EQ EMULATOR

SISTEMA DE PROCESAMIENTO DIGITAL DE SEÑALES EN TIEMPO REAL BASADO EN FILTROS ADAPTATIVOS PARA EL MODELADO DE SISTEMAS ELECTROACÚSTICOS

PROYECTO FINAL

ANEXOS

Versión 1.0
9/02/2024

INFORMACIÓN DEL PROYECTO

Autor	
Nombre completo	Francisco Julián Farré
Legajo	42945
e-mail	franjulianfarre@gmail.com

Tutor	Dr. Ing. Rodrigo Gonzalez
Director	Dr. Ing. Rodrigo Gonzalez
Jurado	Ing. Cesar Boschi
Año Académico	2024
Responsable de la cátedra	Ing. Ana Lattuca

Empresa / Cliente / Laboratorio	Laboratorio de Acústica “Mario Guillermo Camín" UTN - FRM
Patrocinador (Sponsor)	CEGA Electrónica

1. ÍNDICE

2. ANEXOS

2.1 ANEXO A – CÓDIGO SIMULADOR

A continuación, se tienen los scripts de MATLAB utilizados para la etapa de simulación. Los archivos se pueden descargar del repositorio adjunto en bibliografía.

Modelo_EQ_emulator.m

```
clear;
clc;
%% PARAMETROS
unknown_system=0;  %0:real RI; 1:fixed_FIR; 2:fixed_IIR; 3:analog_filter
Fs=48000; %Frecuencia de muestreo
if unknown_system==0
    RI_folder="RI's/Ampeg_V4B_Marshall_4x12";
    RI_filename='Ampeg V4-B_marshal_sm7.wav';
end

%Parametros del filtro adaptativo
mu = 0.0001; %Coeficiente de convergencia
order=128;  %Orden deseado del filtro adaptativo
lms = dsp.LMSFilter(order,'StepSize',mu,'Method','Normalized LMS'); %Objeto del filtro (tipo NMLS)

%Parametros de simulacion
trainSeconds=100; %Segundos que el sistema estará en fase de adaptación
%% Sistema desconocido
switch unknown_system
    case 0 %real_RI
        [RI,~]=audioread(RI_folder+'/'+RI_filename);
        RI=RI(:,1);
        numerator=RI;
        denominator=1;
        [imp_orig,t_orig] = impz(numerator,denominator,length(numerator),Fs);
        [h_orig,f_orig]=freqz(numerator,denominator,10000,Fs);
        mag_orig=10*log10(abs(h_orig));
    case 1 %fixed_FIR
        n=200; %Orden del filtro FIR
        Fc=1000; %Frecuencia de corte
        Wn=Fc/(Fs/2); %Frecuencia de corte normalizada
        numerator=fir1(n,Wn,'low'); %Coeficientes FIR
        denominator=1; %Denominador unitario (por ser FIR)
        [imp_orig,t_orig] = impz(numerator,denominator,length(numerator),Fs);
        [h_orig,f_orig]=freqz(numerator,denominator,10000,Fs);
        mag_orig=10*log10(abs(h_orig));
    case 2 %fixed_IIR
        n=5; %Orden del filtro IIR
        Fc=1000; %Frecuencia de corte
        Wn=Fc/(Fs/2); %Frecuencia de corte normalizada
        [numerator,denominator] = butter(n,Wn); %Diseño filtro butterworth
        [imp_orig,t_orig] = impz(numerator,denominator,1000,Fs);
        [h_orig,f_orig]=freqz(numerator,denominator,10000,Fs);
        mag_orig=10*log10(abs(h_orig));
    case 3 %analog_filter
        n=5; %Orden del filtro
        Fc=1000; %Frecuencia de corte
        Wc = 2*pi*Fc; %Frecuencia de corte angular
        [numerator,denominator] = butter(n,Wc,'s');
        sys = tf(numerator,denominator); %Función de transferencia del filtro
        [imp_orig_un,t_orig]=impulse(sys);
        figure
        plot(t_orig,imp_orig_un),grid on, title 'Respuesta al impulso sistema desconocido'
        xlabel('Time (s)')
        ylabel('Magnitude')
        W=0:((pi*Fs)/10000):(pi*Fs);
        f_orig=W/(2*pi);
        h=freqs(numerator,denominator,W);
        mag_orig = 10*log10(abs(h));
        phase = angle(h);
        phasedeg = phase*180/pi;
        figure
        subplot(2,1,1)
        plot(f_orig,mag_orig), grid on, title 'Respuesta en frecuenciaistema desconocido'
        xlabel('Frequency (Hz)')
        ylabel('Magnitude')
        subplot(2,1,2)
        plot(W/(2*pi),phasedeg), grid on
        xlabel('Frequency (Hz)')
        ylabel('Phase (degrees)')
    otherwise
        disp('Error')
        return
end

if unknown_system~=3
    figure
    plot(t_orig,imp_orig),grid on, title 'Respuesta al impulso sistema desconocido'
    xlabel('Time (s)')
    ylabel('Magnitude')
    figure
    freqz(numerator,denominator,10000,Fs), title 'Respuesta en frecuencia Sistema desconocido'
end
%% ADAPTACIÓN
%Señal de entrada (Ruido blanco)
x = randn(trainSeconds*Fs,1); %Ruido blanco

%Señal deseada (salida del sistema desconocido)
if unknown_system~=3
    d = filter(numerator,denominator,x);
else
    t=0:(1/Fs):((1/Fs)*length(x))-(1/Fs); %Vector de tiempo
    d=lsim(sys,x,t);
end

%Salida del filtro adaptativo (y:salida; e:error; w:coeficientes; d:deseada)
[y,e,w] = lms(x,d); %Entrenamiento del filtro adaptativo
t=0:(1/Fs):((1/Fs)*length(y))-(1/Fs); %Vector de tiempo
figure
plot(t,[d,y,e])
title('Etapa de adaptacion')
```

```
legend('Deseada','Salida','Error')
xlabel('t [s]')
ylabel('Valor')
figure
freqz(w,1,10000,Fs), title 'Filtro adaptado'
figure
stem(w), title 'Coeficientes finales', grid on

%Error cuadrático medio
timeMSE=mse(e); %Cálculo de la variación del error cuadrático medio
t=0:(1/Fs):((1/Fs)*length(y))-(1/Fs); %Vector de tiempo
figure
plot(t,timeMSE), title 'Error cuadratico medio', grid on
xlabel('t [s]')
ylabel('MSE')

%MSE calculado en bloque
blockSize=512;
timeMSE_block=block_mse(e,blockSize); %Cálculo de la variación del error cuadrático medio por bloques (Similar a la implementación práctica)
t=0:(blockSize/Fs):((blockSize/Fs)*length(timeMSE_block))-(blockSize/Fs); %Vector de tiempo
figure
plot(t,timeMSE_block), title 'Error cuadratico medio estimado', grid on
xlabel('t [s]')
ylabel('MSE')

%% Comparacion de filtros
[imp_model,t_model] = impz(w,1,length(w),Fs);
[h_model,f_model]=freqz(w,1,10000,Fs);
mag_model=10*log10(abs(h_model));

if unknown_system==3
    scale_value=max(abs(imp_model))/max(abs(imp_orig_un));
    imp_orig=scale_value*imp_orig_un;
end

figure
plot(t_orig,imp_orig)
hold on
plot(t_model,imp_model,'-o')
grid on
legend('Sistema original','Modelo obtenido')
title('Respuesta al impulso sistema vs modelo')
xlabel('Time (s)')
ylabel('Magnitude')

figure
plot(f_orig,mag_orig)
hold on
plot(f_model,mag_model)
grid on
legend('Sistema original','Modelo obtenido')
title('Comparativa en frecuencia sistema vs modelo')
xlabel('Frequency (Hz)')
ylabel('Magnitude')

%% Algoritmo de ventaneo (no implementado)
[~,I]=max(abs(w));
wind1=ones(I,1);
%wind2=blackman(2*(order-I));
wind2=hamming(2*(order-I));
wind2=wind2(order-I+1:2*(order-I));
wind_as=[wind1; wind2];
w_windowed=w.*wind_as;

figure
stem(wind_as),title 'Ventana aplicada', grid on
figure
stem(w_windowed), title 'Coeficientes con ventaneo', grid on

[imp_model_wind,t_model_wind] = impz(w_windowed,1,length(w_windowed),Fs);
[h_model_wind,f_model_wind]=freqz(w_windowed,1,10000,Fs);
mag_model_wind=10*log10(abs(h_model_wind));

figure
plot(t_orig,imp_orig)
hold on
plot(t_model,imp_model,'-o')
hold on
plot(t_model_wind,imp_model_wind,'-*')
grid on
legend('Sistema original','Modelo obtenido','Modelo con ventaneo')
title('Respuesta al impulso sistema vs modelo')
xlabel('Time (s)')
ylabel('Magnitude')

figure
plot(f_orig,mag_orig)
hold on
plot(f_model,mag_model)
grid on
plot(f_model_wind,mag_model_wind)
grid on
legend('Sistema original','Modelo obtenido','Modelo con ventaneo')
title('Comparativa en frecuencia sistema vs modelo')
xlabel('Frequency (Hz)')
ylabel('Magnitude')

%% FILTRADO
%% Señal de prueba original
[audio,~]=audioread('prueba.wav');
audio1=audio(:,1);
t=0:(1/Fs):((1/Fs)*length(audio1))-(1/Fs);
figure
plot(t,audio1),grid on, title 'Señal de prueba original'
xlabel('t (s)')
sound(audio1,Fs);
%% Señal pasada por el sistema desconocido
if unknown_system~=3
    salidaSist = filter(numerator,denominator,audio1);
else
    salidaSist=lsim(sys,audio1,t);
end
t=0:(1/Fs):((1/Fs)*length(salidaSist))-(1/Fs);
figure
plot(t,salidaSist),grid on, title 'Señal de salida (sistema desconocido)'
xlabel('t (s)')
valormax=max(abs(salidaSist));
salidaSistA=salidaSist/valormax;
sound(salidaSistA,Fs);
%% Señal modelo
salidaModelo = filter(w,1,audio1);
```

```
t=0:(1/Fs):((1/Fs)*length(salidaModelo))-(1/Fs);
figure
plot(t,salidaModelo),grid on, title 'Señal de salida (modelo)'
xlabel('t (s)')
valormax=max(abs(salidaModelo));
salidaModeloA=salidaModelo/valormax;
sound(salidaModeloA,Fs);
```

mse.m

```
function [timeMSE]=mse(error)
    N=length(error);
    timeMSE=zeros(N,1);
    timeMSE(1)=error(1)^2;

    for i = 2:N
        timeMSE(i)=timeMSE(i-1)+(1/i)*((error(i)^2)-timeMSE(i-1));
    end
end
```

block_mse.m

```
function [timeMSE]=block_mse(e,blockSize)
    N=length(e);
    nSamples=floor(N/blockSize);
    timeMSE=zeros(nSamples,1);

    for i=1:nSamples
        timeMSE(i)=0;
        for j=1:blockSize
            timeMSE(i)=timeMSE(i)+e(((i-1)*blockSize)+j)^2;
        end
        timeMSE(i)=timeMSE(i)/blockSize;
    end
end
```

2.2 ANEXO B – CÓDIGO DSP

A continuación, se presenta el programa principal en C del firmware del bloque DSP, implementado sobre un microprocesador STM32F407VG. El proyecto completo se puede descargar del repositorio adjunto en bibliografía.

main.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2022 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

//CMSIS Library
#define ARM_MATH_CM4        //Must be defined here
#include "arm_math.h"

//FLASH Library
#include "MY_FLASH.h"

//Other includes
#include "math.h"
#include <string.h>
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

//Numeric constants
#define CONST_MAX_INT 2147483648.0f      //2^31

//Machine state flags
#define STOP 0
#define ADAPT 1
#define FILTER 2
#define LOOPBACK 3

//Number of filter coefficients
#define FILTER_TAP_NUM 128

//Buffer size
#define BLOCK_SIZE_FLOAT 512
#define BLOCK_SIZE_U16 4*BLOCK_SIZE_FLOAT  //Since we are working with 2 channels, and each 32 bit word is divided in 2 words of 16 bits

//Vumeter levels
#define MAX_VOLUME_LEVEL 0.33f
#define HIGH_THRESHOLD 0.95*MAX_VOLUME_LEVEL
#define MED_THRESHOLD 0.18*MAX_VOLUME_LEVEL
#define LOW_THRESHOLD 0.01*MAX_VOLUME_LEVEL

//FLASH
#define FLASH_SECTOR 11
#define FLASH_START_ADRESS  0x080E0000

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 I2S_HandleTypeDef hi2s2;
DMA_HandleTypeDef hdma_i2s2_ext_rx;
DMA_HandleTypeDef hdma_spi2_tx;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_rx;

/* USER CODE BEGIN PV */

//Machine state variables
uint8_t state=STOP;

//I2S VARIABLES

uint16_t rxBuf[BLOCK_SIZE_U16 * 2];       //Input buffer
uint16_t txBuf[BLOCK_SIZE_U16 * 2];       //Output buffer
float l_buf_in[BLOCK_SIZE_FLOAT * 2];     //Float left input buffer
float r_buf_in[BLOCK_SIZE_FLOAT * 2];     //Float right input buffer
float l_buf_out[BLOCK_SIZE_FLOAT * 2];    //Float left output buffer
float r_buf_out[BLOCK_SIZE_FLOAT * 2];    //Float right output buffer
uint8_t callback_state = 0;  //Allows circular buffer
```

```
//I2S data pointers
int offset_r_ptr;           //uint_16 buffer read pointer
int offset_w_ptr, w_ptr;    //float buffer write pointer

//GENERAL FILTER VARIABLES

static float filter_taps[FILTER_TAP_NUM] = { 0 };      //Filter coefficients array

//ADAPTIVE FILTER VARIABLES

arm_lms_norm_instance_f32 adaptive_filter;             //Adaptive filter instance
float mu = 0.001f;          //Mu step of adaptive filter
float mse=0;    //Mean Square Error
float filter_state[BLOCK_SIZE_FLOAT + FILTER_TAP_NUM - 1];          //Adaptive filter state (size is always number_of_samples + number_of_fir_tabs - 1)

//FIR FILTER VARIABLES

arm_fir_instance_f32 firsettings_l, firsettings_r;
float fir_l_state[BLOCK_SIZE_FLOAT + FILTER_TAP_NUM - 1];           //Fir state size is always number_of_samples + number_of_fir_tabs - 1
float fir_r_state[BLOCK_SIZE_FLOAT + FILTER_TAP_NUM - 1];           //Fir state size is always number_of_samples + number_of_fir_tabs - 1

//UART VARIABLES

uint8_t uart_buffer_tx[16];  //UART Tx buffer
uint8_t uart_buffer_rx[32] = {0};         //UART Rx buffer
uint8_t received_command[] = "";
uint8_t rxByte;
uint16_t rxIdx;
uint8_t rxFlag;


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_I2S2_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
void receiveString(void);
void interpretCommand(void);
void AudioProcess(void);
void TransmitCoefs(void);
void AdaptInit(void);
void FilterInit(void);
void Stop(void);
void Adapt(void);
void Filter(void);
void Loopback(void);
void vumeterR(void);
void vumeterL(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_I2S2_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  //Wait for other devices to initialize
  HAL_Delay(1000);

  //Define FLASH area
  MY_FLASH_SetSectorAddrs(FLASH_SECTOR, FLASH_START_ADRESS);

  //Start i2s with 2048 samples transmission => 4096*u16 words
  HAL_I2SEx_TransmitReceive_DMA(&hi2s2, txBuf, rxBuf, BLOCK_SIZE_U16);

  //Start UART Rx
  __HAL_UART_FLUSH_DRREGISTER(&huart2);
  HAL_UART_Receive_DMA(&huart2, &rxByte, 1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if (callback_state != 0)
    {
                AudioProcess();
    }
```

```
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2S2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2S2_Init(void)
{

  /* USER CODE BEGIN I2S2_Init 0 */

  /* USER CODE END I2S2_Init 0 */

  /* USER CODE BEGIN I2S2_Init 1 */

  /* USER CODE END I2S2_Init 1 */
  hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_24B;
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_ENABLE;
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_48K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
  if (HAL_I2S_Init(&hi2s2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2S2_Init 2 */

  /* USER CODE END I2S2_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */
```

```
}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream3_IRQn);
  /* DMA1_Stream4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream4_IRQn);
  /* DMA1_Stream5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, L_LOW_LED_Pin|L_MED_LED_Pin|L_HIGH_LED_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, R_LOW_LED_Pin|R_MED_LED_Pin|R_HIGH_LED_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : L_LOW_LED_Pin L_MED_LED_Pin L_HIGH_LED_Pin */
  GPIO_InitStruct.Pin = L_LOW_LED_Pin|L_MED_LED_Pin|L_HIGH_LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : R_LOW_LED_Pin R_MED_LED_Pin R_HIGH_LED_Pin */
  GPIO_InitStruct.Pin = R_LOW_LED_Pin|R_MED_LED_Pin|R_HIGH_LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
void HAL_I2SEx_TxRxHalfCpltCallback(I2S_HandleTypeDef *hi2s) {

   callback_state = 1;
}

void HAL_I2SEx_TxRxCpltCallback(I2S_HandleTypeDef *hi2s) {

   callback_state = 2;
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
   if (state==ADAPT)
   {
               sprintf((char *)uart_buffer_tx,"&%.5f#\n",mse);      //Copy new MSE in UART buffer
               HAL_UART_Transmit_IT(&huart2, uart_buffer_tx, strlen((char *) uart_buffer_tx));
   }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
   if (huart->Instance == USART2) {
               __HAL_UART_FLUSH_DRREGISTER(&huart2);
               if (rxByte == '#' || rxIdx >= 31)
               {
                           uart_buffer_rx[rxIdx] = rxByte;
                           receiveString();
                           interpretCommand();
                           for (int i = 0; i < 24; i++)
                                       uart_buffer_rx[i] = 0;
                           rxIdx = 0;
               }
               else
               {
                           uart_buffer_rx[rxIdx] = rxByte;
                           rxIdx++;
               }
   }
}

void receiveString()
{
   char *p1, *p2;
   p1 = strstr((char*)uart_buffer_rx, "%");
   if (p1) {
               p2 = strstr(p1, "#");
               if (p2)
                           snprintf((char*)received_command, p2 - p1 + 1, "%.*s", (int) (p2 - p1 - 1), (p1 + 1));
   }
}

void interpretCommand()
{
    if (strcmp((char*)received_command,"ST")==0)
```

```
    {
                state=STOP;
    }
    else if (strcmp((char*)received_command,"AF")==0)
    {
                mu=0.001f;
                AdaptInit();
                state=ADAPT;
    }
    else if (strcmp((char*)received_command,"AS")==0)
    {
                mu=0.0001f;
                AdaptInit();
                state=ADAPT;
    }
    else if (strcmp((char*)received_command,"SV")==0)
   {
                MY_FLASH_WriteN(0, filter_taps, FILTER_TAP_NUM, DATA_TYPE_32);
   }
    else if (strcmp((char*)received_command,"FI")==0)
    {
                state=FILTER;
                FilterInit();
    }
    else if (strcmp((char*)received_command,"LB")==0)
   {
                state=LOOPBACK;
   }
}

void AdaptInit()
{
   mse=0;

   for (int i=0; i<FILTER_TAP_NUM; i++)
   {
                filter_taps[i]=0;
   }

   //Init adaptive filter structure
   arm_lms_norm_init_f32(&adaptive_filter, FILTER_TAP_NUM, &filter_taps[0], &filter_state[0], mu, BLOCK_SIZE_FLOAT);

   //Initiate MSE transmission
   sprintf((char *)uart_buffer_tx,"&%.5f#\n",mse);
   HAL_UART_Transmit_IT(&huart2, uart_buffer_tx, strlen((char *) uart_buffer_tx));
}

void FilterInit()
{
   //Read filter stored value
   MY_FLASH_ReadN(0, filter_taps, FILTER_TAP_NUM, DATA_TYPE_32);

   //Init FIR structure
   arm_fir_init_f32(&firsettings_l, FILTER_TAP_NUM, &filter_taps[0], &fir_l_state[0], BLOCK_SIZE_FLOAT);
   arm_fir_init_f32(&firsettings_r, FILTER_TAP_NUM, &filter_taps[0], &fir_r_state[0], BLOCK_SIZE_FLOAT);
}

void AudioProcess()
{
   //Decide if it was half or cplt callback
   if (callback_state == 1)  //First half of the buffer
   {
                offset_r_ptr = 0;
                offset_w_ptr = 0;
                w_ptr = 0;
   }
   else if (callback_state == 2)          //Second half of the buffer
   {
                offset_r_ptr = BLOCK_SIZE_U16;
                offset_w_ptr = BLOCK_SIZE_FLOAT;
                w_ptr = BLOCK_SIZE_FLOAT;
   }

   //Restore input sample buffer to float array
   for (int i = offset_r_ptr; i < offset_r_ptr + BLOCK_SIZE_U16; i = i + 4)
   {
                l_buf_in[w_ptr] = ((float) ((int) (rxBuf[i] << 16) | rxBuf[i + 1]))/CONST_MAX_INT;
                r_buf_in[w_ptr] = ((float) ((int) (rxBuf[i + 2] << 16) | rxBuf[i + 3]))/CONST_MAX_INT;
                w_ptr++;
   }

   vumeterR();
   vumeterL();

   switch (state)
   {
   case STOP: Stop();
                break;
   case ADAPT:  Adapt();
                break;
   case FILTER: Filter();
                break;
   case LOOPBACK: Loopback();
                break;
   default:
                break;
   }

   //Restore processed float-array to output sample-buffer
   w_ptr = offset_w_ptr;

   //Write I2S output buffer
   for (int i = offset_r_ptr; i < offset_r_ptr + BLOCK_SIZE_U16; i = i + 4)
   {
                l_buf_out[w_ptr]=l_buf_out[w_ptr]*CONST_MAX_INT;
                r_buf_out[w_ptr]=r_buf_out[w_ptr]*CONST_MAX_INT;

                txBuf[i] = (((int) l_buf_out[w_ptr]) >> 16) & 0xFFFF;
                txBuf[i + 1] = ((int) l_buf_out[w_ptr]) & 0xFFFF;
                txBuf[i + 2] = (((int) r_buf_out[w_ptr]) >> 16) & 0xFFFF;
                txBuf[i + 3] = ((int) r_buf_out[w_ptr]) & 0xFFFF;
                w_ptr++;
   }

   callback_state = 0;
}
```

```
void Stop()
{
  for (int i=offset_w_ptr; i<offset_w_ptr+BLOCK_SIZE_FLOAT; i=i+1)
		{
			l_buf_out[i]=0;
			r_buf_out[i]=0;
		}
}

void Adapt()
{
  //Adaptive Filter

  //right In  --> input signal (x)
  //left In   --> desired signal (d)
  //right Out --> output signal (y)
  //left Out  --> error signal (e)

  //arm_lms_f32 (const arm_lms_instance_f32 *S, const float32_t *pSrc, float32_t *pRef, float32_t *pOut, float32_t *pErr, uint32_t blockSize)

  arm_lms_norm_f32(&adaptive_filter,    &r_buf_in[offset_w_ptr],    &l_buf_in[offset_w_ptr],    &r_buf_out[offset_w_ptr],    &l_buf_out[offset_w_ptr],
BLOCK_SIZE_FLOAT);

  //MSE calculation

  mse=0;
  for (int i=offset_w_ptr; i<offset_w_ptr+BLOCK_SIZE_FLOAT; i=i+1)
  {
		mse=mse+((float)l_buf_out[i]*(float)l_buf_out[i]);
  }
  mse=mse/(float)BLOCK_SIZE_FLOAT;
}

void Filter()
{
  //FIR
  arm_fir_f32 (&firsettings_l, &l_buf_in[offset_w_ptr], &l_buf_out[offset_w_ptr],BLOCK_SIZE_FLOAT);
  arm_fir_f32 (&firsettings_r, &r_buf_in[offset_w_ptr], &r_buf_out[offset_w_ptr],BLOCK_SIZE_FLOAT);
}

void Loopback()
{
  for (int i=offset_w_ptr; i<offset_w_ptr+BLOCK_SIZE_FLOAT; i=i+1)
		{
			l_buf_out[i]=l_buf_in[i];
			r_buf_out[i]=r_buf_in[i];
		}
}

void vumeterR()
{
  float rms_value;
  arm_rms_f32 (&r_buf_in[offset_w_ptr], BLOCK_SIZE_FLOAT, &rms_value);

  if (rms_value>=LOW_THRESHOLD && rms_value<MED_THRESHOLD)
		{
			HAL_GPIO_WritePin(GPIOD, R_LOW_LED_Pin, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOD, R_MED_LED_Pin, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOD, R_HIGH_LED_Pin, GPIO_PIN_RESET);
		}
		else if (rms_value>=MED_THRESHOLD && rms_value<HIGH_THRESHOLD)
		{
			HAL_GPIO_WritePin(GPIOD, R_LOW_LED_Pin, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOD, R_MED_LED_Pin, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOD, R_HIGH_LED_Pin, GPIO_PIN_RESET);
		}
		else if (rms_value>=HIGH_THRESHOLD)
		{
			HAL_GPIO_WritePin(GPIOD, R_LOW_LED_Pin, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOD, R_MED_LED_Pin, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOD, R_HIGH_LED_Pin, GPIO_PIN_SET);
		}
		else
		{
			HAL_GPIO_WritePin(GPIOD, R_LOW_LED_Pin, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOD, R_MED_LED_Pin, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOD, R_HIGH_LED_Pin, GPIO_PIN_RESET);
		}
}

void vumeterL()
{
  float rms_value;
  arm_rms_f32 (&l_buf_in[offset_w_ptr], BLOCK_SIZE_FLOAT, &rms_value);

  if (rms_value>=LOW_THRESHOLD && rms_value<MED_THRESHOLD)
  {
		HAL_GPIO_WritePin(GPIOC, L_LOW_LED_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOC, L_MED_LED_Pin, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOC, L_HIGH_LED_Pin, GPIO_PIN_RESET);
  }
  else if (rms_value>=MED_THRESHOLD && rms_value<HIGH_THRESHOLD)
  {
		HAL_GPIO_WritePin(GPIOC, L_LOW_LED_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOC, L_MED_LED_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOC, L_HIGH_LED_Pin, GPIO_PIN_RESET);
  }
  else if (rms_value>=HIGH_THRESHOLD)
  {
		HAL_GPIO_WritePin(GPIOC, L_LOW_LED_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOC, L_MED_LED_Pin, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOC, L_HIGH_LED_Pin, GPIO_PIN_SET);
  }
  else
  {
		HAL_GPIO_WritePin(GPIOC, L_LOW_LED_Pin, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOC, L_MED_LED_Pin, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOC, L_HIGH_LED_Pin, GPIO_PIN_RESET);
  }
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
```

```c
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
   /* User can add his own implementation to report the HAL error return state */
   __disable_irq();
   while (1) {
   }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

2.3 ANEXO C – CÓDIGO CONTROL DE USUARIO

A continuación, se presenta el programa en C++ del firmware del bloque control de usuario, implementado sobre un microprocesador Dual core Tensilica Xtensa LX6 (ESP32). El proyecto completo se puede descargar del repositorio adjunto en bibliografía.

GUI

```
#include "AudioTools.h"
#include "Free_Fonts.h" // Include the header file attached to this sketch

#include <SPI.h>
#include <TFT_eSPI.h> // Hardware-specific library

TFT_eSPI tft = TFT_eSPI();       // Invoke custom library

//State machine variables
#define MENU 0
#define ADAPT_INSTRUCTIONS_1 1
#define ADAPT_INSTRUCTIONS_2 2
#define ADAPT_SET_VOLUME 3
#define ADAPT_SELECT 4
#define ADAPT 5
#define ADAPT_SAVE 6
#define FILTER_INSTRUCTIONS 7
#define FILTER 8
#define LOOPBACK_INSTRUCTIONS 9
#define LOOPBACK 10

int gui_state;

// Display parameters

#define ROTATION_UP 1 //Pins at right
#define ROTATION_DOWN 3 //Pins at left

uint8_t rotation = ROTATION_DOWN;

// Touch constraints and variables
#define TOUCH_THRESHOLD 250
#define RAW_LOW_LIMIT 150
#define RAW_HIGH_LIMIT 4000

uint16_t x, y;
bool pressed;

//MSE Plot parameters
#define draw_step 20  //Plots one out of draw_step samples

#define scan 20000.0f //Time plotted in miliseconds
#define sample_time 5.37f //Time between samples in miliseconds

#define offset_x 20
#define offset_y 220
#define scale_x (220/N_samples)
#define scale_y 1.7

const float N_samples = round(scan / sample_time);
int value_y;
int value_x;

int samples_plot = 0;
int count_display_mse = draw_step;


//Audio variables
float volume = 0.2;  //Sets volume

typedef int16_t sound_t;                                  // sound will be represented as int16_t (with 2 bytes)
uint16_t sample_rate = 48000;
uint8_t channels = 1;                                     // The stream will have 2 channels
NoiseGenerator<sound_t> noise(32000);                      // subclass of SoundGenerator with max amplitude of 32000
GeneratedSoundStream<sound_t> sound(noise);
I2SStream out;
VolumeStream audio_volume(sound);
StreamCopy copier(out, audio_volume);                             // copies sound into i2s

//Serial pins
#define RXD2 22
#define TXD2 23

//MSE Serial data variables
#define MAX_THEORIC_MSE 4.0f  //Mse cannot be greater, if we receive mse larger must be corrupted data
#define ALPHA 0.999f  //Recursive filter parameter

String received_mse;  //String raw data
float mse;  //Float raw data
float mse_rec;  //Filtered data
float mse_max = 0;  //Max value found
float mse_porc; //Percentage MSE (plotted)

float last_output = 0;  //Last sample for recursive filter


void setup(void)
{
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2);

  //I2S Configuration
  auto config = out.defaultConfig(TX_MODE);
  config.sample_rate = sample_rate;
  config.channels = channels;
  config.bits_per_sample = 16;
  config.i2s_format = I2S_STD_FORMAT;
  config.is_master = true;
  config.port_no = 0;
  config.pin_ws = 4;
  config.pin_bck = 16;
```

```
  config.pin_data = 17;
  config.pin_data_rx = 5;
  config.pin_mck = 0;
  config.use_apll = true;
  out.begin(config);

  noise.begin();

  //Set initial volume
  audio_volume.begin(config); // we need to provide the bits_per_sample and channels
  audio_volume.setVolume(volume);

  //Screen init
  tft.init();
  tft.setRotation(rotation);
  drawInit();
  drawMenu();
}

void loop()
{
  readTouch();

  switch (gui_state)
  {
    case MENU:
      {
        if (pressed)
        {
          if ((x > 30 && x < 100) && (y > 80 && y < 150))
          {
            delay(100);
            drawInstructionsAdapt_1();
          }
          else if ((x > 220 && x < 290) && (y > 80 && y < 150))
          {
            delay(100);
            drawInstructionsFilter();
          }
          else if ((x > 125 && x < 195) && (y > 145 && y < 215))
          {
            delay(100);
            drawInstructionsLoopback();
          }
        }
      }
      break;
    case ADAPT_INSTRUCTIONS_1:
      {
        if (pressed)
        {
          if ((x > 10 && x < 40) && (y > 10 && y < 40))
          {
            delay(100);
            drawMenu();
          }
          else if ((x > 110 && x < 210) && (y > 185 && y < 230))
          {
            delay(100);
            drawInstructionsAdapt_2();
          }
        }
      }
      break;
    case ADAPT_INSTRUCTIONS_2:
      {
        if (pressed)
        {
          if ((x > 10 && x < 40) && (y > 10 && y < 40))
          {
            delay(100);
            drawMenu();
          }
          else if ((x > 110 && x < 210) && (y > 185 && y < 230))
          {
            delay(100);
            Serial2.print("%LB#");
            drawSetVolumeAdapt();
          }
        }
      }
      break;
    case ADAPT_SET_VOLUME:
      {
        copier.copy();
        if (pressed)
        {
          if ((x > 10 && x < 40) && (y > 10 && y < 40))
          {
            Serial2.print("%ST#");
            drawMenu();
          }
          else if ((x > 110 && x < 210) && (y > 185 && y < 230))
          {
            Serial2.print("%ST#");
            drawSelectAdapt();
          }
          else if ((x > 210 && x < 260) && (y > 90 && y < 140))
          {
            if (volume < 1.0)
            {
              volume += 0.001;
              audio_volume.setVolume(volume);
              drawVolumeSelect();
            }
          }
          else if ((x > 50 && x < 100 ) && (y > 90 && y < 140))
          {
            if (volume > 0.0)
            {
              volume -= 0.001;
              audio_volume.setVolume(volume);
              drawVolumeSelect();
            }
          }
        }
      }
      break;
    case ADAPT_SELECT:
      {
        if (pressed)
```

```
    {
      if ((x > 10 && x < 40) && (y > 10 && y < 40))
      {
        delay(100);
        drawMenu();
      }
      else if ((x > 40 && x < 140) && (y > 90 && y < 135))
      {
        delay(100);
        mse_max = 0;
        samples_plot = 0;
        Serial2.print("%AF#");
        drawAdapt();
      }
      else if ((x > 180 && x < 280) && (y > 90 && y < 135))
      {
        delay(100);
        mse_max = 0;
        samples_plot = 0;
        Serial2.print("%AS#");
        drawAdapt();
      }
    }
  }
  break;
case ADAPT:
  {
    copier.copy();
    drawAdaptValue();
    if (pressed)
    {
      if ((x > 10 && x < 40) && (y > 10 && y < 40))
      {
        Serial2.print("%ST#");
        drawMenu();
      }
      else if ((x > 280 && x < 310) && (y > 10 && y < 40))
      {
        Serial2.print("%ST#");
        drawSaveAdapt();
      }
      else if ((x > 255 && x < 305) && (y > 100 && y < 150))
      {
        if (volume < 1.0)
        {
          volume += 0.001;
          audio_volume.setVolume(volume);
        }
      }
      else if ((x > 255 && x < 305) && (y > 170 && y < 220))
      {
        if (volume > 0.0)
        {
          volume -= 0.001;
          audio_volume.setVolume(volume);
        }
      }
    }
  }
  break;
case ADAPT_SAVE:
  {
    if (pressed)
    {
      if ((x > 40 && x < 140) && (y > 110 && y < 155))
      {
        Serial2.print("%SV#");
        delay(100);
        drawMenu();
      }
      else if ((x > 180 && x < 280) && (y > 110 && y < 155))
      {
        delay(100);
        drawMenu();
      }
    }
  }
  break;
case FILTER_INSTRUCTIONS:
  {
    if (pressed)
    {
      if ((x > 10 && x < 40) && (y > 10 && y < 40))
      {
        delay(100);
        drawMenu();
      }
      else if ((x > 110 && x < 210) && (y > 185 && y < 230))
      {
        delay(100);
        Serial2.print("%FI#");
        drawFilter();
      }
    }
  }
  break;
case FILTER:
  {
    if (pressed)
    {
      if ((x > 10 && x < 40) && (y > 10 && y < 40))
      {
        delay(100);
        Serial2.print("%ST#");
        drawMenu();
      }
      else if ((x > 120 && x < 200) && (y > 95 && y < 165))
      {
        delay(100);
        Serial2.print("%LB#");
        drawLoopback();
      }
    }
  }
  break;
case LOOPBACK_INSTRUCTIONS:
  {
    if (pressed)
    {
      if ((x > 10 && x < 40) && (y > 10 && y < 40))
      {
```

```
            delay(100);
            drawMenu();
          }
          else if ((x > 110 && x < 210) && (y > 185 && y < 230))
          {
            delay(100);
            Serial2.print("%LB#");
            drawLoopback();
          }
        }
      }
      break;
    case LOOPBACK:
      {
        if (pressed)
        {
          if ((x > 10 && x < 40) && (y > 10 && y < 40))
          {
            delay(100);
            Serial2.print("%ST#");
            drawMenu();
          }
          else if ((x > 120 && x < 200) && (y > 95 && y < 165))
          {
            delay(100);
            Serial2.print("%FI#");
            drawFilter();
          }
        }
      }
      break;
    default: Serial2.print("%ST#");
      break;
  }

}
```

serial_mse

```
float recursiveFilter(float sample)
{
  float output;

  output = ALPHA * last_output + (1 - ALPHA) * sample;
  last_output = output;

  return output;
}

void readSerial()
{
  if (Serial2.available() > 0) {
    received_mse = cleanString(Serial2.readStringUntil('\n'));
    if (received_mse != "-1")
    {
      mse = received_mse.toFloat();
      if (mse < MAX_THEORIC_MSE)
      {
        mse_rec = recursiveFilter(mse);
        if (mse_rec > mse_max)
        {
          mse_max = mse_rec;
        }
        mse_porc = (mse_rec / mse_max) * 100;
      }
    }
  }
}

String cleanString(String received_word)
{
  String interpreted_word = "";
  int start;
  int end;

  start = received_word.indexOf('&');
  end = received_word.indexOf('#', start);

  if ((start != -1) && (end != -1))
  {
    interpreted_word = received_word.substring(start + 1, end);
  }
  else
  {
    interpreted_word = "-1";
  }

  return interpreted_word;
}
```

touch

```
void readTouch()
{
  uint16_t x_raw, y_raw;
  tft.getTouchRaw(&y_raw, &x_raw);
  if  (tft.getTouchRawZ() > TOUCH_THRESHOLD)
    pressed = true;
  else
    pressed = false;
  if (rotation == ROTATION_UP)
  {
    x = map(x_raw, RAW_HIGH_LIMIT, RAW_LOW_LIMIT, 0, 320);
    y = map(y_raw, RAW_HIGH_LIMIT, RAW_LOW_LIMIT, 0, 240);
  }
  else
  {
    x = map(x_raw, RAW_LOW_LIMIT, RAW_HIGH_LIMIT, 0, 320);
    y = map(y_raw, RAW_LOW_LIMIT, RAW_HIGH_LIMIT, 0, 240);
  }
}
```

symbols

```
void drawBackSymbol()
{
  tft.fillCircle(25, 25, 15, TFT_WHITE);

  tft.drawLine(18, 25, 35, 25, TFT_BLACK);
  tft.drawLine(18, 24, 35, 24, TFT_BLACK);
  tft.drawLine(18, 26, 35, 26, TFT_BLACK);

  tft.drawLine(18, 25, 25, 32, TFT_BLACK);
  tft.drawLine(18, 24, 25, 31, TFT_BLACK);
  tft.drawLine(18, 26, 25, 33, TFT_BLACK);

  tft.drawLine(18, 25, 25, 18, TFT_BLACK);
  tft.drawLine(18, 24, 25, 17, TFT_BLACK);
  tft.drawLine(18, 26, 25, 19, TFT_BLACK);
}

void drawVolumeSelect()
{
  tft.fillRect(130, 108, 60, 30, TFT_BLACK);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setTextDatum(MC_DATUM);
  tft.setFreeFont(FSS18);
  tft.drawFloat(volume * 100, 0, 160, 120, GFXFF);
}

void drawVolume()
{
  tft.fillRect(250, 88, 60, 20, TFT_BLACK);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSS9);
  tft.drawFloat(volume * 100, 0, 280, 95, GFXFF);
}

void drawDoneSymbol()
{
  tft.fillCircle(295, 25, 15, TFT_WHITE);

  tft.drawLine(285, 25, 292, 32, TFT_BLACK);
  tft.drawLine(285, 24, 292, 31, TFT_BLACK);
  tft.drawLine(285, 26, 292, 33, TFT_BLACK);

  tft.drawLine(292, 32, 303, 18, TFT_BLACK);
  tft.drawLine(292, 31, 303, 17, TFT_BLACK);
  tft.drawLine(292, 33, 303, 19, TFT_BLACK);
}
```

windows

```
void drawInit()
{
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(45, 62, 230, 40, 10, TFT_BLUE);
  tft.fillRoundRect(50, 67, 220, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB12);
  tft.drawString("EQ EMULATOR", 160, 80, GFXFF);

  delay(1000);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSSB9);
  tft.drawString("Starting      ", 160, 150, GFXFF);

  delay(2000);
  tft.drawString("Starting .    ", 160, 150, GFXFF);
  delay(2000);
  tft.drawString("Starting . .  ", 160, 150, GFXFF);
  delay(2000);
  tft.drawString("Starting . . .", 160, 150, GFXFF);
  delay(2000);
}

void drawMenu()
{
  gui_state = MENU;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(45, 12, 230, 40, 10, TFT_BLUE);
  tft.fillRoundRect(50, 17, 220, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB12);
  tft.drawString("EQ EMULATOR", 160, 30, GFXFF);

  tft.setFreeFont(FSSB9);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);

  tft.fillCircle(65, 115, 53, TFT_BLUE);
  tft.fillCircle(65, 115, 50, TFT_WHITE);
  tft.drawString("ADAPT", 65, 115, GFXFF);

  tft.fillCircle(255, 115, 53, TFT_BLUE);
  tft.fillCircle(255, 115, 50, TFT_WHITE);
  tft.drawString("FILTER", 255, 115, GFXFF);

  tft.fillCircle(160, 180, 53, TFT_BLUE);
  tft.fillCircle(160, 180, 50, TFT_WHITE);
  tft.drawString("LOOPBACK", 160, 180, GFXFF);
}

void drawInstructionsAdapt_1()
{
  gui_state = ADAPT_INSTRUCTIONS_1;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.setTextDatum(ML_DATUM);
  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
```

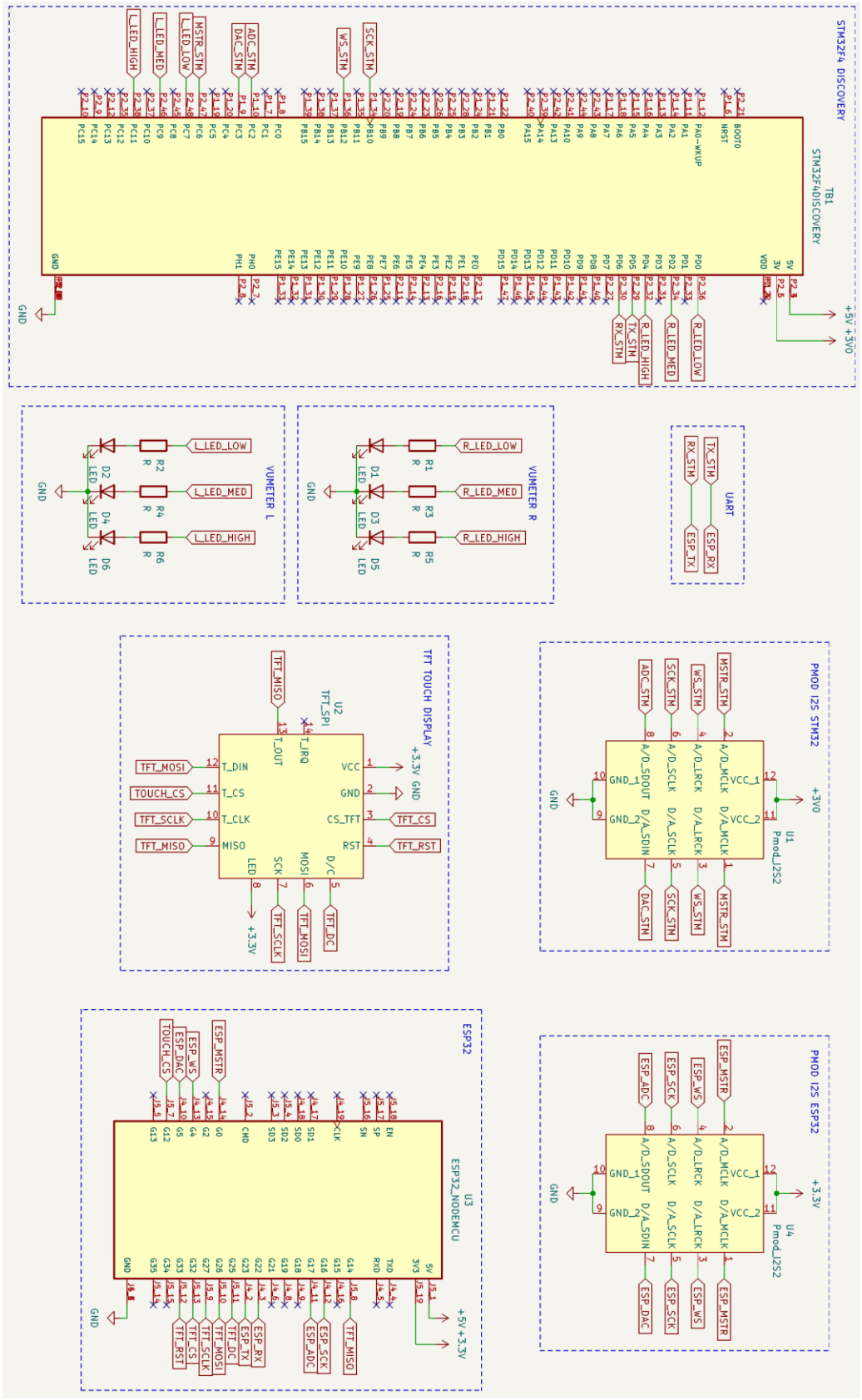
```
  tft.drawString("1) Connect 'Noise output 1'", 10, 60, GFXFF);
  tft.drawString("   to 'Input 1'", 10, 75, GFXFF);
  tft.drawString("2) Connect 'Noise output 2'", 10, 95, GFXFF);
  tft.drawString("   to your sample device", 10, 110, GFXFF);
  tft.drawString("3) Connect your device", 10, 130, GFXFF);
  tft.drawString("   output to 'Input 2'", 10, 145, GFXFF);
  tft.drawString("4) Press 'OK' when done", 10, 165, GFXFF);

  tft.fillRoundRect(105, 180, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(110, 185, 100, 45, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("OK", 160, 205, GFXFF);

  drawBackSymbol();
}

void drawInstructionsAdapt_2()
{
  gui_state = ADAPT_INSTRUCTIONS_2;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.setTextDatum(ML_DATUM);
  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.drawString("5) Check that levels of", 10, 60, GFXFF);
  tft.drawString("   both inputs are similar", 10, 75, GFXFF);
  tft.drawString("6) Use 'Noise Output'", 10, 95, GFXFF);
  tft.drawString("   to regulate 'Input 1'", 10, 110, GFXFF);
  tft.drawString("7) Use your device volume", 10, 130, GFXFF);
  tft.drawString("   to regulate 'Input 2'", 10, 145, GFXFF);
  tft.drawString("4) Press 'OK' when done", 10, 165, GFXFF);

  tft.fillRoundRect(95, 180, 130, 55, 10, TFT_BLUE);
  tft.fillRoundRect(100, 185, 120, 45, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB9);
  tft.drawString("Check levels", 160, 205, GFXFF);

  drawBackSymbol();
}

void drawSetVolumeAdapt()
{
  gui_state = ADAPT_SET_VOLUME;
  volume = 0.2;
  audio_volume.setVolume(volume);
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSS9);
  tft.drawString("Noise Volume", 160,65 , GFXFF);
  drawVolumeSelect();

  tft.fillRoundRect(205, 90, 60, 60, 10, TFT_BLUE);
  tft.fillRoundRect(210, 95, 50, 50, 10, TFT_WHITE);
  tft.fillRoundRect(55, 90, 60, 60, 10, TFT_BLUE);
  tft.fillRoundRect(60, 95, 50, 50, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("+", 230, 115, GFXFF);
  tft.drawString("-", 80, 115, GFXFF);

  tft.fillRoundRect(105, 180, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(110, 185, 100, 45, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("OK", 160, 205, GFXFF);

  drawBackSymbol();
}

void drawSelectAdapt()
{
  gui_state = ADAPT_SELECT;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSSB9);
  tft.drawString("Choose adaptation rate:", 160, 60, GFXFF);

  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB9);
  tft.fillRoundRect(35, 85, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(40, 90, 100, 45, 10, TFT_WHITE);
  tft.drawString("FAST", 90, 112, GFXFF);

  tft.fillRoundRect(175, 85, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(180, 90, 100, 45, 10, TFT_WHITE);
  tft.drawString("SLOW", 230, 112, GFXFF);

  tft.setFreeFont(FS9);
  tft.setTextDatum(ML_DATUM);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.drawString("> Fast convergence", 10, 170, GFXFF);
  tft.drawString("> Less precision", 10, 200, GFXFF);

  tft.drawString("> Slow convergence", 170, 170, GFXFF);
  tft.drawString("> More precision", 170, 200, GFXFF);
```

```
  drawBackSymbol();
}

void drawInstructionsFilter()
{
  gui_state = FILTER_INSTRUCTIONS;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("FILTER", 160, 20, GFXFF);

  tft.setTextDatum(ML_DATUM);
  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.drawString("1) Connect desired source", 10, 75, GFXFF);
  tft.drawString("   to Inputs 1 and 2", 10, 90, GFXFF);
  tft.drawString("2) Connect Outputs 1 and 2", 10, 110, GFXFF);
  tft.drawString("   to audio sink/speaker", 10, 125, GFXFF);
  tft.drawString("4) Press 'OK' when done", 10, 145, GFXFF);

  tft.fillRoundRect(105, 180, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(110, 185, 100, 45, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("OK", 160, 205, GFXFF);

  drawBackSymbol();
}

void drawInstructionsLoopback()
{
  gui_state = LOOPBACK_INSTRUCTIONS;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(95, 2, 130, 40, 10, TFT_BLUE);
  tft.fillRoundRect(100, 7, 120, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("LOOPBACK", 160, 20, GFXFF);

  tft.setTextDatum(ML_DATUM);
  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.drawString("1) Connect desired source", 10, 75, GFXFF);
  tft.drawString("   to Inputs 1 and 2", 10, 90, GFXFF);
  tft.drawString("2) Connect Outputs 1 and 2", 10, 110, GFXFF);
  tft.drawString("   to audio sink/speaker", 10, 125, GFXFF);
  tft.drawString("4) Press 'OK' when done", 10, 145, GFXFF);

  tft.fillRoundRect(105, 180, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(110, 185, 100, 45, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("OK", 160, 205, GFXFF);

  drawBackSymbol();
}

void drawAdapt()
{
  gui_state = ADAPT;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.fillRoundRect(250, 110, 60, 60, 10, TFT_BLUE);
  tft.fillRoundRect(255, 115, 50, 50, 10, TFT_WHITE);
  tft.fillRoundRect(250, 175, 60, 60, 10, TFT_BLUE);
  tft.fillRoundRect(255, 180, 50, 50, 10, TFT_WHITE);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB18);
  tft.drawString("+", 280, 135, GFXFF);
  tft.drawString("-", 280, 200, GFXFF);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSS9);
  tft.drawString("Noise", 280, 55, GFXFF);
  tft.drawString("Volume", 280, 75, GFXFF);
  drawVolume();

  drawBackSymbol();

  drawDoneSymbol();

  drawAdaptPlot();
}

void drawAdaptPlot()
{
  tft.fillRect(0, 45, 250, 200, TFT_BLACK);

  tft.drawLine(20, 92, 240, 92, TFT_DARKGREY);
  tft.drawLine(20, 135, 240, 135, TFT_DARKGREY);
  tft.drawLine(20, 177, 240, 177, TFT_DARKGREY);

  tft.drawLine(196, 50, 196, 220, TFT_DARKGREY);
  tft.drawLine(152, 50, 152, 220, TFT_DARKGREY);
  tft.drawLine(108, 50, 108, 220, TFT_DARKGREY);
  tft.drawLine(64, 50, 64, 220, TFT_DARKGREY);

  tft.drawRect(20, 50, 220, 170, TFT_WHITE);

  tft.setFreeFont(TT1);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setTextDatum(TR_DATUM);
  tft.drawString("100", 20, 50, GFXFF);
  tft.drawString("75", 20, 92, GFXFF);
  tft.drawString("50", 20, 135, GFXFF);
  tft.drawString("25", 20, 177, GFXFF);
  tft.drawString("0", 20, 220, GFXFF);
```

```
  tft.drawFloat(scan / 1000, 0, 245, 225, GFXFF);
  tft.drawFloat(0.8 * (scan / 1000), 0, 200, 225, GFXFF);
  tft.drawFloat(0.6 * (scan / 1000), 0, 155, 225, GFXFF);
  tft.drawFloat(0.4 * (scan / 1000), 0, 113, 225, GFXFF);
  tft.drawFloat(0.2 * (scan / 1000), 0, 70, 225, GFXFF);

  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_BLACK, TFT_GREEN);
  tft.setTextDatum(MC_DATUM);
  tft.drawString("MSE = ", 150, 57, GFXFF);
  tft.drawString("%", 232, 57, GFXFF);
}

void drawAdaptValue()
{
  readSerial();
  if (samples_plot < N_samples)
  {
    value_y = (int)(scale_y * mse_porc);
    value_x = (int)(scale_x * samples_plot);
    digitalWrite(13, !digitalRead(13));   //Utilizado para debuggear el tiempo de recepción del MSE (afecta la constante sample_time)
    if (count_display_mse == draw_step)
    {
      tft.fillCircle(offset_x + value_x, offset_y - value_y, 1, TFT_GREEN);
      tft.setFreeFont(FM9);
      tft.setTextColor(TFT_BLACK, TFT_GREEN);
      tft.fillRect(174, 52, 55, 15, TFT_GREEN);
      tft.drawFloat(mse_porc, 1, 202, 57, GFXFF);
      drawVolume();
      count_display_mse = 0;
    }
    else
    {
      count_display_mse++;
    }
    samples_plot++;
  }
  else
  {
    samples_plot = 0;
    drawAdaptPlot();
  }
}

void drawSaveAdapt()
{
  gui_state = ADAPT_SAVE;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("ADAPT", 160, 20, GFXFF);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSSB9);
  tft.drawString("Save as new filter?", 160, 80, GFXFF);

  tft.setTextColor(TFT_BLACK, TFT_WHITE);
  tft.setFreeFont(FSSB9);
  tft.fillRoundRect(35, 105, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(40, 110, 100, 45, 10, TFT_WHITE);
  tft.drawString("YES", 90, 132, GFXFF);

  tft.fillRoundRect(175, 105, 110, 55, 10, TFT_BLUE);
  tft.fillRoundRect(180, 110, 100, 45, 10, TFT_WHITE);
  tft.drawString("NO", 230, 132, GFXFF);

}

void drawFilter()
{
  gui_state = FILTER;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(105, 2, 110, 40, 10, TFT_BLUE);
  tft.fillRoundRect(110, 7, 100, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("FILTER", 160, 20, GFXFF);

  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);

  tft.drawRect(70, 95, 50, 17, TFT_WHITE);
  tft.drawRect(71, 95, 49, 16, TFT_BLACK);
  tft.fillTriangle(120, 112, 115, 109, 115, 115, TFT_WHITE);
  tft.drawString("Input 1", 70, 85, GFXFF);

  tft.drawRect(70, 148, 50, 17, TFT_WHITE);
  tft.drawRect(71, 149, 49, 16, TFT_BLACK);
  tft.fillTriangle(120, 148, 115, 145, 115, 151, TFT_WHITE);
  tft.drawString("Input 2", 70, 170, GFXFF);

  tft.drawRect(200, 95, 50, 17, TFT_WHITE);
  tft.drawRect(200, 95, 49, 16, TFT_BLACK);
  tft.fillTriangle(250, 95, 253, 100, 247, 100, TFT_WHITE);
  tft.drawString("Output 1", 250, 85, GFXFF);

  tft.drawRect(200, 148, 50, 17, TFT_WHITE);
  tft.drawRect(200, 149, 49, 16, TFT_BLACK);
  tft.fillTriangle(250, 165, 253, 160, 247, 160, TFT_WHITE);
  tft.drawString("Output 2", 250, 170, GFXFF);

  tft.drawRect(120, 95, 80, 70, TFT_WHITE);

  tft.setTextColor(TFT_GREEN, TFT_BLACK);
  tft.setFreeFont(FSSB24);
  tft.drawString("~", 160, 120, GFXFF);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(TT1);
  tft.setFreeFont(FSS9);
  tft.drawString("Press box to change to loopback", 160, 220, GFXFF);

  drawBackSymbol();
}

void drawLoopback()
```

```
{
  gui_state = LOOPBACK;
  tft.fillScreen(TFT_BLACK);
  tft.fillRoundRect(95, 2, 130, 40, 10, TFT_BLUE);
  tft.fillRoundRect(100, 7, 120, 30, 10, TFT_SILVER);
  tft.setTextDatum(MC_DATUM);
  tft.setTextColor(TFT_BLACK, TFT_SILVER);
  tft.setFreeFont(FSSB9);
  tft.drawString("LOOPBACK", 160, 20, GFXFF);

  tft.setFreeFont(FM9);
  tft.setTextColor(TFT_BLACK, TFT_WHITE);

  tft.drawRect(70, 95, 50, 17, TFT_WHITE);
  tft.drawRect(71, 95, 49, 16, TFT_BLACK);
  tft.fillTriangle(120, 112, 115, 109, 115, 115, TFT_WHITE);
  tft.drawString("Input 1", 70, 85, GFXFF);

  tft.drawRect(70, 148, 50, 17, TFT_WHITE);
  tft.drawRect(71, 149, 49, 16, TFT_BLACK);
  tft.fillTriangle(120, 148, 115, 145, 115, 151, TFT_WHITE);
  tft.drawString("Input 2", 70, 170, GFXFF);

  tft.drawRect(200, 95, 50, 17, TFT_WHITE);
  tft.drawRect(200, 95, 49, 16, TFT_BLACK);
  tft.fillTriangle(250, 95, 253, 100, 247, 100, TFT_WHITE);
  tft.drawString("Output 1", 250, 85, GFXFF);

  tft.drawRect(200, 148, 50, 17, TFT_WHITE);
  tft.drawRect(200, 149, 49, 16, TFT_BLACK);
  tft.fillTriangle(250, 165, 253, 160, 247, 160, TFT_WHITE);
  tft.drawString("Output 2", 250, 170, GFXFF);

  tft.drawRect(120, 95, 80, 70, TFT_WHITE);

  tft.drawLine(121, 112, 199, 112, TFT_GREEN);
  tft.drawLine(121, 148, 199, 148, TFT_GREEN);

  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  tft.setFreeFont(FSS9);
  tft.drawString("Press box to change to filter", 160, 220, GFXFF);

  drawBackSymbol();
}
```

2.4 ANEXO D – CIRCUITO ESQUEMÁTICO



2.5 ANEXO E - PCB