Cómo configurar el SPI de una SmartFusion2

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/12/04/como-configurar-el-spi-de-una-smartfusion2/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

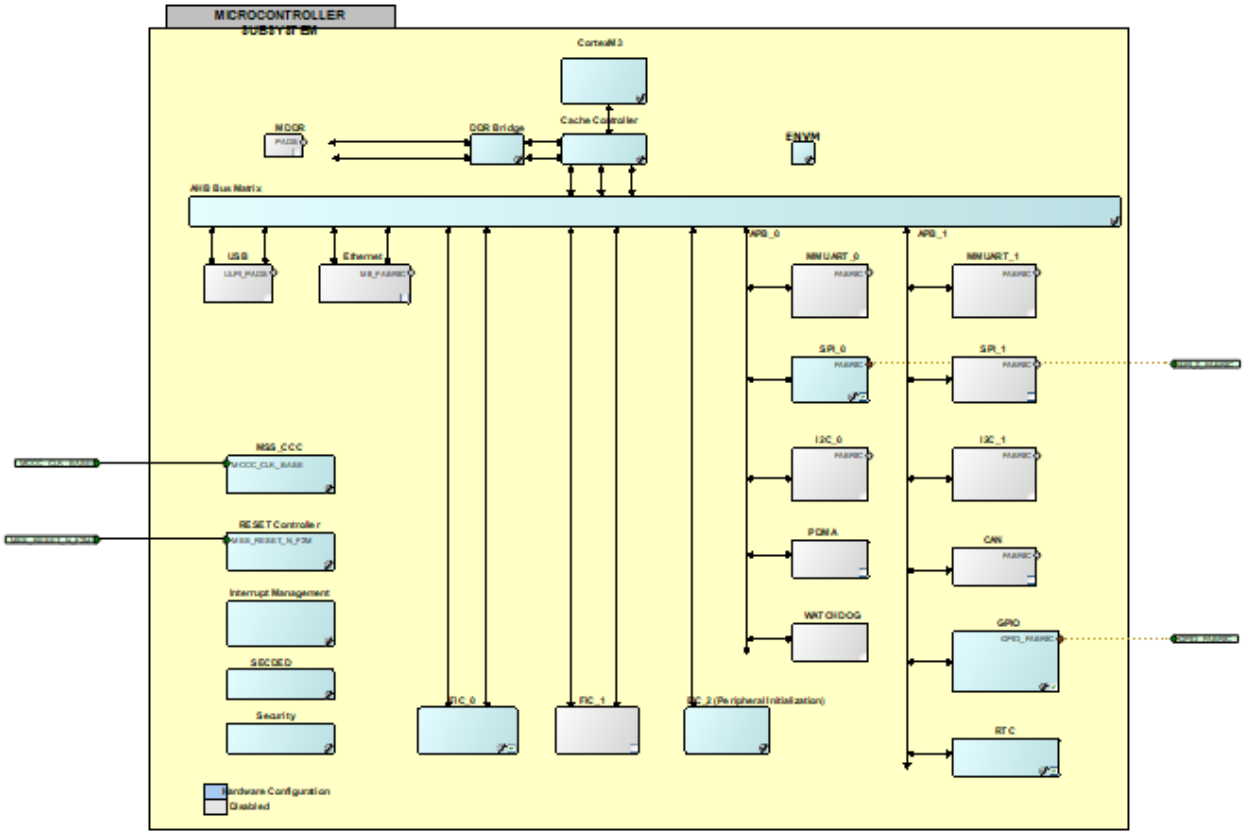
Para esta entrada nos vamos a basar estas tres entradas anteriores.

https://soceame.wordpress.com/2024/12/02/como-crear-un-proyecto-para-una-smartfusion2/
https://soceame.wordpress.com/2024/12/03/como-configurar-la-uart-de-una-smartfusion2/
https://soceame.wordpress.com/2024/12/04/como-configurar-el-i2c-de-una-smartfusion2/

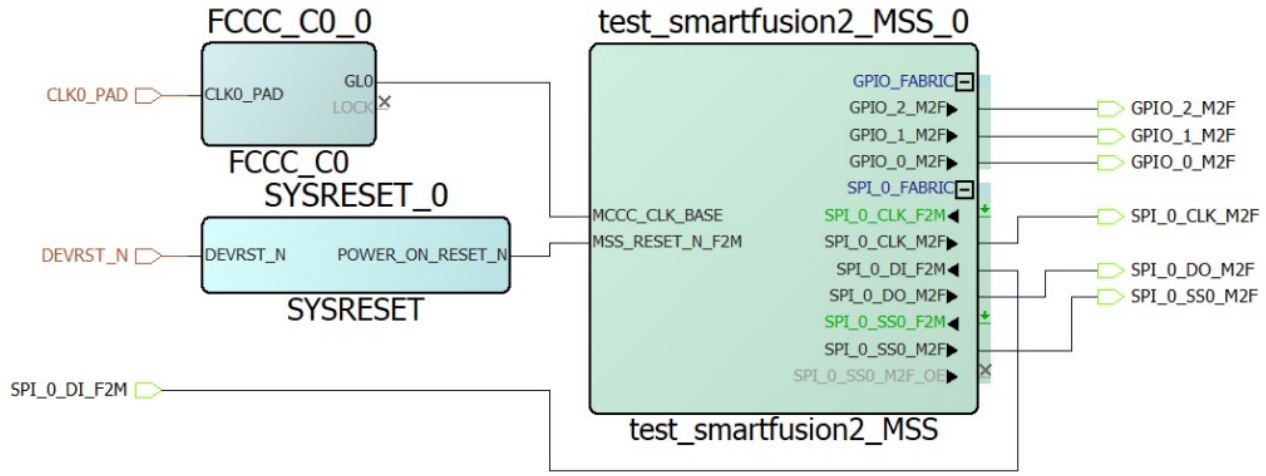
Proyecto en Libero

Lo primero que hay que hacer es crear un proyecto en Libero, utilizando como referencia las entradas anteriores.

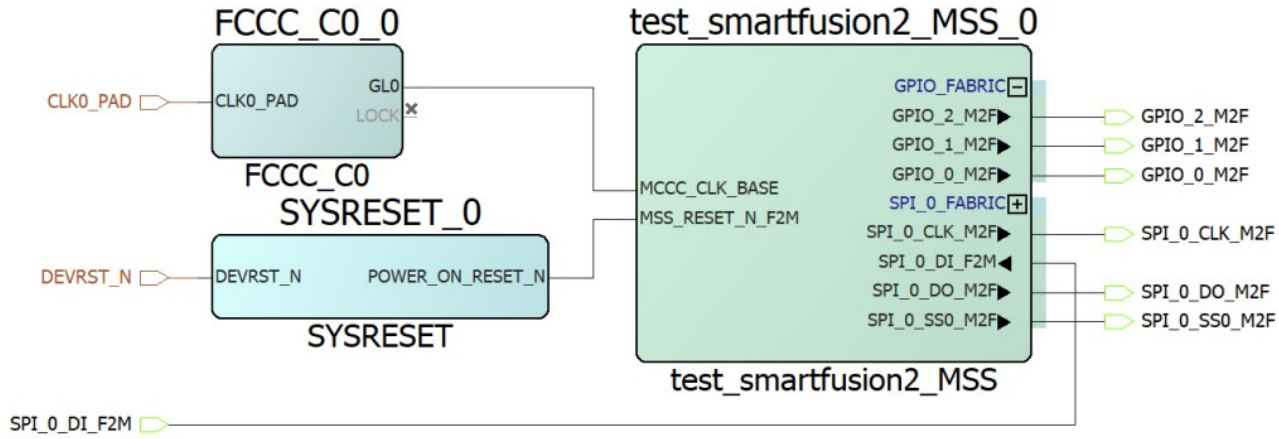
Entonces, dentro del bloque de la SmartFusion2 se configura el SPI0. Para terminar la configuración se da al icono amarillo del engranaje.



El siguiente paso es conectar el SPI0 con el exterior. Al aparecer los pines del SPI, aparecen tanto pines de SPI modo maestro como de esclavo. En nuestro caso elegimos los pines de modo maestro, que es el reloj como salida. También aparece un pin SS0_M2F_OE que es para conseguir que la línea CS del SPI sea bidireccional *(esto se haría como en la entrada anterior de configuración del I2C).*



Simplificado queda de la siguiente manera. Ahora se le da al icono amarillo con el engranaje, y después a *Build Hierarchy.*



El siguiente paso es la selección de los pines, para primero se sintetiza el modelo y después en el *Manage Constraints* se seleccionan los pines en *Edit I/O*.

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Free
1	CLK0_PAD	INPUT	LVCMOS25	23	☑	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	☑	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	☑	OUTBUF	Bank0	TRISTATE
5	GPIO_2_M2F	OUTPUT	LVCMOS25	125	☑	OUTBUF	Bank0	TRISTATE
6	SPI_0_CLK_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE
7	SPI_0_DI_F2M	INPUT	LVCMOS25		☐	INBUF	--	TRISTATE
8	SPI_0_DO_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE
9	SPI_0_SS0_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE

Ahora se eligen los pines.

https://soceame.wordpress.com/2020/12/06/como-generar-un-fichero-xdc/

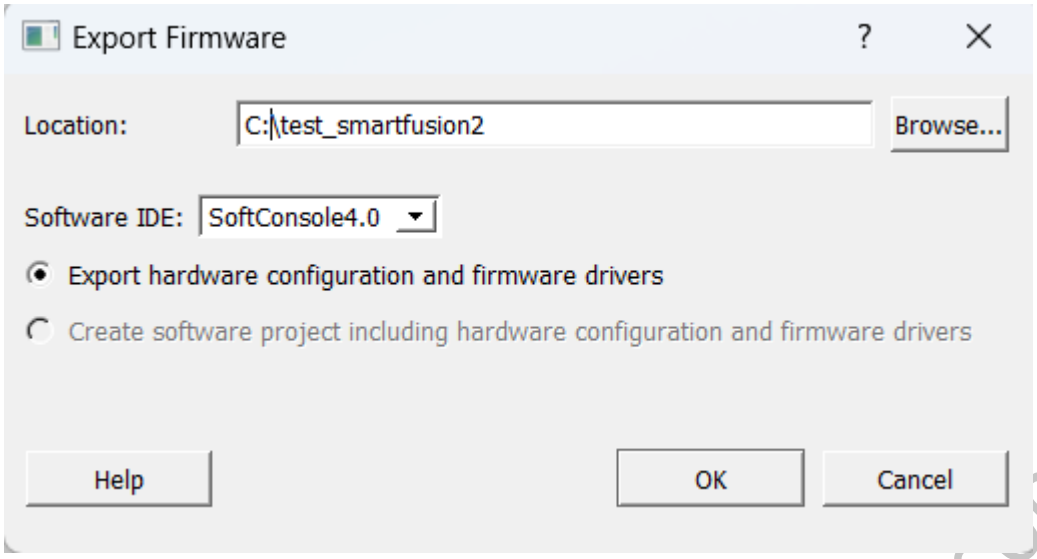
	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Fre
1	CLK0_PAD	INPUT	LVCMOS25	23	☑	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	☑	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	☑	OUTBUF	Bank0	TRISTATE
5	GPIO_2_M2F	OUTPUT	LVCMOS25	125	☑	OUTBUF	Bank0	TRISTATE
6	SPI_0_CLK_M2F	OUTPUT	LVCMOS33	83	☑	OUTBUF	Bank2	TRISTATE
7	SPI_0_DI_F2M	INPUT	LVCMOS33	90	☑	INBUF	Bank2	TRISTATE
8	SPI_0_DO_M2F	OUTPUT	LVCMOS33	81	☑	OUTBUF	Bank2	TRISTATE
9	SPI_0_SS0_M2F	OUTPUT	LVCMOS33	93	☑	OUTBUF	Bank2	TRISTATE

Ahora con los pines configurados, se genera el bitstream. Una vez creado seleccionamos los drivers a exportar.

Para ello en *Configure Firmware Cores* se selecciona los drivers del SPI.

	Generate		Instance Name	Core Type	Version	Compatible Hardware Instance
1	☑		SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.1(	test_smartfusion2_MSS
2	☑		SmartFusion2_MSS_GPIO_Driver_0	SmartFusion2_MSS_GPIO_Driver	2.1.1(	test_smartfusion2_MSS:GPIO
3	☑		SmartFusion2_MSS_HPDMA_Driver_0	SmartFusion2_MSS_HPDMA_Driver	2.2.1(	test_smartfusion2_MSS
4	☑		SmartFusion2_MSS_NVM_Driver_0	SmartFusion2_MSS_NVM_Driver	2.5.1(	test_smartfusion2_MSS
5	☑		SmartFusion2_MSS_RTC_Driver_0	SmartFusion2_MSS_RTC_Driver	2.2.1(	test_smartfusion2_MSS:RTC
6	☑		SmartFusion2_MSS_SPI_Driver_0	SmartFusion2_MSS_SPI_Driver	2.2.1	test_smartfusion2_MSS:SPI_0
7	☑		SmartFusion2_MSS_System_Services_Driver_0	SmartFusion2_MSS_System_Services_Driver	2.9.1(	test_smartfusion2_MSS
8	☑		SmartFusion2_MSS_Timer_Driver_0	SmartFusion2_MSS_Timer_Driver	2.2.1(	test_smartfusion2_MSS

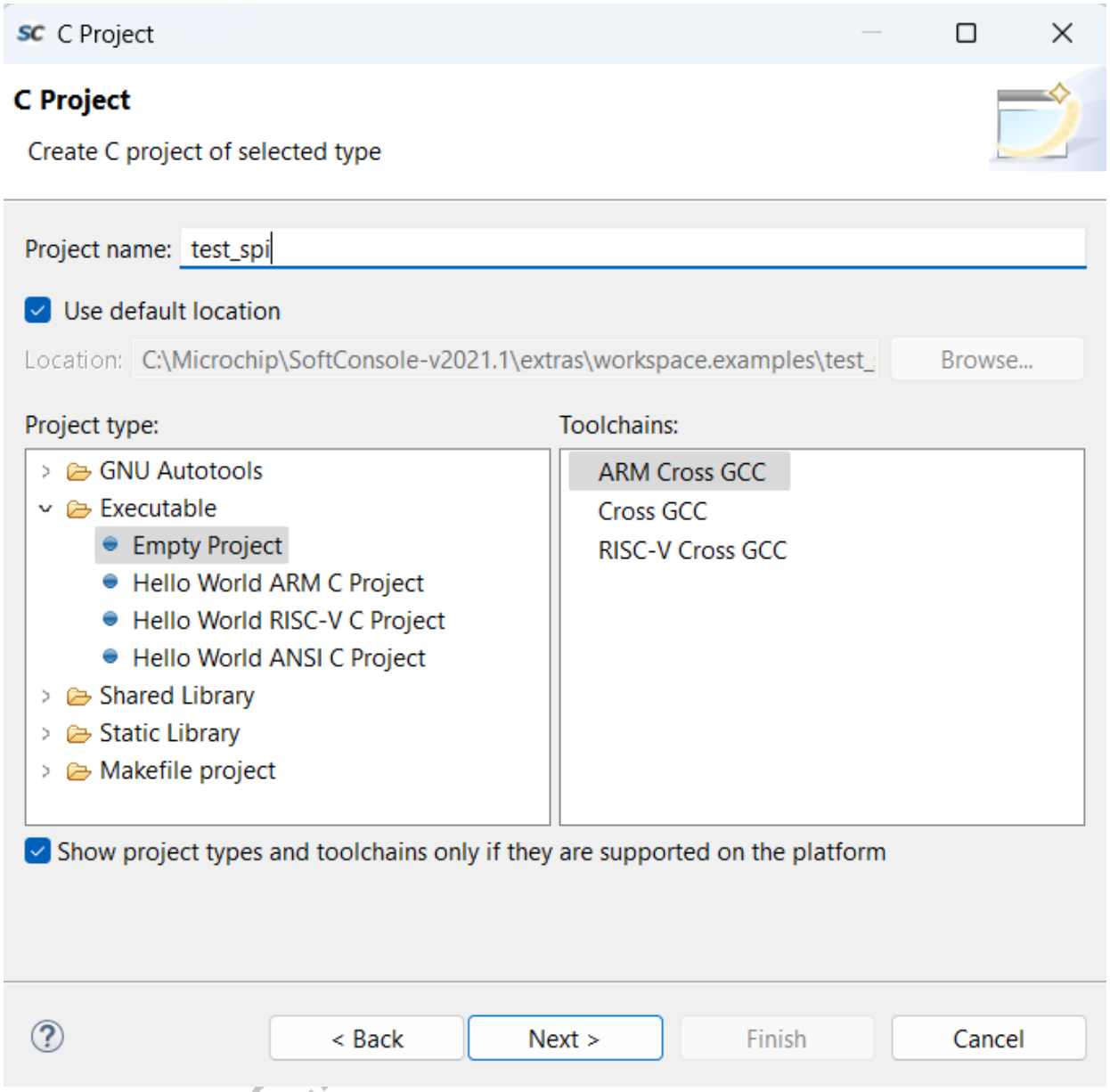
Después los exportamos.



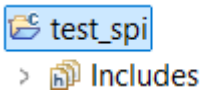
Proyecto en SoftConsole

Para trabajar con SoftConsole, lo primero que se crea es un proyecto en C vacío.

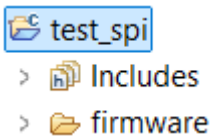
https://soceame.wordpress.com/2020/12/06/como-generar-un-fichero-xdc/



Al crearlo, primero nos crea la carpeta de los *includes*.



Luego importamos la carpeta de los drivers y creamos un fichero llamado *main.c*.



En este fichero main.c utilizamos un código como el siguiente.

```
#include "firmware/drivers/mss_spi/mss_spi.h"


void main(){
    uint8_t master_tx_buffer[10] =
      {
          0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA
      };
      uint8_t slave_rx_buffer[10];

const uint8_t frame_size = 25;

  MSS_SPI_init( &g_mss_spi0 );
  MSS_SPI_configure_master_mode
    (
        &g_mss_spi0,
        MSS_SPI_SLAVE_0,
        MSS_SPI_MODE1,
        256u,
        MSS_SPI_BLOCK_TRANSFER_FRAME_SIZE
     );

  MSS_SPI_set_slave_select( &g_mss_spi0, MSS_SPI_SLAVE_0 );

  MSS_SPI_transfer_block( &g_mss_spi0, master_tx_buffer,
          sizeof(master_tx_buffer),
          slave_rx_buffer,
          sizeof(slave_rx_buffer));

}
```

En él lo primero que hacemos es configuramos el SPI0 (*MSS_SPI_init*), después seleccionamos el esclavo 0 (esto no influye demasiado en caso de no saber el esclavo, se configura y te olvidas). Y por último se utiliza la función de envío.

Esta función de envío lo que hace es coger el dos buffers, uno de envío y otro de recepción, y utilizarlos. En nuestro caso como somos maestros comenzamos escribiendo utilizando el buffer de envío (master_tx_buffer) y después esperamos a que el esclavo nos envío tantos datos como el buffer de recepción sea de grande.

Para entendernos, pongo dos ejemplos.

- **Ejemplo 1**: si como en el ejemplo anterior se quiere enviar 10 datos, lo primero que se crea es un buffer de envío con los 10 datos, y si del esclavo se esperan 10 datos, se crea otro buffer de 10 datos, luego lo que se hace es enviar primero los datos de escritura y después el maestro espera los otros 10 datos que tiene que enviar el esclavo.

NOTA: si el maestro no recibe los datos en tiempo y forma los pone a 0xFF.

Envío (Send)	Recepción (Receive)

- **Ejemplo 2**: si lo que se quiere es un maestro de sólo escritura, lo que hay que hacer es crear un buffer de envío con los datos que se van a mandar al esclavo, y el buffer de recepción anularlo, ¿cómo?, poniendo una función tal que así.

```
MSS_SPI_tranfer_block(&g_mss_spi0, master_tx_buffer, sizeof(master_tx_buffer),
0, 0);
```

Esto hace que el SoC solo envíe datos.

Para hacerlo de solo lectura, se hace igual solo que anulando la escritura.

Ahora se compila el código, recordad configurar las variables del compilador de siempre.

Código

```
#include "firmware/drivers/mss_spi/mss_spi.h"


void main(){
    uint8_t master_tx_buffer[10] =
      {
          0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA
      };
      uint8_t slave_rx_buffer[10];

const uint8_t frame_size = 25;

  MSS_SPI_init( &g_mss_spi0 );
  MSS_SPI_configure_master_mode
    (
        &g_mss_spi0,
        MSS_SPI_SLAVE_0,
        MSS_SPI_MODE1,
        256u,
        MSS_SPI_BLOCK_TRANSFER_FRAME_SIZE
     );

  MSS_SPI_set_slave_select( &g_mss_spi0, MSS_SPI_SLAVE_0 );

  MSS_SPI_transfer_block( &g_mss_spi0, master_tx_buffer,
          sizeof(master_tx_buffer),
          slave_rx_buffer,
          sizeof(slave_rx_buffer));

}
```