How to configure the SPI of a SmartFusion2

Created by: David Rubio G.

Blog post: https://soceame.wordpress.com/2025/03/10/how-to-configure-the-spi-of-a-smartfusion2/

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

GitHub: https://github.com/DRubioG

Last modification date: 10/03/25

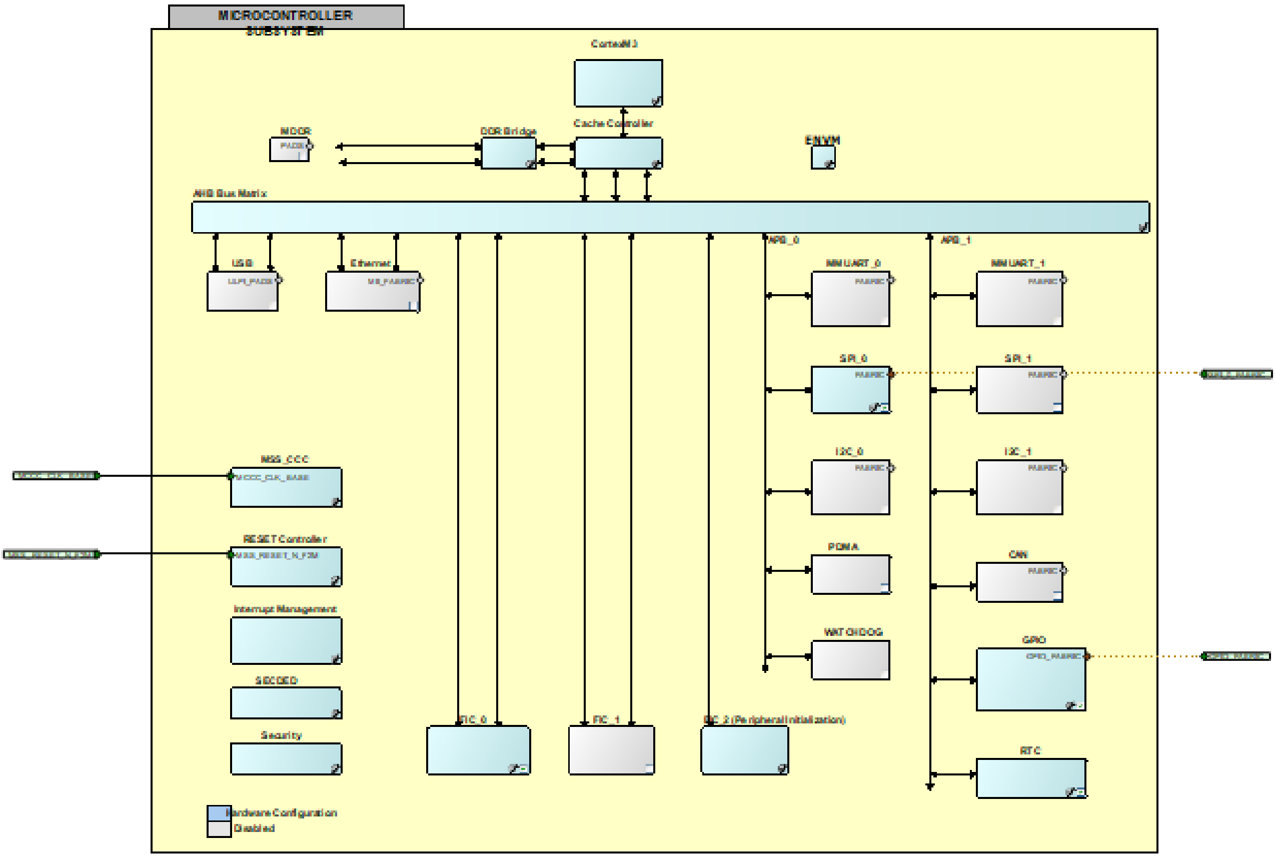
For this post we are going to base ourselves on these three previous posts.

https://soceame.wordpress.com/2025/03/09/how-to-create-a-project-for-a-smartfusion2-board/
https://soceame.wordpress.com/2025/03/10/how-to-configure-the-uart-of-a-smartfusion2/
https://soceame.wordpress.com/2025/03/10/how-to-configure-i2c-on-a-smartfusion2/

Project in Libero

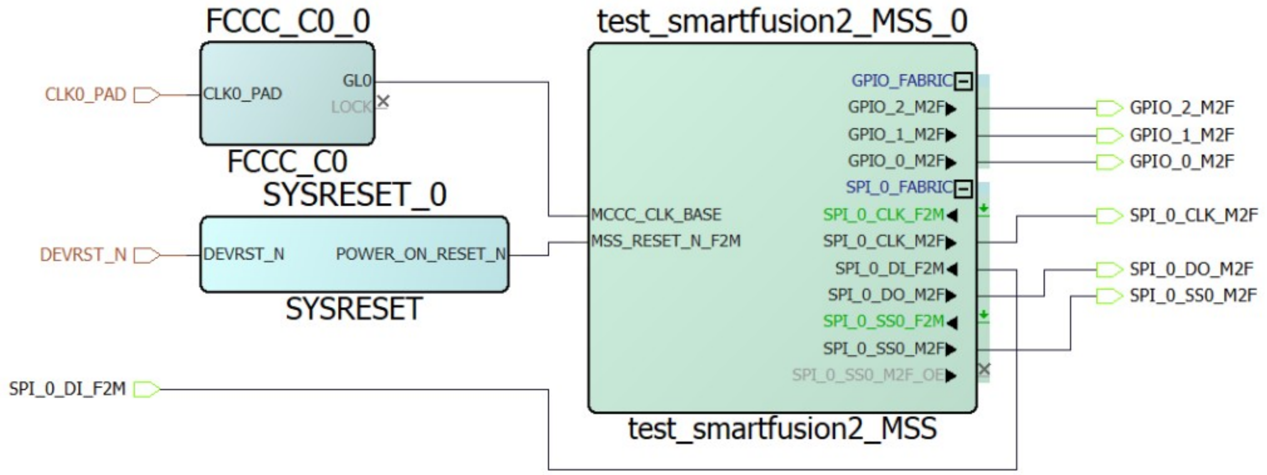
The first thing to do is to create a project in Libero, using the previous posts as a reference.

Then, within the SmartFusion2 block, the SPI0 is configured. To finish the configuration, click on the yellow gear icon.

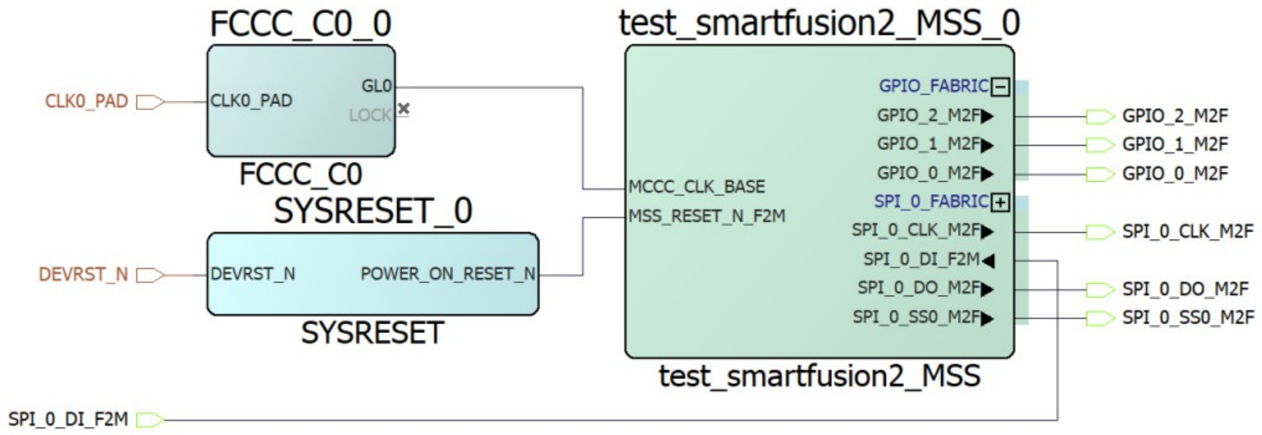


The next step is to connect the SPI0 to the outside. When the SPI pins appear, both master and slave SPI pins appear. In our case, we choose the master mode pins, which is the clock output. There is also a SS0_M2F_OE pin that is used to make the SPI CS line bidirectional *(this would be done as in the previous I2C configuration post)*.

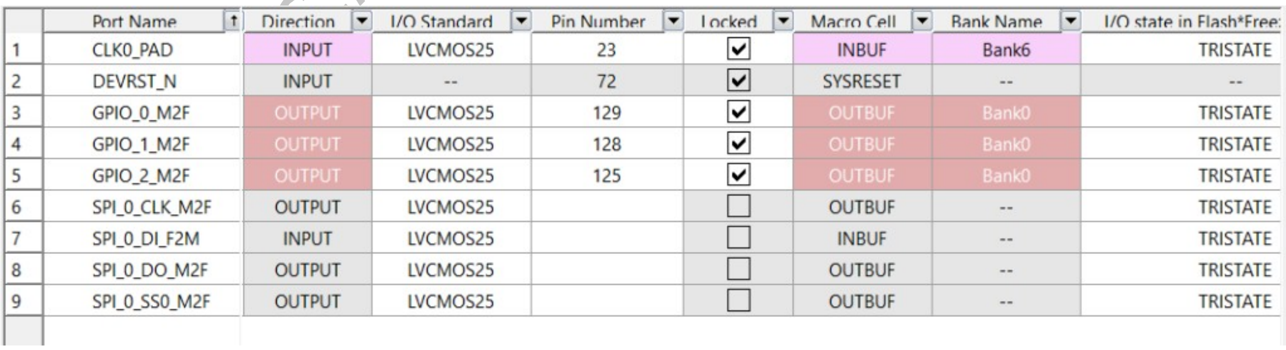
https://soceame.wordpress.com/2025/03/10/how-to-configure-the-spi-of-a-smartfusion2/



Simplified, it looks like this. Now click on the yellow gear icon, and then on *Build Hierarchy*.

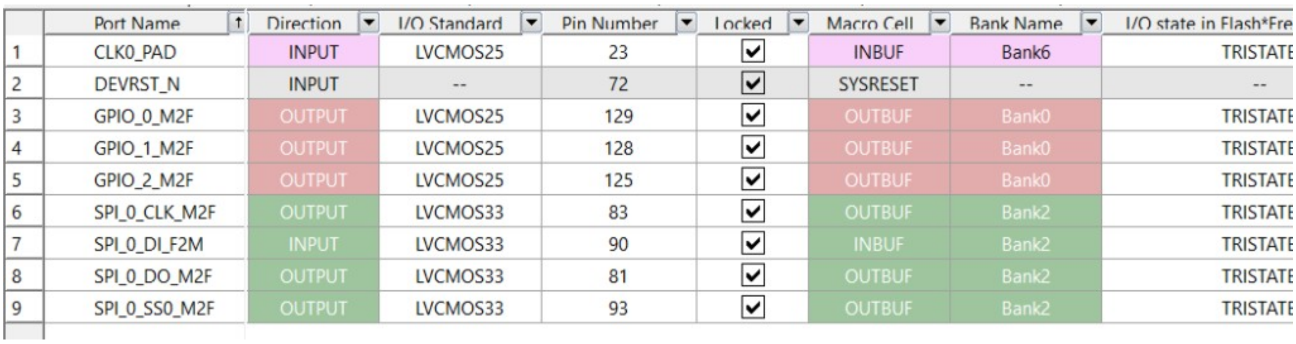


The next step is the selection of the pins. First, the model is synthesized and then the pins are selected in *Edit I/O* in *Manage Constraints*.



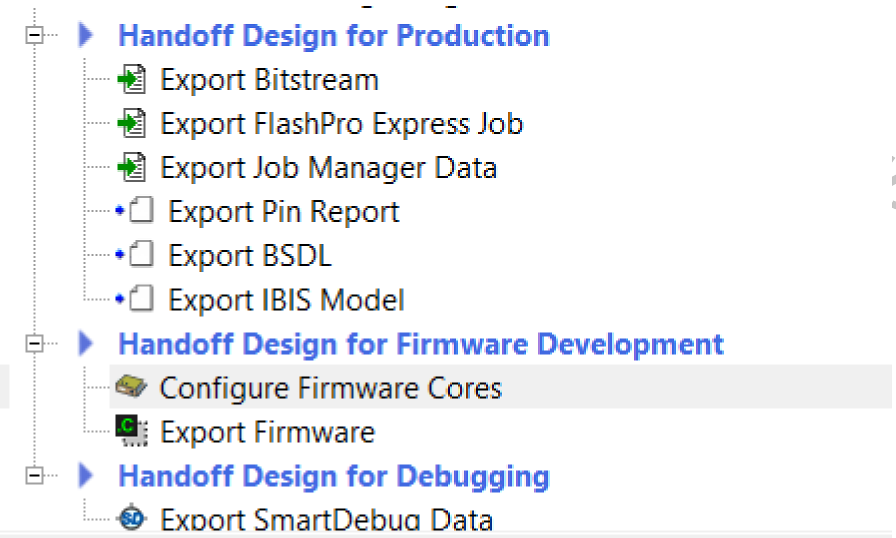
	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Free:
1	CLK0_PAD	INPUT	LVCMOS25	23	☑	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	☑	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	☑	OUTBUF	Bank0	TRISTATE
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

Now the pins are selected.

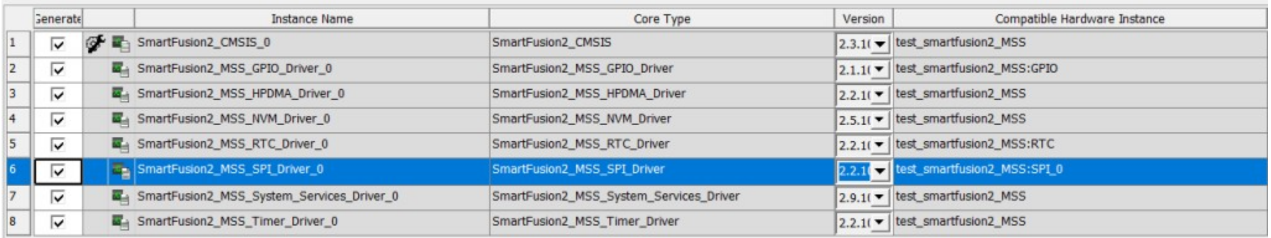


	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Fre
1	CLK0_PAD	INPUT	LVCMOS25	23	☑	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	☑	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	☑	OUTBUF	Bank0	TRISTATE
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

Now with the pins configured, the bitstream is generated. Once created, we select the drivers to export.

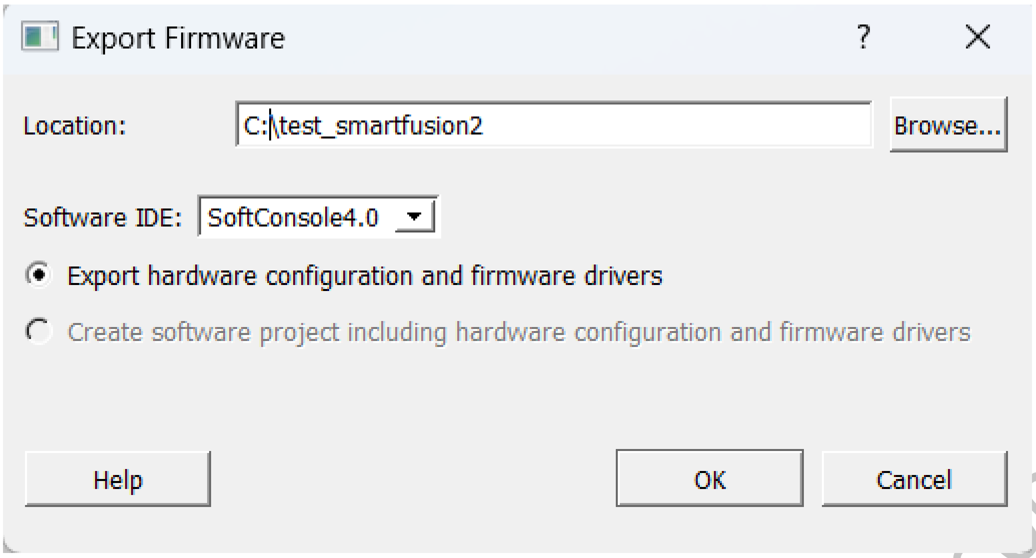


To do this, in *Configure Firmware Cores*, the SPI drivers are selected.



	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

Then we export them.



Export Firmware

Location: C:\test_smartfusion2

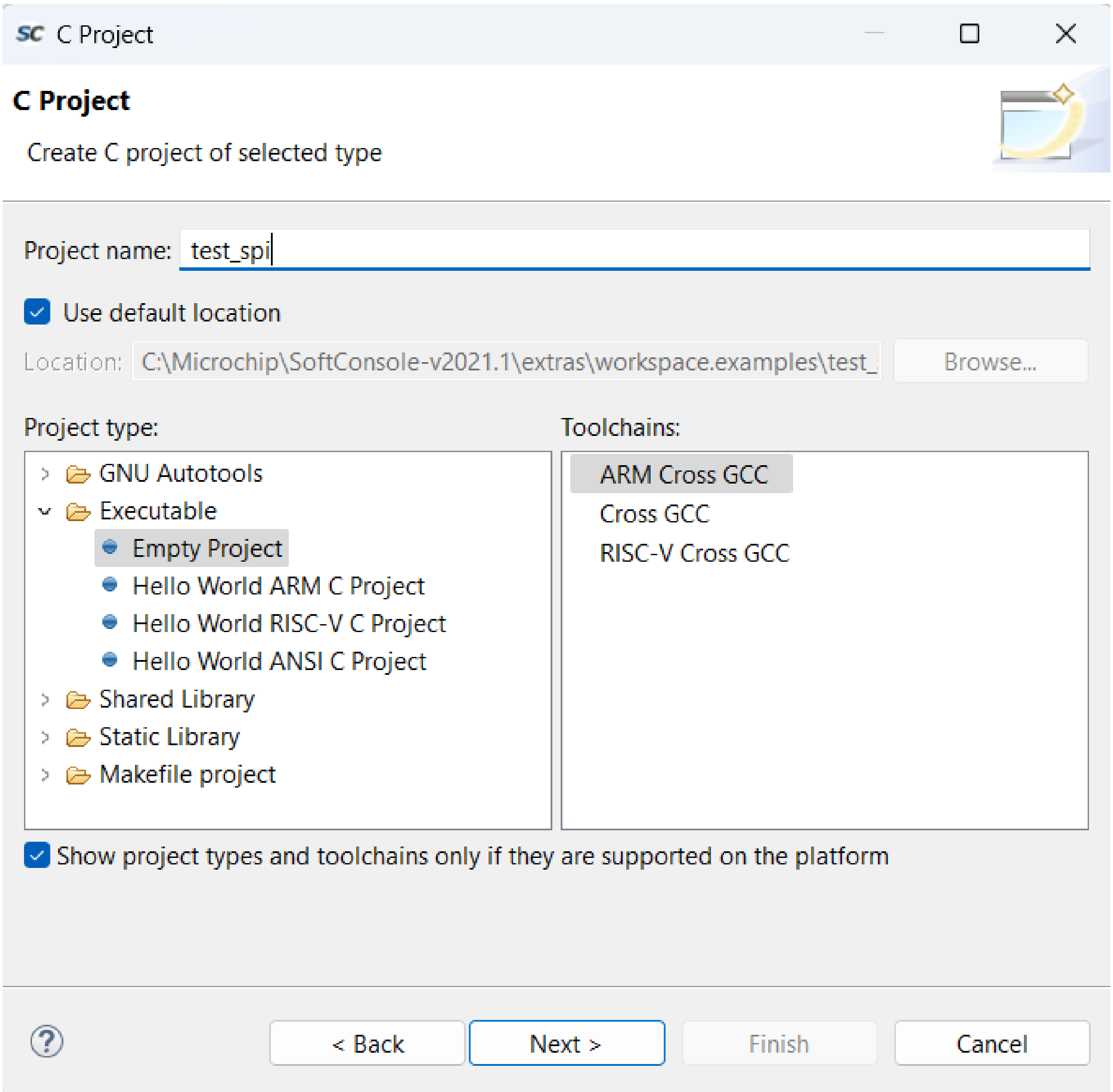
Software IDE: SoftConsole4.0

● Export hardware configuration and firmware drivers

○ Create software project including hardware configuration and firmware drivers

Project in SoftConsole

To work with SoftConsole, the first thing you create is an empty C project.



C Project

C Project

Create C project of selected type

Project name: test_spi

☑ Use default location

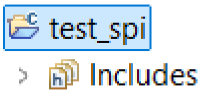
Location: C:\Microchip\SoftConsole-v2021.1\extras\workspace.examples\test_

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ARM C Project
  - Hello World RISC-V C Project
  - Hello World ANSI C Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- ARM Cross GCC
- Cross GCC
- RISC-V Cross GCC

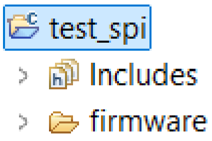
☑ Show project types and toolchains only if they are supported on the platform

When you create it, it first creates the *includes* folder.



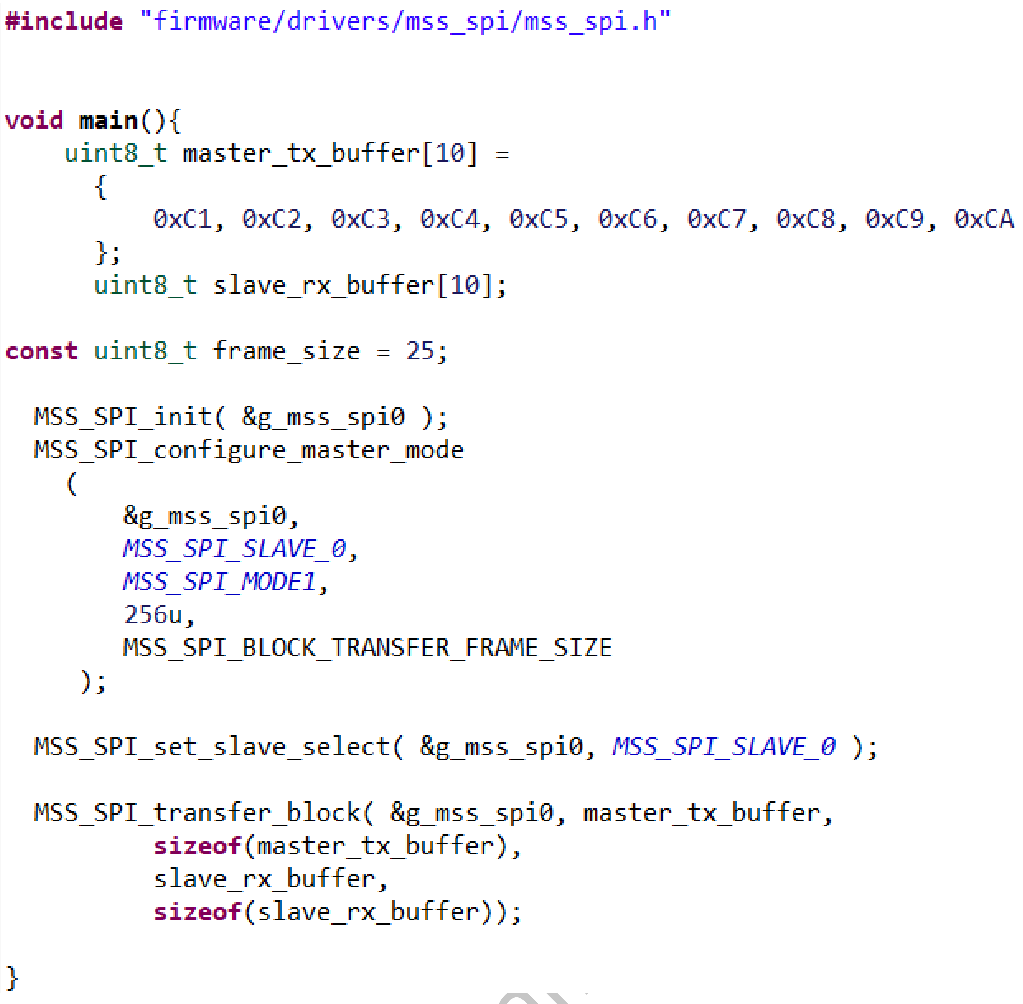
test_spi
- Includes

Then we import the drivers folder and create a file called *main.c*.



test_spi
- Includes
- firmware

In this main.c file we use code like the following.



```
#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));

}
```

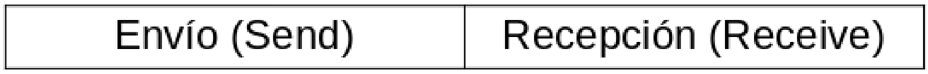
In it, the first thing we do is configure SPI0 (*MSS_SPI_init*), then we select slave 0 (this doesn’t have much of an effect if you don’t know the slave, you configure it and forget it). And finally we use the send function.

What this send function does is take two buffers, one for sending and one for receiving, and use them. In our case, as we are masters, we start by writing using the send buffer (master_tx_buffer) and then we wait for the slave to send us as much data as the receiving buffer is big.

To understand each other, I’ll give you two examples.

- **Example 1:** if, as in the previous example, 10 data are to be sent, the first thing to be created is a sending buffer with the 10 data, and if 10 data are expected from the slave, another buffer of 10 data is created, then what is done is to send the write data first and then the master waits for the other 10 data that the slave has to send.

NOTE: if the master does not receive the data in a timely manner, it sets them to 0xFF.



Envío (Send)	Recepción (Receive)

- **Example 2:** if what is wanted is a write-only master, what must be done is to create a sending buffer with the data that is going to be sent to the slave, and cancel the receiving buffer. How? By putting a function like this.

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

This makes the SoC only send data.

To make it read-only, do the same thing, just cancel the write.

Now compile the code, remember to set the usual compiler variables.

Code

```
#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));

}
```