Práctica 1: Librerías

Las liberas a realizar serían:

E/S de propósito General

En la librería de entrada/salida general, las funcionalidades a implementar serían:

- configurar pin como entrada o salida
- activar desactivar pull-up
- leer en valor del pin
- escribir valor en el pin
- instalar función manejadora de interrupciones para pines de los puertos G y H

Todas las funciones tendrán el pefijo `gpio_` y se crearán los fichero `gpio.h` y `gpio.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_gpio_`.

Comunicación SPI

En la librería de la comunicación SPI las funcionalidades a implementar serían:

- configurar la velocidad de transmisión
- configurar la polaridad, fase y orden de los bits
- enviar un byte
- esperar a que termine envío/recepción
- devolver byte recibidos
- enviar array de bytes
- instalar función manejadora de cuando termine una recepción/envío

Todas las funciones tendrán el pefijo `spi_` y se crearán los fichero `spi.h` y `spi.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_spi_`.

Conversor A/D

En la librería del módulo conversor A/D las funcionalidades a implementar serían:

- configurar conversión de 8 o 10 bits
- configurar tiempo de muestreo
- configurar numero de conversiones sucesivas
- configurar modo de lectura: único pin, pines sucesivos
- configurar pin de inicio
- configurar el modo de conversión continua (SCAN)

- iniciar la conversión
- esperar a que termine conversión
- devolver los valores leídos
- instalar función manejadora para cuando termine conversión

Todas las funciones tendrán el pefijo `ad_` y se crearán los fichero `ad.h` y `ad.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_ad_`.

Generador PWM

En la librería del módulo generador de PWM las funcionalidades a implementar para cada uno de los 4 canales (del 0 al 3) serían:

- configurar el reloj base para un determinado canal.
- configurar la polaridad para un determinado canal.
- configurar el alineamiento para un determinado canal.
- configurar el número de etapas del periodo para un determinado canal.
- habilitar un determinado canal.
- modificar el ciclo de trabajo (en etapas) para un determinado canal.
- modificar el ciclo de trabajo (en porcentaje) para un determinado canal.

Todas las funciones tendrán el prefijo `pwm_` y se crearán los fichero `pwm.h` y `pwm.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_pwm_`.

Temporización

En la librería de temporización las funcionalidades a implementar serían:

- llevar un reloj absoluto (desde que se encendió sistema)
- devolver valor del reloj en microsegundos
- devolver valor del reloj en milisegundos
- realizar esperas solicitadas (sleep)
- instalar función que se ejecute tras un tiempo especificado
- instalar función que se realice periódicamente

Esta librería se tendrá que coordinar con las otras que utilizan en sistema temporizador (frecuencia del temporizador y canales utilizados).

Todas las funciones tendrán el prefijo `timer_` y se crearán los fichero `timer.h` y `timer.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_timer_`.

Generación/Medición de señales

En la librería de generación/medición de señales las funcionalidades a implementar serían:

En la parte de generación

- configurar canal como generador de señales
- configurar la forma de la señal a generar (en nano/micro/milisegundos)
- iniciar la generación
- detener la generación
- generar un determinado número de ciclos

En la parte de medición, para cada canal:

- configurar como medidor de señales
- devolver anchura último pulso recibido
- devolver el periodo del último ciclo recibido
- devolver el porcentaje en alta del último ciclo recibidos
- instalar una función manejadora para cuando se reciba algún flanco
- resetear la cuenta de pulsos
- devolver el número de pulsos recibidos
- instalar una función manejadora para cuando se reciba cierto número de pulsos

Todas las funciones tendrán el prefijo `signal_` y se crearán los fichero `signal.h` y `signal.c`.

Será necesario hacer uno o varios programa para probar todas las funcionalidades cuyos nombres deberán comenzar por `test_signal_`.

Guía de estilo

Al escribir el código fuente se debe de respetar las siguientes reglas:

- Indicar autor y fecha en las primeras líneas del fichero fuente.
- Usar dos espacios para cada nivel de sangrado, utilizando caracteres espacio.
- En cada línea no deberá haber más de una instrucción; aunque una instrucción puede ocupar varias líneas, en ese caso las líneas de continuación deberán tener dos niveles de sagrado más que la inicial (4 espacios más).
- Dejar un espacio antes y después de cada operador, en especial de los de asignación (`=`, `+=`, `-=`, …) y comparación (`>`, `<`, `==`, …).
- Las líneas deberán tener, preferiblemente, menos de 72 caracteres y nunca superar los 80.
- Ya que se compila con el estándar C99, declarar las variables justo antes de ser utilizadas y, si es necesario, hacer la declaración y la inicialización en la misma instrucción.
- No declarar más de una variable por línea.
- Se utilizarán comentarios Doxygen para documentar el módulo y las funciones.