Perférico SSP

Para la comunicación SPI con el módulo transceptor nRF24l01 se necesitó hacer uso del periférico SSP1 del LPC1769. Si bien el microcontrolador cuenta con un módulo llamado SPI dedicado exclusivamente a la comunicación con ese protocolo, no fue posible implementarlo con el Kit Infotronic ya que los pines de salida requeridos estaban siendo usados por otro módulo (en este caso por la UART1 para la comunicación asincrónica), entonces se optó por usar el módulo SSP[[1]](#footnote-1) que se describe a continuación.

El periférico SSP (*Synchronous Serial Port*) es un controlador capaz de operar comunicaciones en distintos protocolos, entre los que se encuentra el SPI.[[2]](#footnote-2)

Las características más relevantes de este periférico son:

* Modo Master, Slave o Loop Back, con pines dedicados (MISO, MOSI, SSEL y SCK).
* Tramas de datos de 4 a 16 bits.
* 4 Fuentes de interrupción.
* Soporte para DMA.
* FIFOs de transmisión y de recepción con capacidad de 8 tramas.

# Registros

A continuación se presentan los registros configurables para los periféricos SSP0 y SSP1:

* **CR0** - Control Register 0

Permite configurar el tipo de trama (SPI), la velocidad del clock, el tamaño de la trama y los bits de configuración del clock CPOL y CPHA.

* **CR1** - Control Register 1

Selecciona los modos de operación y permite habilitar o deshabilitar el periférico.

* **DR** – Data Register

Registro que actúa de intermediario entre los FIFOs de entrada y salida. Los datos a enviar o recibir se escriben o se leen de este registro.

* **SR** – Status Register

Contiene flags que informan sobre el estado actual del módulo

* **CPSR** – Clock Prescale Register

Divisor para configurar la tasa de bits por segundo.

* **IMSC** – Interrupt Mask Set/Clear Register

Registro que permite habilitar las posibles causas de interrupción

* **RIS** – Raw Interrupt Status Register

Registro de sólo lectura que permite visualizar en el momento de la interrupción cuáles fueron las causas posibles (estén o no habilitadas la interrupciones para esos eventos).

* **MIS**

Registro de sólo lectura que permite visualizar en el momento de la interrupción cuáles fueron las causas posibles (habiendo sido habilitadas).

* **ICR**

Registro que permite borrar los flags correspondientes a la interrupción.

* **DMACR**

Registro para configuración de acceso directo a memoria.

Para esta aplicación no se hará uso de la característica de DMA y se trabajará con una trama de 8 bits de datos. La estrategia de programación para el envío y la recepción es muy similar a la implementada por la cátedra en la comunicación asincrónica usada con la UART a través de primitivas y drivers que operan sobre búferes de entrada y de salida. Se usarán interrupciones por lo que también se desarrollará la rutina correspondiente que las atienda.

# Funciones

**Inicialización**

**Transmisión**

La transmisión usando este periférico involucra dos funciones: *uint8\_t WriteTx (uint8\_t dato)* y *void Start\_Transmission (void)*

**Resumen**

La primera función, *WriteTx* es una función primitiva. Se encarga de recoger los datos de la aplicación y escribirlos en un buffer de transmisión global *Buff\_Tx\_SSP1* de tipo circular, el cual escribe usando un índice (puntero) *SSP1\_TxIn\_Idx.* Esta función controla la circularidad del buffer a través del correcto manejo del puntero y además se encarga de habilitar la transmisión, invocando a *Start\_Transmission* sólo cuando es necesario. Los parámetros que devuelve permiten dar información a la aplicación sobre el estado del buffer y de la transmisión.

*Start\_Transmission* se clasifica como función de tipo driver, ya que trabaja directamente con los registros asociados al periférico. Simplemente se encarga de leer el buffer de transmisión, al cual accede con su propio puntero, *SSP1\_TxOut\_Idx*, incrementándolo cada vez que lee un dato. El dato es pasado al registro SR del periférico, donde será transmitido automáticamente si no hay algún dato anterior en el registro FIFO de transmisión. La función implementa pequeñas demoras que son inherentes al funcionamiento del SSP.

**Funcionamiento y secuencia de envío**

Primero la aplicación llama a *WriteTx* pudiendo o no guardar el valor de retorno. *WriteTx* entonces verifica a través un flag (*tx\_buffer\_full*) si está habilitada para escribir el buffer, y en caso afirmativo guarda el dato en él e incrementa el puntero *SSP1\_TxIn\_Idx*. Si el buffer se encuentra lleno, el flag estará activo y la función devolverá “FULL”.

En la primera vez que se invoca a la función, es decir, cuando se escribe el primer dato a enviar, se llama a *Start\_Transmission* y así se enviará automáticamente. Cabe aclarar lo siguiente: Al configurar las interrupciones, este módulo no tiene la capacidad de interrumpir por cada dato (byte) recibido, sino que lo hará únicamente cuando su FIFO de recepción se encuentre *al menos medio lleno*, tal como lo indica el manual (bit RXIM del registro IMSC activado). Al ser un FIFO de 8 datos, eso quiere decir que la primera interrupción por recepción únicamente ocurrirá una vez hayan llegado al menos 4 datos. La función primitiva tiene en cuenta esto y devolverá “WARNING” cada vez que se hayan enviado datos pero posiblemente su interrupción correspondiente[[3]](#footnote-3) no suceda (por ejemplo, se hayan enviado menos de 4 datos). Bajo este criterio, la función devolverá “OK” al haber escrito el cuarto dato y volverá a contar desde cero.

A la hora de enviar el segundo dato y posteriores (en posteriores llamadas a *WriteTx*) ya no se invoca a *Start\_Transmission* sino que se habilitan las interrupciones por transmisión.

El periférico puede interrumpir cuando el FIFO de transmisión se encuentra *por lo menos medio vacío* (bit TXIM del registro IMSC activado). Lo que sucede con este modo es que se generan demasiadas interrupciones innecesarias (dada la velocidad del módulo) ya que esa condición se cumple casi todo el tiempo. Entonces se optó por poder controlar cuándo será habilitada esa condición de interrupción.

Cuando se escribe el nuevo dato se habilita la interrupción por transmisión y pasado un pequeño lapso, ésta se presenta, ejecutándose la función correspondiente que la atiende, *SSP1\_IRQHandler().* Esta función, entre otras cosas, se encarga de verificar a través de *SendSSP1()* (que no es más que una encapsulación de la función *Start\_Transmission* con otros agregados) si dispone datos nuevos en el buffer para enviar. Una vez enviados, deshabilita la interrupción por transmisión, a la espera de que *WriteTx* sea quien la vuelva a habilitar, si es que se escriben otros datos para que sean enviados.

1. El manual de usuario remarca que el módulo SPI está pensado como periférico heredado, y que se pretende se use el SSP como alternativa. [↑](#footnote-ref-1)
2. Ver anexo: Protocolo de Comunicación SPI. [↑](#footnote-ref-2)
3. Como se trata de una comunicación full-duplex, enviar un dato implica recibir otro prácticamente al instante. [↑](#footnote-ref-3)