Práctica 8

Desarrollo de un driver L0 para las UART

Introducción

Una vez que tenemos las excepciones, el controlador de interrupciones y el GPIO correctamente gestionados en nuestro BSP es el momento de comenzar con el desarrollo de *drivers*. En esta práctica y en las siguientes nos centraremos en el desarrollo de un *driver* para las UART de la *Econotag*. Concretamente esta práctica se dedicará a la implementación del *driver* L0 de las UART, en el que se ofrecerá una funcionalidad mínima que consistirá en inicializar la UART y poder mandar y recibir caracteres mediante llamadas a funciones bloqueantes.

Objetivos

- Saber utilizar campos de bits, uniones y estructuras anónimas para mapear correctamente los registros de E/S de un dispositivo.
- Saber asignar el modo de funcionamiento y la dirección correcta a los pines de E/S para que sean gestionados por un dispositivo interno.
- Saber diseñar un *driver* L0 con la funcionalidad mínima para comprobar que un dispositivo funciona.

Las UART de la Econotag

El MC1322x de *Freescale* incorpora dos UART, UART1 y UART2, que permiten establecer una comunicación serie con cualquier dispositivo que también tenga otra UART. Las dos UART tienen exactamente la misma funcionalidad y los mismos registros de control, situados en diferentes direcciones base, tal y como se puede consultar en el capítulo 13 del manual de referencia.

Acceso estructurado a los pines y a los registros de control/estado

Dado que las dos UART de la *Econotag* tienen la misma funcionalidad, a la hora de desarrollar el API es preferible diseñar funciones genéricas para cualquier UART en las que el primer argumento indique la UART a la que se quiere acceder y el resto se usen como parámetros para cada función. Para facilitar el desarrollo de este tipo de funciones es importante definir correctamente las estructuras de datos que darán acceso al mapa de memoria de cada UART y a sus pines de E/S, de forma que se pueda acceder directamente a esta información a partir del identificador de cada UART.

Si consultamos el fichero `drivers/uart.c` de nuestro BSP, podemos observar que se han definido los *arrays* `uart_regs[]` y `uart_pins[]`, que darán acceso a los registros de control y a los pines de cada una de las UART. Para el caso del acceso a los pines, simplemente hay que indicar en cada entrada del *array* `uart_pins` a qué pines están conectadas las señales `TX`, `RX`, `CTS` y `RTS` de cada UART, tal y como se indica en el fichero `drivers/uart.c`. En el caso de los registros de control/estado, el *array* `uart_regs[]` está definido como un *array* de punteros a estructuras del tipo `uart_regs_t`, que mapean los registros de las UART mediante uniones de estructuras y *bitfields*. El

diseño de la estructura `uart_regs_t` se deja propuesto como ejercicio para el alumno. En cuanto a la dirección base de los registros de cada UART, los punteros `UART1_BASE` y `UART2_BASE` están definidos en el fichero `include/system.h`, junto al resto de parámetros de las UART.

Inicialización de las UART

Las dos UART del sistema se inicializan mediante la función `uart_init`, que es llamada dentro del cargador de nuestro BSP (`hal/crt0.s`) en la inicialización del sistema (función `bsp_init`, en `hal/bsp_init.c`). De momento, como vamos a implementar solamente un *driver* L0, el cometido de `uart_init` consiste simplemente en enmascarar la petición de interrupciones mientras se inicializa el dispositivo, fijar la frecuencia de la UART, y configurar adecuadamente el modo de funcionamiento y la dirección de sus pines de E/S. Más adelante, cuando diseñemos los niveles L1 y L2 del *driver* añadiremos el soporte de interrupciones y de dispositivos a la función de inicialización.

Lo primero que hay que hacer a la hora de inicializar un dispositivo es deshabilitarlo y enmascarar la generación de interrupciones. En el caso de las UART de la *Econotag* estas acciones se gestionan mediante el acceso a los campos `MTxR`, `MRxR`, `TxE` y `RxE` del registro `UCON`, que sirven para enmascarar la petición de interrupciones de transmisión, enmascarar la petición de interrupciones de recepción, habilitar la transmisión y habilitar la recepción respectivamente.

Una vez enmascarada la petición de interrupciones y deshabilitada la transmisión y recepción en la UART, se podrá fijar la frecuencia de operación modificando convenientemente los campos `BRINC` y `BRMOD` del registro `UBR`[1]. Los valores adecuados para estos campos vienen especificados en el manual de referencia, en función de la tasa de sobre-muestreo que queramos realizar y de la frecuencia de operación deseada. Sin embargo, si analizamos los valores de estas tablas, podemos deducir que para un sobre-muestreo de 8x (más que suficiente), los valores de `BRMOD` y `BRINC` se pueden obtener mediante las siguientes expresiones:

- `BRMOD = 9999`
- `BRINC = (baudrate * BRMOD) / (CPU_FREQ >> 4)`

Siendo `baudrate` la frecuencia de operación deseada para la UART (segundo parámetro de la función `uart_init`), y `CPU_FREQ` la frecuencia de operación del oscilador del sistema (frecuencia del reloj de la CPU, definida en `include/system.h`).

Una vez que hemos fijado la frecuencia, podemos pasar a definir el modo de funcionamiento y la dirección de los pines que usará la UART para comunicarse con el mundo exterior. Si consultamos la Sección 11.5.1.2 del manual, al asignar el modo de funcionamiento 1 a un pin, éste será gestionado por el periférico interno que le corresponda SÓLO SI EL PERIFÉRICO ESTÁ HABILITADO, por lo que antes de asignar el modo de funcionamiento a los pines debemos volver a habilitar la UART (bits `TxE` y `RxE` del registro `UCON`).

Con la UART habilitada, ya podemos asignar el modo de funcionamiento 1 a los pines `TX`, `RX`, `CTS` y `RTS` con ayuda del *array* `uart_pins` y de la función `gpio_set_pin_func` desarrollada en la práctica anterior. Por último, sólo nos queda definir la dirección de cada uno de los pines, que está descrita en la Sección 13.4.1 del manual de referencia, y retornar un cero para indicar al BSP que la UART se ha inicializado correctamente.

1 Para poder fijar la frecuencia es obligatorio que tanto la transmisión como la recepción de la UART estén deshabilitadas, según indica claramente en la Sección 13.4.3 del manual de referencia.

Envío de caracteres

Como indica el manual de referencia, el envío de un carácter se realiza escribiendo dicho carácter en el campo `Tx_data` del registro `UDATA`. Sin embargo, la escritura del carácter en dicho registro no garantiza que se vaya a enviar por la UART. Las UART del MC1322x de *Freescale* cuentan con una cola FIFO de 32 bytes que están esperando a ser enviados, por lo que la escritura de un carácter en `Tx_data` provocará que se encole dicho carácter en la cola FIFO para ser transmitido siempre que haya hueco en dicha cola. Si la cola FIFO estuviera llena, no se podría almacenar el carácter para ser transmitido, por lo que si simplemente nos limitamos a escribir en `Tx_data` sin comprobar antes si hay sitio en la cola FIFO, podríamos perder caracteres. Para saber si hay algún hueco libre en la cola FIFO se debe consultar el campo `Tx_fifo_addr_diff` del registro `UTxCON` de la UART, que indica el número de huecos libres en la cola FIFO en cada momento.

En la implementación de un *driver* L0 como el que nos ocupa, la función de enviar caracteres debe bloquearse en un bucle mientras que no haya algún hueco libre en la cola FIFO (es decir, mientras que `Tx_fifo_addr_diff` valga cero), y una vez que se salga de dicho bucle (habrá espacio en la cola FIFO), escribir el carácter en `Tx_data` (lo que provocará que dicho carácter se encole para ser transmitido).

Recepción de caracteres

Análogamente al proceso de envío de caracteres, las UART también tienen otra cola FIFO de 32 bytes para la recepción de caracteres, que se va llenando conforme la UART va recibiendo caracteres desde el exterior. En este caso, el campo el campo `Rx_fifo_addr_diff` del registro `URxCON` de la UART nos indica cuántos bytes se han recibido y están actualmente en la cola. La recepción de un carácter se realiza mediante la lectura del campo `Rx_data` del registro `UDATA`, lo que provoca que se extraiga de la cola FIFO dicho carácter.

Obviamente, no se podrán recibir caracteres si la cola FIFO está vacía, por lo que la implementación de la función de recibir caracteres debe bloquearse en un bucle mientras que la cola FIFO esté vacía, y una vez que se salga de dicho bucle (la UART habrá recibido algún carácter, que estará en la cola), leer el carácter en `Rx_data` (lo que provocará que se saque dicho carácter de la cola).

Conexión serie entre la Econotag y nuestro PC

Una vez que hemos desarrollado el *driver* L0 de la UART, ya podremos realizar transferencias de información entre la *Econotag* y cualquier otro dispositivo que disponga de un puerto serie, como por ejemplo nuestro PC. Para ello simplemente tenemos que instalar en nuestro PC un emulador de terminal serie. Dependiendo del SO de nuestro PC tenemos diversas opciones. Para *Linux* podemos contar, entre otros, con `picocom`, `minicom`, `gtkterm` o `putty`. En función del emulador que instalemos, las opciones de configuración serán diferentes. Sin embargo, los parámetros de configuración deben ser los mismos, ya que están dictados por la configuración de la UART de la *Econotag,* que se establece por la función `uart_init` mediante los parámetros definidos en `include/system.h`:

- Frecuencia: 115200 baudios
- Control de flujo: ninguno
- Paridad: ninguna

- Ancho de carácter: 8 bits

En cuanto al puerto de conexión, dependerá de si el programa que se ejecuta en la *Econotag* ha sido cargado mediante *OpenOCD* o la utilidad `mc1322x-load`, o bien por el cargador de la ROM de la placa desde su memoria flash.

En el caso de usar *OpenOCD* o la utilidad `mc1322x-load` para subir el programa a la placa, tanto una como otra usan el puerto `/dev/ttyUSB0` para conectarse a la placa, por lo que el puerto `/dev/ttyUSB1` se conectará a la UART1 del sistema y no habrá forma de acceder a la UART2 de la placa desde el PC.

Sin embargo, si el programa se carga directamente desde la flash del sistema sin ayuda de un cargador en el PC de desarrollo, `/dev/ttyUSB0` se asignará a la UART1 de la placa y `/dev/ttyUSB1` a la UART2, por lo que podremos usar las dos UART de la *Econotag*.

Ejercicios

El objetivo final de esta serie de ejercicios es añadir al BSP que estamos diseñando un *driver* L0 para las UART de la *Econotag*.

Acceso estructurado a los registros de control/estado de las UART

Implementar en el fichero `drivers/uart.c` la estructura `uart_regs_t`, que permita acceder a los registros de control/estado de las UART. Dado que en los registros de las UART hay campos de diversos tamaños, se aconseja usar uniones, *bitfields* y estructuras anónimas para facilitar el acceso a cada uno de ellos, de forma que se pueda acceder a cualquier registro, o bien a cualquier campo dentro de un registro, directamente a partir del puntero `uart_regs[i]`, siendo `i` `uart_1` o `uart_2`, (definidos en `include/uart.h`) en función de la UART que queramos usar.

Inicialización de las UART

Implementar la función `uart_init` para que inicialice correctamente las UART de nuestra plataforma.

Envío de caracteres

Implementar la función `uart_send_byte`, la función de envío bloqueante de nuestro *driver* L0.

Recepción de caracteres

Implementar la función `uart_receive_byte`, la función de recepción bloqueante de nuestro *driver* L0.

Prueba del *driver*

Crear una aplicación que controle el estado de los leds de la placa mediante el teclado del PC. Para ello, el PC abrirá una terminal para comunicarse con la placa a través de la UART1. Si se pulsa la tecla 'g' o la tecla 'r' en el teclado del PC, la aplicación de la placa debe cambiar el estado del led verde o rojo respectivamente (de encendido a apagado y viceversa). La pulsación de cualquier otra tecla motivará que se mande un mensaje de error desde la placa hacia la terminal del PC indicando que las únicas teclas válidas son la 'r' o la 'g'. Dado que los pines que gestionan los led se

configurarán como de salida, no se podrá obtener su estado leyendo del registro de datos del GPIO, por lo que será necesario mantener el estado de los leds en variables de la aplicación.