ARQUITECTURA DE COMPUTADORES - PBL

2.CURSO DE GRADO DE INGENIERÍA INFORMÁTICA

KONPUTAGAILUEN ARKITEKTURA I

AUTOR:

Josu Garralda

Oihane Lameirinhas

Aitor Landa

Ane Sajeras

Ibai Rodriguez

En Arrasate, a 6 de junio de 2019

Tabla de contenido

1. INTRODUCCIÓN

En este documento se describe el funcionamiento de la placa STM32-P407 programada para llevar el control del aforo de un comedor social. Para ello, se ha implementado la conexión serial con el fin de establecer comunicación entre el ordenador y la placa.

El aforo se controlará de manera que cuando el comedor no supere el 70% de su aforo máximo una luz parpadeará en color verde. A medida que los comensales vayan entrando en el comedor, la placa irá incrementado la variable *persona* definida en *main.c* [Ver código]. Cuando el comedor alcance o supere el 70% de su ocupación la luz cambiará a parpadear en amarillo hasta que finalmente el aforo se complete. En ese momento además de cambiar la luz al color rojo, la placa emitirá una señal definida con el carácter *C* por la conexión línea serie y llegará a un ordenador con una aplicación gráfica desarrollada en Java que mostrará un diálogo emergente con el aviso de que el comedor a llegado al limite de su aforo.

Para tener una visión realista del aforo, al salir un comensal del comedor se restará a la variable *persona* para poder monitorizar el comedor en “tiempo real”. El control de la entrada y salida del comedor se controlaría mediante el sensor infrarrojo de proximidad MH-Sensor-Series, pero en esta primera versión del programa el control se lleva a cabo mediante los botones integrados en la placa *TAMPER* y *WKUP.* Estos botones son utilizados mediante el periférico EXTI, mediante el cual se generan una interrupción para cada botón en las cuales se suma o se resta una persona.

En lo relativo a la línea serie, se ha hecho uso del periférico USART3, que se explicará en detalle en el apartado X.

Como el reloj de la placa se ha utilizado el periférico SysTick, el reloj interno de 16MHz que integra la placa.

Por último, para habilitar los distintos periféricos se han usado los GPIOs correspondientes. Además de los GPIOs, siempre es necesario habilitar el periférico a utilizar en el RCC, *Reset and Clock Control*.

Todas las funciones mencionadas a lo largo de este documento están presentes en el código adjuntado.

2. GPIO

La placa STM32-P407 cuenta con diferentes GPIO (General Purpose I/O) para poder escribir/leer en ellos o pueden ser programados para diferentes periféricos. En este caso se han utilizado los GPIOs:

- GPIOA: se hace uso del pin 0 destinado al botón *WKUP,* el cual tiene un comportamiento de *push-pull*.
- GPIOC: es el GPIO destinado al botón *TAMPER* al contrario del anterior el estado normal de este es 1 y al pulsarlo se convierte en 0.
- GPIOD: se configura para el funcionamiento del USART3 utilizado para la comunicación de la línea serial. En este GPIO se habilitan los pines necesarios para enviar y recibir los datos (pines *RX*, *TX*). Estos se configuran en el modo alterno, ya que no es seguro que vayan a enviar o recibir siempre.
- GPIOF: configurados sus pines 6,7 y 8 para hacer que controlar las luces LED y su parpadeo. Se han declarado pines de salidas y gracias a su registro ODR () cada vez que se utiliza la función *togleGpioPinValue* invierte su valor, es decir, pasa de 0 a 1 y viceversa.

3. EXTI

El periférico EXTI es usado para la gestión de interrupciones generadas por otros periféricos. En este caso, se ha utilizado para administrar las interrupciones generadas por los dos botones anteriormente mencionados.

El periférico EXTI puede generar hasta 16 interrupciones diferentes, cada una de ellas se descomponen para cada GPIO (de la *A* a la *F*). Los diferentes EXTIs están agrupados de la siguiente manera:

- Los primeros 5 EXTI generan una interrupción cada uno, es necesario utilizar el periférico NVIC para desenmascarar la interrupción generada.
- Del EXTI5 al EXTI9 generan todos la misma interrupción, por lo que es necesario hacer uso del periférico NVIC para desenmascarar la interrupción del EXTI que estamos utilizando.
- Igual que el caso anterior del EXTI10 al EXTI15 generan la misma interrupción por lo que es necesario utilizar también el periférico NVIC.

Antes de configurar el EXTI en cuestión, es necesario realizar una pequeña configuración en el periférico SYSCFG (*System Configuration Controller*), donde en los registros CRx se debe indicar el EXTI que se quiere utilizar.

Respectivo a la configuración del EXTI en sí, se debe habilitar el FTSR (*Falling Trigger Selection Register*) o RTSR (*Rising Trigger Selection Register*). Estos registros sirven para especificar si se creara una interrupción al pulsar o soltar el botón. Por último, se debe especificar en el registro IMR la máscara del EXTI que se está utilizando.

3.1. NVIC

En el periférico NVIC hay que activar en el registro *ISER* (*Interrupt set-enable register*) el bit en el que se generará la interrupción para ello, se puede utilizar la siguiente línea de código:

$$NVIC \rightarrow ISER\left[\frac{Pos.\,del\ périferico}{32}\right] =$$

$$0x01 << (Posición\ del\ périferico\%32);$$

Esto se debe a que el registro ISER está dividido en 3 partes.

4. USART3

El periférico USART (*Universal Synchronous Asynchronous Receiver Transmitter*) se ha utilizado para implementar la funcionalidad de comunicación entre la aplicación en Java y la placa. La placa STM32-P407 cuenta con los conexiones RS-232, de las cuales se ha escogido utilizar la que se presenta con el periférico USART3. En este caso hemos usado la opción asíncrona del USART, es decir, hemos utilizado un UART. La opción asíncrona permite una comunicación de full dúplex.

Para configurar el UART, se puede dividir en dos partes. Por un lado, la parte que envía los datos, por otra parte, la parte que recibe los datos.

4.1. ENVIO DE DATOS

Para enviar datos desde el UART3 es necesario seguir los siguientes pasos.

1- Habilitar el UART3 en el bus APB1 del RCC. Concretamente, el bit 18 hay que ponerlo a 1.
2- Indicar en el registro BRR (*Baud Rate Register*), es decir, la tasa de señales en segundos que va a recibir, y enviar. Para ello se puede calcular de esta manera:

$USART3 \rightarrow BRR = \frac{clockFrequency}{baudRate}$, donde baudRate=9600

3- Activar en el CR1 (*Control Register 1*) el bit TE, concretamente el bit número 3 hay que ponerlo a 1. Habilita la transmisión de datos.
4- Activar en el CR1 (*Control Register 1*) el bit UE, concretamente el bit número 13 hay que ponerlo a 1. Habilita el USART.
5- Para escribir datos desde la placa al ordenador se usa la función *writeToUart.* En la función hay que mirar que en el registro SR (*Status Register*) el bit TXE (bit número 7) sea uno. Cuando lo sea hay que pasar carácter a carácter el mensaje a el DR (*Data Register*).

4.2. LEER DATOS

Para leer datos desde el UART3 es necesario seguir los siguiente pasos.

1- Habilitar el UART3 en el bus APB1 del RCC. Concretamente, el bit 18 hay que ponerlo a 1.
2- Indicar en el registro BRR (*Baud Rate Register*), es decir, la tasa de señales en segundos que va a recibir, y enviar. Para ello se puede calcular de esta manera:

$USART3 \rightarrow BRR = \frac{clockFrequency}{baudRate}$, donde baudRate=9600

3- Activar en el CR1 (*Control Register 1*) el bit RE, concretamente el bit número 2 hay que ponerlo a 1. Habilita la recepción de datos.
4- Activar en el CR1 (*Control Register 1*) el bit RXNEIE, concretamente el bit número 5 hay que ponerlo a 1. Este registro habilita la creación de interrupciones cuando haya habido una lectura.
5- Activar en el CR1 (*Control Register 1*) el bit UE, concretamente el bit número 13 hay que ponerlo a 1. Habilita el USART.
6- Para leer los datos que se envían del ordenador a la placa se tiene que habilitar en NVIC [1] el bit 7, para llegar a esa conclusión se ha utilizado la formula expuesta en el anterior apartado. Además del NVIC tiene que estar habilitado el bit RXNEIE. Por último, se comprueba que haya datos para leer mirando en el registro SR el bit RXNE (número 5). Si los hay se puede guardar el contenido del registro DR (*Data Register*) en la variable destinada a guardar el mensaje. Se debe poner el bit RXNE a 0 para resetear la interrupción.

Para el *handler* se usa la función *callback* que en el *main.c* se utiliza las interrupciones generadas para leer de la línea serie.

5. SysTick

El reloj SysTick es el reloj que gobierna el programa, es un reloj de 16MHz que genera interrupciones al llegar a 0. Para ello se hace una cuenta atrás. Los campos por configurar son los siguientes.

*SysTick→LOAD = (uint32_t)(getSysClockFrequency()/1000*ms - 1UL);*

SysTick→VAL = 0UL;

SysTick→CTRL |= 0x05; //enable, AHB clock

SysTick→CTRL |= 0x02;

El valor *LOAD* es la frecuencia que se le define al reloj. *VAL* es la variable donde se especifica el valor inicial. Pero al especificar en *LOAD* no es necesario especificar el valor *VAL.* En el registro *CTRL* se especifica, por un lado, si es cuenta atrás o no, y que reloj utiliza, en este caso *AHB*.