_MODE (0)
#define L3_DATA_MODE (1) } Nemotécnicos para identificar los modos de transferencia del bus L3

void L3_init( void ); ..... Deja en reposo el bus L3: L3CLOCK=1 y L3MODE=1
void L3_putByte( uint8 byte, uint8 mode ); ..... Envía un byte en el modo indicado

#endif

```

# Driver de bus L3

## 13.c



```
#define SHORT_DELAY { int8 j; for( j=0; j<4; j++ ); }

void L3_putByte( uint8 byte, uint8 mode )
{
    uint8 rled, lled;

    rled = !led_status( RIGHT_LED );
    lled = !led_status( LEFT_LED ); } Almacena en variables el estado de los leds

    PDATB = ...; ..... Indica el modo a transmitir: L3CLOCK=1 y L3MODE=mode
    SHORT_DELAY; ..... espera tsu(L3)A > 190 ns
    for( i=0; i<8; i++ ) ..... Serializa el byte comenzando por el LSB
    {
        PDATB = ...; ..... Baja la señal de reloj: L3CLOCK=0 y L3MODE=mode
        PDATA = ...; ..... Pone el bit a transmitir: L3DATA = byte[i]
        SHORT_DELAY; ..... espera tCLK(L3)L > 250ns
        PDATB = ...; ..... Sube la señal de reloj: L3CLOCK=1 y L3MODE=mode
        SHORT_DELAY; ..... espera tCLK(L3)H > 250ns
    }
    PDATB = (rled << 10) | (lled << 9) | (1 << 5) | (1 << 4);
}
```

Deja los leds inalterados

Deja en reposo el bus L3  
L3CLOCK=1 y L3MODE=1

# Audio Codec UDA1341TS

## control de operación (i)



- Dispone de 3 registros primarios para controlar su operación
  - Accesibles directamente a través del bus L3
  - STATUS (10): permite programar la frecuencia del sistema y el formato de datos IIC
  - DATA0 (00): permiten realizar ajustes de sonido (volumen, bass, treble, mute...)
  - DATA1 (10): permiten conocer peak level del sonido reproducido
- Dispone de 5 registros secundarios (o extendidos)
  - Accesibles indirectamente a través del registro primario DATA0
  - MA (000): permite ajustar la atenuación del canal de entrada 1
  - MB (001): permite ajustar la atenuación del canal de entrada 2
  - MS/MM (010): permite mezclar o seleccionar uno de los canales de entrada
  - AG/IG (100): permite activar/desactivar el control automático de ganancia (AGC)
  - IG (101): permite ajustar la ganancia del amplificador del canal de entrada 2
  - AT/AL (110): permite ajustar parámetros de AGC
- La dirección del Audio Codec es: 000101
- El micrófono está conectado al canal de entrada 1

# Audio Codec UDA1341TS

## control de operación (ii)



- Para **activar/desactivar el reset** el audio codec
  - STATUS[7]=0 y STATUS[6]=1/0
- Para **indicar la frecuencia del sistema y el formato de datos**
  - STATUS[7]=0 y STATUS[5:4]=2 **256fs**
  - STATUS[7]=0 y STATUS[2:0]=0 **IIS compatible**
- Para **indicar el canal** al que está conectado el micrófono:
  - MS/MM[1:0] = 1 **canal 1**
- Para **encender/apagar el ADC/DAC** con 6dB de ganancia de entrada
  - STATUS[7]=1 y STATUS[1:0]=X **valor según corresponda**
  - STATUS[7]=1 y STATUS[5]=1 **6dB de ganancia**
- Para **activar/desactivar mute**
  - DATA0[7:6] = 2 y DATA0[2]=1/0
- Para **fijar el volumen** de reproducción
  - DATA0[7:6] = 0 y DATA0[5:0]=atenuación (a mayor atenuación, menor volumen)

# Audio Codec UDA1341TS

## control de operación (iii)



- Para escribir en un **registro primario del codec** se envían 2 bytes
  - Se envía el 1er. byte en address mode que incluye:
    - la dirección del dispositivo (000101)
    - la identificación del registro primario a escribir (2 bits)
  - Se envía el 2do. byte en data mode que incluye el dato (8 bits)
- Para escribir en un **registro secundario del codec** se envían 3 bytes
  - Se envía el 1er. byte en address mode que incluye:
    - la dirección del dispositivo (000101)
    - la identificación del registro DATA0 (00)
  - Se envía el 2do. byte en data mode que incluye:
    - un prefijo constante (11000)
    - la identificación del registro secundario (3 bits)
  - Se envía el 3er. byte en data mode que incluye:
    - un prefijo constante (111)
    - el dato (5 bits)
- **No es posible leer registros del CODEC**
  - PDATA[9] (L3DATA) solo puede configurarse como salida y por eso no hemos implementado la función L3\_getbyte que sería necesaria.



# Driver de UDA1341TS

## uda1341ts.h



```
#ifndef __UDA1341TS_H__
#define __UDA1341TS_H__

#include <common_types.h>

#define UDA_DAC (1)
#define UDA_ADC (2) } Nemotécnicos para identificar los conversores

#define VOL_MAX (0x3F)
#define VOL_MED (0x20)
#define VOL_MIN (0x0) } Declara macros para identificar algunos volúmenes

#define MUTE_ON (1)
#define MUTE_OFF (0) } Nemotécnicos para identificar el estado del mute

void uda1341ts_init( void );
void uda1341ts_mute( uint8 on );
void uda1341ts_on( uint8 converter );
void uda1341ts_off( uint8 converter );
uint8 uda1341ts_status( uint8 converter );
void uda1341ts_setvol( uint8 vol );
uint8 uda1341ts_getvol( void );

#endif
```



# Driver de UDA1341TS

## uda1341ts.c

```
#include <l3.h>
#include <uda1341ts.h>

#define ADDRESS (0x05)
#define DATA0 (0x0)
#define DATA1 (0x1)
#define STATUS (0x2)
#define EA (0x18 << 3)
#define ED (0x7 << 5)

static uint8 volume;
static uint8 state;

void uda1341ts_init( void )
{
    L3_init();
    L3_putByte( (ADDRESS << 2) | STATUS, L3_ADDR_MODE );
    L3_putByte( (1 << 6), L3_DATA_MODE );
    L3_putByte( (2 << 4), L3_DATA_MODE ); ..... Quita el reset y fija frecuencia y formato de IIS
    L3_putByte( (ADDRESS << 2) | DATA0, L3_ADDR_MODE );
    L3_putByte( EA | (2), L3_DATA_MODE );
    L3_putByte( ED | 1, L3_DATA_MODE );
    uda1341ts_setvol( VOL_MAX );
    uda1341ts_on( DAC );
    uda1341ts_on( ADC );
}
```

*Nemotécnicos para identificar los registros primarios*

*Nemotécnicos para los prefijos de acceso a los registros secundarios*

*Almacena localmente el volumen y el estado de los ADC/DAC ya que no es posible leer los registros de Audio Codec*

*Resetea el Audio Codec*

*Selecciona el canal 1*

# Programa principal

## reproducción de sonido tabulado



```
#define LOOKUPTABLE_SIZE    (36)

const int16 sine[LOOKUPTABLE_SIZE] = ..... Do central (440Hz) tabulado en forma sinusoidal
{
    0,  157,  314,  472,  579,  689,  779,  846,  887,  900,  887, ...
    0, -157, -314, -472, -579, -689, -779, -846, -887, -900, -887, ...
};

const int16 square[LOOKUPTABLE_SIZE] = ..... Do central (440Hz) tabulado en forma cuadrada
{
    900,  900,  900,  900,  900,  900,  900,  900,  900,  900,  900, ...
   -900, -900, -900, -900, -900, -900, -900, -900, -900, -900, -900, ...
};

const int16 triangle[LOOKUPTABLE_SIZE] = ..... Do central (440Hz) tabulado en forma triangular
{
    0,  100,  200,  300,  400,  500,  600,  700,  800,  900,  800, ...
    0, -100, -200, -300, -400, -500, -600, -700, -800, -900, -800, ...
};

while( ... )
{
    iis_putSample( square[i], square[i] ); ..... Reproduce de muestras tabuladas
    if( ++i > LOOKUPTABLE_SIZE )
        i = 0;
}
```

} incrementa modularmente el índice de muestras



# Programa principal

## reproducción a distintas velocidades

```
#define AUDIOBUFFER_SIZE (16000*10) ..... 10s de audio a 16000sps

for( i=0; i<AUDIOBUFFER_SIZE; i++ )
    iis_getSample( &(buffer.ch0[i]), &(buffer.ch1[i]) ); } Grabación de muestras

for( i=0; i<AUDIOBUFFER_SIZE; i++ )
    iis_putSample( buffer.ch0[i], buffer.ch1[i] ); } Reproduce de muestras
a velocidad normal

for( i=0; i<AUDIOBUFFER_SIZE; i++ )
{
    iis_putSample( buffer.ch0[i], buffer.ch1[i] );
    iis_putSample( buffer.ch0[i], buffer.ch1[i] ); } Reproducción de muestras
a media velocidad

for( i=0; i<AUDIOBUFFER_SIZE; i+=2 )
    iis_putSample( buffer.ch0[i], buffer.ch1[i] ); } Reproducción de muestras
a doble velocidad

for( i=0; i<AUDIOBUFFER_SIZE; i++ )
    iis_putSample(
        buffer.ch0[AUDIOBUFFER_SIZE-i],
        buffer.ch1[AUDIOBUFFER_SIZE-i]
    ); } Reproducción de muestras al revés
```

Cada muestra se reproduce 2 veces

Se reproduce una muestra sí y otra no



# Programa principal

## efectos en dominio temporal: echo & fade

```
audiodelay_init( &delay_buffer, 16000 ); ..... Buffer para retardo de 1s de audio a 16000sps
while( ... )
{
    iis_getSample( &ch0, &ch1 ); ..... Lee la muestra actual
    audiodelay_get( &delay_buffer, &ch0_aux, &ch1_aux ); ..... Saca la muestra
                                                                almacenada hace 1 s
    ch0_aux = ch0 + (ch0_aux >> 1);
    ch1_aux = ch1 + (ch1_aux >> 1); } Reproduce la suma de la muestra actual y la muestra
                                                                almacenada hace 1 s atenuada (echo)
    iis_putSample( ch0_aux, ch1_aux );
    audiodelay_put( &delay_buffer, ch0, ch1 ); ..... Almacena la muestra actual
}

audiodelay_init( &delay_buffer, 16000 );
while( ... )
{
    iis_getSample( &ch0, &ch1 );
    audiodelay_get( &delay_buffer, &ch0_aux, &ch1_aux );
    ch0 = ch0 + (ch0_aux >> 1);
    ch1 = ch1 + (ch1_aux >> 1);
    iis_putSample( ch0, ch1 );
    audiodelay_put( &delay_buffer, ch0, ch1 ); ..... Almacena la muestra reproducida
                                                                (echo & fade)
}
```

# Tareas

## lab9-wav



1. Crear el proyecto **lab9-wav** a partir de una copia de uno anterior.
2. Descargar en el directorio **lab9-wav** el fichero **lab9-wav.c**
3. Descargar en el directorio **BSP/include** los ficheros:
  - **dma.h**, **l3.h**, **iis.h** y **uda1341ts.h**
4. Codificar en **BSP/source** los ficheros:
  - **dma.c**, **l3.c**, **iis.c** y **uda1341ts.c**
5. Refrescar los proyectos **BSP** y **lab9-wav**
6. Compilar primero el proyecto **BSP** y después el proyecto **lab9-wav**
7. Crear la configuración de depuración **lab9-wav** a partir de una anterior.
8. Crear un directorio **wav16b16K** y descargar en él los ficheros:
  - **\*.wav** y **load\_wav.txt**
9. Conectar la placa y encenderla.
10. Arrancar OpenOCD y Termite.
11. Cargar en memoria los BMP ejecutando el script **load\_wav.txt**
12. Arrancar la configuración de depuración **lab9-wav**

# Tareas

## lab9-efectos



1. Crear el proyecto **lab9-efectos** a partir de una copia de uno anterior.
2. Descargar en el directorio **lab9-efectos** el fichero **lab9-efectos.c**
3. Refrescar el proyecto **lab9-efectos**
4. Compilar el proyecto **lab9-efectos**
5. Crear la configuración de depuración **lab9-efectos** a partir de una anterior.
6. Conectar la placa y encenderla.
7. Arrancar OpenOCD y Termite.
8. Arrancar la configuración de depuración **lab9-wav**