Politimer

Sumario

Historia

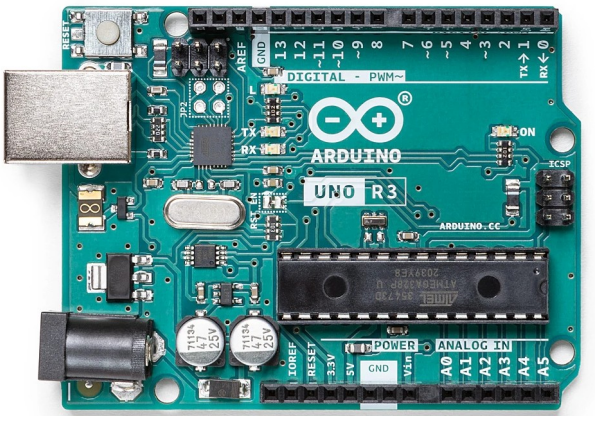
Este proyecto surge a propuesta de la jefatura de estudios del centro para cubrir una necesidad, que es la de tener un dispositivo con un doble display lo suficientemente grande como para ofrecer una buena visibilidad y que permita establecer unos tiempos determinados en los torneos de debate promovidos por la Junta de Andalucía en los centros centros docentes de Andalucía que imparten Enseñanza Secundaria Obligatoria, Ciclos Formativos de Grado Medio y Bachillerato. En el curso 2021-2022 va por la III edición.

El desarrollo de este proyecto quedó paralizado por la llegada de la pandemia de Covid-19 y el confinamiento a partir de marzo de 2020, pero hasta ese momento se habían hecho bastantes avances.

A la fecha en que se escriben estas líneas, seguramente se habría planteado el proyecto de otra manera, realizando una PCB, asegurando la conectorización de otra manera, e incluso usando otro microcontrolador. De hecho, cuando se retome, seguramente se hará teniendo todo esto en cuenta.

Objetivos

Los objetivos de este proyecto, ampliando la propuesta del temporizador para cubrir más funciones de las previstas inicialmente y hecho en este caso para el Arduino UNO (https://store.arduino.cc/products/arduino-uno-rev3/), son los siguientes:



- El dispositivo debe tener buena visibilidad mostrando a cada lado del mismo la información del tiempo restante (en el caso del temporizador).
- Funcionará en principio con una pila de 9V, aunque se puede conectar un alimentador externo para no depender de la pila. Esto hay que testearlo para ver qué es más interesante: pila, alimentador, batería recargable… En principio nos quedamos con la pila.
- Debe tener varios modos de funcionamiento:
  - Reloj.
  - Alarma.
  - Cuenta delante.
  - Cuenta atrás.
  - Marcador.

Requisitos funcionales

Los requisitos funcionales que nos hemos marcado son:

- Debe tener un chasis a medida hecho con impresión 3D que cumpla con las medidas mínimas para tener buena visibilidad, meter todos los componentes, se pueda acceder a la conectorización y se pueda cambiar la pila.
- En cuanto a la electrónica y programación, se podrá cambiar entre los 5 modos de funcionamiento al pulsar el botón de modo:

  1. Reloj – 2. Alarma – 3. Cuenta delante – 4. Cuenta atrás – 5. Marcador
- El funcionamiento de cada uno de los botones es el siguiente:
  - **APAGAR**: apaga/enciende el display (modo ahorro de batería). Hace un sonido cada vez que se cambia. Si los leds no están encendidos no hace caso a ningún otro botón.
  - **RESET**: pone a estado inicial según el modo en el que se encuentre:
    - En el **modo reloj**:
      - En visualización no hace nada.
      - En edición pone todo a cero.
    - En el **modo alarma**:
      - En visualización no hace nada.
      - En edición pone todo con valores de fábrica (12:00).
    - En el **modo cuenta delante**:

- En visualización pone todo con valores de fábrica (00:00).
- En edición pone todo con valores de fábrica (00:00).

▪ En el **modo cuenta atrás**:

- En visualización pone todo con valores de fábrica (10:00).
- En edición pone todo con valores de fábrica (10:00).

▪ En el **modo marcador**:

- Sólo hay un modo (edición) y el RESET lo pone todo con valores de fábrica (00 00).

○ 8 botones de subir/bajar dígitos irán según su propio nombre indica subiendo o bajando los valores del dígito correspondiente a su posición en el chásis.

○ **MODO**:

▪ Con pulsación corta (< 2 segundos) cambia de modo.

▪ Con pulsación larga (> 2 segundos) cambia de visualización del modo en que se encuentre a edición del mismo y viceversa.

○ **PLAY/PAUSA**: arranca o para el reloj o las cuentas delante o atrás. En el modo alarma la activa/desactiva.

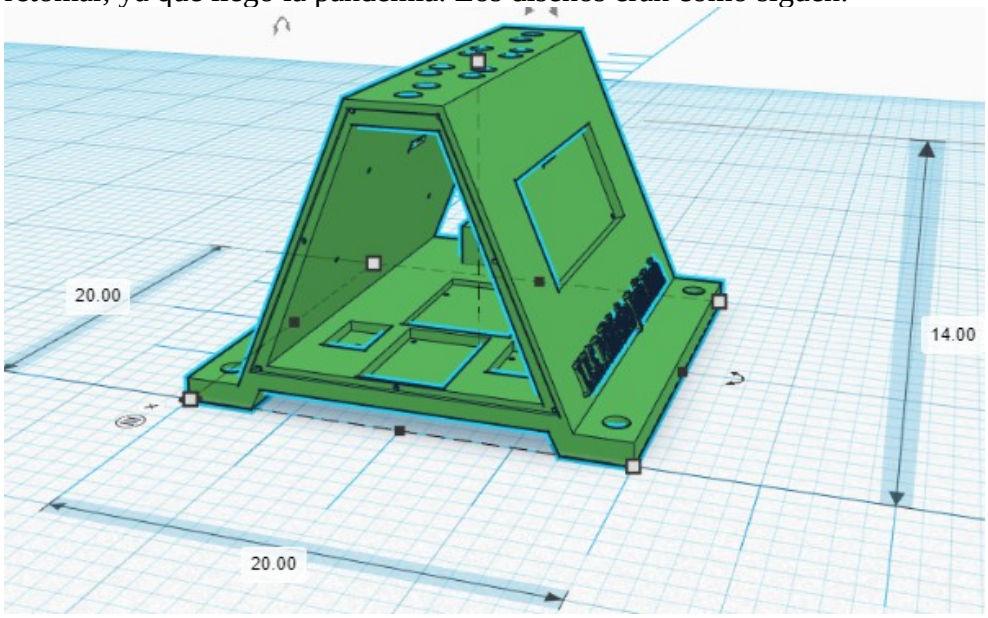
▪ Cuando está desactiva, los dos puntos centrales tienen un parpadeo normal.

▪ Cuando está activa, los dos puntos centrales tienen un parpadeo triple rápido bien visible.

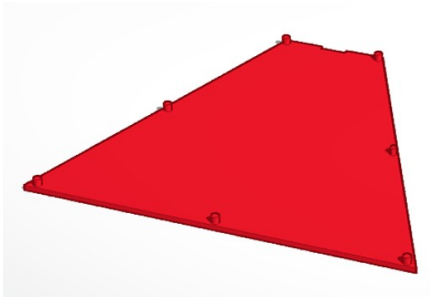
- En general, en todos los modos (salvo el marcador), el parpadeo normal de los dos puntos centrales indica que está en visualización de ese modo. Si los dos puntos están fijos indica que está en modo edición. En el marcador no hay puntos, ya que siempre está listo para ser editado.

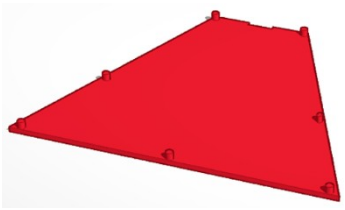
Diseño de chasis

Realizado mediante Tinkercad, la primera versión dio fallos en la impresión y no se pudo volver a retomar, ya que llegó la pandemia. Los diseños eran como siguen:



El chasis principal va cerrado con dos tapas, una más grande que la otra, en los lados para ocultar y proteger todo el cableado.



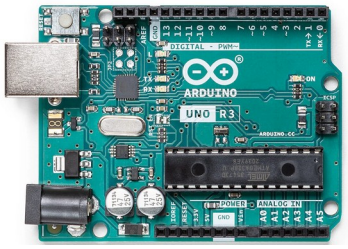


Diseño de PCB (Printed Circuit Board)

Esto está pendiente, en la primera versión, está hecho con una placa de baquelita perforada (para un diseño “rápido”), aunque seguramente habría sido más rápido y provechoso hacerlo directamente en un shield y ahí colocar los conectores.

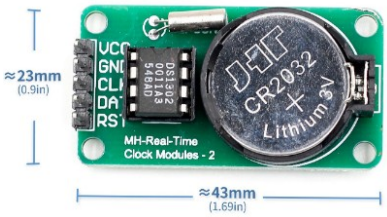
Lista de componentes (para politimer)

- Placa Arduino UNO R3 original o compatible. 4,49€. **Tal vez se cambie en una siguiente versión**.



- Cables dupont (40 pines) para interconexión de Arduino 10 cm 40 uds de cada uno M-M, M-H y H-H. 4,27€. **Se cambiará seguro por conectores adecuados**.

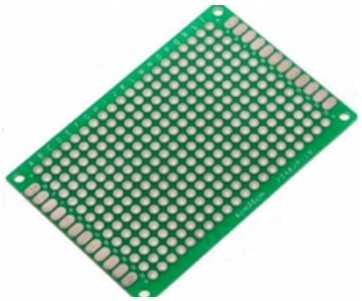
- Módulo de reloj DS1302 (2,3x4,3 cm). 1,31€. S**e cambiará seguramente por el módulo DS3231 mucho más preciso**.



- Display 7 Segmentos I2C gigante (12,0x5,08 cm) con controlador Holtek 16K33 – Rojo y Verde. 24,14€ + 26,50€
  - 5 pines: D → SDA, C → SCL, + → 5V, - → GND, IO → Jumper a 5V

- Botón pulsador normalmente abierto (12 unidades) para uso intensivo. 16,75€ los 12 (12mm tamaño de agujero). Listado botones:
  - Apagar (1).
  - Reset (2).
  - Subir/bajar para cada dígito de izqda a dcha (3-10).
  - Modo (11).
  - Play/pausa (12).

- Placa perforada baquelita (2,11€) (4x6cm) agujeros a 2 mm de bordes. **Se cambiará en la siguiente versión por un escudo con todo integrado**.



- Conector CLIP PILA 9V A DC JACK PARA ARDUINO (0,52€).

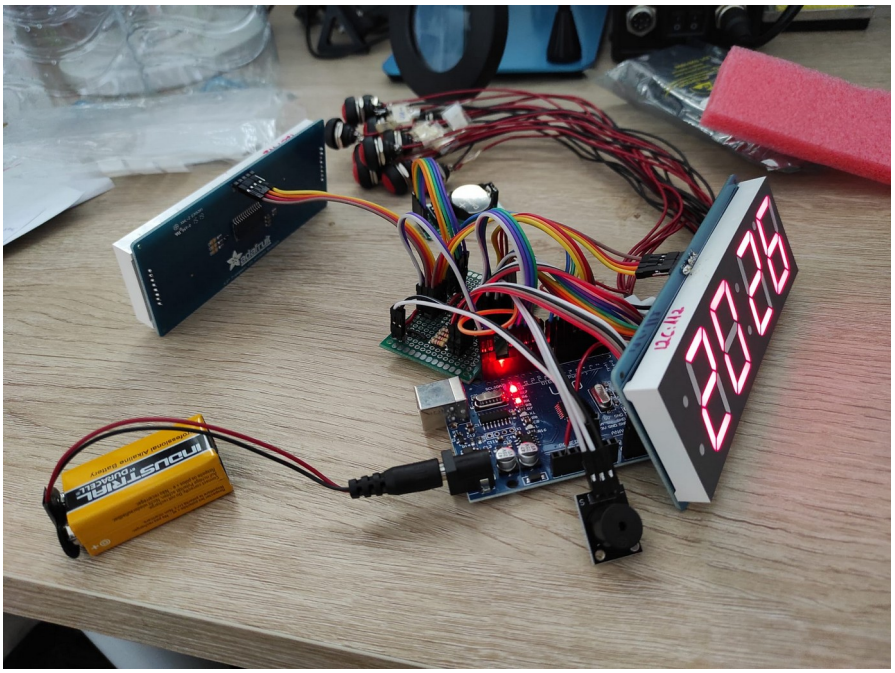
- Módulo zumbador pasivo (0,83€). (1,9x1,5cm) agujeros a 1mm de bordes. **Seguramente se cambiará por el zumbador sólo para que quede integrado en la PCB del escudo**.

- Pila 9V: 5€

- Tornillos y tuercas de métrica 2 y métrica 3 según elemento para afianzar al chásis.

Estado actual

Es totalmente funcional, aunque sin chásis y con mejoras importantes pendientes en el código, que junto a la revisión física, dejarían el actual diseño en “poco reconocible” para mejor.



Líneas futuras

1. Rehacer el diseño electrónico para integrar todos los componentes sueltos en un escudo que se pueda pinchar en el Arduino UNO y sobre él conectar todos los conectores a los botones, a los displays, al módulo de reloj RTC y al módulo zumbador (si es que no se deja fijo en el shield).
2. Revisar y optimizar el código, especialmente la respuesta de los botones, aunque también la extensión, posible externalización de funciones a librerías…
3. Rehacer el diseño del chásis para hacerlo un poco más compacto, teniendo en cuenta que quedaría todo más integrado con el uso de un escudo para el Arduino.
4. Publicar todo el código ya mejorado y esquemas eléctricos optimizados.

Código

Es un poco extenso porque cubre mucha funcionalidad y también porque falta optimizarlo. Está desarrollado usando el IDE Visual Studio Code + Platformio, por eso hay algunas llamadas a librerías hechas de forma peculiar.
Se apoya en una librería creada a propósito para el proyecto.

MAIN.CPP:

```
// Inclusión de librerias
#include <Arduino.h>         // Necesario para que compile en vscode+platformio
#include <Wire.h>            // Enable this line if using Arduino Uno, Mega, etc.
#include <Adafruit_GFX.h>    //En platformio hacen falta las librerias: Adafruit GFX
Library, Adafruit ILI9341, Adafruit LED Backpack
#include <Adafruit_SPITFT.h> //Sólo necesaria en platformio
#include "Adafruit_LEDBackpack.h"
#include <ThreeWire.h>
#include <RtcDS1302.h>
#include <LibreriaTimer.h>

/*-------------------------------------------------*/
//DEFINICIÓN DE CONSTANTES
/*-------------------------------------------------*/
const int RETARDO_REBOTES = 250;   //Retardo para eliminar rebotes de los pulsadores
const int MODO_RELOJ = 0;          //politimer funcionando como un reloj
const int MODO_ALARMA = 1;         //politimer funcionando como una alarma
const int MODO_CUENTA_DELANTE = 2; //politimer funcionando como una cuenta adelante
const int MODO_CUENTA_ATRAS = 3;   //politimer funcionando como una cuenta atrás
const int MODO_MARCADOR = 4;       //politimer funcionando como un marcador deportivo

//---BOTONES---//
const int PIN_SUBIR_DECENAS_MINUTOS = 2;    //Pin asociado a botón de subir decenas de
minutos
```

```
const int PIN_BAJAR_DECENAS_MINUTOS = 3;    //Pin asociado a botón de bajar decenas de minutos
const int PIN_SUBIR_UNIDADES_MINUTOS = 4;   //Pin asociado a botón de subir unidades de minutos
const int PIN_BAJAR_UNIDADES_MINUTOS = 5;   //Pin asociado a botón de bajar unidades de minutos
const int PIN_SUBIR_DECENAS_SEGUNDOS = 10;  //Pin asociado a botón de subir decenas de segundos
const int PIN_BAJAR_DECENAS_SEGUNDOS = 11;  //Pin asociado a botón de bajar decenas de segundos
const int PIN_SUBIR_UNIDADES_SEGUNDOS = 12; //Pin asociado a botón de subir unidades de segundos
const int PIN_BAJAR_UNIDADES_SEGUNDOS = 13; //Pin asociado a botón de bajar unidades de segundos
//Dejamos los analógicos para los botones de modo, ya que por encima del 13 son analógicos
//funcionando como digitales y tienen una respuesta ligeramente diferente a los digitales
const int PIN_APAGAR = 14;     //Realmente es un pin analógico (A0) siendo usado como digital
const int PIN_RESET = 15;      //Realmente es un pin analógico (A1) siendo usado como digital
const int PIN_MODO = 16;       //Realmente es un pin analógico (A2) siendo usado como digital
const int PIN_PAUSA_PLAY = 17; //Realmente es un pin analógico (A3) siendo usado como digital
//---BOTONES---//

const int PIN_CLK_RTC = 6; //Pin asociado a pin CLK del módulo de reloj DS1302
const int PIN_DAT_RTC = 7; //Pin asociado a pin DAT del módulo de reloj DS1302
const int PIN_RST_RTC = 8; //Pin asociado a pin RST del módulo de reloj DS1302

const int PIN_SIG_BUZZER = 9; //Pin asociado a pin del módulo de buzzer pasivo

//Para lectura de pulsaciones largas (cambio de modo activo a modo edición)
const unsigned long TPULSACIONLARGA = 2000; //Tiempo pulsacion larga

//Valores por defecto a meter en cada modo
const byte VALOR_FABRICA_RELOJ_HORAS = 0;
const byte VALOR_FABRICA_RELOJ_MINUTOS = 0;
const byte VALOR_FABRICA_RELOJ_SEGUNDOS = 0;
const byte VALOR_FABRICA_ALARMA_HORAS = 12;
const byte VALOR_FABRICA_ALARMA_MINUTOS = 0;
const byte VALOR_FABRICA_ALARMA_SEGUNDOS = 0;
const byte VALOR_FABRICA_CUENTA_DELANTE_MINUTOS = 0;
const byte VALOR_FABRICA_CUENTA_DELANTE_SEGUNDOS = 0;
const byte VALOR_FABRICA_CUENTA_ATRAS_MINUTOS = 10;
const byte VALOR_FABRICA_CUENTA_ATRAS_SEGUNDOS = 0;
const byte VALOR_FABRICA_MARCADOR_LOCAL = 0;
const byte VALOR_FABRICA_MARCADOR_VISITANTE = 0;

/*-----------------------------------------------*/
//DEFINICIÓN DE VARIABLES
/*-----------------------------------------------*/
Adafruit_7segment display = Adafruit_7segment(); //Instancia del display de 7 segmentos
```

```
unsigned long tiempo = 0;                          //Control del tiempo
byte modo = MODO_RELOJ;                            //Por defecto, modo reloj
boolean modoEdicion = false;                       //Variable para controlar si estamos
en edición o en visualización
boolean modoLedsApagados = false;                  //Variable para controlar el
encendido o apagado de los leds de los displays (modo ahorro de energía)
boolean alarmaActiva = false;                      //Variable para controlar si se
activa o desactiva la alarma
boolean contandoDelante = false;                   //Variable para controlar si la
cuenta adelante está en marcha o no
boolean contandoAtras = false;                     //Variable para controlar si la
cuenta atrás está en marcha o no
boolean arrancaCuentaDelante = false;              //Variable para controlar si la
cuenta adelante inicia por primera vez
boolean arrancaCuentaAtras = false;                //Variable para controlar si la
cuenta atrás inicia por primera vez
int lecturaBotonApagar = 0;                        //inicializo botón
int lecturaBotonReset = 0;                         //inicializo botón
int lecturaBotonSubirDecenasMinutos = 0;           //inicializo botón
int lecturaBotonBajarDecenasMinutos = 0;           //inicializo botón
int lecturaBotonSubirUnidadesMinutos = 0;          //inicializo botón
int lecturaBotonBajarUnidadesMinutos = 0;          //inicializo botón
int lecturaBotonSubirDecenasSegundos = 0;          //inicializo botón
int lecturaBotonBajarDecenasSegundos = 0;          //inicializo botón
int lecturaBotonSubirUnidadesSegundos = 0;         //inicializo botón
int lecturaBotonBajarUnidadesSegundos = 0;         //inicializo botón
int lecturaBotonModo = 0;                          //inicializo botón
int lecturaBotonPausaPlay = 0;                     //inicializo botón

byte horas = -1;    //Definición de variable para controlar las horas del reloj
byte minutos = -1;  //Definición de variable para controlar los minutos del reloj
byte segundos = -1; //Definición de variable para controlar los segundos del reloj

byte horasA = -1;    //Definición de variable para controlar las horas de la alarma
byte minutosA = -1;  //Definición de variable para controlar los minutos de la alarma
byte segundosA = -1; //Definición de variable para controlar los segundos de la alarma

byte minutosCuentaDelante = -1;         //Definición de variable para controlar los
minutos de la cuenta delante
byte segundosCuentaDelante = -1;        //Definición de variable para controlar los
segundos de la cuenta delante
byte minutosCuentaDelanteInicial = -1;  //Definición de variable para controlar el
valor incial de los minutos de la cuenta delante
byte segundosCuentaDelanteInicial = -1; //Definición de variable para controlar el
valor incial de los segundos de la cuenta delante

byte minutosCuentaAtras = -1;                //Definición de variable para controlar
los minutos de la cuenta atrás
byte segundosCuentaAtras = -1;               //Definición de variable para controlar
los segundos de la cuenta atrás
byte minutosCuentaAtrasReseteoEnMarcha = -1;  //Definición de variable para controlar
el valor incial de los minutos de la cuenta atrás
byte segundosCuentaAtrasReseteoEnMarcha = -1; //Definición de variable para controlar
el valor incial de los segundos de la cuenta atrás
long segundosTotalesCuentaAtras;             //Definición de variable para controlar
los segundos totales contando los minutos de la cuenta atrás
```

```
long segundosRestantesCuentaAtras;            //Definición de variable para controlar los segundos restantes contando los minutos de la cuenta atrás una vez que ya ha empezado

byte marcadorLocal = -1;     //Definición de variable para controlar el valor del marcador local
byte marcadorVisitante = -1; //Definición de variable para controlar el valor del marcador visitante

boolean drawDots = true;                                  //Variable para controlar el encendido de los dos puntos centrales del display
ThreeWire myWire(PIN_DAT_RTC, PIN_CLK_RTC, PIN_RST_RTC); // DAT/IO, CLK/SCLK, RST/CE. Declaración de variable necesaria para el control del reloj RTC
RtcDS1302<ThreeWire> rtc(myWire);                         //Declaración del reloj RTC

//Para lectura de pulsaciones largas (cambio de modo normal a modo edición)
unsigned long tPulsacion;
unsigned long tInicio;
unsigned long tFin;
int estadoActual; //Estado actual pulsador
int estadoUltimo; //Estado ultimo pulsador

unsigned long contadorSegundosDelanteInicial;
unsigned long contadorSegundosDelante;
unsigned long contadorSegundosAtrasInicial;
unsigned long contadorSegundosAtras;

/*-------------------------------------------------*/
//FUNCIONES A IMPLEMENTAR EN MAIN
/*-------------------------------------------------*/

/*
  Función para realizar la lectura de todos los botones conectados al politimer e introducir los valores leidos en las variables dispuestas a tal fin
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void LeerBotones()
{
  //Se leen todos los botones
  lecturaBotonApagar = digitalRead(PIN_APAGAR);
  lecturaBotonReset = digitalRead(PIN_RESET);

  lecturaBotonSubirDecenasMinutos = digitalRead(PIN_SUBIR_DECENAS_MINUTOS);
  lecturaBotonBajarDecenasMinutos = digitalRead(PIN_BAJAR_DECENAS_MINUTOS);
  lecturaBotonSubirUnidadesMinutos = digitalRead(PIN_SUBIR_UNIDADES_MINUTOS);
  lecturaBotonBajarUnidadesMinutos = digitalRead(PIN_BAJAR_UNIDADES_MINUTOS);
  lecturaBotonSubirDecenasSegundos = digitalRead(PIN_SUBIR_DECENAS_SEGUNDOS);
  lecturaBotonBajarDecenasSegundos = digitalRead(PIN_BAJAR_DECENAS_SEGUNDOS);
  lecturaBotonSubirUnidadesSegundos = digitalRead(PIN_SUBIR_UNIDADES_SEGUNDOS);
  lecturaBotonBajarUnidadesSegundos = digitalRead(PIN_BAJAR_UNIDADES_SEGUNDOS);

  lecturaBotonModo = digitalRead(PIN_MODO);
  lecturaBotonPausaPlay = digitalRead(PIN_PAUSA_PLAY);

  estadoActual = lecturaBotonModo;
```

```
}

/*
  Función para realizar un flasheo rápido de los puntos centrales del display a modo de notificación de un evento
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void FlashDots()
{
  for (byte indice = 0; indice < 4; indice++)
  {
    display.drawColon(drawDots);
    drawDots = !drawDots;
    display.writeDisplay();
    delay(30);
  }
}

/*
  Función para controlar por completo el modo reloj.
  Se realiza la visualización en los displays y se controla la edición del mismo.
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void Reloj()
{
  RtcDateTime now = rtc.GetDateTime();

  printDateTime(now);
  Serial.println();
  if (modoEdicion)
  {
    drawDots = true;
  }
  else
  {
    drawDots = !drawDots;
  }
  display.writeDigitNum(0, (now.Hour() / 10), drawDots);
  display.writeDigitNum(1, (now.Hour() % 10), drawDots);
  display.drawColon(drawDots);
  display.writeDigitNum(3, (now.Minute() / 10), drawDots);
  display.writeDigitNum(4, (now.Minute() % 10), drawDots);
  display.writeDisplay();

  if (!now.IsValid())
  {
    // Common Causes:
    //    1) the battery on the device is low or even missing and the power line was disconnected
    Serial.println("RTC lost confidence in the DateTime! LA PILA NO VA...");
  }

  if (modoEdicion)
  {
```

```
horas = now.Hour();
minutos = now.Minute();
segundos = now.Second();
if (lecturaBotonReset == HIGH)
{
  horas = VALOR_FABRICA_RELOJ_HORAS;
  minutos = VALOR_FABRICA_RELOJ_MINUTOS;
  segundos = VALOR_FABRICA_RELOJ_SEGUNDOS;
}
if (lecturaBotonSubirDecenasMinutos == HIGH)
{
  horas = horas + 10;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarDecenasMinutos == HIGH)
{
  horas = horas - 10;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirUnidadesMinutos == HIGH)
{
  horas = horas + 1;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarUnidadesMinutos == HIGH)
{
  horas = horas - 1;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirDecenasSegundos == HIGH)
{
  minutos = minutos + 10;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarDecenasSegundos == HIGH)
{
  minutos = minutos - 10;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirUnidadesSegundos == HIGH)
{
  minutos = minutos + 1;
  delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarUnidadesSegundos == HIGH)
{
  minutos = minutos - 1;
  delay(RETARDO_REBOTES);
}

horas = corrigeHoras(horas);
minutos = corrigeMinSeg(minutos);
```

```
    RtcDateTime dt = RtcDateTime(now.Year(), now.Month(), now.Day(), horas, minutos,
segundos);
    rtc.SetDateTime(dt);
  }

  delay(500); // cada medio segundo actualiza
}

/*
  Función para establecer en la alarma un valor suministrado a través de tres
variables tipo byte
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void EstablecerAlarma(byte horas, byte minutos, byte segundos)
{
  horasA = horas;
  minutosA = minutos;
  segundosA = segundos;
}

/*
  Función para controlar por completo el modo alarma.
  Se realiza la visualización en los displays y se controla la edición del mismo con
llamada a la función EstablecerAlarma.
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void Alarma()
{
  if (modoEdicion)
  {
    drawDots = true;
  }
  else
  {
    drawDots = !drawDots;
  }

  //Activar alarma
  if (lecturaBotonPausaPlay == HIGH)
  {
    alarmaActiva = !alarmaActiva;
    FlashDots();
    if (alarmaActiva)
    {
      modoEdicion = false;
    }
  }

  if (alarmaActiva) //Si la alarma está activa los puntos centrales flashean
visiblemente al ver la hora de la alarma
  {
    FlashDots();
  }
```

```
  horasA = corrigeHoras(horasA);
  minutosA = corrigeMinSeg(minutosA);
  segundosA = corrigeMinSeg(segundosA);
  display.writeDigitNum(0, (horasA / 10), drawDots);
  display.writeDigitNum(1, (horasA % 10), drawDots);
  display.drawColon(drawDots);
  display.writeDigitNum(3, (minutosA / 10), drawDots);
  display.writeDigitNum(4, (minutosA % 10), drawDots);
  display.writeDisplay();

  if (modoEdicion)
  {
    if (lecturaBotonReset == HIGH)
    {
      EstablecerAlarma(VALOR_FABRICA_ALARMA_HORAS, VALOR_FABRICA_ALARMA_MINUTOS,
VALOR_FABRICA_ALARMA_SEGUNDOS);
  }
  if (lecturaBotonSubirDecenasMinutos == HIGH)
  {
   horasA = horasA + 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarDecenasMinutos == HIGH)
  {
   horasA = horasA - 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirUnidadesMinutos == HIGH)
  {
   horasA = horasA + 1;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarUnidadesMinutos == HIGH)
  {
   horasA = horasA - 1;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirDecenasSegundos == HIGH)
  {
   minutosA = minutosA + 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarDecenasSegundos == HIGH)
  {
   minutosA = minutosA - 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirUnidadesSegundos == HIGH)
  {
   minutosA = minutosA + 1;
   delay(RETARDO_REBOTES);
```

```
  }
  if (lecturaBotonBajarUnidadesSegundos == HIGH)
  {
   minutosA = minutosA - 1;
   delay(RETARDO_REBOTES);
  }
 }
 delay(500); // actualiza cada medio segundo
}

/*
 Función para establecer en la cuenta delante un valor suministrado a través de dos variables tipo byte
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void EstablecerCuentaDelante(byte minutos, byte segundos)
{
 minutosCuentaDelante = minutos;
 segundosCuentaDelante = segundos;
}

/*
 Función para controlar el modo Cuenta Delante.
 Se realiza la visualización en los displays y se controla la edición del mismo con llamada a la función EstablecerCuentaDelante.
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void CuentaDelante()
{

 if (modoEdicion)
 {
  if (lecturaBotonReset == HIGH)
  {
   EstablecerCuentaDelante(VALOR_FABRICA_CUENTA_DELANTE_MINUTOS, VALOR_FABRICA_CUENTA_DELANTE_SEGUNDOS);
  }
  if (lecturaBotonSubirDecenasMinutos == HIGH)
  {
   minutosCuentaDelante = minutosCuentaDelante + 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarDecenasMinutos == HIGH)
  {
   minutosCuentaDelante = minutosCuentaDelante - 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirUnidadesMinutos == HIGH)
  {
   minutosCuentaDelante = minutosCuentaDelante + 1;
   delay(RETARDO_REBOTES);
```

```
 }
 if (lecturaBotonBajarUnidadesMinutos == HIGH)
 {
  minutosCuentaDelante = minutosCuentaDelante - 1;
  delay(RETARDO_REBOTES);
 }
 if (lecturaBotonSubirDecenasSegundos == HIGH)
 {
  segundosCuentaDelante = segundosCuentaDelante + 10;
  delay(RETARDO_REBOTES);
 }
 if (lecturaBotonBajarDecenasSegundos == HIGH)
 {
  segundosCuentaDelante = segundosCuentaDelante - 10;
  delay(RETARDO_REBOTES);
 }
 if (lecturaBotonSubirUnidadesSegundos == HIGH)
 {
  segundosCuentaDelante = segundosCuentaDelante + 1;
  delay(RETARDO_REBOTES);
 }
 if (lecturaBotonBajarUnidadesSegundos == HIGH)
 {
  segundosCuentaDelante = segundosCuentaDelante - 1;
  delay(RETARDO_REBOTES);
 }

 drawDots = true;
}
else //modo normal, no edición
{
 drawDots = !drawDots;
 if (lecturaBotonReset == HIGH)
 {
  EstablecerCuentaDelante(VALOR_FABRICA_CUENTA_DELANTE_MINUTOS,
VALOR_FABRICA_CUENTA_DELANTE_SEGUNDOS);
  contandoDelante = false;
  arrancaCuentaDelante = false;
 }
}

if (lecturaBotonPausaPlay == HIGH) //Activar/parar cuenta
{
 contandoDelante = !contandoDelante;
 delay(RETARDO_REBOTES);
 if (!contandoDelante)
 {
  arrancaCuentaDelante = false;
 }
}
```

```
 if (contandoDelante)
 {
  if (!arrancaCuentaDelante)
  {
   arrancaCuentaDelante = true;
   contadorSegundosDelanteInicial = millis();
   contadorSegundosDelante = 0;
   minutosCuentaDelanteInicial = minutosCuentaDelante;
   segundosCuentaDelanteInicial = segundosCuentaDelante;
  }
  else
  {
   contadorSegundosDelante = diferenciaCorrigiendoOverflow(millis(),
contadorSegundosDelanteInicial) / 1000;
  }
  //Correcciones para la cuenta de minutos y segundos
  byte minutosContados = contadorSegundosDelante / 60;
  int segundosContados = contadorSegundosDelante % 60;

  minutosCuentaDelante = minutosCuentaDelanteInicial + minutosContados;
  segundosCuentaDelante = segundosCuentaDelanteInicial + segundosContados;
  minutosCuentaDelante = minutosCuentaDelante + segundosCuentaDelante / 60;
  if (segundosCuentaDelante > 59)
  {
   segundosCuentaDelante = segundosCuentaDelante % 60;
  }
 }

 minutosCuentaDelante = corrigeMinSeg(minutosCuentaDelante);
 segundosCuentaDelante = corrigeMinSeg(segundosCuentaDelante);
 display.writeDigitNum(0, (minutosCuentaDelante / 10), drawDots);
 display.writeDigitNum(1, (minutosCuentaDelante % 10), drawDots);
 display.drawColon(drawDots);
 display.writeDigitNum(3, (segundosCuentaDelante / 10), drawDots);
 display.writeDigitNum(4, (segundosCuentaDelante % 10), drawDots);
 display.writeDisplay();
 delay(500);
}

/*
 Función para establecer en la cuenta atrás un valor suministrado a través de dos
variables tipo byte
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void EstablecerCuentaAtras(byte minutos, byte segundos)
{
 minutosCuentaAtras = minutos;
 segundosCuentaAtras = segundos;
 minutosCuentaAtrasReseteoEnMarcha = minutos;
 segundosCuentaAtrasReseteoEnMarcha = segundos;
```

```
}

/*
 Función para controlar el modo Cuenta Atrás.
 Se realiza la visualización en los displays y se controla la edición del mismo con
llamada a la función EstablecerCuentaAtrás.
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void CuentaAtras()
{

 if (modoEdicion)
 {
  if (lecturaBotonReset == HIGH)
  {
   EstablecerCuentaAtras(VALOR_FABRICA_CUENTA_ATRAS_MINUTOS,
VALOR_FABRICA_CUENTA_ATRAS_SEGUNDOS);
  }
  if (lecturaBotonSubirDecenasMinutos == HIGH)
  {
   minutosCuentaAtras = minutosCuentaAtras + 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarDecenasMinutos == HIGH)
  {
   minutosCuentaAtras = minutosCuentaAtras - 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirUnidadesMinutos == HIGH)
  {
   minutosCuentaAtras = minutosCuentaAtras + 1;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarUnidadesMinutos == HIGH)
  {
   minutosCuentaAtras = minutosCuentaAtras - 1;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirDecenasSegundos == HIGH)
  {
   segundosCuentaAtras = segundosCuentaAtras + 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarDecenasSegundos == HIGH)
  {
   segundosCuentaAtras = segundosCuentaAtras - 10;
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonSubirUnidadesSegundos == HIGH)
  {
   segundosCuentaAtras = segundosCuentaAtras + 1;
```

```
   delay(RETARDO_REBOTES);
  }
  if (lecturaBotonBajarUnidadesSegundos == HIGH)
  {
   segundosCuentaAtras = segundosCuentaAtras - 1;
   delay(RETARDO_REBOTES);
  }

  drawDots = true;
 }
 else //modo normal, no edición
 {
  drawDots = !drawDots;
  if (lecturaBotonReset == HIGH)
  {
   EstablecerCuentaAtras(minutosCuentaAtrasReseteoEnMarcha,
segundosCuentaAtrasReseteoEnMarcha);
   contandoAtras = false;
   arrancaCuentaAtras = false;
  }
 }

 if (lecturaBotonPausaPlay == HIGH) //Activar/parar cuenta
 {
  contandoAtras = !contandoAtras;
  delay(RETARDO_REBOTES);
  if (!contandoAtras)
  {
   arrancaCuentaAtras = false;
  }
 }

 if (contandoAtras)
 {
  if (!arrancaCuentaAtras)
  {
   arrancaCuentaAtras = true;
   contadorSegundosAtrasInicial = millis();
   segundosTotalesCuentaAtras = minutosCuentaAtras * 60 + segundosCuentaAtras;
   contadorSegundosAtras = 0;
  }
  else
  {
   contadorSegundosAtras = diferenciaCorrigiendoOverflow(millis(),
contadorSegundosAtrasInicial) / 1000;
  }
  segundosRestantesCuentaAtras = segundosTotalesCuentaAtras - contadorSegundosAtras;

  if (segundosRestantesCuentaAtras <= 0)
  {
   arrancaCuentaAtras = false;
```

```
   contandoAtras = false;
   segundosRestantesCuentaAtras = 0;
   display.writeDigitNum(0, 0, drawDots); //se duplica aquí este código para que aparezca el 00:00 antes de empezar a sonar la sirena
   display.writeDigitNum(1, 0, drawDots);
   display.drawColon(drawDots);
   display.writeDigitNum(3, 0, drawDots);
   display.writeDigitNum(4, 0, drawDots);
   display.writeDisplay();
   SonarSirena(PIN_SIG_BUZZER);
  }

  minutosCuentaAtras = segundosRestantesCuentaAtras / 60;
  segundosCuentaAtras = segundosRestantesCuentaAtras % 60;
 }

 minutosCuentaAtras = corrigeMinSeg(minutosCuentaAtras);
 segundosCuentaAtras = corrigeMinSeg(segundosCuentaAtras);
 display.writeDigitNum(0, (minutosCuentaAtras / 10), drawDots);
 display.writeDigitNum(1, (minutosCuentaAtras % 10), drawDots);
 display.drawColon(drawDots);
 display.writeDigitNum(3, (segundosCuentaAtras / 10), drawDots);
 display.writeDigitNum(4, (segundosCuentaAtras % 10), drawDots);
 display.writeDisplay();
 delay(500); //actualiza cada medio segundo
}

/*
 Función para establecer en el marcador un valor suministrado a través de dos variables tipo byte
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void EstablecerMarcador(byte local, byte visitante)
{
 marcadorLocal = local;
 marcadorVisitante = visitante;
}

/*
 Función para controlar el modo Marcador.
 Se realiza la visualización en los displays y se controla la edición del mismo con llamada a la función EstablecerMarcador.
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void Marcador()
{
 if (lecturaBotonReset == HIGH)
 {
  EstablecerMarcador(VALOR_FABRICA_MARCADOR_LOCAL, VALOR_FABRICA_MARCADOR_VISITANTE);
 }
 if (lecturaBotonSubirDecenasMinutos == HIGH)
 {
```

```
 marcadorLocal = marcadorLocal + 10;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarDecenasMinutos == HIGH)
{
 marcadorLocal = marcadorLocal - 10;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirUnidadesMinutos == HIGH)
{
 marcadorLocal = marcadorLocal + 1;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarUnidadesMinutos == HIGH)
{
 marcadorLocal = marcadorLocal - 1;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirDecenasSegundos == HIGH)
{
 marcadorVisitante = marcadorVisitante + 10;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarDecenasSegundos == HIGH)
{
 marcadorVisitante = marcadorVisitante - 10;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonSubirUnidadesSegundos == HIGH)
{
 marcadorVisitante = marcadorVisitante + 1;
 delay(RETARDO_REBOTES);
}
if (lecturaBotonBajarUnidadesSegundos == HIGH)
{
 marcadorVisitante = marcadorVisitante - 1;
 delay(RETARDO_REBOTES);
}

 drawDots = false;
 marcadorLocal = corrigeMarcador(marcadorLocal);
 marcadorVisitante = corrigeMarcador(marcadorVisitante);
 display.writeDigitNum(0, (marcadorLocal / 10), drawDots);
 display.writeDigitNum(1, (marcadorLocal % 10), drawDots);
 display.drawColon(drawDots);
 display.writeDigitNum(3, (marcadorVisitante / 10), drawDots);
 display.writeDigitNum(4, (marcadorVisitante % 10), drawDots);
 display.writeDisplay();
}
```

```
/*
 Función de ahorro de energía. Apaga todos los leds
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void ApagarDisplay()
{
 display.writeDigitRaw(0, 0); //digito más a la izquierda
 display.writeDigitRaw(1, 0);
 display.writeDigitRaw(2, 0); //dos puntos
 display.writeDigitRaw(3, 0);
 display.writeDigitRaw(4, 0); //dígito más a la derecha
 display.writeDisplay();
}

/*
 Función para leer de forma especial pulsación corta/larga del botón de modo
 Si es larga se cambia entre modo edición/normal.
 Si es corta se cambia de modo reloj / alarma / cuenta adelante / cuenta atrás / marcador
 Created by José Luis Guerrero Marín, May 9, 2020.
*/
void LecturaModos()
{
 if (estadoActual != estadoUltimo)
 {
  if (estadoActual == HIGH)
  {
   tInicio = millis();
  }
  else
  {
   tFin = millis();
   tPulsacion = diferenciaCorrigiendoOverflow(tFin, tInicio);
  }
  estadoUltimo = estadoActual;
  if ((tPulsacion < TPULSACIONLARGA) && (estadoActual == LOW))
  {
   Serial.println("Cambio de modo");
   modo++;
   modoEdicion = false;
   if (modo >= 5)
   {
    modo = 0;
   }
   SonarPitido(PIN_SIG_BUZZER, 1000);
   delay(RETARDO_REBOTES);
  }
 }
 else if ((estadoActual == estadoUltimo) && (estadoActual == HIGH) && (modo !=
MODO_MARCADOR))
 {
  if (tInicio != 0)
```

```
  {
   tFin = millis();
   tPulsacion = diferenciaCorrigiendoOverflow(tFin, tInicio);
  }
  if (tPulsacion > TPULSACIONLARGA)
  {
   Serial.println("Cambio a EDICIÓN");
   modoEdicion = !modoEdicion;
   SonarPitido(PIN_SIG_BUZZER, 2000);
   SonarPitido(PIN_SIG_BUZZER, 2000);
   delay(RETARDO_REBOTES);
   tPulsacion = 0;
   tInicio = 0;
  }
 }
}

/*
 Función de inicio de ARDUINO.
 Aquí se establecen todos los valores iniciales o "de fábrica".
 Aquí es donde se llama cada vez que se hace un HARD RESET pulsando el reset de la
placa de Arduino.

 put your setup code here, to run once:
*/
void setup()
{
 Serial.begin(9600); //Establecemos velocidad de comunicación
#ifndef __AVR_ATtiny85__
 Serial.println("Arranca setup");
#endif
 display.begin(0x70);

 //ASIGNACIÓN DE BOTONES CON PINES DE LA PLACA COMO ENTRADAS
 pinMode(PIN_APAGAR, INPUT);        //Establecemos el pin del botón de apagado como
entrada
 pinMode(PIN_RESET, INPUT);        //Establecemos el pin del botón de reset como
entrada
 pinMode(PIN_SUBIR_DECENAS_MINUTOS, INPUT); //Establecemos el pin del botón de subir
decenas de minutos como entrada
 pinMode(PIN_BAJAR_DECENAS_MINUTOS, INPUT); //Establecemos el pin del botón de bajar
decenas de minutos como entrada
 pinMode(PIN_SUBIR_UNIDADES_MINUTOS, INPUT); //Establecemos el pin del botón de subir
decenas de minutos como entrada
 pinMode(PIN_BAJAR_UNIDADES_MINUTOS, INPUT); //Establecemos el pin del botón de bajar
decenas de minutos como entrada
 pinMode(PIN_SUBIR_DECENAS_SEGUNDOS, INPUT); //Establecemos el pin del botón de subir
decenas de segundos como entrada
 pinMode(PIN_BAJAR_DECENAS_SEGUNDOS, INPUT); //Establecemos el pin del botón de bajar
decenas de segundos como entrada
 pinMode(PIN_SUBIR_UNIDADES_SEGUNDOS, INPUT); //Establecemos el pin del botón de subir
unidades de segundos como entrada
```

```
 pinMode(PIN_BAJAR_UNIDADES_SEGUNDOS, INPUT); //Establecemos el pin del botón de bajar unidades de segundos como entrada
 pinMode(PIN_MODO, INPUT);         //Establecemos el pin del botón de modo como entrada
 pinMode(PIN_PAUSA_PLAY, INPUT);      //Establecemos el pin del botón de pausa/play como entrada

 tiempo = millis();
 SetupReloj(rtc);
 EstablecerAlarma(VALOR_FABRICA_ALARMA_HORAS, VALOR_FABRICA_ALARMA_MINUTOS, VALOR_FABRICA_ALARMA_SEGUNDOS);
 EstablecerCuentaDelante(VALOR_FABRICA_CUENTA_DELANTE_MINUTOS, VALOR_FABRICA_CUENTA_DELANTE_SEGUNDOS);
 EstablecerCuentaAtras(VALOR_FABRICA_CUENTA_ATRAS_MINUTOS, VALOR_FABRICA_CUENTA_ATRAS_SEGUNDOS);
 EstablecerMarcador(VALOR_FABRICA_MARCADOR_LOCAL, VALOR_FABRICA_MARCADOR_VISITANTE);

 modoEdicion = false;
 tPulsacion = 0;
 tInicio = 0;
 tFin = 0;
 estadoActual = 0;
 estadoUltimo = 0;

 alarmaActiva = false;

 contandoDelante = false;
 contandoAtras = false;
 arrancaCuentaDelante = false;
 arrancaCuentaAtras = false;
 contadorSegundosDelante = 0;
 contadorSegundosDelanteInicial = 0;
 contadorSegundosAtras = 0;
 contadorSegundosAtrasInicial = 0;
 segundosTotalesCuentaAtras = 0;
 segundosRestantesCuentaAtras = 0;
}

/*
 Función de lazo de ARDUINO.
 Esto es lo que se ejecuta de forma contínua en Arduina tras la lectura de librerías, definición de constantes, variables y la llamada al setup.

 put your main code here, to run repeatedly:
*/
void loop()
{
 RtcDateTime ahora = rtc.GetDateTime();
 if ((alarmaActiva) && (ahora.Hour() == horasA) && (ahora.Minute() == minutosA))
 {
  SonarMarchaImperial(PIN_SIG_BUZZER); //La alarma hace sonar la marcha imperial. Este sonido no se puede parar hasta que acaba
  alarmaActiva = false;
```

```
 }

 if (millis() > tiempo + RETARDO_REBOTES) //Lectura contínua de los botones por si se
están presionando y metiendo un retardo para evitar rebotes vía software
 {
  LeerBotones();
  tiempo = millis(); //Se toma referencia del tiempo para el control de los rebotes
 }

 if (lecturaBotonApagar == HIGH) //Este es un botón prioritario. Si los leds están
apagados, los demás botones no hacen nada, lógicamente.
 {
  Serial.println("Leds apagados/encendidos");
  modoLedsApagados = !modoLedsApagados;
  SonarPitidoApagado(PIN_SIG_BUZZER); //Suena un pitido especial grave para diferenciar
el botón APAGADO/ENCENDIDO de los leds de los demás
  delay(RETARDO_REBOTES);
 }

 if (modoLedsApagados) //Si los leds están apagados, los demás botones no hacen nada,
lógicamente.
 {
  ApagarDisplay();
 }
 else        //Si los leds están encendidos ya sí puede funcionar todo lo demás.
 {

  LecturaModos(); //Lectura especial del botón de modo para diferenciar pulsación
larga/corta
  if (lecturaBotonPausaPlay == HIGH)
  {
   Serial.println("Pausa/Play");
   SonarPitidoPlayPause(PIN_SIG_BUZZER); //Suena un pitido especial agudo para
diferenciar el botón PLAY/PAUSE de los demás
  }
  else if (lecturaBotonReset == HIGH)
  {
   Serial.println("Reset");
   SonarPitidoReset(PIN_SIG_BUZZER); //Suena un pitido especial para diferenciar el
botón RESET de los demás
  }

  switch (modo) //Estructura para derivar el flujo según el modo de funcionamiento
  {
  case MODO_RELOJ:
   Reloj();
   break;
  case MODO_ALARMA:
   Alarma();
   break;
  case MODO_CUENTA_DELANTE:
   CuentaDelante();
```

```
   break;
  case MODO_CUENTA_ATRAS:
   CuentaAtras();
   break;
  case MODO_MARCADOR:
   Marcador();
   break;
  default: // Por defecto se va al modo reloj
   Reloj();
   break;
  }
 }
}
```

LIBRERIATIMER.H:

```
/*
  LibreriaTimer.h - Library for support of Politecnico Timer
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
#ifndef LibreriaTimer_h
#define LibreriaTimer_h

#include "Arduino.h"

unsigned long diferenciaCorrigiendoOverflow(unsigned long fin,unsigned long inicio);
byte corrigeHoras(byte val);
byte corrigeMinSeg(byte val);
byte corrigeMarcador(byte val);
void printDateTime(const RtcDateTime &dt);
String getStringDateTime(const RtcDateTime &dt);
RtcDS1302<ThreeWire> SetupReloj(RtcDS1302<ThreeWire> rtc);
void SonarTono(int pin_SIG_buzzer);
void SonarPitido(int pin_SIG_buzzer, int frec);
void SonarPitido(int pin_SIG_buzzer, int frec, int ms);
void SonarPitidoApagado(int pin_SIG_buzzer);
void SonarPitidoPlayPause(int pin_SIG_buzzer);
void SonarPitidoReset(int pin_SIG_buzzer);
void SonarSirena(int pin_SIG_buzzer);
void SonarMarchaImperial(int pin_SIG_buzzer);


#endif
```

LIBRERIATIMER.CPP:

```
#include <Arduino.h> //Sólo necesaria en platformio
#include <ThreeWire.h>
```

```
#include <RtcDS1302.h>
#include <LibreriaTimer.h>


/*-----------------------------------------------*/
//------------DEFINICIÓN DE CONSTANTES-------------//
/*-----------------------------------------------*/
const int TONOS[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494};
const int numTonos = sizeof(TONOS) / sizeof(TONOS[0]);

const int c = 261;
const int d = 294;
const int e = 329;
const int f = 349;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 466;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int dSH = 622;
const int eH = 659;
const int fH = 698;
const int fSH = 740;
const int gH = 784;
const int gSH = 830;
const int aH = 880;

/*-----------------------------------------------*/
//------------FUNCIONES IMPLEMENTADAS-------------//
/*-----------------------------------------------*/

/*
  Función para corregir el overflow de Arduino que ocurre aproxidamente una vez cada 50 días de uso continuo.
  Problema descrito en la documentación oficial de Arduino inherente a la naturaleza del tipo de datos que devuelve millis()
  https://www.arduino.cc/reference/en/language/functions/time/millis/
  https://www.arduino.cc/en/Reference.UnsignedLong
*/
unsigned long diferenciaCorrigiendoOverflow(unsigned long fin,unsigned long inicio)
{
  if (fin > inicio)
  {
    return (fin-inicio);
  }
  else
  {
    return (4294967295-inicio+fin);  //4294967295 es el máximo valor de un unsigned long
  }
}
```

```
/*
  Función para corrección de horas. Devuelve un valor siempre correcto de las horas no permitiendo valores fuera de rango para un reloj.
*/
byte corrigeHoras(byte val)
{
  if (val > 23 || val < 0)
  {
    return 0;
  }
  else
  {
    return val;
  }
}

/*
  Función para corrección de minutos o segundos. Devuelve un valor siempre correcto de las horas no permitiendo valores fuera de rango para un reloj.
*/
byte corrigeMinSeg(byte val)
{
  if (val > 59 || val < 0)
  {
    return 0;
  }
  else
  {
    return val;
  }
}

/*
  Función para corrección del marcador hasta 100.
  Para valores por encima de 100 solo contará el resto de dividir entre 100, es decir se queda con decenas y unidades.
*/
byte corrigeMarcador(byte val)
{
  if (val > 99)
  {
    return (val % 100);
  }
  else if (val < 0)
  {
    return 0;
  }
  else
  {
    return val;
  }
}
```

```
/*
  Función auxiliar para imprimir en monitor serie la fecha y hora
*/
#define countof(a) (sizeof(a) / sizeof(a[0]))
void printDateTime(const RtcDateTime &dt)
{
  char datestring[20];
  snprintf_P(datestring,
             countof(datestring),
             PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
             dt.Month(),
             dt.Day(),
             dt.Year(),
             dt.Hour(),
             dt.Minute(),
             dt.Second());
  Serial.print(datestring);
}

/*
  Función para iniciar el reloj copiando la hora del ordenador si el Arduino está
conectado por USB.
*/
RtcDS1302<ThreeWire> SetupReloj(RtcDS1302<ThreeWire> rtc)
{
  RtcDS1302<ThreeWire> myRtc = rtc;

  Serial.print("compiled: ");
  Serial.print(__DATE__);
  Serial.println(__TIME__);

  myRtc.Begin();

  RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
  Serial.print("EMPEZAMOS TOMANDO FECHA DEL SISTEMA: ");
  printDateTime(compiled);
  Serial.println();

  if (!myRtc.IsDateTimeValid())
  {
    // Common Causes:
    //    1) first time you ran and the device wasn't running yet
    //    2) the battery on the device is low or even missing

    Serial.println("RTC lost confidence in the DateTime! (first time )");
    myRtc.SetDateTime(compiled);
  }

  if (myRtc.GetIsWriteProtected())
  {
    Serial.println("RTC was write protected, enabling writing now");
    myRtc.SetIsWriteProtected(false);
  }
```

```
  if (!myRtc.GetIsRunning())
  {
    Serial.println("RTC was not actively running, starting now");
    myRtc.SetIsRunning(true);
  }

  RtcDateTime now = myRtc.GetDateTime();
  if (now < compiled)
  {
    Serial.println("RTC is older than compile time!  (Updating DateTime)");
    myRtc.SetDateTime(compiled);
  }
  else if (now > compiled)
  {
    Serial.println("RTC is newer than compile time. (this is expected)");
  }
  else if (now == compiled)
  {
    Serial.println("RTC is the same as compile time! (not expected but all is fine)");
  }

  return myRtc;
}

/*
  Función para hacer sonar en el buzzer pasivo una corta melodía con una serie de
tonos predefinidos
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarTono(int pin_SIG_buzzer)
{
  for (unsigned int iTono = 0; iTono < numTonos; iTono++)
  {
    tone(pin_SIG_buzzer, TONOS[iTono]);
    delay(200);
  }
  noTone(pin_SIG_buzzer);
}

/*
  Función para hacer sonar en el buzzer pasivo un pitido a una frecuencia durante
medio segundo
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarPitido(int pin_SIG_buzzer, int frec)
{
  tone(pin_SIG_buzzer, frec);
  delay(500);
  noTone(pin_SIG_buzzer);
}

/*
  Función para hacer sonar en el buzzer pasivo un pitido a una frecuencia un tiempo
definido en ms
```

```
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarPitido(int pin_SIG_buzzer, int frec, int ms)
{
  tone(pin_SIG_buzzer, frec);
  delay(ms);
  noTone(pin_SIG_buzzer);
}

/*
  Función para hacer sonar en el buzzer pasivo un pitido grave reservado para el botón
de APAGADO de leds
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarPitidoApagado(int pin_SIG_buzzer)
{
  SonarPitido(pin_SIG_buzzer, 150);
}

/*
  Función para hacer sonar en el buzzer pasivo un pitido agudo reservado para el botón
de PLAY/PAUSE
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarPitidoPlayPause(int pin_SIG_buzzer)
{
  SonarPitido(pin_SIG_buzzer, 1500);
}

/*
  Función para hacer sonar en el buzzer pasivo un pitido triple pensado para el reset
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarPitidoReset(int pin_SIG_buzzer)
{
  SonarPitido(pin_SIG_buzzer, b, 200);
  delay(20);
  SonarPitido(pin_SIG_buzzer, b, 200);
  delay(20);
  SonarPitido(pin_SIG_buzzer, b, 200);
  delay(20);
}

/*
  Función para hacer sonar en el buzzer pasivo un sonido de sirena
  Created by José Luis Guerrero Marín, May 9, 2020.
*/
void SonarSirena(int pin_SIG_buzzer)
{
  SonarPitido(pin_SIG_buzzer, b, 500);
  SonarPitido(pin_SIG_buzzer, gS, 500);
  delay(20);
  SonarPitido(pin_SIG_buzzer, b, 500);
  SonarPitido(pin_SIG_buzzer, gS, 500);
```

```
  delay(20);
  SonarPitido(pin_SIG_buzzer, b, 500);
  SonarPitido(pin_SIG_buzzer, gS, 500);
  delay(20);
}

/*
  Función para hacer sonar en el buzzer pasivo la marcha imperial de Star Wars
*/
void SonarMarchaImperial(int pin_SIG_buzzer)
{

//PRIMERA SECCIÓN
  SonarPitido(pin_SIG_buzzer, a, 500);
  SonarPitido(pin_SIG_buzzer, a, 500);
  SonarPitido(pin_SIG_buzzer, a, 500);
  SonarPitido(pin_SIG_buzzer, f, 350);
  SonarPitido(pin_SIG_buzzer, cH, 150);
  SonarPitido(pin_SIG_buzzer, a, 500);
  SonarPitido(pin_SIG_buzzer, f, 350);
  SonarPitido(pin_SIG_buzzer, cH, 150);
  SonarPitido(pin_SIG_buzzer, a, 650);
  delay(500);

  SonarPitido(pin_SIG_buzzer, eH, 500);
  SonarPitido(pin_SIG_buzzer, eH, 500);
  SonarPitido(pin_SIG_buzzer, eH, 500);
  SonarPitido(pin_SIG_buzzer, fH, 350);
  SonarPitido(pin_SIG_buzzer, cH, 150);
  SonarPitido(pin_SIG_buzzer, gS, 500);
  SonarPitido(pin_SIG_buzzer, f, 350);
  SonarPitido(pin_SIG_buzzer, cH, 150);
  SonarPitido(pin_SIG_buzzer, a, 650);
  delay(500);


//SEGUNDA SECCIÓN

  SonarPitido(pin_SIG_buzzer, aH, 500);
  SonarPitido(pin_SIG_buzzer, a, 300);
  SonarPitido(pin_SIG_buzzer, a, 150);
  SonarPitido(pin_SIG_buzzer, aH, 500);
  SonarPitido(pin_SIG_buzzer, gSH, 325);
  SonarPitido(pin_SIG_buzzer, gH, 175);
  SonarPitido(pin_SIG_buzzer, fSH, 125);
  SonarPitido(pin_SIG_buzzer, fH, 125);
  SonarPitido(pin_SIG_buzzer, fSH, 250);
  delay(325);

  SonarPitido(pin_SIG_buzzer, aS, 250);
  SonarPitido(pin_SIG_buzzer, dSH, 500);
  SonarPitido(pin_SIG_buzzer, dH, 325);
  SonarPitido(pin_SIG_buzzer, cSH, 175);
  SonarPitido(pin_SIG_buzzer, cH, 125);
```

```
  SonarPitido(pin_SIG_buzzer, b, 125);
  SonarPitido(pin_SIG_buzzer, cH, 250);
  delay(350);

// Variante 1
  SonarPitido(pin_SIG_buzzer, f, 250);
  SonarPitido(pin_SIG_buzzer, gS, 500);
  SonarPitido(pin_SIG_buzzer, f, 350);
  SonarPitido(pin_SIG_buzzer, a, 125);
  SonarPitido(pin_SIG_buzzer, cH, 500);
  SonarPitido(pin_SIG_buzzer, a, 375);
  SonarPitido(pin_SIG_buzzer, cH, 125);
  SonarPitido(pin_SIG_buzzer, eH, 650);
  delay(500);

// Se repite la segunda sección
  SonarPitido(pin_SIG_buzzer, aH, 500);
  SonarPitido(pin_SIG_buzzer, a, 300);
  SonarPitido(pin_SIG_buzzer, a, 150);
  SonarPitido(pin_SIG_buzzer, aH, 500);
  SonarPitido(pin_SIG_buzzer, gSH, 325);
  SonarPitido(pin_SIG_buzzer, gH, 175);
  SonarPitido(pin_SIG_buzzer, fSH, 125);
  SonarPitido(pin_SIG_buzzer, fH, 125);
  SonarPitido(pin_SIG_buzzer, fSH, 250);
  delay(325);

  SonarPitido(pin_SIG_buzzer, aS, 250);
  SonarPitido(pin_SIG_buzzer, dSH, 500);
  SonarPitido(pin_SIG_buzzer, dH, 325);
  SonarPitido(pin_SIG_buzzer, cSH, 175);
  SonarPitido(pin_SIG_buzzer, cH, 125);
  SonarPitido(pin_SIG_buzzer, b, 125);
  SonarPitido(pin_SIG_buzzer, cH, 250);
  delay(350);

// Variante 2
  SonarPitido(pin_SIG_buzzer, f, 250);
  SonarPitido(pin_SIG_buzzer, gS, 500);
  SonarPitido(pin_SIG_buzzer, f, 375);
  SonarPitido(pin_SIG_buzzer, cH, 125);
  SonarPitido(pin_SIG_buzzer, a, 500);
  SonarPitido(pin_SIG_buzzer, f, 375);
  SonarPitido(pin_SIG_buzzer, cH, 125);
  SonarPitido(pin_SIG_buzzer, a, 650);
  delay(650);
}
```