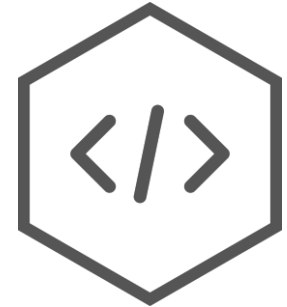




APLICACIONES DE IoT



UNIDAD I. ADQUISICIÓN Y PROCESAMIENTO DE DATOS

El alumno manipulará sensores y actuadores conectados a hardware abierto para procesar y almacenar datos.



1.1 ALMACENAMIENTO DE DATOS

DSM

UNIDAD I

ENERO 2022

UTM



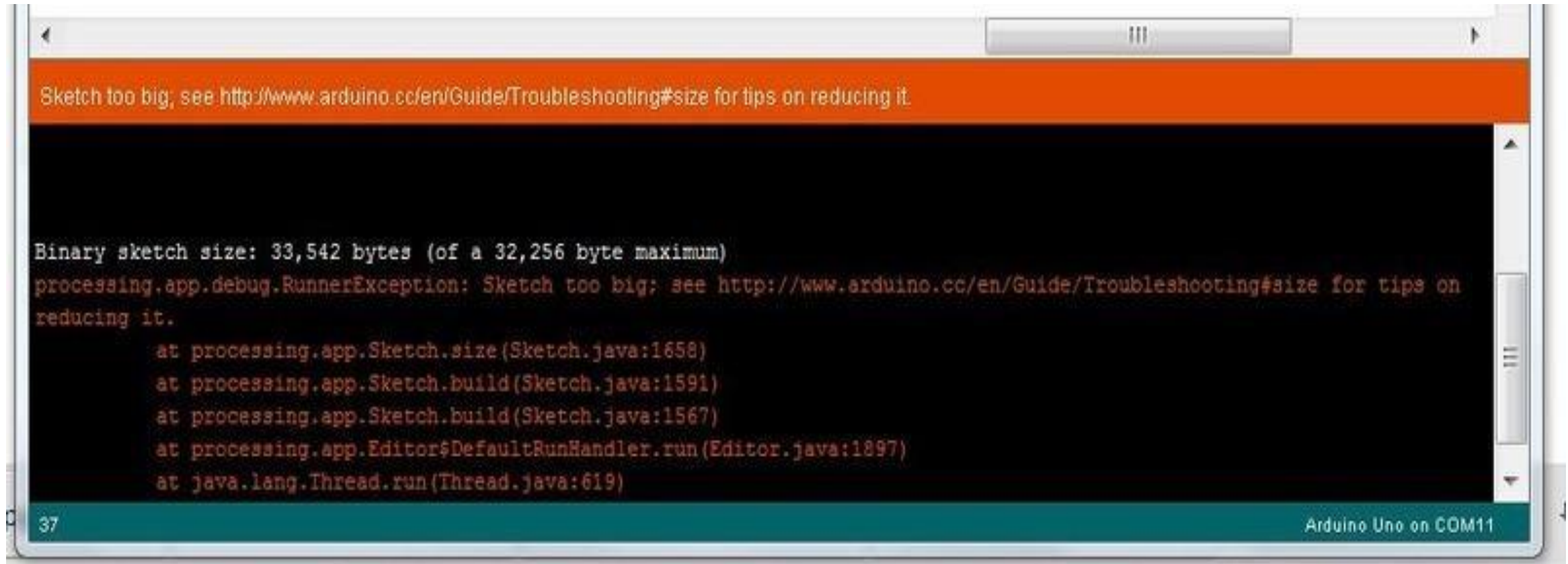
AGEND

- Introducción
- Arquitecturas de memoria
- Memorias en microcontroladores
- Memorias en Arduino™
- Midiendo la memoria utilizada en Arduino™
- Conclusión

INTRODUCCIÓN

- ¿Sabías que las tarjetas Arduino tienen una cantidad de memoria limitada, y que en algún momento puede ser llenada en su totalidad?
- Síntomas: cuando el compilador dice que tu “sketch” es demasiado grande. Otro, que el programa puede que cargue, pero que no funcione.

SÍNTOMAS



The screenshot shows the Arduino IDE's serial monitor window. At the top, an orange banner displays the message: "Sketch too big, see <http://www.arduino.cc/en/Guide/Troubleshooting#size> for tips on reducing it." Below this, the console text reads: "Binary sketch size: 33,542 bytes (of a 32,256 byte maximum)" followed by "processing.app.debug.RunnerException: Sketch too big; see <http://www.arduino.cc/en/Guide/Troubleshooting#size> for tips on reducing it." A stack trace follows, listing several lines of code from the processing.app and java.lang packages. The bottom status bar of the IDE shows "37" on the left and "Arduino Uno on COM11" on the right.

```
Sketch too big, see http://www.arduino.cc/en/Guide/Troubleshooting#size for tips on reducing it.

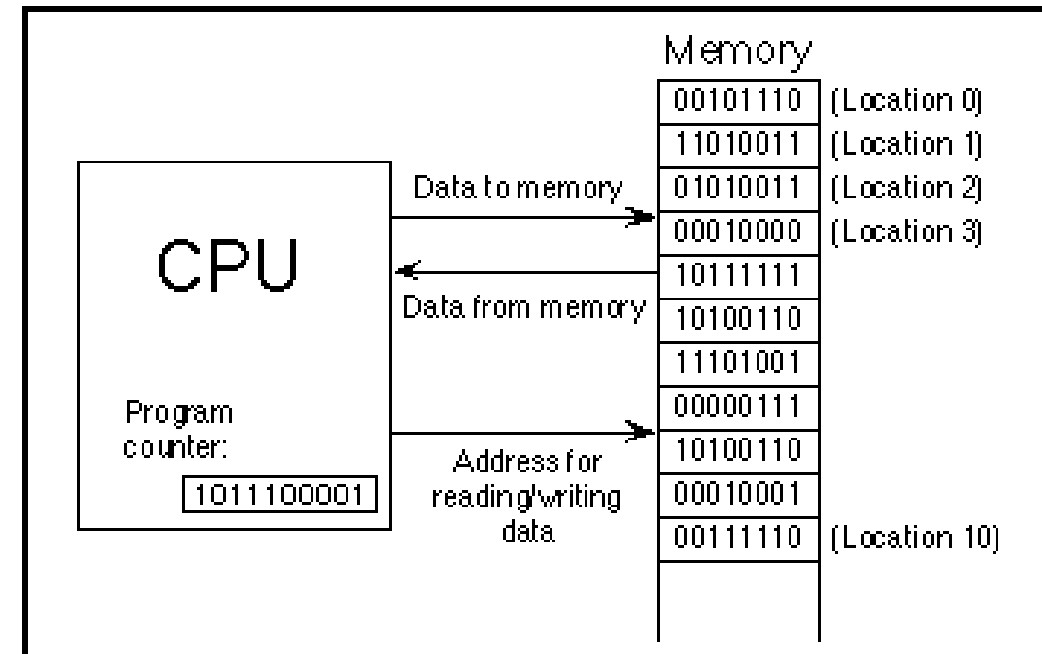
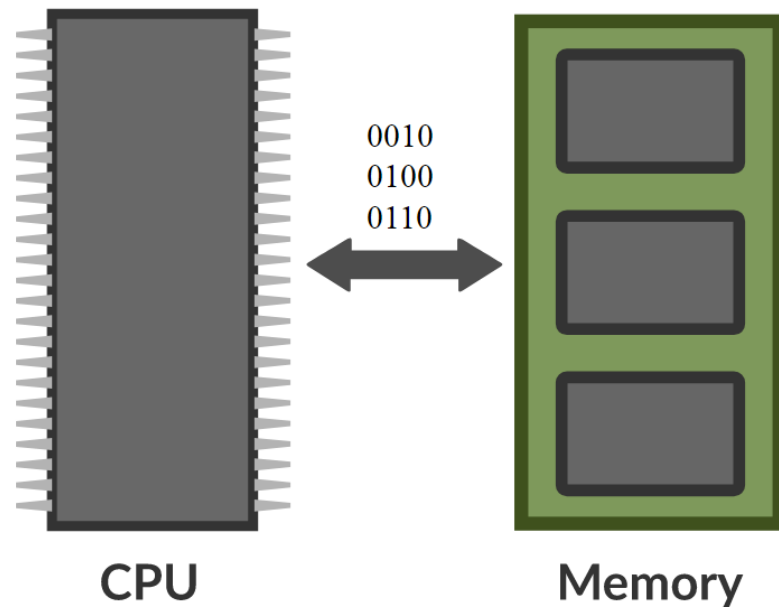
Binary sketch size: 33,542 bytes (of a 32,256 byte maximum)
processing.app.debug.RunnerException: Sketch too big; see http://www.arduino.cc/en/Guide/Troubleshooting#size for tips on
reducing it.
    at processing.app.Sketch.size(Sketch.java:1658)
    at processing.app.Sketch.build(Sketch.java:1591)
    at processing.app.Sketch.build(Sketch.java:1567)
    at processing.app.Editor$DefaultRunHandler.run(Editor.java:1897)
    at java.lang.Thread.run(Thread.java:619)

37 Arduino Uno on COM11
```

ARQUITECTURAS DE MEMORIA: HARVARD V/S PRINCETON

- En los primeros días de la computación electrónica, dos arquitecturas de procesador/memoria diferentes surgieron:
- **Princeton**: desarrollada para la computadora ENIAC, utiliza las mismas líneas de memoria tanto para el programa como para almacenamiento
- **Harvard**: por la Harvard Mark 1, usaba memoria para datos y memoria para aplicaciones por separado.

CPU <-> MEMORY



MEMORIAS EN MICROCONTROLADORES

- Los microcontroladores, como los que se incluyen en las tarjetas **Arduino**, están diseñados para aplicaciones embebidas. un procesador embebido típicamente se enfoca en realizar una tarea en específico de manera confiable, eficiente y a un costo mínimo.
- El modelo **Harvard** resulta una acertada opción para aplicaciones embebidas, como el Atmega328 presente en el Arduino UNO, el cual presenta una arquitectura Harvard casi pura. Los programas se guardan en la memoria **flash** y lo datos en la memoria **SRAM**.
- Una diferencia entre microcontroladores y una PC es la cantidad de memoria disponible. El Arduino UNO solo tiene 32K bytes de memoria Flash y 2K bytes de memoria SRAM. 100.000 veces menos memoria que la incluida en PC.

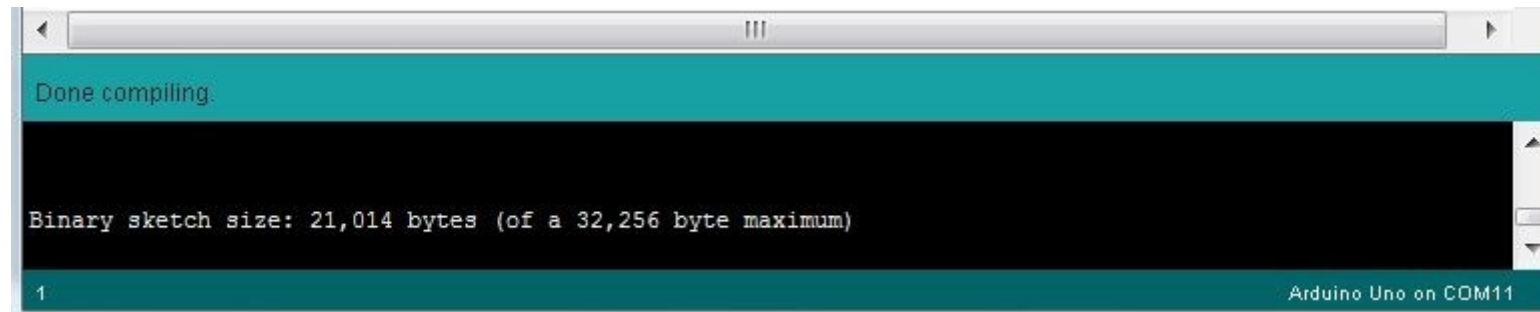
MEMORIAS EN ARDUINO

Existen 3 tipos de memoria en Arduino:

1. Memoria Flash o memoria de programas.
2. SRAM.
3. EEPROM.

MEMORIA FLASH

- La memoria flash no volátil, donde grabamos el sketch (incluido el bootloader).
- La memoria flash usa la misma tecnología encontrada en los pen drives y memorias SD.
- Cada vez que compila un sketch se puede medir la memoria disponible.

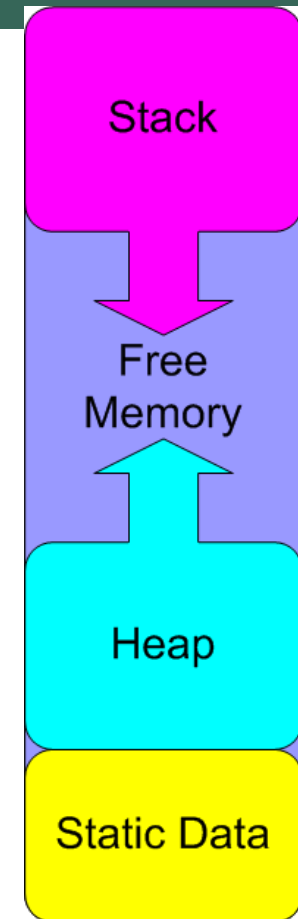


```
Done compiling.  
  
Binary sketch size: 21,014 bytes (of a 32,256 byte maximum)  
  
1 Arduino Uno on COM11
```

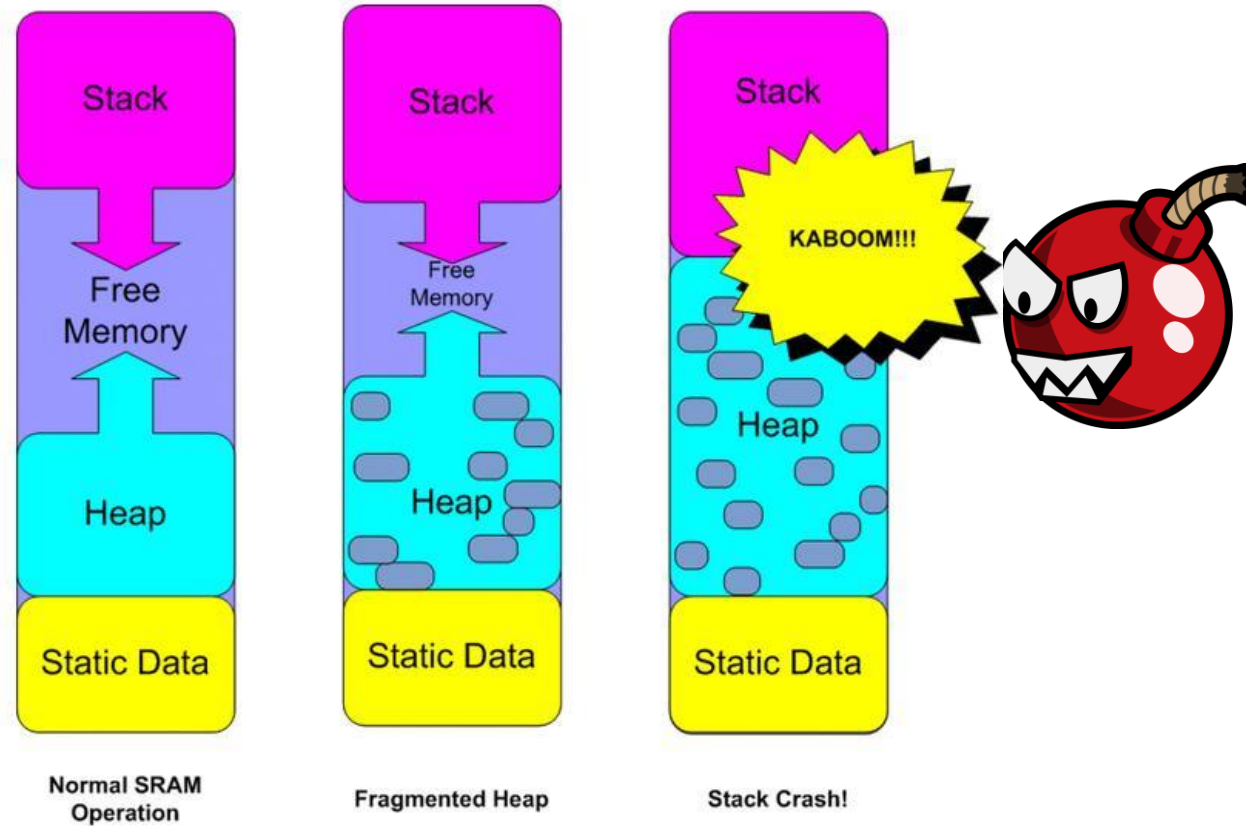
SRAM

La memoria SRAM o "Static Random Access Memory". Volátil, donde se almacenan las variables durante el funcionamiento. Está compuesta por:

1. Static: es un bloque de memoria reservado de SRAM para las variables globales y estáticas de un programa. Los valores iniciales son copiados desde la memoria Flash cuando el programa inicia.
2. Heap: se usa para datos dinámicos y crece desde la parte superior de los datos estáticos a medida que más datos se asignan a la memoria.
3. Stack: se usa para variables locales y para mantener un registro de las interrupciones y las llamadas de función. El Stack crece desde el tope de la memoria hasta el Heap. Cada interrupción, llamada de función o direccionamiento de variable, hace que crezca el Stack.



SRAM



EEPROM

- La EEPROM es otra forma de memoria no volátil que puede ser leída y escrita desde un programa en ejecución. Solo puede ser leída byte a byte así que puede ser algo incomoda de usar. También es mucho más lenta que la SRAM.
- La memoria EEPROM es un recurso más escaso que el resto de memorias. La mayoría de modelos de Arduino disponen de 1KB, mientras que el Mega tiene 4KB.
- Se puede usar para almacenar un dato que más tarde podemos recuperar, pero necesitaremos saber la dirección en la que está guardada la variable, así como su tipo.

COMPARACIÓN DE MEMORIAS EN ARDUINO

Arduino	Processor	Flash	SRAM	EEPROM
UNO, Uno Ethernet, Menta, Boarduino	Atmega328	32K	2K	1K
Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora	Atmega 32U4	32K	2.5K	1K
Mega, MegaADK	Atmega2560	256K	8K	4K

MIDIENDO LA CANTIDAD DE MEMORIA USADA

- Memoria **Flash**: medir la cantidad de memoria flash en uso es trivial como ya vimos, ya que el compilador muestra ese dato cada vez que se compila.
- El uso de memoria **SRAM**: es más dinámico y por lo tanto mas difícil de medir. La función `free_ram()`, es una forma de hacer esto. Se puede agregar esta función en partes de tu código para que reporte la cantidad de SRAM libre.
- De la memoria **EEPROM**: se puede tener un control de un 100% ya que se debe escribir y leer cada byte en una dirección en específico, así que no hay excusa para no saber exactamente cuántos están en uso.

RESOLVIENDO PROBLEMAS DE MEMORIA

- La memoria es un recurso finito de estos pequeños procesadores y algunas aplicaciones son de plano muy grandes para un Arduino, pero la mayoría de los códigos tienen espacio para algo de optimización, así que si un programa está llegando a niveles elevados de memoria probablemente con algo de optimización pueda funcionar de forma correcta nuevamente.

OPTIMIZANDO MEMORIA DE PROGRAMA

- Cuando compilas tu código, el programa IDE te mostrara que tan grande es la imagen de tu código. Si es que has llegado o excedido el espacio disponible, algunas de las siguientes optimizaciones te ayudaran a disminuir tu memoria utilizada.

OPTIMIZANDO MEMORIA DE PROGRAMA..

- **Remover código muerto:** Si tu proyecto es una combinación de código de varias fuentes, entonces existe la posibilidad que partes de él no se estén usando y puedan ser eliminadas para ahorrar espacio. ☹
 - Librerías en desuso: ¿Se están usando todas las variables #include?
 - Funciones en desuso: ¿Se está llamando a todas las funciones?
 - Variables en desuso: ¿Se están usando todas las variables declaradas?
 - Código inalcanzable: ¿Existen expresiones en el código que nunca serán ciertas?

OPTIMIZANDO SRAM

- La SRAM es el recurso más preciado de Arduino y la causa más común de los problemas de memoria en el mismo, además de ser los más difíciles de diagnosticar.
- Tips que se pueden hacer para reducir la cantidad de SRAM usada:
- Remover variables en desuso
- Manejo de Strings
- Mover datos constantes a PROGMEM
- Reducir tamaños de buffer
- Variables globales y estáticas
- Variables locales
- Direcciones dinámicas

USO DE EEPROM

- La EEPROM es una memoria de almacenamiento que funciona bien para guardar datos de calibración o constantes que no son prácticos de guardar en memoria Flash.
- Es inusual quedarse sin memoria EEPROM.
- Para usar la memoria EEPROM es necesario importar la librería EEPROM:

```
#include <EEPROM.h>
```

- Esta librería entrega dos funciones:
 - `uint8_t read(int)` //Lee un byte de la dirección específica de la memoria EEPROM.
 - `void write(int, uint8_t)` //Escribe un byte en la dirección específica de la memoria EEPROM.

VIDEO

- https://youtu.be/VAzjiYY_oH0
- ¿Qué tipo de memoria menciona el video?

CONCLUSIÓN

- La mayor de las veces no le damos la importancia debida al manejo de la memoria de los microcontroladores ya que pensamos que es un recurso ilimitado.
- Además vimos que existen tres tipos de memorias en Arduino y que además pueden ser volátiles y no volátiles.
- La recomendación final es optimizar el código para no tener problemas de funcionamiento en nuestros proyectos de IoT.

REFERENCIAS

Aprovechar al máximo la memoria de Arduino.

<https://cursos.mcielectronics.cl/2019/06/18/aprovechar-al-maximo-la-memoria-de-tu-arduino-2>

Measuring memory usage.

<https://learn.adafruit.com/memories-of-an-arduino/measuring-free-memory>

Blog Luis Llamas.

<https://www.luisllamas.es/guardar-variables-entre-reinicios-con-arduino-y-la-memoria-no-volatil-eeeprom/>

PREGUNTAS Y COMENTARIOS

