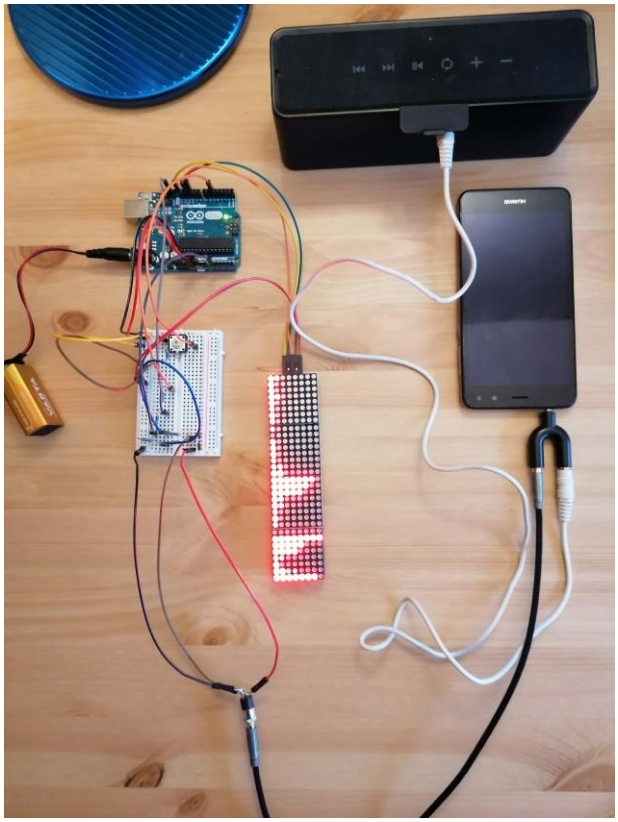


ANALIZADOR DEL ESPECTRO SONORO DE 32 BANDAS

Diseño de Sistemas Empotrados

GRUPO 7
Eric Martínez Gamero

Contenido

1. Introducción

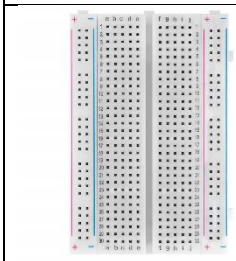
Este proyecto ha sido desarrollado por el grupo 7 de la asignatura Diseño de Sistemas Empotrados, conformado por Eric Martínez Gamero, durante el curso 2021-22.

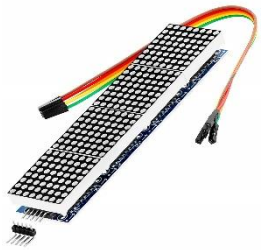
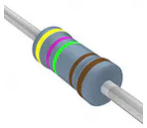
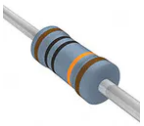
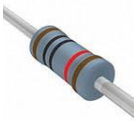
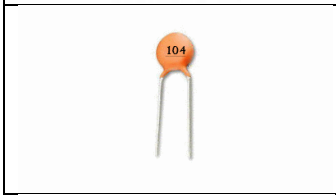
Se trata de un **visualizador LED del espectro de audio de 32 bandas**. El *display* está compuesto por cuatro módulos de 8 x 8 LEDs rojos, que se iluminan en función de la señal analógica digitalizada proveniente de una fuente de sonido.

El objetivo del proyecto es mostrar de diferentes modos las fluctuaciones de frecuencia que se observan en la señal analógica de una canción.

La idea original proviene de la entrada del blog oficial de Arduino publicada por el usuario *shajeeb* en 2019: https://create.arduino.cc/projecthub/shajeeb/32-band-audio-spectrum-visualizer-analyzer-902f51?ref=platform&ref_id=424_trending___&offset=22

2. Componentes

Imagen	Componente	Número	Precio
	Placa Arduino Uno	1	Provisto por la URJC
	Placa de inserción	1	Provisto por la URJC
	Cable de audio estéreo	2	En propiedad
	Cables DuPont	17	8 €

	Matriz LED 8x32 MAX7219	1	10 €
	Socket hembra para jack de 3.5 ''	1	1.37 €
	Splitter de audio 1 a 2 para jack de 3.5''	1	0.50 €
	Resistencias de 4.7k ohm	3	13 €
	Resistencias de 100k ohm	2	
	Resistencia de 10k ohm	1	
	Condensador cerámico 100 nF	2	0.61 €
	Botón	1	Provisto por la URJC
	Conector Pila 9V	1	5 €

	Pila alcalina 9V	1	3 €

3. Herramientas

Imagen	Herramienta	Número	Precio
	Soldador de estaño	1	24 €
	Polímetro	1	En propiedad

4. Hardware

El sistema recibe la señal analógica de un dispositivo móvil gracias a un cable de audio estéreo. Este cable está conectado a un socket hembra que separa los canales izquierdo, derecho y tierra. Con el soldador de estaño se fijaron a los pines del socket tres cables que van directos a la placa de inserción.

Por otro lado, se instaló en la placa un botón para posibilitar el cambio entre los tres modos de visualización en el *display*.

Respecto a la salida del sistema, esta se visualiza a través de una matriz LED 8x32 conectada mediante cables jumper a la placa Arduino y constituida por cuatro módulos 8x8.

En cuanto a la fuente de alimentación, se optó por utilizar el jack de alimentación con adaptador de DC-AC para conectar una batería de 9V.

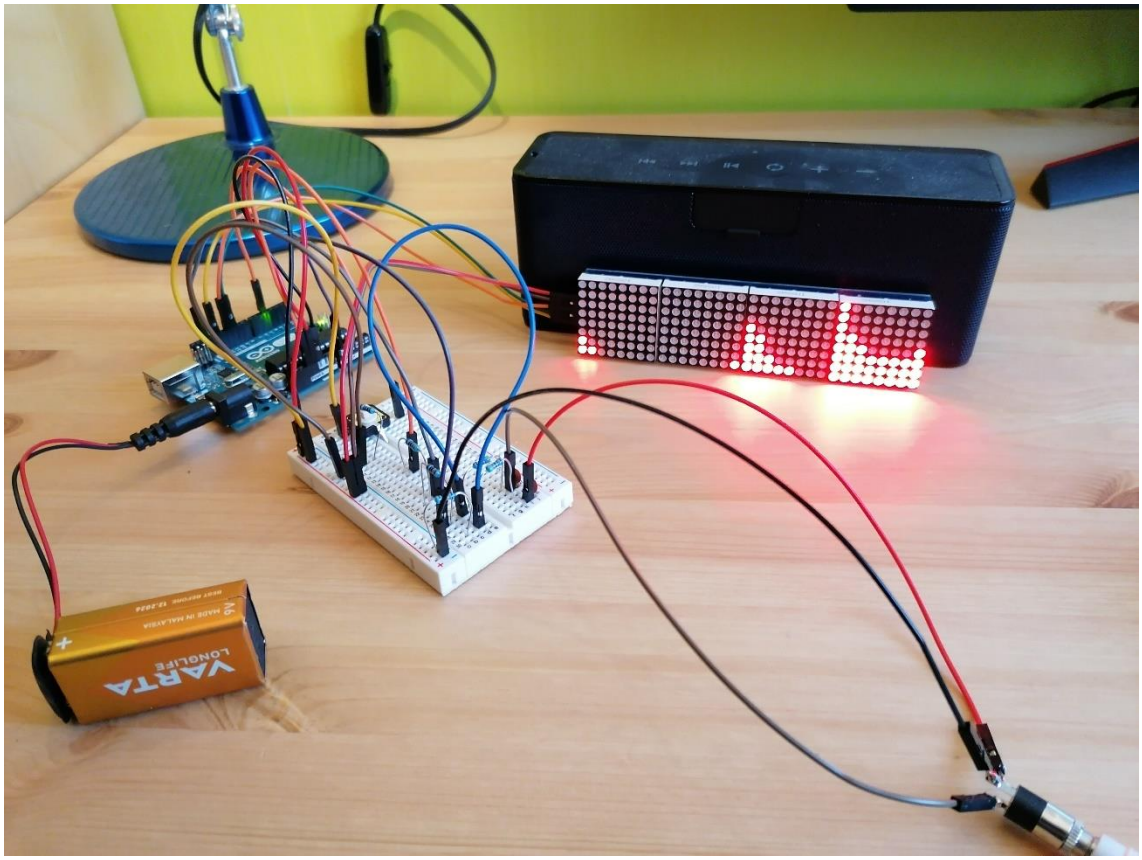


Ilustración 1. Conexiones hardwware

5. Esquema

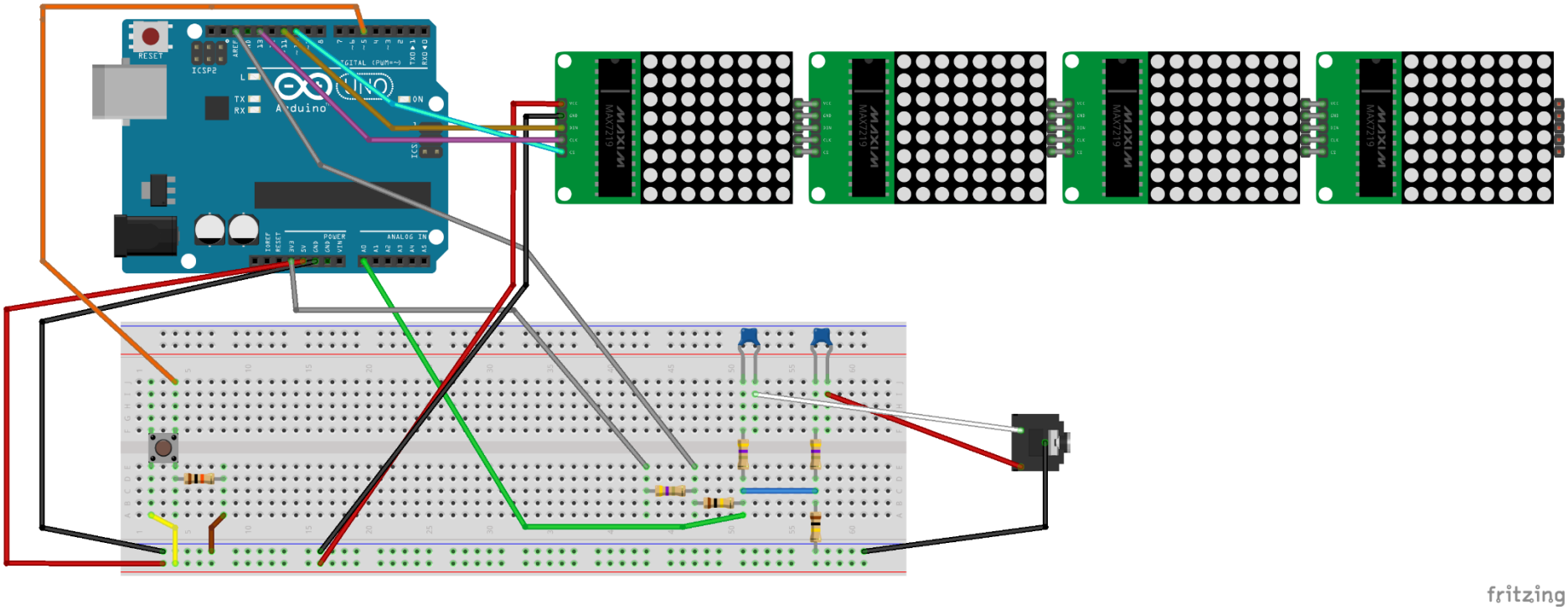


Ilustración 2. Diagrama fritzing del visualizador

6. Software

6.1 Librerías utilizadas

Para la implementación del proyecto a nivel software, se han utilizado las librerías de Arduino MD_MAX72xx.h y arduinoFFT.h.

```
#include <arduinoFFT.h>
#include <MD_MAX72xx.h>
```

La librería **arduinoFFT** permite calcular la frecuencia de una señal muestreada. Implementa el algoritmo de la transformada rápida de Fourier (*Fast Fourier Transform* o FFT) en coma flotante. Esto le permite el tratamiento digital de señales de audio.

Según el algoritmo, se impone que la señal muestreada a transformar debe consistir de un número de muestras potencia de dos.

```
#define MUESTRAS 64
```

Por otro lado, la librería **MD_MAX72XX** implementa funciones para trabajar con matrices LED 8x8. Es necesario realizar un test inicial sobre la matriz comprada para conocer el tipo de hardware y poder especificarlo para la librería.

```
#define TIPO_HARDWARE MD_MAX72XX::ICSTATION_HW
```

Las sentencias siguientes definen una serie de valores relacionados con el *display*: número de módulos, pines, columnas y filas.

```
#define MODULOS  4
#define CLK_PIN   13
#define DATA_PIN  11
#define CS_PIN    10

#define COLUMNAS 32
#define FILAS 8
```

Para usar cualquiera de las librerías antes mencionadas primero se generan los objetos correspondientes para poder invocar funciones:

```
MD_MAX72XX mx = MD_MAX72XX(TIPO_HARDWARE, CS_PIN, MODULOS);
arduinoFFT FFT = arduinoFFT();
```

6.2 Explicación del código

6.2.1 Variables globales

Tras las sentencias define iniciales descritas en el apartado anterior, se declaran una serie de variables globales.

Primero, los modos de visualización del *display*. Estos se encuentran codificados en binario en un array de 9 elementos. En cada posición se encuentra el valor binario que debe tomar la columna específica del *display* para que, de manera progresiva, se enciendan los LEDs necesarios para representar cada valor en escala (cada LED puesto a 1 se iluminará en dicha columna). El primer array es el modo seleccionado, por defecto, será el modo 1:

```
int MODO[]= {0, 128, 192, 224, 240, 248, 252, 254, 255};
int MODO_1[]= {0, 128, 192, 224, 240, 248, 252, 254, 255};
int MODO_2[]= {0, 128, 64, 32, 16, 8, 4, 2, 1};
int MODO_3[]= {0, 1, 3, 7, 15, 31, 63, 127, 255};
```

```
int MODO_4[]= {0, 1, 2, 4, 8, 16, 32, 64, 128};
```

Como ejemplo, el primer modo de visualización se ilustraría de la siguiente manera:

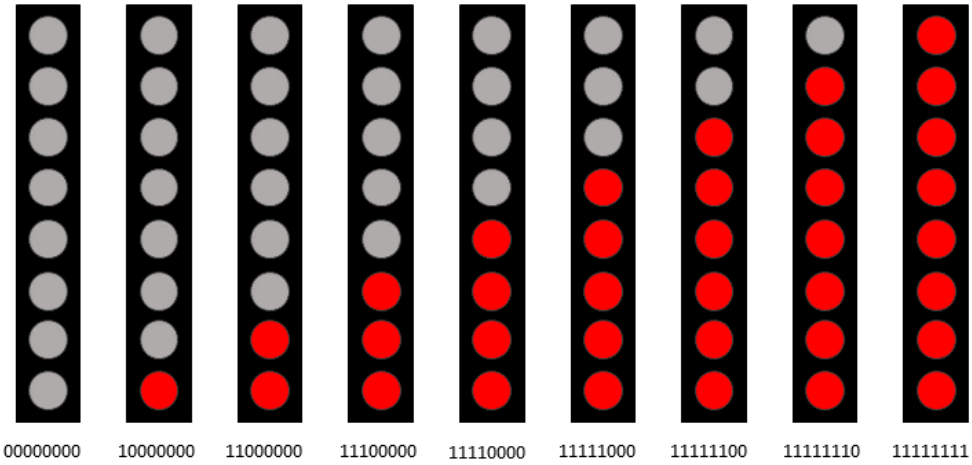


Ilustración 3. Modo de visualización 1

Las variables inmediatamente posteriores son necesarias para almacenar los valores reales e imaginarios resultantes de la FFT para cada una de las muestras:

```
double vReal[MUESTRAS];
double vImag[MUESTRAS];
```

El objetivo de la siguiente variable es almacenar en última instancia, para cada columna, el valor de o a 8 que debe tomar cada columna.

```
char val[COLUMNAS];
```

Los dos enteros que la siguen indicarán la columna a iluminar y el valor en binario de la posición del modo correspondiente.

```
int columnaDisplay, valorDisplay;
```

Las últimas variables globales indican el pin del botón, el estado (HIGH o LOW) actual y el estado previo; todo ello para controlar las pulsaciones. `modo`, por su parte, guarda el modo de visualización actual:

```
const int BOTON_PIN = 5;
int estado = HIGH;
int estadoPrevio = LOW;
int modo = 1;
```

Es importante no olvidar la creación de los objetos display y FFT:

```
MD_MAX72XX mx = MD_MAX72XX(TIPO_HARDWARE, CS_PIN, MODULOS);
arduinoFFT FFT = arduinoFFT();
```

6.2.2 Setup

La primera línea de `setup()` hace pone el conversor analógico digital de la Arduino Uno en free running mode cambiando el valor del registro ADSRA:

```
ADCSRA = 0b11100101;
```

Así se permite que el ADC se mantenga realizando conversiones consecutivas. Cuando solo se tiene un pin y se pretende ahorrar tiempo entre conversiones, se debe utilizar este modo. Esta sentencia también indica el valor del *prescaler* a 32.

```
ADMUX = 0b00000000;
pinMode(BOTON_PIN, INPUT);
mx.begin();
delay(50);
```

Con ADMUX seleccionamos el pin **A0**, después configuramos el pin digital del botón como entrada, inicializamos la matriz de LEDs e introducimos un delay para recoger un voltaje de referencia estable a través de AREF.

6.2.3 Loop

6.2.3.1 Muestreo

```
for(int i=0; i<MUESTRAS; i++)

    {
      while(!(ADCSRA & 0x10));
      ADCSRA = 0b11110101 ;
      int valor = ADC - 512 ;
      vReal[i] = valor/8;
      vImag[i] = 0;
    }
```

Para muestrear la señal de audio, primero se espera con un bucle `while` a que el ADC termine la conversión actual. Una vez finalizada, se debe limpiar el valor de ADIF para que el ADC pueda realizar la siguiente operación. Después se toma el valor del ADC con el offset correspondiente y se escala antes de almacenarlo en el array de valores reales que va a procesar FFT.

6.2.3.2 Configuración de FFT

Para configurar FFT y realizar el cálculo de la transformada, se introducen las tres líneas siguientes:

```
FFT.Windowing(vReal, MUESTRAS, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
FFT.Compute(vReal, vImag, MUESTRAS, FFT_FORWARD);
FFT.ComplexToMagnitude(vReal, vImag, MUESTRAS);
```

6.2.3.3 Cálculo de valores medios

Hemos realizado M muestras, que debemos escalar a C columnas. Para ello, será necesario hacer el cálculo de la media en cada intervalo con dos bucles `for` anidados.

```
int step = (MUESTRAS/2) / COLUMNAS;
int c = 0;
for(int i=0; i < (MUESTRAS/2); i++)
    {
      val[c] = 0;
      for (int k = 0 ; k < step ; k++)
```

```
            val[c] = val[c] + vReal[i+k];
        val[c] = val[c]/step;
        c++;
      }
```

En nuestro caso no sería preciso hacer medias, debido a que el valor del intervalo es 1 (`step` = 1), pero gracias a este código podríamos trabajar con mayor número de muestras.

6.2.3.4 Iluminar la matriz

Por cada columna i se restringe su valor correspondiente entre 0 y 80 y se mapea a un entero entre 0 y el número de filas.

```
for(int i=0; i < COLUMNAS; i++)
    {
      val[i] = constrain(val[i],0,80);
      val[i] = map(val[i], 0, 80, 0, FILAS);
```

Una vez tenemos un valor entero en [0,8], sabemos qué LEDs iluminar accediendo a esta posición en el array de modo actual. Además, conocemos la columna particular que vamos a iluminar (31 – i):

```
      valorDisplay = MODO[val[i]];
      columnaDisplay = 31-i;
```

Finalmente, con `setColumn`, cambiamos el estado de la columna.

```
     mx.setColumn(columnaDisplay, valorDisplay);
    }
```

Antes de reiniciar el `loop()`, hay que comprobar si se ha pulsado el botón para cambiar de modo. La función `cambioModo()` implementa esta funcionalidad:

```
cambioModo ();
```

6.2.4 Función para cambiar de modo

Se trata de una sencilla función que lee el pin de botón, y si este se ha pulsado, procede a cambiar el modo de visualización al siguiente con la sentencia `switch`:

```
int leido = digitalRead(BOTON_PIN);
if (leido == HIGH && estadoPrevio == LOW)
  {
   switch (modo) {
    case 1:    // del modo 1 al 2
      modo = 2;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_2[i];
      }
      break;
    case 2:    // del modo 2 al 3
      modo = 3;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_3[i];
      }
      break;
    case 3:    // del modo 3 al 4
      modo = 4;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_4[i];
```

```
      }
      break;
    case 4:     // del modo 4 al 1
      modo = 1;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_1[i];
      }
      break;
    }
  }
  estadoPrevio = leido;
```

7. Problemas encontrados durante la implementación y soluciones

7.1 Contactos

El principal problema que se presentó durante la realización del proyecto fueron las interferencias electromagnéticas que provocaban la aparición de fluctuaciones en el *display*. Sin conectar una fuente de sonido al sistema, se iluminaría la matriz sin motivo aparente.

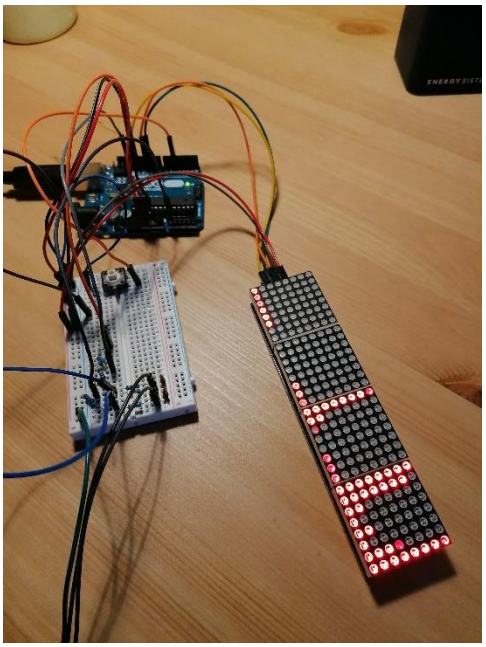


Ilustración 4. Interferencias en el sistema

Esto se debía a que algún o algunos componentes estaban causando interferencias al no hacer contacto con la placa de inserción de manera adecuada. No solo aparecían interferencias al manipular cables o resistencias, sino también al mover las soldaduras en el socket de audio.

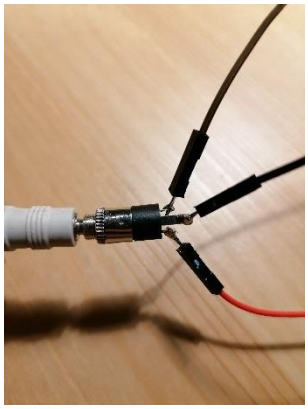


Ilustración 5. Soldadura

Se hizo uso del polímetro para determinar las partes del sistema que provocaban mayores perturbaciones.

Posteriormente, se reemplazaron las resistencias por otras de mayor grosor para lograr mejor contacto con la placa de inserción. Se sustituyeron los cables por unos nuevos y se volvió a realizar la soldadura con un nuevo socket.

Aunque las interferencias no se han eliminado por completo, gracias a estos cambios se han reducido considerablemente. Ahora, si no se manipula el sistema, el módulo LED se mantiene apagado.

7.2 Modo de funcionamiento libre o *free running mode*

El segundo problema surgió al intentar tomar la señal analógica del pin `A0` directamente con una llamada a `analogRead(A0)`:

```
for(int i=0; i < MUESTRAS; i++)
    {
      vReal[i] = analogRead(A0);
      vImag[i] = 0;
    }
```

Esto provoca importantes interferencias y latencias que hacen imposible visualizar la señal. Para solucionarlo, se investigó sobre el tratamiento de señales de audio y se tuvo que modificar el código e incorporar las siguientes líneas en `setup()`:

```
ADCSRA = 0b11100101;
ADMUX = 0b00000000;
```

Por otro lado, para realizar el muestreo se sustituyó la llamada a `analogRead(A0)` por:

```
for(int i=0; i < MUESTRAS; i++)
    {
      while(!(ADCSRA & 0x10));
      ADCSRA = 0b11110101;
      int valor = ADC - 512 ;
      vReal[i] = valor/8;
      vImag[i] = 0;
    }
```

Las nuevas líneas de código permiten poner el conversor analógico a digital (ADC) en **modo de funcionamiento libre**. Cuando se quiere leer de un único pin analógico se puede ahorrar

tiempo entre conversiones configurando el ADC en este modo. Entonces el ADC estará continuamente convirtiendo, es decir, hace una conversión y luego comienza la siguiente. La explicación más detallada de cada línea de código se encuentra en el apartado *6.2. Explicación del código*.

ANEXO

Código completo

```
#include <arduinoFFT.h> // librería que permite calcular la frecuencia
de una señal muestreada
#include <MD_MAX72xx.h> // librería que permite utilizar una matriz
LED como pantalla direccionable por píxeles

#define MUESTRAS 64            // debe ser potencia de dos para el
algoritmo de muestreo
#define TIPO_HARDWARE MD_MAX72XX::ICSTATION_HW   // tipo de display
para que la librería MD_MAX72XX lo reconozca
#define MODULOS  4   // número de módulos LED independientes
#define CLK_PIN   13  // pin CLOCK del display
#define DATA_PIN  11  // pin DATA del display
#define CS_PIN    10  // pin CONTROL del display

#define COLUMNAS 32      // número de columnas en el display (muestras
/ 2)
#define FILAS 8       // número de filas en el display

// modos de visualización - cada LED puesto a 1 en binario se
iluminará
int MODO[]= {0, 128, 192, 224, 240, 248, 252, 254, 255}; // por
defecto será el patrón estándar
int MODO_1[]= {0, 128, 192, 224, 240, 248, 252, 254, 255}; // patrón
estándar
int MODO_2[]= {0, 128, 64, 32, 16, 8, 4, 2, 1}; // solo parte superior
int MODO_3[]= {0, 1, 3, 7, 15, 31, 63, 127, 255}; // patrón estándar
del revés
int MODO_4[]= {0, 1, 2, 4, 8, 16, 32, 64, 128}; // solo parte superior
del revés


double vReal[MUESTRAS];
double vImag[MUESTRAS];
char val[COLUMNAS];

int columnaDisplay, valorDisplay;

const int BOTON_PIN = 5;    // pin del botón
int estado = HIGH;             // el valor actual leído del pin
int estadoPrevio = LOW;   // el valor previo leído del pin
int modo = 1;

MD_MAX72XX mx = MD_MAX72XX(TIPO_HARDWARE, CS_PIN, MODULOS);   //
objeto display
arduinoFFT FFT = arduinoFFT();                                //
objeto fft


void setup() {

    ADCSRA = 0b11100101;      // poner conversor analógico digital en
modo de funcionamiento libre y poner prescaler a 32
```

```
    ADMUX = 0b00000000;       // usar el pin A0 y referencia de
voltaje externa
    pinMode(BOTON_PIN, INPUT);
    mx.begin();               // inicialización del display
    delay(50);                // espera para obtener voltaje de
referencia estable
}

void loop() {
   // muestreo
   for(int i=0; i<MUESTRAS; i++)
    {
      while(!(ADCSRA & 0x10));        // esperar que ADC termine la
conversión actual
      ADCSRA = 0b11110101 ;           // limpiar registro ADCSRA para
que ADC pueda realizar la siguiente conversión (0xf5)
      int valor = ADC - 512 ;         // leer del ADC
      vReal[i] = valor/8;
      vImag[i] = 0;
    }


    // configuración de FFT
    FFT.Windowing(vReal, MUESTRAS, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, MUESTRAS, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, MUESTRAS);


    // reorganizar resultado FFT para adaptarse al número de columnas
    int step = (MUESTRAS/2) / COLUMNAS;
    int c = 0;
    for(int i=0; i < (MUESTRAS/2); i++)
    {
      val[c] = 0;
      for (int k = 0 ; k < step ; k++)
          val[c] = val[c] + vReal[i+k];
      val[c] = val[c]/step;
      c++;
    }


    // enviar el valor medido a cada columna del display
    for(int i=0; i < COLUMNAS; i++)
    {
      val[i] = constrain(val[i],0,80);            // restringir valor
entre 0 y 80
      val[i] = map(val[i], 0, 80, 0, FILAS);      // mapear el valor
entre 0 y número de filas

      valorDisplay = MODO[val[i]];
      columnaDisplay = 31-i;
      mx.setColumn(columnaDisplay, valorDisplay);              //
poner todos los LEDs de la columna en el nuevo estado
    }

    cambioModo ();          // comprueba si el botón se ha presionado
para cambiar de modo
}

void cambioModo() {
  int leido = digitalRead(BOTON_PIN);
```

```
  if (leido == HIGH && estadoPrevio == LOW) // al presionar el botón
cambiar de modo
  {
   switch (modo) {
    case 1:    // del modo 1 al 2
      modo = 2;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_2[i];
      }
      break;
    case 2:    // del modo 2 al 3
      modo = 3;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_3[i];
      }
      break;
    case 3:    // del modo 3 al 4
      modo = 4;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_4[i];
      }
      break;
    case 4:    // del modo 4 al 1
      modo = 1;
      for (int i=0 ; i<=8 ; i++ ) {
        MODO[i] = MODO_1[i];
      }
      break;
    }
  }
  estadoPrevio = leido;
}
```