Herramientas GNU para programación de uC de arquitectura ARM

Guillermo Steiner[a], Gastón Araguás, Agustín Henze
[a]gsteiner@scdt.frc.utn.edu.ar

Centro de Investigación en Informática para la Ingeniería (CIII)
Universidad Tecnológica Nacional - Facultad Regional Córdoba

Resumen Crear un proyecto en microcontroladores supone siempre la utilización de herramientas provistas por el fabricante del microcontrolador o por terceros. Sin embargo, la mayoría de estas heramientas presentan varias desventajas frente a las herramientas provistas por el proyecto GNU. En este trabajo se describen las herramientas GNU involucradas en la creación de un proyecto de sistema embebido de arquitectura ARM, detallando los programas utilizados y los archivos involucrados para cada etapa, finalizando con una descripción del hardware para la implementación final.

GNU ARM toolchain lpc2114

1. Introducción

Crear un proyecto en microcontroladores supone siempre la utilización de herramientas provistas por el fabricante del microcontrolador o por terceros. Sin embargo, la mayoría de estas heramientas presentan varias desventajas frente a las herramientas provistas por el proyecto GNU. Algunas sólo aportan un IDE al usuario, pero utilizan para la compilación y enlace de las librerías los mismos recursos que las herramientas GNU (Makefile, linker script, etc) [2, 3, 1]. Este aporte es una ventaja ínfima e incluso a veces puede ser una desventaja, ya que llevar adelante un proyecto para microcontroladores utilizando un IDE, oculta al usuario el verdadero funcionamiento de las herramientas involucradas, perdiendo así la posibilidad de hacer ajustes particulares a cada proyecto en el proceso de creación. Otras de mejor calidad como Keil [4] o Iar [5] tiene licencias muy costosas que las hacen prácticamente inviables en proyectos de menor escala. En este punto también queda restringido el uso de estas herramientas en la Universidad, porque si bien se puede enseñar una herramienta comercial con licencia para uso educativos, los proyectos desarrollados en ese marco no podrán ser comercializados ni transferidos a la comunidad.

En este trabajo se describen en forma detallada las herramientas y archivos involucrados en la creación de un proyecto para un microcontrolador de arquitectura ARM. En la sección que sigue, se describen las herramientas GNU, en

la sección 3 se detallan los pasos necesarios para crear un proyecto y en la sección 4 se desarrolla un proyecto GNU desde cero. Por último en las sección 5 se muestran el hardware desarrollado para este proyecto y en la sección 6 las conclusiones.

Una descripción detallada de este trabajo se puede encontrar en `https://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/HerramientasProgramacion/`

2. GNUARM toolchain

GNU toolchain es un término general que se utiliza para referir a una colección de herramientas de programación producidas por el proyecto GNU. Estas herramientas permiten realizar todo el proceso de compilación enlazado y o depuración de una aplicación. GNUARM toolchain refiere específicamente al conjunto de herramientas GNU que forman parte de los procesos de compilación, depuración, etc. de aplicaciones en ARM. Estas herramientas están basadas en las GNU toolchains pero son de tipo "cross-development toolchains", es decir se desarrollan y compilan en una plataforma y pueden generar código para otra, en general son usadas en una plataforma x86 para generar código para una plataforma ARM.

La herramientas básicas que conforman el GNUARM toolchain son las siguientes:

- GCC: Compilador GNU para C y C++.
  (`http://gcc.gnu.org/`)
- NewLib: Implementación open source para sistemas embebidos de la biblioteca estándar de C.
  (`http://sourceware.org/newlib/`)
- GDB: Herramienta para realizar debugging, tanto local como remoto.
  (`http://www.gnu.org/software/gdb/`)
- Binutils: Utilidades para enlazar o ensamblar un proyecto o programa. Sus principales componentes son el compilador para lenguaje assembler (as) y el enlazador (ld), pero dispone también de otras herramientas como:
  - objcopy: permite cambiar de un formato a otro un archivo ya enlazado
  - objdump: provee información de archivos objetos, como su estructura, tamaño y posición de los diferentes bloques que los contituye, desensablado de código, etc.

  (`http://www.gnu.org/software/binutils/`)
- Insight: IDE para realizar debugging y simulación.
  (`http://sourceware.org/insight/`)

3. Un proyecto GNU ARM

En la sección anterior se listan las herramientas utilizadas en al creación de un proyecto. En esta sección se describe en forma completa un proyecto GNU ARM, comenzando por sus archivos principales.

De los archivos que conforman el proyecto algunos serán de configuración de hardware o configuración del proyecto y otros formarán la aplicación que correrá en el microcontrolador.

- *Archivo de cabecera:* 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 datos numéricos de configuración, el uso de un archivo de cabecera sustituye estos números por nemónicos mas fáciles de recordar e independientes (en lo posible) del hardware usado.
- *head.s:* Cabecera de la aplicación necesaria para:
  - Establecer en la posición adecuada los vectores de interrupción del microcontrolador.
  - Configurar el PLL. (hardware)
  - Copiar variables preinicializadas y código en RAM.
  - Genera los espacios necesarios en memoria y configura los stack.
  - Invocar la función main() de la aplicación. (para proyectos escritos en C)
- *linker script:* Contiene las directivas necesarias para el enlazador, indicando a este cómo ubicar los archivos de entrada en el archivo de salida y cómo configurar la distribución de memoria en este archivo, Estos bloques de datos pueden ser separados de la siguiente manera:
  - *Código.* Contiene el bloque ejecutable de la aplicación, consta de dos partes
    - STARTUP: Codificado generalmente en el archivo *head.*s descripto anteriormente, realiza todas las funciones asignadas a él. Su principal característica es que debe ser ubicado en una posición determinada y fija en la memoria del microcontrolador, esto es debido a que en su código se encuentran los vectores de interrupción.
    - Código de la Aplicación (C,C++,etc): aquí se agrupa todo lo relacionado con la aplicación (funciones, funciones de interrupción, etc,), puede encontrarse en un solo bloque compacto o dividida en diferentes partes de la memoria del micro (incluso en la memoria ram).
  - *Constantes (const char cadena[] = "Hola Mundo", archivos binarios, etc.).*
  - *Variables globales Inicializadas.*
  - *Variables globales no Inicializadas.*

  Para mas detalles ver (`http://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/HerramientasProgramacion/LinkerScript`)
- *Makefile:* Archivo de configuración para la herramienta *make*, automatiza la construcción de archivos binarios o librerías, partiendo de uno o varios programas fuente y librerías externas .
- *Archivos de la aplicación*: Son todos los archivos referidos propiamente a la aplicación, cuando la misma es en C, se dispondrá de un archivo fuente con la función main() llamado *main.c* o con algun nombre identificatorio y achivos adicionales con subrutinas, funciones y manejos de interrupciones.

- *Librerías*: En aplicaciónes C se dispone de un set de librerías provistas por el proyecto *NewLibs,* éstas y cualquier otra libería externa al proyecto deberán ser incluidas en las directivas del enlazador dentro del *Makefile.*
- *Archivos de salida:* Construidos según el *Makefile*
  - *Archivo ELF (Executable and Linkable Format).* Archivo binario resultante del enlazado, posee una estructura estándar de archivo ejecutable o librería compartida de UNIX (`http://en.wikipedia.org/wiki/Executable_and_Linkable_Format`).
  - *Archivo HEX.* Imagen a grabar en el microcontrolador.
  - *Archivos varios de información.* Archivos conteniendo información variada cómo listado en assembler del código, tamaño a ocupar en la memoria, posiciones de variables en la memoria, etc.

En la Figura 1 se puede ver un diagrama de cómo se conforma un proyecto GNU ARM. En el mismo se detalla un ejemplo de una aplicación en C compuesta por dos archivos fuentes, main.c y func.c, con sus correspondientes archivos de cabecera. El archivo *Makefile* contiene las dependencias y reglas que se muestran en el diagrama para construir el proyecto con *make* (véase la sección 4.4).

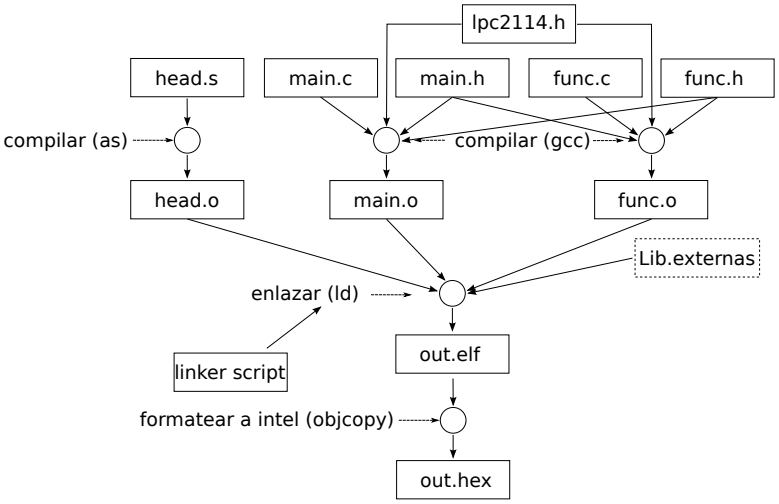

Figura 1. Proyecto GNU ARM

4. Implementación del proyecto

Como se describió anteriormente un proyecto está constituido por un conjunto de archivos, algunos muy similares a los que se encuentran en un proyecto en lenguaje C de una computadora personal, otros específicos de un proyecto

que correrá en un sistema embebido. A continuación se detallan los pasos para la implementación de un proyecto basado en un microcontrolador lpc2114 de la firma NXP con núcleo ARM7TDMI, configurando cada uno de los archivos enumerados en la sección anterior.

4.1. Archivo de cabecera *lpc2114.h*

Es un archivo estándar para cualquier proyecto no depende del proyecto en sí sino del microcontrolador usado, por eso generalmente lo provee el fabricante o se encuentra en la web con nombre igual al microcontrolador y extensión *.h* (en este caso *lpc2114.h*). No requiere configuración especial.

4.2. Archivo *head.s*

Es un archivo escrito en lenguaje assembler que realiza una serie de configuraciones generales a nivel hardware. Un típico archivo de este tipo se detalla a continuación.

En primer lugar se encuentran los siete vectores de interrupción (véase Algoritmo 1). Estos vectores se encuentran a partir de la posición 0x00 del mapa de memoria del microcontrolador, esto significa que su ubicación es fija dentro de la memoria del ARM.

El primer vector implementado es el de *reset*, esto implica que cuando el microcontrolador sale de una condición de reset salta a la posición mas baja de su mapa (0x00). En ese lugar se encuentra con otra instrucción de salto la cual permite saltar todos los vectores a la posición inmediata posterior a ellos, comenzando en ese lugar la inicialización del hardware. El vector número seis es el vector de IRQ, el cual recibe un tratamiento especial. En este caso en lugar de un salto se realiza directamente la actualización del Program Counter (PC) con el valor del registro VicVectAddr, este registro contiene la dirección de la función que atenderá esta interrupción. Este mecanismo de direccionamiento indirecto le permite al microprocesador modificar el valor de este registro dependiendo del hardware que generó la interrupción.

Luego de los vectores de interrupción se encuentran la inicialización del hardware (véase Algoritmo 2). Esta inicialización es relativamente independiente al proyecto a desarrollar y consiste en:

- Configurar el PLL (oscilador interno de mayor frecuencia que se engancha con el oscilador externo a cristal)
- Copiar desde la memoria flash las variables globales preinicializadas a sus posiciones respectivas en la RAM
- Borrar las variables globales no inicializadas y
- Destinar un espacio a cada uno de los stacks del microcontrolador (stack general, de IRQ y FIQ)

Hay que notar que la información de donde se encuentra las variables no inicializadas, las variables preinicializadas y de estas ultimas el lugar de los valores de

Algoritmo 1 Listado *head.s*, primera parte

```
.....
_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, #-0x0FF0]/* VicVectAddr */
b loop                  /* FIQ */
```

inicio, la extrae el enlazador, por medios de directivas en el linker script y transferidas a este archivo (head.s), de esa forma y por medio de constantes como data_start, data_end, etc. se pueden utilizar esta información para las rutinas de copia e inicialización.

Una vez finalizada la inicialización, se llama a la función main(), a partir de aquí toma el control la aplicación.

4.3. Archivo linker script *lpc2114_flash.ld*

Este es un archivo fuertemente dependiente del hardware y es provisto en general por el fabricante. A diferencia del archivo de cabecera, el linker script puede ser modificado de acuerdo a los requerimientos del proyecto. Típicamente un liker script consiste en dos partes principales, MEMORY y SECTIONS. Las especificaciones básica del hardware viene dada en la primeras directivas de MEMORY: posición y tamaño de la memoria flash, y posición y tamaño de la memoria RAM. En SECTIONS se determinan las diferentes secciones del proyecto y su ubicación dentro de la memoria RAM y flash (Algoritmo 3).

4.4. *Makefile*

El uso de la utilidad *make* (`http://www.gnu.org/software/make/`) a traves de su *Makefile* en la GNUARM toolchain es del modo clásico, a modo de ejemplo se detalla el siguiente *Makefile* (Algoritmo 4).

En primer lugar se declaran las herramientas a usar (compilador, enlazador, etc.) y las banderas necesarias de esas herramientas. Como se ve mediante la bandera -T se pasa al enlazador el nombre del archivo linker script (*lpc2114_flash.ld*) detallado antes. Luego se construyen todas las dependencias en base a los fuentes del programa y finalmente se graba el binario al microcontrolador usando la herramienta *GNU lpc21isp* (`http://lpc21isp.sourceforge.net/`).

4.5. Aplicación *main.c*

El archivo *main.c* contiene la función main() que implementa la aplicación propiamente dicha, este y todos los archivos que conformen la aplicación son los fuentes que construyen el binario y su manejo es el clásico de una aplicación en C.

5. Placa de desarrollo

Desarrollar en microcontroladores de 32 bits implica en la mayoría de los casos utilizar placas desarrolladas por terceros, ya que utilizar placas de desarrollos simples o los viejos protoboard se hace evidentemente imposible por el tamaño de los encapsulados de estos micros y por la complejidad de su conexión con los elementos de soporte para su funcionamiento (interfaz de programación, condensadores de desacople, cristal, fuentes de alimentación, etc).

La propuesta en este punto es implementar una placa mínima, con solo los recursos necesarios para el funcionamiento del micro controlador (Figura 2) Este hardware mńimo puede ser utilizado en diferentes aplicaciones como para hacer pruebas rapidas debido a su sencillez, o como placa de soporte para realizar enzayos mas avanzados o como hardware final, luego de haber completado la etapa de desarrollo, ya que cuenta con los elementos mńimos necesarios para funcionar y su costo es muy bajo. Las partes componentes de la placa son:

- Micro controlador LPC2114
- Cristal
- Alimentación unica de $5V$ con dos reguladores lineales de $3,3V$ y $1,8V$ para alimentación del microcontrolador
- Condensadores de desacoples recomendados por el fabricante
- Disponibilidad por medio de dos conectores paralelos del pin-out de microcontrolador.
- Pequeño conector para interfaz con grabador y debug por JTAG (`http://en.wikipedia.org/wiki/Jtag`)

Luego una segunda placa denominada placa de soporte, implementa la interfaz con JTAG y el adaptador de señal para grabar utilizando el puerto serie de la PC (Figura 3). Esta placa de soporte contiene todos los elementos necesarios en la etapa de grabación y debug, con lo que se logra que la placa mínima no contenga componentes que no hacen al funcionamiento especifico del microcontrolador.

6. Conclusiones

En este trabajo se desarrollaron las diferentes etapas que conforman un proyecto GNU para arquitectura ARM, describiendo aspectos que muchas veces estan ocultos en herramientas pagas como *linker script*, *head.s*, etc, permitiendo de esta forma actuar en un ajuste mas personal del mismo.

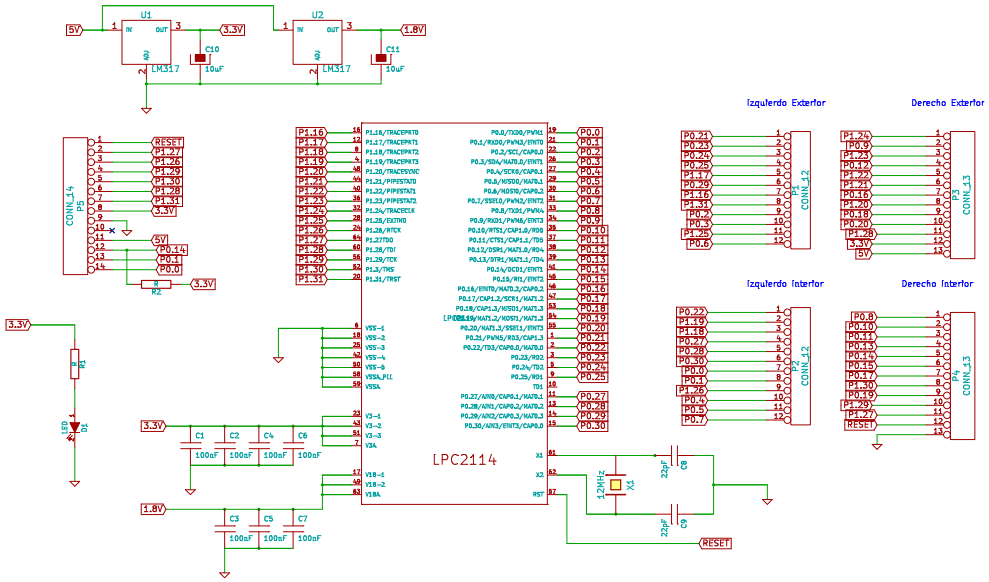

Figura 2. Esquemático Placa

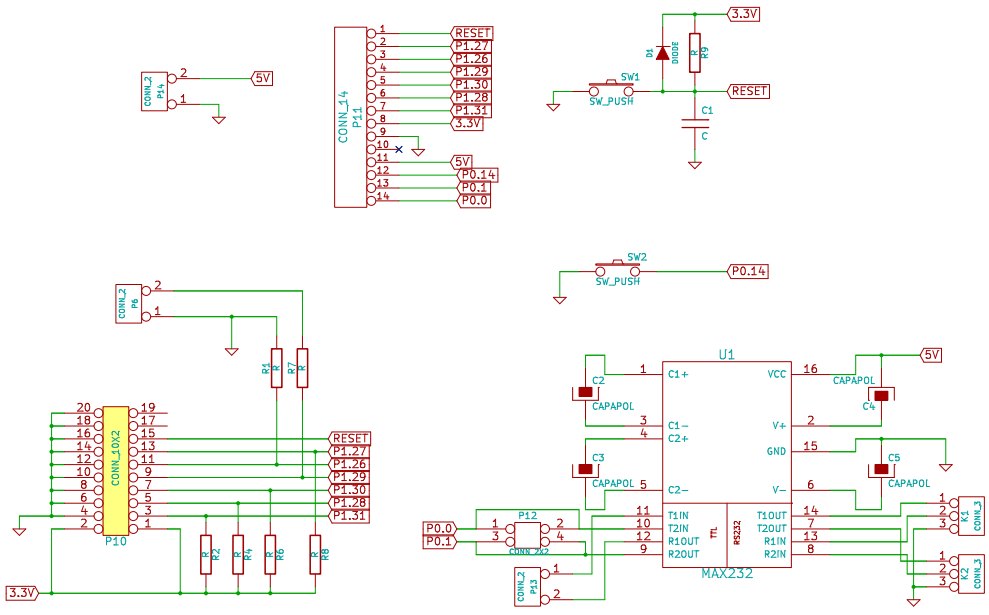

Figura 3. Esquemático Placa Soporte

Detalles del trabajo presentado, planos de hardware y herramientas disponibles se puede encontrar en (`https://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/HerramientasProgramacion`), y ejemplos de implementaciones usando estas herramientas en (`https://ciii.frc.utn.edu.ar/TecnicasDigitalesII/WebHome/TrabajosPracticos/PracticoASM5`).

Algoritmo 2 Listado *head.s* - Segunda parte

```
reset:
/* Establecer PLL------------------------------*/
ldr r0, PLLBASE
mov r3, #PLLCFG_VALUE
str r3, [r0, #PLLCFG_OFFSET]
mov r3, #PLLCON_PLLE
str r3, [r0, #PLLCON_OFFSET]
mov r1, #PLLFEED1
mov r2, #PLLFEED2
str r1, [r0, #PLLFEED_OFFSET]
str r2, [r0, #PLLFEED_OFFSET]
pll_loop:
ldr r3, [r0, #PLLSTAT_OFFSET]
tst r3, #PLLSTAT_PLOCK
beq pll_loop
mov r3, #PLLCON_PLLC|PLLCON_PLLE
str r3, [r0, #PLLCON_OFFSET]
str r1, [r0, #PLLFEED_OFFSET]
str r2, [r0, #PLLFEED_OFFSET]
*/ Establecer Stack ------------------------------*/
stacks_init:
ldr r0, STACK_START
/* stack de FIQ */
msr CPSR_c, #FIQ_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp,r0
sub r0, r0, #FIQ_STACK_SIZE
/* stack IRQ */
msr CPSR_c, #IRQ_MODE|IRQ_DISABLE|FIQ_DISABLE
mov sp, r0
sub r0, r0, #IRQ_STACK_SIZE
msr CPSR_c, #SYS_MODE
mov sp,r0
/* Copiar .data*/
ldr r0, data_source
ldr r1, data_start
ldr r2, data_end
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
ldr r2, bss_end
clear_bss:
cmp r1, r2
strne r0, [r1], #4
bne clear_bss
seguir:
/* finalizo la inicialización de hardware ahora se llama a main() */
bl main
loop: b loop
/* Constantes (calculo de stack, si la SRAM del LPC */
/* comienza en 0x40000000, y mide 16Kb = 4000h) */
stack_addr: .word 0x40004000
PLLBASE: .word 0xE01FC080
MAMBASE: .word 0xE01FC000
STACK_START: .word 0x40004000
*
las constantes
.......
.......
.......
*/
```

Algoritmo 3 linker script *lpc2114_flash.ld*

```
MEMORY
{
flash (rx) : org = 0x00000000, len = 0x00020000
sram (rw) : org = 0x40000000, len = 0x00004000
}
SECTIONS
{
/* ---------------------------------------------------------------------
* .text section (executable code)
* ---------------------------------------------------------------------
*/
.text :
{
*head.o (.text)
*(.text)
*(.glue_7t) *(.glue_7)
} > flash
. = ALIGN(4);
/* ---------------------------------------------------------------------
* .rodata section (read-only (const) initialized variables)
* ---------------------------------------------------------------------
*/
.rodata :
{
*(.rodata)
} > flash
. = ALIGN(4);
/* End-of-text symbols */
_etext = . ;
PROVIDE (etext = .);
/* ---------------------------------------------------------------------
* .data section (read/write initialized variables)
* ---------------------------------------------------------------------
*/
.data : AT (_etext)
{
_data = . ;
*(.data)
_edata = . ;
PROVIDE (edata = .);
} > sram
. = ALIGN(4);
/* ---------------------------------------------------------------------
* .bss section (uninitialized variables)
* ---------------------------------------------------------------------
*/
.bss :
{
__bss_start__ = .;
_bss = . ;
*(.bss)
*(COMMON)
_ebss = . ;
} > sram
. = ALIGN(4);
_end = .;
__bss_end__ = .;
PROVIDE (end = .);
.....
```

Algoritmo 4 *Makefile* del proyecto.

```
# ------------------------------------------------------------------------
# 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
# nombre del archivo de salida
TARGET = led_irq.hex
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
#opción de compilación para .c
%o:%.s
$(AS) $(AFLAGS) $< -o $@
%o:%.c
$(CC) $(CFLAGS) -c $<
$(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
```

Bibliografía

[1] Rowley Associates. Professional tools for arm developers. URL http://www.rowley.co.uk/arm/index.htm.

[2] Embest Info&Tech Co. Embest ide for arm. URL http://www.embedinfo.com/English/Product/idemain.asp.

[3] ImageCraft Creations Inc. Arm development tools. URL http://www.imagecraft.com/devtools_ARM.html.

[4] KEIL. Mdk-arm microcontroller development kit. URL http://www.keil.com/arm.

[5] IAR System. Iar embedded workbench® for arm. URL http://www.iar.com/ewarm.