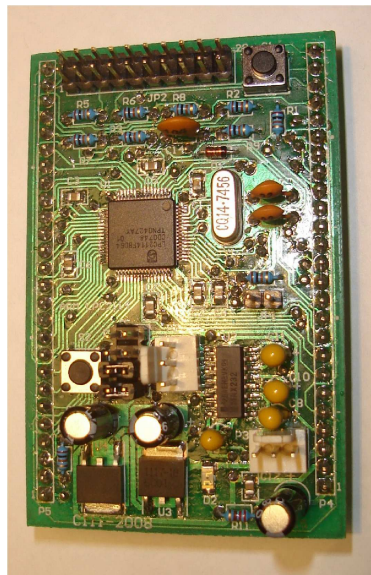


## Placa de Desarrollo 2009

Objetivo: Hardware para proveer una plataforma sencilla de desarrollo

Características de la placa

- Alimentación única de 5V
- Conversor de señales RS232 / TTL
- Conector JTAG
- Circuitos de soporte del LPC2114 (capacitores de desacople, cristal, reguladores de tensión 1,8 y 3, pulsador reset, pulsador de programación)



# Placa de Desarrollo 2010

Objetivo: Hardware para disminuir costo de desarrollo y aplicación final

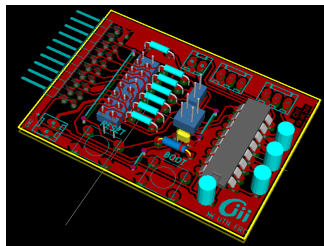
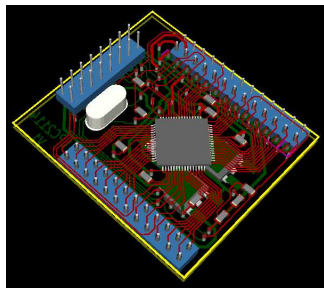
Consta de dos placas

- Placa ARM

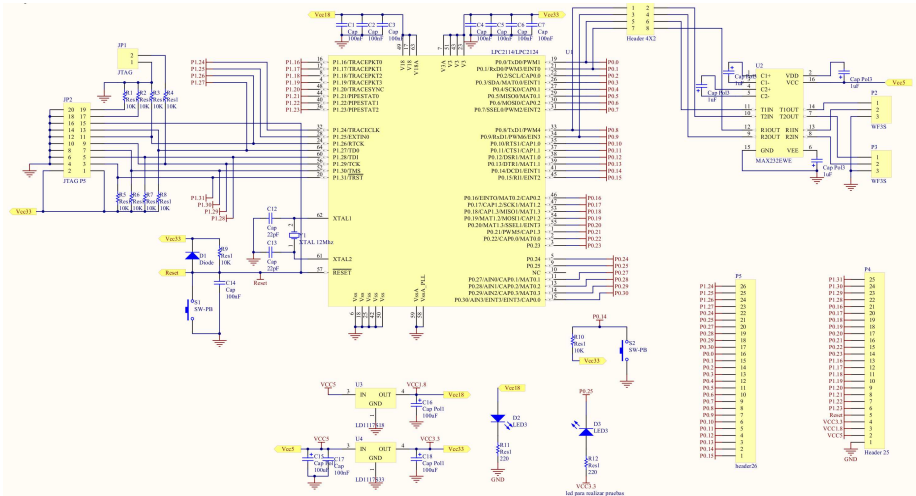
- ▶ Alimentación única de 5V
- ▶ Conector multipropósito para RS232 / JTAG
- ▶ Circuitos de soporte del LPC2114 (capacitores de desacople, cristal, reguladores 1,8 y 3)

- Grabadora

- ▶ Alimentación desde o hacia la placa ARM en 5V
- ▶ Conector RS232 y JTAG
- ▶ pulsador reset, pulsador de programación



# Esquemático



# Herramientas de Desarrollo

- Herramientas Libres GNU-ARM

**YAGARTO** IDE basado en GNU-ARM e interfase Eclipse, disponible solo para Windows <http://www.yagarto.de/>

**Tutoriales** Paginas con pequeños script o tutoriales para compilar el GNU-ARM

- ▶ OpenHardware

[http://openhardware.net/Embedded\\_ARM/Toolchain/](http://openhardware.net/Embedded_ARM/Toolchain/)

- ▶ rod.info <http://rod.info/ARM7Micro>

- ▶ gnu-arm-toolchain-installer

<http://mcuprogramming.com/forum/arm/gnu-arm-toolchain>

# Herramientas de Desarrollo

- Herramientas Pagas

Keil Compilador propio <http://www.keil.com/>

IAR Compilador propio <http://www.iar.com/>

CrossWorks Compilador GNU <http://www.rowley.co.uk/arm/index.htm>

Embest Compilador GNU <http://www.embedinfo.com>

Raisonance Compilador GNU con librerías optimizadas

<http://www.mcu-raisonance.com/>

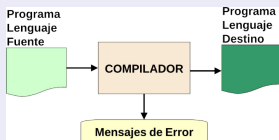
CodeSourcery Compilador GNU, versión LITE que permite usar el compilador GNU [http://www.codesourcery.com/abi\\_testsuite/](http://www.codesourcery.com/abi_testsuite/)

# Compilación

## Compilador

Un compilador es un programa que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación.

Se divide en Front-End, Middle-End y Back-End.



## Compilación Cruzada

Un compilador cruzado (cross-compiler) es un compilador capaz de generar ejecutables para otra plataforma diferente de aquella en la cual está corriendo.

## Tiempo de Compilación - Tiempo de Ejecución

Compile-time acciones relacionadas en la etapa de compilación.

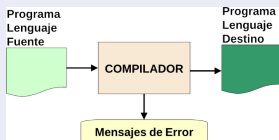
Run-time acciones al momento en que el programa esta corriendo.

# Compilación

## Compilador

Un compilador es un programa que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación.

Se divide en Front-End, Middle-End y Back-End.



## Compilación Cruzada

Un compilador cruzado (cross-compiler) es un compilador capaz de generar ejecutables para otra plataforma diferente de aquella en la cual está corriendo.

## Tiempo de Compilación - Tiempo de Ejecución

Compile-time acciones relacionadas en la etapa de compilación.

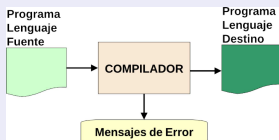
Run-time acciones al momento en que el programa esta corriendo.

# Compilación

## Compilador

Un compilador es un programa que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación.

Se divide en Front-End, Middle-End y Back-End.



## Compilación Cruzada

Un compilador cruzado (cross-compiler) es un compilador capaz de generar ejecutables para otra plataforma diferente de aquella en la cual está corriendo.

## Tiempo de Compilación - Tiempo de Ejecución

Compile-time acciones relacionadas en la etapa de compilación.

Run-time acciones al momento en que el programa esta corriendo.



# Proyecto GNU-ARM

## GNU toolchain

Es un término general para nombrar a una colección de herramientas de programación producidas por el proyecto GNU.

Estas herramientas permiten realizar todo el proceso de compilación enlazado y depuración de una aplicación.

## GNU ARM toolchain

Se refiere a la utilización de estas herramientas para generar código ARM.

# Proyecto GNU-ARM

## GNU toolchain

Es un término general para nombrar a una colección de herramientas de programación producidas por el proyecto GNU.

Estas herramientas permiten realizar todo el proceso de compilación enlazado y depuración de una aplicación.

## GNU ARM toolchain

Se refiere a la utilización de estas herramientas para generar código ARM.

# GNU-ARM

Provee las herramientas básicas para compilar un aplicación en ARM y realizar un debugging en este paquete dispondremos de

## GCC

Compilador GNU para C y C++

## GDB

Herramienta para realizar debugging, tanto local como remoto

## Insight

IDE para realizar debugging y simulación.

## NewLib

Implementación open source para sistemas embebidos de la biblioteca estándar de C

## Binutils

Serie de utilidades para enlazar o ensamblar un proyecto o programa y proveer información de códigos objetos.

# GNU-ARM

Provee las herramientas básicas para compilar un aplicación en ARM y realizar un debugging en este paquete dispondremos de

## GCC

Compilador GNU para C y C++

## GDB

Herramienta para realizar debugging, tanto local como remoto

## Insight

IDE para realizar debugging y simulación.

## NewLib

Implementación open source para sistemas embebidos de la biblioteca estándar de C

## Binutils

Serie de utilidades para enlazar o ensamblar un proyecto o programa y proveer información de códigos objetos.

# GNU-ARM

Provee las herramientas básicas para compilar un aplicación en ARM y realizar un debugging en este paquete dispondremos de

## GCC

Compilador GNU para C y C++

## GDB

Herramienta para realizar debugging, tanto local como remoto

## Insight

IDE para realizar debugging y simulación.

## NewLib

Implementación open source para sistemas embebidos de la biblioteca estándar de C

## Binutils

Serie de utilidades para enlazar o ensamblar un proyecto o programa y proveer información de códigos objetos.

# GNU-ARM

Provee las herramientas básicas para compilar un aplicación en ARM y realizar un debugging en este paquete dispondremos de

## GCC

Compilador GNU para C y C++

## GDB

Herramienta para realizar debugging, tanto local como remoto

## Insight

IDE para realizar debugging y simulación.

## NewLib

Implementación open source para sistemas embebidos de la biblioteca estándar de C

## Binutils

Serie de utilidades para enlazar o ensamblar un proyecto o programa y proveer información de códigos objetos.

# GNU-ARM

Provee las herramientas básicas para compilar un aplicación en ARM y realizar un debugging en este paquete dispondremos de

## GCC

Compilador GNU para C y C++

## GDB

Herramienta para realizar debugging, tanto local como remoto

## Insight

IDE para realizar debugging y simulación.

## NewLib

Implementación open source para sistemas embebidos de la biblioteca estándar de C

## Binutils

Serie de utilidades para enlazar o ensamblar un proyecto o programa y proveer información de códigos objetos.

# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

Dentro de la programación de un microcontrolador, se recurre constantemente a periféricos y/o características del hardware propias del modelo de microcontrolador usado, estos datos se resumen generalmente en direcciones de memoria o números de configuración, el uso de un archivo de cabecera sustituye estos números, por nemónicos mas fácil de recordar e independiente (en lo posible) al hardware usado.



# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

Es el encargado de describir al enlazador cómo las secciones de los archivos de entrada deben ser ubicada en el archivo de salida, y para configurar la distribución de memoria en este archivo.

## Otros Usos

Proveer a los archivos objetos de constantes con información de posiciones de los bloques, dar las herramientas para generar código reubicable, etc.

# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

Utilidad que permite automatizar los pasos para construir un binario o librería, partiendo de uno o varios programas destino.  
Se basa en dependencias.

# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

## Cabecera de la aplicación

Necesaria para:

- Establecer en la posición adecuada los vectores del microcontrolador
- configuración de hardware PLL
- copia de variables preinicializadas y código en RAM
- invocar el main() de la aplicación

# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

Series de programas que realizan la tarea para la cual tiene sentido el hardware

# Proyecto con GNU-ARM

Un proyecto con GNU-ARM de sistemas embebidos, está compuesto de las siguientes partes

- archivo de cabecera
- linker script
- Makefile
- head.s
- aplicación
- archivos de salida

Resultado del makefile, compuesto por:

- Archivo ELF (Executable and Linkable Format), archivo binario resultante del enlazado, posee una estructura estándar de archivo ejecutable o librería compartidas en UNIX.
- Archivo HEX, Imagen a grabar en el microcontrolador.
- Archivos de información, listado en assembler del código, tamaño a ocupar en la memoria, posiciones de variables en la memoria etc.

# MakeFile

```
# -----  
# Makefile for ex1.elf  
# -----  
LOADER = lpc21isp  
AS      = arm-elf-as  
CC      = arm-elf-gcc  
LD      = arm-elf-ld  
OBJCOPY = arm-elf-objcopy  
OBJDUMP = arm-elf-objdump  
  
AFLAGS = -mcpu=arm7tdmi -mapcs-32 --gstabs+  
CFLAGS = -Wall -O0 -mcpu=arm7tdmi -gstabs+  
LDFLAGS = -Tlpc2114_flash.ld -nostartfiles  
  
# =[ fuentes del programa ]=====   
SOURCES = head.s main.c led.c uart.c  
# =[ nombre del archivo de salida ]=====   
TARGET = led.hex
```

# MakeFile

```
OBJS1=$(SOURCES:.c=.o)
OBJS=$(OBJS1:.s=.o)
ELF=$(TARGET:.hex=.elf)
LST=$(TARGET:.hex=.lst)
MAP=$(TARGET:.hex=.map)

all: $(TARGET) $(LST)

depend.lst: $(SOURCES)
    $(CC) -MM $^ > depend.lst

#cargar dependencias
include depend.lst

#opcion de compilación para .c y .s
%.o:%.s
    $(AS) $(AFLAGS) $< -o $@
%.o:%.c
    $(CC) $(CFLAGS) -c $<
```

# MakeFile

```
$(TARGET): $(ELF)
    $(OBJCOPY) -O ihex $< $@

$(ELF): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) $^ -o $@

$(LST): $(ELF)
    $(OBJDUMP) -S $(ELF) > $(LST)

clean:
    rm *.o *.elf *.hex *.lst

grabar:
    $(LOADER) -wipe -hex $(TARGET) /dev/ttyUSB0 115200 14745
```



# LinkerScript

## Tipos de información en una imagen ROM

- STARTUP (Assembler)
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (const char cadena[] = "Hola Mundo", archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

# LinkerScript

## Tipos de información en una imagen ROM

- **STARTUP (Assembler)**
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (const char cadena[] = "Hola Mundo", archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

# LinkerScript

## Tipos de información en una imagen ROM

- STARTUP (Assembler)
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (`const char cadena[] = "Hola Mundo"`, archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

# LinkerScript

## Tipos de información en una imagen ROM

- STARTUP (Assembler)
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (const char cadena[] = "Hola Mundo", archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

# LinkerScript

## Tipos de información en una imagen ROM

- STARTUP (Assembler)
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (const char cadena[] = "Hola Mundo", archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

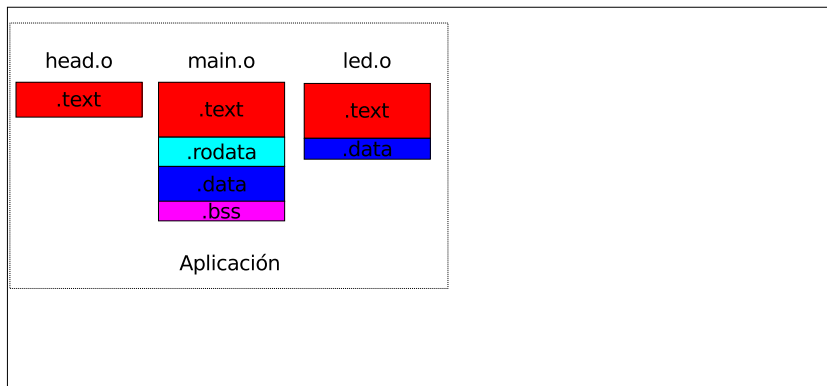
# LinkerScript

## Tipos de información en una imagen ROM

- STARTUP (Assembler)
  - ▶ Vectores de Interrupción
  - ▶ Configuración de Hardware
  - ▶ Inicialización de memoria y copias de rutinas a SRAM
- Código de la Aplicación (C,C++,etc) Aplicación que va a correr en el sistema.
- Constantes (`const char cadena[] = "Hola Mundo"`, archivos binarios, etc.)
- Variables Inicializadas
- Variables no Inicializadas

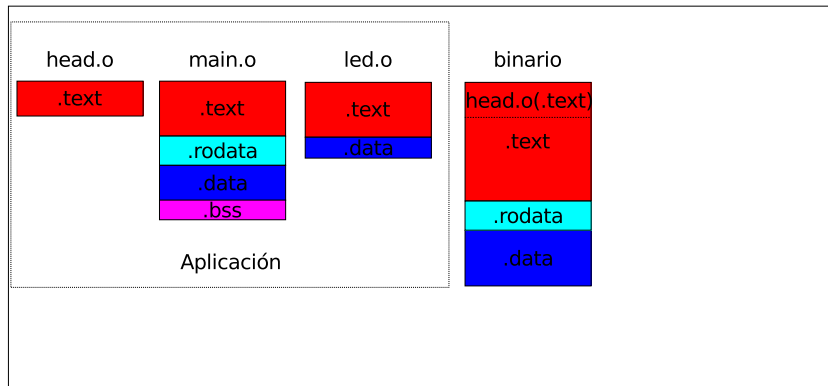
# LinkerScript (Código en Flash)

La aplicación.



# LinkerScript (Código en Flash)

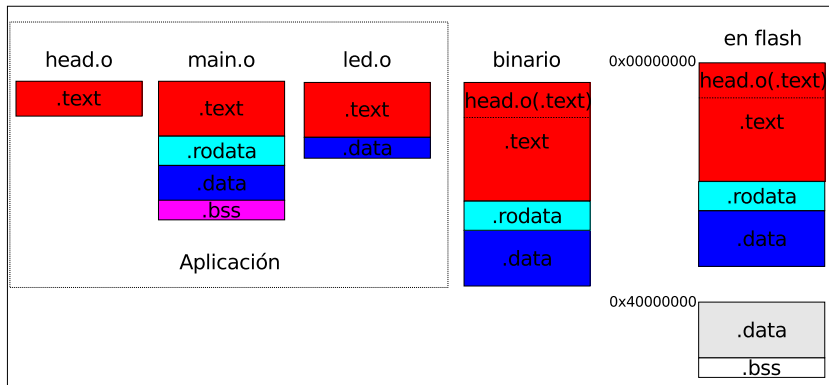
Archivo Enlazado.





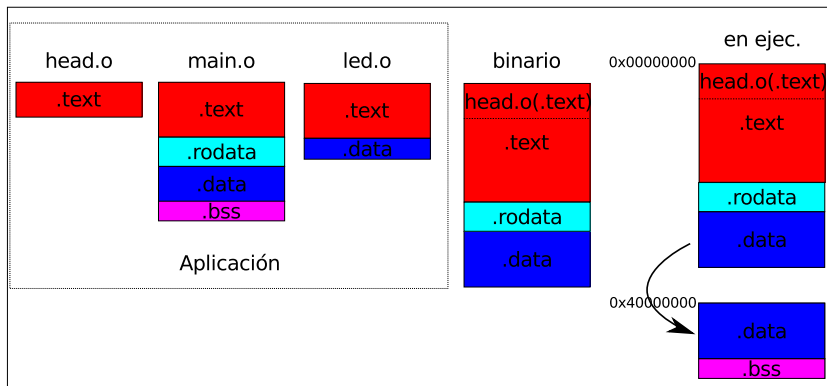
# LinkerScript (Código en Flash)

Archivo binario grabado al microcontrolador.



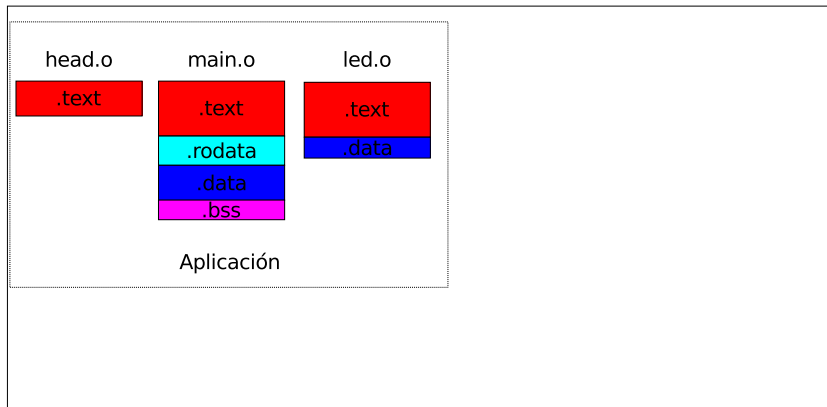
# LinkerScript (Código en Flash)

Aplicación en ejecución.



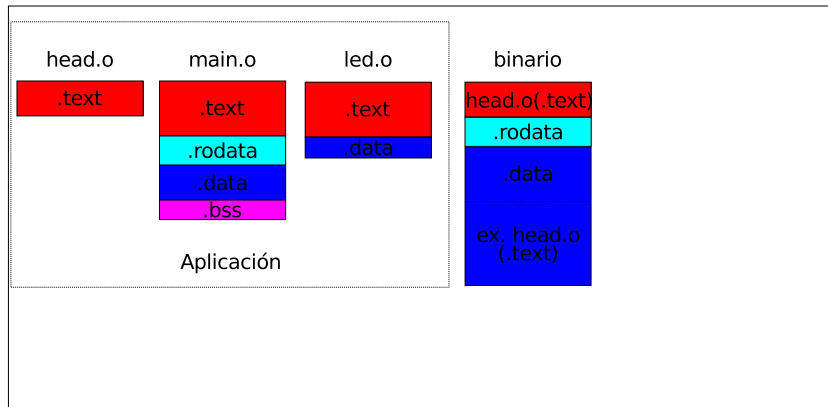
# LinkerScript (Código en SRAM)

La aplicación.



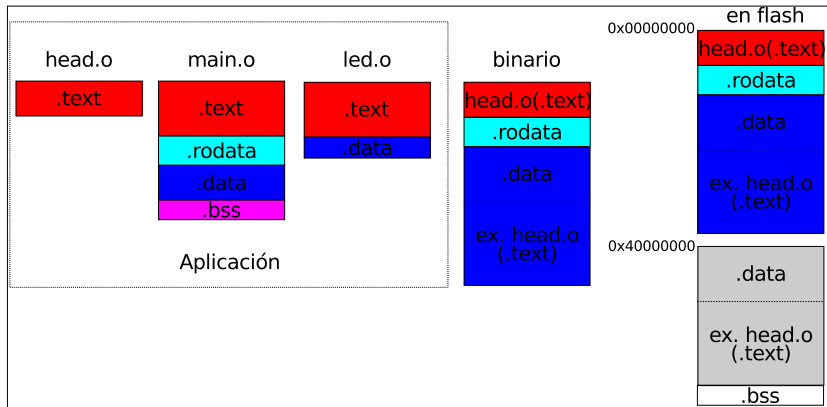
# LinkerScript (Código en SRAM)

Archivo Enlazado.



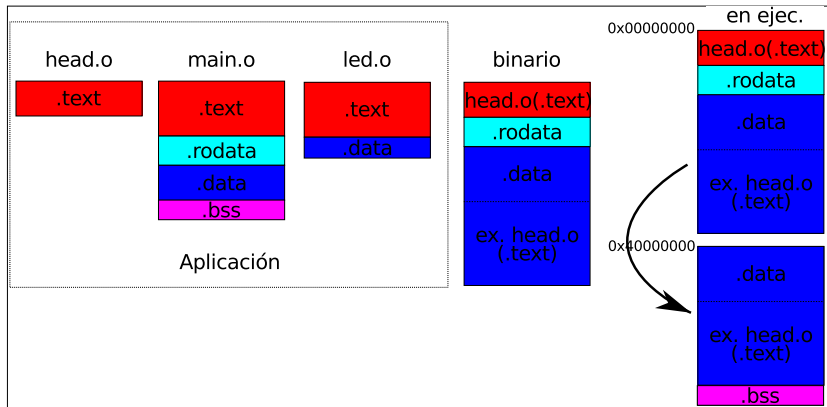
# LinkerScript (Código en SRAM)

Archivo binario grabado al microcontrolador.



# LinkerScript (Código en SRAM)

Aplicación en ejecución.



# LinkerScript

MEMORY

```
{
    flash (rx) : org = 0x00000000, len = 0x00020000
    sram  (rw) : org = 0x40000000, len = 0x00004000
}
```

SECTIONS

```
{
/* .text Sección de código ejecutable */
.text :
{
    *head.o (.text)
    *(.text)
} > flash
. = ALIGN(4);
/* .rodata Sección de variables de solo lectura */
.rodata :
{
    *(.rodata)
} > flash
. = ALIGN(4);
```

# LinkerScript

```
/* .data Sección de variables preinicializadas */
_etext = . ;
.data : AT (_etext)
{
    _data = . ;
    *(.data)
    _edata = . ;
} > sram
. = ALIGN(4);

/* .bss Sección de variables no inicializadas */
.bss :
{
    _bss = . ;
    *(.bss)
    _ebss = . ;
} > sram
. = ALIGN(4);
_end = .;

/* Stabs debugging sections. */
```



# head.s

## Tareas a realizar para aplicaciones en C

- Declarar los vectores
- Establecer el PLL
- Copiar la sección .data ( variables inicializadas ) a la SRAM
- Borrar .bss (variables no inicializadas)
- Establecer el stack pointer
- Saltar al main del C

## head.s Declarar Vectores

```
...
...
_start:
b reset /* reset */
b loop /* undefined instruction */
b loop /* software interrupt */
b loop /* prefetch abort */
b loop /* data abort */
nop /* reserved for the bootloader checksum */
ldr pc, [pc, #-0xFF0] /* VicVectAddr */
b loop /* FIQ */
...
...
```

## head.s Copiar la sección .data y borrar .bss

```
...
/* Copiar .data */
ldr r0, data_source @ indica la posición donde están guardados los valores
ldr r1, data_start  @ indica la pos.de la RAM donde comienza la zona de var
ldr r2, data_end    @ indica la pos.de la RAM donde finaliza la zona de var
copy_data:
cmp    r1, r2
ldrne r3, [r0], #4
strne r3, [r1], #4
bne    copy_data
/* Borrar el sector de variables no inicializadas .bss */
ldr r0, =0
ldr r1, bss_start   @ indica la pos.de la RAM donde comienza la zona de vari
ldr r2, bss_end     @ indica la pos.de la RAM donde finaliza la zona de vari
clear_bss:
cmp    r1, r2
strne r0, [r1], #4
bne    clear_bss
....
```

## head.s Saltar a main y valores tomados del linker

```
...
...
ldr    r10,=main
mov    lr,pc
bx     r10
/* si retorna del main entonces entrar en un bucle infinito*/
loop:  b    loop
...
...
/* Simbolos del linker script */
data_source:    .word    _etext
data_start:    .word    _data
data_end:      .word    _edata
bss_start:     .word    _bss
bss_end:       .word    _ebss
...
...
```