PROYECTO FINAL DE CARRERA

Diseño e implementación de un circuito electrónico inteligente para la emisión de sonido para control de plagas

Alumno: Daniel Marfil Reguero

Ingeniería Técnica Informática de Sistemas

Director del PFC: José Vicente Busquets Mataix

15/06/2011

Palabras Clave: Microcontrolador, Arduino, ATmega, DuinOS.

ÍNDICE

1. INTRODUCCIÓN AL PFC:

En este proyecto de final de carrera, se ha marcado el objetivo de diseñar e implementar un prototipo que sea capaz de detectar umbrales de sonido y generar, si dicho umbral cumple determinadas condiciones, un sonido como respuesta a dicha entrada.

Contaremos con un micrófono como forma de obtener los cambios en el ruido del lugar en el que nos encontremos y un tweeter para emitir el sonido.

Debemos de ser capaces de, utilizando un microcontrolador, poder leer el ruido que detecta el micrófono, decidir si éste supera el umbral definido y posteriormente emitir una alarma que además habremos programado nosotros.

Además se hará uso de un reloj, para que en todo momento se sepa en qué hora del día nos encontramos, ya que dependiendo del momento la sensibilidad del micrófono será mayor o menor.

Haremos uso también de una interfaz para la comunicación mediante el puerto serie con el ordenador. De esta forma visualizamos información como la hora del día, cuántas veces ha saltado el sonido o la sensibilidad en ese momento del micrófono. Además de poder cambiar de forma manual esta última.

Finalmente también deberemos configurar un pulsador para que cuando se active, salte el sonido.

2. DESCRIPCIÓN DE COMPONENTES:

Tras acordar hacer uso de un microcontrolador ATMega 328 (integrado en Arduino UNO, que explicaremos posteriormente), se contempló como primera opción utilizar el entorno de desarrollo en C AVR-CodeVision. Más tarde el proyecto se implementó con el entorno de desarrollo de Arduino, al ser un software open-source, gratuito y muy intuitivo.

El microcontrolador ATMega 328 sigue la arquitectura AVR. Los AVR son una familia de microcontroladores RISC de Atmel.

El AVR es una CPU de arquitectura Harvard. Tiene 32 registros de 8 bits. Algunas instrucciones sólo operan en un subconjunto de estos registros. La concatenación de los 32 registros, los registros de entrada/salida y la memoria de datos conforman un espacio de direcciones unificado, al cual se accede a través de operaciones de carga/almacenamiento. A diferencia de los microcontroladores PIC, el stack se ubica en este espacio de memoria unificado, y no está limitado a un tamaño fijo.

El AVR fue diseñado desde un comienzo para la ejecución eficiente de código C compilado. Como este lenguaje utiliza profusamente punteros para el manejo de variables en memoria, los tres últimos pares de registros internos del procesador, son usados como punteros de 16 bit al espacio de memoria externa, bajo los nombres X, Y y Z. Esto es un compromiso que se hace en arquitecturas de ocho bit desde los tiempos de Intel 8008, ya que su tamaño de palabra nativo de 8 bit (256 localidades accedidas) es pobre para direccionar. Por otro lado, hacer que todo el banco superior de 16 registros de 8 bit tenga un comportamiento alterno como un banco de 8 registros de 16 bit, complicaría mucho el diseño, violando la premisa original de su simplicidad.

El set de instrucciones AVR está implementado físicamente y disponible en el mercado en diferentes dispositivos, que comparten el mismo núcleo AVR pero tienen distintos periféricos y cantidades de RAM y ROM: desde el microcontrolador de la familia Tiny AVR ATtiny11 con 1KB de memoria flash y sin RAM (sólo los 32 registros), y 8 pines, hasta el microcontrolador de la famila Mega AVRATmega2560 con 256KB de memoria flash, 8KB de memoria RAM, 4KB de memoria EEPROM, conversor análogo digital de 10 bits y 16 canales, temporizadores, comparador analógico, JTAG, etc. La compatibilidad entre los distintos modelos es preservada en un grado razonable.

Los microcontroladores AVR tiene un pipeline con dos etapas (cargar y ejecutar), que les permite ejecutar la mayoría en un ciclo de reloj, lo que los hace relativamente rápidos entre los microcontroladores de 8-bit.

El set de instrucciones de los AVR es más regular que la de la mayoría de los microcontroladores de 8-bit (por ejemplo, los PIC). Sin embargo, no es completamente ortogonal:

-Los registros punteros X, Y y Z tienen capacidades de direccionamiento diferentes entre sí.

-Los registros 0 al 15 tienen diferentes capacidades de direccionamiento que los registros 16 al 31.

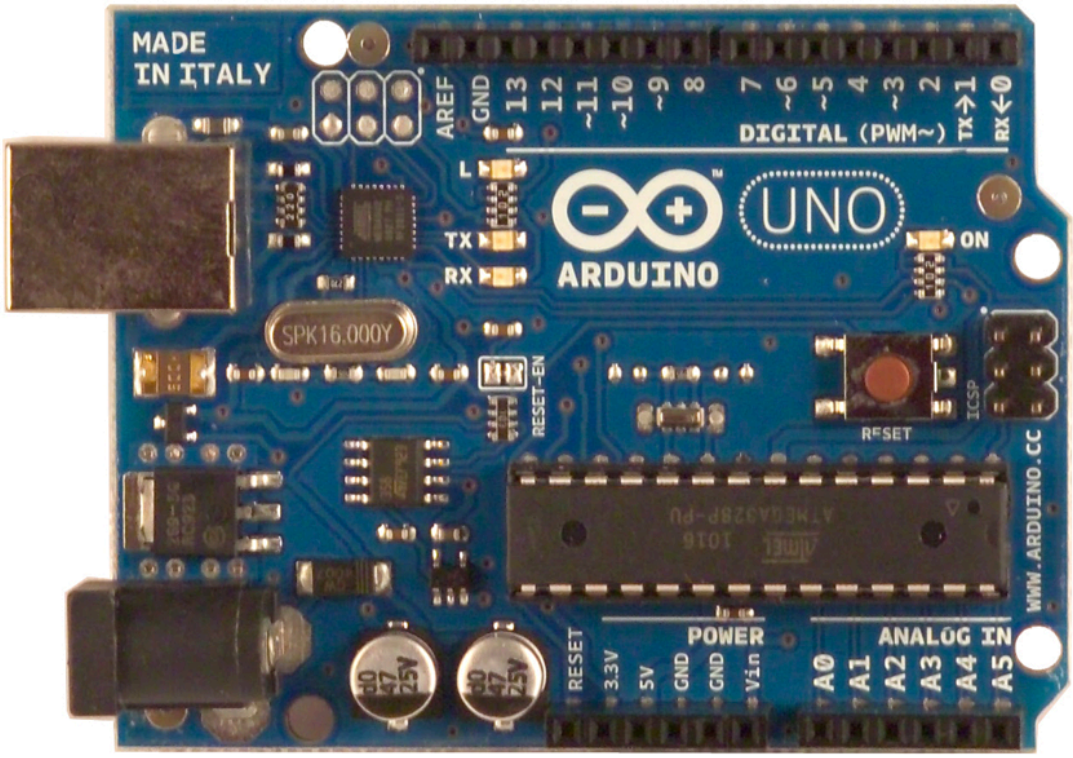
-Las registros de I/O 0 al 31 tienen distintas características que las posiciones 32 al 63.

-La instrucción CLR afecta los 'flag', mientras que la instrucción SER no lo hace, a pesar de que parecen ser instrucciones complementarias (dejar todos los bits en 1, y dejar todos los bits en 0 respectivamente).

Como los PIC, tiene una comunidad de seguidores principalmente debido a la existencia de herramientas de desarrollo gratuitas o de bajo coste. Estos microcontroladores están soportados por tarjetas de desarrollo de costo razonable, capaces de descargar el código al microcontrolador, y por una versión de las herramientas GNU. Esto último es posible por su uniformidad en el acceso al espacio de memoria, propiedad de la que carecen los procesadores de memoria segmentada o por bancos, como el PIC o el 8051 y sus derivados.

El ATMega 328 cuenta con 32KB de memoria flash, 2KB de memoria RAM y 1KB de memoria EEPROM (el doble que el ATMega 168).

2.1. USO DEL ARDUINO UNO:



Arduino es una plataforma de hardware libre basada en una sencilla placa con un microcontrolador y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring. Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Las plataformas Arduino están basadas en los microcontroladores Atmega168, Atmega328, Atmega1280, ATmega8 y otros similares, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños.

Al ser open-hardware, tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

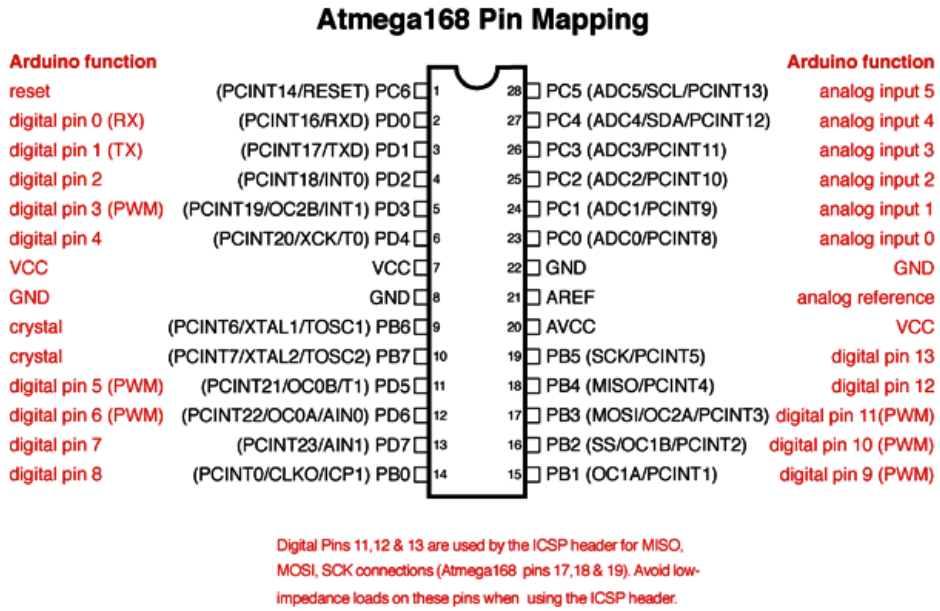
Arduino UNO es una placa basada en el microcontrolador ATMega328. Tiene 14 pines de E/S digital (6 de los cuales pueden ser utilizados como salidas PWM), 6 entradas analógicas, un oscilador de cristal de 16MHz, conexión USB, un botón de reset, cabeceras ICSP y una entrada de alimentación. Contiene todo lo necesario para hacer funcionar el microcontrolador, simplemente se conecta a un ordenador mediante un cable USB o se alimenta con pilas o adaptadores a corriente continua.

Arduino UNO se diferencia del resto de placas de Arduino en que no hace uso del driver para el chip USB-a-serial FDTI. En lugar de esto, utiliza un ATMega8U2 programado para comportarse como un conversor USB a serie.

En resumen Arduino UNO consta de las siguientes características:

Microcontrolador	ATmega328
Tensión operativa	5V
Tensión de entrada (recomendada)	7-12V
Tensión de entrada (límite)	6-20V
Pines E/S digitales	14 (de los cuales 6 pueden generar PWM)
Pines de entrada analógica	6
Corriente por pines E/S	40 mA
Corriente para el pin de 3.3V	50 mA
Memoria Flash	32 KB 0.5 KB utilizados para el bootloader
SRAM	2 KB
EEPROM	1 KB
Velocidad del reloj	16 MHz

El mapeado realizado por Arduino para el microcontrolador ATMega328 es el siguiente:



2.1.1. INTRODUCCIÓN A LA PROGRAMACIÓN:

El lenguaje de programación de Arduino es el "wiring", este lenguaje está basado en el lenguaje "processing".

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue iniciado por Ben Fry y Casey Reas a partir de reflexiones en el Aesthetics and Computation Group del MIT Media Lab.

Processing es desarrollado por artistas y diseñadores como una herramienta alternativa al software propietario. Puede ser utilizado tanto para aplicaciones locales así como aplicaciones para la web (Applets).

Se distribuye bajo la licencia GNU GPL.

2.1.2. IMPLEMENTACIÓN DE DuinOS:

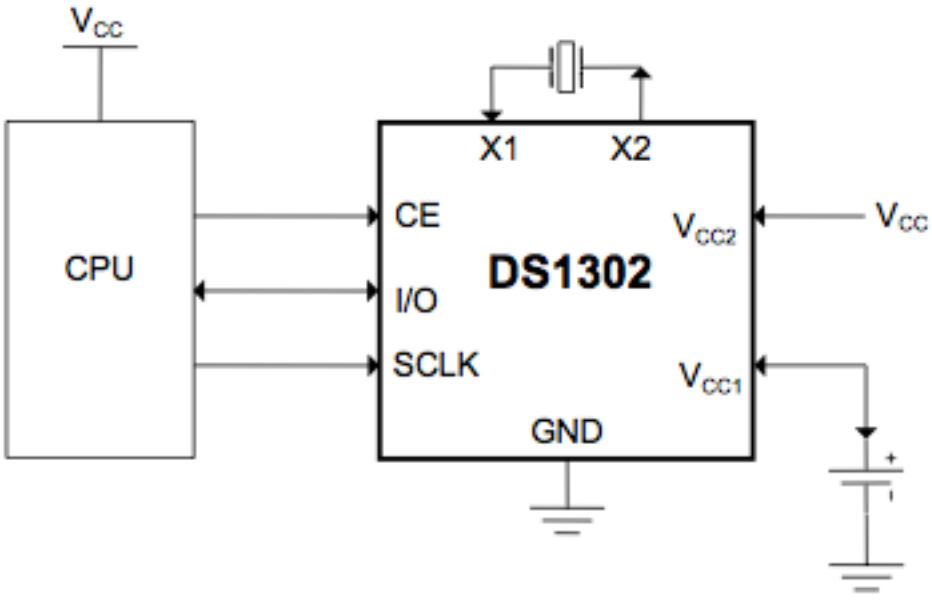
Debido a las características del proyecto, una forma de optimizar su funcionamiento es dividiendo el trabajo en tareas que trabajen de forma paralela. Debido a esto, se ha hecho uso del sistema operativo DuinOS, que se puede integrar fácilmente en el entorno de desarrollo de Arduino y nos permite crear tareas, además de asignar a cada una tres tipos distintos de prioridad.

DuinOS es un sistema operativo de tiempo real desarrollado por RobotGroup. Este sistema agrega multitarea preventiva al entorno Arduino. Está basado en un robusto kernel como lo es FreeRTOS y es completamente de código abierto.

En la explicación del código se hará más claro el uso y la necesidad de implementar ese sencillo sistema operativo.

2.2. USO DEL RTC DS1302:

El chip DS1302 contiene un reloj de tiempo real/calendario de 31 Bytes de capacidad SRAM. Se comunica con el microcontrolador mediante una sencilla interfaz en serie. Este reloj proporciona los segundos, minutos, horas, día, fecha, mes y año. El final de cada mes está auto-ajustado incluyendo los años bisiestos. Además el reloj puede operar en formato de 24h o 12h con indicador AM/PM.



Para poder utilizar este reloj de tiempo real, se ha tenido que obtener unas librerías específicas para su correcto uso.

3. ESPECIFICACIÓN DE REQUISITOS:

Este proyecto debe cumplir unos determinados objetivos:

-El micrófono deberá amplificar su señal mediante un circuito, ya que sin él la señal generada es muy débil y el microcontrolador no es capaz de leer ningún valor.

-El sonido emitido deberá recorrer una frecuencia desde 1'6KHz hasta 20KHz, siendo de esta forma molesta y audible sin ningún tipo de problemas.

-Deberemos configurar un reloj de tiempo real (RTC DS1302) con la fecha y hora actual, para poder dividir el día en tres franjas. De esta forma se podrá cambiar automáticamente la sensibilidad del micrófono según en qué momento del día nos encontremos.

-El micrófono genera una señal analógica, por lo que el microcontrolador deberá ser capaz de leer esta señal y convertirla a señal digital. Además se mapeará esta señal para que recorra un rango de cero a cien, para facilitar la configuración de la sensibilidad y ser más intuitiva.

-Haciendo uso de la alimentación del propio microcontrolador, deberemos suministrar energía al RTC, al esquema de amplificación del micrófono y al circuito del pulsador para activar el sonido manualmente. El tweeter se conectará desde el pin que genere la frecuencia de señal cuadrada a tierra.

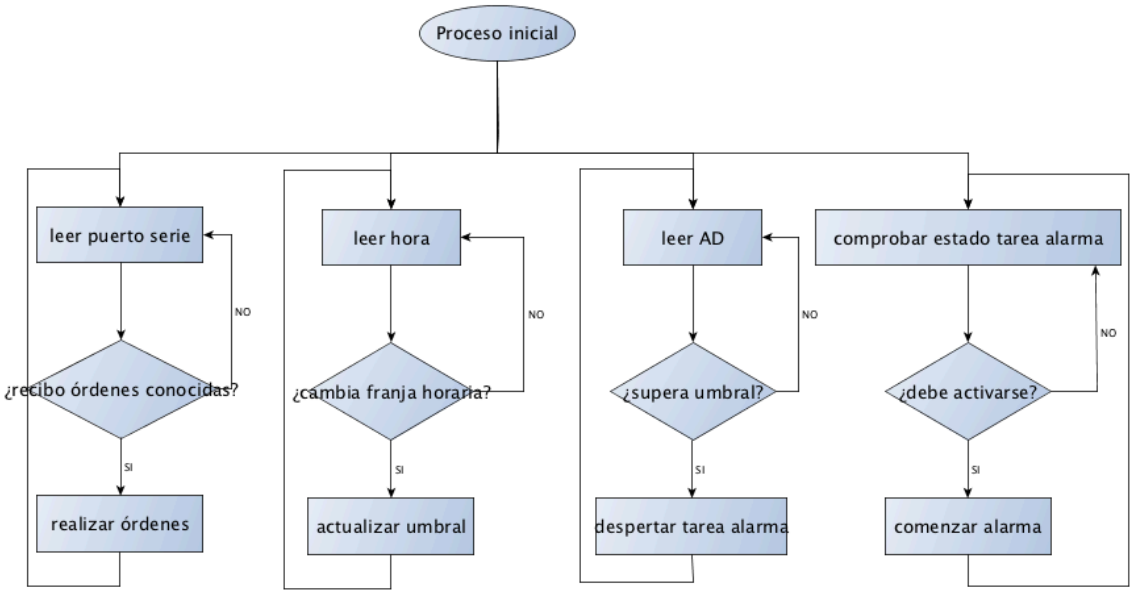
4. DISEÑO E IMPLEMENTACIÓN:

El código fuente original se puede ver en el Anexo I de este documento.

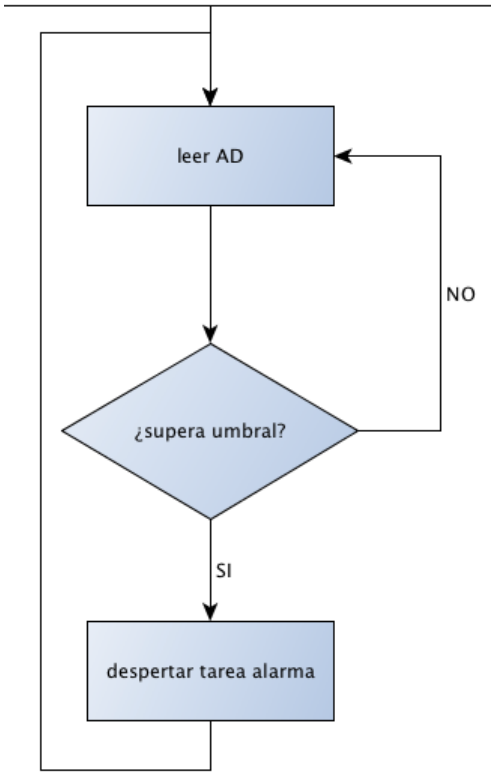
Se explicará ahora el código implementado por partes, según su funcionalidad.

Hay que aclarar que en el código llamaremos alarma a la emisión del sonido programado, ya que a pesar de no ser una alarma como tal, realiza una función similar. Por tanto se entenderá que cuando hablamos de la alarma, nos referimos a la emisión del sonido creado.

Vista general del diagrama de flujo del programa, se expondrá cada tarea individualmente para una mayor comprensión más adelante:



4.1. LECTURA DE SEÑALES:




```
//VARIABLES PARA LECTURA DE INFORMACION
const int buttonPin = 5;// pin pulsador
int estadoBoton;//variable que cambiara segun el pulsador
boolean activado = false;//comprobara que se ha pulsado el boton
const int sensorPin = A0;//pin para la entrada analogica del microfono
int sensorValue = 0;// variable para almacenar el valor de entrada analogica
int oldSensorValue = sensorValue;//variable utilizada para comparar
                                 //el valor de entrada anterior con el actual
```

Las variables buttonPin y sensorPin se utilizarán para definir los pines de entrada. El primero se refiere a la entrada del pulsador y el segundo a la entrada analógica del micrófono. La variable estadoBoton leerá la señal que se reciba de buttonPin, estando las opciones HIGH y LOW y el booleano activado indicará si se ha pulsado el botón y debe saltar el sonido o no. Las variables sensorValue y oldSensorValue se utilizarán para la entrada analógica del micrófono, siendo la entrada actual sensorValue y la entrada obtenida el instante anterior oldSensorValue.

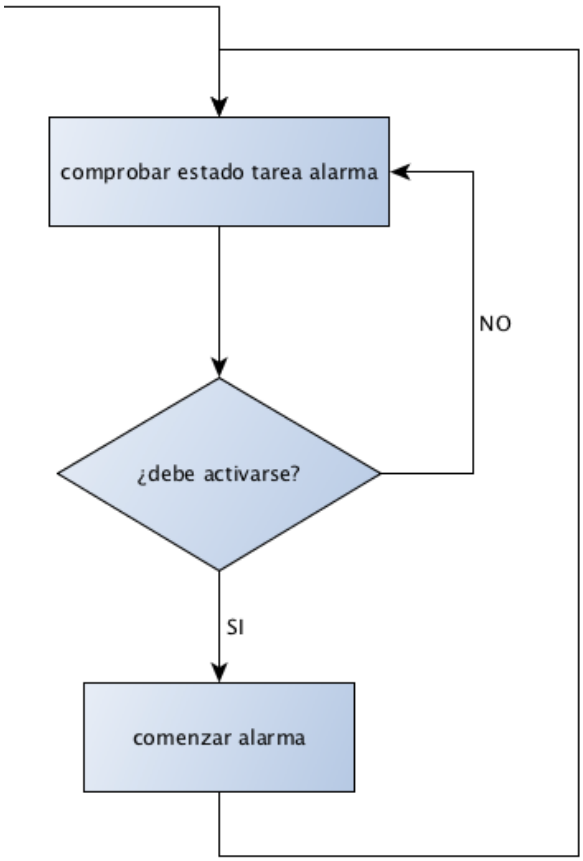
Ahora veremos la tarea lectura:

```
/**TAREA LECTURA DE DATOS
//tarea encargada de leer tanto el microfono como el pulsador
//si se pulsa el boton se debera activar la alarma
//al igual que si se supera la sensibilidad limite*/
taskLoop(lectura){
  estadoBoton = digitalRead(buttonPin);
  oldSensorValue = sensorValue;
  sensorValue = analogRead(sensorPin);
  sensorValue=map(sensorValue, 0, 1023, 0, 100);
  if(estadoBoton == HIGH){
    if(!activado) activado = true;
  }

  if(sensorValue - oldSensorValue > sensibilidad || activado){
    resumeTask(alarma);
  }

  resumeTask(consultarRTC);
}
```

Teniendo en cuenta lo explicado anteriormente, vemos como se almacena la información en las variables indicadas. Se define el rango de entrada del micrófono de 0 a 100 y además se comprueba si se está pulsando el botón.

Si bien el botón ha sido pulsado o bien se ha detectado una diferencia mayor entre los valores de entrada analógica mayores que la sensibilidad establecida, deberá despertarse a la tarea alarma, la cual explicaremos a continuación, al igual que la variable sensibilidad.

4.2. EMISIÓN DEL SONIDO:




```
//VARIABLES PARA LA ALARMA
int cont = 0;
const int speaker=7;//pin que emitira señal de alarma
const int minimo = 620; //periodo microseg para 1'6kHz
const int maximo = 50; //periodo microseg para 20kHz
int valor = minimo;//variable utilizada para que el sonido emitido varie de tonalidad
```

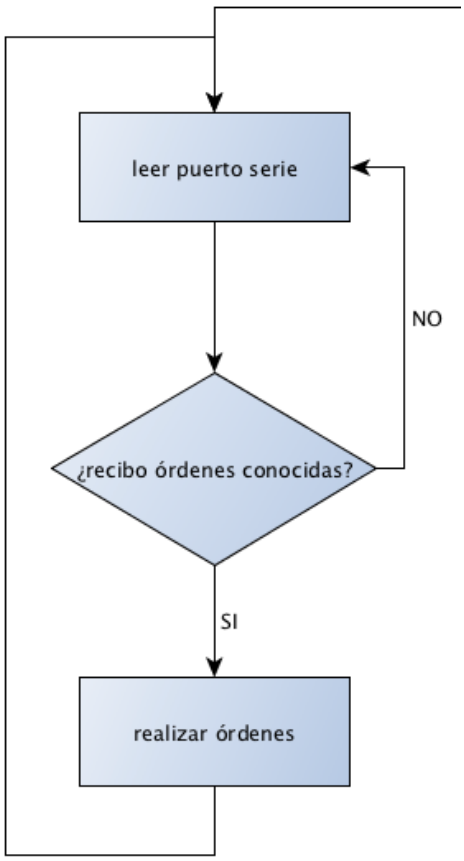
Estas variables se utilizarán en la tarea del sonido. La variable speaker sirve para definir el pin por el que se emitirá la señal de salida, estará conectado a un tweeter para convertirla a una señal acústica. Las constantes minimo y maximo sirven para delimitar el rango de frecuencias por las que la señal se transmitirá, la variable valor se utilizara para variar dicha frecuencia. El entero cont lo utilizaremos para definir el tiempo que debe de sonar el sonido.

```
/**TAREA ALARMA
//tarea encargada en hacer saltar la alarma
//esta se activara y recorrera una franja de
//frecuencias molestas durante un tiempo determinado por
//la variable interna cont*/
taskLoop(alarma){
  do{
    for(int i=0;i<5;i++){
      digitalWrite(speaker,HIGH);
      delayMicroseconds(valor/2);

      digitalWrite(speaker,LOW);
      delayMicroseconds(valor/2);
    }
    valor = valor - 2;
    if(valor==maximo+2){valor=minimo;}
    cont++;
  }while(valor>=maximo && cont < 1000);
  nalarmas++;
  valor = minimo;
  cont=0;
  if(activado){
    estadoBoton = digitalRead(buttonPin);
    if(activado) activado = false;
  }
  suspend();
}
```

Aquí vemos el cuerpo de la tarea alarma. Vemos como variando la variable valor, iremos cambiando la frecuencia de la señal de salida para el tweeter, todo esto mientras cont sea menor de 1000. Este límite se puede aumentar o disminuir para cambiar la duración del sonido. Una vez ha dejado de sonar el sonido, se incremente el contador nalarmas (explicado en el apartado de comunicación con el puerto serie). Además de cambiar a falso el booleano activado (que nos indicaba que el pulsador habia sido pueso a HIGH).

Tras esto, la tarea alarma queda en estado de suspensión hasta que la tarea lectura la despierte de nuevo bien porque se ha pulsado el botón, bien porque se haya superado el umbral de sonido determinado por la sensibilidad del micrófono.

4.3. COMUNICACIÓN CON EL PUERTO SERIE:



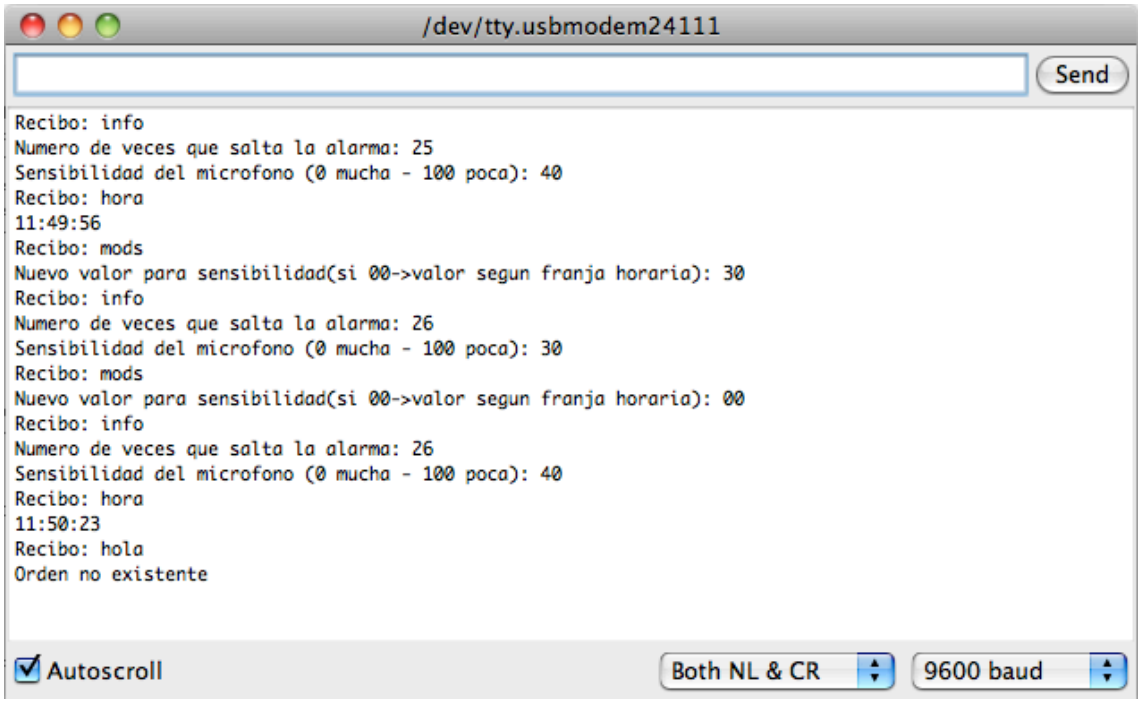
En esta tarea, no será necesario declarar variables globales propias, aunque sí se utilizarán otras variables globales ya explicadas o que se explicarán en el apartado de la consulta al RTC DS1302.

En un primer momento, se vacía el buffer del puerto serie, para evitar que haya una mala comunicación a la hora de transmitir y recibir datos.

La variable input constara de cuatro caracteres y será donde almacenemos la entrada del puerto serie, previo envío del carácter '!', utilizado como identificación de que va a llegar al puerto serie una órden que debe ser interpretada.

Se han definido tres órdenes: info, hora y mods. La forma de escribirlo en el terminal sera anteponiendo el signo de exclamación (p.ej. !info). Si se escribe alguna cadena con mas de cuatro caracteres, se leerán los cuatro primeros y el resto serán ignorados. Además si la orden es desconocida se advertirá mediante un mensaje del error.

Un ejemplo de las funciones de esta tarea es la siguiente imagen:



Tanto la órden info como hora son solamente a modo de consulta. En cambio la órden mods (modificar sensibilidad) deberá acompañarse de un número de dos dígitos, por ejemplo !mods50, con lo cual cambiaría la sensibilidad a 50 hasta que se envía la órden !mods00, que hace volver a la sensibilidad establecida según la hora del día. Esto es así porque se ha programado que si a dicha órden se le envía los dígitos 00, debe volver a la sensibilidad por defecto dependiente de la hora.

Veamos ahora el código implementado:

```
/**TAREA COMUNICACION
//Se encarga, segun los comandos de 4 caracteres propuestos,
//de modificar la sensibilidad del microfono
//o de solicitar informacion sobre la configuracion
//de la sensibilidad actual y el numero de veces
//que ha saltado la alarma, ademas de poder consultar la
//hora*/
taskLoop(comunicacionSerie){
  Serial.flush();
  char input[5];
  char sens[5];
  char talarm[5];
  char modsens[4] = {'m', 'o', 'd', 's'};
  char infor[4] = {'i', 'n', 'f', 'o'};
  char tiempo[4] = {'h', 'o', 'r', 'a'};
  memset(input,'\0',5);
  byte inByte = '\0';
  boolean modifisens = false;
  boolean inform = false;
  boolean tmp = false;
  int nuevasens = 0;
  char nsens[3];
  memset(nsens,'\0',3);

  while(inByte != '!'){inByte = Serial.read();}
```

Hasta aquí lo que se ha escrito son las variables locales utilizadas para registrar los datos de entrada y las órdenes que hay programadas.

```
if(inByte == '!'){
  while(Serial.available() < 4) {;}
  for (int i=0; i < 4; i++) {
    input[i] = Serial.read();
  }

  delay(200);
  Serial.print("Recibo: ");
  Serial.println(input);

  if(input[0] == modsens[0] && input[1] == modsens[1] &&
    input[2] == modsens[2] && input[3] == modsens[3]) modifisens = true;
  else if(input[0] == infor[0] && input[1] == infor[1] &&
    input[2] == infor[2] && input[3] == infor[3]) inform = true;
  else if(input[0] == tiempo[0] && input[1] == tiempo[1] &&
    input[2] == tiempo[2] && input[3] == tiempo[3]) tmp = true;
  else{delay(200); Serial.println("Orden no existente");}

  if(modifisens){
    while(Serial.available() < 2) {;}
    for (int i=0; i < 2; i++) {
      nsens[i] = Serial.read();
    }
    nuevasens = atoi(nsens);
    if(nuevasens != 0){sensman = true; sensibilidad = nuevasens;}
    else sensman = false;
    delay(200);
    Serial.print("Nuevo valor para sensibilidad(si 00->valor segun franja horaria): ");
    Serial.println(nsens);
    if(nuevasens != 0)sensman = true;
    else sensman = false;
  }
```

Como se ha explicado antes, si recibimos el signo de exclamación pasaremos a esperar cuatro caracteres que deberán interpretarse dando una respuesta correspondiente. Al solo poder recibir/enviar caracteres, se deberán usar funciones como atoi() para poder interpretarlas de forma correcta. Por ejemplo hacemos uso de esta función cuando queremos cambiar manualmente la sensibilidad del micrófono.

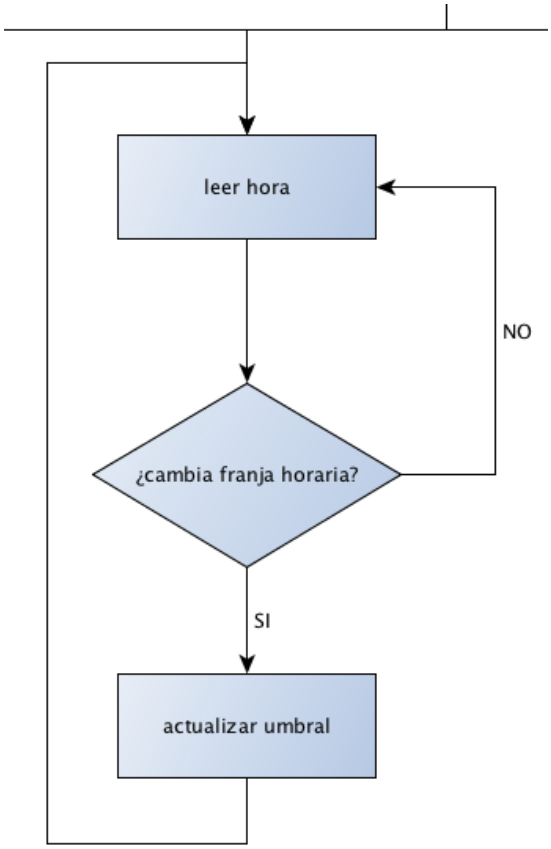
Un caso concreto es la órden mods, podemos ver que tras detectar dicha órden, debemos recibir además otros dos caracteres, que deberán convertirse en enteros para actualizar así la sensibilidad. Para volver a la sensibilidad por defecto (según la franja del día) los dos caracteres a enviar deberán ser 00.

```
    if(inform){
      delay(100);
      Serial.print("Numero de veces que salta la alarma: ");
      itoa(nalarmas, talarm, 10);
      Serial.println(talarm);
      delay(100);
      Serial.print("Sensibilidad del microfono (0 mucha - 100 poca): ");
      itoa(sensibilidad, sens, 10);
      Serial.println(sens);
    }

    if(tmp){
      delay(100);
      Serial.println(hhmm);
    }
  }
}
```

En esta última parte vemos que al enviar la órden info obtenemos el número de veces que ha saltado el sonido y también la sensibilidad a la que está configurado el micrófono actualmente.

También apreciamos que con la órden hora recibimos como respuesta la hora, almacenada en la variable hhmm, explicada a continuación.

4.4. CONSULTA DE LA HORA:



En este apartado explicaremos las variables, método y tarea encargados de configurar y consultar el RTC DS1302, haciendo uso de una librería open-source obtenida de http://www.henningkarlsen.com/electronics/a_l_ds1302.php.

Para ello, tras haber instalado la librería en el entorno Arduino, deberemos incluir al principio del programa la línea

#include <DS1302.h>

Tras esto ya podremos declarar una variable reloj, indicando los pines que se utilizarán, como podemos ver el la próxima imagen. Además podemos ver las distintas sensibilidades definidas para cada franja del día. Por otro lado vemos un booleano que se encargará de indicar si se ha cambiado manualmente la sensibilidad, por lo que se deberá ignorar la franja del día en la que se esté.

Podemos observar que hemos dividido el día en tres franjas: mañana (6 a 11:59:59), tarde (12 a 19:59:59) y noche (20 a 5:59:59).

La variable nalarmas de encargará de contar las veces que ha saltado el sonido, en la tarea con el mismo nombre. La variable hhmm se encargará de almacenar la hora consultada al RTC DS1302.

```
//VARIABLES PARA LA ACTUALIZACION Y COMUNICACION CON EL RTC
DS1302 rtc(2, 3, 4);
// DS1302:  CE pin    -> Arduino Digital 2
//          I/O pin   -> Arduino Digital 3
//          SCLK pin  -> Arduino Digital 4
int sensibilidadManana = 40; //...
int sensibilidadTarde = 60; //...----->sensibilidades segun parte del dia
int sensibilidadNoche = 20; //...
int sensibilidad = 90; //inicialmente, se actualiza de inmediato al consultar el rtc
boolean sensman = false;//indicara si se ha cambiado manualmente la sensibilidad
char* hhmm;//utilizado en la tarea de consultar al rtc
int finmanana =12;//hora en la que acaba la mañana
int fintarde=20;//hora en la que acaba la tarde
int finnoche=06;//hora en la que acaba la noche
int nalarmas = -1;//Inicialmente a -1 porque al encenderse el arduino, salta la alarma
                  //contador de alarma
```

Como podemos apreciar, la tarea consultaRTC es muy sencilla, ya que solamente debe consultar la hora al reloj mediante getTimeStr(), que nos devolverá una cadena de char. Posteriormente y mediante el método ecuentraRango, que tendrá como entrada la hora, deberemos obtener la sensibilidad correspondiente tras haber confirmado la hora en la que estamos.

```
/**TAREA CONSULTAR RTC
//tarea encargada de consultar el rtc
//y obtener la hora, ademas de actualizar
//la sensibilidad llamando al metodo
//encuentraRango()*/
taskLoop(consultarRTC){
  delay(1000);
  hhmm = rtc.getTimeStr();
  sensibilidad = encuentraRango(hhmm);
}
```

El siguiente método se encarga de deducir la hora en la que nos encontramos y devolver la sensibilidad correspondiente:

```
/**METODO OBTENER SENSIBILIDAD SEGUN FRANJA HORARIA
//metodo encargado de obtener la franja horaria en la que nos encontramos
//y modificar la sensibilidad segun dicha franja
//recibe un char* con la hora del rtc en formato hh:mm
//devuelve la sensibilidad con la que se debe comprobar la entrada del microfono*/
int encuentraRango(char* hora){
  if(!sensman){
    char aux[2]={0, 0};
    int horaint = 0;
    for(int i=0;i<2;i++){//solo los 2 primeros componentes del string -> hh:mm queremos hh
      aux[i]=hora[i];
    }
    horaint = atoi(aux);
    if(horaint >= finnoche && horaint < finmanana){return sensibilidadManana;}
    if(horaint >= finmanana && horaint < fintarde){return sensibilidadTarde;}
    if(horaint >= fintarde && horaint < finnoche){return sensibilidadNoche;}
  }
  return sensibilidad;
}
```

Puesto que solamente nos interesa saber la hora y ni siquiera los minutos, debemos comparar los dos primeros componentes del string con las variables finmanana, fintarde y finnoche. Mediante una serie de condiciones podemos obtener la sensibilidad correspondiente según la franja del día.

Por otro lado vemos que todo lo dicho se ejecutará únicamente si no se ha cambiado la señal manualmente, es decir, si la variable sensman está a trae, significará que la hemos modificado personalmente y por tanto, no se deberá comprobar en qué hora estamos para actualizar la sensibilidad. Aunque si podremos consultar la hora a través del puerto serie.

4.5. SETUP Y LOOP:

```
/**SETUP
//aqui se inicializaran las tareas indicando su prioridad, determinar
//pines de entrada y salida, inicializacion del puerto serie
//ademas de programar inicialmente la fecha para el rtc
*/
void setup() {
  pinMode(speaker, OUTPUT);
  pinMode(buttonPin, INPUT);

  createTaskLoop(lectura, NORMAL_PRIORITY);
  createTaskLoop(consultarRTC, NORMAL_PRIORITY);
  createTaskLoop(alarma, HIGH_PRIORITY);
  createTaskLoop(comunicacionSerie, NORMAL_PRIORITY);

  Serial.begin(9600);

  // poner reloj en run-mode y deshabilitar proteccion de escritura
  rtc.halt(false);
  rtc.writeProtect(false);
  //Para sobreescribir valores del rtc, descomentar las lineas de abajo
  //rtc.setDOW(THURSDAY);        // Set Day-of-Week to FRIDAY
  //rtc.setTime(18, 10, 0);     // Set the time to 12:00:00 (24hr format)
  //rtc.setDate(9, 6, 2011);   // Set the date to August 6th, 2010
}


/**TAREA LOOP
//tarea propia del entorno arduino
//por defecto sera de prioridad baja
//se debe poner nextTask() para evitar
//interbloqueos
*/
void loop() {
  nextTask();
}
```

Estas tareas son propias y obligatorias para programar con Arduino.

En el setup() definimos opciones tales como declarar si ciertos pines son de entrada o salida, como por ejemplo speaker debe ser de salida o el pulsador de entrada.

Además al hacer uso del reloj RTC DS1302 se deben quitar los permisos de protección de escritura y ponerlo en run-mode, lo cual se consigue con los métodos writeProtection() y halt() utilizando como parámetros de entrada el booleano false.

Por otra parte si queremos configurar inicialmente la hora, fecha y día de la semana del reloj, se deberá descomentar los comandos setDOW, setTime y setDate. Lo común es configurarlo la primera vez y si posteriormente se actualiza parte del código, comentarlo para que no se desactualice la hora.

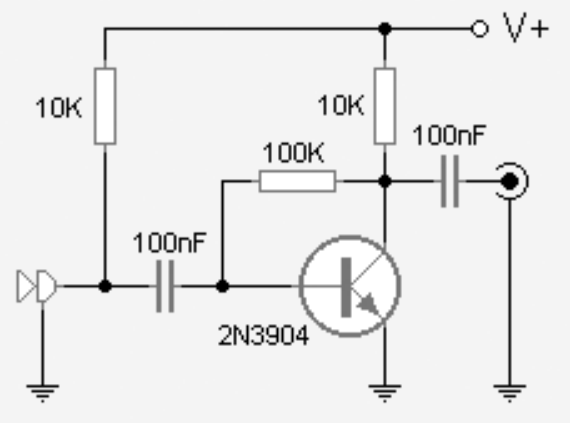
El loop() si no hicieramos uso de tareas, es donde debería ir todo el código. Pero al hacer uso del DuinOS, este queda vacío y debemos poner en él la órden nextTask(), esto está indicado por parte del propio SO, y se escribe para evitar interbloqueos y no se llegue a colgar el SO.

Con esto está explicado todo el código necesario y relevante que necesita ser explicado. Adjunto a este documento está el código completo del proyecto.

5. ESQUEMAS EXTERNOS:

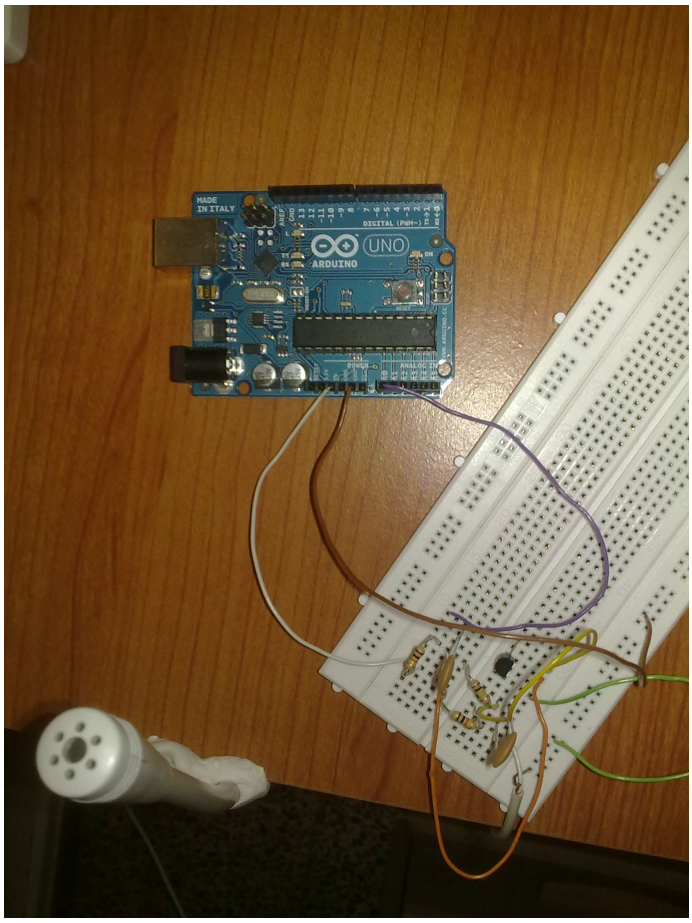
5.1. ESQUEMA DEL MICRÓFONO:

Como entrada al sistema, tenemos un micrófono, del cual tenemos que amplificar para que llegue una señal detectable por el Arduino UNO. El esquema a implementar es el siguiente:



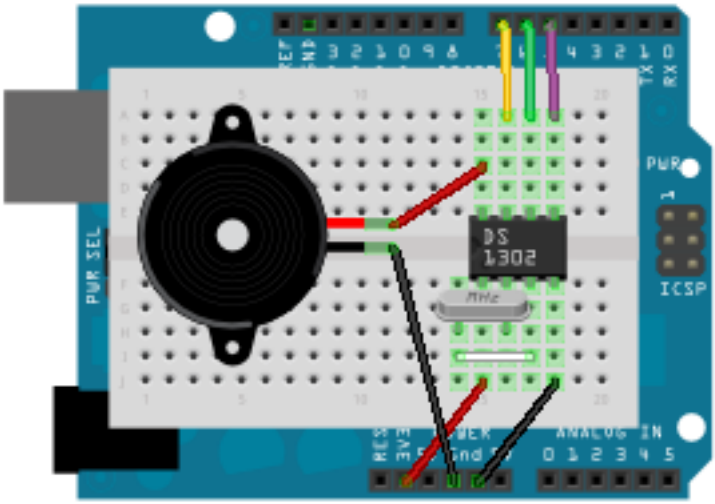
Siendo la fuente de alimentación el propio Arduino. Este es un circuito sencillo que nos permite aumentar la tensión dependiendo de la entrada del micrófono y detectando este hecho en el Arduino.

Este es dicho esquema implementado en una proto-board:

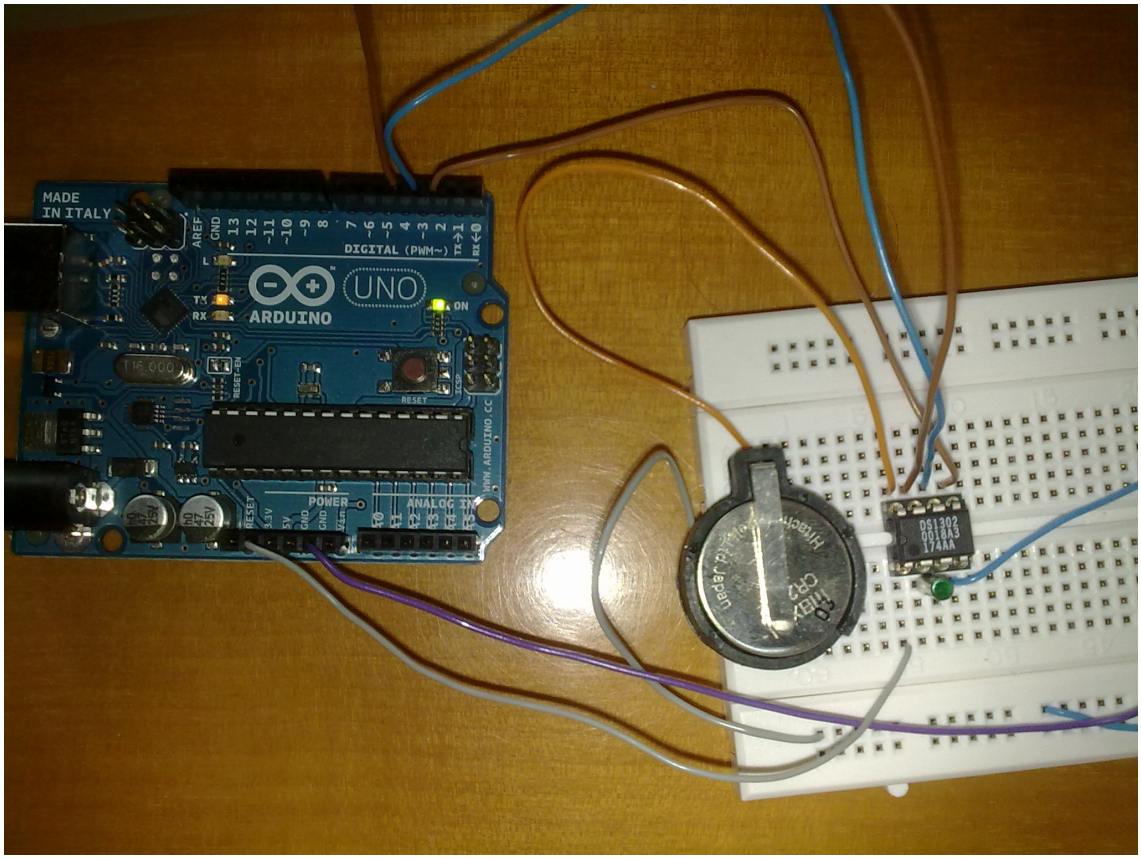


5.2. ESQUEMA DEL RTC DS1302:

Se hace uso de un RTC, en concreto del DS1302, para que, según la hora del día, nuestro microcontrolador sepa cuál es la sensibilidad a la que debe hacer saltar el sonido. Para ello programamos el ds1302 como nos indica la siguiente imagen y lo programamos con la fecha y hora actual, para su posterior uso por parte del microcontrolador.



Esta es una imagen en la que vemos implementado el RTC en una proto-board:



5.3. ESQUEMA FINAL:

Este diseño es el final, en el que vemos el micrófono con su esquema de amplificación, el RTC DS1302 conectado y además hemos añadido un pulsador que activará el sonido, con la misma duración que cuando esta salta de forma automática.

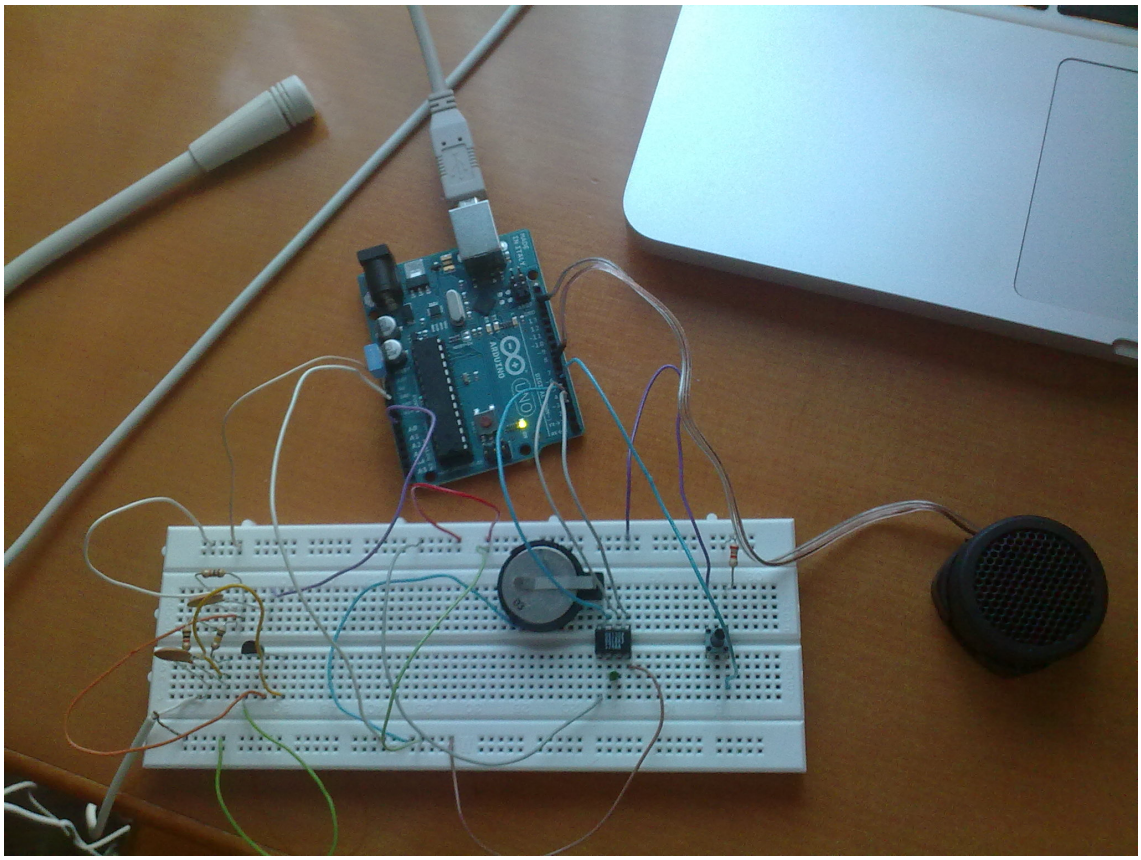
El tweeter además está conectado a tierra y al pin 7, no siendo necesario un circuito de amplificación ya que el ruido emitido es suficiente y en parte, molesto.

Por otro lado, se ha añadido un condensador de 100nF, conectando los pines RESET y GND de Arduino. Esto se ha hecho para evitar que al conectar por usb el microcontrolador, se auto-reinicie, ya que por defecto así lo hace y esta es la única forma de evitarlo si no se quiere desoldar parte de la placa.

Una anotación es que a la hora de volver a programar la placa, se debe quitar dicho condensador, para que se puedan transmitir los datos, reiniciar la placa y ejecutar el nuevo código.

Para utilizar este RTC con Arduino, se ha usado la librería de:

http://www.henningkarlsen.com/electronics/a_l_ds1302.php



6. CONCLUSIÓN PERSONAL:

He de reconocer que al elegir este proyecto, no sabía realmente cómo iba hacerlo ni implementarlo, ni con qué materiales contaría.

Pero finalmente el proyecto de fin de carrera que acabamos de ver, me ha aportado unos conocimientos y una experiencia que antes no tenía y que recomiendo fuertemente a todos los interesados en una programación no tan enfocada a una "pantalla de ordenador".

De no saber nada concreto sobre microcontroladores, a base de trabajo y de buscar información he sido capaz de programar uno, además de ser capaz de introducir un SO de carácter preemtivo para que realice tareas concurrentes.

Por otra parte me ha dado a conocer el mundo del hardware libre con Arduino, ya que también desconocía este entorno y programar en un lenguaje de programación de muy alto nivel, wiring.

En un principio para familiarizarme con el entorno de programación, hice uso de una gran variedad de códigos de ejemplo. Además traté de separar en pequeños programas todas las tareas del código final, para comprobar que independientemente unas de otras, funcionaban. Posteriormente solo tuve que juntar todas estas tareas y adaptarlas en un mismo programa. Lo que resultó relativamente sencillo ya que al ser diversas tareas, hubo que adaptar pines y parte del código para que no ocurrieran fallos. De hecho hasta llegar a la versión final del código, había veces en el que el microcontrolador se quedaba colgado, por ejemplo al no poner un delay antes enviar/recibir datos por serie, ya que se recomienda si se hace uso del DuinOS, hacer este delay para evitar interbloqueos.

A pesar de plantearse este proyecto con cierta dificultad al principio, ya que prácticamente no conocía nada de lo relacionado con microcontroladores, a medida que he estado trabajando en él y se han solucionado problemas, que mayormente surgían por la falta de conocimientos, he visto que una de las principales características de haber trabajado con Arduino es la sencillez y la facilidad de encontrar material muy diverso por Internet. Al tratarse de hardware libre, todo el código es open-source y está enfocado en su mayor parte a poder concentrarte en qué quieres hacer y no en cómo conseguirlo.

La realización de este proyecto ha conseguido que tenga un interés en todo este mundo de microcontroladores, ya que son piezas muy asequibles y con una gran versatilidad, con las que se pueden hacer una infinidad de instrumentos, desde un termómetro, una matriz de leds, hacer uso de displays lcd, hasta este proyecto.

7. AGRADECIMIENTOS:

Quiero agradecer la ayuda recibida por parte del Director del Proyecto D. José Vicente Busquets Mataix, por la ayuda recibida y el suministro del material necesario, así como el apoyo de mis compañeros que me han ayudado en la búsqueda de información y conocimientos para todo este entorno y el proyecto en general. También quiero agradecer a mis familiares la paciencia que han tenido conmigo durante toda esta carrera.

8. BIBLIOGRAFÍA:

http://www.atmel.info/dyn/resources/prod_documents/8271S.pdf

http://es.wikipedia.org/wiki/AVR

http://www.arduino.cc/es/

http://es.wikipedia.org/wiki/Arduino

http://www.multiplo.org/duinos/wiki/index.php?title=Main_Page

http://blog.bricogeek.com/noticias/arduino/duinos-sistema-operativo-multitarea-para-arduino/

http://www.henningkarlsen.com/electronics/a_l_ds1302.php

9. ANEXO I:

CÓDIGO IMPLEMENTADO:

```
#include <DS1302.h>



//VARIABLES PARA LECTURA DE INFORMACION
const int buttonPin = 5;// pin pulsador
int estadoBoton;//variable que cambiara segun el pulsador
boolean activado = false;//comprobara que se ha pulsado el boton
const int sensorPin = A0;//pin para la entrada analogica del microfono
int sensorValue = 0;// variable para almacenar el valor de entrada analogica
int oldSensorValue = sensorValue;//variable utilizada para comparar
                        //el valor de entrada anterior con el actual



//VARIABLES PARA EL SONIDO
int cont = 0;
const int speaker=7;//pin que emitira señal de alarma
const int minimo = 620; //periodo microseg para 1'6kHz
const int maximo = 50; //periodo microseg para 20kHz
int valor = minimo;//variable utilizada para que el sonido emitido varie de tonalidad



//VARIABLES PARA LA ACTUALIZACION Y COMUNICACION CON EL RTC
DS1302 rtc(2, 3, 4);
// DS1302:  CE pin    -> Arduino Digital 2
//          I/O pin   -> Arduino Digital 3
//          SCLK pin  -> Arduino Digital 4
int sensibilidadManana = 40; //...
int sensibilidadTarde = 60; //...----->sensibilidades segun parte del dia
int sensibilidadNoche = 20; //...
int sensibilidad = 90; //inicialmente, se actualiza de inmediato al consultar el rtc
```

```
boolean sensman = false;//indicara si se ha cambiado manualmente la sensibilidad
char* hhmm;//utilizado en la tarea de consultar al rtc
int finmanana =12;//hora en la que acaba la mañana
int fintarde=20;//hora en la que acaba la tarde
int finnoche=06;//hora en la que acaba la noche
int nalarmas = -1;//Inicialmente a -1 porque al encenderse el arduino, salta el sonido
                  //contador de alarma


//DECLARACION DE TAREAS
declareTaskLoop(lectura);
declareTaskLoop(consultarRTC);
declareTaskLoop(alarma);
declareTaskLoop(comunicacionSerie);


/**METODO OBTENER SENSIBILIDAD SEGUN FRANJA HORARIA
//metodo encargado de obtener la franja horaria en la que nos encontramos
//y modificar la sensibilidad segun dicha franja
//recibe un char* con la hora del rtc en formato hh:mm
//devuelve la sensibilidad con la que se debe comprobar la entrada del microfono*/
int encuentraRango(char* hora){
 if(!sensman){
   char aux[2]={0, 0};
   int horaint = 0;
   for(int i=0;i<2;i++){//solo los 2 primeros componentes del string -> hh:mm queremos hh
     aux[i]=hora[i];
   }
   horaint = atoi(aux);
   if(horaint >= finnoche && horaint < finmanana){return sensibilidadManana;}
   if(horaint >= finmanana && horaint < fintarde){return sensibilidadTarde;}
   if(horaint >= fintarde && horaint < finnoche){return sensibilidadNoche;}
```

```
 }
 return sensibilidad;
}


/**TAREA LECTURA DE DATOS
//tarea encargada de leer tanto el microfono como el pulsador
//si se pulsa el boton se debera activar el sonido
//al igual que si se supera la sensibilidad limite*/
taskLoop(lectura){
 estadoBoton = digitalRead(buttonPin);
 oldSensorValue = sensorValue;
 sensorValue = analogRead(sensorPin);
 sensorValue=map(sensorValue, 0, 1023, 0, 100);
 if(estadoBoton == HIGH){
  if(!activado) activado = true;
 }

 if(sensorValue - oldSensorValue > sensibilidad || activado){
  resumeTask(alarma);
 }

 resumeTask(consultarRTC);
}


/**TAREA ALARMA
//tarea encargada en hacer saltar el sonido
//esta se activara y recorrera una franja de
//frecuencias molestas durante un tiempo determinado por
//la variable interna cont*/
taskLoop(alarma){
```

```
 do{
  for(int i=0;i<5;i++){
    digitalWrite(speaker,HIGH);
    delayMicroseconds(valor/2);

    digitalWrite(speaker,LOW);
    delayMicroseconds(valor/2);
  }
  valor = valor - 2;
  if(valor==maximo+2){valor=minimo;}
  cont++;
 }while(valor>=maximo && cont < 1000);
 nalarmas++;
 valor = minimo;
 cont=0;
 if(activado){
  estadoBoton = digitalRead(buttonPin);
  if(activado) activado = false;
 }
 suspend();
}


/**TAREA CONSULTAR RTC
//tarea encargada de consultar el rtc
//y obtener la hora, ademas de actualizar
//la sensibilidad llamando al metodo
//encuentraRango()*/
taskLoop(consultarRTC){
 delay(1000);
 hhmm = rtc.getTimeStr();
 sensibilidad = encuentraRango(hhmm);
```

```
}


/**TAREA COMUNICACION
//Se encarga, segun los comandos de 4 caracteres propuestos,
//de modificar la sensibilidad del microfono
//o de solicitar informacion sobre la configuracion
//de la sensibilidad actual y el numero de veces
//que ha saltado el sonido, ademas de poder consultar la
//hora*/
taskLoop(comunicacionSerie){
 Serial.flush();
 char input[5];
 char sens[5];
 char talarm[5];
 char modsens[4] = {'m', 'o', 'd', 's'};
 char infor[4] = {'i', 'n', 'f', 'o'};
 char tiempo[4] = {'h', 'o', 'r', 'a'};
 memset(input,'\0',5);
 byte inByte = '\0';
 boolean modifisens = false;
 boolean inform = false;
 boolean tmp = false;
 int nuevasens = 0;
 char nsens[3];
 memset(nsens,'\0',3);

 while(inByte != '!'){inByte = Serial.read();}

 if(inByte == '!'){
  while(Serial.available() < 4) {;}
  for (int i=0; i < 4; i++) {
```

```
    input[i] = Serial.read();
  }

  delay(200);
  Serial.print("Recibo: ");
  Serial.println(input);

  if(input[0] == modsens[0] && input[1] == modsens[1] && input[2] == modsens[2] && input[3] == modsens[3]) modifisens = true;
  else if(input[0] == infor[0] && input[1] == infor[1] && input[2] == infor[2] && input[3] == infor[3]) inform = true;
  else if(input[0] == tiempo[0] && input[1] == tiempo[1] && input[2] == tiempo[2] && input[3] == tiempo[3]) tmp = true;
  else{delay(200); Serial.println("Orden no existente");}

  if(modifisens){
    while(Serial.available() < 2) {;}
    for (int i=0; i < 2; i++) {
      nsens[i] = Serial.read();
    }
    nuevasens = atoi(nsens);
    if(nuevasens != 0){sensman = true; sensibilidad = nuevasens;}
    else sensman = false;
    delay(200);
    Serial.print("Nuevo valor para sensibilidad(si 00->valor segun franja horaria): ");
    Serial.println(nsens);
    if(nuevasens != 0)sensman = true;
    else sensman = false;
  }

  if(inform){
    delay(100);
    Serial.print("Numero de veces que salta la alarma: ");
    itoa(nalarmas, talarm, 10);
```

```
    Serial.println(talarm);
    delay(100);
    Serial.print("Sensibilidad del microfono (0 mucha - 100 poca): ");
    itoa(sensibilidad, sens, 10);
    Serial.println(sens);
   }

   if(tmp){
    delay(100);
    Serial.println(hhmm);
   }
  }
}


/**SETUP
//aqui se inicializaran las tareas indicando su prioridad, determinar
//pines de entrada y salida, inicializacion del puerto serie
//ademas de programar inicialmente la fecha para el rtc
*/
void setup() {
  pinMode(speaker, OUTPUT);
  pinMode(buttonPin, INPUT);

  createTaskLoop(lectura, NORMAL_PRIORITY);
  createTaskLoop(consultarRTC, NORMAL_PRIORITY);
  createTaskLoop(alarma, HIGH_PRIORITY);
  createTaskLoop(comunicacionSerie, NORMAL_PRIORITY);

  Serial.begin(9600);

  // poner reloj en run-mode y deshabilitar proteccion de escritura
```

```
  rtc.halt(false);
  rtc.writeProtect(false);
  //Para sobreescribir valores del rtc, descomentar las lineas de abajo
  //rtc.setDOW(THURSDAY);       // Set Day-of-Week to FRIDAY
  //rtc.setTime(18, 10, 0);     // Set the time to 12:00:00 (24hr format)
  //rtc.setDate(9, 6, 2011);   // Set the date to August 6th, 2010
}


/**TAREA LOOP
//tarea propia del entorno arduino
//por defecto sera de prioridad baja
//se debe poner nextTask() para evitar
//interbloqueos
*/
void loop() {
  nextTask();
}
```