EEPROM

Nosotros cuando guardamos la información en variables dentro del Arduino estas se actualizan dentro de la VRAM, esta es una memoria volátil por lo cual en caso de un apagado del Arduino toda esta información se perderá.

Para esto existe la memoria EEPROM la cual es una memoria no volátil de lectura y escritura, esta es como un pequeño disco duro para nuestro microcontrolador en la cual podremos guardar la información que necesitemos sin miedo a perderla.

Algo a recalcar es que la información almacenada que tengamos no cambiara por más que cambiemos el código cargado en el microcontrolador mientras claro no cambiemos alteremos la información.

Los microcontroladores de las placas arduino tienen incorporado una memoria EEPROM de distintas capacidades, estas memorias se separan Bytes y nosotros tenemos la capacidad de alterar uno a la vez.

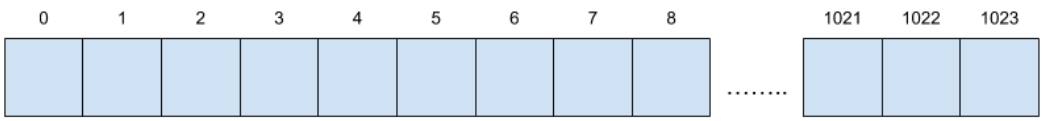
MICROCONTROLADOR	EEPROM
ATmega328 (Arduino UNO, Nano, Mini)	1024 bytes
ATmega168 (Arduino Nano)	512 bytes
ATmega2560 (Arduino Mega)	4096 bytes

Algo importante a recalcar es que cada Byte tiene una vida útil de 100’000 ciclos de escritura

Como escribir en ella?

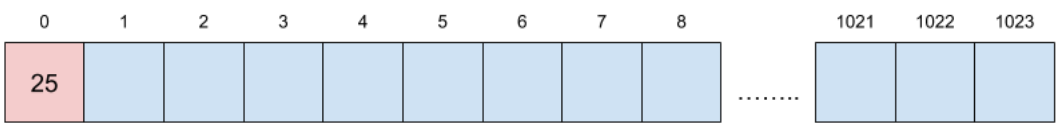
La memoria EEPROM está dividida en huecos. Cada hueco representa un byte (8bit o 255).

Entonces en el caso del Arduino Nano con un ATmega328 tendríamos:

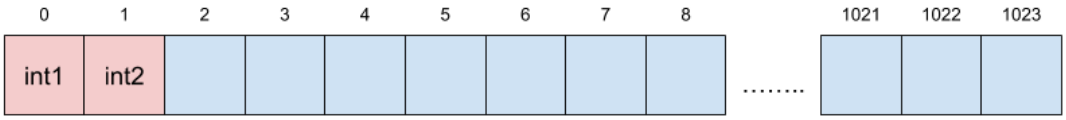


Si queremos guardar información dentro de esta tendremos que especificar la dirección a la cual nos referimos y el valor que queremos cambiar

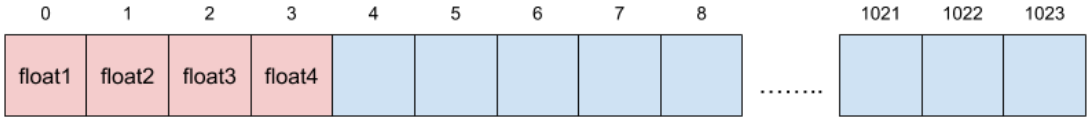
Guardar (dirección: 0, variable: 25)



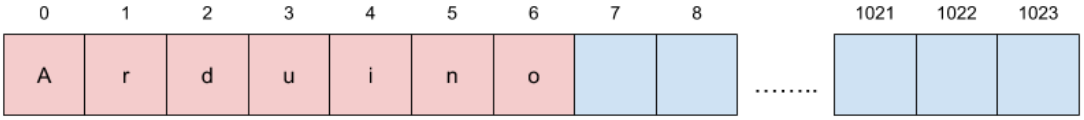
Si quisiéramos guardar una variable int la cual ocupa 2bit de información (32,767 y -32,768) nos quedaría de la siguiente forma:



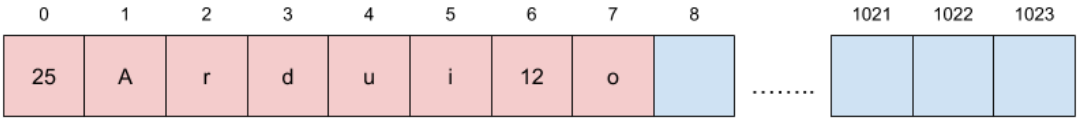
Lo mismo con el float que ocupa 4bit:



En caso de ser una cadena de texto, cada carácter ocupara un byte como por ejemplo la palabra “Arduino”:



Todo esto nos indica que la gestión de la memoria debemos hacerlo de forma manual y consiente para evitar sobre escrituras y errores de lectura:



Arduino nos facilita todo esto incorporando una biblioteca dentro de mismo IDE para el manejo de la EEPROM, la cual la podemos llamar de la siguiente forma: #include <EEPROM.h>

Funciones:

Antes de empezar con las funciones es importante resaltar el número de bytes que ocupa las diferentes variables dentro el microcontrolador:

Tipo	Descripcion
Void	Reservado para la declaración de funciones sin valor de retorno.
Byte	Un número entero del 0 al 255 codificado en 1 byte (8 bits)
Int	Un número entero entre 32,767 y -32,768 codificado en 2 byte (16 bits)
Long	Un entero comprendido entre 2,147,483,647 y – 2,147,483,648 y codificado en 32 bits (equivalente a 4 bytes)
Float	Un número real (con decimales) almacenado en 4 bytes (es decir 32 bits) y comprendido entre 3.4028325E+38 y -3.4028325E+38
Unsigned int	Un número natural (entero positivo) almacenado en 16 bits (2 bytes) y comprendido entre 0 y 65,545

Unsigned long	Un número natural (entero positivo) almacenado en 32 bits (4 bytes) y comprendido entre 0 y 4,294,967,296
Word	Lo mismo que unsigned int
Boolean	Una variable booleana que puede tener solamente dos valores: true (verdadero) o false, ocupa 1 byte
Char	Un carácter ASCII almacenado en 8 bits (un byte). Esto permite almacenar caracteres como valores numéricos(su código ASCII asociado).
Unsigned Char	Este tipo de datos es idéntico al tipo byte explicado arriba. Se utiliza para codificar números de 0 hasta 255. Ocupa 1 byte de memoria.

Dado que en códigos largos no siempre vamos a estar seguros de que cantidad de byte ocupa nuestras variable o para evitar posibles errores podemos usar la función sizeof() la cual nos devolverá el número de bit que ocupa una variable o cadena.

Ejemplo:

```
void loop()
{
  word mamaHuevo = 4646143;
  valor = sizeof(mamaHuevo);

//muestra el valor
  Serial.println(valor);
  delay(500);
}
```

La cual nos debería mostrar en el serial el numero 4

read():

Esta función nos permite leer un byte de la dirección que especifiquemos y se llamara de la siguiente forma `EEPROM.read(dirección)` siendo dirección como máximo el que nos permita nuestro microcontrolador

Codigo de ejemplo:

```
#include <EEPROM.h>

void setup()
{
  Serial.begin(9600);
}
void loop()
{
  int direccion = 5;//que dirreccion que leera
  int valor = EEPROM.read(direccion);

//muestra el valor
  Serial.println(valor);
  delay(500);
}
```

write():

Esta función nos permite escribir un byte en la dirección que especifiquemos y se llamara de la siguiente forma `EEPROM.write(dirección, valor)` siendo dirección como máximo el que nos permita nuestro microcontrolador y valor como máximo 255.

Algo a recordar es que una escritura de EEPROM tarda 3,3 ms en completarse.

Código de ejemplo:

```
#include <EEPROM.h>
void setup()
{
  Serial.begin(9600);
  EEPROM.write(5, 420);//valor escrito
}
void loop()
{
//lectura de lo que escribimos
  Serial.println(EEPROM.read(5));
  delay(500);
}
```

El valor que nos debería mostrar en pantalla es el que agregamos.

Estas 2 funciones serian nuestras bases con las cuales nos permitirán trabajar con la EEPROM, luego de estas existen sus variantes las cuales nos harán la vida más fácil a la hora de trabajar.

update():

Esta función trabaja igual que write() pero tiene la ventaja que solo escribe el valor si es diferente al ya guardado en esa dirección. Esto nos es de mucha ayuda para evitar sobrescribir varias veces el mismo valor sobre una dirección lo que acortaría su vida útil de forma tonta, se llamara de la siguiente forma: `EEPROM.update(direccion, valor)`

get():

Esta función trabaja igual que read() pero nos da la ventaja de que nos permite trabajar con cualquier tipo de dato, por ejemplo en caso de querer leer un int que ocupa 2 bytes de información leerá desde la dirección que le digamos a 2 bytes a la derecha. Se llamara de la siguiente forma `EEPROM.get(dirección, variableDeGuardado)` siendo el tipo de variable que incluyamos la que nos indique que espacio deberá leer y guardara ahí lo que lea

Código de ejemplo:

```
void setup(){

  float variable = 0.00f;
  int direccion = 0;

  Serial.begin( 9600 );
  EEPROM.get( direccion, variable );
  Serial.println( variable, 3);

}
```

put():

Esta función trabajara igual que read() pero algo a destacar es que esta función usara update() por lo cual debemos despreocuparnos sobre la reescritura, al igual que get() nos permite trabajar con cualquier tipo de dato y usara la misma lógica, será llamado de la siguiente forma `EEPROM.put(direccion, variable)` siendo el tipo de variable que incluyamos la que indique que cuantos bytes ocupara la información guarda

Código de ejemplo:

```
void setup() {
  Serial.begin(9600);
  float var = 123.456f;
  int direccion = 0;
  float lectura;
//guardado de información
  EEPROM.put(direccion, var);
}
void loop() {
//lectura de lo que escribimos
  EEPROM.get(dirreccion,lectura);
  Serial.println(lectura);
  delay(500);
}
```

Siendo que nos debería devolver en el serial 123.456

EEPROM [ ]:

Esta función es bastante simple y nos permite usar a la EEPROM como si fuera una matriz, las celdas de la EEPROM se pueden escribir y leer directamente con este comando, la llamaremos de la siguiente forma `EEPROM[dirección]`

Código de ejemplo:

```
byte escritura = 5;
byte lectura = 5;

void setup(){
  EEPROM[0] = escritura;
  pinMode(13, OUTPUT);
}

void loop(){
  if(EEPROM[0] == lectura)
    digitalWrite(13, !digitalRead(13));
    delay(500);
}
```

Si se observa el el arduino podra ver que el led que se indica con una L el cual es una led de prueba empezara a parpadear indicando que EEPROM[0] = 5, de esto podemos sacar que EEPROM[0] funciona igual que cualquier variable común, y segundo de que EEPROM[ ] solo funciona para una celda determinada a diferencia de get() y put()

Codigo de ejemplo general:

Se busca colocar una clave en mitad del espacio máximo de la EEPROM

```
#include <EEPROM.h>
void setup() {

  Serial.begin(9600);
  unsigned long clavePrueba = 854653;
  unsigned long lecturaClave = 0;
  int direccion = sizeof(clavePrueba); //Move address to the next byte after float 'f'.

  Serial.println(direccion);
  EEPROM.put((1024 / 2) - direccion, clavePrueba);
  EEPROM.get((1024 / 2) - direccion, lecturaClave);

  Serial.println(lecturaClave);
}
void loop(){
}
```