Buscar

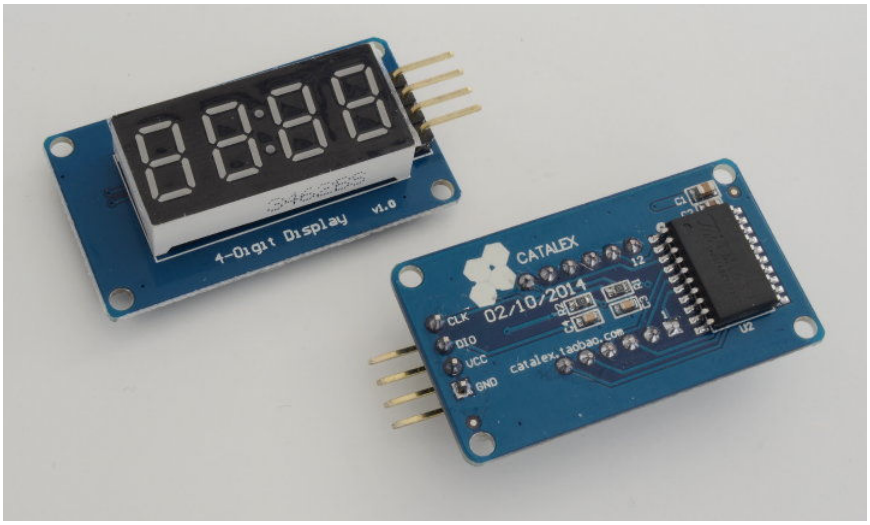
Polaridad.es
que circule la corriente

16 MAR 2015

Librería Arduino para controlar un display LED de dígitos de 7 segmentos con TM1637

por Víctor Ventura | publicado en: Portada | 0

El TM1637 es un driver para display de LED de 7 segmentos y teclado, muy popular por ser muy barato y muy sencillo de usar, tanto a nivel electrónico como a nivel de software. Existe en formato de superficie y de inserción y hay gran número de módulos, sobre todo para gestionar display de LED, que lo incluyen; lo que hace que este integrado sea muy usado para pequeños montajes que muestren valores numéricos.



Como se explica en el anterior artículo sobre el driver TM1637, la comunicación con este dispositivo es del tipo serie de dos hilos: una línea para una señal de reloj y otra línea para la información. Aunque el funcionamiento es muy similar al más conocido I²C, el TM1637 utiliza su propio protocolo TWSI.

Al disponer de resistencias pull-up en las líneas de reloj y datos, cuando las comunicaciones se encuentran inactivas el nivel de la señal es alto. Para realizar las pruebas no es crítico pero para el funcionamiento normal sí es relevante considerar que debe establecerse el estado de alta impedancia en los pines de comunicaciones del microcontrolador para enviar un valor 1, y no un nivel alto de salida, ya que serán las resistencias las responsables de subir el nivel en la línea de comunicaciones. Enviar un valor 0 sí que se consigue estableciendo un nivel bajo de salida. Traducido al lenguaje usado en Arduino, sería necesario usar pinMode(numero_pin,INPUT); (pin «numero_pin» como entrada, para llevarlo al estado de alta impedancia) para conseguir el nivel alto en la línea conectada a «numero_pin» y pinMode(numero_pin,OUTPUT); digitalWrite(numero_pin,LOW); para el nivel bajo.

```
1 // Establecer niveles alto y bajo en comunicaciones TWSI con resistencias pull-up
2
3 // Establecer el nivel alto (valor 1)
4 pinMode(numero_pin,INPUT); // Establecer el modo de alta impedancia en el pin numero_pin para que las resistencias
5
6 // Establecer el nivel bajo (valor 0)
7 pinMode(numero_pin,OUTPUT);
8 digitalWrite(numero_pin,LOW);
```

A grandes rasgos, el protocolo de comunicaciones con el TM1637 consiste en el envío de una orden que describa la operación que se desea realizar y si procede, el envío y/o recepción de información del mismo. Cada byte de datos que se envía, ya corresponda a orden o a datos, va precedido por una señal de inicio y seguido de la recepción de una señal de estado y el envío de una señal de terminación.

Buscar

Entradas recientes

- Librería para manejar Arduin con un mando a distancia por infrarrojos
- Librería Arduino para control un display LED de dígitos de segmentos con TM1637
- TM1637 driver para display L de 6 dígitos de 7 segmentos teclado de 16 pulsadores
- Librería Arduino para monitorización de la frecuen cardíaca con oxímetro de pul
- Principio de funcionamiento d oxímetro para monitorizació pulso

Archivos

- marzo 2015
- Febrero 2015
- enero 2015
- diciembre 2014

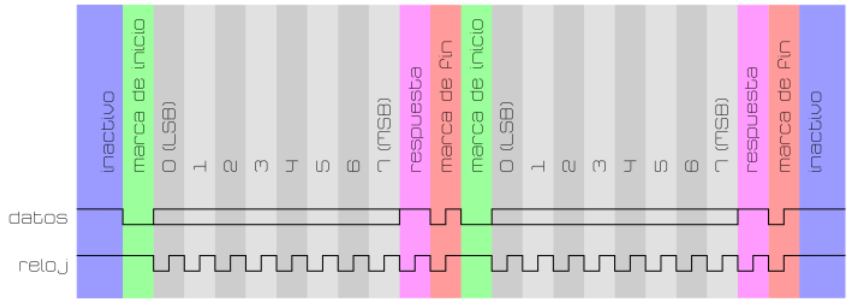
Categorías

- Herramientas
- Noticias
- Portada
- Quiénes somos
- Trucos
- Tutoriales

Meta

- Acceder
- RSS de las entradas
- RSS de los comentarios
- WordPress.org

Como puede verse en el gráfico de abajo, la marca de inicio consiste en establecer durante dos ciclos de reloj a nivel bajo la línea de datos y a nivel alto la de reloj. En el resto de los ciclos, incluyendo el de la marca de fin, la señal de reloj alterna nivel bajo y alto (en este orden) Los bits de datos ocupan dos ciclos cada uno y se envían empezando en el LSB (bit menos significativo) y terminando en el MSB (bit más significativo)



Para conseguir una frecuencia máxima de 250 KHz se puede optar por establecer el nivel correspondiente y no cambiarlo al siguiente hasta que el tiempo de medio ciclo no haya pasado, lo que supone la opción más ortodoxa, o detener la ejecución del programa en el microcontrolador durante un intervalo de tiempo antes de cambiarlo. En la librería para Arduino incluida más abajo he optado por esta segunda alternativa porque los paquetes de datos que se envían son muy pequeños, porque se envían a intervalos muy largos en proporción y porque la pausa para conseguir reproducir la frecuencia es relativamente breve: despreciable en los microcontroladores de menores prestaciones; sólo en los más rápidos merecería la pena modificarla para tratar de aprovechar más recurso de cómputo de microcontrolador infrautilizado en esta versión.

La pausa necesaria para conseguir la frecuencia de 250 KHz es de 4 microsengundos por ciclo, 2 microsegundos a nivel alto y 2 a nivel bajo. Como el integrado se fabrica con cierta tolerancia y la frecuencia es la máxima, debe hacerse una pausa un poco mayor (por seguridad, algo más del 5 % indicado en la datasheet podría ser suficiente) como en Arduino la pausa mínima es de un microsegundo, he optado por hacer pausas de 3 microsegundos entre los dos tiempos de cada ciclo, lo que asegura, aunque con cierto «desperdicio» de tiempo, que la librería proporciona una frecuencia operativa.

En el uso del TM1637 como driver para display de LED de 7 segmentos se utilizan 3 de las órdenes de su protocolo de funcionamiento Además del código 0XC0 ó 0B11000000 que indica el inicio del envío de datos, ya sea usado como byte de inicio o sumado a la información enviada.

Activar-Configurar 0x80/0B10000000 El cuarto bit activa (enciende) o desactiva (apaga) el dispositivo según se envíe en nivel alto o bajo respectivamente. Con los tres últimos bits se establece el valor del brillo, siendo cero el menor brillo (encendido, pero con la menor intensidad) Una operación OR permite realizar directamente el activado del dispositivo con cierto valor de brillo. Como el valor mostrado en el display no se borra, se puede cambiar el brillo mientras se sigue mostrando información.

Como explicaba más arriba, cada comando va precedido de una señal de inicio, la consulta del resultado del envío y una señal de terminación. De este modo, por ejemplo, para activar el dispositivo con un brillo medio se enviaría:

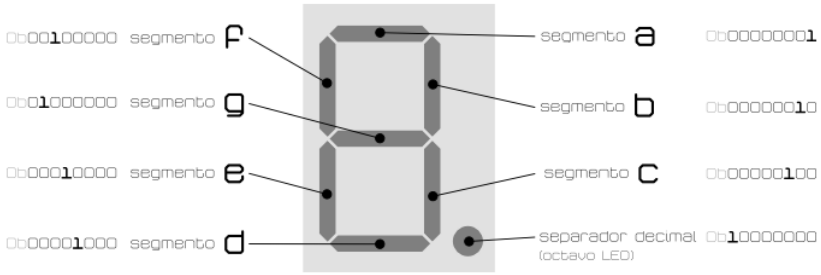
1. Marca de inicio
2. 0b10000000 OR 0b00001000 OR 0b00000011 -> 0b10001011/0x8b (Configurar OR Activar OR brillo nivel 3)
3. Consultar estado
4. Marca de fin

Escribir en una posición 0x44/0b01000100 Se utiliza para cambiar los segmentos encendidos/apagados en un dígito concreto (que corresponde a una dirección de memoria del dispositivo) El protocolo conforme al cual se utiliza esta orden requiere el envío del comando (con las marcas y consultas explicadas antes), el código de inicio de envío de datos (0xC0) al que, con una operación OR, se añade la posición (el dígito que se modifica) y por último el nuevo valor que debe mostrarse. De esta forma, por ejemplo, para cambiar el segundo dígito y dibujar un guión (el segmento g) se seguiría la secuencia:

1. Marca de inicio
2. 0x44/0b01000100
3. Consultar estado
4. Marca de fin
5. Marca de inicio
6. 0xC0/0b01000100 OR 00000001 (Código de inicio de datos OR segundo dígito)
7. 0x40/0b01000000 (segmento g)
8. Consultar estado
9. Marca de fin

Escribir en posiciones consecutivas 0x40/0b01000000 Sirve para cambiar varios dígitos consecutivos (normalmente todos) en una única operación. Esta orden escribe en una posición e incrementa automáticamente para escribir otro valor en la siguiente. Por ejemplo, para modificar los tres primeros dígitos y representar tres líneas horizontales (los segmentos a, g y d) se utilizaría el siguiente protocolo

1. Marca de inicio
2. 0x40/0b01000000
3. Consultar estado
4. Marca de fin
5. Marca de inicio
6. 0xC0/0b01000100 OR 00000000 (Código de inicio de datos OR primer dígito)
7. 0x40/0b00000001 (segmento a)
8. 0x40/0b01000000 (segmento g)
9. 0x40/0b00001000 (segmento d)
10. Consultar estado
11. Marca de fin



Para hacer más cómoda la escritura del código y más sencilla su lectura, en la librería se utilizan nombres (macros o variables) para las órdenes, para los valores de los segmentos y de sus combinaciones (el dibujo de los dígitos) formando un vector (segmentos_numero) con el que acceder más fácilmente a la representación de cada número, y una macro más para la pausa con la que se establece la frecuencia de funcionamiento del driver.

API de la librería

Los objetos de la clase definida en la librería TM1637 están asociados a los pines por los que se comunican con el microcontrolador y con la cantidad de dígitos del driver, que se indican al crear los objetos junto con el brillo con el que inicialmente se activa el dispositivo. Una vez creado el objeto se puede modificar el brillo mientras el driver sigue funcionando pero, lógicamente, no así los pines ni el número de dígitos.

Los métodos de la librería que se describen a continuación permiten presentar en el driver desde valores completos con formato (hora y minutos, por ejemplo) hasta segmentos arbitrarios de uno de los grupos de LED con los que, por ejemplo, «dibujar» letras.

El valor de retorno de la mayoría de las funciones es el estado que el driver devuelve en cada operación. Como lo normal es que los valores se vayan reescribiendo periódicamente, es frecuente que simplemente se ignore este valor de retorno y se muestre correctamente al siguiente envío por lo que podría mostrarse un valor incorrecto durante el intervalo de refresco del display.

activar() Al crear el objeto TM1637 el display no tiene ningún LED encendido ni está preparado para enviar información (se encuentra en estado de alta impedancia) Usando esta función el display permanece apagado y el driver se prepara para recibir datos.

desactivar() Envía al driver la señal de desactivar (apagar) el display sin necesidad de destruir el objeto.

cambiar_brillo(nuevo_brillo) Al crear el objeto se puede indicar el brillo con el que se mostrará inicialmente (el mínimo, por defecto) la función cambiar_brillo se puede establecer un nuevo valor. El TM1637 puede utilizar 8 niveles de brillo (de 0 a 7) siendo cero el mínimo valor encendido visible, es decir, cero no significa apagado sino encendido con la mínima intensidad.

borrar() Una ventaja de usar un driver para LED como el TM1637 consiste en que el microcontrolador sólo tienen que enviar el valor que se desea representar y no es necesario refrescar la información para que se siga mostrando; por contra, esta información permanecerá en el display hasta que se borre, por lo que es necesario incluir esta función en la API de la librería.

escribir_segmento(posición,segmentos) Con este método se cambia el estado de los LED de un dígito individual definiendo en bruto los que se deben encender o apagar en determinada posición. El parámetro «segmentos» es un byte en el que los bits a uno representan los segmentos encendidos (según el esquema de la imagen de arriba) y los bits a cero los segmentos apagados. El parámetro «posición» indica el número del dígito que se modifica.

```
// Mostrar tres líneas horizontales en el segundo dígito

#include "TM1637.h"

#define CANTIDAD_DIGITOS 4 // El display tiene 4 dígitos (el driver soporta hasta 6 dígitos)
#define PAUSA 3000 // Una pausa de 3 segundos para prepararse para ver el comportamiento del montaje

TM1637 pantalla_led(8,9,CANTIDAD_DIGITOS,0); // pin reloj 8, pin datos 9, cantidad dígitos 4, brillo 0 (mínimo
byte segmentos=SEGMENTO_A|SEGMENTO_G|SEGMENTO_D; // encender los segmentos a (horizontal superior) g (medio) y
byte posicion=1; // Segundo dígito

void setup()
{
  pantalla_led.activar();
  pantalla_led.borrar();
  delay(PAUSA);
}

void loop()
{
  pantalla_led.escribir_segmento(posicion,segmentos);
  delay(PAUSA);
  pantalla_led.borrar();
  delay(PAUSA);
}
```

escribir_segmentos(puntero_segmentos) Este método sirve para escribir varios dígitos desde la primera posición. El driver permite empezar en cualquier posición y escribir cualquier número de dígitos. Con esta función se empieza siempre en la primera posición (dígito número cero) y se espera escribir en todas las posiciones del display. La función espera el puntero a una matriz de caracteres cuya longitud corresponde con el número de dígitos que se indicara cuando se creó el objeto TM1637.

```
// Escribir segmentos arbitrarios en 4 segmentos formando el texto «HOLA»

#include "TM1637.h"

#define CANTIDAD_DIGITOS 4 // El display tiene 4 dígitos (el driver soporta hasta 6 dígitos)
#define PAUSA 3000 // Una pausa de 3 segundos para prepararse para ver el comportamiento del montaje

TM1637 pantalla_led(8,9,CANTIDAD_DIGITOS,0); // pin reloj 8, pin datos 9, cantidad dígitos 4, brillo 0 (mínimo
byte segmentos[]= // Un ejemplo de segmentos arbitrarios (HOLA)
     {
       SEGMENTO_B|SEGMENTO_C|SEGMENTO_E|SEGMENTO_F|SEGMENTO_G,
       SEGMENTO_A|SEGMENTO_B|SEGMENTO_C|SEGMENTO_D|SEGMENTO_E|SEGMENTO_F,
       SEGMENTO_D|SEGMENTO_E|SEGMENTO_F,
       SEGMENTO_A|SEGMENTO_B|SEGMENTO_C|SEGMENTO_E|SEGMENTO_F|SEGMENTO_G
     };

void setup()
{
  pantalla_led.activar();
  pantalla_led.borrar();
  delay(PAUSA);
}

void loop()
{
  pantalla_led.escribir_segmentos(segmentos);
  delay(PAUSA);
  pantalla_led.borrar();
  delay(PAUSA);
}
```

escribir_digito(posición,dígito,coma) Con el anterior método se puede escribir una configuración arbitraria de segmentos en un dígito pero lo normal es usar los display de LED de 7 segmentos, como su nombre sugiere, para mostrar valores numéricos con ellos. Con esta función se puede escribir un dígito numérico, el parámetro «dígito», en una posición concreta del display, el parámetro «posición», que opcionalmente puede activar su segmento separador (punto decimal o dos puntos horarios) si el parámetro «coma» tiene el valor true.

escribir_digitos(dígitos,posicion_coma) Con esta función se pueden escribir todos los dígitos del display en formato numérico. El parámetro «dígitos» es un puntero a un vector de dígitos numéricos (bytes) de longitud correspondiente a la usada para crear el objeto TM1637. Con el parámetro «posicion_coma» se puede indicar el dígito que tendrá activado el separador, ya sea punto decimal o dos puntos para información horaria. Para reservar el valor cero como indicador de que no se usa el separador, la posición de la coma empieza a contar desde uno y llega hasta la cantidad de dígitos del display.

numero_a_segmento(número) Con esta función pueden obtenerse los valores de los segmentos que corresponden a los dígitos numéricos de forma que pueden usarse los métodos de escritura arbitraria de segmentos mezclándolos con representaciones numéricas de manera sencilla. El parámetro «número» corresponde con el valor que se traducirá en los segmentos que lo representan.

escribir_valor(valor,posicion_coma) Sirve para escribir un valor numérico completo (el parámetro «valor»), que puede requerir varios dígitos. Opcionalmente se puede activar el separador decimal en la posición indicada por el parámetro «posicion_coma»

```
// Mostrar valores numéricos en el display LED

#include "TM1637.h"

#define CANTIDAD_DIGITOS 4 // El display tiene 4 dígitos (el driver soporta hasta 6 dígitos)
#define PAUSA_CORTA 150 // Una pausa breve para distinguir el cambio de valor
#define PAUSA_LARGA 5000 // Una pausa de 5 segundos para prepararse para ver el comportamiento del montaje

TM1637 pantalla_led(8,9,CANTIDAD_DIGITOS,0); // pin reloj 8, pin datos 9, cantidad dígitos 4, brillo 0 (mínimo

void setup()
{
  pantalla_led.activar();
  pantalla_led.borrar();
  delay(PAUSA_LARGA);
}

void loop()
{
  for(int valor=-999;valor<10000;valor++)
  {
    pantalla_led.escribir_valor(valor); // Al omitir el parámetro de la posición del separador decimal no se a
    delay(PAUSA_CORTA);
  }
  delay(PAUSA_LARGA);
  pantalla_led.borrar();
  delay(PAUSA_LARGA);
}
```

escribir_horas_minutos(horas,minutos,puntos,cero_inicial) Esta función muestra en el display dos valores ya formateados suponiendo que dispone de 4 dígitos destinados a información horaria: el separador son dos puntos. Escribe las horas (parámetro «horas») y los minutos (parámetro «minutos») encendiendo el separador (dos puntos en un display horario) cuando el parámetro «puntos» vale true y los valores con un cero inicial cuando son menores de 10 si el parámetro «cero_inicial» es true.

escribir_minutos_segundos(minutos,segundos,puntos,cero_inicial) Este método es un envoltorio (wrapper) del anterior para que el código que se escriba usándolo gane en claridad. Los métodos de la librería para mostrar información horaria no verifican los valores indicados por los parámetros de entrada, sólo simplifican su uso.

escribir_horas_minutos_segundos(horas,minutos,segundos,cero_inicial) Con esta función, desarrollada para usar un display horario (con separador de dos puntos) de 6 dígitos, el máximo que soporta el TM1637, se puede representar la hora, los minutos y los segundos en una única operación que se usa como las anteriores.

```
// Mostrar las horas y los minutos desde que inicia el programa

#include "TM1637.h"

#define CANTIDAD_DIGITOS 4 // El display tiene 4 dígitos (el driver soporta hasta 6 dígitos)
#define SEGUNDO 1000 // Milisegundos de un segundo
#define MINUTO 60000 // Milisegundos en un minuto (1000*60)
#define FINAL 6000000 // 6000000 milisegundos = 100 minutos (100*1000*60) Posteriormente parpadea en lugar de
#define PAUSA_PARPADEO 300 // Al llegar al valor FINAL no se muestra el tiempo sino que se hace parpadear al d

TM1637 pantalla_led(8,9,CANTIDAD_DIGITOS,0); // pin reloj 8, pin datos 9, cantidad dígitos 4, brillo 0 (mínimo
// byte desbordamiento[]={SEGMENTO_G,SEGMENTO_G,SEGMENTO_G,SEGMENTO_G}; // Como alternativa pueden hacerse par

byte minutos;
byte segundos;
unsigned long reloj;
boolean estado_parpadeo=false;

void setup()
{
  pantalla_led.activar();
  pantalla_led.borrar();
  reloj=0;
}

void loop()
{
  if(millis()>FINAL)
  {
    estado_parpadeo=!estado_parpadeo;
    if(estado_parpadeo)
    {
      // pantalla_led.escribir_segmentos(desbordamiento); // Como alternativa pueden hacerse parpadear cuatro
      pantalla_led.escribir_segmento(1,CON_PUNTOS);
    }
    else
    {
      pantalla_led.borrar();
    }
    delay(PAUSA_PARPADEO);
  }
  else
  {
    if(millis()>reloj)
    {
      reloj=millis()+SEGUNDO;
      minutos=millis()/MINUTO; // Como no hay horas, se muestran más de 59 minutos
      segundos=(millis()-minutos*MINUTO)/SEGUNDO;
      pantalla_led.escribir_minutos_segundos(minutos,segundos); // Por defecto muestra el separador horario y
    }
  }
}
```

Librería Arduino para controlar un display LED con el driver TM1637

```
//TM1637.h

#if defined(ARDUINO) && ARDUINO>=100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#define TM1637_PAUSA 3 // Un poco más de dos, la mitad de cuatro microsegundos, para una frecuencia máxima de 250

#define TM1637_ACTIVAR 0x88 // Inicializar OR Activar [OR brillo]
#define TM1637_DESACTIVAR 0x80 // Inicializar (sin activar ni brillo)
#define TM1637_ESCRIBIR_EN_POSICION 0x44
#define TM1637_ESCRIBIR_INCREMENTAR 0x40
#define TM1637_INICIO_DATOS 0xC0

#define SEGMENTO_A 0B00000001
#define SEGMENTO_B 0B00000010
#define SEGMENTO_C 0B00000100
#define SEGMENTO_D 0B00001000
#define SEGMENTO_E 0B00010000
#define SEGMENTO_F 0B00100000
#define SEGMENTO_G 0B01000000
#define CON_PUNTOS 0B10000000  // Para mostrar el punto decimal (o dos puntos en un display de reloj) haciendo una

class TM1637
{
  private:
    byte pin_reloj;
    byte pin_datos;
    byte cantidad_digitos;
    byte brillo;
    byte segmentos_numero[11]; // Representación del 0 al 9 en un display de 7 segmentos (el último es el que repr
    long valores_maximos[7]; // El TM1637 puede representar 6 dígitos como máximo
    long valores_minimos[7];
    void inicio_envio();
    void fin_envio();
    boolean enviar(byte dato);
  protected:
  public:
    TM1637(byte nuevo_pin_reloj,byte nuevo_pin_datos,byte nuevo_numero_digitos=4,byte nuevo_brillo=0);
    ~TM1637();
    boolean activar();
    boolean desactivar();
    boolean cambiar_brillo(byte nuevo_brillo);
    boolean borrar();
    boolean escribir_valor(long valor,byte coma=0);
    boolean escribir_horas_minutos(byte horas,byte minutos,boolean puntos=true,boolean cero_inicial=false);
    boolean escribir_minutos_segundos(byte horas,byte minutos,boolean puntos=true,boolean cero_inicial=true);
    boolean escribir_horas_minutos_segundos(byte horas,byte minutos,byte segundos,boolean cero_inicial=false);
    boolean escribir_digito(byte posicion,byte digito,boolean coma=false);
    boolean escribir_digitos(byte *digito,byte coma=0);
    boolean escribir_segmento(byte posicion,byte segmento);
    boolean escribir_segmentos(byte *segmento);
    byte numero_a_segmento(byte numero);
};
```

```
//TM1637.cpp

#include "TM1637.h"

TM1637::TM1637(byte nuevo_pin_reloj,byte nuevo_pin_datos,byte nuevo_cantidad_digitos,byte nuevo_brillo)
{
  // Representación de los decimales en un display de 7 segmentos
  segmentos_numero[0]=0x3F;
  segmentos_numero[1]=0x06;
  segmentos_numero[2]=0x5B;
  segmentos_numero[3]=0x4F;
  segmentos_numero[4]=0x66;
  segmentos_numero[5]=0x6D;
  segmentos_numero[6]=0x7D; // 0x7C; El 6 feo ahorra energía
  segmentos_numero[7]=0x27; // 0x07; El 7 clásico ahorra energía
  segmentos_numero[8]=0x7F;
  segmentos_numero[9]=0x6F; // 0x67; El 9 de palo ahorra energía
  segmentos_numero[10]=0x40; // Dibujo usado para simbolizar un de error (será una tira de guiones creada con el
  // Valores máximos (más uno, para ahorrar en el bucle) El TM1637 puede representar 6 dígitos como máximo. Más
  valores_maximos[0]=0; // Añadido para poder comparar directamente con el número de dígitos
  valores_maximos[1]=10;
  valores_maximos[2]=100;
  valores_maximos[3]=1000;
  valores_maximos[4]=10000;
  valores_maximos[5]=100000;
  valores_maximos[6]=1000000;
  // Valores mínimos. El TM1637 puede representar 6 dígitos como máximo. Más rápido almacenar los valores que ca
  valores_minimos[0]=0; // Añadido para poder comparar directamente con el número de dígitos
  valores_minimos[1]=0;
  valores_minimos[2]=-9;
  valores_minimos[3]=-99;
  valores_minimos[4]=-999;
  valores_minimos[5]=-9999;
  valores_minimos[6]=-99999;
  pin_reloj=nuevo_pin_reloj;
  pin_datos=nuevo_pin_datos;
  cantidad_digitos=nuevo_cantidad_digitos;
  brillo=nuevo_brillo;
}

TM1637::~TM1637()
{
  desactivar();
}

boolean TM1637::activar()
{
  pinMode(pin_reloj,INPUT);
  pinMode(pin_datos,INPUT);
  delayMicroseconds(TM1637_PAUSA);
  delayMicroseconds(TM1637_PAUSA);
  return cambiar_brillo(brillo);
}

boolean TM1637::desactivar()
{
  boolean estado;
  inicio_envio();
  estado=enviar(TM1637_DESACTIVAR);
  fin_envio();
  return estado;
}

boolean TM1637::cambiar_brillo(byte nuevo_brillo) // Como no se borra el contenido de los registros también puede
{
  boolean estado;
  inicio_envio();
  estado=enviar(TM1637_ACTIVAR|(nuevo_brillo&0B00000111)); // Ignorar todos los bits menos los tres últimos para
  fin_envio();
  return estado;
}

boolean TM1637::borrar()
{
  byte todo_ceros[cantidad_digitos];
  for(byte numero_digito=0;numero_digito<cantidad_digitos;numero_digito++)
  {
    todo_ceros[numero_digito]=0;
  }
  return escribir_segmentos(todo_ceros);
}

boolean TM1637::escribir_valor(long valor,byte coma)
{
  byte segmento[cantidad_digitos];
  byte numero_digito=0;
  long resto=abs(valor);
  byte digito;
  if(valor>valores_maximos[cantidad_digitos]-1||valor<valores_minimos[cantidad_digitos])
  {
    for(numero_digito=0;numero_digito<cantidad_digitos;numero_digito++)
    {
      segmento[numero_digito]=SEGMENTO_G;
    }
  }
  else
  {
    if(valor==0)
    {
      numero_digito++;
      segmento[cantidad_digitos-numero_digito]=segmentos_numero[0];
```

```
    }
    else
    {
      while(numero_digito<cantidad_digitos&&resto>0)
      {
        numero_digito++;
        digito=resto%10;
        segmento[cantidad_digitos-numero_digito]=segmentos_numero[digito];
        resto/=10;
      }
      if(valor<0)
      {
        numero_digito++;
        segmento[cantidad_digitos-numero_digito]=SEGMENTO_G;
      }
    }
    while(numero_digito<cantidad_digitos)
    {
      numero_digito++;
      segmento[cantidad_digitos-numero_digito]=0; // Apagar los que no se usen
    }
  }
  return escribir_segmentos(segmento);
}

boolean TM1637::escribir_horas_minutos(byte horas,byte minutos,boolean puntos,boolean cero_inicial)
{
  // No se verifican los valores para poder usarla de forma flexible
  byte segmento[4]; // hora y minutos siempre son 4 digitos
  segmento[0]=horas/10;
  if(segmento[0]>0||cero_inicial)
  {
    segmento[0]=segmentos_numero[segmento[0]];
  }
  else
  {
    segmento[0]=0;
  }
  segmento[1]=segmentos_numero[horas%10];
  if(puntos)
  {
    segmento[1]|=CON_PUNTOS;
  }
  segmento[2]=segmentos_numero[minutos/10];
  segmento[3]=segmentos_numero[minutos%10];
  return escribir_segmentos(segmento);
}

boolean TM1637::escribir_minutos_segundos(byte horas,byte minutos,boolean puntos,boolean cero_inicial)
{
  return escribir_horas_minutos(horas,minutos,puntos,cero_inicial);
}

boolean TM1637::escribir_horas_minutos_segundos(byte horas,byte minutos,byte segundos,boolean cero_inicial)
{
  byte segmento[6]; // hora, minutos y segundos siempre son 6 digitos
  segmento[0]=horas/10;
  if(segmento[0]>0||cero_inicial)
  {
    segmento[0]=segmentos_numero[segmento[0]];
  }
  else
  {
    segmento[0]=0;
  }
  segmento[1]=segmentos_numero[horas%10];
  segmento[2]=segmentos_numero[minutos/10]|CON_PUNTOS; // Siempre con puntos
  segmento[3]=segmentos_numero[minutos%10];
  segmento[4]=segmentos_numero[segundos/10]|CON_PUNTOS; // Siempre con puntos
  segmento[5]=segmentos_numero[segundos%10];
  return escribir_segmentos(segmento);
}

boolean TM1637::escribir_digito(byte posicion,byte digito,boolean coma)
{
  return escribir_segmento(posicion,numero_a_segmento(digito));
}

boolean TM1637::escribir_digitos(byte *digito,byte coma)
{
  byte segmento[cantidad_digitos];
  for(byte numero_digito=0;numero_digito<cantidad_digitos;numero_digito++)
  {
    segmento[numero_digito]=numero_a_segmento(digito[numero_digito]);
    if(numero_digito==coma-1)
    {
      segmento[numero_digito]|=CON_PUNTOS;
    }
  }
  return escribir_segmentos(segmento);
}

boolean TM1637::escribir_segmento(byte posicion,byte segmento)
{
  boolean estado;
  inicio_envio();
  estado=enviar(TM1637_ESCRIBIR_EN_POSICION);
  fin_envio();
  inicio_envio();
  estado|=enviar(TM1637_INICIO_DATOS|(posicion&0B00000011)); // Ignorar todos los bits menos los dos últimos para
  estado|=enviar(segmento);
```

```
  fin_envio();
  return estado;
}

boolean TM1637::escribir_segmentos(byte *segmento) // Escribe toda la pantalla, el vector debe contener todos los
{
  boolean estado;
  byte numero_digito;
  inicio_envio();
  estado=enviar(TM1637_ESCRIBIR_INCREMENTAR);
  fin_envio();
  inicio_envio();
  estado|=enviar(TM1637_INICIO_DATOS);
  for(numero_digito=0;numero_digito<cantidad_digitos;numero_digito++)
  {
    estado|=enviar(segmento[numero_digito]);
  }
  fin_envio();
  return estado;
}

// No se utiliza digitalWrite para enviar un nivel alto. Establecer el modo como entrada hace que el pin pase a e

boolean TM1637::enviar(byte dato)
{
  boolean respuesta;
  for(byte numero_bit=0;numero_bit<8;numero_bit++)
  {
    pinMode(pin_reloj,OUTPUT);
    digitalWrite(pin_reloj,LOW);
    if(dato&(1<<numero_bit))
    {
      pinMode(pin_datos,INPUT);
    }
    else
    {
      pinMode(pin_datos,OUTPUT);
      digitalWrite(pin_datos,LOW);
    }
    delayMicroseconds(TM1637_PAUSA);
    pinMode(pin_reloj,INPUT);
    delayMicroseconds(TM1637_PAUSA);
  }
  // Leer respuesta TM1637 (estado)
  pinMode(pin_reloj,OUTPUT);
  digitalWrite(pin_reloj,LOW);
  delayMicroseconds(TM1637_PAUSA);
  pinMode(pin_datos,INPUT);
  pinMode(pin_reloj,INPUT);
  delayMicroseconds(TM1637_PAUSA);
  respuesta=digitalRead(pin_datos);
  return respuesta;
}

void TM1637::inicio_envio()
{
  pinMode(pin_reloj,INPUT);
  pinMode(pin_datos,OUTPUT);
  digitalWrite(pin_datos,LOW);
  delayMicroseconds(TM1637_PAUSA);
  delayMicroseconds(TM1637_PAUSA);
}

void TM1637::fin_envio()
{
  pinMode(pin_reloj,OUTPUT);
  digitalWrite(pin_reloj,LOW);
  pinMode(pin_datos,OUTPUT);
  digitalWrite(pin_datos,LOW);
  delayMicroseconds(TM1637_PAUSA);
  pinMode(pin_reloj,INPUT);
  delayMicroseconds(TM1637_PAUSA);
  pinMode(pin_datos,INPUT); // Al terminar, reloj y datos quedan en alta impedancia hasta la siguiente operación
}

byte TM1637::numero_a_segmento(byte numero)
{
  if(numero>9)
  {
    return(segmentos_numero[10]);
  }
  else
  {
    return(segmentos_numero[numero]);
  }
}
```

7 segmentos, Arduino, display, driver, LED, librería, TM1637

Sobre el Autor | Últimos mensajes

Víctor Ventura
Programador multimedia y web. Mejor con software libre.

Desarrollando aplicaciones para la web conocí el potencial de internet de las cosas, encontré la excusa perfecta para satisfacer la inquietud de aprender electrónica que tenía desde niño y ahora puedo darme el gusto de programar las cosas que yo mismo diseño y fabrico.

Dejar una opinión

Nombre *

Email (no será publicado) *

Sitio Web

Comentarios

Enviar

☐
Por favor confirma que eres humano antes de comentar