Titulo:	Reproductor WAV

Ciclo Lectivo **2017**	Curso N°	R2001	Grupo N°	2

Integrantes	Apellido Nombres	Legajo	Calificación individual	Fecha
Serra, Ramiro		159.224-5		
Ozán, Santiago Gabriel		159-353-5		
Domínguez, Néstor Matías		159.021-2		

Calificación grupal:		Fecha:

Profesor:	Giura, Marcelo
Auxiliar/es Docente:	Soccodato, Gabriel

Observaciones primera entrega	
Observaciones segunda entrega	

CONTENIDOS MÍNIMOS DEL TPO

- Índice
- Desarrollo de la idea fuerza
- Introducción
  - Objetivos
  - Diagramas en bloques

- Descripción Detallada
  - Bloques.
  - Especificaciones.

- Descripción del Hardware utilizado.
  - La descripción de los elementos de HW utilizados debe hacerse en conjunto con el SW desarrollado para su funcionamiento. Es decir el SW debe ayudar a la comprensión del dispositivo presentado.
- Maquina de estados de la aplicación.

- Problemas encontrados a lo largo del desarrollo del TPO
  - Puesta en marcha.
  - Algoritmos
  - Falta de información
  - Etc.

- Beneficios encontrados a lo largo del desarrollo del TPO
  - Valoración del TPO expresado por los integrantes del grupo
  - Desde su lugar de alumno que elementos agregaría o quitaría para el desarrollo del TPO

- Conclusiones.
- Bibliografía, links, etc.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

• Desarrollo de la Idea Fuerza

Nuestro proyecto consiste en un Reproductor de archivos .WAV, cuyo medio de almacenamiento es mediante una tarjeta de memoria SD en formato FAT. Desde el micro, la lectura de la misma se logra mediante un sistema de archivos, el cual posee funciones que permiten el manejo de estos.

La música se reproduce por el DAC, conectando unos parlantes a la salida. Los botones que posee el KIT, son los que nos permiten movernos a través de los archivos, pausar y reproducir.

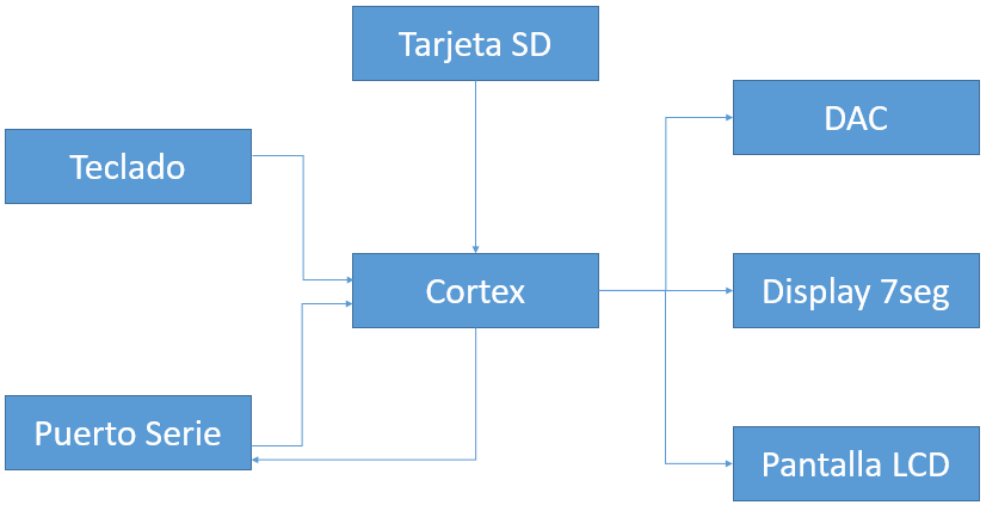
• Introducción

Objetivos

Desarrollar una aplicación capaz de reproducir desde una tarjeta SD archivos de audio en formato WAV y controlar su reproducción tanto, por el teclado (disponible en el Infotrónic), como por una aplicación interfaz gráfica desde una PC, comunicada por puerto serie.

La aplicación deberá funcionar tanto por sí sola, como conectada a la pc.

Diagrama de Bloques



• Descripción detallada

Bloques

Teclado: de este, salen los comandos que recibe el Cortex, para tomar las decisiones correspondientes, como “Siguiente tema”, “Anterior Tema”, “Pausa”, “Play”.
Puerto serie: este permite la comunicación entre el Kit, y la interfaz gráfica, pudiendo así lograr que la interfaz muestre lo que sucede en el kit en tiempo real y, a su vez, poder controlarlo.
Tarjeta SD: es el almacenamiento del reproductor. En él se encuentras los archivos a reproducir.
Cortex: es el cerebro de la aplicación. Es el encargado de hacer el manejo de los buffers circulares, que permiten la reproducción, maneja múltiples temporizaciones simultáneamente, el intérprete de las tramas de las series, y de las interrupciones.
DAC: Se encarga de transformar las muestras, que el Cortex le provee, en una señal analógica, para reproducirla por un parlante.
Display 7 segmento: mediante el RTC que posee el Cortex, este indica el tiempo de reproducción del archivo. Cambiando el archivo, se resetea el RTC.
LCD: Recibe los nombres de las canciones y los imprime en pantalla.

Especificaciones

Soporte para tarjetas SD, en formato FAT.
Soporte para archivos WAV, de 8 bits, muestreo de 16KHz, mono.
Teclado de 3x1, para manejo de reproducción de los archivos.
Led RGB para indicar conexión con la PC.
Displays de 7 segmentos, para indicar la temporización de la reproducción.
Pantalla LCD para mostrar el archivo actual.
Interfaz gráfica diseñada en QT, comunicándose por puerto serie.

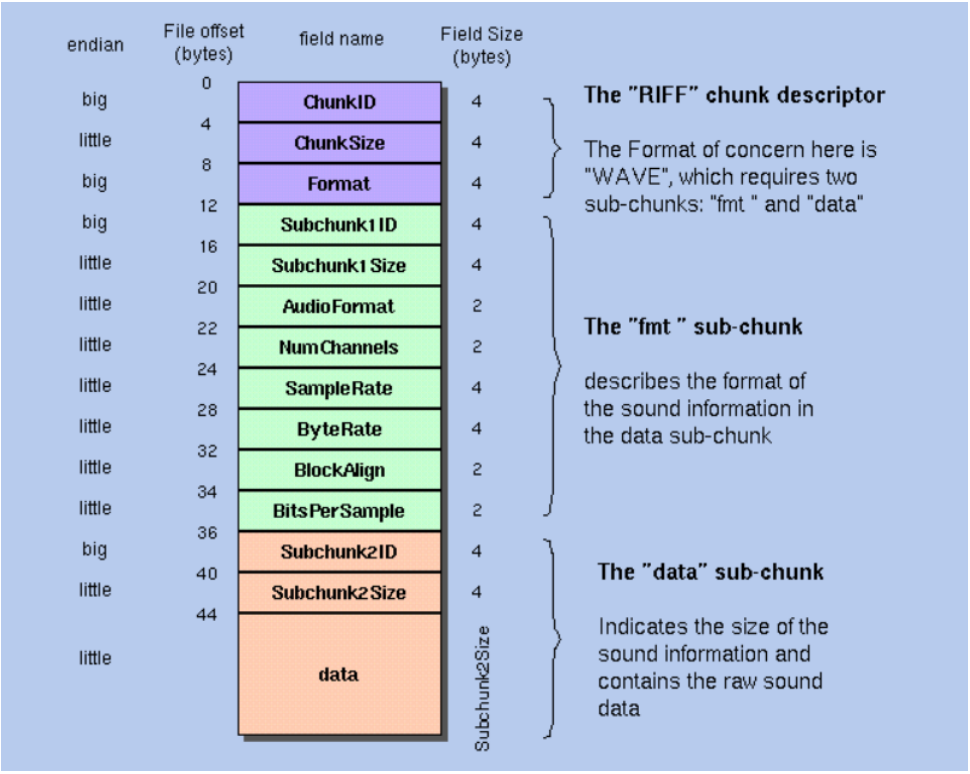
• Hardware utilizado

DAC: Para la reproducción del audio. Descripción del funcionamiento:
Timers: Temporización de envío de muestras al DAC.
Systick: Marca pasos del sistema, barrido de Displays, teclado, y temporizaciones varias.
RTC: Contador de tiempo de reproducción.
Expansión 2: Mostrado de tiempo de reproducción en los Displays de 7 segmentos.
Kit Infotrónic: Teclado 4x1, LED RGB.
UART0: Envío y recepción de datos.
SPI: Realizado de la interface SD

Descripción del funcionamiento:

Todo el sistema se basa en el filesystem FAT, para el cual utilizamos un módulo desarrollado llamado FATFs by chaN, éste ultimo nos provee una capa de abstracción, ya que con algunas modificaciones mínimas, e incluyendo el código en nuestro trabajo, podremos usar unas funciones utilitarias muy parecidas al de manejo de archivos, para manejar la tarjeta SD.

También es necesario aclarar, el formato WAV en el que está almacenado el audio tiene la siguiente forma:



(44 bits de encabezado con información del archivo, y luego los datos).

Lo que nosotros hicimos, por simplicidad, es convertir todos los archivos de audio a WAV de 8 bits, 16kHz,MONO, y no preocuparnos de ahora en más de la cabecera.

El primer paso es inicializar la tarjeta SD, mediante la siguiente linea (el sistema se cuelga hasta que la tarjeta sea ingresada).

```
while ((res = f_mount(&FatFs, "", 1))!=FR_OK);
```

Luego, podremos utilizar varias funciones muy parecidas a las de manejo de archivos, por ejemplo

- File Access
  - f_open - Open/Create a file
  - f_close - Close an open file
  - f_read - Read data from the file
  - f_write - Write data to the file
  - f_lseek - Move read/write pointer, Expand size
  - f_truncate - Truncate file size
  - f_sync - Flush cached data
  - f_forward - Forward data to the stream
  - f_expand - Allocate a contiguous block to the file
  - f_gets - Read a string
  - f_putc - Write a character
  - f_puts - Write a string
  - f_printf - Write a formatted string
  - f_tell - Get current read/write pointer
  - f_eof - Test for end-of-file
  - f_size - Get size
  - f_error - Test for an error

Haciendo uso de éstas, el próximo paso es llenar un vector de strings con los nombres de la canciones, ya que la funcion f_open se utiliza con el nombre del archivo.

```
FileList_Fill();
```

Ahora es el momento de empezar con la reproducción, es importante destacar, que por la falta de memoria en el microcontrolador, no nos es posible hacer un buffer lo suficientemente grande para guardar el archivo completo, y reproducirlo desde alli.

Ante esta problemática utilizamos un buffer circular de 2 posiciones, con 1024 bytes en cada una, y el reproductor va a ir llenando y cambiando de buffer constantemente. De esto se ocupa el match del Timer0, que está seteado para interrumpir a 16kHz, como la frecuencia de muestreo que tienen los archivos de audio.

```
void TIMER0_IRQHandler (void)
{

    static uint32_t i=0;
    static uint8_t actual=0;
    uint8_t aux=0;
    static uint32_t cont=0;

    if(T0IR & MR0){

        T0IR |= MR0;
        if (cant<1024){
            ApagarTimer0();
            flagtermino=1;
            return;
        }else{
            Write_Dac(bufferswav[actual][i]);
            i++;
            i%=cant;
            if (!i){
                cont++;
                aux=actual;
                actual++;
                actual%=2;
                ApagarTimer0();
                f_read (&File,bufferswav[aux],1024,(UINT*)&cant);
                PrenderTimer0();

            }
        }
    }
}
```

Esta rutina de interrupción es de lo más importante del programa, para empezar, cant almacena la cantidad de bytes leídos del archivo, si ésta ultima es menor que 1024, significa que el tema terminó y debe pasar al siguiente.
Luego, la variable i es el indice de muestras que se mandan al dac, si esta ultima llega a 1024, significa que leyó el buffer entero y debe pasar al siguiente, sobrescribiendo el anterior con información nueva.

Ahora pasamos a la parte de control de la reproducción, se implemento mediante una maquina de estados.

```
void maqreproduccion(void){
    switch (estado2){

        case (REPOSO):

            if (tecla==1){
                ArrancarReproduccion();
            }
            break;

        case (REPRODUCIENDO):

            if (tecla==1){
            PausarReproduccion();
            break;
            }
            if (tecla==2){
                AnteriorTema();
                break;
            }
            if (tecla==3){
                SiguienteTema();
                break;
            }
            break;
    }
}
```

Es bastante simple y auto explicativa.

Explicaremos una sola función de esas, las demás serán muy similares.

```
void SiguienteTema(void){
    tecla=NO_KEY;
    ApagarTimer0();
    ArchivoActual--;

    if (ArchivoActual>MAX_FILE-1){
            ArchivoActual=(MAX_FILE-2);
    }
    f_open(&File, FileList[ArchivoActual] , FA_READ);
    cargarBuffer();
    PrenderTimer0();

    reiniciarRTC();
    prenderRTC();

    WComando8(LCD_CLEAR);
    WComando8(LCD_HOME1);
    WString(FileList[ArchivoActual]);
    EnviarString0("SIGUIENTE$");
    PrenderTimer0();
}
```

Primero, limpiamos la tecla, y apagamos el timer para que no interrumpa mientras que manejamos los buffers. Luego, cambiamos el indice del archivo actual y nos aseguramos que no hayamos sobrepasado los limites. Con ayuda de la lista de nombres que habiamos llenado al principio, abrimos el archivo desde la sd y cargamos los buffers. Prendemos el timer para que empiece a interrumpir, reiniciamos el RTC para que cuente desde 0, enviamos el nombre del archivo a la pantalla LCD y enviamos por puerto serie una trama que indique la operación.

Ahora, la comunicación serie:

Para ésta ultima utilizaremos otra maquina de estados.

```
switch (estado){

    case REPOSO:
        if (recibido==INICIO_TRAMA) estado=VALIDACION;
        break;

    case VALIDACION:
        if (recibido!='S' && recibido!='A' && recibido!='P' && recibido!='p' &&
        estado=REPOSO;
        break;
        }else{
            contenidoTrama=recibido; estado=FINDETRAMA;break;}
        break;

    case FINDETRAMA:
        if (recibido!=FINAL_TRAMA){
            estado=REPOSO;break;
        }
        if(contenidoTrama=='S'){
            tecla=3;estado=REPOSO;break;
        }

        if(contenidoTrama=='A'){
            tecla=2;estado=REPOSO;break;
        }
        if(contenidoTrama=='p'){
            tecla=1;estado=REPOSO;break;
        }
        if(contenidoTrama=='P'){
            tecla=1;estado=REPOSO;break;
        }
        if(contenidoTrama=='N'){
                mandarlista();timeoutSerie=TIMEOUT_SERIE;estado=REPOSO;break;
        }
        if(contenidoTrama=='Y'){
                timeoutSerie=TIMEOUT_SERIE;estado=REPOSO;break;
        }

        reproducirActual(contenidoTrama);
        estado2=REPRODUCIENDO;
        break;
```

La maquina descarta errores y llega a fin de trama donde según el valor de ContenidoTrama, que es lo que se envió, cambia tecla por la correspondiente al comando, lo hicimos de esta forma para manejar los estados de la otra maquina desde el puerto serie.

Cuando le llega una N, esto quiere decir que la interfaz gráfica no tiene la lista de canciones, y hay que enviársela.

La Y es la respuesta al ACK que envía la interfaz.

Si el contenido de la trama no corresponde a ningún comando, quiere decir que lo que llegó corresponde al número de canción que se quiere reproducir. Por tanto se invoca a la función reproducirActual();

Ultima parte a destacar del lpc, el systick:

Además de todos los barridos, temporizaciones varias, demoras para el LCD, etcétera; El systick se encarga de mandar un ACK cada 400ms. Lo especial de este ACK es que contiene además el número de la cancion actual (indice), los segundos y minutos de la reproducción, para poder realizar la sincronización en tiempo real con la interfaz gráfica.

```
if(!ack){
    ack=TIMEOUT_ACK;
    armarTramaACK(aux);
    EnviarString0(aux);
```

```
void armarTramaACK(uint8_t* aux){

    uint8_t aux2[4];
    aux2[0]=ArchivoActual+1;
    aux2[1]=(uint8_t)RTC_MIN+1;
    aux2[2]=(uint8_t)RTC_SEC;
    aux2[3]='\0';

    strcat(aux,"ack");
    strcat(aux,aux2);
    strcat(aux,"$");
}
```

Llegamos a la interfaz gráfica, lo destacable acá es el procesamiento de las tramas:

La señal de recepción de datos del puerto está conectada con el SLOT onDatosRecibidos();

```
void MainWindow::onDatosRecibidos(){
    QByteArray bytes;
    QByteArray static word;
    int cant =puerto->bytesAvailable();
    bytes.resize(cant);
    puerto->read(bytes.data(),bytes.size());
    word.append(bytes);
    if(word.startsWith("%")&&word.endsWith("$")){
        maqestado(word);
        word.clear();
        word.resize(0);
    }
}
```

Como la trama no siempre llega entera, no se procesa hasta que arranque con % y termine con $.

El proceso de la trama lo realiza maqestado() :

El proceso de sincronización se produce cuando recibe un ack:

```
if ((data.contains("ack"))){
    if (flagcambiar){
        if(!this->tengoCanciones){
            puerto->write("aNj");
        }else{puerto->write("aYj");}
        auxiliar=data;
        auxiliar.chop(1);
        auxiliar.remove(0,4);
        char *p = auxiliar.data();
        aux2=p[0]-1;
        ui->lcdNumberM->display(p[1]-1);
        ui->lcdNumberS->display(p[2]);
        //ui->listacanciones->setDisabled(true);
        int j = ui->listacanciones->currentIndex();
        if((ui->listacanciones->currentIndex()!= aux2) && (flagcambiar)){
            flagcambiar=false;
            QTimer::singleShot(200, this, SLOT(bajarflag()));
            ui->listacanciones->setCurrentIndex(aux2);

         }
       // ui->listacanciones->setDisabled(false);
        return;
       }
 }else{
   llenarComboboxTemas(data);
   tengoCanciones=true;
   return;
}
```

Lo que hace primero es cortar la parte inútil con chop y remove, para luego, con un puntero a char, obtener los datos de la trama.

La primera posicion tiene el numero de cancion. Por tanto, si el comboBox no está actualizado lo actualiza.
La segunda posicion tiene los minutos y la tercera los segundos.

De esta manera sincronizamos la interfaz con el kit cada 400 ms.

Problemas encontrados:

- El prmero fue adaptar el código de FatFs al infotronic, ya que si bien hay una versión para LPC1769, ésta esta diseñada para una placa distinta a la nuestra, esto es un trabajo muy tedioso ya que el código es inmenso y casi ilegible.

- Otro desafio fue que el LCD utiliza demoras de systick, y eso empezó a molestar cuando lo usamos dentro de otras interrupciones, ya que entraban en juego las priorizaciones de estas ultimas.

- Tuvimos inconvenientes tambien en el armado de la trama del ACK, ya que al usar la funcion strcat, cuando el archivo actual, los minutos o los segundos valian 0, el strcat interpretaba el '\0' y nos cortaba la trama, un error dificil de detectar.

- El RTC nos trajo problemas a la hora de mostrar los min. y seg. en el display 7seg, porque aparecian intercambiados, o los segundos duplicados en el lugar de los minutos.

- La sincronizacion con la interfaz no fue nada facil, ya que cuando cambiabamos desde la aplicación el tema, no llegaba a actualizar la cancion actual en el kit antes de que este envie un ack para sincronizar la cancion de la interfaz y tuvimos que implementar timers.

- Teniamos problemas con el slot de on_listacanciones_currentIndexChanged del combobox de los nombres de las canciones, ya que a veces llegaba con index = -1, y no podiamos detectar la razon.

- El llenado del comboBox de la interfaz con los temas tampoco fue nada fácil, implementamos una trama con los nombres de las canciones separados por comas.

- Cuando usabamos el debugger del LPC, para probar la funcion que armaba el ACK, tenia problemas mientras estabamos con el programa en "pausa", ocacionando errores raros, que no ocurrian si dejabamos que corra el programa normalmente.

- No pudimos implementar en la interfaz, el doble 0 en el lcdNumber cuando los segundos o los minutos valian de 0 a 9 (00, 01, 02, en vez de 0,1,2 etc)

- El uso de Git como herramienta, aunque muy conveniente, resulto dificil de utilizar al tener problemas cuando el workspace del LPC era distinto de la carpeta del repositorio, muchos tipos de archivo debieron ser incluidos al .gitignore para facilitar y acelerar los commits, etc.

- Al ir agregando mas codigo, mas funciones y mas archivos fuente, el proyecto se empieza a desordenar rapidamente y genera confusion

Valoracion del TPO:

Nos sirvio para afianzar ideas, ya que la teoria a veces no es suficiente, y surgen muchos problemas de temporizacion, sincronizacion e interaccion entre diferentes modulos, y se termina con un entendimiento mayor de el funcionamiento de las distintas partes, que viene con la experiencia. Tambien sirvio para mejorar aun mas el trabajo en equipo, el relevamiento de tareas que afianza la metodologia de programacion la discusion del codigo (por ejemplo cuando se presenta un error o un problema dificil de solucionar) para un aprendizaje mas profundo.

Elementos a quitar/agregar al desarrollo del tpo:

Desde nuestro punto de vista, el desarrollo del tpo esta bien, no necesita ni le falta nada, en nuestro caso.

Conclusiones:

Aunque funciona relativamente bien para estar usando archivos MONO de 16 bits, la calidad de audio no es tan buena, ademas, sin usar un amplificador por hardware la calidad es significativamente peor, hay mas ruido e interferencia.

El uso de la tarjeta sd nos llevo a usar un filesystem que no desarrollamos nosotros, pero al tener muchas similitudes con el manejo de archivos en C visto en informatica I, pudimos adaptar y utilizar el codigo nuevo.

El QT fue rapido para programar (por la comodidad de el uso de multiples funciones de libreria) pero lento de hacer funcionar. Estuvimos mucho mas tiempo arreglando errores de sincronizacion y manejo de la interfaz que implementando las funciones para hacer uso del puerto serie y armado de tramas, por ejemplo.

Bibliografia:

http://elm-chan.org/fsw/ff/00index_e.html Codigo filesystem
http://doc.qt.io/ Documentacion C++
https://stackoverflow.com/ Dudas y dificultades