

Sistemas Embebidos

Materia Optativa de grado

Ing. Electrónica

Ing. Telecomunicaciones

Presentación de la cátedra

Cursada

- **Docentes:**
 - Dr. Ing. Federico Guerrero
 - Dr. Ing. Marcelo Haberman
- **Horarios:**
 - Teórico/Práctico: Viernes 8:30 a 12:30.
 - Práctica/Consultas: Horario a definir
- **Desarrollo de las clases**
 - Todas en laboratorio Barcala
 - Uso de kit con microcontrolador y accesorios
- **Evaluación**
 - Parcial y recuperatorio módulos 1 y 2 + Flotante
 - Parcial usando Moodle y programando Kit
 - 1 trabajo especial por módulo con defensa
- **Comunicación**
 - Todas las novedades se comunicarán en la clase o en la página
 - www.ing.unlp.edu.ar/catedras/E1504/
 - Allí se encuentra publicado el cronograma tentativo.
 - El cronograma incluye las fechas de evaluación y de entrega de los trabajos especiales.



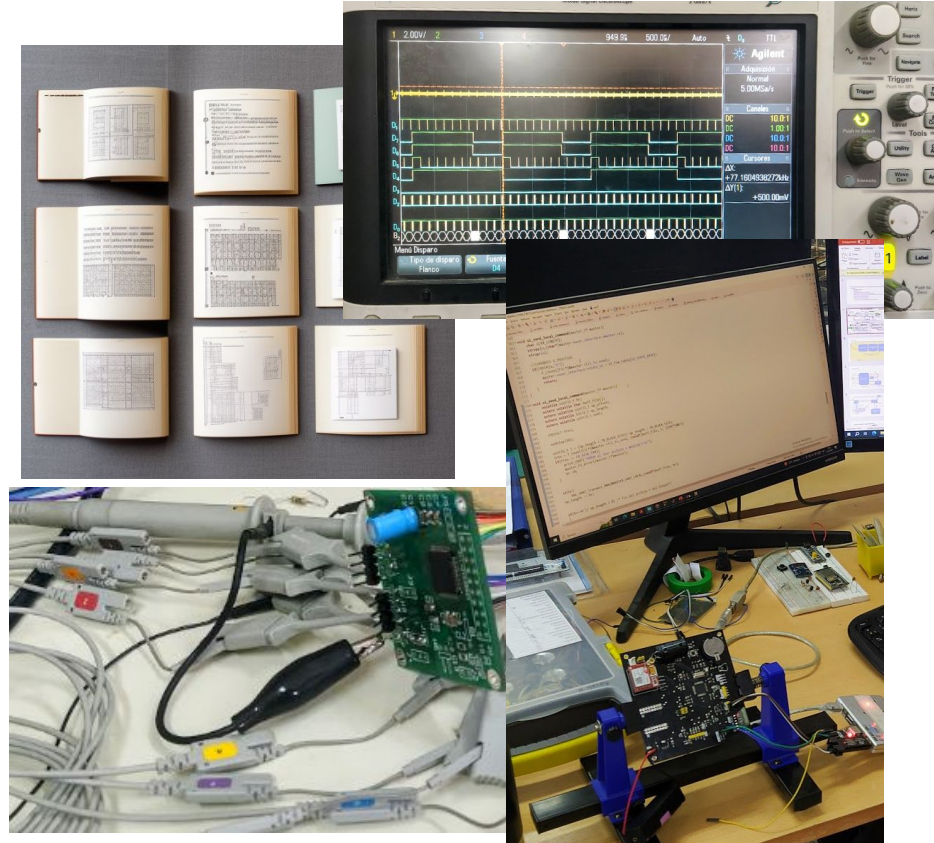
Objetivos del curso

Interpretar la documentación (hojas de datos, manuales de referencia, documentación de software) de los **microcontroladores y entornos de desarrollo** para diseñar sistemas que cumplan con **restricciones de tiempo real**

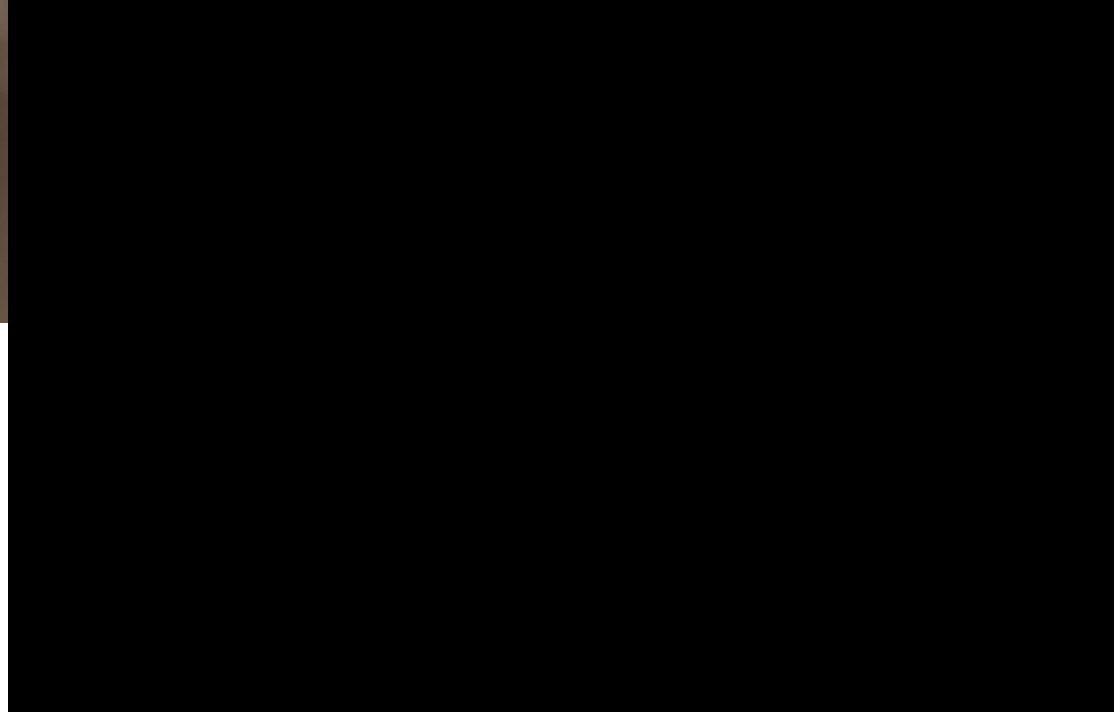
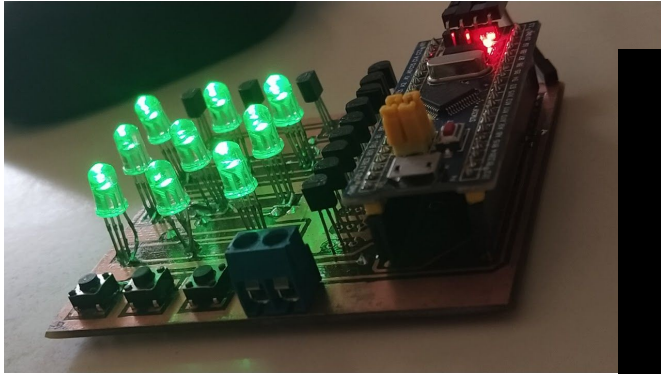
Emplear **técnicas de depuración y utilizar instrumentos de diagnóstico** de sistemas digitales para evaluar y corregir implementaciones de software y hardware durante el **desarrollo de un sistema embebido**

Implementar **estrategias de organización del software** de sistemas embebidos desde estructuras básicas hasta sistemas operativos de tiempo real para crear software capaz de administrar efectivamente un sistema embebido basado en microcontroladores de 32 bits.

Integrar periféricos internos y externos de microcontroladores para implementar sistemas embebidos con **capacidades de interacción con el usuario y otros sistemas**.



Trabajos anteriores



Sistemas embebidos: Aplicaciones y estado del arte

Definición de Sistema Embebido

Un sistema embebido (SE) es cualquier dispositivo que incluye capacidad de cómputo para un propósito específico (en oposición al propósito general).

En general, la unidad de cómputo está incluida en un sistema electrónico o electromecánico.

Lo diferenciamos de lo que llamamos propiamente “computadora” que ofrece al usuario una plataforma de cómputo genérica donde pueden correrse los programas que se quieran.

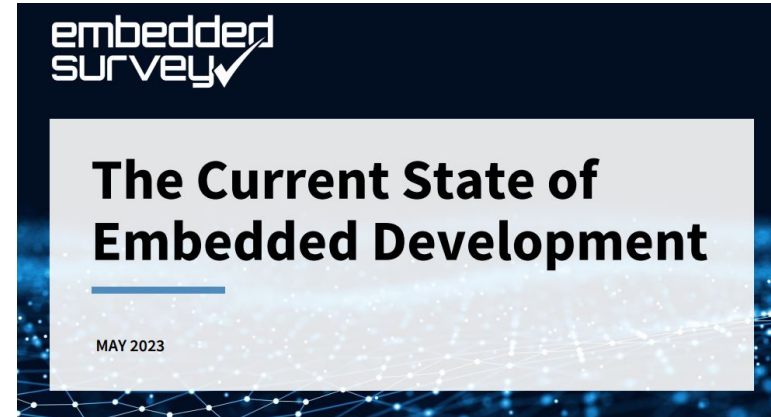


Estado del arte

Encuesta a nivel mundial sobre los sistemas embebidos realizada por embedded.com

Regiones incluidas:

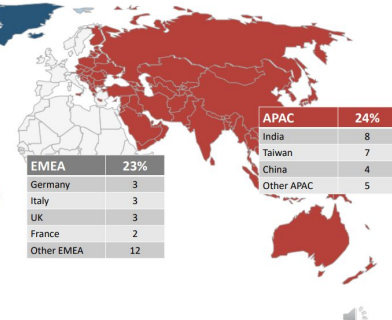
- América
- EMEA: Europa, Oriente Medio y África
- APAC: Asia-Pacífico



Methodology

Online survey

| | |
|-----------------|-----------------|
| Americas | 53% |
| North America | 49 |
| South America | Estado del arte |
| Central America | 1 |

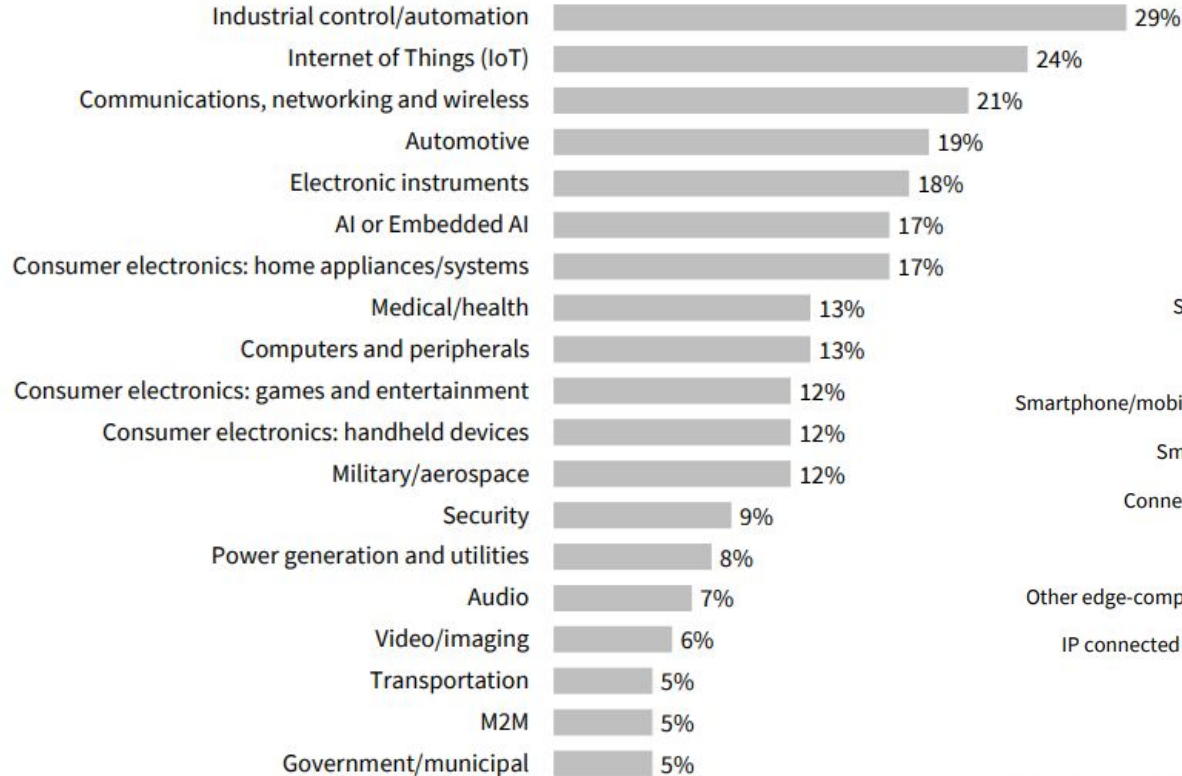


| | |
|------------|-----|
| EMEA | 23% |
| Germany | 3 |
| Italy | 3 |
| UK | 3 |
| France | 2 |
| Other EMEA | 12 |

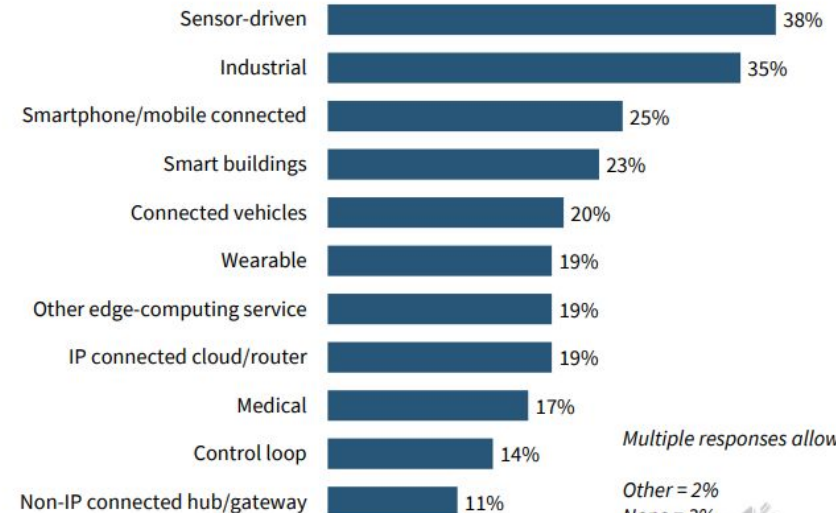
| | |
|------------|-----|
| APAC | 24% |
| India | 8 |
| Taiwan | 7 |
| China | 4 |
| Other APAC | 5 |

- **Field Dates:** Feb 9 to March 3, 2023
- Respondents screened for **engineering** responsibilities and **experience with embedded applications**
- Results based on **655 responses** (confidence level +/- 3.7%)

Aplicaciones



Types of Applications for IoT



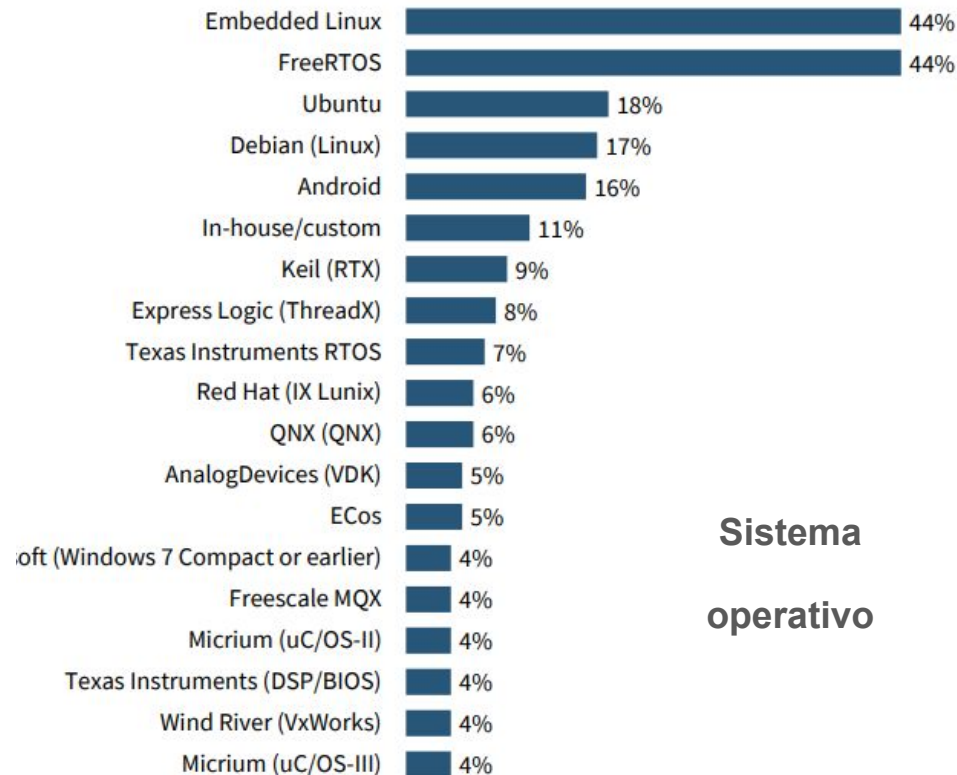
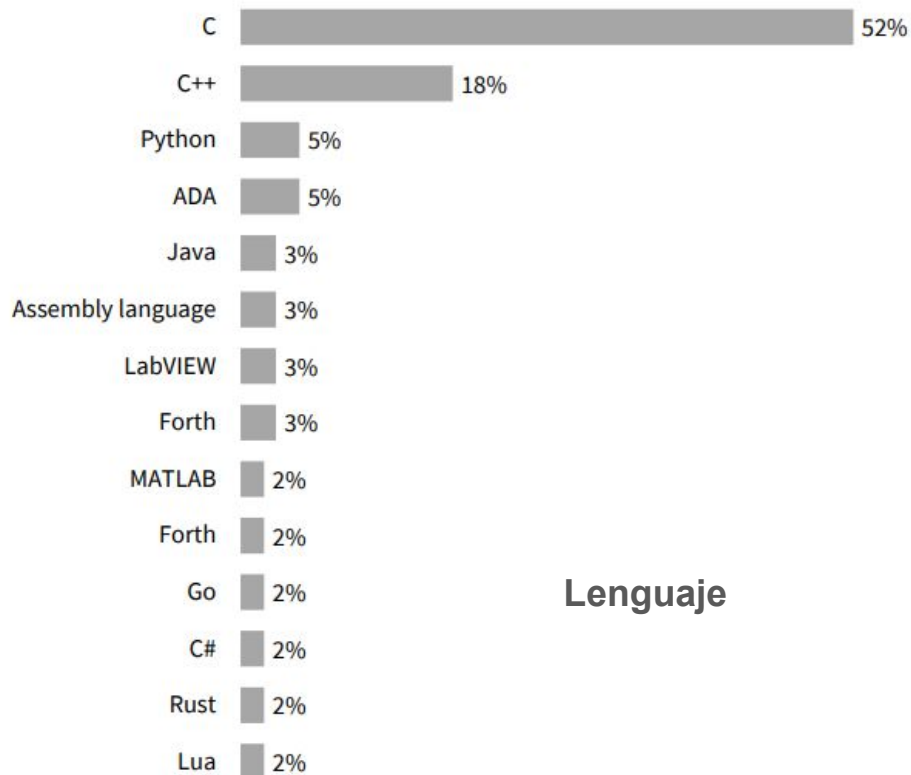
Multiple responses allowed

Other = 2%

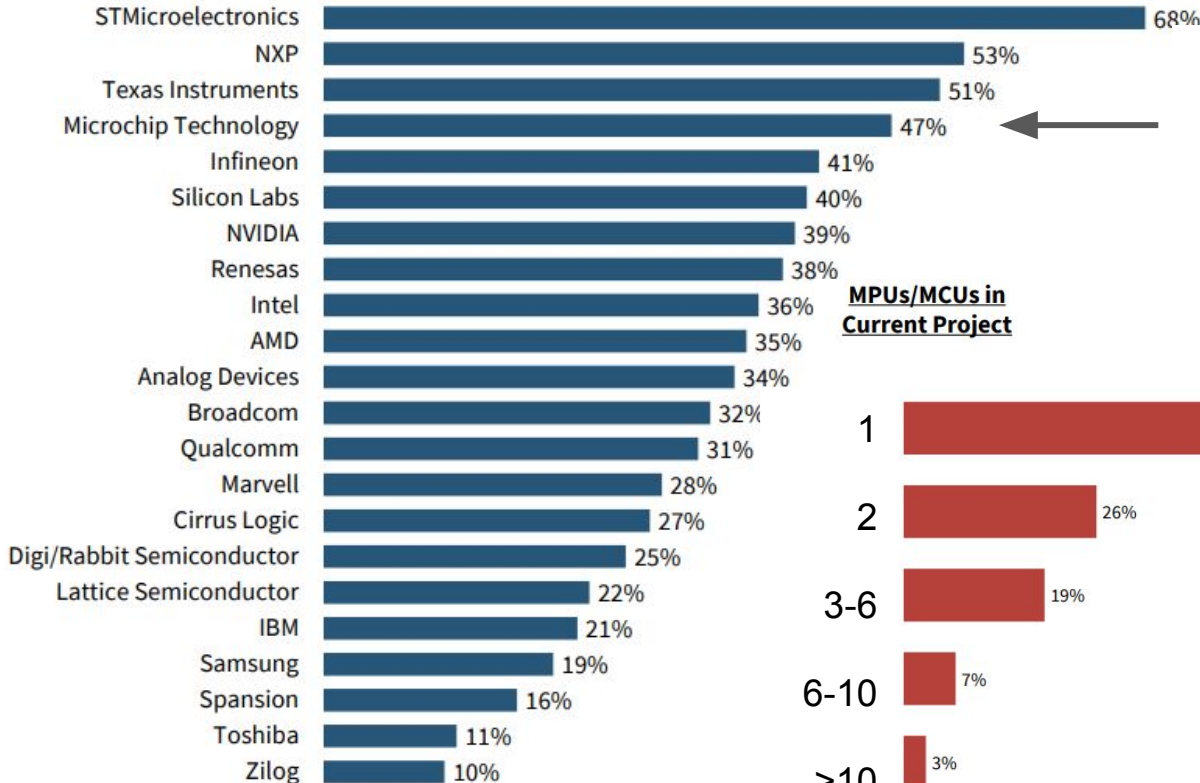
None = 3%



Firmware

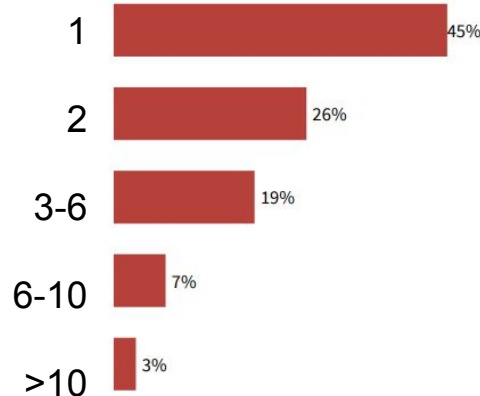


Microcontroladores

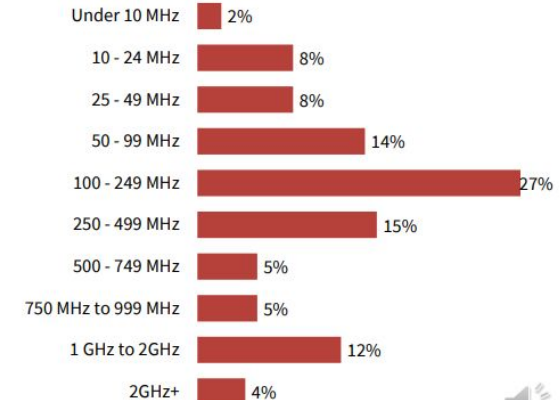


Qué proveedor considera usar en el futuro

MPUs/MCUs in Current Project



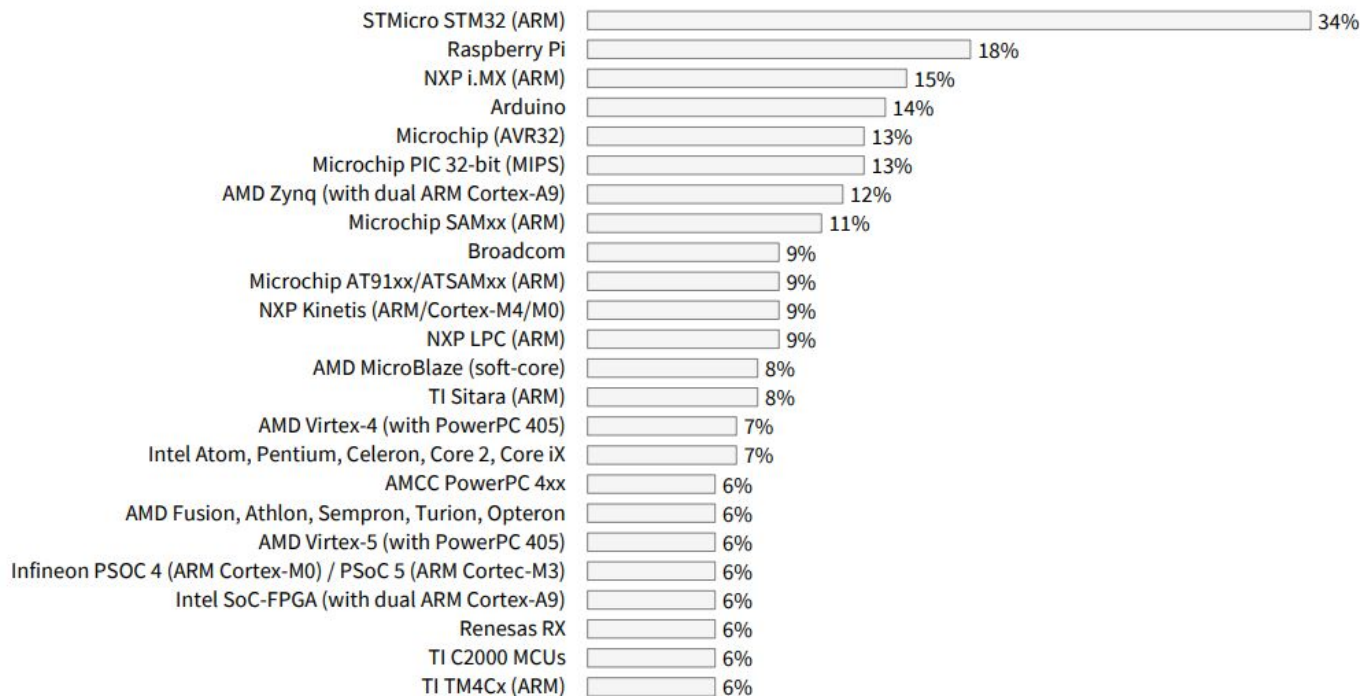
Main Processor Clock Rate



Microcontroladores

Future consideration of 32-bit processor families

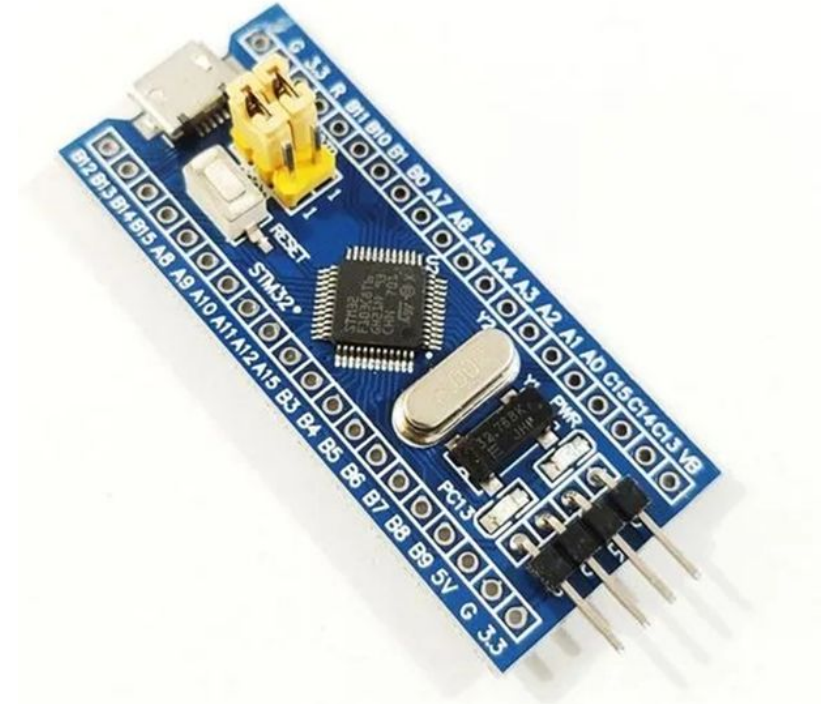
STMicro's STM32 is most widely considered, followed by Raspberry Pi, NXP's i.MX, Arduino and Microchip's AVR32



Plataforma de trabajo en la cátedra

Microcontrolador y placa de desarrollo

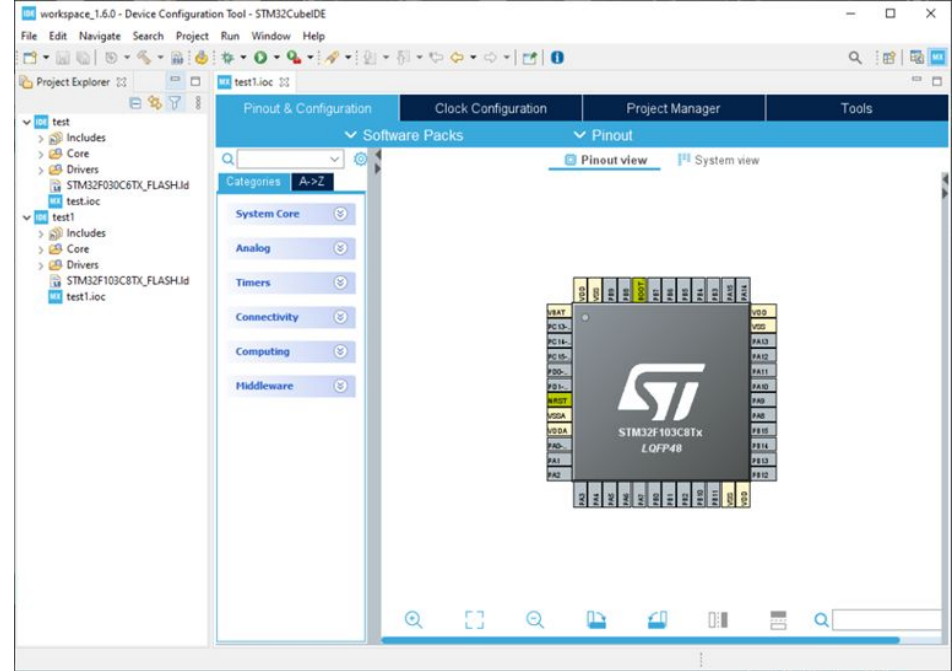
- STM32 Blue Pill
 - STM32F103C8T6 ARM Cortex M3
 - Frecuencia máxima de 72MHz
 - 64kB de flash (algunos con 128KB)
 - 20kB de RAM
 - 37 GPIO
 - 2 ADC de 12 bits y 16 canales (10 en blue pill)
 - 7 timers (3 de 16bits, 1 PWM, 2WDT, 1 del sistema)
 - 3 USARTs
 - 2 I2C - 2 SPI – CAN 2.0 – USB
 - DMA 7 canales (con timers, ADC, SPI, I2C y USARTs)
 - RTC



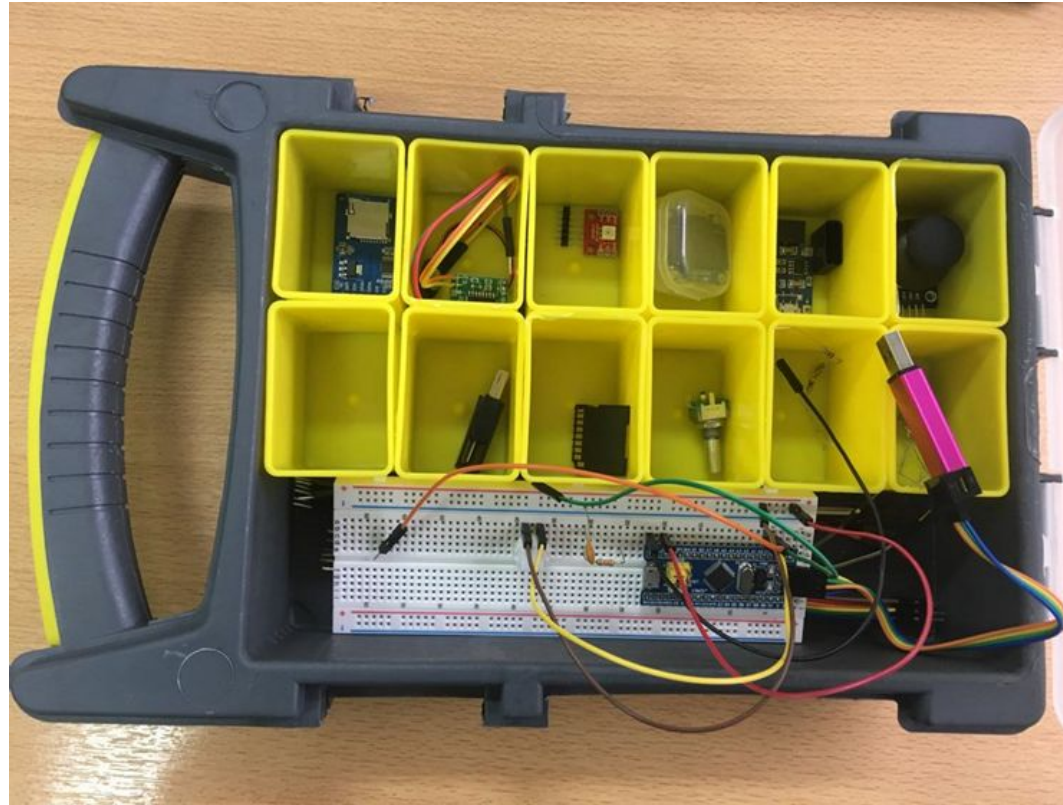
Entorno de desarrollo

- STM32CubeIDE
 - Herramienta profesional
 - Acceso a facilidades de depuración avanzadas
 - Herramientas de configuración gráfica útiles
 - Basado en Eclipse

<https://www.st.com/en/development-tools/stm32cubeide.html>



Nuestro Kit



¿Microcontrolador?

Estructura básica



- El microcontrolador se compone de un procesador con sus registros, y periféricos que incluyen memoria.
- El procesador puede ejecutar un set de instrucciones y en general todo lo que hace es leer, modificar y escribir registros
- los periféricos se administran, también, leyendo y escribiendo registros

Microcontrolador

Procesador

R0

R1

R2

...

Búsqueda,
Decodificación
y Ejecución de
Instrucciones

ALU

Periférico 1

RP0

RP1...

Periférico n

RP0

RP1...

Memoria



Programación

```
L0:  MOV    R1, #a
      MOV    R2, #b

L1:  LD      R3, (R1)
      CMP    R3, #0
      BNE    L3

L2:  MOV    R4, #1
      JMP    L4

L3:  MOV    R4, #0

L4:  ST      (R2), R4
```



“Memoria”

| Dirección | Contenido |
|------------|-----------|
| 0x00000200 | |
| 0x00000201 | |
| 0x00000202 | |
| 0x00000203 | |
| 0x00000204 | |
| 0x00000205 | |
| 0x00000206 | |

Programación

```
49 js.src = "/connect.facebook.net/en_US/sdk.js#xfbml=1&version=v2.6&appId=1000000000000000";
50 fjs.parentNode.insertBefore(js, fjs);
51 }(document, 'script', 'facebook-jssdk'));</script>
52 <div id="page" class="site">
53 <a class="skip-link screen-reader-text" href="#content" title="Skip to content">Skip to content</a>
54
55 <header id="masthead" class="site-header">
56 <div class="site-branding">
57 <div class="masthead">
58 <?php if(!is_home())>
59 <a href="#" id="home-link" class="fa fa-home">Home</a>
60 <?php elseif(is_home())>
61 <a href="#" id="home-link" class="fa fa-home">Home</a>
62 <?php >
63 </div>
64 <div class="logo">
65 <a href="#">
66 
67 </a>
68 </div>
69 <div class="search-box">
70 <?php get_search_form()>
71 </div>
72 <div class="submit-bn hidden">
73 <a href="#">Submit</a>
74 </div>
75 <div class="user-info null-right"></div>
```



#a
#b
(R1)
#0

#1

#0



“Memoria”

| Dirección | Contenido |
|------------|-----------|
| 0x00000200 | |
| 0x00000201 | |
| 0x00000202 | |
| 0x00000203 | |
| 0x00000204 | |
| 0x00000205 | |
| 0x00000206 | |

Lenguaje C en Sistemas Embebidos

Algunos temas para repasar y algunos temas nuevos:

- **Acceso al bajo nivel**
 - Tipos de datos estandarizados
 - Sistemas de representación
 - Operadores a nivel de bits
- **Lazo entre el bajo y alto nivel**
 - Estructuras
 - Macros
- **Herramientas de alto nivel**
 - Enum
 - Pasaje por referencia
 - Librerías
 - Modificadores: volatile, extern, const, static
 - Punteros a funciones

Sistemas de representación

- Números enteros
 - Positivos: binario puro
 - Negativos: complemento a 2
- Números reales
 - float
 - double
- Cadenas de caracteres
 - Cada caracter es un byte ascii
 - Terminan en '\0'

Sistemas de representación

- Donde sea conveniente, usaremos tipos de datos enteros menos “ambiguos”
- En la PC, hay que incluir la librería `stdint.h` donde están definidos.
- En el firmware no será necesario, ya se incluyen las librerías necesarias.
- Un microcontrolador tiene un “ancho de palabra” que es el tamaño de sus registros:
 - 8 bits
 - 16 bits
 - 32 bits
- Para verlo en binario puro:
 - `uint8_t`
 - `uint16_t`
 - `uint32_t`

```
#include <stdint.h>

int main()
{
    int32_t x;
    uint16_t y;

    char c;
    int8_t d;

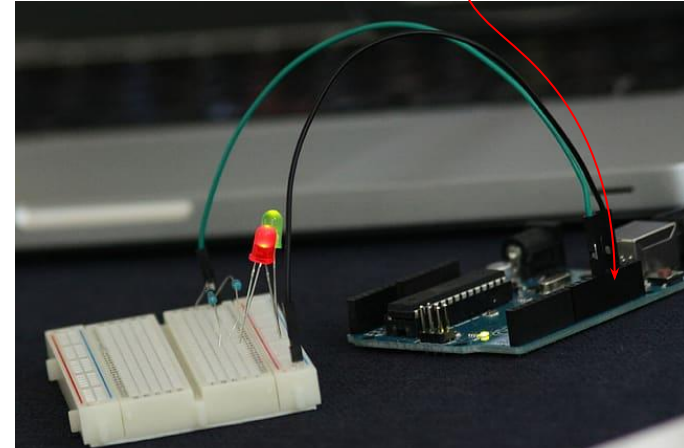
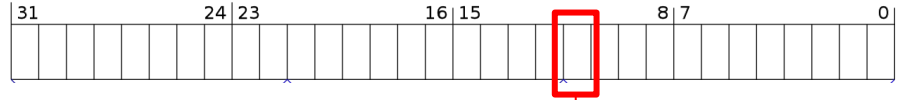
    float v;
    double w;

    return 0;
}
```


Operaciones bit a bit

- Sirven para manipular variables pero **“bit a bit”**
- Los operadores son `&`, `|`, `^`, `~`, `>>`, `<<`
- Ejemplo:

```
uint8_t x = 0xF0;  
uint8_t y = ~x; // y=0x0F
```
- Muchas veces interesa cambiar un único bit sin alterar el resto: Se usan **“máscaras”**
- Por ejemplo: El byte 2 del registro de la posición de memoria 0x00004121 controla 8 pines (0 al 7) del puerto B del GPIO.
- Se desea encender el LED conectado al pin 3 del puerto B sin modificar el estado de los otros leds, poniendo en 1 ese bit
- ```
uint32_t *reg = 0x00004121; //Asignar dir
uint32_t mask = 0x00000800; // máscara
uint32_t mask = 1<<11; // >mas fácil
*reg = (*reg) | mask; // encender
*reg = (*reg) & ~mask; // apagar
*reg |= mask; // encender
```



# Estructuras, Macros

- Los `#define` sirven para crear “etiquetas” que pueden usarse en lugar de números para hacer más legible el código

```
#define NUM_REPS 1230
```

- Pero también pueden usarse para crear pequeñas funciones

```
#define DOBLE(x) (2*x)
...
y = DOBLE(34);
```

- Este “truco” aparece en las librerías

```
#define TIM_BASE_ADDR 0x0002A000
```

```
#define TIM1_OFFSET 0x00
```

```
#define TIM2_OFFSET 0x0F
```

```
#define TIM3_OFFSET 0x1E
```

```
typedef struct{
 uint32_t control_reg;
 uint32_t counter_reg;
 uint32_t reload_reg;
} TIMERN;
```

```
#define SET_BIT(REG,n) (REG |=1<<n)
```

```
TIMERN * h_tim2 = TIM_BASE_ADDR + TIM2_OFFSET;
SET_BIT(h_tim2->control_reg, 8);
```

# Archivos separados

- Para organizar el código en C se usa compilación en archivos separados
- Por ejemplo en la PC siempre incluimos el archivo `<stdio.h>` para poder usar las funciones `printf()` o `scanf()`
- Esto no sólo ordena el código sino que lo hace independiente: nosotros trabajamos en un archivo y usamos lo que hay en otros sin la necesidad de modificarlos
- Una “librería” es simplemente un archivo donde están programadas varias funciones y tipos de datos que podemos usar al incluirlos en nuestro proyecto

# Archivos separados

```
main.c X contador.h X contador.c X
1 #include <stdio.h>
2 #include "contador.h"
3 int main() {
4 cont_init(500);
5 while(1){
6 cont_incremento();
7 if(cont_leer_flag() == 1){
8 buzz();
9 }
10 }
11 return 0;
12 }
13
14 void buzz(void) {}
15
```

```
main.c X contador.h X contador.c X
1 /*
2 * contador.h
3 */
4 #ifndef INC_CONTADOR_H_
5 #define INC_CONTADOR_H_
6
7 void cont_init(int cuentas);
8 void cont_incremento(void);
9 void cont_decremento(void);
10 int cont_leer_flag(void);
11
12 #endif /* INC_CONTADOR_H_ */
13
```

# Archivos separados

| main.c X | contador.h X                 | contador.c X |
|----------|------------------------------|--------------|
| 1        | /*                           | 1            |
| 2        | * <u>contador.h</u>          | 2            |
| 3        | */                           | 3            |
| 4        | #ifndef INC_CONTADOR_H       | 4            |
| 5        | #define INC_CONTADOR_H       | 5            |
| 6        |                              | 6            |
| 7        | void cont_init(int cuentas); | 7            |
| 8        | void cont_incremento(void);  | 8            |
| 9        | void cont_decremento(void);  | 9            |
| 10       | int cont_leer_flag(void);    | 10           |
| 11       |                              | 11           |
| 12       | #endif /* INC_CONTADOR_H */  | 12           |
| 13       |                              | 13           |

| main.c X | contador.h X | contador.c X |
|----------|--------------|--------------|
|          | 1            | 1            |
|          | 2            | 2            |
|          | 3            | 3            |
|          | 4            | 4            |
|          | 5            | 5            |
|          | 6            | 6            |
|          | 7            | 7            |
|          | 8            | 8            |
|          | 9            | 9            |
|          | 10           | 10           |
|          | 11           | 11           |
|          | 12           | 12           |
|          | 13           | 13           |
|          | 14           | 14           |
|          | 15           | 15           |
|          | 16           | 16           |

```

17 void cont_incremento(void){
18 i++;
19 if(i>=n_cuenta){
20 i = 0;
21 cont_flag = 1;
22 }
23 }
24
25 void cont_decremento(void){
26 if(i>0){
27 i--;
28 }
29 }
30
31 int cont_leer_flag(void){
32 if(cont_flag == 1){
33 cont_flag = 0;
34 return 1;
35 }
36 else{
37 return 0;
38 }
39 }
40

```

# Modificadores

- **volatile**
  - Evita que el compilador optimice el código eliminando accesos a memoria para lecturas del valor de la variable que cree que no se modifica.
- **const**
  - Genera variables constantes (no se pueden modificar) lo cual tiene como consecuencia que puede cambiar el lugar donde se almacena la información (e.g. memoria flash en lugar de memoria RAM)
- **extern**
  - Permite utilizar variables que se declararon en otros archivos.
- **static**
  - Restringe la visibilidad de las variables globales o funciones al archivo donde se está trabajando.
  - Un uso importante es que si se usa dentro de una función permite retener el valor entre ejecuciones de la misma.

```
volatile int flag=0;

int main(){
 while(1){

 else if(flag == 1){
 prender_led();
 delayms(1000);
 apagar_led();
 flag = 0;
 }
 }
}

void interrupcion_pulsador(){
 flag = 1;
}
```

# Modificadores

- **volatile**
  - Evita que el compilador optimice el código eliminando accesos a memoria para lecturas del valor de la variable que cree que no se modifica.
- **const**
  - Genera variables constantes (no se pueden modificar) lo cual tiene como consecuencia que puede cambiar el lugar donde se almacena la información (e.g. memoria flash en lugar de memoria RAM)
- **extern**
  - Permite utilizar variables que se declararon en otros archivos.
- **static**
  - Restringe la visibilidad de las variables globales o funciones al archivo donde se está trabajando.
  - Un uso importante es que si se usa dentro de una función permite retener el valor entre ejecuciones de la misma.

```
main.c X contador.h X contador.c X
1 #include <stdio.h>
2 #include "contador.h"
3
4 extern volatile int cont_flag;
5
6 void buzz(void);
7
8 int main() {
9
10 cont_init(500);
11
12 while(1){
13
14 cont_incremento();
15 if(cont_flag == 1){
16 cont_flag = 0;
17 buzz();
18 }
19 }
20
21 return 0;
22 }
```

```
main.c X contador.h X *contador.c X
1 /*
2 * contador.c
3 */
4
5 #include "contador.h"
6
7 volatile int cont_flag;
8 volatile int n_cuenta;
9 volatile int i;
10
11 void cont_init(int cuentas){
12 n_cuenta = cuentas;
13 i=0;
14 cont_flag = 0;
15 }
16
17 void cont_incremento(void){
18 i++;
19 if(i>=n_cuenta){
20 i = 0;
21 cont_flag = 1;
22 }
23 }
24
25 }
```

# Modificadores

- **volatile**
  - Evita que el compilador optimice el código eliminando accesos a memoria para lecturas del valor de la variable que cree que no se modifica.
- **const**
  - Genera variables constantes (no se pueden modificar) lo cual tiene como consecuencia que puede cambiar el lugar donde se almacena la información (e.g. memoria flash en lugar de memoria RAM)
- **extern**
  - Permite utilizar variables que se declararon en otros archivos.
- **static**
  - Restringe la visibilidad de las variables globales o funciones al archivo donde se está trabajando.
  - Un uso importante es que si se usa dentro de una función permite retener el valor entre ejecuciones de la misma.

```
void func(){
 static int x = 0;

 x++;
 printf("%d\n", x);
}

int main() {

 func();
 func();
 func();

 return 0;
}
```



# Enums

- Los enums crean grupos de etiquetas agrupadas bajo un mismo tipo
- Les daremos uso para evitar usar códigos numéricos que deben recordarse, parecido a los defines
- Son muy usados, también, en máquinas de estados

```
enum resultado_e {res_ok, res_falla, res_indef}
enum resultado_e res;
res = res_ok;
```

```
typedef enum { ADS_FREQ_1k,
 ADS_FREQ_2k,
 ADS_FREQ_4k} ads_freq_e;
typedef enum { ADS_GAIN_1,
 ADS_GAIN_10} ads_gain_e;
void ads_init(ads_freq_e, ads_gain_e);
...
ads_init(ADS_FREQ_2k, ADS_GAIN_10);
```

# Enums

```
// INCLUDES
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

// DEFINES
#define COMM_HEADER 0xF0F0F0F0

// TIPOS
typedef enum{ res_ok,
 res_falla,
 res_indef} resultado_e;

typedef struct {
 uint32_t header;
 uint32_t cuenta;
 uint8_t datos[40];
} datos_t;
```

```
// DECLARACIONES
resultado_e get_datos(datos_t * d);

// DEFINICIONES
int main(){
 datos_t datos;
 resultado_e res = res_indef;
 while (res!= res_ok){
 res = get_datos(&datos);
 }
 return 0;
}

resultado_e get_datos(datos_t * d){
 d->header = 0;
 uart_rx(d, sizeof(datos_t));
 if(d->header == COMM_HEADER){
 return res_ok;
 }
 return res_falla;
}
```

# Enums y máquinas de estados

```
typedef enum {EINI, EWAIT, ETX, EFIN} estados_e;
estados_e estado = EINI;

. . .
switch(estado){
 case EINI:
 ntx = 0;
 estado = EWAIT;
 break;
 case EWAIT:
 if(tx_enable_flag == 1) estado = ETX;
 break;
 case ETX:
 uart_tx(dato[ntx++]);
 if(ntx > N_PACKET) estado = EFIN;
 break;
 case EFIN:
 tx_ready_flag = 1;
 estado = EINI;
 break;
}
```

# Punteros a funciones

```
int suma(int,int);
int resta(int,int);

int main()
{

 int a;
 a = suma(3,4);
 a = resta(3,4);

 int
 (*pfun)(int,int);

 pfun = &suma;
 a= (*pfun)(3,4);

 pfun = &resta;
 a= (*pfun)(3,4);

 return 0;
}
```

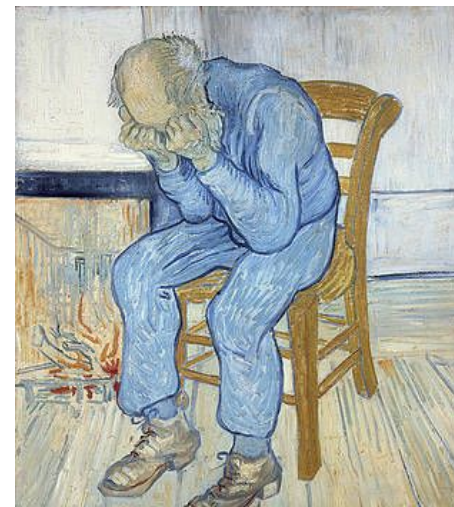
```
int suma(int,int);
int resta(int,int);
int mult(int,int);

int main()
{
 int (*oparr[])(int,int) =
 {&suma, &resta, &mult
};

 int a;

 for(int i=0; i<3; i++){
 a = (*oparr[i])(3,4);
 printf("%d\n",a);
 }

 return 0;
}
```

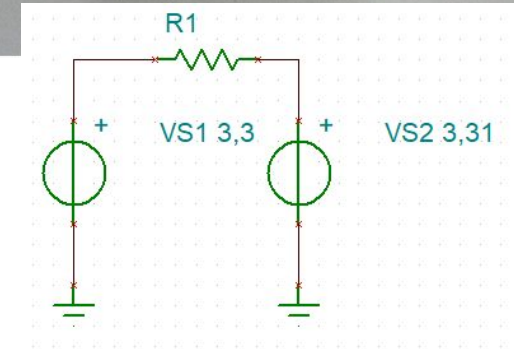
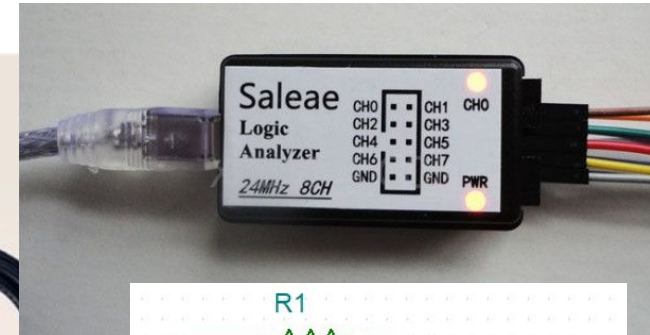
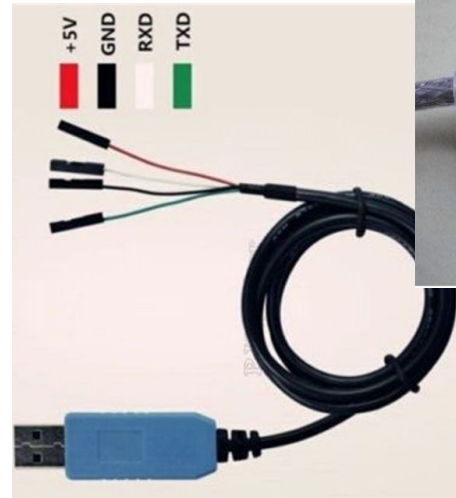
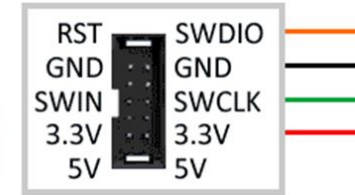


Comience en el nombre de la variable.  
Mire hacia la derecha sin saltar sobre un paréntesis derecho; diga lo que ve.  
Mire hacia la izquierda nuevamente sin saltar sobre un paréntesis; diga lo que ve.  
Salte un nivel de paréntesis si los hay y repita.  
Continúe de esta manera hasta que diga el tipo de variable o el tipo de retorno.

# Precauciones con la electrónica

# Sistema con múltiples fuentes

- Si tenemos dos fuentes de alimentación y no tenemos seguridad de que sean derivaciones de una misma fuente, debemos
  - **Asegurar** conectar las masas de los sistemas para asegurar la misma referencia de potencial.
  - **Evitar** tener dos alimentaciones distintas conectadas.
  - **Evitar** aplicar tensión en terminales de un sistema sin alimentación.
- Esto sucederá a menudo porque utilizaremos un programador, un adaptador UART-USB y un analizador lógico, los 3 con alimentación por USB y pines que permiten alimentar a su vez un sistema.

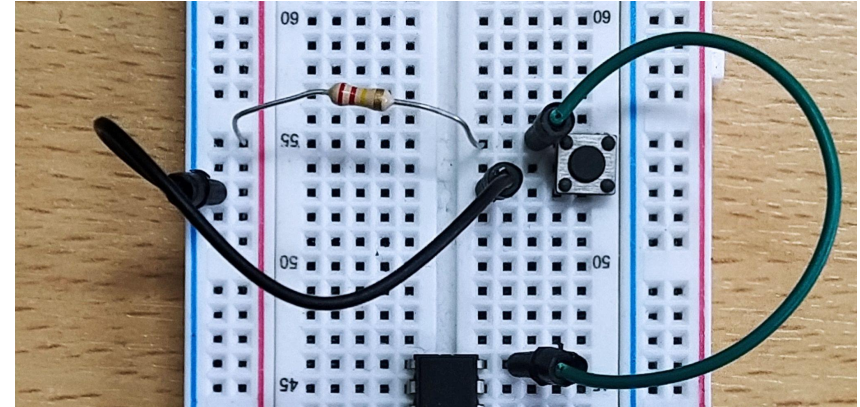


## Sistemas con múltiples fuentes

- Si tenemos dos fuentes de alimentación y no tenemos seguridad de que sean derivaciones de una misma fuente, debemos
  - **Asegurar** conectar las masas de los sistemas para asegurar la misma referencia de potencial.
  - **Evitar** tener dos alimentaciones distintas conectadas.
  - **Evitar** aplicar tensión en terminales de un sistema sin alimentación.
- Esto sucederá a menudo porque utilizaremos un programador, un adaptador UART-USB y un analizador lógico, los 3 con alimentación por USB y pines que permiten alimentar a su vez un sistema.

# Conexiones

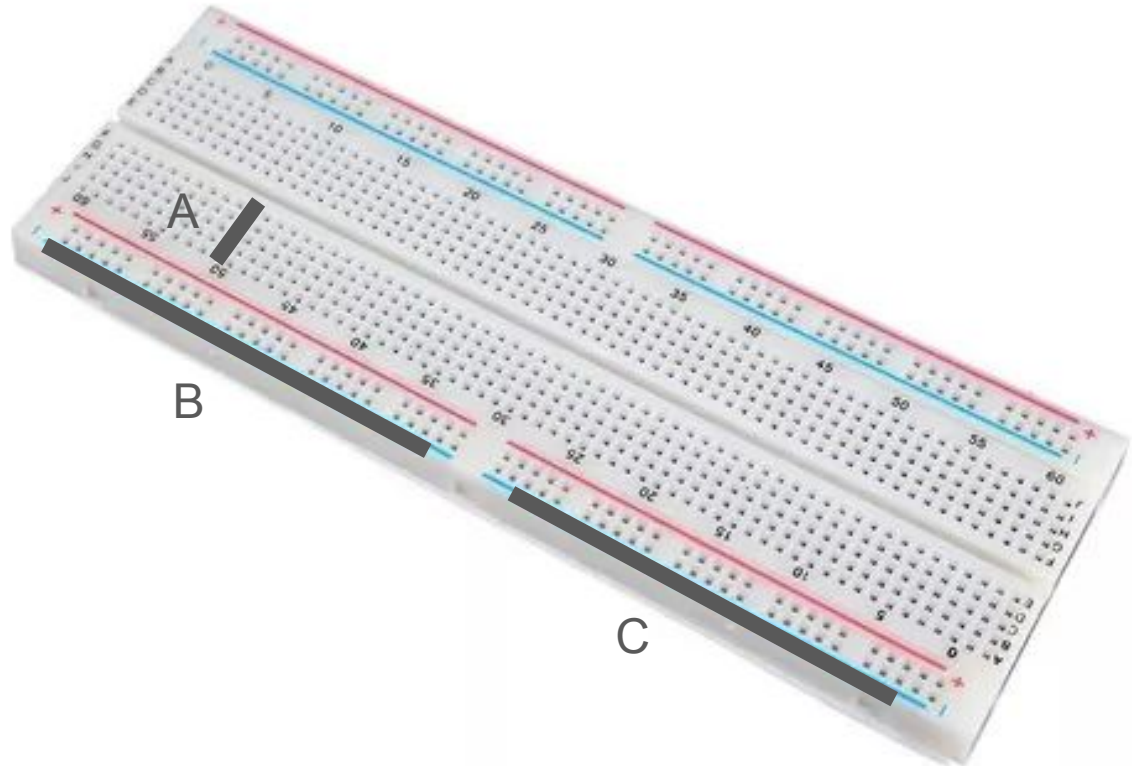
- Si no hay nada conectado a un terminal, la tensión de ese terminal es impredecible
- En ocasiones se quiere asegurar con una tensión determinada: Vcc o Gnd
- Para eso se utilizan resistencias de pull up o pull down, en general valor relativamente alto, de manera de que sea posible modificar esa tensión con un driver externo
- En los microcontroladores este componente suele estar incluido dentro del encapsulado como una configuración opcional





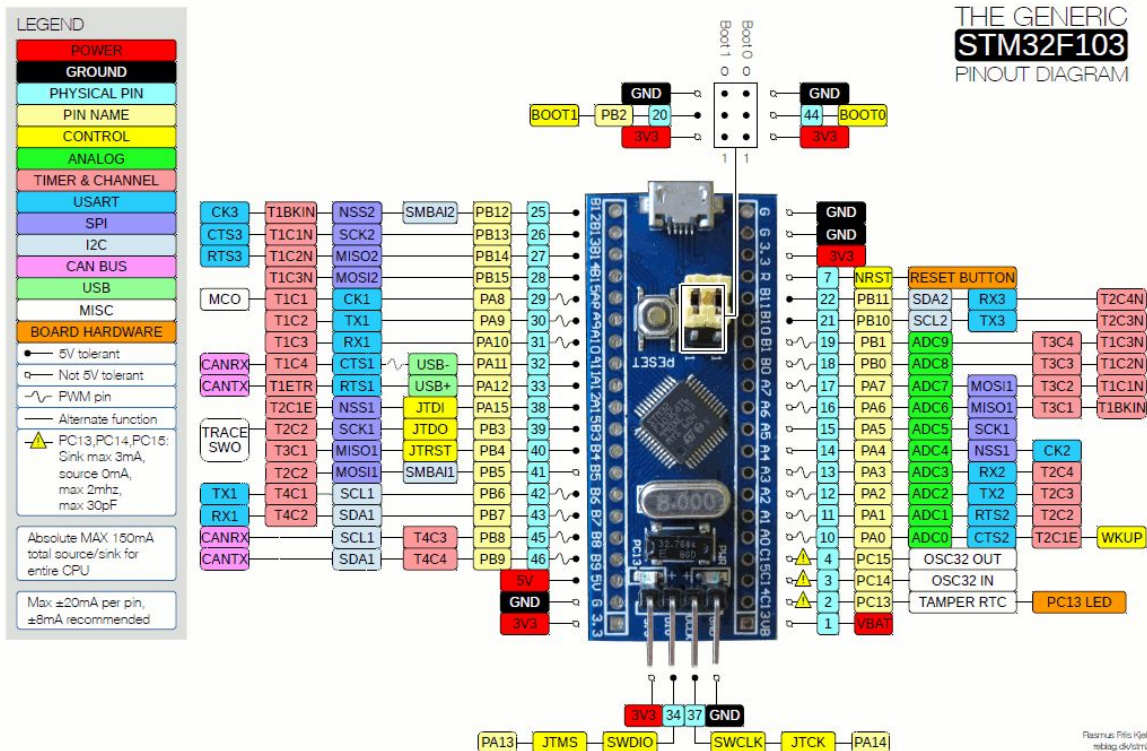
# Protoboard

- El protoboard tiene conexiones en el sentido marcado con las líneas negras (y repitiendo)
- NOTAR que los segmentos B y C están conectados en algunos protoboards, pero en la mayoría NO

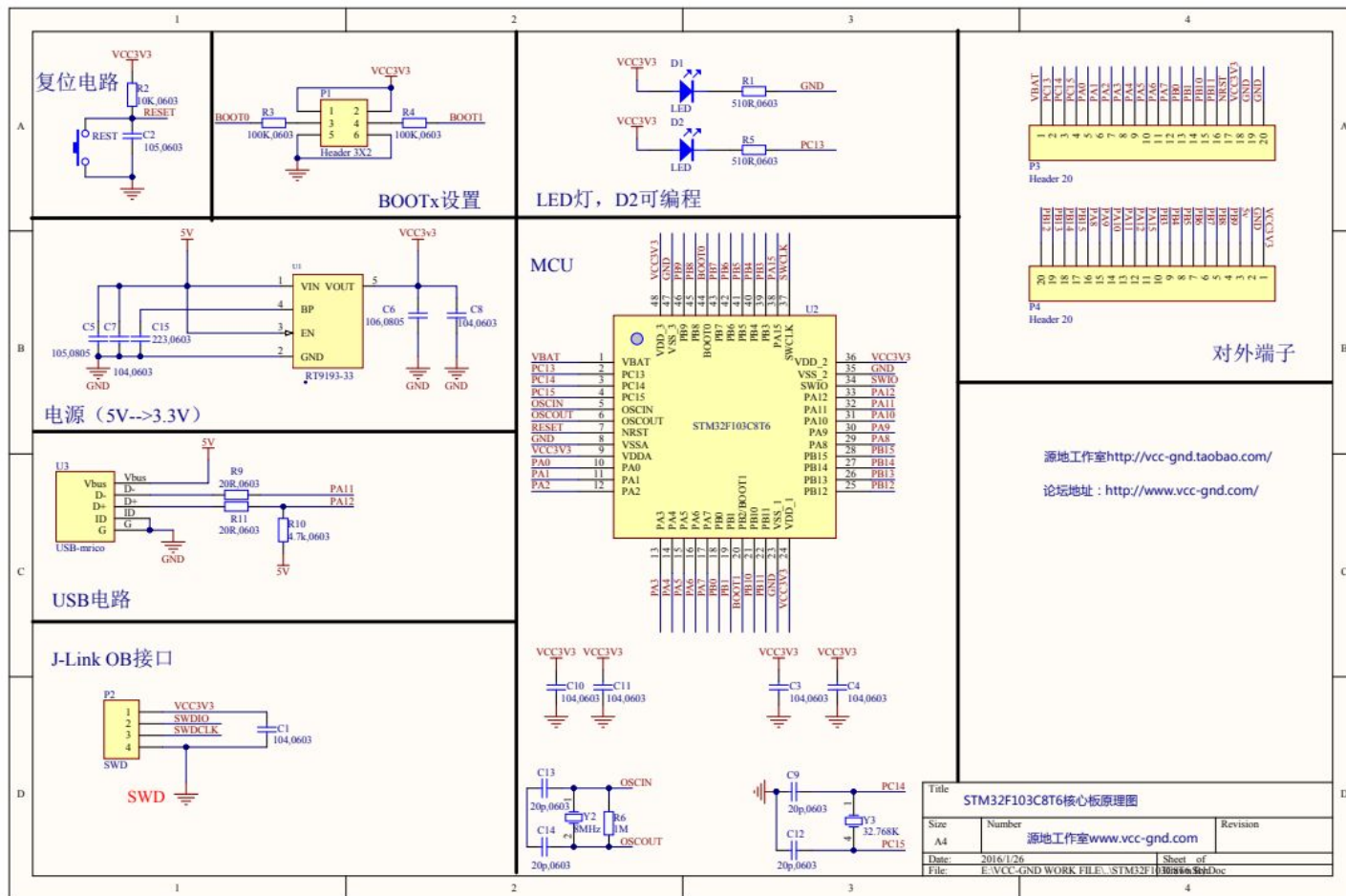


# “Caso de estudio”

# Placa de desarrollo



## Esquemático



# Microcontrolador

El nombre del dispositivo ayuda a navegar la documentación



STM32 F 1 03 c 8 t 6

**Familia**  
F es la familia  
“base”

**Procesador**  
1: Cortex M3

**Línea**

**Número de pines**  
C:48

**Tamaño de la  
memoria Flash**  
8: 64 kB  
B: 128 kB

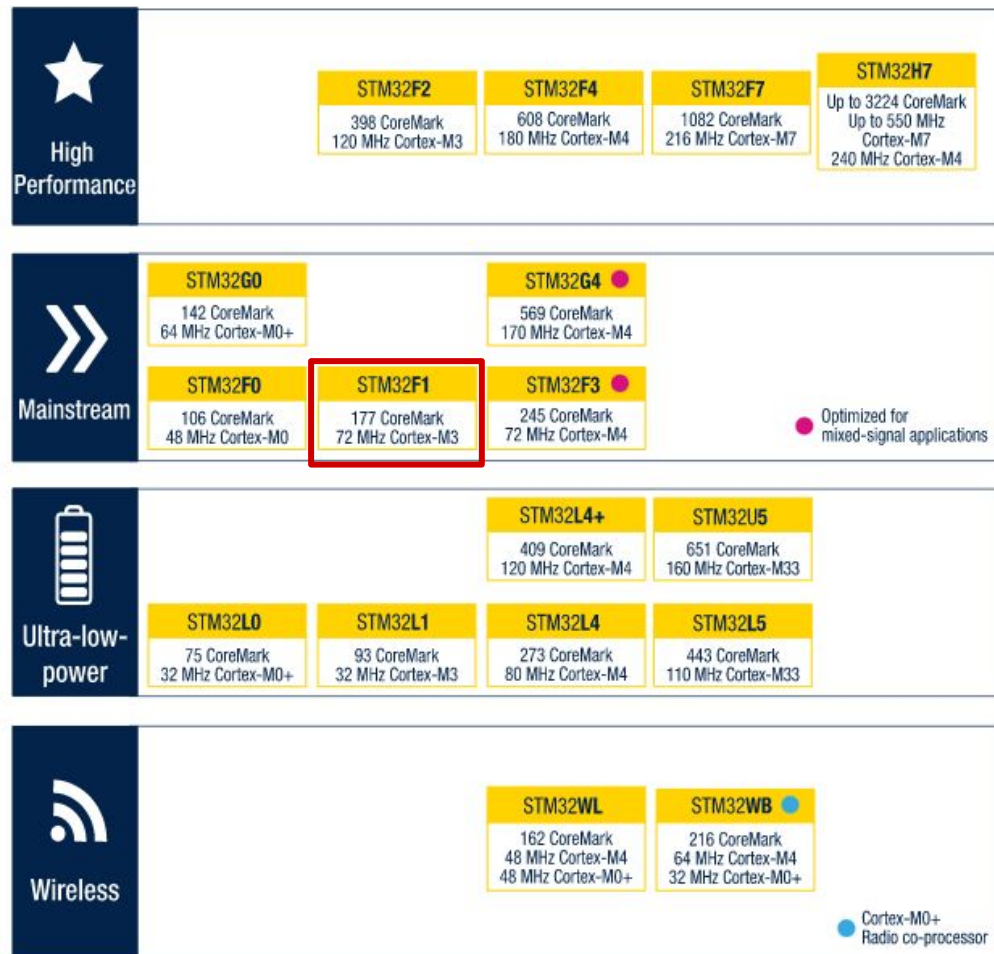
**Rango de  
temperaturas**

**Encapsulado**

# Familia de microprocesadores de ST Microelectronics

- Familias: En general aplicable a todos los fabricantes: apuntan a distintas aplicaciones y dentro de cada una a un rango de prestaciones
- En este caso ST eligió el benchmark CoreMark para compararlos
- En particular el STM32F103C8 es un microcontrolador de 32 bits, basado en Cortex-M3

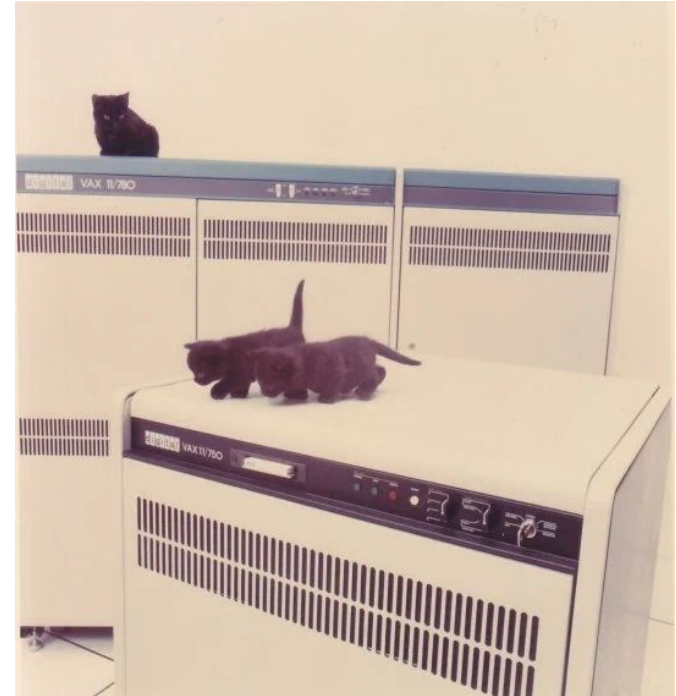
<https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>





## Breve comentario sobre benchmarks

- Histórico y vigente por la disponibilidad de evaluaciones: Dhrystone
- Un test como el D. mide “cuánto puede hacer” un procesador vs las mega-instrucciones por segundo (MIPS)
- La VAX 11/780 ejecutaba 1757 ciclos del test de Dhrystone por segundo
- Si un procesador ejecuta  $N \cdot 1757$  veces el test en 1s es una máquina de  $N$  DMIPS, y si lo hace con un clock de  $M$  MHz, es de  $N/M$  DMIPS/MHz




# Benchmarks actuales

- Embedded Microprocessor Benchmark Consortium <https://www.eembc.org/>
  - “EEMBC develops industry-standard benchmarks for the hardware and software used in autonomous driving, the Internet of Things, machine learning, and many other applications”
- Por ejemplo:
  - **CoreMark** evalúa el CPU e incluye recorrer listas, operaciones matemáticas con matrices y procesamiento con máquinas de estado
  - **ULPBench** evalúa el consumo de energía apuntando a aplicaciones de consumo ultra-bajo. Se consider ULP si dura más de 4 años ejecutando una función activa 1 vez por segundo con una batería de 225 mAh. Su primer test es Core Profile que mide la eficiencia del CPU, reloj de tiempo real, calendario, y modos de ahorro de energía.

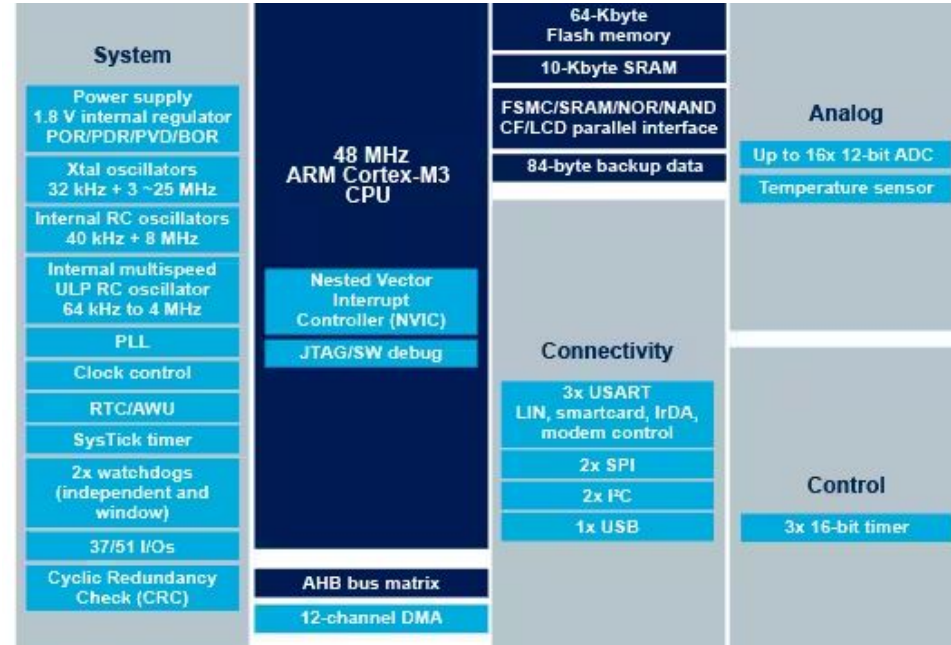
| Feature                      | <a href="#">Cortex-M0</a> | <a href="#">Cortex-M0+</a> | <a href="#">Cortex-M1</a> | <a href="#">Cortex-M23</a> | <a href="#">Cortex-M3</a> | <a href="#">Cortex-M4</a> | <a href="#">Cortex-M33</a> | <a href="#">Cortex-M35P</a> | <a href="#">Cortex-M55</a> | <a href="#">Cortex-M7</a> |
|------------------------------|---------------------------|----------------------------|---------------------------|----------------------------|---------------------------|---------------------------|----------------------------|-----------------------------|----------------------------|---------------------------|
| Instruction Set Architecture | Armv6-M                   | Armv6-M                    | Armv6-M                   | Armv8-M Baseline           | Armv7-M                   | Armv7-M                   | Armv8-M Mainline           | Armv8-M Mainline            | Armv8.1-M Mainline         | Armv7-M                   |
| DMIPS/MHz*                   | 0.87                      | 0.95                       | 0.8                       | 0.98                       | 1.25                      | 1.25                      | 1.5                        | 1.5                         | 1.6                        | 2.14                      |
| CoreMark®/MHz*               | 2.33                      | 2.46                       | 1.85                      | 2.64                       | 3.34                      | 3.42                      | 4.02                       | 4.02                        | 4.2                        | 5.01                      |



# Características y uso del STM32F103C8

STM32F103C8 ACTIVE  Save to MyST

Mainstream Performance line, Arm Cortex-M3 MCU with 64 Kbytes of Flash memory, 72 MHz CPU, motor control, USB and CAN



<https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html>

# Documentación y literatura

- Documentación de los fabricantes
  - **Datasheet** del microcontrolador “STM32F103x8 STM32F103xB Datasheet”
    - Características eléctricas y descripción breve de la organización
  - **Manual de referencia del microcontrolador** “ST RM0008 Reference Manual”
    - Funcionalidad de todos los periféricos (muy utilizado)
  - **Manual de programación del Cortex M3** de ST: “ST PM0056 M32F10xxx/20xxx/21xxx/L1xxxx Cortex®-M3 programming manual”
    - Detalles de la ISA escritos por ST (más resumidos y prácticos que la documentación de ARM que es muy detallada en este respecto)
  - **Manual de referencia técnico del Cortex-M3** “Cortex-M3 Revision r2p0 Technical Reference Manual”
    - Descripción de la organización del procesador
  - **Manual de referencia del set de instrucciones** “ARMv7-M Architecture Reference Manual”
    - Descripción de “alto” nivel del set de instrucciones
- Libros
  - Joseph Yiu - The Definitive Guide to ARM Cortex -M3 and Cortex-M4 Processors
  - Carmine Noviello - Mastering STM32
  - D.W. Lewis - Fundamentals of Embedded Software with the Arm Cortex M3
- Otra documentación
  - Debido al uso de una placa de desarrollo, necesitaremos el esquemático de la placa y quizás documentación de alguno de sus componentes