uc3m

Departamento Tecnología Electrónica

Tema 5: Entorno de Desarrollo

Sistemas Digitales Basados en Microprocesadores

Universidad Carlos III de Madrid

Dpto. Tecnología Electrónica

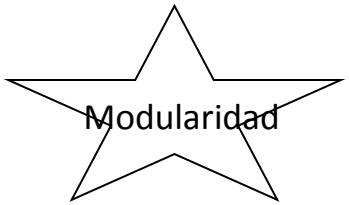
Índice

- Ciclo de Desarrollo
- Diagramas de Flujo
- La placa de Desarrollo STM32L-DISCOVERY
- El entorno de trabajo µVision5 y Cube MX
- Peculiaridades de la Programación en C en Microcontroladores

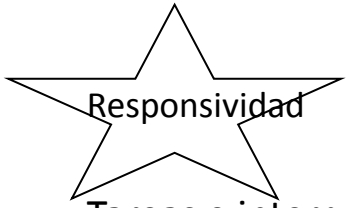
Ciclo de Desarrollo

Diseño estructurado de aplicaciones: Estrategia Top/Down

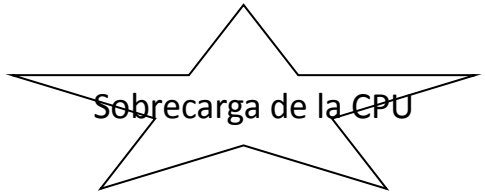
1. Determinar los requisitos y especificaciones del sistema
2. Determinar la *arquitectura* del sistema para cumplir con los requisitos y especificaciones, determinando los bloques SW y HW. Explota el uso de periféricos (¡HW mejor que SW!)
3. Describir la funcionalidad de los bloques SW usando uno o varios ***algoritmos*** que resuelven el problema. La descripción puede hacerse con pseudo-codigo o con diagramas de flujo
4. Los diagramas de flujo nos ayudan a estructurar el programa, usando un número reducido de estructuras posibles.
5. Estructura los algoritmos en ***funciones*** (rutinas), definiendo las ***variables*** que se necesitan y su tamaño
6. Haz uso de ***interrupciones*** con rutinas de servicio ***rápidas*** y sin bucles. Usa el sistema de prioridades.
7. Confecciona el programa usando las estructuras que conoces y trata de ser fiel a la estructura diseñada en el diagrama de flujo
8. Genera un proyecto en la herramienta de desarrollo y ***simula*** la funcionalidad del programa completo y de cada una de las funciones desarrolladas. ¿ Funcionan los algoritmos diseñados como esperábamos?
9. ***Depura*** los algoritmos y el código hasta obtener un ***código compacto y funcional***. Evalúa las prestaciones de tiempo real ¿Se cumplen todos los requisitos y especificaciones funcionales?
10. Prueba el código en el hardware usando ***depuración en circuito*** (o emulación), y realiza la última depuración



Programa de acuerdo a tareas



Tareas e interrupciones cortas

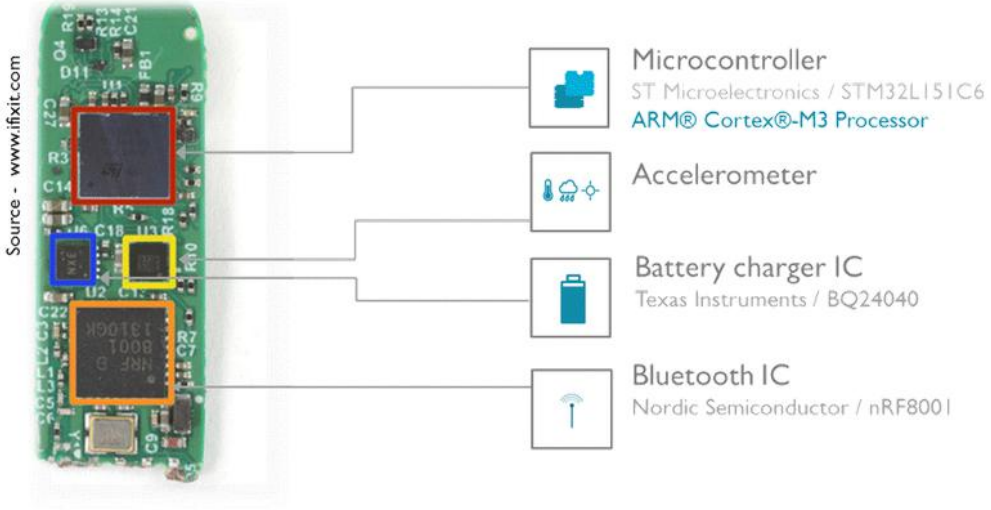


Usa el HW de los periféricos
Evita bucles de espera

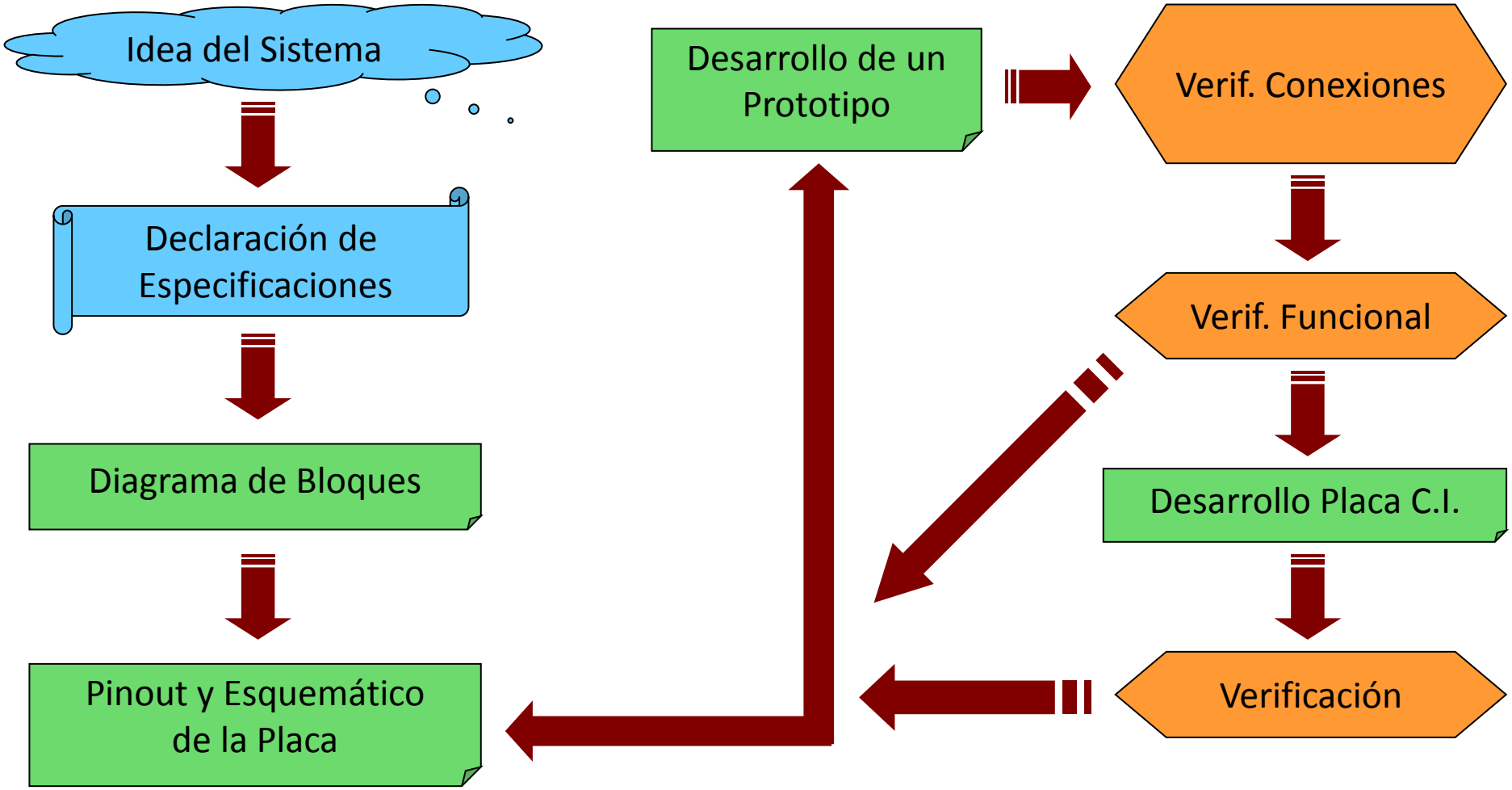
Prototipos y Aplicaciones finales

En prototipos es habitual usar placas de desarrollo o evaluación de un fabricante

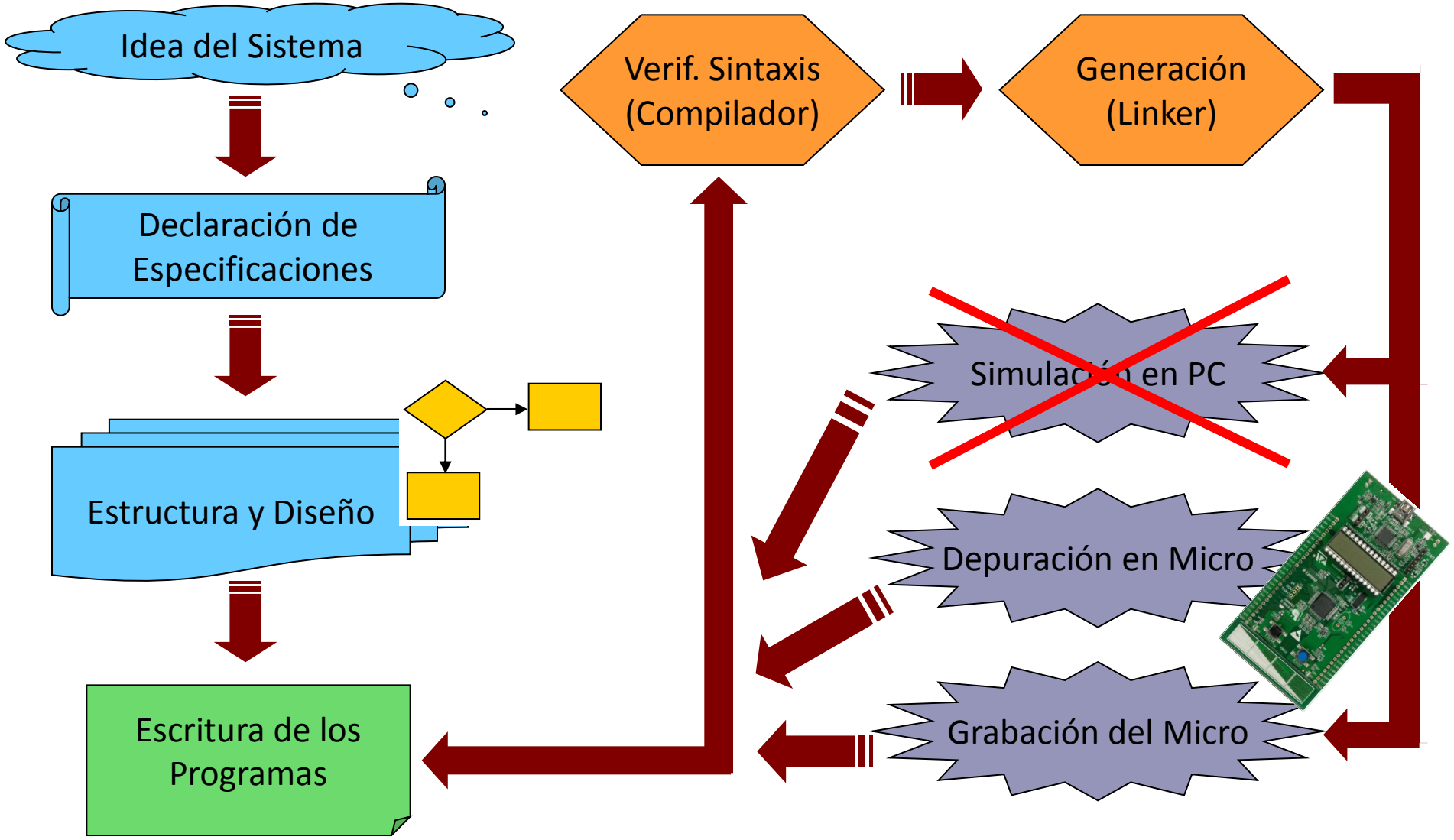
En el producto todo el HW es diseñado para la aplicación



Ciclo de Desarrollo Hardware



Ciclo de Desarrollo Software



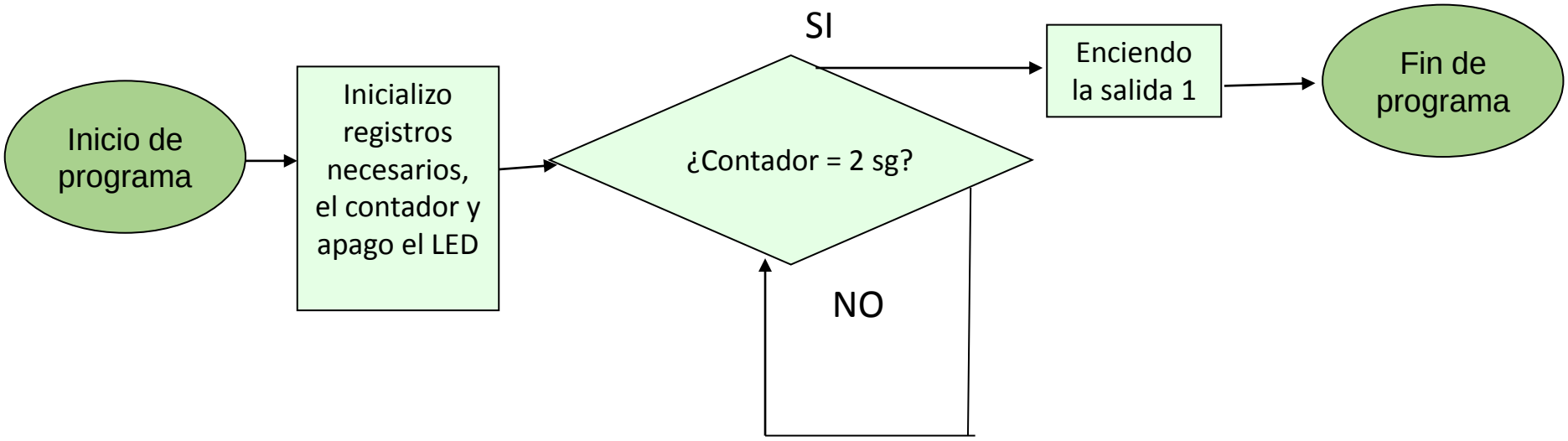
Diagramas de Flujo

Diagramas de Flujo

- Son representaciones del funcionamiento de un programa
  - De forma genérica – independiente de la arquitectura
    - Nunca pueden contener referencias a registros de la arquitectura utilizada, ni a instrucciones de la CPU
  - Que muestran la solución al problema planteado
  - Que tienen que servir de guía, tanto al programador como a los posibles programadores que tengan que tocar ese programa
- Se pueden escribir a distintos niveles de detalle/abstracción
  - El nivel de detalle que debe ser utilizado dependerá de la situación
- Tradicionalmente se utilizarán sólo los símbolos sencillos:
  - Elipse o círculo, para indicar una etiqueta
  - Rectángulo, para indicar un proceso
  - Rombo, para indicar una decisión

Ejemplo

Programa que espera 2 segundos para encender un LED en una salida digital del micro y luego la deja encendida para siempre



La placa de Desarrollo STM32L-DISCOVERY

STM32L-Discovery

STM32L-DISCOVERY

STM32L ultralow power discovery board

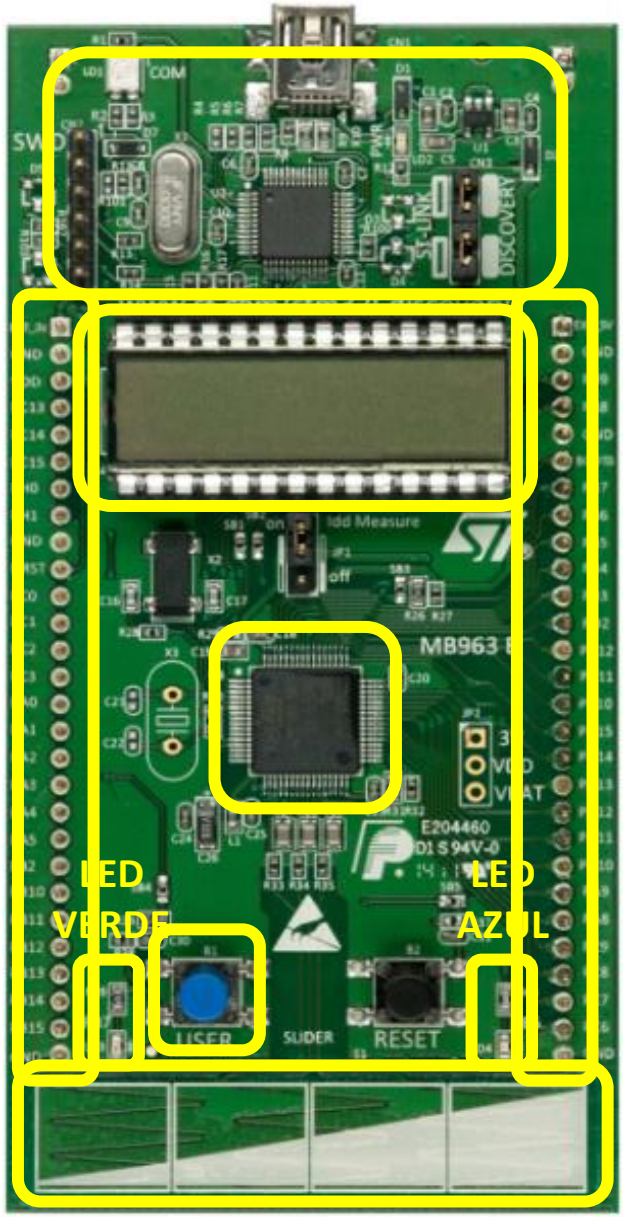
Data brief

Features

- STM32L152RBT6 microcontroller featuring 128 KB Flash, 16 KB RAM, 4 KB EEPROM, in an LQFP64 package
- On-board ST-Link/V2 with selection mode switch to use the kit as a standalone ST-Link/V2 (with SWD connector for programming and debugging)
- Board power supply: through USB bus or from an external 3.3 or 5 V supply voltage
- External application power supply: 3 V and 5 V
- I_{DD} current measurement
- LCD
  - DIP28 package
  - 24 segments, 4 commons
- Four LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power on
  - Two user LEDs, LD3 (green) and LD4 (blue)
- Two pushbuttons (user and reset)
- One linear touch sensor or four touchkeys
- Extension header for LQFP64 I/Os for quick connection to prototyping board and easy probing

STM32L-Discovery

- La placa de desarrollo tiene las siguientes funcionalidades:
  - Microcontrolador STM32L152RB
  - Interfaz de depuración ST-LINK/V2 incluido (conectado al ordenador a través de Mini-USB)
  - Una pantalla LCD de 24 segmentos y 4 comunes
  - 4 LEDs
    - 2 de ellos programables por el usuario (LED_VERDE, LED AZUL)
  - Un sensor táctil lineal, con posibilidad de ser utilizado como 4 teclas individuales
  - Botón programable por el usuario (USER)
  - 2 Puertos de expansión a placa adicional (P1 y P2)



D. de Bloques y Layout

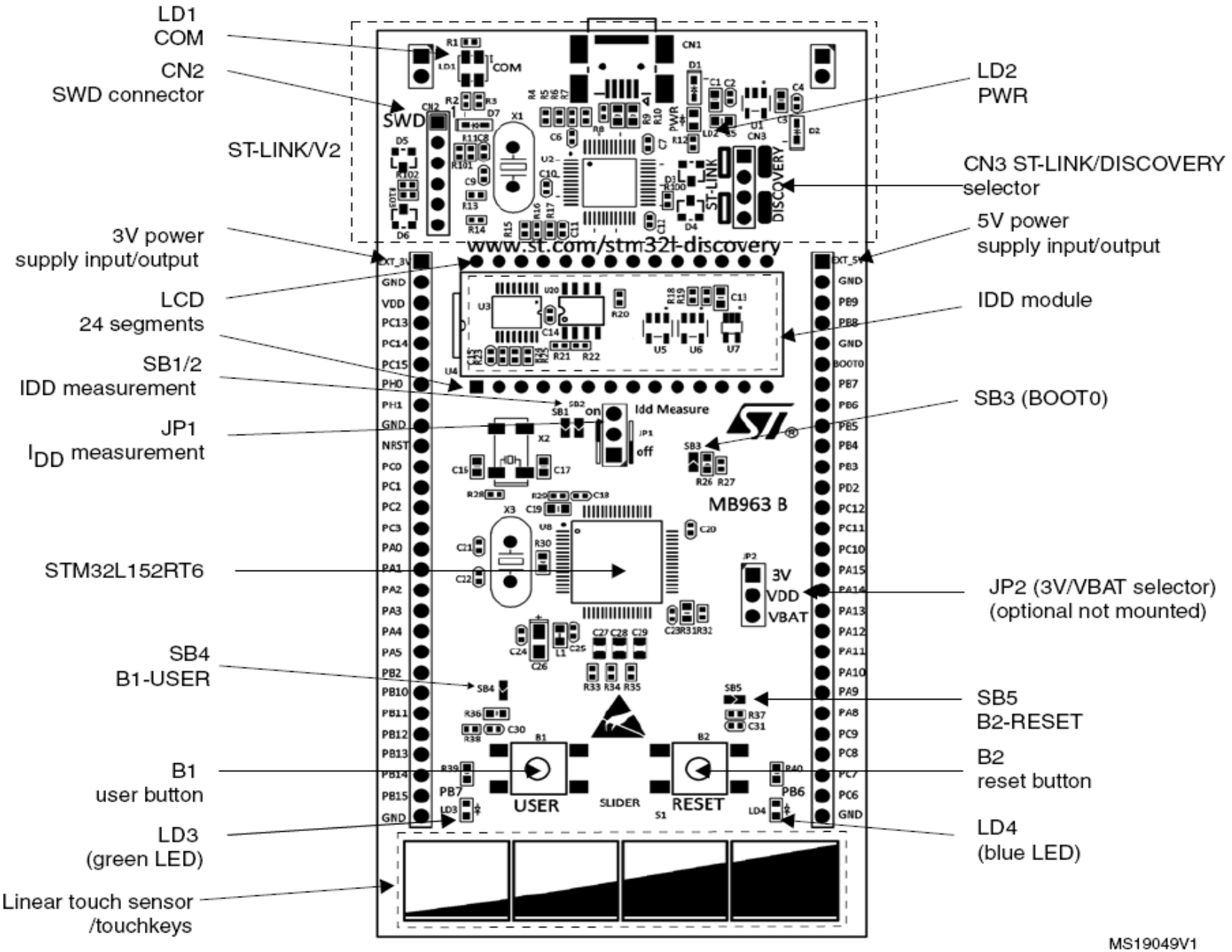
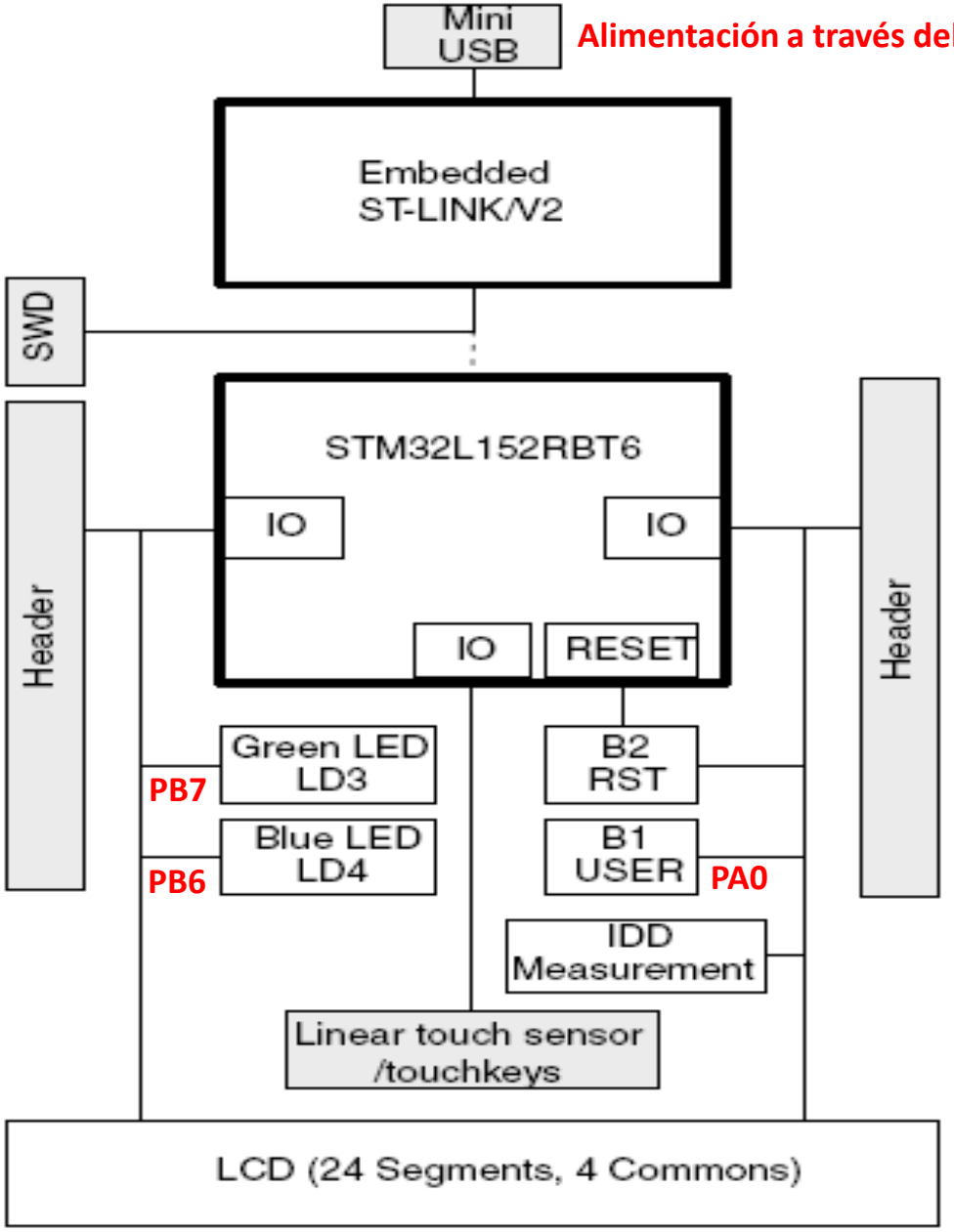


Diagrama de Bloques y Layout



PUERTO P1:

04 – PC13
15 – PA0
ADC_IN0
TIM2_CH1_ETR
19 – PA4
ADC_IN4
DAC_OUT1
20 – PA5
SPI1_SCK
ADC_IN5
DAC_OUT2
TIM2_CH1_ETR

PUERTO P2:

07 – PB7
I2C1_SDA
TIM4_CH2
USART1_RX
08 – PB6
I2C1_SCL
TIM4_CH1
USART1_TX
12 – PD2
TIM3_ETR
13 – PC12
19 – PA12
USBDP
SPI1_MOSI
20 – PA11
USBDM
SPI1_MISO

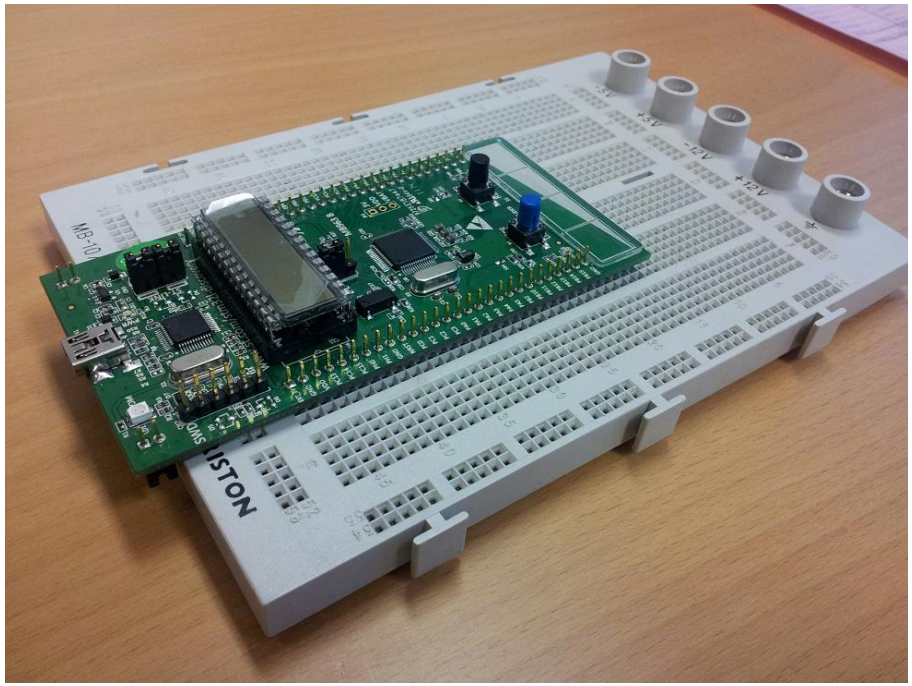
Préstamo de la Placa de Desarrollo

- Durante el curso, se va a permitir el préstamo personal de:
  - La Placa de Desarrollo STM32L-Discovery
- El préstamo se hace con la condición de que el material se devuelva el último día de prácticas
  - De no ser así, los alumnos implicados no serán evaluados en la asignatura
- Para obtener el material:
  - Imprimir, rellenar y firmar **DOS** copias del formulario de préstamo del material. El formulario se encuentra en Aula Global.
  - Entregar al coordinador de la asignatura el formulario, para que lo firme y se quede con una copia.
- El material se recogerá en los laboratorios el último día de prácticas

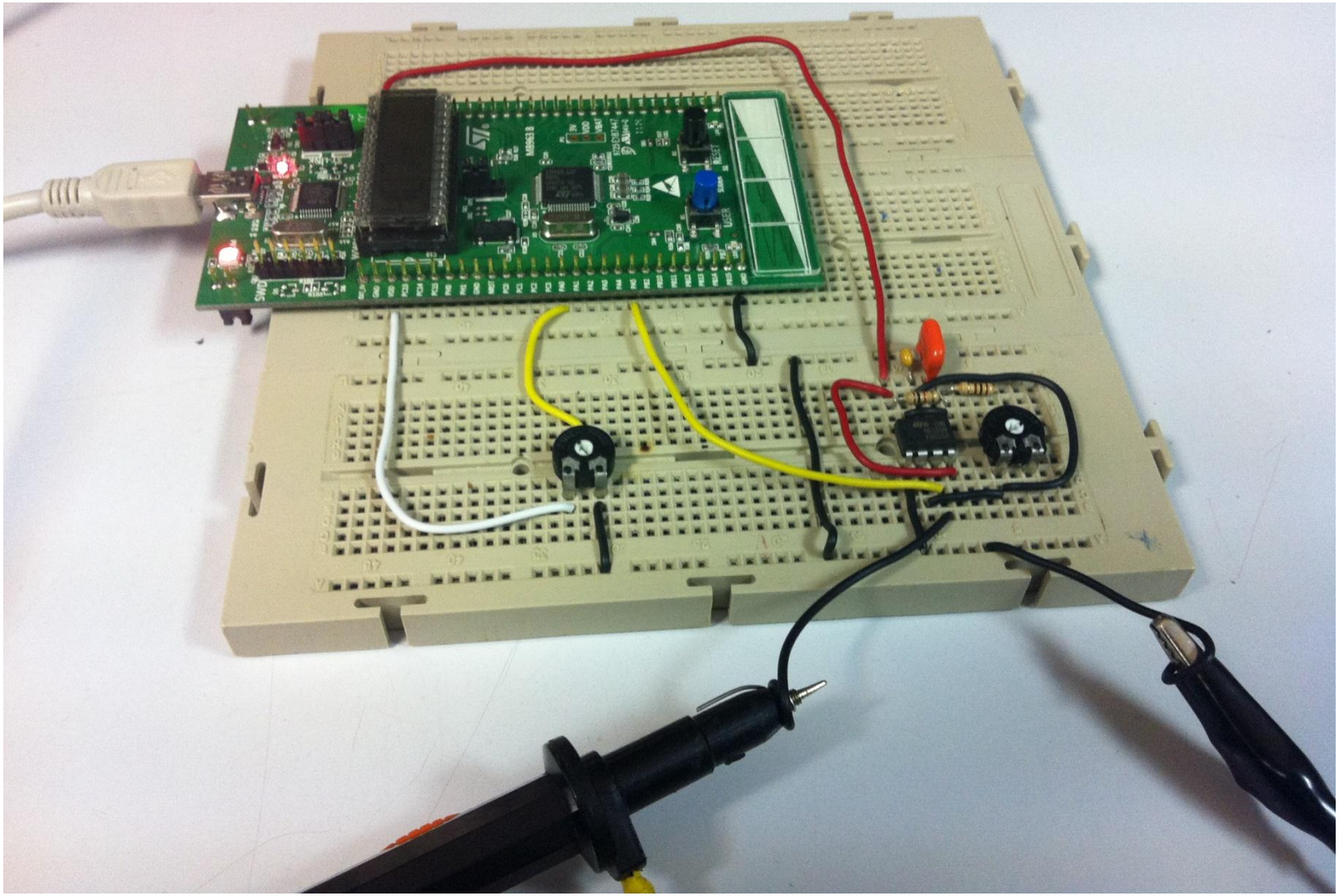
Recomendaciones para el uso de la placa

- Para poder utilizar mucho mejor la placa STM32L1-Discovery, es aconsejable:
  - Pincharla en una protoboard (o en un conjunto de ellas), de forma que los pines no se cortocircuiten y además dejen huecos para conectar cables
  - Meter el conjunto de la protoboard, la placa, así como las conexiones realizadas, en una caja, para su transporte sin que se suelten las conexiones
- En la siguiente transparencia se puede ver el detalle de conexión, así como un ejemplo de uso

Inserción de la placa en una protoboard



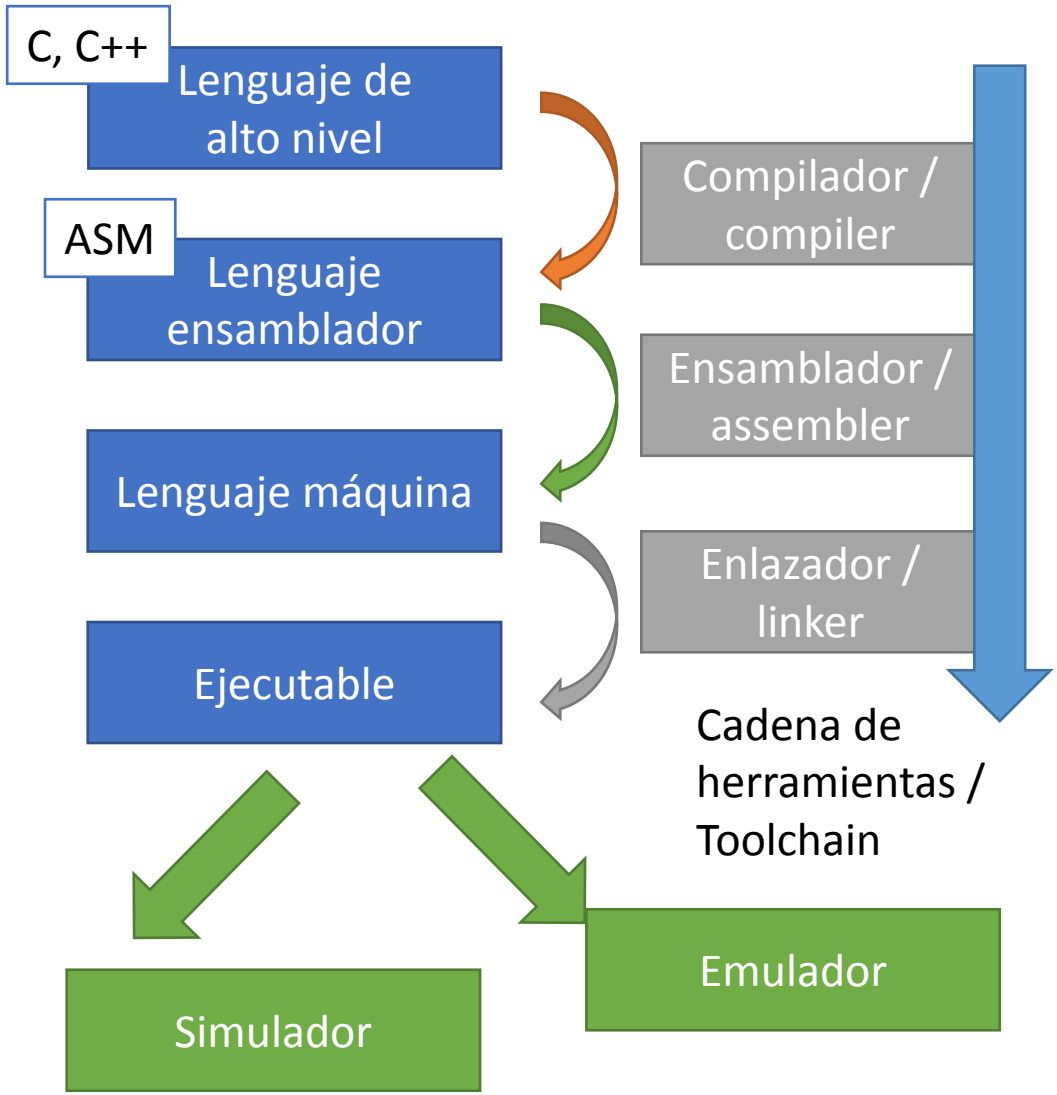
Ejemplo de Uso



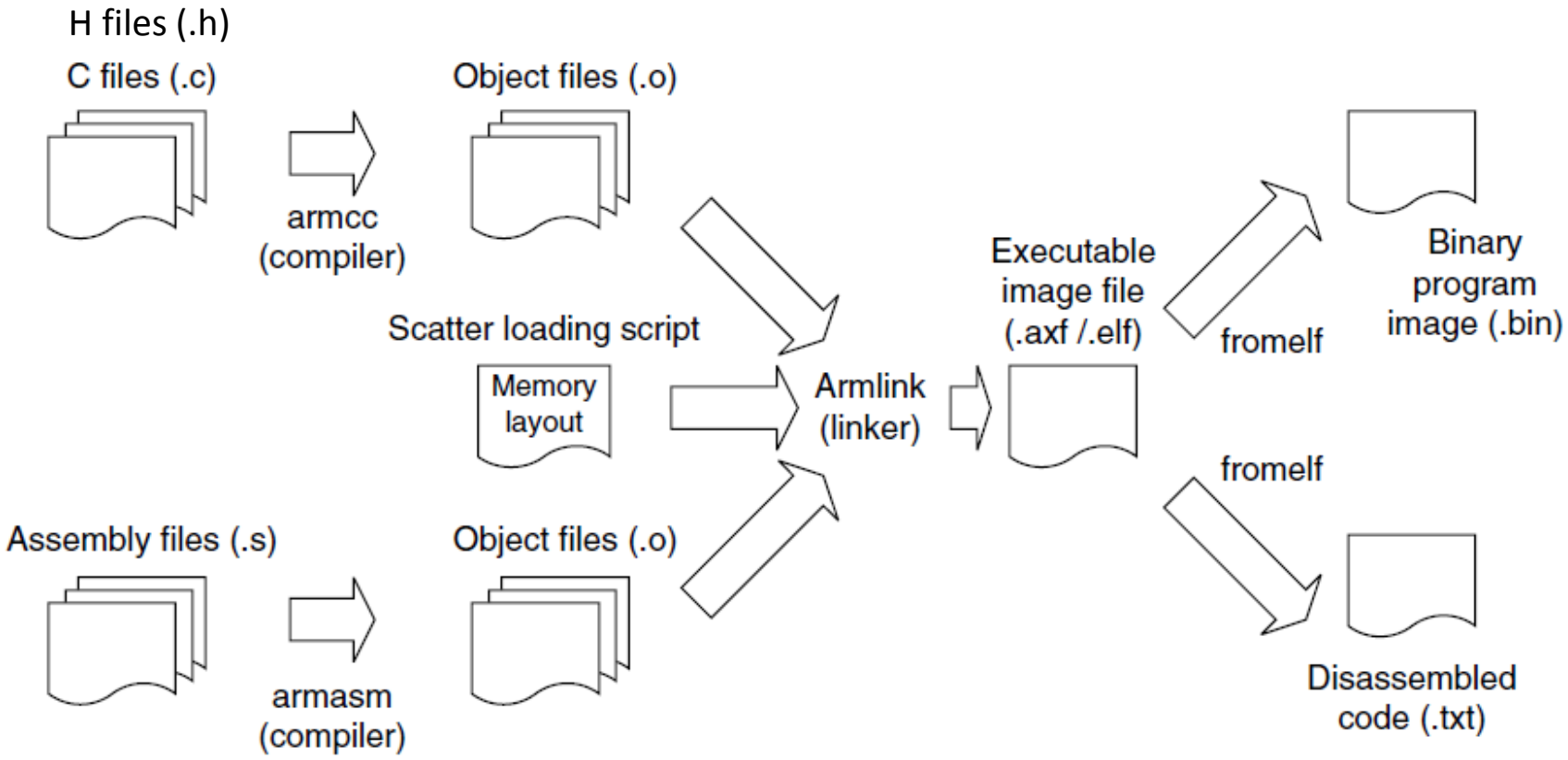
El entorno MDK-ARM

Herramientas necesarias para la programación estructurada

- Lenguaje de alto nivel
  - Se compila a lenguaje ensamblador
- Necesitamos saber cómo se traduce el código C a ensamblador
  - Para usar el procesador de manera eficiente
  - Para analizar el código con precisión
  - Para identificar problemas de prestaciones
- Estudiaremos un breve resumen del lenguaje C y en qué se transforma al compilarlo
  - Módulo de arranque
  - Llamadas a subrutinas
  - Tipos de datos
  - Punteros
  - Control de flujo del programa



MDK-ARM toolchain (entorno Keil)



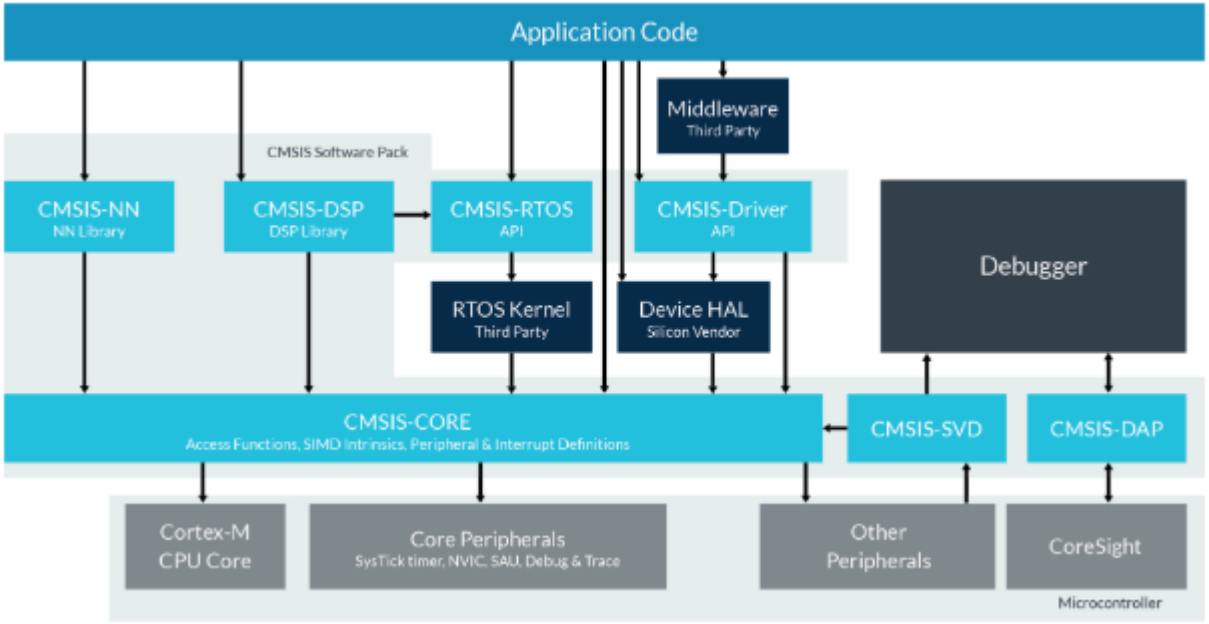
¡¡COMPILACIÓN CRUZADA!!

Ficheros clave en un proyecto “tipo” MDK-ARM

- Módulo de arranque
  - Está en un fichero startup_stm32l152xc.s
  - Contiene un código en ensamblador que inicializa la CPU, llama a una función de inicialización contenida en system_stm32l1xx.c y llama a main() en main.c
  - El fichero system_stm32l1xx.c (forma parte de CMSIS-Driver)
  - Arranca el reloj y configura la tabla de interrupciones
- Código principal
  - Está en el fichero main.c
- Interrupciones
  - Están en el fichero smt32l1xx_it.c
- Definiciones del mapeado de registros en memoria
  - Está en el fichero stm32l152xc.h

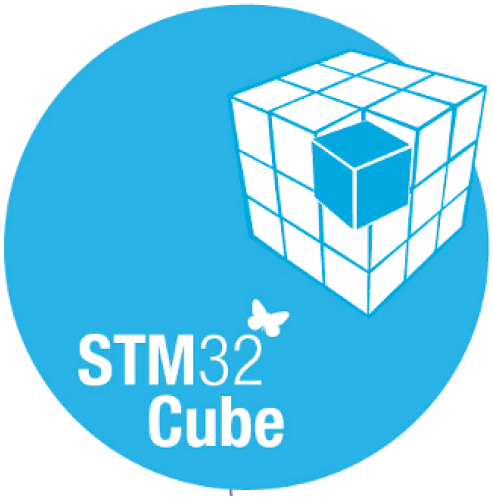
Bibliotecas (Libraries)

- CMSIS: Cortex Microcontroller Software Interface Standard (ARM estandard)
- HAL: Hardware Abstraction Layer (específico de cada fabricante)



Generador automático de código

- Herramienta GUI que genera código de inicialización de los periféricos usando las librerías HAL del fabricante.

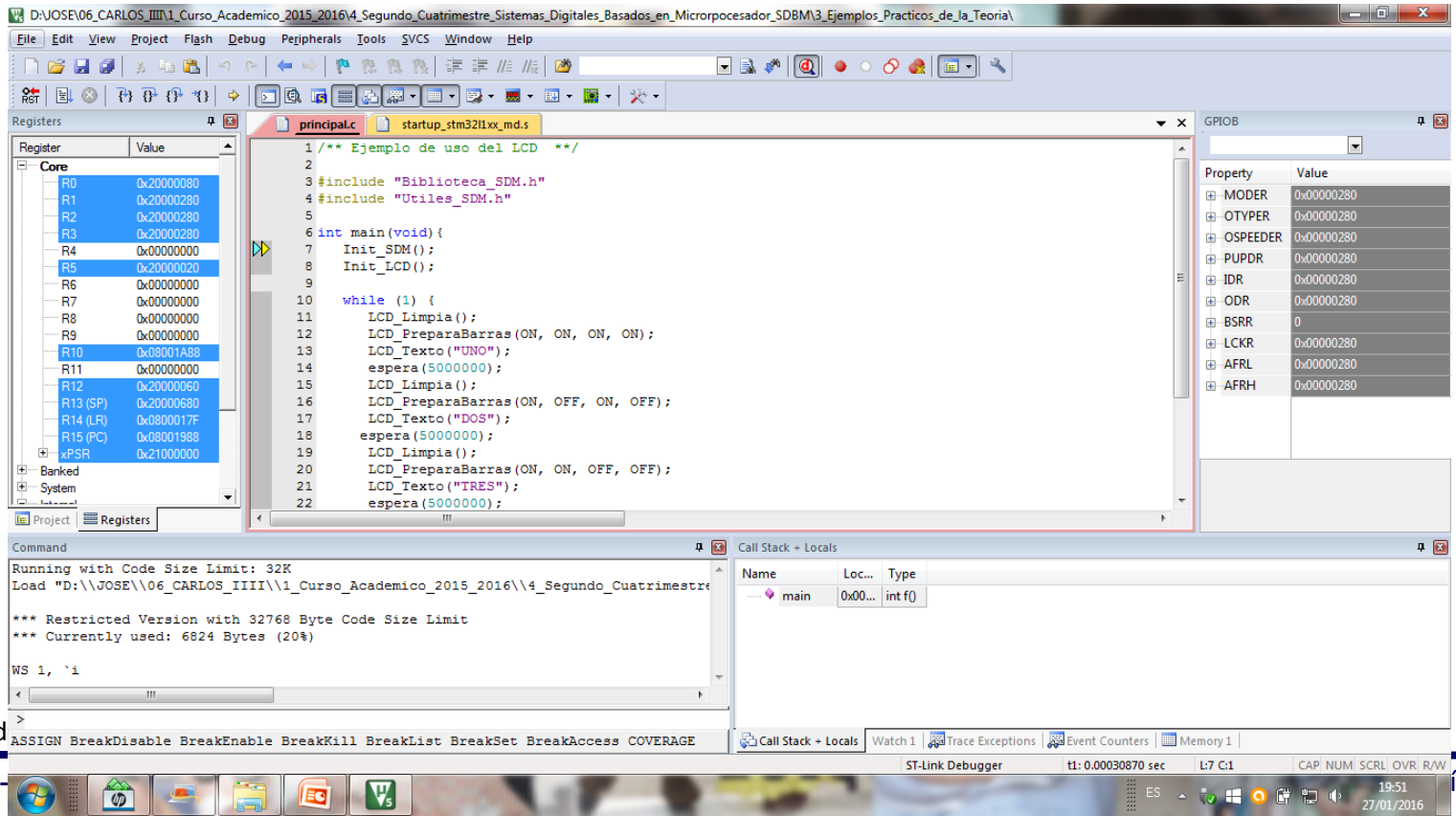


Genera todos los ficheros del proyecto aptos para usar con MDK-ARM

Proporciona una plantilla para el programador, que debe poner su código entre los comentarios previstos

Depuración de un Proyecto con Keil

- Una vez compilado correctamente, conecte la placa al USB y ejecute el depurador con el botón: 
  - Tardará bastante hasta que cargue y luego aparecerá un aviso del modo de evaluación. Pulse Aceptar.
- Si se abre una ventana con ensamblador, pulse  para cerrarla.
- Como puede ver, el programa se ha ejecutado hasta llegar a la función main(). La flecha amarilla indica el punto del programa donde se encuentra la ejecución



Depuración de un Proyecto con Keil

- Para depurar, se pueden utilizar las siguientes opciones:
  - 1) Ejecutar de corrido todo el programa desde el punto actual. Si hay puntos de ruptura, se para en ese punto 
  - 2) Se depura la función actual, pero sin entrar en ella 
  - 3) Se ejecuta todo el programa hasta el punto donde actualmente se encuentra el cursor 
  - 4) Se depura la función actual, entrando en ella, si es necesario 
  - 5) Se sale de la función actual (se ejecuta lo que queda de función y se pausa la ejecución al finalizarla). Solo funciona cuando se ha activado la anterior previamente 
- Además, puede poner un punto de ruptura (breakpoint, es decir, pausa la ejecución cuando llega a ese punto) haciendo doble clic en cualquier punto a la izquierda del número de línea.
  - Aparecerá un círculo en rojo.
  - Si se vuelve a hacer doble clic, se quita el breakpoint.

Depuración de un Proyecto con Keil

- Para examinar el valor de una variable, abra una Watch Window (View►Watch Windows►Watch 1)
  - En la parte inferior derecha, junto con Call Stack y con Memory 1 se abre una nueva pestaña denominada Watch 1
  - En dicha pestaña, escriba el nombre de la variable, registro o conjunto de registros.
    - Por ejemplo, escriba RCC y verá una estructura completa con los registros del reloj
  - En cada variable se puede ver su valor (sólo si se entra en la función donde se ejecuta) o incluso modificarlo para que sea tenido en cuenta en el siguiente paso de ejecución.
- Para ver los registros asociados a los periféricos o modificar su valor, seleccione la opción "Peripherals ►System Viewer" y ahora haga visible el registro deseado (por ejemplo: GPIO -> GPIOB)
  - Por ejemplo, active el LED verde y rojo en PB6 y PB7, activando los bits BS6 y BS7 del registro BSRR en GPIOB o desactívelos activando los bits BR6 y BR7 de dicho registro (cuando el depurador esté dentro de la funcion WHILE(1)

Programación en C para ARM en Keil (MDK-ARM Toolchain)

Particularidades de C en sistemas empotrados

Normalmente hay pequeñas diferencias con respecto a C estándar (norma ISO)
Y dependen del compilador concreto.

En nuestro caso, MDK-ARM es compatible con C99 (modificación de ANSI C o C89)

No se suele usar reserva dinámica de memoria. Función malloc()???

La entrada/salida estándar no es una pantalla. Función printf()???
A veces está redirigida a un puerto serie, en principio no tiene por qué existir.

Los recursos del procesador son muy limitados, se suele limitar: el número y tamaño de variables globales, el anidamiento de funciones, el uso de tipos de datos no soportados directamente por el HW del procesador (ej. tipo float)

Organización de un programa en C

Fichero cabecera(.h)
I. Inclusión de librerias #include
II. Definición de macros y constantes
III. Definición de funciones o prototipos
IV. Definición de variables globales
V. Definición de cuerpo de funciones declaradas

Fichero fuente (.c)
I. Directivas del pre-procesador #include y #define
II. Declaración de Prototipos (Declaración de funciones)
III. Declaración de variables globales
IV. Funciones definidas por el usuario
V. Función Principal main( )

Organización de un programa en C

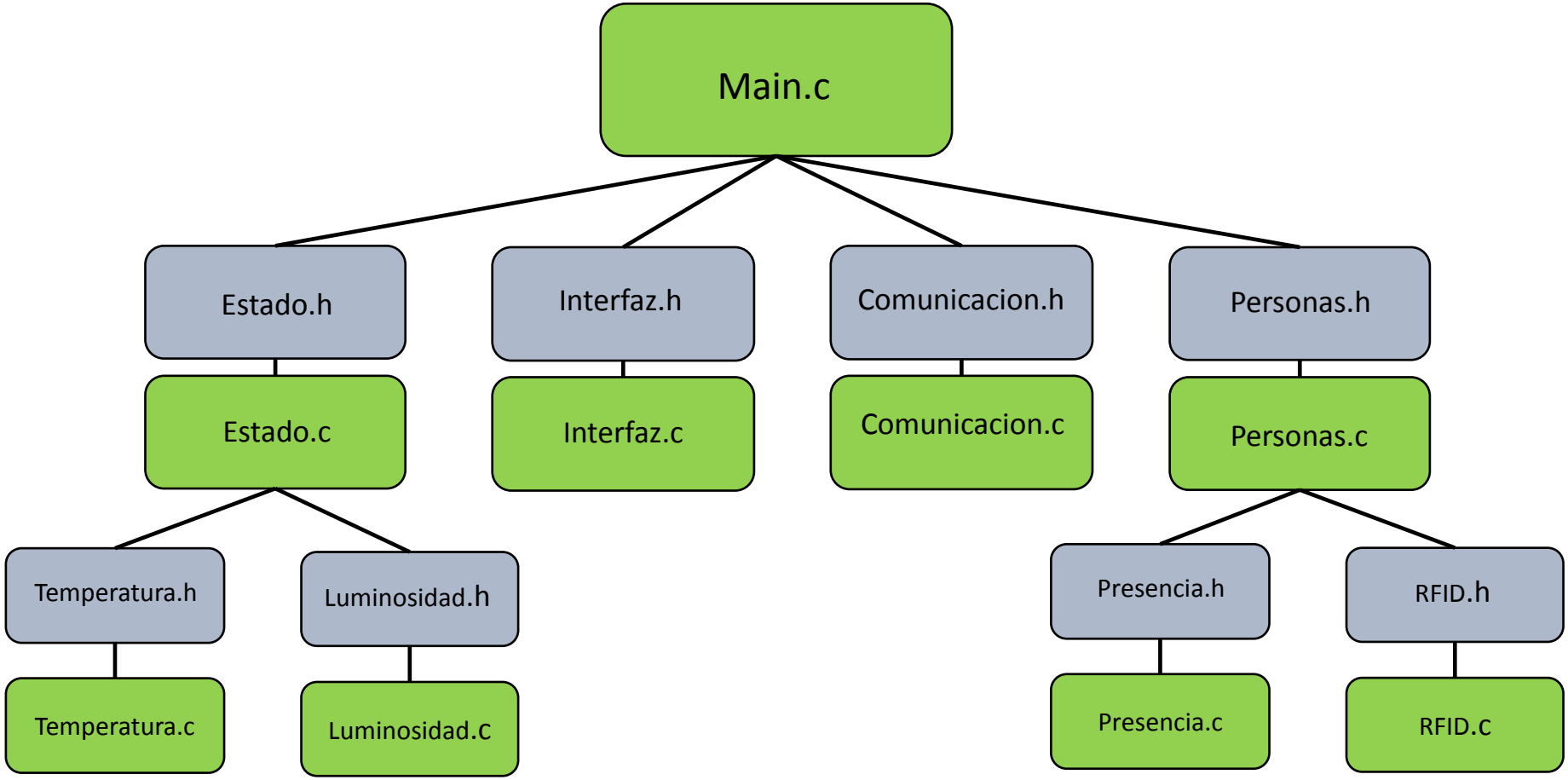
```
#include “utils_SDBM.h”

#define LEN 30

void rotate(int);

int nMensajes;
```

Fichero cabecera(.h)

```
#include “main.h”
#include “stdint.h”
#define MAXBUFFER 8
uint32_t mideTemp(void);
void init(void);
char nMedidas = 0;

int main (void){
   init();
   while (1){
      if (……);
         mideTemp();
      if(nMedidas == ….)
         ….
   }
}
void init(void){
  …
}
uint32_t mideTemp(void){
  …
  nMedidas++;
  return …;
}
void rotate(int){
  …
}
```

Fichero fuente (.c)

Ejemplo


Main.c
Estado.h
Interfaz.h
Comunicacion.h
Personas.h
Estado.c
Interfaz.c
Comunicacion.c
Personas.c
Temperatura.h
Luminosidad.h
Presencia.h
RFID.h
Temperatura.c
Luminosidad.c
Presencia.c
RFID.c

Tipos de datos en MDK-ARM

ANSI C (C89)

- Enteros
  - **char**
  - short
  - int
  - long
- Punto flotante (reales)
  - float
  - double
- Modificadores
  - short
  - long
  - unsigned
  - signed
- Punteros

Tamaño dependiente del procesador

Tipos fáciles de recordar (en stdint.h)

Tipos C99 y su redefinición para ARM

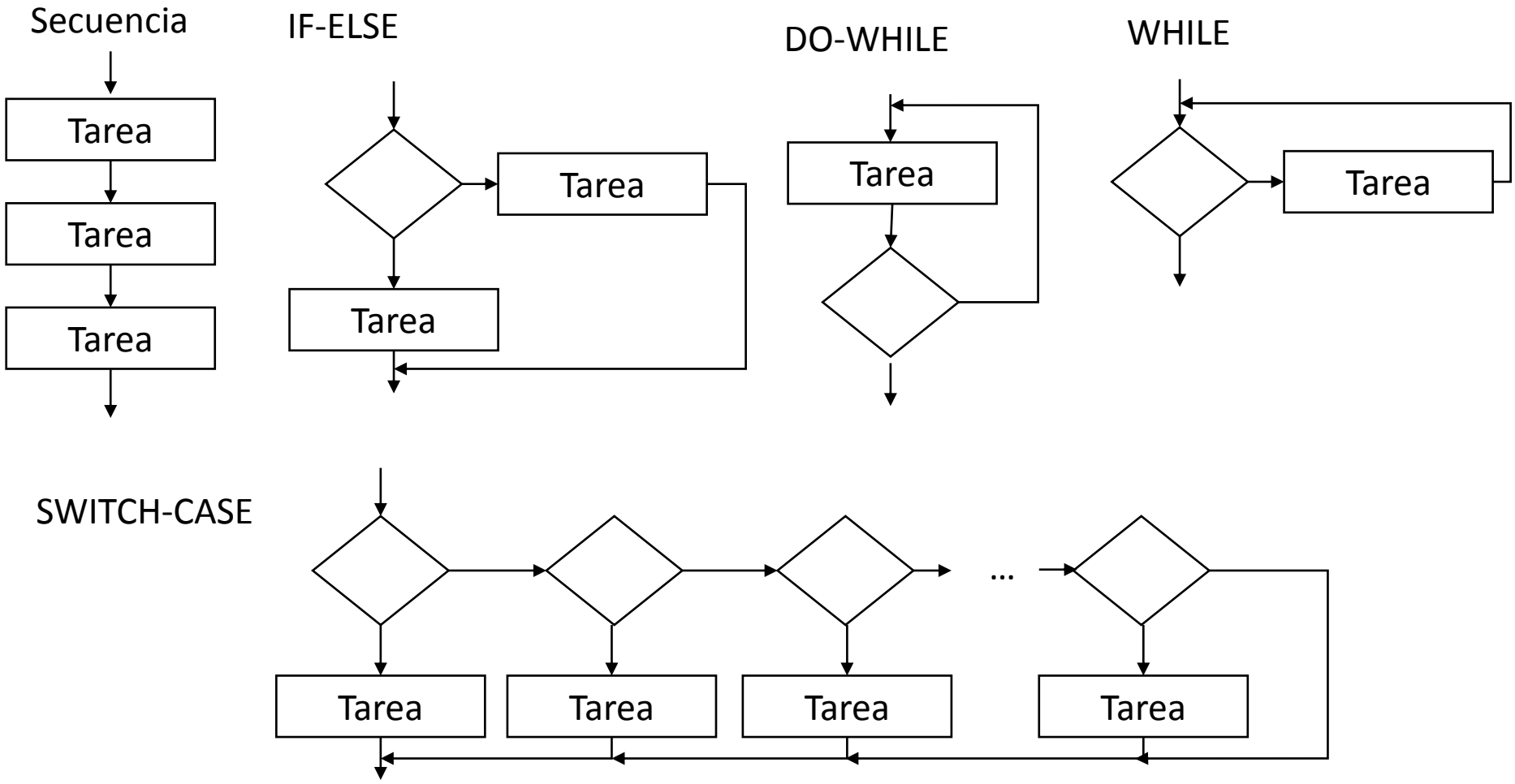
```
/* exact-width signed integer types */
typedef   signed char          int8_t;
typedef   signed short int     int16_t;
typedef   signed int           int32_t;
typedef   signed long long     int64_t;

   /* exact-width unsigned integer types */
typedef unsigned  char         uint8_t;
typedef unsigned short  int    uint16_t;
typedef unsigned int           uint32_t;
typedef unsigned long long     uint64_t;
```

Los tipos de datos “nativos” son enteros de 32 bits (se iguala int a long)

¡OJO! → Compilador armcc
Sin modificador el dato se interpreta como **unsigned si es char, signed para el resto**

Utilizar siempre el tamaño menor entre los posibles!

Estructuras comunes en C



Sentencias típicas (I)

Estructuras permitidas

IF

```
if(expresion)
{

}
else
{

}
```

SWITCH

```
switch(variable) {
        case const_expr1:
                statement1;
                break;
        case const_expr2:
                statement2;
                break;
        case const_expr3:
                statement3;
                break;
        default:
                statement0;
}
```

FOR

```
for(expr1;expr2;expr3)
{

}
```

```
for(i=1;i<10;i++)
        sum=sum+1;
```

Sentencias típicas (II)

Estructuras permitidas

WHILE

```
while(expresion)
{

}
```

```
while(i<10)
        sum=sum+1;
```

DO WHILE

```
do
        statement;

while(expresion)
```

Operaciones booleanas

Expresiones condicionales con variables en if(expresion) **& while**(expresion)

==	iguales
!=	no iguales
>	mayor que
>=	mayor o igual que
<	menor que
<=	menor o igual que
&&	and
||	or
!	not

Dato = 0 => Falso

Dato ≠ 0 => Verdadero

Operaciones con variables no boolenas

Aritméticos	
+	suma
-	resta
*	multiplicación
/	división
%	resto
++	incremento (+1)
--	decremento (-1)

Lógicos	
&	and
|	or
^	xor
~	not
>>	desplazamiento a derecha (extiende el signo)
<<	desplazamiento a izquierda

Uso de máscaras

```
#define bit_3 0x00000008U

regB |= bit_3;
regA &= ~ bit_3;
```

unisgned

tipos de datos

- Constantes
  - Valor de cualquier tipo que nunca cambia. Se prefiere almacenar en memoria de código para ahorrar RAM, se usa el identificador const en la declaración. Ejemplo: **const** char a =1;
- Variables
  - Una variable es una forma de referirnos a una posición de memoria utilizada en un programa, cuyo valor cambia durante la ejecución del programa. Ejemplo: char a;

Las variables se declaran al principio!!!

<tipo de dato> espacio(s) <identificador>;

Etiquetas

- Asigna un identificador a una constante

```
#define MAX_BUFFER  8
#define count_seg      13500
```

Con esto evitamos números mágicos

tipos de variables

- Globales
  - Declaradas fuera de las funciones en una cabecera o antes del main()
  - Espacio: cualquier punto del código fuente incluido en el mismo fichero. Se puede exportar a otro fichero, declarándola de nuevo con el identificador *extern*
  - Vida útil: ejecución del programa completo, es permanente
  - Consejo: **no abuse de ellas**, el espacio de RAM que consumen no puede reusarse. Solo si son indispensables (ISRs)
- Locales
  - Declaradas dentro de una función
  - Espacio: Limitado a la función en la que se declara
  - Vida útil: mientras la función se ejecuta, cuando la función termina su valor puede reescribirse
  - Consejo: úselas siempre que sea posible para ahorrar RAM

Puede usarse *static* para recuperar el último valor guardado entre ejecución y ejecución de la función

Puede usarse *volatile* para indicar que la variable puede ser modificada externamente (por el HW, o bien por una ISR). Es el caso de los registros mapeados en memoria

Punteros

- Variable que almacena una dirección de memoria.
- Operadores
  - **Operador de dirección (&)**
    Representa la dirección de memoria de la variable que le sigue:
    &var1 representa la dirección de **var1**.

  - **Operador de contenido o indirección (*)**
    El operador * aplicado al nombre de un puntero indica el valor de la variable apuntada.
    Este operador tiene muy **poca prioridad** (utilizar con paréntesis)

- Se declaran como:

<tipo de dato><*> espacio(s) <identificador>;

- Ejemplo:
  - int *a=NULL;
  - int b=2,c=1;
  - a=&b; /*Guarda la direc. de b en a */
  - c=*a; /*c vale 2 */

- Cuando se declara un puntero se reserva memoria para albergar una dirección de memoria, pero **NO PARA ALMACENAR EL DATO AL QUE APUNTA EL PUNTERO. → Solo una posición de memoria**

Arrays y Matrices

- Array

  char a[3]; //Array de longitud 3

  a se gestiona como un puntero que apunta al principio del array. a[0]=1 es equivalente a *a=1

  Para pasarlo como argumento de una función, se usa la referencia o puntero
  void func(char *a), en lugar de pasar el array (que ocuparía lo mismo que los 3 valores)

- Matriz

  int b[4][5]; //matriz de 4 filas y 5 columnas

  int a[2][3]={ { 1, 2, 3}, {4, 5, 6} }; //inicializacion

Struct

- Los usamos para agrupar datos que van consecutivos en memoria, y que pueden tener distintos tipos dentro

```
typedef struct {
        unsigned char car1;
        signed int num2;
}AlphaNum;

AlphaNum var1;
AlphaNum *pvar1;

var1.car1='A';
var1.num2=310;

pvar = &var1;

pvar1->car1='B';
Pvar1->num2=350;
```

Paso por valor / paso por referencia

- Al llamar una función, podemos pasar los parámetros por valor o por referencia
- **No se declaran las variables que llegan como parámetro!**

- Paso por valor:
  - Pasamos el 'valor' del dato

    int sumar (int op1, int op2) ;
- Paso por referencia:
  - Pasamos la 'dirección' del dato

    int sumar (int *pop1, int *pop2) ;

Como los parámetros por valor generan copias, éstas no se modifican

Si se necesita que el valor de la variable que se pasa como parámetro se afecte, se pasa como parámetro por referencia.

Paso por valor / paso por referencia

- Cuando llamamos a una función, siempre se genera una copia de las variables que le pasamos en los registros o en la pila.

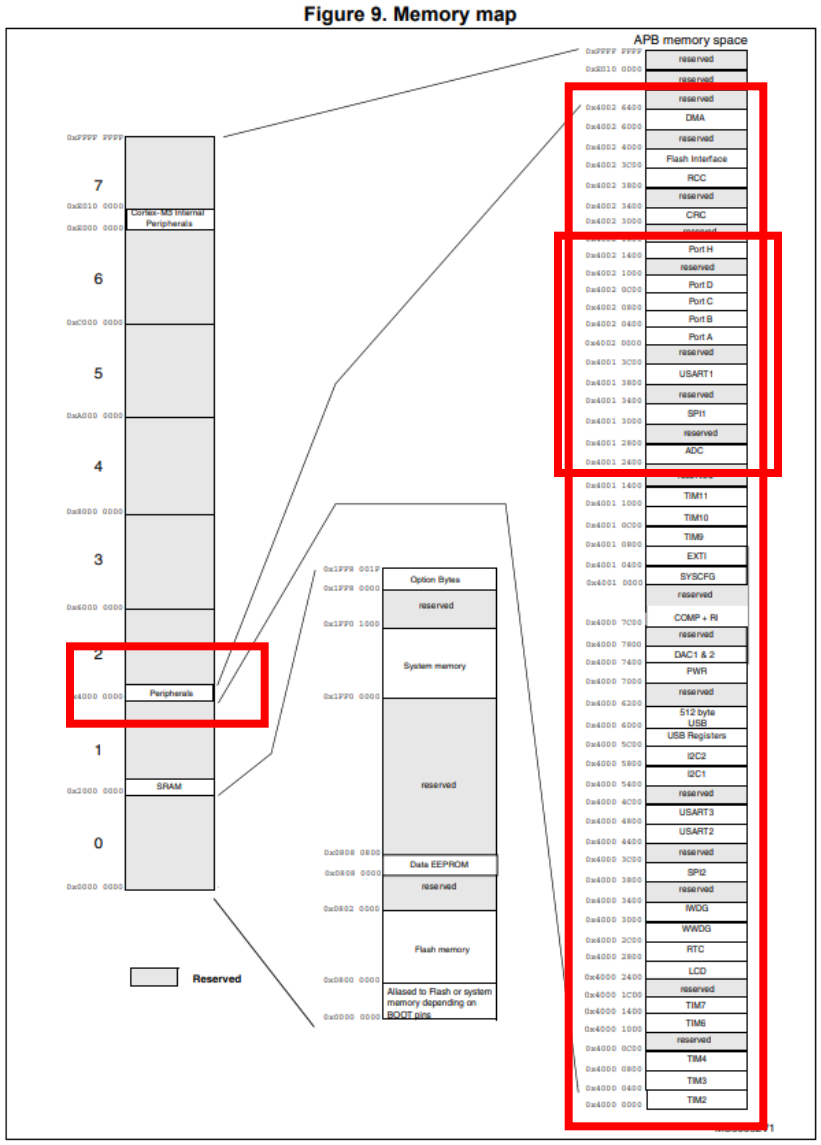
- Pero… y si necesitamos pasar un struct??

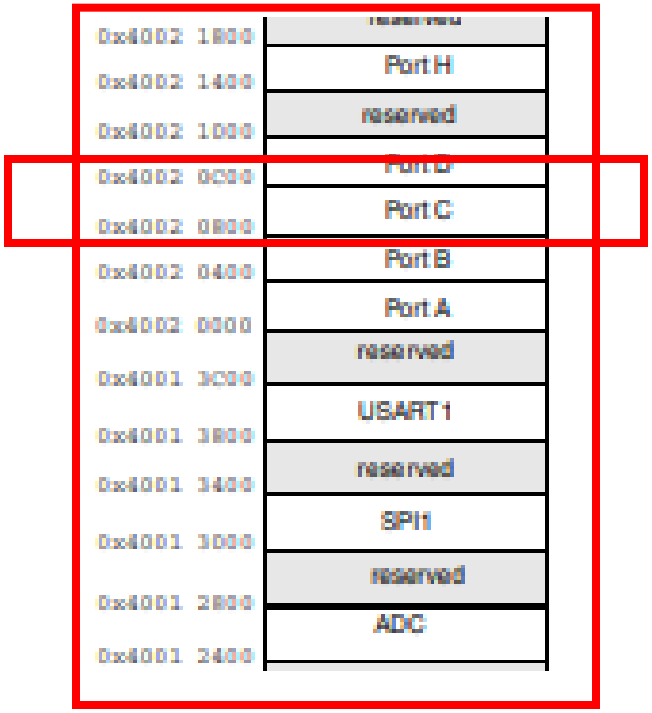
```
struct informacion{
        int id;
        char nombre[20];
        char email[40];
} info, *pinfo;


void updateInfo(info i);
void updateInfo(info* pinfo);
```

Periféricos mapeados en memoria

- En el microcontrolador hay un conjunto de registros mapeados en direcciones de memoria para poder usar los distitntos periféricos(puertos, timers,…).
- Se define un puntero a esa zona de memoria de forma que su modificación sea más sencilla.
- Estas definiciones están declaradas en un fichero de cabecera (stm32l152xc.h)

Ej. – Periféricos mapeados en memoria





Ej. – Periféricos mapeados en memoria

7.3.3 I/O port control registers

Each of the GPIOs has four 32-bit memory-mapped control registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR) to configure up to 16 I/Os.

The GPIOx_MODER register is used to select the I/O direction (input, output, AF, analog). The GPIOx_OTYPER and GPIOx_OSPEEDR registers are used to select the output type (push-pull or open-drain) and speed (the I/O speed pins are directly connected to the corresponding GPIOx_OSPEEDR register bits whatever the I/O direction). The GPIOx_PUPDR register is used to select the pull-up/pull-down whatever the I/O direction.

7.3.4 I/O port data registers

Each GPIO has two 16-bit memory-mapped data registers: input and output data registers (GPIOx_IDR and GPIOx_ODR). GPIOx_ODR stores the data to be output, it is read/write accessible. The data input through the I/O are stored into the input data register (GPIOx_IDR), a read-only register.

See *Section 7.4.5: GPIO port input data register (GPIOx_IDR) (x = A..H)* and *Section 7.4.6: GPIO port output data register (GPIOx_ODR) (x = A..H)* for the register descriptions.

7.4.6 GPIO port output data register (GPIOx_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

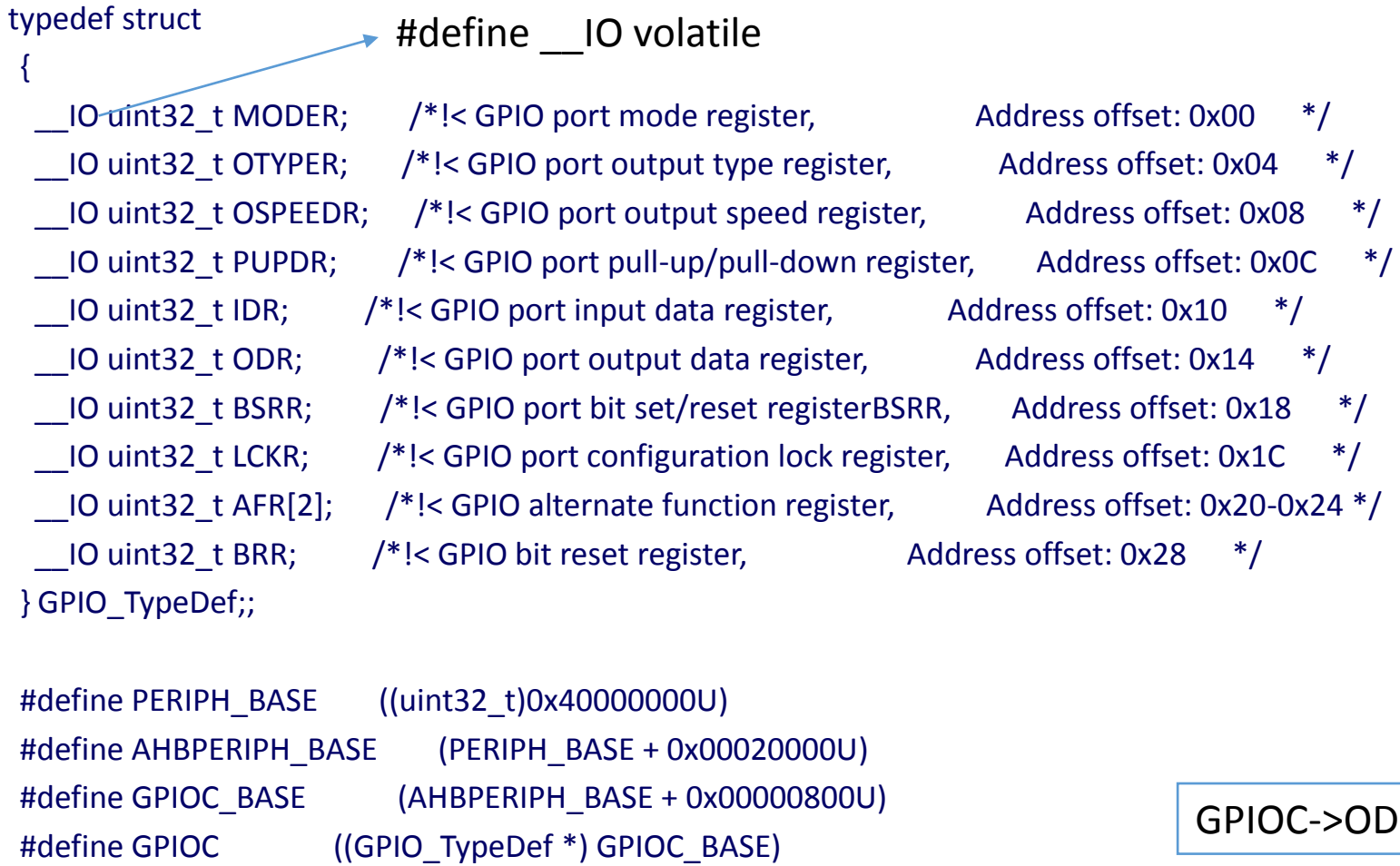
Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..H).

Struct en MDK-ARM

- En el fichero stm32l152xc.h encontramos



```
typedef struct
 {
  __IO uint32_t MODER;      /*!< GPIO port mode register,               Address offset: 0x00     */
  __IO uint32_t OTYPER;     /*!< GPIO port output type register,        Address offset: 0x04     */
  __IO uint32_t OSPEEDR;    /*!< GPIO port output speed register,       Address offset: 0x08     */
  __IO uint32_t PUPDR;      /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C     */
  __IO uint32_t IDR;        /*!< GPIO port input data register,         Address offset: 0x10     */
  __IO uint32_t ODR;        /*!< GPIO port output data register,        Address offset: 0x14     */
  __IO uint32_t BSRR;       /*!< GPIO port bit set/reset registerBSRR,  Address offset: 0x18     */
  __IO uint32_t LCKR;       /*!< GPIO port configuration lock register, Address offset: 0x1C     */
  __IO uint32_t AFR[2];     /*!< GPIO alternate function register,      Address offset: 0x20-0x24 */
  __IO uint32_t BRR;        /*!< GPIO bit reset register,               Address offset: 0x28     */
 } GPIO_TypeDef;;

 #define PERIPH_BASE       ((uint32_t)0x40000000U)
 #define AHBPERIPH_BASE    (PERIPH_BASE + 0x00020000U)
 #define GPIOC_BASE        (AHBPERIPH_BASE + 0x00000800U)
 #define GPIOC             ((GPIO_TypeDef *) GPIOC_BASE)
```

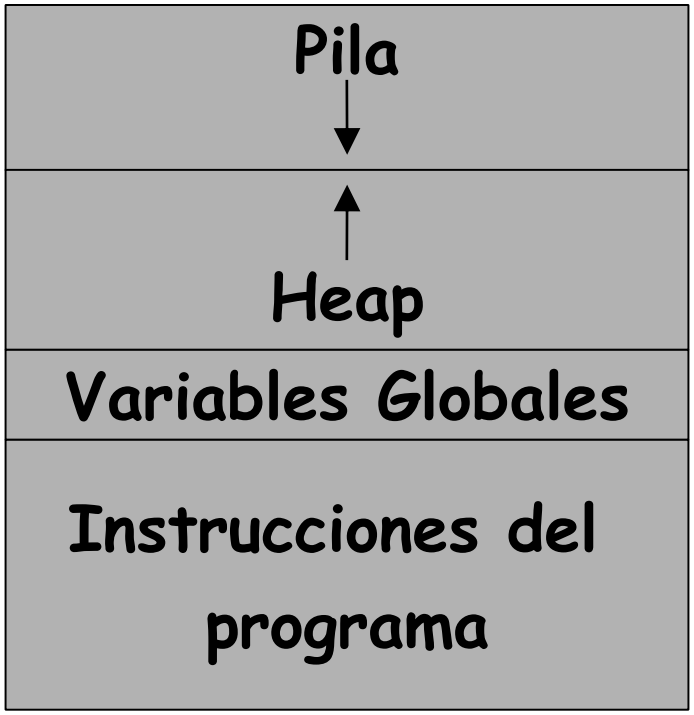
```
#define __IO volatile
```

```
GPIOC->ODR=3;
```

Asignación de memoria

- ¿La información puede cambiar?
  - No. Se asigna a memoria de sólo lectura
    - Programa
    - Operandos constantes
    - Valores de inicialización
  - Sí. Se asigna a memoria de lectura/escritura
    - Variables
    - Cálculos intermedios
    - Direcciones de retorno
    - Otros datos

Asignación de memoria



- ¿Cuánto tiempo necesitan preservarse los datos?
  - Asignados estáticamente (variables globales)
    - Existen desde el principio hasta el final del programa
    - Cada variable tiene localización fija
    - Su espacio no se reutiliza
    - Accesibles desde cualquier función
  - Asignados automáticamente (variables locales)
    - Existen desde el principio hasta el final de una función
    - Su espacio puede reutilizarse
    - Accesibles dentro de una función
  - Asignados dinámicamente (no se usarán)
    - Existen desde que se asignan hasta que se liberan (explícitamente)
    - Su espacio puede reutilizarse

Prólogo y Epílogo de funciones

- El compilador genera código a la entrada y a la salida de cada función
- Prólogo
  - Guarda registros de uso general
  - Gestiona los argumentos de la función
- Epílogo
  - Restaura registros
  - Vuelve a la función que hizo la llamada

```
   249: void arrays(unsigned char n, unsigned char j) {
   250:   volatile int i;
0x0800022C B508      PUSH     {r3,lr}
   251:   i = buff2[0] + buff[n];
```

```
   252:   i += buff3[n][j];
)x0800023C EB000280  ADD      r2,r0,r0,LSL #2
)x08000240 4B06      LDR      r3,[pc,#24]  ; @0x0800025C
)x08000242 EB0302C2  ADD      r2,r3,r2,LSL #3
)x08000246 F8522021  LDR      r2,[r2,r1,LSL #2]
)x0800024A 9B00      LDR      r3,[sp,#0x00]
)x0800024C 441A      ADD      r2,r2,r3
)x0800024E 9200      STR      r2,[sp,#0x00]
   253: }
)x08000250 BD08      POP      {r3,pc}
```

Argumentos de función y valor de retorno (1)

- ¿Cómo se pasan los argumentos y el resultado?
  - Se hace por registros (R0-R3 pila si es necesario)
  - El número de registros es limitado

- ¿Dónde se guardan las variables locales?
  - Registros (R4.... + pila si es necesario)
  - El número de registros es limitado

Argumentos de función y valor de retorno (2)

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.