Práctica 2 : Digitalizador de audio usando ESP32

Marlon Sneider Mora Cortes - 20152005034, Sara Valentina Barrero Medina -20191005170
Miguel Ángel Fuentes Ramírez-20182005007
Ingeniería Electrónica, *Universidad Distrital "Francisco José de Caldas"*
Bogotá, Colombia

***Resumen*—El diseño de un digitalizador de audio empleando ESP32 requiere manejar conceptos como ADC,DAC, Muestreo , así como requiere manejo de las interrupciones en un microcontrolador, en este caso el ESP32, el concepto de rebote en los pulsadores es algo que puede presentar serios problemas así como el tiempo que tarda la visualización en una pantalla TFT.**

***Palabras Clave*–**Esp32, ADC, DAC, Digitalización, Interrupción, Muestreo

I. Introducción

En este informe se presenta la propuesta de solución al proyecto 2 de la asignatura Sistemas Embebidos 1, el cual consiste en la elaboración de un digitalizador de audio empleando la tarjeta de desarrollo ESP32, por medio de una salida de audio generada por un computador o celular, para emular una señal analógica se requiere de un Convertidor Analogico-Digital, el ESP32 incorpora un DAC con dos canales de 8 bits, conectados a los pines GPIO25 (Canal 1) y GPIO26 (Canal 2), por otro lado para la lectura y digitalización de una señal análoga el ESP32 incorpora un ADC con una resolución de 9 a 12 bits y soporta 18 canales de medida, distribuidos de la siguiente manera :

- ADC1 (8 canales), conectados a los GPIOs (32 - 39)
- ADC2 (10 canales), conectados a los GPIOs (0, 2, 4, 12 - 15 y 25 - 27).

En este proyecto se emplea una pantalla TFT que permite la viauslizacion de la forma de onda de la señal generada por el DAC, para el control de dicha pantalla se emplea la comunicación SPI, la cual consiste en una arquitectura de tipo maestro-esclavo. El dispositivo maestro (master) puede iniciar la comunicación con uno o varios dispositivos esclavos (slave), y enviar o recibir datos de ellos. Los dispositivos esclavos no pueden iniciar la comunicación, ni intercambiar datos entre ellos directamente.

En el bus SPI la comunicación de datos entre maestros y esclavo se realiza en dos líneas independientes, una del maestro a los esclavos, y otra de los esclavos al maestro, por tanto la comunicación es Full Duplex.El bus SPI requiere un mínimo de 3 líneas, además de una línea SS (Slave Select) para cada dispositivo esclavo conectado, para seleccionar el dispositivo con el que se va a realizar la comunicación.

II. Formulación del Problema

El problema consiste en diseñar e implementar un digitalizador de audio utilizando el ADC interno del ESP32, se debe ingresar una señal de audio, específicamente una canción que contenga voz y sonidos generados por instrumentos musicales, digitalizarla en tiempo real y luego reproducirla con el DAC interno del ESP32.

Se tendrán dos pulsadores N.A. para controlar digitalmente el volumen de la señal, que tendrá diez niveles de volumen fácilmente diferenciables entre sí, estos valores se mostrarán en una LCD TFT ILI9163 (o similar), de manera numérica y gráficamente con una barra gráfica de tipo horizontal dividida en 10 secciones, cada una de ellas cambiará de acuerdo a nivel de volumen seleccionado, además, se visualizará la forma de onda que se está generando en el DAC (por lo menos los últimos 5 segundos).

III. Diseño y Modelo de solución

Lo primero que se contemplo para la elaboración del proyecto son los elementos requeridos, en primer lugar se empleo 1 pantalla TFT de 128x128, en dicha pantalla se mostrara la forma de onda de salida de la señal reconstruida, así como también se mostrara un indicador de nivel de volumen en forma de barra horizontal, se requieren de 2 pulsadores N.A para el control de volumen, finalmente también se requiere de un circuito externo tanto a la entrada de audio al ADC, como a la salida del DAC para la amplificación de la señal y reproducción empleando un parlante.

Se debe tener el consideración que se empleara la resolución mas alta disponible para el muestro de la señal empleando el canal 6 del ADC, se configuro una resolución de 12 Bits para esta tarea, para el DAC se tendrá una resolución de 16 Bits para la reconstrucción de la señal, sse obtuvo una frecuencia de muestro de aproximadamente 3.3 kHz, para el control de volumen se propone una ecualización de 0 a 1 con pasos de 0.1 sobre la variable de salida proveniente de la lectura realizada en el ADC.

El circuito propuesto a la entrada para el acondicionamiento de la señal que entra al ADC del ESP32 es el que se aprecia en la figura 1, para la etapa de salida se emplea el amplificador de 3W PAM8409.

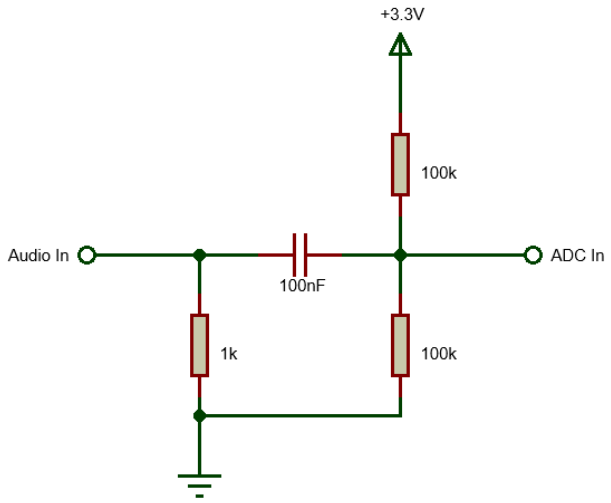

Figure 1: Circuito de acondicionamiento de entrada

A. *Pines Empleados*

Para el control de los botones encargados del de volumen, se emplean pines con PULL-UP interno, se requieren un total de 2 pines para esta tarea, para el ADC y DAC son requeridos 2 pines mas, finalmente para la pantalla TFT empleando comunicación SPI se requieren 4 pines, por lo que en total se requerirán de 8 pines para el desarrollo de la practica, estos pines se encuentran resumidos en la tabla I.

Tarea	#Pin	Funcionalidad
ADC	GPIO34	Entrada ADC Canal 6
DAC	GPIO 25	Salida DAC
Control de Pantalla TFT	GPIO5 GPIO3 GPIO23 GPIO18	CS DC MOSI SCK
Control de Botones	GPIO21 GPIO19	Controla Pulsador de Subida de Volumen Controla Pulsador de Bajada de Volumen

Table I: Distribucón Pines Empleados

Para el control de la pantalla se empleo la comunicación SPI, en la cual se emplearon las conexiones se aprecia en la figura 2.

Figure 2: Conexión SPI

Para esta conexión es importante considerar, que por defecto en la comunicación SPI se tiene una velocidad de 8 MHz, sin embargo esto presenta un gran problema a la hora de imprimir en tiempo real la señal que se esta reconstruyendo a la salida del ADC, puesto que el tiempo de escritura en la pantalla no es lo suficientemente rápido y genera que se corte un poco la señal de audio, para corregir esto se propone, optimizar en primer lugar la cantidad de elementos a imprimir en pantalla, en segundo lugar se sube la frecuencia de funcionamiento de esta comunicación hasta los 12 MHz.

Posteriormente se procedió a desarrollar e implementar con la ayuda del entorno de desarrollo de arduino IDE el código adecuado para el funcionamiento del digitalizador de audio y el cual podemos apreciar en el anexo 1.

IV. Resultados

Se desarrollo el siguiente montaje :

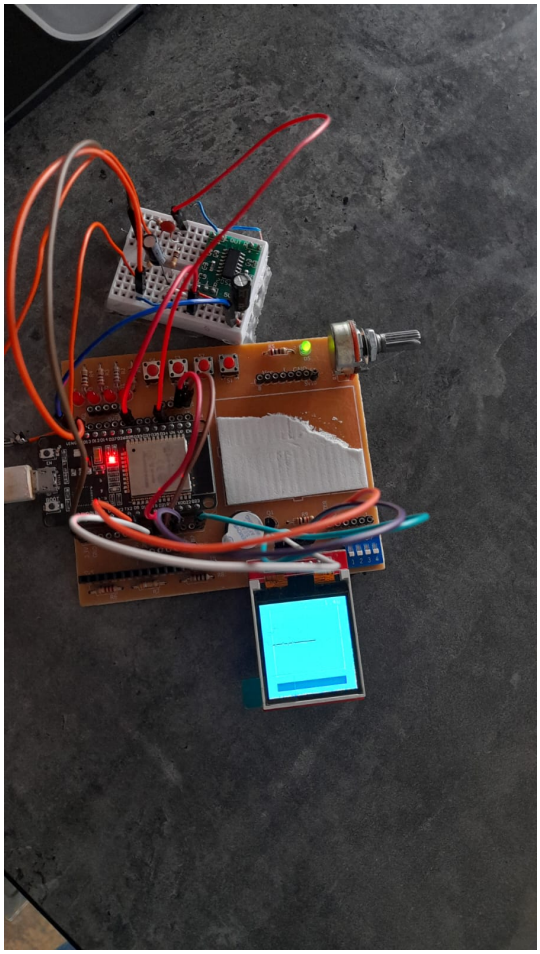

Figure 3: Montaje Final

En la figura 3 se aprecia el montaje final realizado, en la protoboard de encuentra el circuito de entrada de acondicionamiento para la señal, así como el circuito de potencia de salida del DAC hacia cualquier parlante.

V. Conclusiones

- El tiempo de impresión en pantalla afecta directamente al comportamiento del sistema, ya que si se desea realizar estas actividades al mismo, se presentaran problemas con la impresión en tiempo real, ya que la lectura del ADC, su digitalización y posterior reconstrucción para el DAC consume un determinado tiempo que se vera cortado por la impresión en pantalla.
- El ruido y cortes que se presenta en el proceso de la digitalización del audio y su posterior reconstrucción se debe a la frecuencia de muestreo que se consiguió, así como a la interferencia generada por el cableado entre los pines y el circuito externo empelado para el acondicionamiento para el procesamiento de la señal.
- Gracias a que esta tarjeta tiene la posibilidad de ser programada en el entorno de desarrollo correspondiente a Arduino IDE el cual utiliza C como lenguaje de programación y nos facilita diversas librerías que son de gran utilidad al momento de usar diferentes características o componentes externos como por ejemplo la pantalla TFT y su implementación de la comunicación SPI, etc.
- Comprender el manejo del ADC, DAC y comunicación empleando SPI en el ESP32 es de vital importancia para el desarrollo de un digitalizador de audio, en donde tener claras las resoluciones máximas, frecuencia de muestreo y de digitalización disponibles en el manejo con ESP32 permiten desarrollar un procesamiento de audio casi ideal.

VI. Anexos

```
#include <SPI.h>
#include <TFT_ILI9163C.h>
#include <Adafruit_GFX.h>

//Pin RESET conectado a VCC (+3.3v)
//Pin LED conectado a VCC (+3.3v)
#define CS 5 //pin CS0
#define DC 3 //pin A0
#define MOSI 23 //pin SDA (predefinido en la libreria SPI)
#define SCK 18 //pin CLK (predefinido en la libreria SPI)

TFT_ILI9163C tft=TFT_ILI9163C(CS,DC); //Se crea el objeto de nombre "tft"
//Pines para controlar el volumen
#define subirVol 21
#define bajarVol 19

//Definici n de colores en formato RGB565
#define BLACK   0x0000
#define WHITE   0xFFFF
#define RED     0xF800 //0b1111100000000000
#define GREEN   0x07E0 //0b0000011111100000
#define BLUE    0x001F //0b0000000000011111
#define YELLOW  0xFFE0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define ORANGE  0xFD20

#define ADC1_CH6 34 //Canal 6 del ADC1 del ESP32 en GPIO34
#define DAC1 25 // DAC canal

uint8_t xiPos=15,yiPos=2,xPos,yPos,muestreo,maxim;
uint32_t out = 0;
uint16_t volt;
uint16_t voltEsc;


volatile uint8_t contador=0;

void screenPlantilla(){
  tft.fillScreen(CYAN);
  tft.drawLine(15,10,15, 100, WHITE);
  tft.drawLine(15,10,115, 10, WHITE);
  tft.drawLine(15,100,115, 100, WHITE);
  tft.drawLine(115,10,115, 100, WHITE);
  tft.drawLine(15,110,115, 110, BLUE);
  tft.drawLine(15,110,15, 120, BLUE);
  tft.drawLine(115,110,115,120, BLUE);
  tft.drawLine(15,120,115, 120, BLUE);
  if(contador<10){
    for(int i=10;i>(9-contador);i--){
      tft.fillRect(15+10*(10-i),110,10,10,BLUE);
    }
  }

}

// TIMER
volatile uint8_t bandera;
hw_timer_t *timer=NULL;
void IRAM_ATTR Int_Timer(){
  tft.setCursor(97,4);
  tft.print(volt*(3.3/4095.0)); //Escalizada medici n del ADC en milivoltios
  tft.drawPixel(xPos,yPos+maxim-voltEsc,BLACK); //Dibujar el punto asociado al valor leido del ADC
  if(xPos>=115){
    xPos=xiPos;
    screenPlantilla();
  }else{
    xPos++;
  }
}
```

```
//Pulsadores
volatile uint8_t Rebote;
volatile uint8_t bandera1;
volatile uint8_t bandera2;

void IRAM_ATTR subirVolumen(){
  Rebote = millis();
  bandera1 = 1;
}
void IRAM_ATTR bajarVolumen(){
  Rebote = millis();
  bandera2 = 1;

}

void setup() {
  tft.begin(); //Inicializar la TFT
  tft.setTextColor(WHITE,CYAN);
  xPos=xiPos;
  yPos=yiPos;
  muestreo=100;
  maxim=100;
  volt = 0;
  voltEsc = 0;
  screenPlantilla();
  analogReadResolution(12);
  //timmer
  timer=timerBegin(0,80,true); //Configurar Timer 0 con preesc. x80 (1MHz) y conteo ascendente
  timerAlarmWrite(timer,200000,true); //Timer 0 conf. a 1s (2000000us) y con autorecarga
  timerAttachInterrupt(timer,Int_Timer,true); //Hab. int. del Timer 0
  timerAlarmEnable(timer); //Habilitar el Timer 0
  //Pulsadores Volumen
  pinMode(subirVol,INPUT_PULLUP);
  pinMode(bajarVol, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(subirVol),subirVolumen,FALLING);
  attachInterrupt(digitalPinToInterrupt(bajarVol),bajarVolumen,FALLING);
}

void loop() {
  volt=analogRead(ADC1_CH6); //Lee valor del ADC
  voltEsc=map(volt,0,4095,0,maxim); //Escaliza valor leido del ADC para graficar
  out = volt/16;
  dacWrite(DAC1, out*(0.1*(contador+1)));
  if((millis()-Rebote)>300&bandera1){
    if(contador<9){
      contador++;
      tft.fillRect(15+10*contador,110,10,10,BLUE);
    }
    bandera1 = 0;
  }
  if((millis()-Rebote)>300&bandera2){
    if(contador>0){
      tft.fillRect(15+10*contador,111,10,9,CYAN);
      contador--;
    }
    bandera2 = 0;
  }
}
```

Listing 1: Código Proyecto 2