Práctica 7

Gestión de los pines de entrada/salida

Introducción

Hasta ahora hemos estado gestionando los leds y los botones de la placa mediante el acceso directo a los registros de E/S del GPIO. Aunque esta forma de hacer las cosas es asumible para programas de test como los que hemos venido haciendo, es claramente ineficaz cuando nos enfrentamos al diseño de *drivers* para dispositivos más complejos, como por ejemplo una UART. Además, no es conveniente que se permita el acceso directo a los registros de E/S de la plataforma desde la aplicación. El acceso a dichos registros debería hacerse mediante un API adecuada y de forma controlada por nuestro BSP. Por tanto, esta práctica se centrará en implementar un *driver* para el GPIO de nuestra placa, que nos permita gestionar los pines de E/S de una forma sencilla y segura.

Objetivos

- Saber diseñar estructuras de datos que se adapten correctamente al mapa de memoria de los registros de E/S de un dispositivo.
- Saber diseñar un conjunto mínimo de funciones sencillas que permitan hacer un uso básico de un dispositivo sin necesidad de conocer el mapa de memoria de la plataforma.
- Añadir a nuestro BSP capacidades de E/S básicas.

El GPIO de la Econotag

Normalmente todos los sistemas cuentan con un controlador de E/S paralela programable. La función de este controlador, aunque muy sencilla, es básica para la gestión de la E/S en un sistema, ya que es el circuito que se encarga de asignar la dirección a los pines (entrada o salida). En el caso de sistemas que incorporen periféricos o controladores dentro del propio chip (UARTs, temporizadores, etc.), el GPIO también nos permitirá definir si los pines se conectan a los dispositivos internos o bien a otros dispositivos externos al chip (sensores, motores, pantallas, memorias, etc.). Por último, en algunos casos, el GPIO también permite activar resistencias *pull-up* o *pull-down* para los pines que se configuren de entrada, lo que facilita la conexión de dispositivos o conectores externos al chip.

La forma de gestionar los pines del sistema, así como las posibles configuraciones para cada pin dependerán del fabricante, por lo que es fundamental contar con el manual de referencia. En nuestro caso, podemos consultar toda la información referente al GPIO de la *Econotag* en el capítulo 11 del manual de referencia del MC1322X de *Freescale*. En dicho manual podemos observar cómo existen registros para cada tipo de operación que se desee realizar sobre los pines de E/S. La activación de una opción en un determinado pin consistirá en poner a 1 el bit que le corresponda dentro del registro del GPIO. Como los registros de E/S son de 32 bits, y el MC1322X tiene 64 pines de E/S, dichos pines están agrupados en dos puertos, el puerto 0 que gestiona los pines 0 – 31, y el puerto 1 que gestiona los pines 32 – 63. La escritura de una máscara de 32 bits en un registro de control de un puerto permitirá, por tanto, configurar los 32 bits de puerto simultáneamente.

El driver del GPIO dentro de nuestro BSP

En esta práctica nos centraremos sólo en desarrollar un *driver* con la funcionalidad básica, que consiste en asignar direcciones, valores y modos de funcionamiento tanto a pines individuales como a múltiples pines dentro del mismo puerto de forma simultánea. La implementación de nuestro *driver* se encuentra en los ficheros `include/gpio.h` y `drivers/gpio.c` de nuestro BSP. El primero, que expone el API pública del *driver* a las aplicaciones, está completo. Sin embargo, el segundo fichero tiene las estructuras de datos y los cuerpos de las funciones sin implementar, por lo que a lo largo de esta práctica se plantearán ejercicios que irán completándolo.

Por otro lado, el fichero `drivers/gpio.c` depende del símbolo `GPIO_BASE`, que como todos los parámetros del BSP, está definido en `include/system.h`.

Gestión de la dirección de los pines

La gestión de la dirección de los pines de un puerto se realiza mediante los registros `GPIO_PAD_DIR`, `GPIO_DIR_SET` y `GPIO_DIR_RESET`. El primero de ellos es de lectura/escritura, por lo que permite consultar la dirección asignada a los pines del puerto así como fijar su dirección. Para cada uno de los pines, si su bit correspondiente vale cero será de entrada y si vale 1 será de salida.

La modificación de pines individuales de `GPIO_PAD_DIR` sin afectar el resto de pines del puerto requiere el uso de operaciones con máscaras de bits, por lo que para modificar la dirección de los pines se prefiere el uso de los registros de sólo-escritura `GPIO_DIR_SET` y `GPIO_DIR_RESET`. Una escritura en `GPIO_DIR_SET` fijará como de salida a todos aquellos pines en cuyo bit se escriba un 1 (escribirá unos en los bits correspondientes de `GPIO_PAD_DIR`), dejando la dirección del resto de los pines inalterada. Análogamente, una escritura en `GPIO_DIR_RESET` fijará como de entrada a todos aquellos pines en cuyo bit se escriba un 1 (escribirá ceros en los bits correspondientes de `GPIO_PAD_DIR`), sin afectar a la dirección del resto de los pines del puerto.

Lectura/escritura a través de los pines de E/S

Las operaciones de entrada/salida a través de los pines de realizan mediante los registros `GPIO_DATA`, `GPIO_DATA_SET` y `GPIO_DATA_RESET`. Al igual que en la gestión de la dirección de los pines, el registro `GPIO_DATA` es de lectura/escritura, por lo que sirve tanto para leer el estado de los pines de entrada como para escribir en los pines de salida. Análogamente, y aunque se puede escribir en un pin de salida modificando directamente los bits correspondientes del registro `GPIO_DATA`, se prefiere el uso de los registros de sólo-escritura `GPIO_DATA_SET` y `GPIO_DATA_RESET`, ya que se evita el uso de máscaras para realizar las operaciones de salida.

Selección del modo de funcionamiento de cada pin

El uso de los pines de E/S de un SoC suele estar multiplexado para minimizar su coste, de forma que desde el GPIO se puede gestionar a qué dispositivo físico estará finalmente conectado cada pin, ya sea un dispositivo interno o externo al chip. En el caso del GPIO del MC1322X de *Freescale*, cada pin puede tener 4 modos de funcionamiento diferentes, según se indica en el capítulo 11 de su manual de referencia. Para ello se usan los registros `GPIO_FUNC_SEL0`, `GPIO_FUNC_SEL1`, `GPIO_FUNC_SEL2` y `GPIO_FUNC_SEL3`. Cada uno dispone de dos bits para seleccionar cada uno de los cuatro modos de funcionamiento que puede tener un pin. El modo de funcionamiento 0 (*Normal Mode*) es la entrada/salida de propósito general, y se usa para poder gestionar dispositivos externos al chip. El resto de los modos de funcionamiento (1, 2 y 3, que se corresponden a los modos

Alternate Mode 1, *Alternate Mode* 2 y *Alternate Mode* 3 respectivamente) dependen de los dispositivos internos del chip asociados a cada pin.

Ejercicios

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

Acceso estructurado a los registros de control/estado del GPIO

Implementar en el fichero `drivers/gpio.c` la estructura `gpio_regs_t`, que permita acceder a los registros del GPIO de forma cómoda mediante las funciones del *driver*. Prestar especial atención a la forma en la que están mapeados los registros para decidir la implementación más adecuada (estructura de campos simples, estructura de *arrays* o *array* de punteros a estructuras).

Comprobar que los campos de la estructura están en la dirección correcta. Este paso es fundamental para que el resto de la práctica funcione correctamente. Para ello se puede hacer uso de la función de inspeccionar variables del depurador, indicándole que nos muestre las direcciones en memoria de cada uno de los campos de la estructura.

Asignación de direcciones a los pines del GPIO

Implementar en el fichero `drivers/gpio.c` las funciones `gpio_set_port_dir_input`, `gpio_set_port_dir_output`, `gpio_set_pin_dir_input` y `gpio_set_pin_dir_output`. Se recomienda hacer uso de los registros `GPIO_DIR_SET` y `GPIO_DIR_RESET` en vez de acceder directamente a `GPIO_PAD_DIR` para simplificar la implementación de estas funciones.

Operaciones básicas de E/S

Implementar en el fichero `drivers/gpio.c` las funciones `gpio_set_port`, `gpio_clear_port`, `gpio_set_pin`, `gpio_clear_pin`, `gpio_get_port` y `gpio_get_pin`. Al igual que en el ejercicio anterior, para las funciones de salida se recomienda hacer uso de los registros `GPIO_DATA_SET` y `GPIO_DATA_RESET` en vez de acceder directamente a `GPIO_DATA` para simplificar la implementación de estas funciones.

Selección del modo de funcionamiento de cada pin de E/S

Implementar en el fichero `drivers/gpio.c` las funciones `gpio_set_port_func` y `gpio_set_pin_func`, que permiten fijar el modo de funcionamiento de varios pines del mismo puerto, o bien el modo de funcionamiento de un determinado pin respectivamente. Estas funciones serán muy importantes en las siguientes prácticas, cuando implementemos el *driver* de las UART del sistema, ya que tendremos que asignar a sus pines de E/S el modo de funcionamiento adecuado para que sean gestionados por las UART en vez de por el GPIO.

Prueba del driver

Crear una aplicación que siempre mantenga un diodo de la *Econotag* parpadeando. Por defecto parpadeará el diodo rojo, aunque se podrá cambiar qué diodo debe parpadear mediante las pulsación de los botones. La pulsación del botón S3 hará que deje de parpadear el led rojo para que parpadee el verde, mientras que el botón S2 volverá a dejar la placa en su estado por defecto (el led rojo

parpadeando y el verde apagado). Para el desarrollo de esta aplicación sólo se podrá usar el API del *driver* del GPIO desarrollado en esta práctica, aunque se puede partir del código desarrollado en prácticas anteriores.