Herramientas de Software

Diseño Integrado de Sistemas Embebidos

Eduardo Daniel Cohen
dcohen@herrera.unt.edu.ar
http://www.microprocesadores.unt.edu.ar/sistemasembebidos

Entorno de desarrollo

- Desarrollo en Lenguaje C
  - Concepto de programacion portable
  - Automatización de la compilación
- Herramientas de desarrollo
  - Control de versiones
  - Documentación

Programación en C

- Desarrollado en los laboratorios Bell (AT&T) entre1969-1973 por Kernighan y Ritchi
- Pensado para escribir el Kernel de Unix, antes era escrito en assembler
- Amplia difusión en los sistemas embebidos
- Permite reusar el código entre distintas plataformas

Estructura de un proyecto

- Archivos H:
  - Definen tipos de datos y macros publicos
  - Declaran variables compartidas
  - Declaran funciones publicas
- Archivos C:
  - Definen tipos de datos y macros internos
  - Declaran y definen variables privadas
  - Definen variables publicas
  - Declaran y definen funciones privadas
  - Definen las funciones publicas

Diseño orientado a objetos

- Para un diseño orientado a objectos
  - Un archivo C implementa una clase.
  - Los métodos privados se declaran en el propio archivo C.
  - Los métodos publicos se declaran en el archivo H correspondiente.
  - Es recomendable que no existan variables globales compartidas.

Versiones de C

- K&R 1969 al 1973
- K&R 1978
- ANSI C → K&R Second Edition 1988
- ANSI C or C89 → ANSI X3.159-1989
- C90 → ISO/IEC 9899:1990
- C95 → ISO/IEC 9899:1995
- C99 → ISO/IEC 9899:1999
- C11 → ISO/IEC 9899:2011

C90

- Primera versión estándar (100% compatible C89)
- Solo soporta comentarios con /* */
- No se soporta tipos de 64 bits
- No se soporta definición de variables fuera del comienzo de un bloque.
- Definición de valores de la estructura sin nombre.
- No se soportan arrays de largo variables.
- No se soporta funciones inline
- Es el mínimo común denominador, si se programa en C90 siempre se podrá compilar el código.

Programación portable

- El lenguaje C tiene muchas variantes, ANSI, C90, C99, C11 y además extensiones especificas de cada compilador.
- Cuando se quiere programar para muchas plataformas y compiladores se debe tener esto en cuenta.
- Si se desea reutilizar el código también es importante tenerlo en cuenta.
- Usamos GCC con **`-std=C90 -pedantic -Wall -Eerror`** para conseguir la máxima portabilidad

Directiva `const`

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

Definicion de variables

- Diferencia entre estas definiciones
  - `int * const foo;`
    - Puntero a un entero constante
  - `int const * foo;`
    - Puntero constante a un entero
  - `const int * foo;`
    - Puntero constante a un entero
  - `int const * const foo;`
    - Puntero constante a un entero constante
  - `const int * const foo;`

Variables y constantes

- Las variables se convierten automaticamente a constantes
- El siguiente codigo 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);
```

Directiva `volatile`

- Se aplica a variables locales o globales
- Le indica al procesador que el contenido puede cambiar en cualquier momento sin intervención del código
- Por ejemplo se aplica en:
  - Registros de entrada/salida
  - Variables compartidas con interrupciones
  - Variables compartidas entre procesos
- Desactiva las optimizaciones de código para el acceso a esa variable

Directiva `static`

- Aplicado a variables locales las convierte en variables globales sin visibilidad para el resto del codigo
  - Conservan el valor entre llamadas
- Aplicadas a variables globales y procedimientos impide el acceso a los mismos desde fuera del codigo C
  - Impide el uso de **`extern`** en otro modulo

Opreadores a nivel de bits

- **`A & B`** Realiza una operación AND bit a bit entre de A y el de B
  - A = 0x0F & 0x13 ➔ A = 0x03
- **`A | B`** Realiza una operación OR bit a bit entre el valor de A y el de B
  - A = 0x18 |0x03 ➔ A = 0x1B
- **`A ^ B`** Realiza una operación OR EXCUSIVO bit a bit entre A y B
  - A = 0x18 |0x33 ➔ A = 0x2B

Opreadores a nivel de bits

- **~A** Realiza una inversión de los bit del valor de A (Complemento a 1)
  - A = ~0x2A ➔ A = 0xE4
- **A << B** Corre el B lugares a la izquierda valor de A
  - A = 0x0F << 2 ➔ A = 0x3C
- **A >> B** Corre el B lugares a la derechavalor de A
  - A = 0x18 >> 1 ➔ A = 0x0C

Como funciona el compilador

- En general se llama compilacion al proceso de traducción del codigo fuente al codigo binario que ejecuta el procesador.
- En el caso de C este proceso se desarrolla en tres fases:
  - Preprocesamiento
  - Compilacion
  - Enlazado

Preprocesamiento

- Normalmente toma un unico archivo c y entrega un archivo c 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 automaticamente antes de compilar

Compilación

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

Enlazado

- El proceso de enlazado toma todos los archivos con codigo objeto y las librerias y el ejecutable definitivo.
- El enlazador tiene como tareas
  - Resolver las referencias entre modulos
  - Asignar las zonas de memoria para cada segmento de codigo o de variables
  - Reubicar los bloques de codigo compilado
  - Generar la imagen final del ejecutable

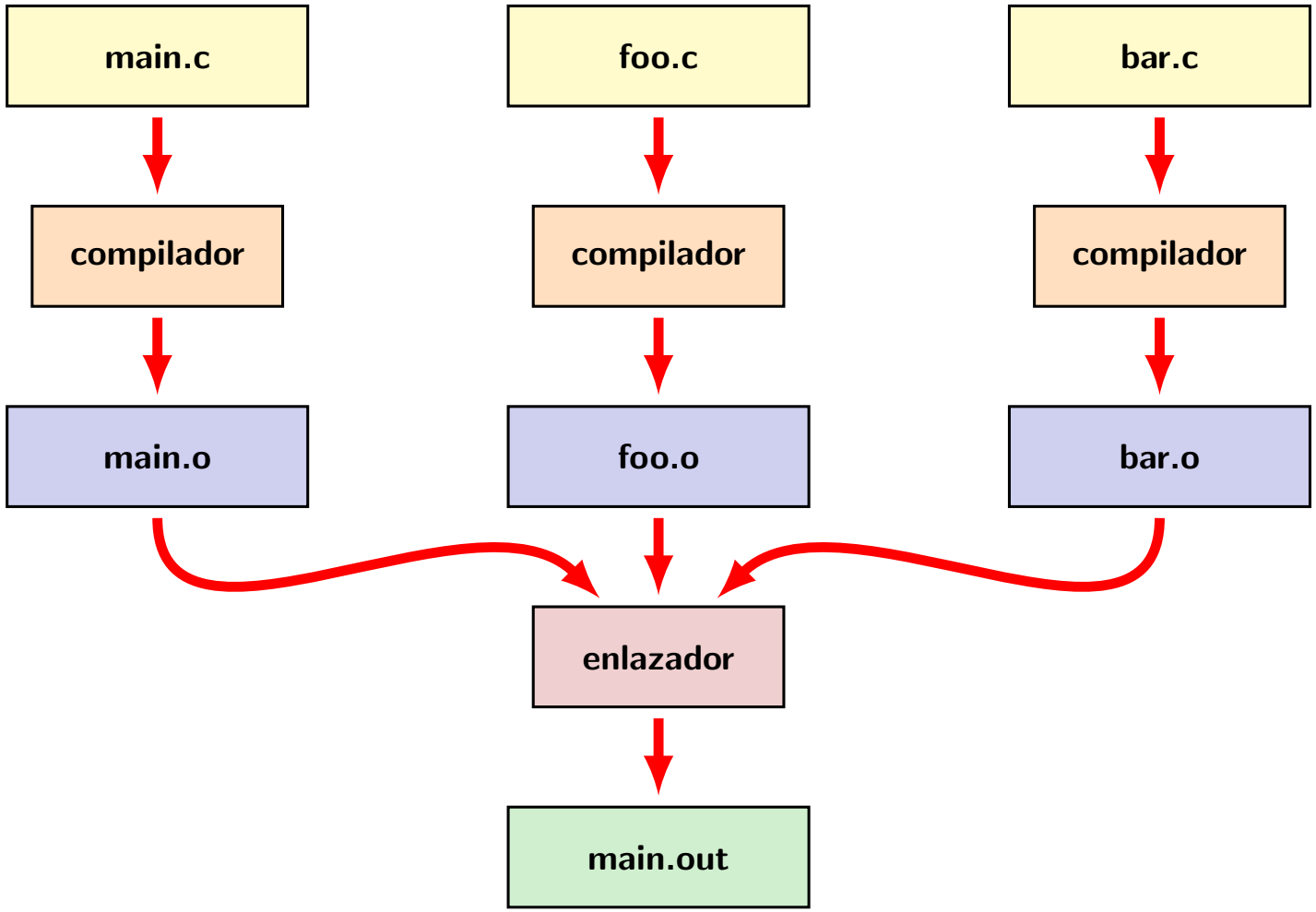
Compilando un proyecto

- Un proyecto esta formado normalmente por varios archivos de cabeceras, fuentes y librerias.
- Enm este caso se debe compilar cada archivo por separado y despues se enlazar los archivos de codigo objeto con las librerias.
- Para resolver este proceso existe una herramienta que automatiza el proceso de compilacion.

Make y Makefile

- Make es una herramienta destinada a automatizar el proceso de compilación de proyectos voluminosos.
- Forma parte de las herramientas del estandar POSIX.
- Uiliza un archivo llamado makefile que define una serie de reglas en funcion de las cuales make ejecuta las acciones necesarias.

Compilando 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 main.o`
- `gcc -c bar.c -o bar.o`
- `gcc bar.o main.o foo.o -o app`

Makefile para el proyecto

```
all: foo.o bar.o main.o
     gcc -o app 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 mas 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 $(OBJ_FILES) -o $(OBJ_DIR)/app.out

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

Coding guidelines

- Son un conjunto de convenciones sobre como escribir el codigo:
  - Aumentan la legibilidad
  - Disminuyen los errores
- Existen versiones gratuitas y veriones 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.

http://caxapa.ru/thumbs/468328/misra-c-2004.pdf

Pruebas

- Las pruebas se utilizan para detectar errores
- Existen dos clases de pruebas:
  - Pruebas estaticas (Verificación)
    - El codigo no es compilado ni ejecutado
    - Verifica el cumplimiento de los estandares
  - Pruebas dinamicas (Test)
    - El codigo es compilado y ejecutado
    - Verifica el cumplimiento de los requisitos

Pruebas estaticas

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

Testing

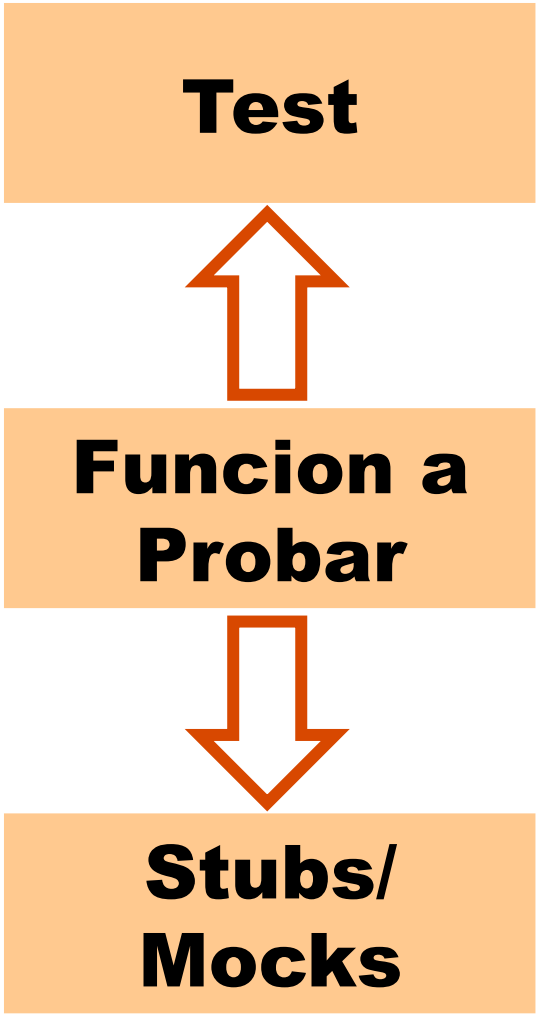
- Los tests son pruebas dinamicas
- Verifican el cumplimento de los requisitos
- Existen distintos niveles de testing
  - Pruebas unitarias
    - Verifica un unico archivo C o una clase
  - Pruebas de modulo
    - Verifica el conjunto de archivos que forma un modulo
  - Pruebas del sistema
    - Verfica el sistema completo

Test Driven Development

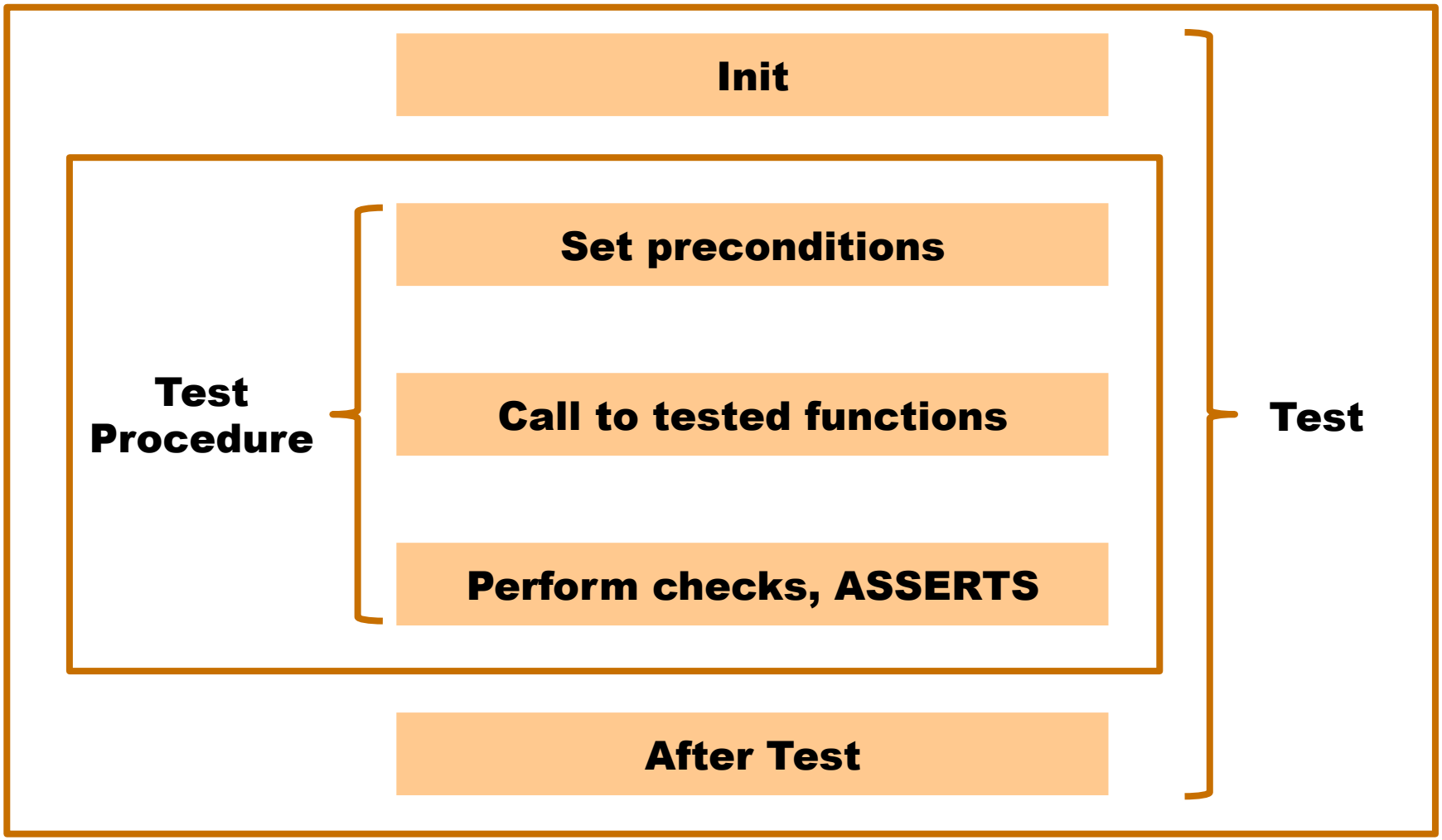
- Consiste en el diseño de los Tests al mismo tiempo que se definen los requisitos del sistema.
- El desarrollo del sistema se hace guiado por las pruebas que debe superar.
- Como cada prueba verifica el cumplimiento de un requisito,entonces el sistema esta completo cuando pasa todas las pruebas
- Cuando se hacen cambios se verifica que el sistema sigue cumpliento todos los requisitos, es decir que no hay una regresion.

Pruebas Unitarias

- Pruebas unicamente un archivo c
- Se prueban unicamente las funciones externas
- Todas las funciones utilizadas de otros archivos utilizadas por el codigo se reemplazan por funciones Stub o Mock



Estructura de los tests



Unity

- Es una librería para testing de código C a nivel de unidades y modulos.
- Permite escribir Test muy rapidamente y en una forma muy conceptual.
- Se expresan muy claramente los resultados esperados de una operación.

```
void test_calcularCRC(void) {
    char data[11] = "0123456789";
    TEST_ASSERT_EQUAL(0x443D, calcularCRC(data, 10));
}
```

CMock

- Es una librería para generar funciones de Stub o de Mock
- Emulan a las funciones externas al archivo que se esta probando
  - Las funciones stub que no hacen nada y solo permiten la compilacion del test
  - Las funciones mock son funciones verifican los parametros y devuelven resultados predefinidos

Funciones de Mock

```
int32_t open(...) {
   /* ... */
   void* ptr = malloc()
   sem_wait(&sem);
   /* ... */
   sem_post(&sem);
   /* ... */
}

void test_openMallocReturnNull(void) {
    malloc_ExpectAndReturn(NULL)
    TEST_ASSERT_TRUE(open(„/dev/serial“) == -1 )
}
```

Ceedling

- Es un framework que trabaja en conjunto con unity y cmock
- Automatiza las tareas de testing
  - Genera automaticamente los mock
  - Ejecuta los test
  - Recopila los resultados
- Automatiza tambien la creación del makefile

Coverage

- Un factor de calidad del software es el grado de cobertura de los test
- Existen distintas métricas
  - Function coverage
  - Statment coverage
  - Branch coverage
  - Condition coverage
- Cuanto mayor la cobertura de los test menor es la posibilidad de errores

Diagramas UML con Plantuml

- Permite generar la mayoria 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 facil de mantener y muy rapido dado que no hay que preocuparse por la parte grafica.

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
```

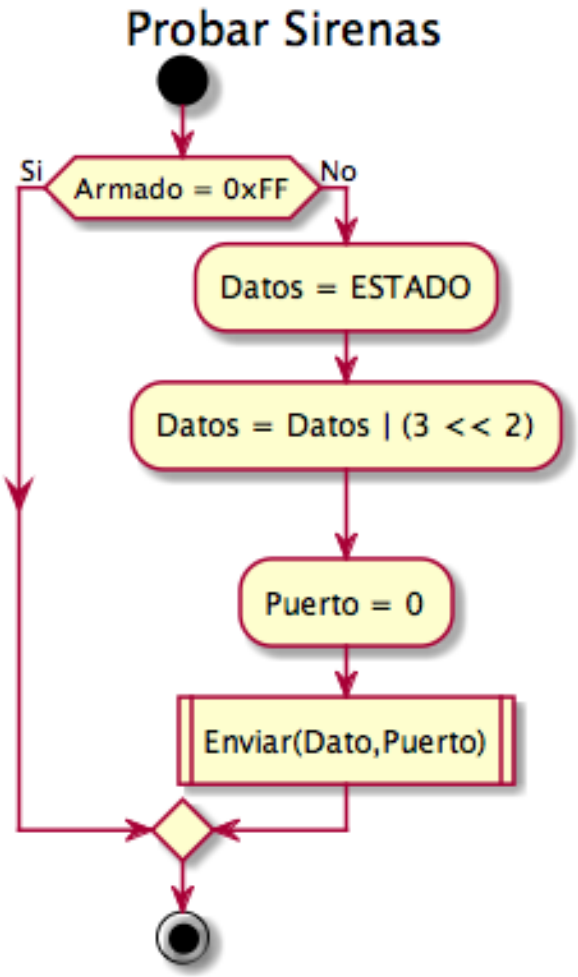


Diagrama de secuencia

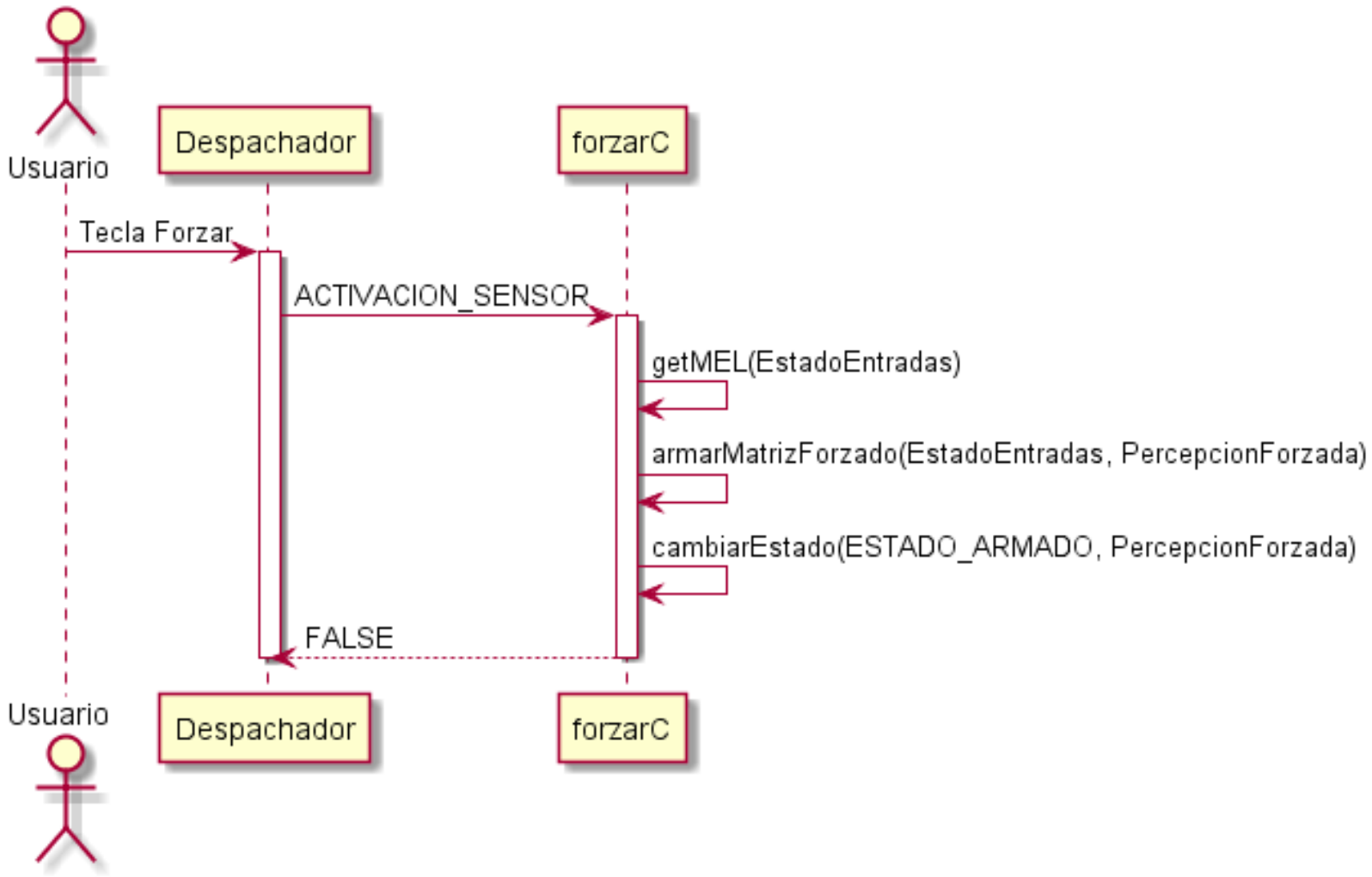
```
@startuml
actor Usuario
participant Despachador
participant forzarC

Usuario -> Despachador: Tecla Forzar
activate Despachador
Despachador -> forzarC: ACTIVACION_SENSOR
activate forzarC

forzarC -> forzarC: getMEL(EstadoEntradas)
forzarC -> forzarC: armarMatrizForzado(EstadoEntradas,
PercepcionForzada)
forzarC -> forzarC: cambiarEstado(ESTADO_ARMADO, PercepcionForzada)

forzarC --> Despachador: FALSE
deactivate Despachador
deactivate forzarC
@enduml
```

Diagrama de secuencia



Documentación con Doxygen

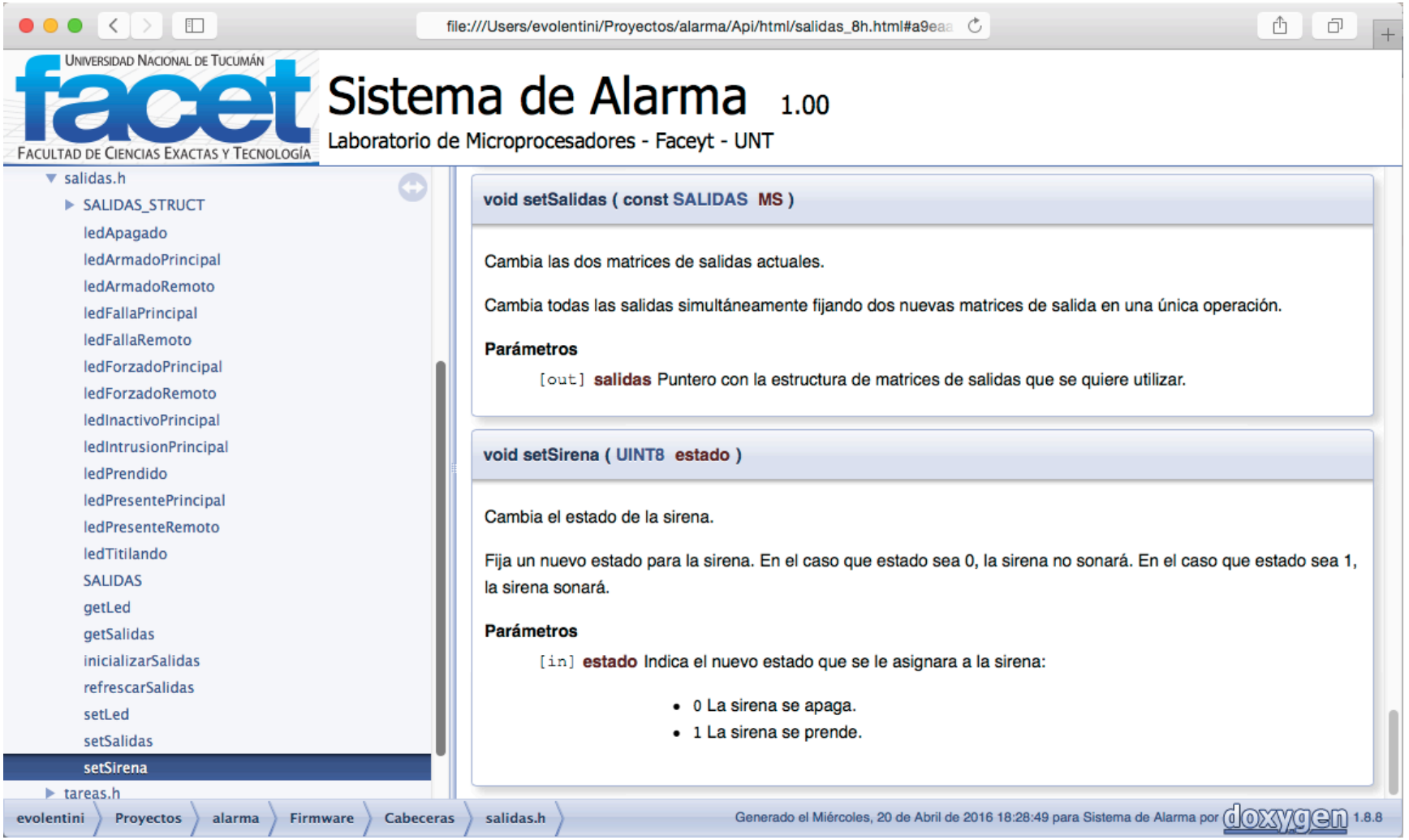
- Permite generar documentación de estructuras, tipos y funciones.
- Se genera a partir de comentarios especiales en el codigo fuente.
- La documentacion puede generarse en HTML, Latex, PDF, Man Pages, RTF y XML.
- Tambien puede documentase parte del diseño utilizandolo con plantuml.

Ejemplo de documentación

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

//! 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



Control de versiones

- Son herramientas diseñadas para gestionar los cambios en el proyecto.
- Los sistemas mas conocidos son CVS, Subversion, GIT, Bazaar y Mercurial.
- Cualquiera de ellos permite:
  - Almacenar los archivos del proyecto.
  - Almacenar los cambios en el proyecto.
  - Efectuar un seguimiento de los cambios efectuados en los archivos del proyecto.

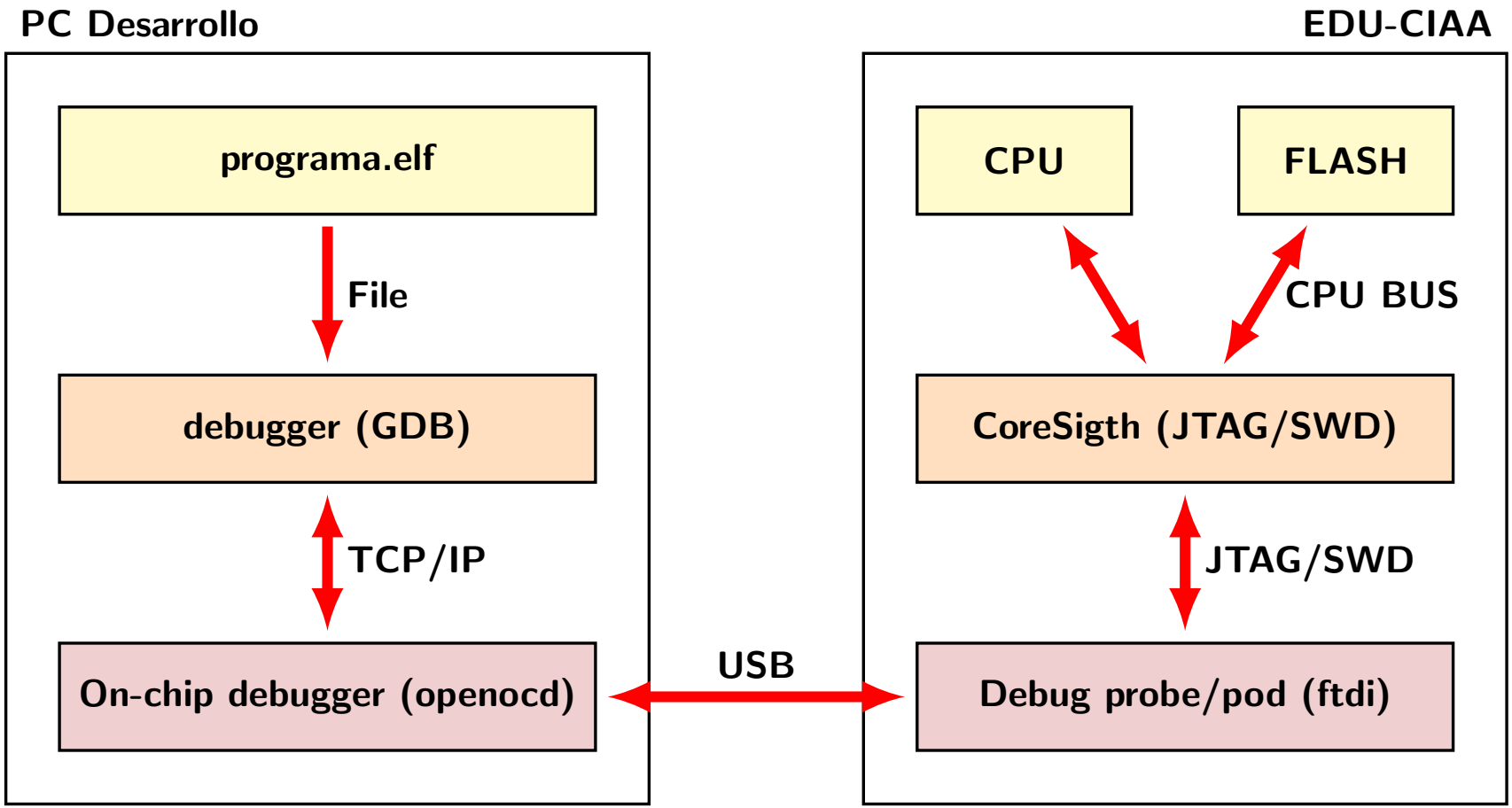
GIT

- Es una herramienta para control de versiones diseñada por Linus Torvald.
- Es un sistema distribuido donde cada programador tiene una copia completa del repositorio.
- Los cambios pueden intercambiarse directamente entre clientes usando http, ssh o rsync.
- El servidor no es mas que un cliente que esta siempre disponible.

Grabacion y depuración

- Cuando se compila un proyecto para un sistema embebido el resultado es una imagen 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



Componentes

- Imagen EFL: Imagen binaria del programa lista para ser ubicada en la memoria del procesador
- Debugger (GDB): Utilidad con capacidad para leer el archivo ELF, prepararlo para ejecutar y realizar la depuración del mismo:
  - Inspeccion de variables
  - Ejecución paso a paso

Componentes

- OpenOCD: Servidor para depuración remota que traduce los comandos de GDB en secuencias de comandos comprensibles por la interface de depuracion (POD)
- POD: Circuito electronico que recibe los comandos de OpenOCD y genera las señales electricas y tramas especificadas por la interface de depuracion del procesador (JTAG/SWD)