El paso del tiempo

por Aurelio Gallardo

29 de Enero de 2013

Construyendo un reloj digital con ARDUINO...

Índice General

Apartado I

Introducción

El tiempo es la variable física más escurridiza y enigmática de todas las magnitudes que pueden medirse. Paradójicamente, es una de las más controlables y reproducibles. Es la menos tangible por los sentidos y, a la vez, de la que mantenemos una mayor percepción.

No puede, además, definirse. Nadie sabe qué es el tiempo. Tenemos que definirlo a través de sus características. Por ejemplo, el tiempo es continuo creciente, ya que fijada una referencia, toma todos los valores numéricos reales posibles, por lo tanto un escalar.

El tiempo es irreversible, asimétrico. Ningún suceso puede contradecir los principios de causa-efecto. Esto quiere decir que una taza siempre que caiga se romperá y no sucede que exista un devenir en el que los trozos de la taza se unan. Así, aunque la existencia de un tiempo negativo no está prohibido *de facto* por ninguna ley física, condiciona nuestra realidad. Sabemos que el pasado es inmutable, que el futuro es una incógnita lleno de probabilidades y que el presente es la frontera entre ambos: claramente asimétrico. Esta concepción parece estar relacionada con el aumento de entropía del Universo.

El tiempo es homogéneo. No transcurre algunas veces más rápido y otras menos (para el mismo sistema de referencia).

El tiempo es isótropo. En cualquier dirección que nos movamos, transcurre igual.

En Mecánica Clásica, no relativista, el tiempo era absoluto, igual para cualquier observador e independiente del movimiento de éstos. Einstein, sin embargo, en su teoría relativista, baja la categoría de variable independiente a variable dependiente. El transcurso del tiempo para dos observadores que se muevan a velocidades cercanas a la de la luz, uno respecto del otro, difiere. Depende, pues, del sistema de referencia usado, del observador.

Apartado II

La medida del tiempo

Probablemente el ser humano adquiere la conciencia del transcurso del tiempo desde los albores de nuestra raza, ligando el transcurso del mismo a ciclos naturales. No es difícil imaginar a nuestro ancestro en una cueva, haciendo una muesca en un palo cada noche, esperando ver aparecer la luna llena de nuevo; y quizás, marcando en otro palo las apariciones de ésta, predijese cuál es la siguiente época de lluvias, o la luna en la que apareció esa manada de bisontes por el desfiladero.

Me hubiese gustado estar presente la primera vez que clavó un palo vertical en la tierra y observó la sombra del sol y su recorrido. Por el ángulo podía saber el momento del día. Por su longitud variable, la época del año.

Los descubrimientos de otros fenómenos cíclicos, más o menos complicados, dieron como resultado otros tipos de relojes. Desde los relojes de arena a los relojes atómicos. Todos se basan en algún acontecimiento repetitivo, en algún fenómeno oscilatorio: un péndulo, una lámina de cuarzo piezoeléctrico por el que pasa una corriente, la frecuencia de resonancia de los electrones del Cesio... Y contar las veces que se repite.

Contar el tiempo es, pues, una necesidad humana. El conocimiento que permite llegar a diseñar un dispositivo preciso que lo consiga, un reto.

¿Podré construir un reloj con ARDUINO?

Apartado III

Que tendrá ARDUINO para poder hacerme un reloj con él

La cuenta del tiempo...

ARDUINO es una placa controladora, con un microprocesador capaz de ejecutar las órdenes que se almacenen en su memoria. Posee una serie de entradas y salidas, digitales y analógicas, configurables muchas de ellas mediante código.

Una pieza esencial de su electrónica, común a todos los circuitos digitales secuenciales, es el dispositivo que le permite obtener una señal cíclica, repetitiva: la señal de reloj. Su frecuencia 16MHz lo cual significa que puede realizar 16 millones de operaciones básicas por segundo. ¿Podrá contar el tiempo? ¡¡Por supuesto!!

De hecho, en el lenguaje de programación que acompaña a ARDUINO encontramos funciones específicas muy relacionadas con la cuenta del tiempo:

1. **millis()**. Función que cuenta los milisegundos transcurridos desde que se enciende la unidad. Devuelve un número tipo long. La cuenta se desborda y vuelve a cero aproximadamente a los 50 días. [1]
2. **micros()**. Análoga a la anterior, cuenta microsegundos. Se desborda volviendo a cero a los 70 minutos.
3. **delay(n)**. Pausa el programa durante ***n*** milisegundos.
4. **delayMicroseconds(m)**. Pausa el programa durante ***m*** microsegundos. Se recomienda $m > 3\mu s$.

Y generar las señales para activar unos displays...

Como sabemos Arduino UNO tiene una capacidad de 14 entradas o salidas digitales, algunas de las cuales pueden ser moduladas analógicamente, más 6 entradas analógicas. Nuestros displays de 7 segmentos necesitan conectarse a 7 salidas digitales, numeradas de la "a" a la "g", que activen cada led del display más un cátodo común que, conectado a través de una resistencia, lleve la intensidad a tierra.

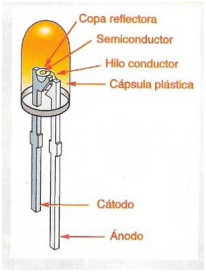

Figura 1: LED. Ánodo y Cátodo

Un reloj que muestre horas y minutos necesitaría cuatro displays de este tipo, con lo cual hablamos de 28 conexiones, el doble de las que puede ofrecer Arduino. Además, conectaríamos a través de una resistencia pequeña (unos $100\Omega - 500\Omega$) cada cátodo a tierra.Sin embargo si los cátodos no son conectados a tierra, sino a Vcc (tensión de alimentación), no circulará intensidad por el diodo y no se iluminarán. Este hecho puedo usarlo para reducir el conexionado de los displays a Arduino ***activando los cuatro displays en secuencia***.

Efectivamente, ***conectando siete salidas digitales de Arduino a los displays, en paralelo, sólo se activará el display que en ese momento tenga el cátodo conectado a tierra***. Así que, además de esas siete salidas digitales, necesito conectar cuatro salidas a cada cátodo de cada display y enviar en secuencia un cero lógico a la salida que quiero activar (podríamos decir que esas cuatro salidas son activas por lógica inversa). *Dando un barrido del primer al cuarto dígito, del primer al cuarto display, si la velocidad de*

[1] si es un unsigned long, se desborda en el número $4,294,967,295 = 2^{32} - 1$. Un simple cálculo nos indica que corresponde a 49,71 días. http://arduino.cc/en/Reference/UnsignedLong

cambio es lo suficientemente lenta para activar los leds del display y lo suficientemente rápida para que el ojo humano no distinga el parpadeo podremos ofrecer los cuatros dígitos a la vez.

Y habremos usado, en total, 11 salidas digitales de las 14 disponibles. En el montaje que sigue, una de las tres sobrantes se usará como medida de los segundos, activando el típico parpadeo de los dos puntos luminosos que separan la hora de los minutos y otra como salida hacia un buzzer para hacer sonar la alarma.

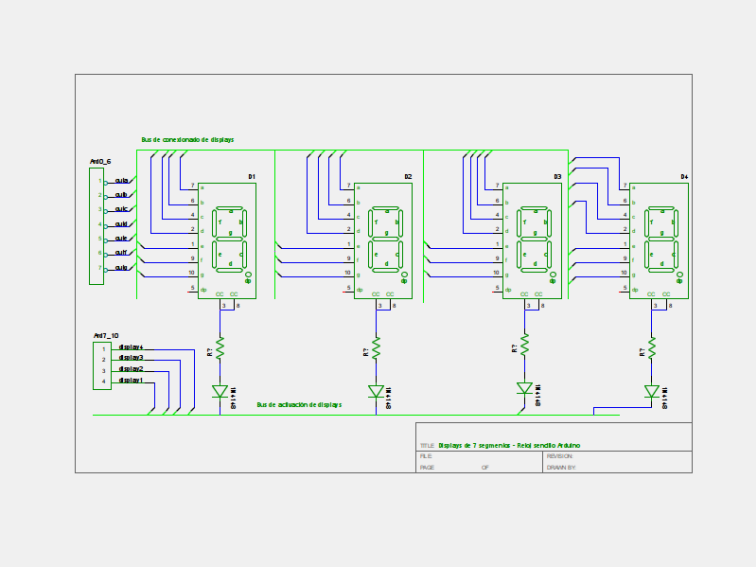

Figura 2: Esquemático de 4 displays

Y detectar ciertas señales analógicas...

Para poder tener un reloj es necesario ponerlo en hora. Cuanto menos, necesito dos pulsadores para ello: uno que haga de horaria y pueda ajustarme la hora, y otro que haga de minutero y pueda ajustarme los minutos. Sin embargo, me hacen falta entradas digitales: de antes, sólo me sobraba una. Aunque sí que dispongo aún de seis entradas analógicas.

La solución es utilizar tres entradas analógicas como entradas digitales. No, ARDUINO no las lee como las digitales. En las entradas analógicas lee niveles de tensión convertidas a un rango numérico entre 0 y 1023. Pero claramente un 0 lógico perfectamente está en el umbral inferior a la mitad y un 1 lógico en el umbral superior. Detectar esta señal “digital” es tan fácil como discernir si el rango de entrada es mayor o menor que un valor intermedio (por ejemplo, 500).

Me hará falta, si además quiero una alarma, una entrada suplementaria. Combinada con las otras dos podré poner la hora y los minutos de la alarma, y activarla o desactivarla.

Apartado IV

Montaje del reloj digital

Display comercial ATA3492BR-1

Entrando en la web comercial http://www.cooking-hacks.com/, se pueden encontrar varios tipos de displays de 7 segmentos (enlace). De un precio más que asequible disponemos de un set de 4 displays

formando el display de un reloj digital, y, además en varios colores: rojo, verde, azul, amarillo... Las conexiones de la "a" a la "g" que activa cada segmento están ya en paralelo (con la salvedad que son de ÁNODO COMÚN) y sólo necesito activar los cátodos correspondientes. El resto de pines se usan para activar los dos puntos luminosos centrales L1 y L2 (segundos), el punto decimal D1 y el punto superior L3.

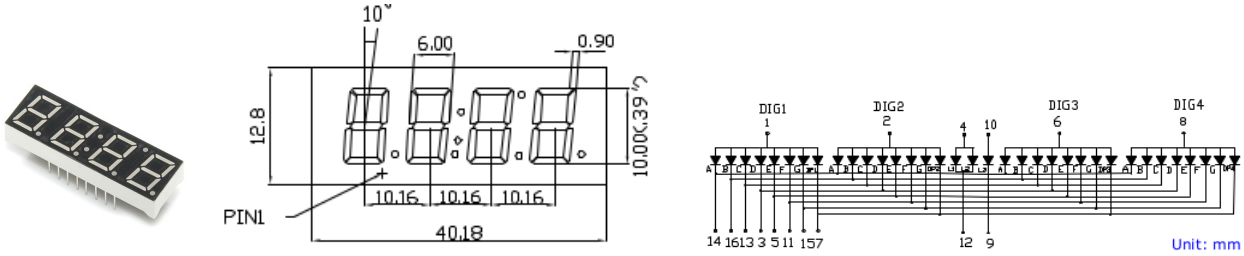

Figura 3: Set de displays 7-segmentos. Esquema.

Las especificaciones técnicas de este dispositivo pueden encontrarse en el siguiente datasheet: especificaciones técnicas display. Destacamos que posee 16 pines de conexión, y el conteo se realiza de la patilla 1 (esquina inferior izquierda) dando la vuelta de izquierda a derecha, y arriba de derecha a izquierda hasta el pin 16 (esquina superior izquierda).

Usamos el siguiente conexionado a la placa ARDUINO UNO.

Podemos observar que los cuatro ánodos se unen cada uno a una patilla de la placa ARDUINO. En concreto los comunes COM1, COM2, COM3 y COM4 se activan con las patillas 7, 8, 9 y 10. El dígito visualizado es el que se activa cuando su ÁNODO está conectado a 5 V, teniendo que estar el resto de los ÁNODOS conectados a 0 V (para que no circule intensidad por los leds). Podríamos decir entonces que cada dígito se activa por LÓGICA POSITIVA.

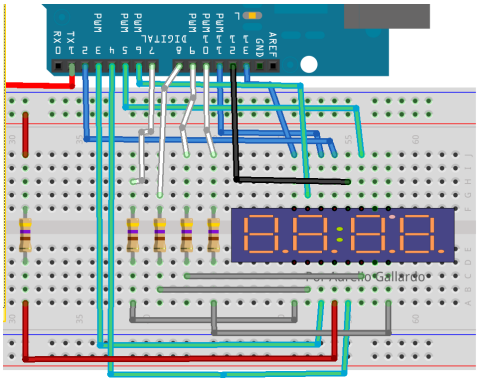

Figura 4: Conexionado de la placa arduino

Los comunes tienen una resistencia que limita la intensidad que circula por los LEDS de 470 Ohmios. Puede aumentarse la luminosidad con reducir el valor de resistencia hasta un mínimo de unos 100 Ohmios . [2]

El resto de las salidas digitales activan:

- Los segmentos de los displays: [a,11] - [b,13] - [c,2] - [d,3] - [e,4] - [f,5] -[g,6]. Por LÓGICA NEGATIVA, o sea, poniendo un "0" lógico a la salida de los pines que corresponden a los segmentos que quiero visualizar, para que circule por ellos la corriente desde su ánodo.
- Los LEDS centrales L1 y L2 del segundero. La salida digital 12 es elegida para tal fin.
- La salida 1 se elige para activar un buzzer de alarma.

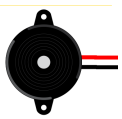

Figura 5: Buzzer de alarma

[2] Cálculos más precisos necesitan estudiar los datos técnicos de cada LED, su tensión umbral y la intensidad máxima soportada (Página WEB)

Pulsadores

Usamos los pulsadores para enviar señales de entrada hacia nuestro ARDUINO. Como no podemos usar más entradas digitales, optamos por las entradas analógicas que ofrece el dispositivo.

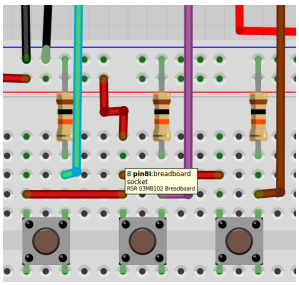

Figura 6: Pulsadores

Los pulsadores, de izquierda a derecha, corresponden a las funciones de cambio de hora, de cambio de los minutos, y de cambio de la hora-minutos de la alarma, o su desactivación-activación.

Cada pulsador, en estado OFF, o abierto, conecta a tierra la señal analógica correspondiente a través de una resistencia de $10K\Omega$. Con ello, evitamos cualquier lectura espuria en la entrada y aseguramos que cualquier pequeña carga acumulada en el pin en cuestión tenga un camino a tierra. Mantenemos un "0" lógico en la entrada analógica.

Al pulsar, estado ON, o cerrado, conectamos la entrada analógica a 5V. Ponemos un "1" lógico en dicha entrada.

¿Qué ocurre realmente, si no son entradas digitales, sino analógicas? El conversor analógico-digital de ARDUINO convierte los valores de tensión ofrecidos en estas entradas a un rango numérico de 1024 valores. Teóricamente, 0 Voltios (tierra) de tensión los traduce al valor numérico 0; 5 Voltios (Vcc) al valor 1023. Un valor intermedio, por ejemplo, 1.9 Voltios lo traduce a:

$$1,9\,V \cdot \frac{1023}{5\,V} \simeq 389$$

Resumiendo: interpretar a la entrada de las señales analógicas un 0 o un 1 lógico es tan sencillo como discernir si estamos por debajo o por encima de un valor intermedio. Elegí el número 500 por ser un número redondo próximo a la mitad exacta (512 -> 2.5 Voltios) y, como veremos, detectaré un "0" lógico si el valor de la entrada analógica está por debajo de 500 y un "1" lógico si por encima.

Montaje Final

Este es el esquema del montaje final (Fritzing).

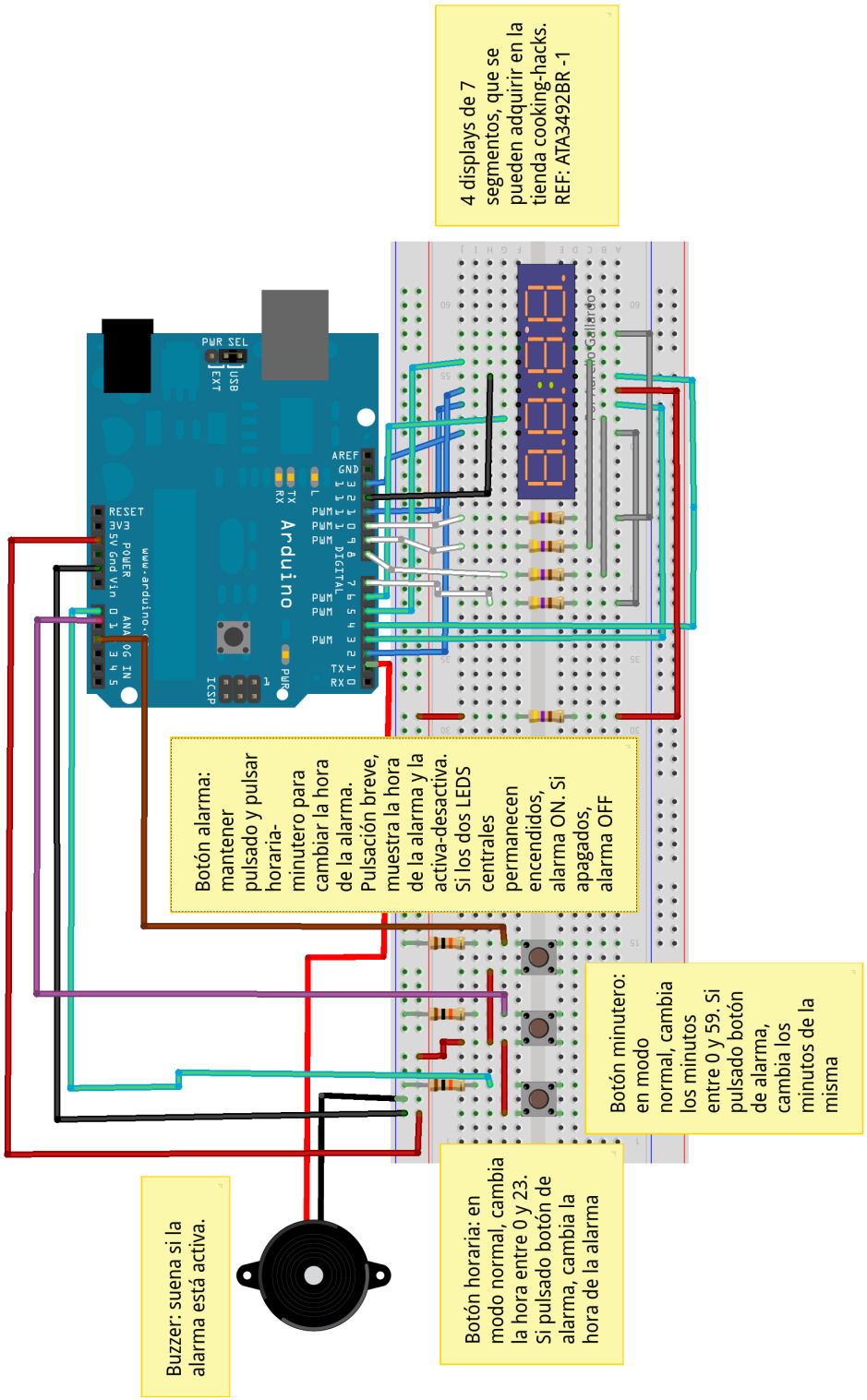

Figura 7: Montaje Final (Fritzing)

¿A que no es complicado? No, si lo bueno viene ahora que tenemos que programar...

Apartado V

¿Cómo programo mi reloj? Ideas fracasadas e ideas correctas

No es de extrañar que una primera idea resulte incorrecta, por muy atractiva que pueda parecernos a priori. Es lo que me ha pasado a mí estudiando este proyecto. Mi primera idea de diseño del programa consistía en:

1. El bucle principal del programa **void loop() { }** lo basaría en:
   a) Un contador recursivo y un retraso de 1s - *o delay(1000)* -.
   b) Órdenes y llamadas a funciones que calculan:
      1) La hora (unidad y decenas) y los minutos (unidad y decenas), en base a ese contador recursivo.
      2) las diferentes salidas digitales que permiten visualizar los diversos números en el display.
2. El convencimiento de que el tiempo de ejecución de un ciclo de la rutina ***loop*** podía ajustarlo a 1 segundo con la suficiente precisión como para usarlo de base de cuenta de tiempo del programa...
   a) Ya que suponía que el retraso provocado por el delay era muy preciso, de exactamente 1 segundo: $T = 1\,s$.
   b) Y que el resto de las funciones de visualización o cálculo eran muy rápidas en su ejecución y no provocarían un retardo mucho mayor. Si suponía que el retardo provocado por estas funciones era t_0, pensaba que $t_0 \ll T$. Los errores arrastrados en la ejecución de estas órdenes no retrasarían mucho el reloj.

Sin embargo, y como pude comprobar experimentalmente, el tiempo que tardan las órdenes en ejecutarse sí que son relevantes. Incluso retrasos menores como incluso de 1 ms - *delay(1)* -, no permitían evitar que los errores se acumulasen al poco de funcionar el reloj[3]. La otra orden que podía usar, **delayMicroseconds()**, quizás me sirviese y reduje con ella aún más el tiempo de espera; pero seguían acumulándose retrasos. Por lo que deduje que las órdenes de cálculo y visualización cumplían que $t_0 \sim 1\,s$.

No me servía esta idea para construir un reloj.

Sin embargo, aún me quedaba la orden **millis()**, que cuenta los milisegundos transcurridos desde que se enciende la unidad. La variable long asignada a millis() abarca un período de 50 días aproximadamente. ¿Serviría para llevar la cuenta del tiempo? Debían cumplirse dos condiciones básicas:

1. La cuenta de este tiempo es ***independiente*** de las órdenes que se ejecuten. Éstas no deben añadirle retrasos.
2. El programa debe ser capaz de convertir una variable que crece continua y linealmente en cuentas cíclicas.

Evidentemente, ésta ha sido la solución.

[3] Evidentemente, el contador ya no contaba segundos, sino milisegundos.

Apartado VI

Programación

Variables

Variables de salida digital

```
int outa = 11;
int outb = 13;
int outc = 2;
int outd = 3;
int oute = 4;
int outf = 5;
int outg = 6;
int outsegundero = 12;
int outalarma=1;
int anodo[4]={7,8,9,10};
```

Cada variable define el pin que activa cada tramo de los displays, los leds de los segundos y la salida que controla el buzzer de alarma. También inicializo un array en el que asocio cada ánodo a un pin de activación.

Variables de entrada analógica

```
int inhoraria = 0;
int inminutero = 1;
int inalarma = 2;
```

Define los pines de entrada analógica. El pin 0 para la horaria, el 1 para el minutero y el 2 para las funciones de alarma.

Variables cálculos temporales

```
unsigned long hora = 0; // de 0 a 23.
unsigned long minuto = 0; // de 0 a 59
unsigned long hora0 = 10; // de 0 a 23.
unsigned long minuto0 = 10; // de 0 a 59
unsigned long segundo0 = 0; // de 0 a 59
unsigned long segundo = 0; // de 0 a 59
unsigned long segundostranscurridos = 0;
unsigned long mediosegundo = 0;
```

1. Las variables hora, minuto y segundo almacenan los valores correspondientes al momento temporal visualizado en el display.
2. Las variables hora0, minuto0 y segundo0 almacenan la hora de desfase. Añadiendo un desfase a las variables hora-minuto-segundo podré poner en hora al reloj. El reloj inicialmente está ajustado a las 10 y 10 (como los relojes de manecillas).
3. La variable **segundostranscurridos** toma valores cíclicos entre 0 y 86399 segundos, según los valores de la variable **tiempo1** (definida posteriormente) que cuenta los milisegundos transcurridos desde que se enciende la unidad. A la variable **segundostranscurridos** se le añade los segundos de desfase.
4. La variable mediosegundo toma valores 0 ó 1 cada medio segundo, lo cual permite activar-desactivar los leds centrales parpadeantes, y es aprovechada para generar un tono variable en la alarma.

Variables de alarma

unsigned long horaalarma=10; int alarma=0;
unsigned long minutoalarma=5;

1. **horaalarma** y **minutoalarma** (de inicio a las 10:05) permiten almacenar la hora en la que se activará la alarma si...
2. ...la activamos con la variable alarma (0-> desactivada y 1-> activada).

Variables de los displays, de tiempo y contador

unsigned long displays[4]; unsigned long tiempo1=0; int i=0;
int cualactivo = 0 ; int retrasoboton = 100;

1. El array **displays** almacena el dígito que tiene que mostrar cada display individual del set de 4.
   a) displays[0] corresponde al valor más a la izquierda, la decena de las horas; displays[1] a las unidades de las horas.
   b) displays[2] y displays[3] lo mismo pero con los minutos.
2. La variable **cualactivo** recorre cíclicamente los valores de 0 a 3, en cada ciclo de loop(), y activa el display correspondiente al valor que toma. El ciclo es tan rápido que el ojo humano no puede distinguir cuál está encendido y el efecto es el de mostrar los 4 dígitos a la vez, sin parpadeos.
3. La variable **tiempo1** almacena el valor de la función *millis()*.
4. Los botones de entrada pueden detectar más de una señal encendido-apagado al ser pulsados. Para poder asegurar que han sido pulsados de forma correcta se comprueba el estado del botón en un período de tiempo que puede ajustarse mediante la variable retrasoboton (en milisegundos).
5. La variable i es un contador de propósito general.

Funciones y subrutinas

Mostrar un valor en uno de los cuatro displays

```
void mandasenal(int numdisplay, int valor) {
// ****************************************
// a) activa cada tramo del display segun valor
// ****************************************
// recibe el numero de display y el valor que tiene que entregar
switch (valor) {
case 0:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,LOW);
digitalWrite(outf,LOW);
digitalWrite(outg,HIGH);
break;
...
...
```

```
...
case 10:
digitalWrite(outa,HIGH);
digitalWrite(outb,HIGH);
digitalWrite(outc,HIGH);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,HIGH);
digitalWrite(outg,LOW);
break;
}
// ********************************
// b) activo un solo display cada vez
// ********************************
for (int i=0;i<=3;i++) { // recorrido de 0 hasta 3
if (i==numdisplay) {
digitalWrite(anodo[i],HIGH); // activo
} else {
digitalWrite(anodo[i],LOW); // desactivo
} // fin del if
} // fin del for
}
```

Esta función tiene dos partes bien diferenciadas:

1. En la primera, y según lo que valga la variable *valor*, se activan (a nivel bajo, por LÓGICA NEGATIVA), los segmentos de los displays: los números del 0 al 9, y si $valor = 10$, un guión.
2. En la segunda parte, y según lo que valga la variable numdisplay, activo el ánodo que le corresponde, de manera que el valor del dígito se aplica sólo a él.

La rutina "valordisplay"

```
void valordisplay(unsigned long h,unsigned long m,int rayah, int rayam) {
if (rayah==LOW) {
displays[0]= h/10;
displays[1]= h %10; // resto de la division entre 10
} else {
displays[0]= 10;
displays[1]= 10; // guion
}
if (rayam==LOW) {
displays[2]= m / 10;
displays[3]= m %10; // resto de la division entre 10
} else {
displays[2]= 10;
displays[3]= 10; // guion
}
}
```

La rutina *valordisplay* convierte el valor que toma la variable hora (*h*) y la variable minuto (*m*) en los cuatro números que hay que aplicar a cada display. Se aprovecha para aplicar un guión (valor de display "10") si se activan a nivel alto las variables *rayah* o *rayam*. Se usará en el cambio de hora o en el cambio de minuto para, de alguna manera, mostrar sólo lo que se está cambiando.

La rutina que calcula el tiempo

```
void calculamuestratiempo() {
// 1) calculos para el reloj
segundostranscurridos = hora0*3600+minuto0*60+segundo0+((tiempo1/1000) %86400);
segundo= segundostranscurridos %60;
minuto = segundostranscurridos/60;
hora = minuto/60;
hora = hora %24;
minuto = minuto %60;
}
```

Constituye uno de los núcleos centrales de la programación:

1. La variable **segundostranscurridos** convierte el valor de **tiempo1** en el segundo del día en el que me encuentro. Cuando acaba su cuenta vuelve a cero. Poner el reloj en hora es añadir un desfase a esta variable, a través de las variables hora0, minuto0 y segundo0.
2. Los segundos cuentan de 60 en 60. Si la variable **segundostranscurridos** supera los 60 segundos, ya se ha superado un minuto. Los minutos transcurridos se obtienen dividiendo la variable entre 60. Los segundos que pasan de esos minutos calculando su resto (operación %).
3. De igual forma, las horas transcurridas se obtienen dividiendo entre 60 los minutos, y los minutos que pasan de esas horas calculando su resto al dividir entre 60.
4. Para las horas, cuando se desborda la cuenta al llegar a 24, hay que obtener el resto de la división por ese número.

Bucle setup

Simplemente activa los pines digitales como salidas, según las variables que elegí al principio.

Bucle loop

La primera instrucción es **tiempo1=millis();** asigno a la variable **tiempo1** el conteo de milisegundos transcurridos desde que se encendió la unidad.

Compruebo el botón horaria

```
if (analogRead(inhoraria)>500 && analogRead(inalarma)<500) { //compara
delay(retrasoboton);
if (analogRead(inhoraria)>500) {
hora0=(hora0+1)*(hora0<23);
calculamuestratiempo();
valordisplay(hora,0,LOW,HIGH);
for(i=0;i<=300;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
}
}
```

1. El primer condicional comprueba si la lectura analógica del botón de horaria está en estado “1” (recordemos: valor analógico mayor que el umbral), espera un poco, y vuelve a comprobarlo. Si sigue pulsado, se activan el resto de órdenes. Así, una pulsación leve o una falsa lectura no activa el cambio de hora. En el primer condicional, se comprueba además que no se haya pulsado el botón de alarma, porque si se hace, lo que cambia no es la hora sino la hora de la alarma.

2. La pulsación añade un hora más al desfase (con desbordamiento). Si hora0>=23 al pulsar vuelve a cero.
3. Se calcula el tiempo, y por tanto la variable hora llamando a calculamuestratiempo.
4. En los displays se visualiza la hora y un guión en los minutos (último parámetro de valordisplay a HIGH).
5. Durante aproximadamente unos 300 ms se muestra lo que hay en el array displays. Así la nueva hora y un guión aparecen brevemente en el reloj.

Compruebo el botón minutero

```
if (analogRead(inminutero)>500 && analogRead(inalarma)<500) { //compara
delay(retrasoboton);
if (analogRead(inminutero)>500) {
minuto0=(minuto0+1)*(minuto0<59);
calculamuestratiempo();
valordisplay(0,minuto,HIGH,LOW);
for(i=0;i<=300;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
}
}
```

Idéntico al bloque anterior, se opera ahora con la variable minutos. Es la única diferencia.

Compruebo el botón alarma

```
visualizo:
if (analogRead(inalarma)>500) { //compara
delay(retrasoboton);
if (analogRead(inalarma)>500 && analogRead(inhoraria)<500 && analogRead(inminutero)
    <500) {
alarma=!alarma;
valordisplay(horaalarma,minutoalarma,LOW,LOW);
digitalWrite(outsegundero,!alarma);
for(i=0;i<=2000;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
} else if(analogRead(inalarma)>500 && analogRead(inhoraria)>500 && analogRead(
    inminutero)<500) { // caso que pulse las dos a la vez
horaalarma=(horaalarma+1)*(horaalarma<23);
valordisplay(horaalarma,minutoalarma,LOW,LOW);
for(i=0;i<=500;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
goto visualizo;
} else if(analogRead(inalarma)>500 && analogRead(inhoraria)<500 && analogRead(
    inminutero)>500) {
minutoalarma=(minutoalarma+1)*(minutoalarma<59);
valordisplay(horaalarma,minutoalarma,LOW,LOW);
for(i=0;i<=500;i++) {
mandasenal(i%4,displays[i%4]);
```

```
delay(1);
}
goto visualizo;
} // fin del if secundario
} // fin del if principal
```

1. El primer ***if*** comprueba si se ha pulsado el botón de alarma, y sólo ese, no en combinación con los otros. Modifica la variable alarma (o 0 ó 1, desactivada o activada), apaga o enciende los leds centrales del parpadeo cada segundo y muestra la hora de alarma durante unos dos segundos. Si los leds centrales aparecen apagados, la alarma está desconectada. Si aparecen encendidos, conectada.
2. El segundo condicional ***if*** comprueba si se han pulsado los botones de alarma y hora a la vez. En este caso, se cambia y se muestra la hora de la alarma.
3. Lo mismo para el tercer condicional ***if***, pero en este caso para los minutos de la hora de alarma.

Desbordamiento de la variable tiempo1

```
if (tiempo1<=10 && segundostranscurridos>=1) {
hora0=hora;
minuto0=minuto;
segundo0=segundo;
}
```

¿Qué ocurre cuando la variable **tiempo1** se desborde, al cabo de los 50 días? Si fuese múltiplo de 86400 no pasaría nada, porque la cuenta cíclica de los segundos y de la variable **tiempo1** estarían sincronizadas... Pero no es así. En este caso puede producirse un desfase que desajustaría la hora del reloj.

Para evitarlo, este condicional ***if***. No está probado, pues no he tenido paciencia de esperar a ver qué pasa a los cincuenta días. He razonado lo siguiente:

1. Cuando la variable **tiempo1** es muy pequeña y **segundostranscurridos** es mayor que 1 (mayor de 1000 ms), evidentemente o hemos puesto ya el reloj en hora e introducido un desfase, o se ha alcanzado el desbordamiento de **tiempo1**, o ambas cosas.
2. Como todavía no se ha vuelto a calcular la hora de visualización en el bucle (la orden *calculamuestratiempo* es posterior), si fuerzo a que el desfase sea la hora actual, al ser **tiempo1** muy pequeño, la variable **segundostranscurridos** se ajustaría bien ya que:
   a) segundostranscurridos = hora0*3600+minuto0*60+segundo0+((tiempo1/1000)% 86400);
   b) ((tiempo1/1000)% 86400)=0, si tiempo1<=10
   c) y **segundostranscurridos** correspondería a los segundos transcurridos simplemente por el desfase.
3. También se ha supuesto que la variable **tiempo1** se refresca con una cadencia inferior a 10ms. Si el código no funcionase, se podría intentar variar la condición para **tiempo1**; si **tiempo1**<=n, siendo n<1000. Por ejemplo, con n=100, n=500, n=900...

Cálculo y muestra del segundero

```
//a) segundero
mediosegundo = ((tiempo1/500) %2);
digitalWrite(outsegundero,mediosegundo);
```

Calculo un 0 o un 1 para la variable mediosegundo, y pongo ese valor en los leds centrales. Poco más que explicar.

Mostrando la hora

```
// b) calculo el tiempo
calculamuestratiempo();
valordisplay(hora,minuto,LOW,LOW);
//c) visualizo
cualactivo=(cualactivo+1)*(cualactivo<3);
mandasenal(cualactivo,displays[cualactivo]);
```

1. Vuelve a calcular el tiempo con la función calculamuestratiempo.
2. Refresca los valores del array **displays** con la llamada a la rutina valordisplay.
3. La variable **cualactivo**, en cada ejecución del loop, va tomando valores entre 0 y 3.
4. Con la función mandasenal, activo uno de los cuatro displays (el que indique **cualactivo**) y le paso el valor del número que tiene que mostrar.

Evidentemente, es otro de los núcleos centrales del programa. Las llamadas a las funciones permiten que en un ciclo de loop se elija uno de los cuatro displays y se visualice el valor que le corresponda, pero antes necesito que se refresquen las variables que cuentan el tiempo y éstas a su vez refresquen los valores del array **displays**. Es instructivo que, una vez que tengamos el reloj funcionando, hagamos algunas modificaciones que nos permitan comprobar su forma de operar. Por ejemplo:

- Cambiar cualactivo=(cualactivo+1)*(cualactivo<***3***) por cualactivo=(cualactivo+1)*(cualactivo<***2***). Así nunca veremos el último dígito.
- Añadir la orden ***delay(500);*** al final de este bloque de órdenes. Veremos como se suceden las activaciones de los 4 displays lentamente. Reducir ese valor poco a poco, y se irán activando cada vez más rápido hasta que no pueda distinguirse el cambio.

Activando la alarma

```
// d) activo alarma
if (horaalarma==hora && minutoalarma==minuto && alarma==1) { // control de la alarma
    . En la hora, en el minuto y si alarma esta activado
tone(outalarma,100*mediosegundo*2,100);
} else {
noTone(outalarma);
}
}
// Fin del programa
```

El condicional ***if*** comprueba, simplemente, si la hora actual coincide con los de la variable alarma y, si esta está activada, toca un tono que, dependiendo si mediosegundo es 0 ó 1 lo conmuta entre los 100 y 300 Hz. Si no estamos en hora de alarma, desactiva el tono.

Apartado VII

Fotografías

Video en youtube: http://youtu.be/vMsYxfWGz1k

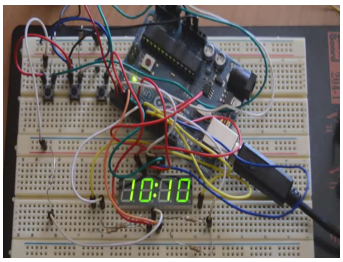

Figura 8: estado inicial del reloj

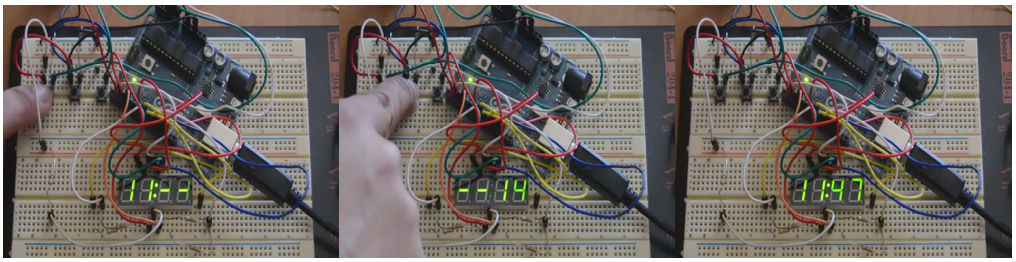

Figura 9: cambiando los dígitos de la hora y de los minutos; reloj en hora

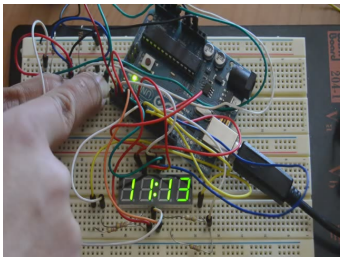

Figura 10: cambiando la hora de la alarma

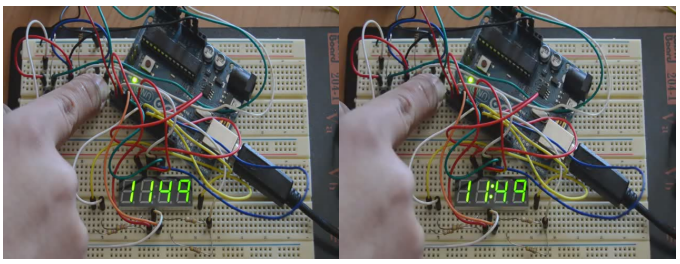

Figura 11: Desactivando y activando la alarma

Apartado VIII

Posibles mejoras

Segundero

Aún quedan tres entradas analógicas por usar. Sería posible, por ejemplo, programar otro botón para que mostrara los segundos que están transcurriendo.

Display con entrada serie

En la página de cooking-hacks encontramos un set de 4 displays de 7 segmentos (http://www.cooking-hacks.com/index.php/7-segment-serial-display-kelly-green.html) con activación serie, de forma que los 13 posibles pines de los que hemos hablado con anterioridad se transforman en tres. La ventaja es evidente: la visualización de la hora deja ahora libre muchos pines digitales para poder usar como salidas o entradas. El handicap radicaría en modificar el programa para usar este dispositivo. Su desventaja: el precio, mucho más caro que el display anterior.

Figura 12: Set de 4 displays de7-segmentos entrada serie.

Calendario

¿Podría ir más allá y configurar un calendario? Creo que sí. Ahora tendría que tener una variable **días**, que contase los días transcurridos desde que encendiese el reloj más un desfase. Necesitaría un botón o dos, para que al pulsar se viese el mes y el día, y en otro el año. Otros botones más para incluir el desfase del día y ajustar la fecha.

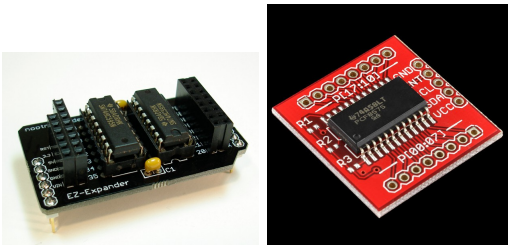

Figura 13: EZ EXPANDER

Además, necesito funciones que transformasen la cuenta de los días al formato día-mes-año. Quizás la función inversa. Otras funciones que pudiesen servir de calendario perpetuo y, al pulsar otro botón, me mostrasen el día de la semana. Hay muchas posibilidades. Evidentemente, si aumentan tanto las posibles entradas de datos, el set de displays en serie es una buena opción para la visualización. Otra posibilidad es estudiar el shield de ARDUINO "EZ-EXPANDER" (placa de expansión EZ-EXPANDER), que convierte mediante registros de desplazamiento tres salidas digitales en dieciséis. También en cooking-hacks encontramos la placa "Breakout Board for PCF8575 I2C Expander" (http://www.cooking-hacks.com/index.php/breakout-board-for-pcf8575-i2c-expander.html).

Melodía

Parece lógico mejorar este aspecto del reloj. La salida de alarma queda algo monótona e incluso molesta... ¿No podría ser que sonase una melodía? Hay un par de programas disponibles en la página de ARDUINO que hablan de este tema:

1. Melody: http://www.arduino.cc/en/Tutorial/Melody
2. PlayMelody: http://www.arduino.cc/en/Tutorial/PlayMelody

Sin embargo, aunque llegué a incorporar dichos programas a la programación del reloj, el resultado no fue satisfactorio. En ambos, se usan rutinas ***delay()*** de forma habitual y eso provoca que el display de la hora tenga retardos y la visualización no sea correcta. No queda bien que cuando suene la alarma los dígitos aparezcan de uno en uno.

Posibles soluciones que incorporen el código de estos dos programas musicales podrían ser la visualización en el display de algún símbolo que recordase que se está tocando una melodía o el uso del display en serie que quizás minimice el efecto del retardo. O estudiar a fondo la cuestión y ver si pueden generarse notas musicales de otras formas, incluso con circuitos externos o complementarios a ARDUINO.

Cronómetro

Sería interesante la modificación del código para conseguir, no un reloj, sino un cronómetro. Quizás con exactamente el mismo hardware. Un proyecto para estudiar.

Apartado IX

Idea de negocio: LilyPad y relojes montados en tela.

Las posibilidades de hardware relacionados con ARDUINO es muy amplia. Sólo hay que echar un vistazo a la página de cooking-hacks. Una de las versiones de ARDUINO está implementada en una placa pequeña, plana, especialmente diseñada para ser cosida a la tela. Una serie de componentes (pulsadores, sensores, leds...) están especialmente diseñados para usarse en combinación con LilyPad.

Teóricamente, puede trasladarse el mismo programa reloj, de ARDUINO UNO a LilyPad. Es posible montar un dispositivo de este tipo en cualquier prenda.

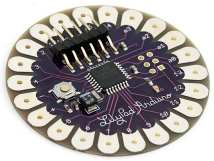

Figura 14: LilyPad 328 Main Board

Montar un reloj con un set de 4 displays como los que hemos presentado puede ser algo problemático por el grosor de los mismos y por ser componentes relativamente grandes y poco flexibles. El resto de componentes necesarios (resistencias, pulsadores y cableado) son más fáciles de esconder entre los forros y la tela. No obstante, puede hacerse, y no cabe duda que es un valor añadido a una prenda.

Una versión del reloj podría ser el que indique horas y minutos formando una esfera, mediante simples LEDs. Necesitaría 12 de un color, para las horas, y 60 de otro color para los minutos (o algún otro diseño original). El bloque básico de programación del reloj puede conservarse; cambiaría la parte de visualización. Es casi seguro que necesitaría una ampliación de las salidas digitales (seguramente tendría que

usar uno o varios “EZ-EXPANDER”). Si unimos la idea de implementar un reloj en una prenda y este tipo de reloj, podrían diseñarse relojes-cojines, por ejemplo. Los cojines son elementos altamente decorativos de un hogar, y ahora, podrían también dar la hora.

Figura 15: Cojines dando la hora

Apartado X

Anexo: programa completo.

```
// siete salidas digitales a cada tramo del display
int outa = 11;
int outb = 13;
int outc = 2;
int outd = 3;
int oute = 4;
int outf = 5;
int outg = 6;
int outsegundero = 12;
int outalarma=1;
// a los anodos de los displays. Actuan por logica inversa
int anodo[4]={7,8,9,10};
// entradas para modificar la hora y minuto
int inhoraria = 0;
int inminutero = 1; //entrada 1
int inalarma = 2; //entrada 2
// cuenta el tiempo
unsigned long hora = 0; // de 0 a 23.
unsigned long minuto = 0; // de 0 a 59
unsigned long hora0 = 10; // de 0 a 23.
unsigned long minuto0 = 10; // de 0 a 59 // :-) Como los relojes de esfera, a las 10
    y 10.
unsigned long segundo0 = 0; // de 0 a 59
unsigned long segundo = 0; // de 0 a 59
unsigned long segundostranscurridos = 0; // de 0 a 59. segundostranscurridos
   transcurridos
unsigned long mediosegundo = 0; // de 0 a 59
unsigned long muestraminuto = 0;
unsigned long muestrahora = 0;
// alarma //Inicializo a las 10:05
unsigned long horaalarma=10;
unsigned long minutoalarma=5;
int alarma=0; // desactivada 0 y activada 1
// variables de los displays
unsigned long displays[4]; // variable para cada display
int cualactivo = 0 ; //variable para guardar cual se activa
//contador general de tiempo
unsigned long tiempo1=0;
int retrasoboton = 100; //retraso del botn al pulsar
// contador de proposito general
int i=0;
/* ====================== */
/* Funciones y subrutinas */
/* ====================== */
// ***********************************************
// Funcion que pone un valor en un display a la vez
// ***********************************************
void mandasenal(int numdisplay, int valor) {
// *****************************************
// a) activa cada tramo del display segun valor
// *****************************************
```

```
// recibe el numero de display y el valor que tiene que entregar
switch (valor) {
case 0:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,LOW);
digitalWrite(outf,LOW);
digitalWrite(outg,HIGH);
break;
break;
case 1:
digitalWrite(outa,HIGH);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,HIGH);
digitalWrite(outg,HIGH);
break;
case 2:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,HIGH);
digitalWrite(outd,LOW);
digitalWrite(oute,LOW);
digitalWrite(outf,HIGH);
digitalWrite(outg,LOW);
break;
case 3:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,HIGH);
digitalWrite(outf,HIGH);
digitalWrite(outg,LOW);
break;
case 4:
digitalWrite(outa,HIGH);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,LOW);
digitalWrite(outg,LOW);
break;
case 5:
digitalWrite(outa,LOW);
digitalWrite(outb,HIGH);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,HIGH);
digitalWrite(outf,LOW);
digitalWrite(outg,LOW);
```

```
break;
case 6:
digitalWrite(outa,HIGH);
digitalWrite(outb,HIGH);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,LOW);
digitalWrite(outf,LOW);
digitalWrite(outg,LOW);
break;
case 7:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,HIGH);
digitalWrite(outg,HIGH);
break;
case 8:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,LOW);
digitalWrite(oute,LOW);
digitalWrite(outf,LOW);
digitalWrite(outg,LOW);
break;
case 9:
digitalWrite(outa,LOW);
digitalWrite(outb,LOW);
digitalWrite(outc,LOW);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,LOW);
digitalWrite(outg,LOW);
break;
case 10:
digitalWrite(outa,HIGH);
digitalWrite(outb,HIGH);
digitalWrite(outc,HIGH);
digitalWrite(outd,HIGH);
digitalWrite(oute,HIGH);
digitalWrite(outf,HIGH);
digitalWrite(outg,LOW);
break;
}
// *********************************
// b) activo un solo display cada vez
// *********************************
for (int i=0;i<=3;i++) { // recorrido de 0 hasta 3
if (i==numdisplay) {
digitalWrite(anodo[i],HIGH); // activo por
} else {
digitalWrite(anodo[i],LOW); // desactivo por
} // fin del if
```

```
} // fin del for
}
// ********************************
// displays
// ********************************
void valordisplay(unsigned long h,unsigned long m,int rayah, int rayam) {
if (rayah==LOW) {
displays[0]= h/10;
displays[1]= h %10; // resto de la division entre 10
} else {
displays[0]= 10;
displays[1]= 10; // guion
}
if (rayam==LOW) {
displays[2]= m / 10;
displays[3]= m %10; // resto de la division entre 10
} else {
displays[2]= 10;
displays[3]= 10; // guion
}
}
// ********************************
// calcula el tiempo
// ********************************
void calculamuestratiempo() {
// 1) calculos para el reloj
segundostranscurridos = hora0*3600+minuto0*60+segundo0+((tiempo1/1000) %86400);
segundo= segundostranscurridos %60;
minuto = segundostranscurridos/60;
hora = minuto/60;
hora = hora %24;
minuto = minuto %60;
}
/* ===================================== */
/* Bucles principales del programa: setup */
/* ===================================== */
void setup () { // inicializo
pinMode(outa, OUTPUT);
pinMode(outb, OUTPUT);
pinMode(outc, OUTPUT);
pinMode(outd, OUTPUT);
pinMode(oute, OUTPUT);
pinMode(outf, OUTPUT);
pinMode(outg, OUTPUT);
pinMode(outsegundero, OUTPUT);
pinMode(anodo[0], OUTPUT);
pinMode(anodo[1], OUTPUT);
pinMode(anodo[2], OUTPUT);
pinMode(anodo[3], OUTPUT);
pinMode(outalarma, OUTPUT);
// Serial.begin(115200);
}
/* ===================================== */
/* Bucles principales del programa: loop */
/* ===================================== */
void loop () { // bucle
```

```
tiempo1=millis(); // milisegundostranscurridos al empezar el bucle
// 1) comprueba horaria
if (analogRead(inhoraria)>500 && analogRead(inalarma)<500) { //compara
delay(retrasoboton); // tiempo de espera 50 milisegundostranscurridos de pulsacion
    de boton. El boton debe mantenerse pulsado 50 milisegundostranscurridos
// hacerlo de esta manera evita que una pulsacion fisica sea reconocida como muchas
    pulsaciones
// ajustar el retraso por cada boton. Se ha puesto un valor de 50 pero podria tener
    que ser mas
if (analogRead(inhoraria)>500) {
hora0=(hora0+1)*(hora0<23);
calculamuestratiempo();
valordisplay(hora,0,LOW,HIGH);
for(i=0;i<=300;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
}
}
// 2) comprueba minutero
if (analogRead(inminutero)>500 && analogRead(inalarma)<500) { //compara
delay(retrasoboton); // tiempo de espera 50 milisegundostranscurridos de pulsacion
    de boton. El boton debe mantenerse pulsado 50 milisegundostranscurridos
if (analogRead(inminutero)>500) {
minuto0=(minuto0+1)*(minuto0<59);
calculamuestratiempo();
valordisplay(0,minuto,HIGH,LOW);
for(i=0;i<=300;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
}
}
// 3) comprueba segundero. Solo verlo
visualizo:
if (analogRead(inalarma)>500) { //compara
delay(retrasoboton); // tiempo de espera 50 milisegundostranscurridos de pulsacion
    de boton. El boton debe mantenerse pulsado 50 milisegundostranscurridos
if (analogRead(inalarma)>500 && analogRead(inhoraria)<500 && analogRead(inminutero)
    <500) {
alarma=!alarma;
valordisplay(horaalarma,minutoalarma,LOW,LOW);
digitalWrite(outsegundero,!alarma);
for(i=0;i<=2000;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
} else if(analogRead(inalarma)>500 && analogRead(inhoraria)>500 && analogRead(
    inminutero)<500) { // caso que pulse las dos a la vez
horaalarma=(horaalarma+1)*(horaalarma<23);
valordisplay(horaalarma,minutoalarma,LOW,LOW);
for(i=0;i<=500;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
goto visualizo;
```

```
} else if(analogRead(inalarma)>500 && analogRead(inhoraria)<500 && analogRead(
    inminutero)>500) {
minutoalarma=(minutoalarma+1)*(minutoalarma<59);
valordisplay(horaalarma,minutoalarma,LOW,LOW);
for(i=0;i<=500;i++) {
mandasenal(i%4,displays[i%4]);
delay(1);
}
goto visualizo;
} // fin del if secundario
} // fin del if principal
// ****************************************
// previo: si llega a la cuenta de los 50 dias
// ****************************************
if (tiempo1<=10 && segundostranscurridos>=1) { // esta situacin no se puede dar de
    inicio
// si segundostranscurridos>=1 entonces como tiempo1>=1000
// Esta situacion puede darse si se alcanzan los 50 dias (limite maximo del conteo
    de milisegundos) y se reinicie la cuenta de tiempo1
// En ese caso fuerzo a que la hora inicial sea la hora actual
hora0=hora;
minuto0=minuto; // pongo la hora y los minutos de inicio como las actuales
segundo0=segundo;
}
//a) segundero
mediosegundo = ((tiempo1/500) %2);
digitalWrite(outsegundero,mediosegundo);
// b) calculo el tiempo
calculamuestratiempo(); // llama a la funcin que calcula las variables temporales
valordisplay(hora,minuto,LOW,LOW);
//c) visualizo
cualactivo=(cualactivo+1)*(cualactivo<3);
mandasenal(cualactivo,displays[cualactivo]); // activa los pines del display elegido
// d) activo alarma
if (horaalarma==hora && minutoalarma==minuto && alarma==1) { // control de la alarma
    . En la hora, en el minuto y si alarma esta activado
tone(outalarma,100*mediosegundo*2,100);
} else {
noTone(outalarma);
}
}
// Fin del programa
```