



- [Inicio](#)
- [Tutoriales](#)
- [Proyectos](#)
- [A pie de Calle](#)
- [Contacto](#)

Navigation ▼

noviembre 12, 2014 [Artículo](#) [6 Comments](#)

Optimización de Memoria de nuestro código de Arduino

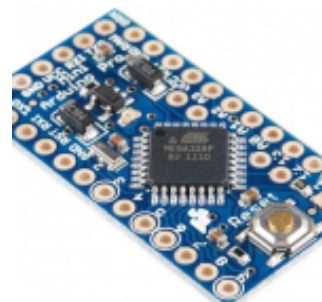
```
void store_log (void)
{
  /* we continuously update the "current" entry */
  int logno = get_minutes_since_midnight () / LOG_INTERVAL;
  if ((logno < 0) ||
      (logno >= (24*60/LOG_INTERVAL)))
    logno = 0;
  // datalog[logno].temperature = cached_temperature;
  // datalog[logno].humidity = cached_humidity;
  // datalog[logno].moisture = moisture_read()//cached_moisture * 100 / moisture_calib;
}

void setup(){
  reset_settings (config);
}
```

A veces en proyectos grandes, hay que tener muy en cuenta el tamaño de los datos y el tamaño de nuestro programa ya que arduino, como cualquier otro programa para sistemas empujados, tiene un hardware limitado y por lo tanto hay que tener en cuenta varios puntos en cuenta. Lo primero que hay que saber, es diferenciar el tipo de memoria que hay disponible en Arduino:

- Flash: Esta memoria almacena el código del programa.
- RAM: Esta memoria almacena los datos del programa y es volátil cuando se apaga arduino se pierden los datos.
- EEPROM: Esta memoria almacena datos que son persistentes. Esta memoria puede ser utilizada para guardar configuración o cualquier otro dato que podamos utilizar si se apaga y enciende Arduino.

Con estos datos, podemos ver en primer lugar, que tipo de Arduino puede ser más correcto para nuestro proyecto. Aquí os dejamos una tabla de comparación con algunas de las placas más famosas de Arduino.

UNO**MEGA 2560****Mini****PRO Mini**

32KB Flash	256KB Flash	32KB Flash	16KB Flash
2KB RAM	8KB RAM	2KB RAM	1KB RAM
1KB EEPROM	4KB EEPROM	1KB EEPROM	512 B EEPROM

Una vez hemos visto las distintas placas que podemos obtener, vamos a ver las distintas técnicas que podemos utilizar para mejorar la memoria y poder optimizar el código de nuestros proyectos al máximo.

1. Utilizar tipos de datos correctos
2. Optimizar las cadenas de caracteres
3. Optimizar nuestro código de programa
4. Optimizar las variables del programa pasándolas a la PROGMEM.
5. Optimizar los datos guardando información en la EEPROM.

Estas son las técnicas que vamos a estudiar en este artículo. Ahora las analizaremos una por una para mostrar como utilizarlas.

1. Utilizar tipos de datos correctos.

Obviamente lo más sencillo y básico que podemos hacer es utilizar correctamente los tipos de dato que utilizaremos en nuestras variables. Ya que en función de lo que necesitemos, podemos utilizar uno u otro. Seguidamente mostramos una tabla con los tipos de datos básicos con Arduino:

	Tipo	Tamaño	Rango
byte	entero	1 byte	0-255
short/int	entero	2byte	-32,767 a 32768
long	entero	4byte	2,147,483,647 a -2,147,483,648
float	real	4byte	3.4028235E+38 a -3.4028235E+38.

Con los datos de la tabla debemos elegir los tipos de datos correctos para nuestros proyectos.

2. Optimizar Cadenas de Caracteres

Otro de los problemas que más memoria puede gastar, es las cadenas de caracteres con respecto a los mensajes que mostraremos. Esto puede ser un problema de gasto de memoria RAM. Por ello, se puede utilizar una función llamada F la cual hace que las cadenas de caracteres, queden almacenadas en la memoria de programa y no en la memoria RAM.

```
Serial.print("Hola Mundo");//Gasto de memoria RAM.
Serial.print(F("Hola Mundo"));//Gasto de Memoria Flash.
```

Por lo tanto para no malgastar RAM, interesa utilizar la función F.

3. Optimizar el código de nuestro programa

Otro tema que tenemos que tener en cuenta, es la optimización de nuestro código para Arduino; hay que tener en cuenta que necesitaremos gastar el mínimo de tamaño de nuestro programa ya que como hemos visto anteriormente el tamaño de programa esta limitada. Os dejamos unos consejos para optimizar este tamaño:

1. Utilizar Constantes:

Para variables que no cambien tamaño, es mejor utilizar constantes y así no gastar memoria ni de programa ni RAM:

```
int led = 13; // Utilizar una variable para definir un PIN gasta RAM y Memoria de Programa
#define LED 13 // SI utilizamos una constante no se gastará RAM y se gastará mucho menos e
```

2. Utilizar Macros

A la hora de crear funciones, podemos utilizar Macros para operaciones mecánicas. De esta manera ahorraremos en primer lugar, RAM, Memoria de Programa e incluso evitaremos el desbordamiento de PILA ya que no se utiliza la pila de programa a la hora de utilizar una macro.

```
void mostrarenpantalla(String cadena){ lcd.print(cadena); } // esta función gasta RAM y
#define mostrarenpantalla(cadena){ lcd.print(cadena); } // no gasta RAM pero si memoria
```

3. Revisar nuestro código

Por último, tenemos que tener en cuenta que debemos utilizar poco código de manera eficiente para poder gastar menos memoria de programa. Uno de las técnicas que se puede utilizar, es usar el bucle 'for' de manera inversa:

```
for(byte i=0; i < 10; i++) // este es un bucle for normal
for(byte i=10; i >= 0; i--) // Este bucle for se recorre de forma inversa para optimizar
```

Otro método, es utilizar los operadores de asignación aritmética.

```
c = c+1; // esto no es eficiente
c++; // Esto es más eficiente
```

4. Optimizar las variables del programa pasándolas a la PROGMEM.

Para ahorrar RAM, podemos utilizar la palabra PROGMEM para poder definir las variables dentro de la memoria de programa y así ahorrar memoria RAM. Aquí definiremos como utilizar esta palabra. Esta palabra reservada, se utiliza para definir arrays dentro de la memoria de programa.

Además, existen unos tipos de datos predefinidos para poder definir variables de tipos básicos en la memoria de Programa.

Tipo	Descripción
prog_char	Caracter de 1 byte
prog_uchar	Caracter sin signo de 1 byte
prog_int16_t	Entero de 2 bytes
prog_uint16_t	Entero sin signo de 2 bytes
prog_int32_t	Entero largo de 4 bytes

prog_uint32_t

Entero largo sin signo de 4 bytes

utilizando estos datos, podemos utilizar parte de la memoria de programa para nuestras variables. Recordamos que esta memoria también es finita y que tampoco podemos utilizarla para todo.

5. Guardar las variables en la EEPROM

Otro aspecto para guardar nuestros datos de forma persistente, es utilizar la EEPROM la cual nos permite guardar datos en la memoria de manera de que si se apaga nuestra placa de arduino, no se pierdan estos datos. Esto es interesante para poder guardar variables de configuración o cualquier otro tipo de dato. Os dejamos un ejemplo de código.

```
#include < EEPROM.h > //incluimos la libreria para guardar o leer de la EEPROM
int i=0;
void setup(){
  Serial.begin(9600);
}
void loop(){

  for(byte i=10;i > 0;i--){
    EEPROM.write(i,i); //Escritura en EEPROM.
  }

  for(byte i=10;i > 0;i--){
    int dato=EEPROM.read(i);
    Serial.println(i);
  }
}
```

Con este último ejemplo, ya hemos visto los mecanismos que tenemos para optimizar nuestro código en Arduino y mejorar el uso tanto de RAM como de memoria de programa. Más adelante, hablaremos de como aumentar la memoria por medio de una tarjeta SD o de memoria RAM adicional.

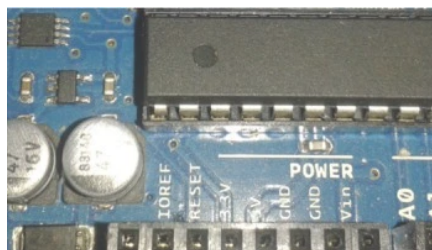
Compártelo:

[✉ Correo electrónico](#)[f Facebook 4](#)[t Twitter](#)[G+ Google](#)[in LinkedIn](#)

Relacionado

[Comenzar con Arduino](#)

En "tutoriales"

[Adquisición de datos con Arduino I: Tiempo de muestreo y Resolución](#)

En "tutoriales"

[Internet de las Cosas I: Conexión Raspberry Pi con Arduino](#)

En "Proyectos"

[arduino](#), [código](#), [memoria](#), [optimizar](#)

6 Responses to Optimización de Memoria de nuestro código de Arduino



1. *diego* dice:
[agosto 16, 2015 a las 11:58 pm](#)

Pues sigo con la misma duda de Juan Carlos.

Porqué recorrer un bucle a la inversa es mejor?, no encuentro la respuesta en tu contestación.

Por cierto, buen artículo, muy interesante.

[Responder](#)



2. *Alvaro M.* dice:
[junio 7, 2015 a las 12:28 am](#)

Excelente!

Gracias a estos consejos, conseguí, de estar en situación de inestabilidad (<75% de memoria usada) hasta un 64% de memoria usada.

Muy recomendable =)

[Responder](#)



- o *administracion* dice:
[junio 8, 2015 a las 8:42 am](#)

Nos alegra saber que te ha servido la información.

[Responder](#)



3. *Maximiliano Duarte* dice:
[febrero 27, 2015 a las 8:19 pm](#)

Me ayudó mucho a mejorar mi forma de programar, olvidada de C++ cuando eran cientos de define.

Y muy util las macros cuando usamos display o salidas por serial para debuger.

[Responder](#)



4. *Juan Carlos* dice:
[enero 15, 2015 a las 12:43 am](#)

No me quedo claro porque recorrer bucles for a la inversa, es mejor.

Un saludo.

[Responder](#)



- o *administracion* dice:
[enero 15, 2015 a las 8:17 am](#)

Es más eficiente ya que el procesador del arduino, accede a la memoria pero no obtiene por ejemplo cada dato de la memoria RAM de uno en uno. Si no que lo hace en bloques por lo que obtiene los bloques adyacentes. Muchas veces El propio Compilador optimiza este acceso. Por eso al obtener el bloque entero, el acceso es mucho más rápido ya que siempre el acceso a la RAM es mucho más lento que usar por ejemplo una pequeña Cache que trae el microcontrolador.

[Responder](#)

Deja un comentario

Tu dirección de correo electrónico no será publicada. Los campos necesarios están marcados *

Comentario

Nombre *

Correo electrónico *

Web



Código CAPTCHA*

Publicar comentario

☐ Recibir un email con los siguientes comentarios a esta entrada.

☐ Recibir un email con cada nueva entrada.

•

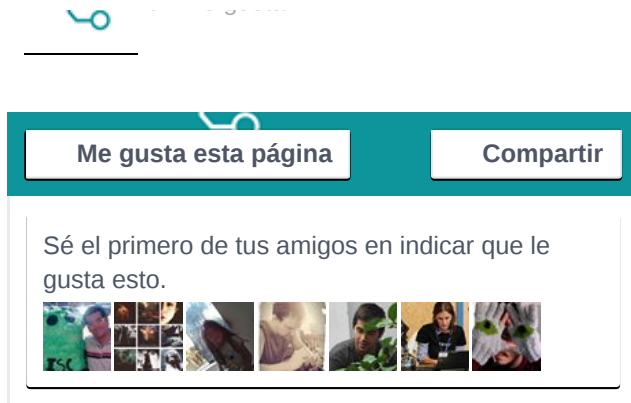
• Últimas Entradas

- [:Montad Neumáticos Blandos! Llega CES \(Campeonato Español de Simulación\)](#) enero 10, 2016
- [Análisis Raspberry Pi Zero](#) diciembre 19, 2015
- [El Internet de las cosas III: Monitor de Temperatura](#) octubre 29, 2015
- [Github IV: Ramas](#) octubre 8, 2015
- [El Hackaton 2015](#) septiembre 9, 2015

[BooleanBite](#)



Booleanbite
64 Me gusta



-
- **Encuesta**






¿Que te gustaría que trataramos en Booleanbite.com?

- ☐ Tutoriales
- ☐ Artículos de Opinión
- ☐ Artículos sobre proyectos
- ☐ Otro tipo

[Vote](#)

[View Results](#)

- **Entradas + Populares**

	Adquisición de datos con ... 299 vistas publicado el febrero 10, 2015
	Optimización de Memoria d... 168 vistas publicado el noviembre 12, 2014
	Internet de las Cosas I: ... 114 vistas publicado el diciembre 12, 2014
	Arduino Bajo Consumo 104 vistas publicado el mayo 15, 2015
	Guia Rápida Para Fritzing... 63 vistas publicado el abril 15, 2015

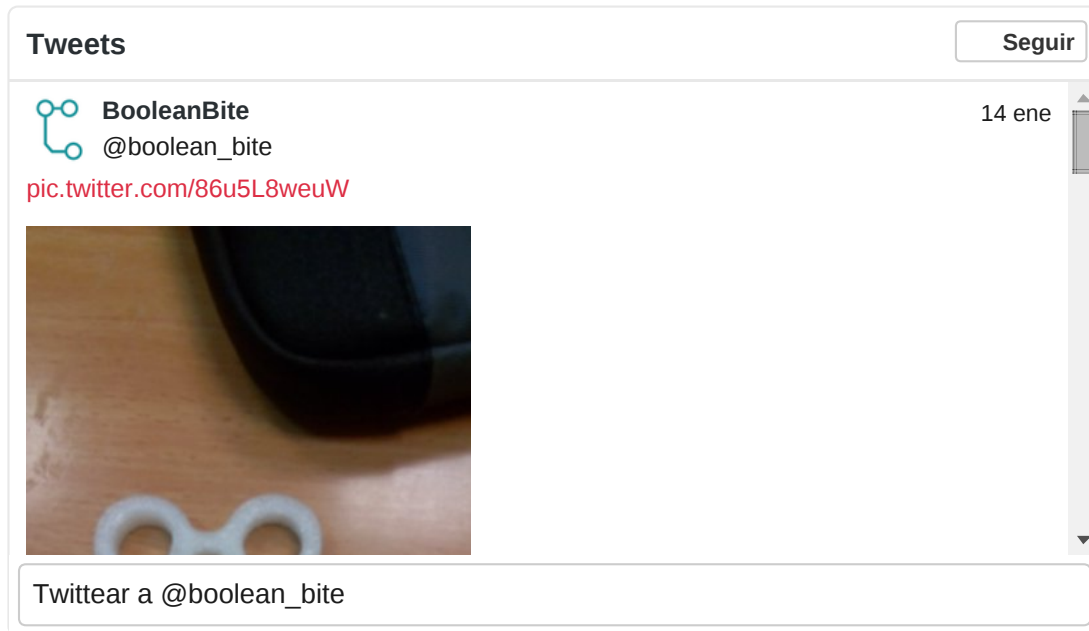


-
- **Donación**

Hola; si te gusta Booleanbite.com, puedes ayudarnos con una simple donación.

[Donar](#)

- **Timeline**



- **Cloud Tag**

[555](#) [alarma](#) [Amplificador operacional](#) [analisis](#) [Analógica](#) [AO](#) [a pie de calle](#) [arduino](#) [articulo](#) [bienvenida](#) [blink](#) [booleanbite](#) [c++](#) [comenzar](#) [concurso](#) [código](#) [Electrónica](#) [fritzing](#) [git](#) [github](#) [hc-sr04](#) [IDE](#) [inbox](#) [iniciación](#) [internet de las cosas](#) [iot](#) [java](#) [kit](#) [led](#) [libreria](#) [matriz](#) [PCB](#) [pi](#) [programacion](#) [python](#) [raspberrypi](#) [raspberrypi](#) [raspberrypi 2](#) [repositorio](#) [serial](#) [serpiente](#) [tutorial](#) [UA741](#) [ultrasonido](#) [videojuego](#)

-

- **Suscribete a nuestro Newsletter**

Introduce tu correo electrónico para suscribirte a este blog y recibir notificaciones de nuevas entradas.

Únete a otros 11 suscriptores

BooleanBite © 2016 - [Aviso Legal](#) - [Política de Privacidad](#) - [Política de Cookies](#) - [Google +](#) - [SiteMap](#)