Práctica 4

Arranque e inicialización del entorno de ejecución

Introducción

En esta práctica introduciremos la estructura del BSP que iremos desarrollando a lo largo de la asignatura. Una vez entendidos su funcionamiento y organización, nos centraremos en el desarrollo del cargador de arranque para el BSP, que una vez finalizado nos permitirá ejecutar código *C* en nuestra plataforma. Para ello será necesario entender el proceso de arranque de la placa, que está descrito en su manual de referencia.

Objetivos

- Conocer la estructura y organización del BSP.
- Conocer el proceso de arranque de la placa.
- Diseñar un cargador de arranque para el BSP que se adapte a las características de nuestra placa de desarrollo.
- Diseñar un *script* de enlazado para el BSP que permita ejecutar aplicaciones en *C* en la placa de desarrollo.
- Construir el BSP.
- Hacer uso del BSP para generar una imagen de *firmware* ejecutable en la placa de desarrollo a partir de una aplicación en *C*.
- Saber borrar la memoria flash de la placa y grabar la imagen del *firmware* en ella.

Descripción del BSP

En `/fenix/depar/atc/se/pracs/bsp` podemos encontrar los ficheros que componen el BSP que iremos desarrollando a lo largo de la asignatura. Si lo construimos podremos observar que el resultado de la construcción es un fichero de biblioteca llamado `libbsp.a` que contiene todo el código del BSP preparado para que las aplicaciones puedan enlazarse con las funciones que necesiten.

Desde el punto de vista del desarrollador de aplicaciones, el BSP se ve como una biblioteca (`libbsp.a` en nuestro caso) con la que hay que enlazar la aplicación para que pueda ejecutarse en la placa, junto con un *script* de enlazado que indica cómo se debe mapear la aplicación en la memoria de la placa, y un conjunto de ficheros cabecera que muestran el API de los diferentes servicios proporcionados por el BSP. Toda esta información se encuentra recogida en el fichero `bsp.mk`. Si lo editamos podremos comprobar cómo se definen variables que indicarán al `Makefile` de la aplicación cómo se llama el fichero de biblioteca que contiene el BSP, qué fichero se debe usar como *script* de enlazado, la lista de directorios que contienen las cabeceras de las funciones implementadas en el BSP, y cómo enlazar la aplicación con el BSP y el resto de bibliotecas estándar para lograr generar un ejecutable.

Por otro lado, desde el punto de vista del desarrollo del BSP, podemos observar que está estructurado en diferentes carpetas, un `Makefile` y un *script* de enlazado. Realmente el `Makefile` no hace uso del *script* de enlazado. El `Makefile` simplemente compila todos los ficheros *C* o ensamblador de todos los directorios del BSP y los incluye en una biblioteca llamada `libbsp.a`. Sin embargo, es necesario incluir un *script* de enlazado en el BSP para indicar al enlazador cómo se debe generar una imagen ejecutable a partir del código de una aplicación que se enlace con nuestro BSP.

Centrándonos en las carpetas del BSP, podemos observar que hay una dedicada a los *drivers*, otra para capa de abstracción hardware y otra para utilidades. Se pueden añadir todas las carpetas que se necesiten, ya que el `Makefile` las procesará adecuadamente siempre que dentro tengan código *C* o ensamblador.

En esta práctica nos centraremos sólo en el *script* de enlazado y en el fichero `hal/crt0.s`, que son los únicos ficheros del BSP necesarios para implementar arranque de la placa y la ejecución de una aplicación en *C*. El resto de las funcionalidades del BSP se irán cubriendo en prácticas sucesivas.

Proceso de arranque de la placa

Aunque la secuencia de pasos necesarios para arrancar una plataforma y establecer un entorno de ejecución *C* es siempre la misma, la forma de llevarla a cabo dependerá de cada plataforma en concreto, ya que, por un lado los periféricos y controladores serán diferentes, y por otro, cada placa puede llevar una ROM pre-configurada de fábrica con una serie de funciones ya implementadas que simplifiquen el proceso de arranque. Por tanto, lo primero que hay que hacer es buscar en la documentación el proceso de arranque de cada placa concreta.

En el caso de la *Redwire Econotag*, la Sección 3.12, junto con el Apéndice C, detallan el proceso de arranque diseñado por *Freescale* para esta placa. El primer paso que debemos seguir para implementar el arranque de esta placa es leer bien la documentación para ver qué tareas se hacen por el *Bootloader* grabado en la ROM de la placa, y qué tareas debemos implementar nosotros para completar el arranque y el entorno de ejecución de C.

A grandes rasgos podemos sacar la siguiente conclusiones:

- La placa tiene una memoria ROM mapeada a la dirección 0x0000 0000 en la que hay instalado un *bootolader.*
- El *bootloader* identifica el origen de la imagen del *firmware,* buscando en la memoria Flash, en la UART, en el puerto SPI o en el puerto I2C por este orden.
- Si se encuentra una imagen válida, se limpia toda la RAM (se inicializa a cero) y se copia la imagen al principio de la RAM (dirección 0x0040 0000) y se salta al principio de la RAM (dirección 0x0040 0000).
- Los cuatro primeros bytes de la imagen deben usarse para indicar su tamaño, ya que el *bootloader* de la ROM leerá el contenido de estos cuatro bytes para saber cuántos bytes tiene que copiar desde el origen de la imagen hacia la RAM.
- Como la tabla de vectores de excepción debe residir en las direcciones 0x0000 0000 – 0x0000 000f según el ABI de la arquitectura *ARM,* y estas direcciones están mapeadas a una ROM, no se podrán modificar para definir manejadores a la medida del BSP o de la aplicación. Para solucionar este problema, en vez de usar un controlador de memoria que permita un remapeo de la RAM a la dirección 0x0000 0000 como en la mayoría de los

microcontroladores basados en un procesador *ARM, Freescale* ha codificado esta tabla de vectores como saltos a otra tabla de vectores secundaria que residirá al principio de la RAM, concretamente a direcciones 0x0040 0000 – 0x0040 000f. Por lo tanto, debemos configurar nuestra imagen para que aloje una tabla de vectores de excepción en estas direcciones.

Cómo explotar el cargador de arranque de la placa

El hecho de que el *bootloader* de la placa pruebe a buscar una imagen ejecutable en la UART o el puerto SPI si no la encuentra en la flash abre muchas posibilidades interesantes para agilizar el desarrollo de aplicaciones para la placa.

Ejecución de una imagen de firmware sin usar *OpenOCD*

Para ello, simplemente tendríamos que escribir un pequeño programa que se ejecute en nuestro PC y que transmita, a través del puerto serie al que está conectada la *Econotag*, y en cuanto se pulse el botón de *reset* de la placa, primero 4 bytes indicando el tamaño de la imagen, y después la propia imagen. Aunque el desarrollo de esta herramienta no es muy complejo, queda fuera de los contenidos de la asignatura, por lo que para ello se usará una herramienta llamada `mc1322x-load`, desarrollada por Mariano Alvira, el diseñador de la *Econotag*, dentro de su biblioteca *libmc1322x*[1]. El código fuente de dicha herramienta está en `/fenix/depar/atc/se/pracs/tools`.

Para ejecutar una imagen en la placa simplemente hay que ejecutar la siguiente orden:

```
mc1322x-load -f <imagen.bin> -t /dev/ttyUSB1
```

Podemos comprobar que subir y ejecutar una imagen de esta forma es mucho más rápido que si lo hacemos a través de *OpenOCD*.

Grabación de una imagen ejecutable en la flash

Es relativamente sencillo si usamos la herramienta `mc1322x-load` para ejecutar en la placa una aplicación que grabe su flash con el contenido que queramos. Lo interesante en este punto es saber escribir una aplicación que se ejecute en la *Econotag* y que escriba en su memoria flash otra imagen diferente. Aunque toda la información que necesitamos está en el manual de referencia del *Freescale MC1322x*, de momento es demasiado pronto para abordar una aplicación de esa envergadura, por lo que volveremos a hacer uso de la biblioteca *libmc1322x* de Mariano Alvira, que en su directorio `tests` nos proporciona un programa llamado `flasher.c` que hace justo eso.

Para no tener que construir toda la biblioteca de Mariano, podemos encontrar la imagen de `flasher.c` construida para la *Econotag* en `/fenix/depar/atc/se/pracs/tools`. Haciendo uso de la herramienta `mc1322x-load` y de la imagen `flasher_redbee-econotag.bin` podemos grabar cualquier imagen ejecutable en la flash de la placa mediante la siguiente orden:

```
mc1322x-load -f flasher_redbee-econotag.bin -s <imagen.bin> -t /dev/ttyUSB1
```

Si no hay errores a la hora de grabar la imagen en la flash, en cuanto reseteemos la placa comenzará su ejecución.

Borrado de la flash de la placa

Si queremos ejecutar otra aplicación en la placa, grabar la flash con otro *firmware*, o bien si ha ocurrido cualquier error en el proceso de grabación de la flash y la placa ya no arranca, tendremos

1 https://github.com/malvira/libmc1322x/wiki

que borrar la flash.

Para ello haremos uso de la herramienta `bbmc`, que también forma parte de la biblioteca *libmc1322x*, y que borra la flash de la placa mediante *bit banging*. El código fuente de dicha herramienta está en `/fenix/depar/atc/se/pracs/tools`. Para construirlo hay que tener instalada la biblioteca de desarrollo de *ftdi*, que ya instalamos para poder construir *OpenOCD*, por lo que su construcción no debería plantear ningún problema.

Para borrar la flash de una placa simplemente hay que ejecutar la siguiente orden:

```
bbmc -l redbee-econotag erase
```

Ejercicios

El objetivo final de esta serie de ejercicios completar el *script* de enlazado y el entorno de ejecución del BSP, que podemos encontrar en `/fenix/depar/atc/se/pracs/bsp`, para que se pueda ejecutar la aplicación *C* que se encuentra en `/fenix/depar/atc/se/pracs/app`.

Para el desarrollo de los ejercicios se usará *OpenOCD* para copiar la imagen del PC de desarrollo a la RAM de la placa, por lo que no se deben insertar al comienzo de la imagen los 4 bytes indicando su tamaño. Una vez que comprobemos que la imagen funciona correctamente, se usará la herramienta `mc1322x-load`, que insertará esta cabecera en la imagen antes de grabarla en la flash de la placa.

El *script* de enlazado de la plataforma

Cuando se hace un *port* de un BSP a una plataforma nueva es fundamental generar un *script* de enlazado que se adapte al mapa de memoria y a las restricciones de arranque de la placa. Para ello simplemente hay que describir en la directiva *Regions* la configuración de las memorias de la placa y en la directiva *Sections* cómo se debe mapear la imagen a la plataforma.

Como en nuestro caso la placa contiene un *bootloader* que limpia la RAM al arrancar y copia la imagen al principio de la RAM, el *script* de enlazado es bastante sencillo, ya que simplemente hay que indicar en la directiva *Memory* los datos referentes al mapa de memoria de la RAM y en la directiva *Sections* cómo debe organizarse la imagen en la RAM para que pueda ejecutarse correctamente. Dado que la copia de la imagen en la RAM se realiza por el *bootloader*, no será necesario definir direcciones físicas en el *script* de enlazado, por lo que a todas las secciones que definamos se le asignarán sólo direcciones virtuales en la RAM.

La primera sección que debemos definir es una sección que indicará la estructura de la imagen que se copiará a la RAM. Teniendo en cuenta la información de la documentación de la placa, al principio de la imagen debe estar el código del cargador, que es el que tiene definidos los vectores de excepción (fichero `hal/crt0.s`). Tras el código del cargador podemos mapear el resto de secciones habituales (`.text`, `.data` y `.rodata*`).

Una vez mapeada la imagen a la RAM, nos queda por definir el resto de secciones que nos ayudarán a configurar el entorno de ejecución. Necesitaremos una sección `.bss` a continuación de la imagen para mapear todas las variables inicializadas a cero y sin inicializar (secciones `.bss` y `.COMMON` de los objetos que enlacemos), una sección `.stacks` para las pilas colocada al final de la memoria RAM, y otra sección `.heap` mapeada al espacio de RAM restante entre la sección `.bss` y las pilas.

Con la ayuda de estas secciones podremos definir los siguientes símbolos, que son imprescindibles

para que nuestro BSP funcione correctamente:

- `_bss_end` Final de la sección `.bss`. Es necesario porque además también marca el principio del *heap*
- `_heap_start` Comienzo del *heap*. Debe coincidir con el final de la sección `.bss`
- `_heap_end` Final del *heap*. Además debe coincidir con el fondo de la zona de memoria reservada para las pilas
- `_sys_stack_top` Tope de la pila del modo *System/User*
- `_svc_stack_top` Tope de la pila del modo *Supervisor*
- `_abt_stack_top` Tope de la pila del modo *Abort*
- `_und_stack_top` Tope de la pila del modo *Undefined*
- `_irq_stack_top` Tope de la pila del modo *IRQ*
- `_fiq_stack_top` Tope de la pila del modo *FIQ*

El entorno de ejecución de *C*

Normalmente, todos los BSP tienen un fichero `crt0.s` en el que se define el entorno de ejecución *C*. El nombre se deriva de que es el paso inicial (paso 0) de la inicialización del *C RunTime* (CRT). En nuestro caso, dicho fichero se encuentra en `hal/crt0.s`. En dicho fichero se realizan todos los pasos necesarios para arrancar la placa, inicializar sus dispositivos, colocar la imagen de la aplicación en el mapa de memoria, preparar las pilas y el *heap,* habilitar las interrupciones y saltar a *main.* El contenido del fichero `crt0.s` es totalmente dependiente de la plataforma, ya que, por un lado cada plataforma tiene un mapa de memoria diferente, y por otro, los pasos a llevar a cabo dependerán de la existencia de un cargador de arranque en la ROM y de las funciones que lleve a cabo dicho cargador.

Como en nuestro caso existe un cargador que limpiará toda la RAM, transferirá la imagen al comienzo de la RAM y saltará a la primera dirección de la RAM, nuestro cargador de arranque sólo tendrá que:

- Definir las tablas de vectores y de manejadores de excepción al principio de la RAM, ya que según el manual del *Freescale MC13224* se espera que los primeros 32 bytes de la RAM contengan la tabla de vectores de excepción. El vector de la excepción *reset* se codificará como un salto a *_start*, la primera instrucción de nuestro cargador.
- Inicializar las pilas de cada modo de ejecución cuanto antes para poder hacer el resto de las funciones de inicialización en *C*.
- Llamar a la función *bsp_init*, que inicializa los dispositivos y la E/S estándar en la placa. Se recomienda inspeccionar el código de esta función para entender las tareas que se llevan a cabo a la hora de terminar de generar el entorno de ejecución *C*.
- Cambiar a modo *User* y habilitar las interrupciones.
- Saltar a *main*.

Comprobación de los ejercicios anteriores

En `/fenix/depar/atc/se/pracs/app` hay una aplicación escrita en *C* estándar para la *Econotag*. Construir y ejecutar la aplicación en la placa. Su los ejercicios anteriores se han realizado correctamente, la aplicación se ejecutará sin problemas.

Primer programa en *C*

Modificar la aplicación anterior para que se pueda cambiar el led que está parpadeando (rojo o verde) según el botón que se pulse. La pulsación del botón S3 seleccionará el led verde, mientras que el botón S2 seleccionará el led rojo. Al iniciarse el programa estará seleccionado el led rojo por defecto. Para el desarrollo de esta aplicación se puede adaptar el código desarrollado en la práctica anterior en ensamblador.

Ejecución de nuestro programa *C* en la placa haciendo uso de mc1322x-load

Ejecutar la imagen de nuestro programa en la placa. Para ello disponemos de la regla `run2` en el `Makefile` de la aplicación de ejemplo.