Desarrollo de software en sistemas embebidos

Electrónica IV

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165

Lenguaje C

- Desarrollado en los laboratorios Bell (AT&T) entre 1969 y 1973 por Kernighan y Ritchi.
- Pensado para escribir el Kernel de Unix, que antes era escrito en ensamblador.
- Amplia difusión en los sistemas embebidos.
- Permite reutilizar el código entre distintas plataformas.

Lenguaje C

- K&R 1969 al 1973
- K&R 1978
- K&R Second Edition 1988
- C89 ⇛ ANSI X3.159-1989
- C90 ⇛ ISO/IEC 9899:1990 (Mayor nivel de compatibilidad)
- C95 ⇛ ISO/IEC 9899:1995
- C99 ⇛ ISO/IEC 9899:1999
- C11 ⇛ ISO/IEC 9899:2011
- C17 ⇛ ISO/IEC 9899:2018
- C23 ⇛ ISO/IEC 9899:2024 (Todavía en revisión)

Lenguaje C90

- Primera versión del estándar
- 100 % compatible con C89 y anteriores
- Solo soporta comentarios con `/* */`
- No soporta tipos de 64 bits
- Definición de variables al inicio de un bloque
- Definición de la estructura sin nombre
- No se soporta **`inline`**
- Es el mínimo común denominador: Si se programa en C90 siempre se podrá compilar el código

Lenguaje C

- El tipo de datos **`int`** permite la definición de un número entero con signo
- El tamaño puede cambiar según la plataforma
  - Se representa como mínimo con 16 bits
  - En general coincide con el tamaño de palabra del procesador
  - Se puede modificar con **`signed`**, **`unsigned`**, **`short`** y **`long`**

Lenguaje C

- El estándar C99 agrega la biblioteca **`stdint.h`**
- Define tipos de datos con tamaño fijo
- Facilita la portabilidad del código

Nombre	Signo	Tamaño
`int8_t`	con signo	8 bits
`int16_t`	con signo	16 bits
`int32_`	con signo	32 bits
`uint8_t`	sin signo	8 bits
`uint16_t`	sin signo	16 bits
`uint32_`	sin signo	32 bits

Declaración y definición en C

- Una variable se declara como

```
extern int suma;
```

- Una variable se define como

```
int suma;
```

- Una función se declara con un prototipo

```
int main(int argc, char *argv[]);
```

- Una función se define como

```
int main(int argc, char *argv[]) {
    ...
}
```

La directiva `const`

- Declara una variable como constante
- También califica a un parámetro para impedir que el mismo sea alterado dentro de una función.
- También califica a un puntero para saber si se puede cambiar la memoria apuntada por dicho puntero.
- En los sistemas embebidos implica además que la misma se ubica en FLASH y no en RAM.

Declaraciones de constantes

- Puntero a un entero constante

```
int const * foo;
const int * foo;
```

- Puntero constante a un entero

```
int * const foo;
```

- Puntero constante a un entero constante

```
int const * const foo;
const int * const foo;
```

Variables y constantes

- Las variables se convierten automáticamente a constantes
- El siguiente código compila y funciona sin problemas

```
int suma(const int * a, const int * b) {
    return (*a + *b);
}
void main(void) {
    int r, a = 10, b = 15;
    r = suma(&a, &b);
    ...
}
```

La directiva `volatile`

- Se aplica a variables locales o globales.
- Indica al procesador que el contenido puede cambiar en cualquier momento sin intervención del código.
- Desactiva la optimización de código para la variable: Se lee el valor de memoria en cada acceso.
- Por ejemplos/ejemplo se aplica en:
  - Registros de entrada/salida
  - Variables compartidas con interrupciones
  - Variables compartidas entre procesos

La directiva `static`

- En variables locales
  - Conservan el valor entre llamadas
  - Las convierte en variables globales
  - Sin visibilidad para el resto del código
- En variables globales y procedimientos
  - No se exporta el símbolo en el archivo objeto
  - Impide el uso de **`extern`** en otro modulo
  - Sin visibilidad para el resto del proyecto

Operadores a nivel de bits

- **`A & B`** realiza un AND bit a bit entre A y B
  - **`B = 0x13; A = 0x0F & B;`** → A = 0x03

- **`A | B`** realiza un OR bit a bit entre A y B
  - **`B = 0x13; A = 0x0F | B;`** → A = 0x1F

- **`A ^ B`** realiza un XOR bit a bit entre A y B
  - **`B = 0x13; A = 0x0F ^ B;`** → A = 0x1C

Operadores a nivel de bits

- `~A` realiza la inversión de los bits de A
  - **`B = 0x13; A = ~B;`** → A = 0xEC

- **`A << B`** corre B bits a izquierda el valor A
  - **`B = 0x13; A = B << 1;`** → A = 0x26

- **`A >> B`** corre B bits a derecha el valor A
  - **`B = 0x13; A = B >> 1;`** → A = 0x09

Mascaras de bits

- Verificar el estado de un bit en una variable:
  - **`(A & (1 << 3))`** → true si el bit 3 de A es uno.
- Cambiar un bit determinado en una variable:
  - **`A |= (1 << 3);`** → pone en uno el bit 3 de A
  - **`A &= ~(1 << 3);`** → pone en cero el bit 3 de A
  - **`A ^= (1 << 3);`** → invierte el bit 3 de A

Estructuras

- Las estructuras son tipos de datos compuestos formada por campos
- Siempre ocupa direcciones de memoria consecutivos
- El espacio ocupado por la estructura puede ser mayor que la suma de todos los campos
- Se puede limitar el tamaño de los campos a fracciones de bytes

Acceso a los dispositivos del MCU

- Se declara una estructura donde cada campo corresponde a un registro del dispositivo
- Se declara un puntero a esa estructura
- Se declara una constante como un cast al puntero de la dirección de memoria base de dispositivo

Estructuras y punteros

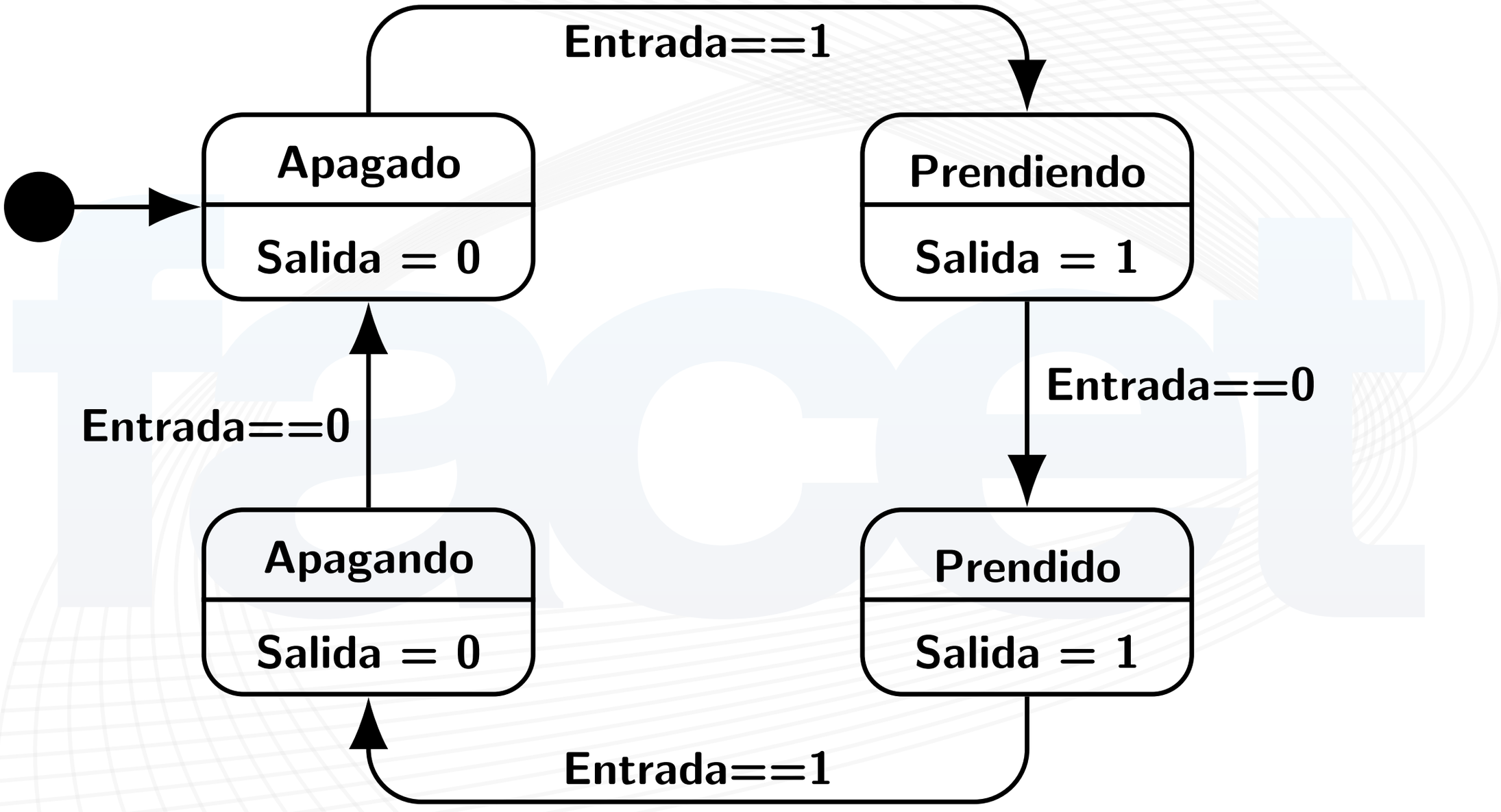
```
typedef struct salida_s {
   void * puerto;
   uint8_t terminal;
   struct opciones_s {
      bool invertida : 1;
      bool activa : 1;
   } opciones;
} * salida_t;

struct salida_s led;
led.puerto = PORT_A;
led.terminal = 1;
led.opciones.invertida = TRUE;
Activar(&led);
```

```
static const salida_t
led = &(struct salida_s)
{
   .puerto = PORT_A,
   .terminal = 1,
   .opciones = {
      .invertida = TRUE,
      .activa = FALSE
   }
}

Activar(led)
```

Máquinas de estados finitos



Estructura de datos

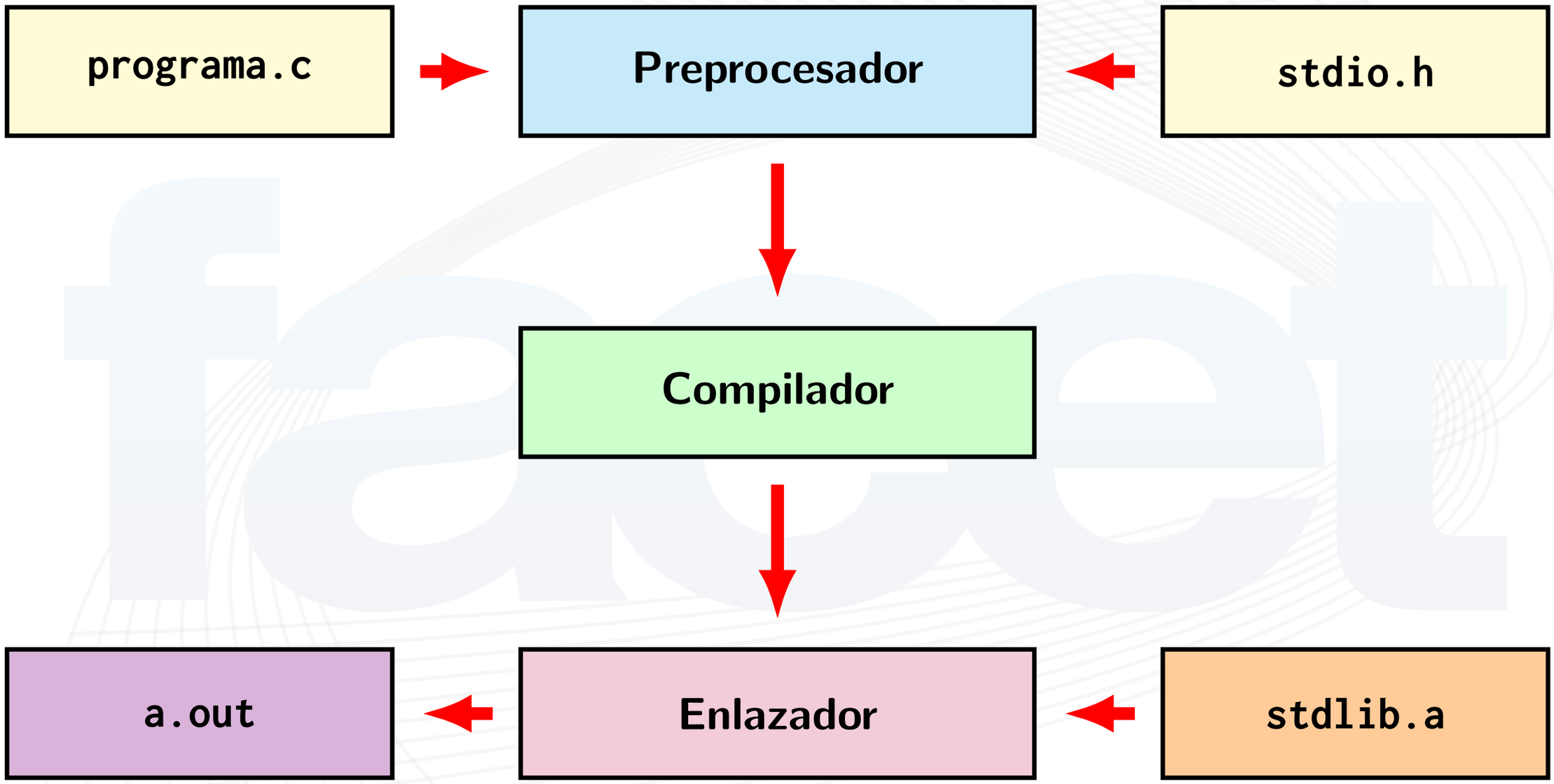
```
typedef enum estado_e {
      APAGADO,
      PRENDIEDO,
      PRENDIDO,
      APAGANDO,
} estado_t;

static estado_t estado;
```

Implementación

```
switch (estado) {
      case APAGADO:
            salida = 0;
            if (entrada == 1) estado = PRENDIEDO;
      break;
      case PRENDIEDO:
            salida = 1;
            if (entrada == 0) estado = PRENDIDO;
      break;
      ...
}
```

Compilación de un archivo C



Preprocesador

- Toma un archivo de código fuente y entrega un archivo preprocesado.
- El procesador tiene como tareas:
  - Expandir los macros.
  - Expandir los archivos incluidos.
- El resultado es un archivo autocontenido listo para compilar.
- En general el preprocesamiento se hace automáticamente antes de compilar.

Compilador

- El proceso de compilación toma un archivo preprocesado y entrega un archivo con código objeto.
- El compilador tiene como tareas:
  - Traducir el código C a código binario ejecutable por el procesador.
  - Generar una tabla con las referencias externas al archivo.
- El resultado es un archivo con secuencias de código ejecutable.

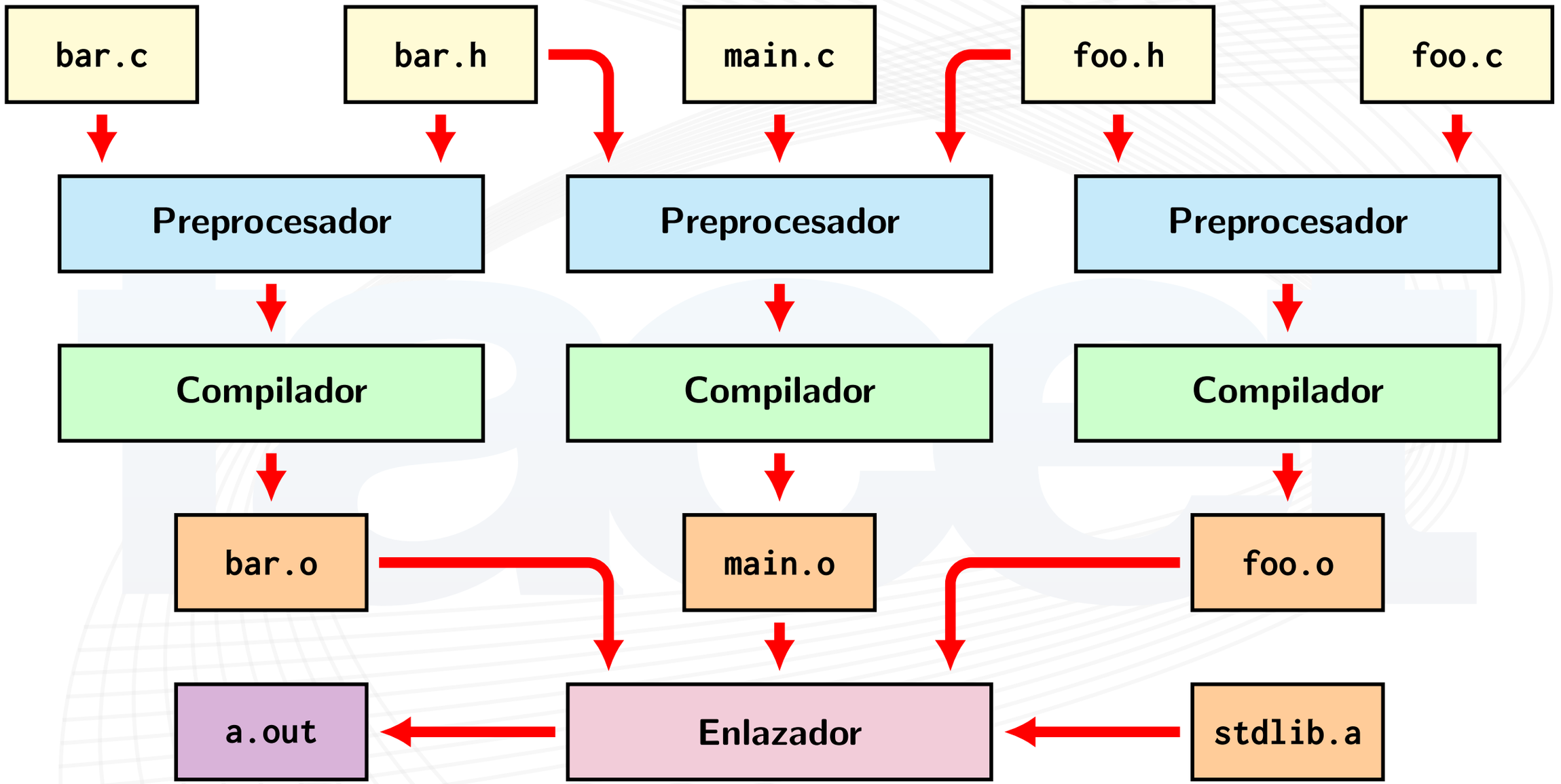
Enlazador

- El proceso de enlazado toma todos los archivos con código objeto y entrega un archivo ejecutable.
- El enlazador tiene como tareas:
  - Ubicar cada segmento de código o de variables en áreas especificas de memoria.
  - Resolver las referencias cruzadas entre las secuencias de código objeto.
- El resultado es un archivo ejecutable.

Compilación de un proyecto

- Un proyecto esta formado normalmente por varios archivos de cabeceras, fuentes y bibliotecas.
- Se debe compilar cada archivo de código fuente por separado
- Se debe enlazar los archivos de código objeto con las bibliotecas.

Compilación de un proyecto



Compilando un proyecto

- `gcc main.c`
- `gcc main.c foo.c bar.c`

Error

- `gcc -c main.c -o main.o`
- `gcc -c foo.c -o foo.o`
- `gcc -c bar.c -o bar.o`
- `gcc -o a.out bar.o main.o foo.o`

Make y Makefile

- Make es una herramienta pensada para automatizar el proceso de compilación de proyectos voluminosos.
- Utiliza un archivo llamado makefile que define una serie de reglas en función de las cuales make ejecuta las acciones necesarias.
- Forma parte de las herramientas del estándar POSIX.

Un makefile para el proyecto

```
all: foo.o bar.o main.o
    gcc -o a.out foo.o bar.o main.o

foo.o: foo.c
    gcc -c foo.c

bar.o: bar.c
    gcc -c bar.c

main.o: main.c
    gcc -c main.c
```

Un makefile más inteligente

```
SRC_DIR = ./source
OBJ_DIR = ./build
SRC_FILES = $(wildcard $(SRC_DIR)/*.c)
OBJ_FILES = $(patsubst $(SRC_DIR)%.c,
            $(OBJ_DIR)%.o,$(SRC_FILES))

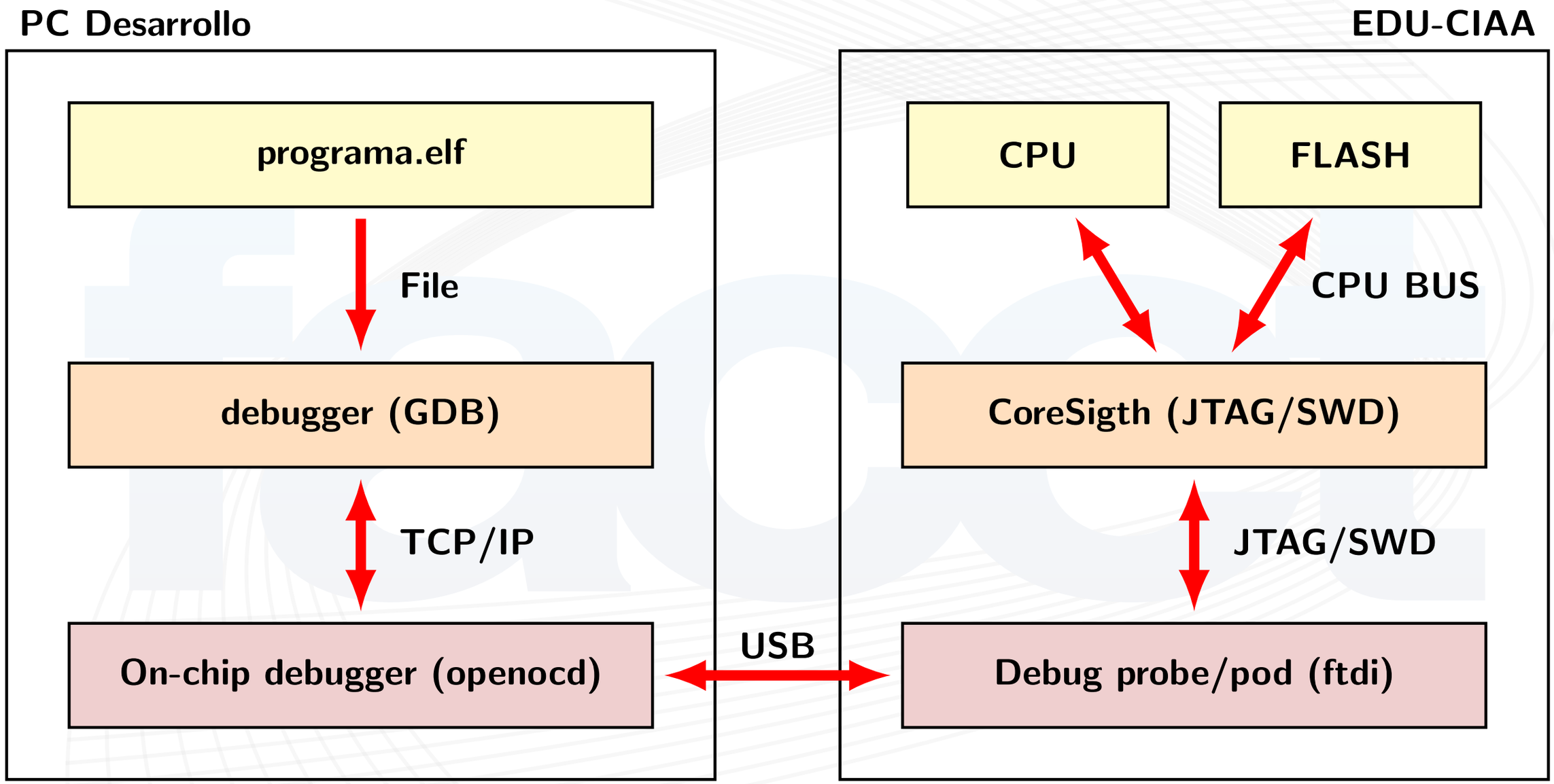
all: $(OBJ_FILES)
  gcc -o $(OBJ_DIR)/a.out $(OBJ_FILES)

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.c
  gcc -c $< -o $@
```

Grabación y depuración

- Cuando se compila un proyecto para un sistema embebido el resultado es una imagen con el contenido de la memoria flash.
- Esta imagen debe grabarse en la memoria interna del microcontrolador.
- Para ello el procesador dispone de un modulo destinado a comunicarse con las herramientas de desarrollo.
- Las interfaces mas difundidas para las herramientas de desarrollo son JTAG (Join Test Action Group) y SWD (Serial Wire Debug).

Diagrama de Bloques



¿Que es el testing de software?

- Es el proceso centrado en el objetivo de encontrar defectos en un sistema.
- Prueba Estáticas (Verificación)
  - El código no es ejecutado
  - Verificar el cumplimiento del estándar
- Pruebas Dinámicas (Test)
  - El código es ejecutado
  - Verificar el cumplimiento de los requisitos

Verificación

- Pueden ser realizadas por herramientas automáticas (Linter)
  - Verifican cumplimiento de los prototipos
  - Eliminan situaciones ambiguas
- También pueden ser realizadas por otra persona (Review)
  - Mejora la calidad del software
  - Mejora la calidad del programador
  - Mejora el trabajo en equipo

Coding Gidelines

- Es un conjunto de convenciones sobre como escribir el código:
  - Aumentan la legibilidad
  - Disminuyen los errores
- Hay Guidelines publicas y pagas
  - MISRA -C
  - NASA SEL-94-003

MISRA 2004

- Rule 1.1 Code shall be conform to C90
- Rule 6.1 The plain char type shall be used only for the storage and use of character values.
- Rule 8.1 Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.
- Rule 8.8 An external object or function shall be declared in one and only one file.
- Rule 9.1 All automatic variables shall have been assigned a value before being used.

Clang-Format

- Es una herramienta para ajustar y/o verificar el formato de código fuente en C.
- Soporta varios estilos conocidos como Google, LLVM, Chromium, Mozilla, WebKit, Microsoft y GNU.
- Sobre los formatos de base se pueden ajustar muchos parámetros.
- La configuración se puede almacenar en un archivo dentro del repositorio.

Cppcheck

```
1  #include <stdio.h>
2  int main()
3  {
4    char c;
5    while (c != 'x');
6    {
7      c = getchar();
8      if (c = 'x')
9      return 0;
10     switch (c) {
11       case '\n':
12       case '\r':
13         printf("NewLine\n");
14       default:
15         printf("%c", c);
16     }
17   }
18   return 0;
19 }
```

- Variable c used before definition
- Suspected infinite loop. No value used in loop test (c) is modified by test or loop body.
- Assignment of int to char: c =getchar()
- Test expression for if is assignment expression: c = 'x'
- Test expression for if not boolean, type char: c = 'x'
- Fall through case (no preceding break)

Los hook de Git

- Git permite definir funciones de usuario que se ejecutan al realizar un commit o un push.
- En los hooks se pueden instalar scripts que revisen o cambien el formato el código.
- Un ejemplo es https://pre-commit.com/
- Esta herramienta tambien guarda la configuración en un archivo dentro del mismo repositorio.

Diagramas UML con Plantuml

- Permite generar la mayoría de los diagramas de UML para documentar el diseño del proyecto.
- Los diagramas se generan a partir de un archivo de texto con comandos muy sencillos.
- Es muy fácil de mantener y muy rápido dado que no hay que preocuparse por la parte gráfica.

Diagrama de actividades

```
@startuml
title Probar Sirenas
start
if (Armado = 0xFF) then (Si)
else (No)
     :Datos = ESTADO;
     :Datos = Datos | (3 << 2);
     :Puerto = 0;
     :Enviar(Dato,Puerto)|
endif
stop
@enduml
```

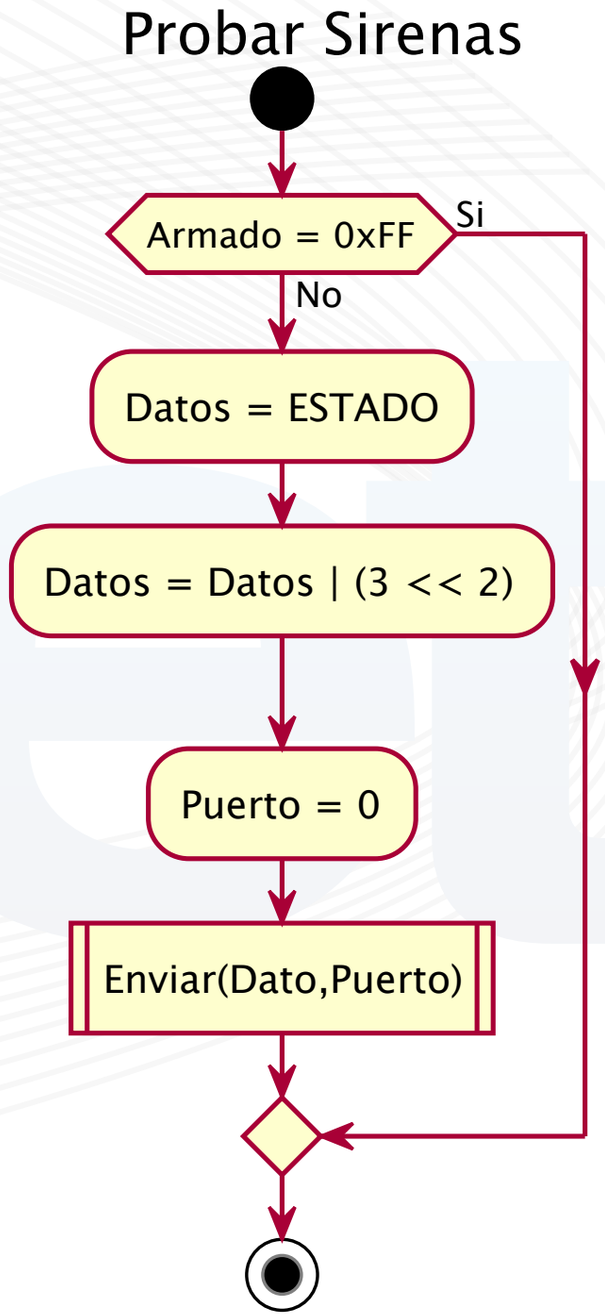
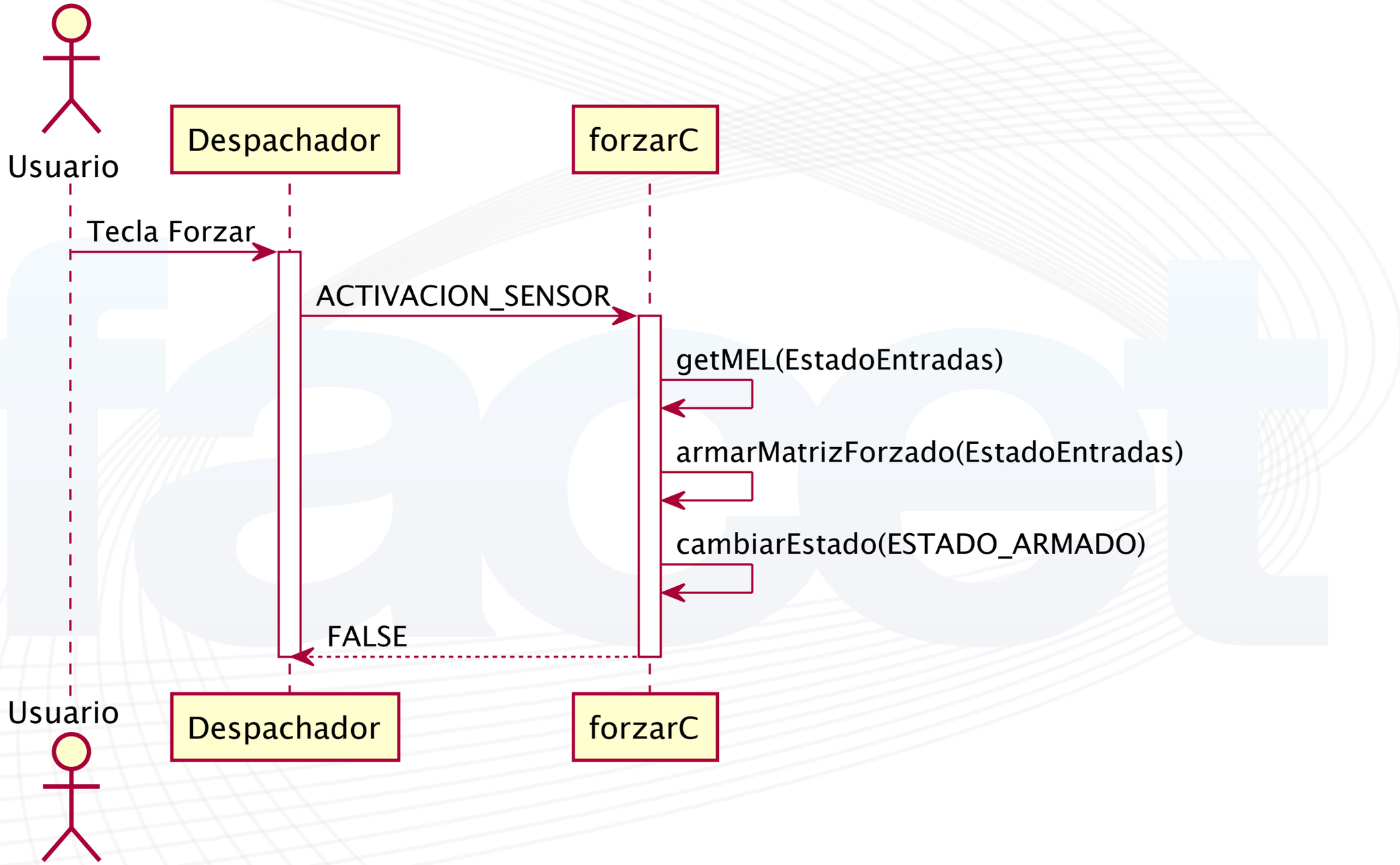


Diagrama de secuencia

```
@startuml
actor Usuario
participant Despachador
participant forzarC
Usuario -> Despachador: Tecla Forzar
activate Despachador
Despachador -> forzarC: ACTIVACION_SENSOR
activate forzarC
forzarC -> forzarC: getMEL(EstadoEntradas)
forzarC -> forzarC: armarMatrizForzado(EstadoEntradas)
forzarC -> forzarC: cambiarEstado(ESTADO_ARMADO)
forzarC --> Despachador: FALSE
deactivate Despachador
deactivate forzarC
@enduml
```

Diagrama de Secuencia



Markdown

- Permite generar páginas HTML a partir de texto plano con marcas simples
- Mantiene la facilidad de lectura en el archivo de texto fuente
- Muy utilizado para documentación:
  - Wikipedia
  - GitHub
  - Reddit

Ejemplo de Markdown

```
# Trabajo Practico Numero 1

Repositorio inicial para el Trabajo Practico 1 de la asignatura de
Sistemas Embebidos

## Primera Parte

1. Clonar este repositorio en su computadora y desplegar la rama
*master*.

2. Agregar al archivo alumnos una funcion que serialice sus datos
personales similar a la del siguiente ejemplo. La declaración y la
definición de la funcion debe agregarse abajo de las existentes.

    ```c
    bool EstebanVolentini(char * cadena, size_t espacio) {
        struct alumno_s alumno = {
            .apellidos = "VOLENTINI",
            .nombres = "Esteban Daniel",
            .documento = "23.517.968",
        };

        SerializarAlumno(cadena, sizeof(cadena), &alumno);
    }
    ```

3. Confirmar los cambios, resolver los conflictos y subir los
cambios al servidor.
```

Trabajo Practico Numero 1

Repositorio inicial para el Trabajo Practico 1 de la asignatura de Sistemas Embebidos

Primera Parte

1. Clonar este repositorio en su computadora y desplegar la rama *master*.
2. Agregar al archivo alumnos una funcion que serialice sus datos personales similar a la del siguiente ejemplo. La declaración y la definición de la funcion debe agregarse abajo de las existentes.

```c
bool EstebanVolentini(char * cadena, size_t espacio) {
    struct alumno_s alumno = {
        .apellidos = "VOLENTINI",
        .nombres = "Esteban Daniel",
        .documento = "23.517.968",
    };

    SerializarAlumno(cadena, sizeof(cadena), &alumno);
}
```

3. Confirmar los cambios, resolver los conflictos y subir los cambios al servidor.

Documentación con Doxygen

- Permite generar documentación de estructuras, tipos y funciones
- Se genera a partir de comentarios especiales en el código fuente
- La documentación puede generarse en HTML, Latex, PDF, Man Pages, RTF y XML
- También puede documentase parte del diseño utilizándolo con plantuml y markdown

Ejemplo de documentación

```
//! Define un entero de un byte sin signo.
typedef unsigned char UINT8;

/*! @brief Cambia el estado de la sirena.
Fija un nuevo estado para la sirena. En el caso
que estado sea 0, la sirena no sonará. En el
caso que estado sea 1, la sirena sonará.
@param[in] estado Nuevo estado que se asigna a
la sirena:
      @li @c 0 La sirena se apaga.
      @li @c 1 La sirena se prende.
 */
void setSirena(UINT8 estado);
```

Resultado en HTML