How to configure I2C on a SmartFusion2

Created by: David Rubio G.

Blog post: https://soceame.wordpress.com/2025/03/10/how-to-configure-i2c-on-a-smartfusion2/

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

GitHub: https://github.com/DRubioG

Last modification date: 10/03/25

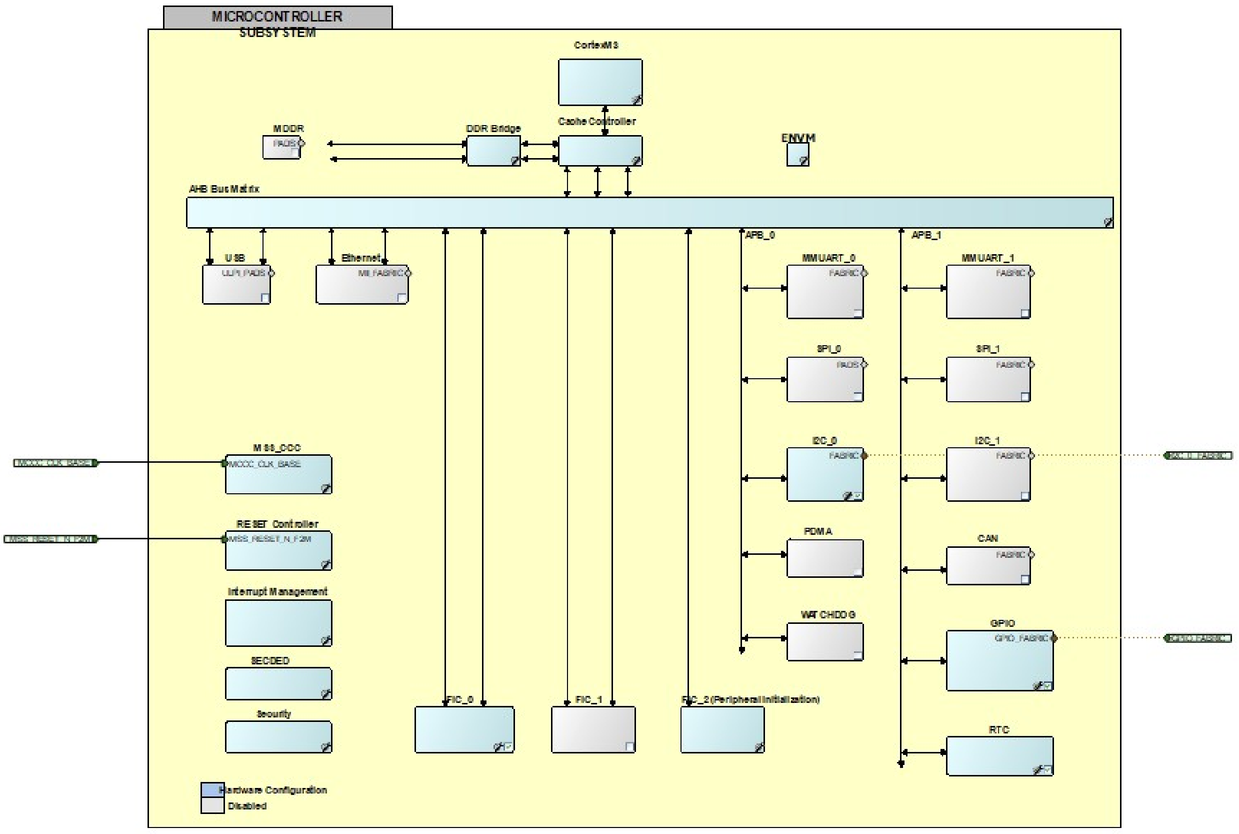
For this post we are going to base ourselves on these two previous posts.

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/

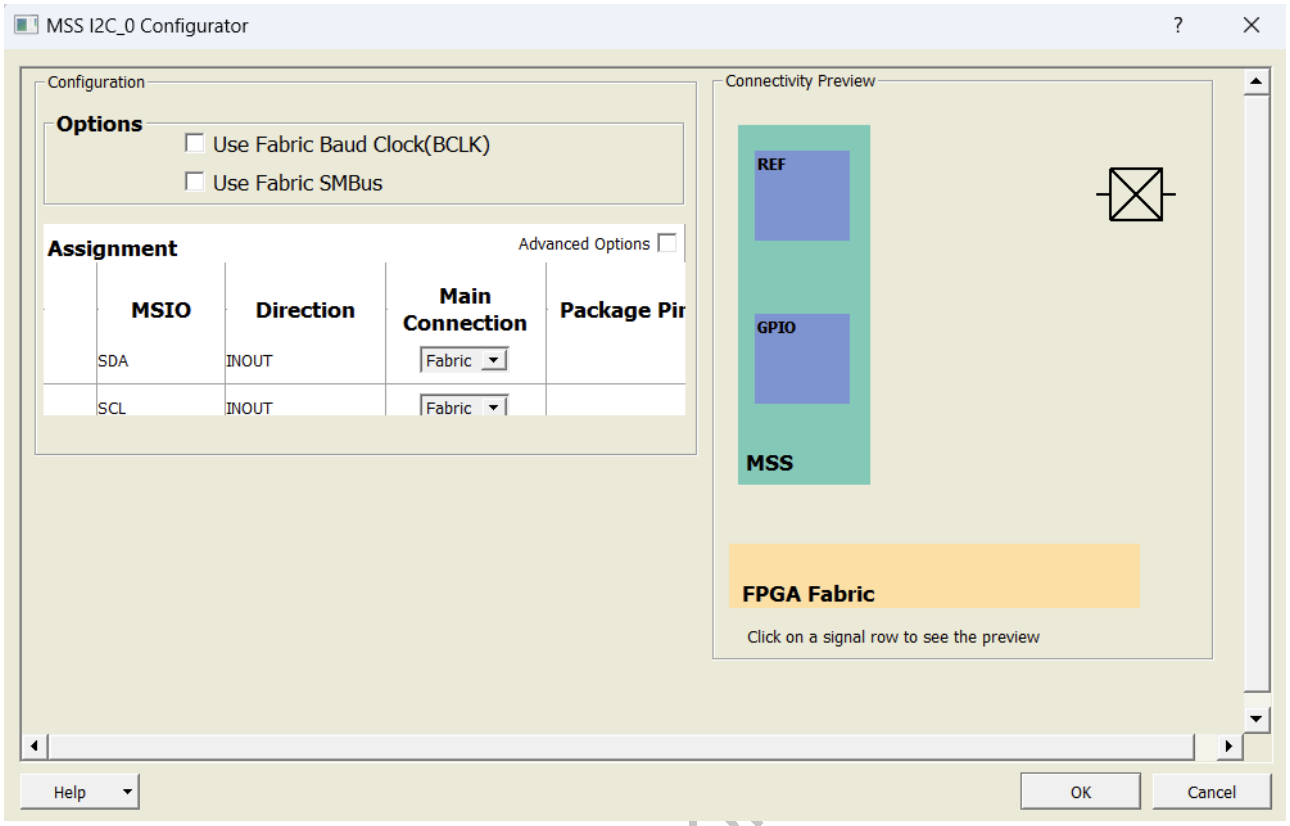
Project in Libero

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

Then, we select the I2C0 to use it as an output.



If we go into the I2C to see how it is configured, we can choose if we want to use any pin of the SoC or the specific pins for the I2C of the SoC. In our case we use any pin of the FPGA (*Fabric*).


MSS I2C_0 Configurator

Configuration

Options

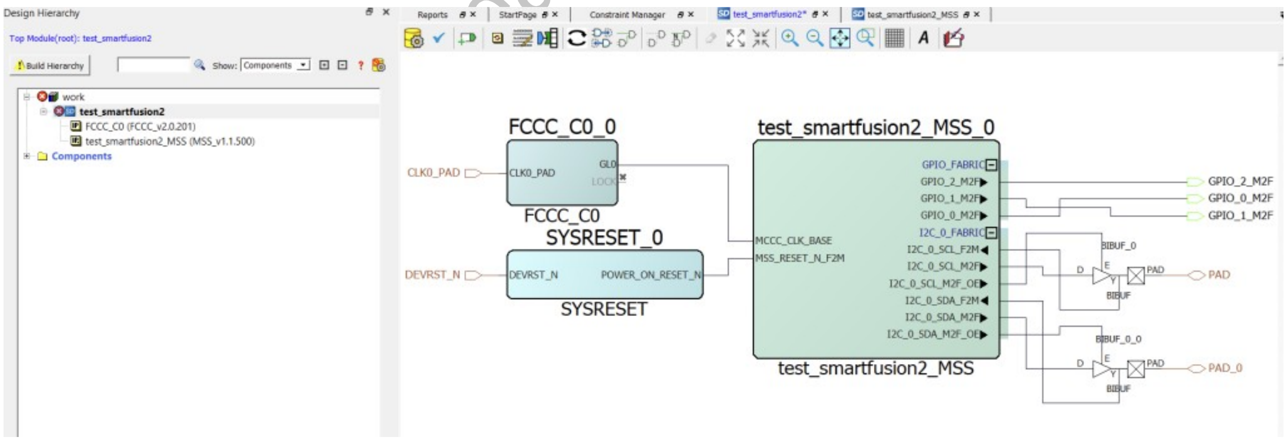
- [ ] Use Fabric Baud Clock(BCLK)
- [ ] Use Fabric SMBus

Assignment Advanced Options [ ]

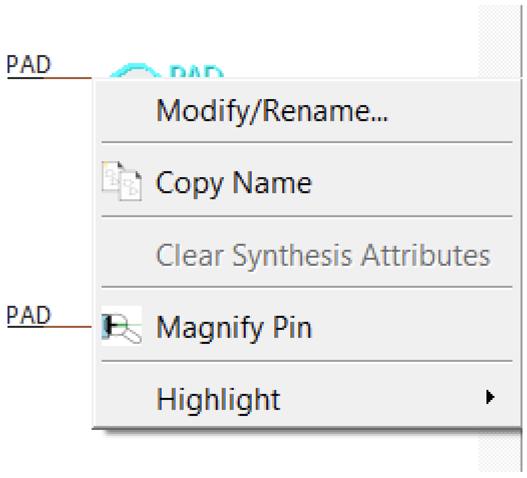
MSIO	Direction	Main Connection	Package Pir
SDA	INOUT	Fabric	
SCL	INOUT	Fabric	

Connectivity Preview

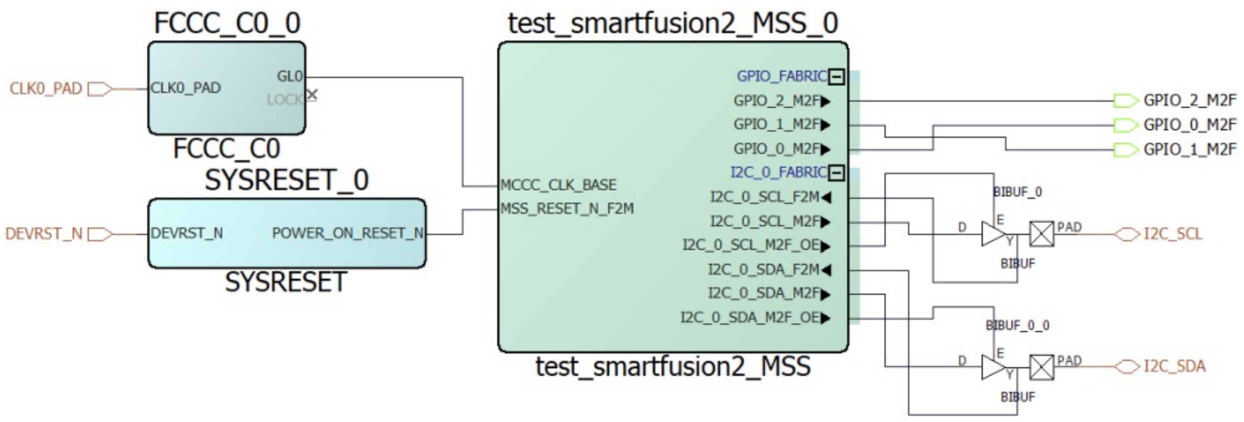
Once we have generated the I2C outputs of the core, we have to configure them. To configure the outputs, because the I2C is a bidirectional protocol, we have to use *BIBUF* blocks, which allow us to select the directionality of the pin with the OE output of the core. (The BIBUF block is in the Catalog)



Then what we do is change the name of the pin, to do this right click on the *Modify/Rename* output and choose the new name.

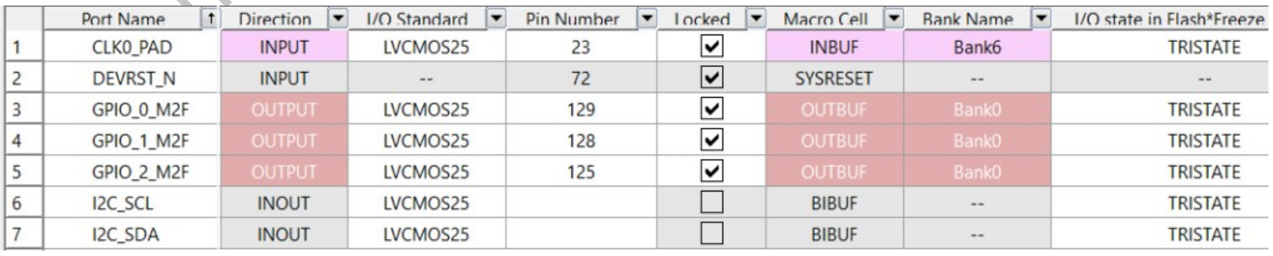


Once we have renamed the pins, it looks like this.



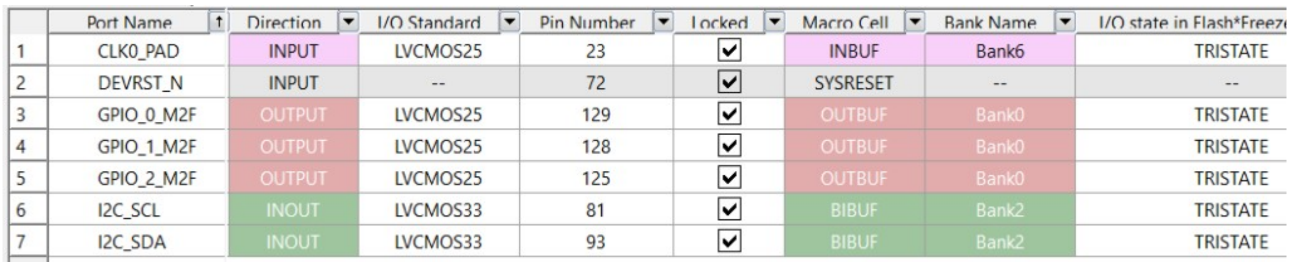
Now what we have to do is finish configuring the model, clicking on the yellow gear icon, and then on *Build Hierarchy* so that it adds the *BIBUFs* to the hierarchy. Then we synthesize the project, *remember to generate the Memory Map.*

After synthesizing we choose the new *constraints* for the pins. We open the *Edit for the I/O*.



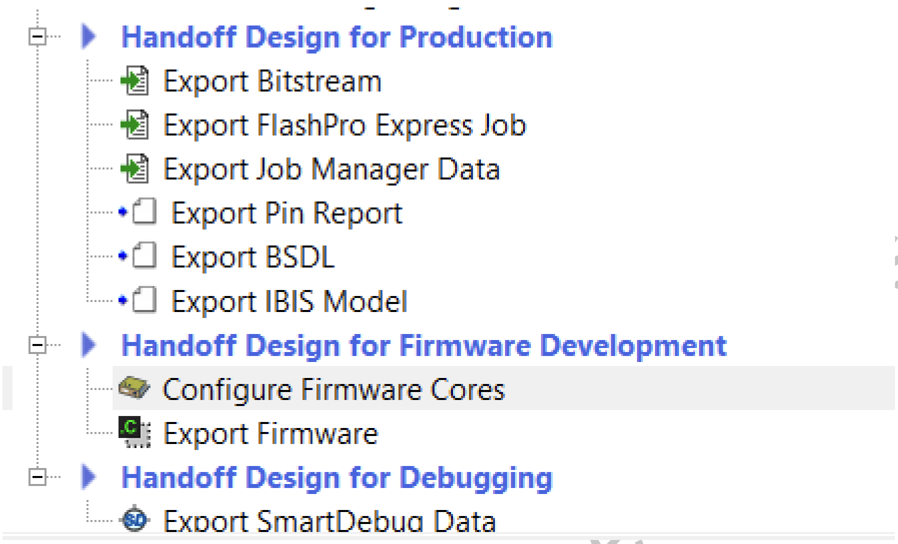
	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze
1	CLK0_PAD	INPUT	LVCMOS25	23	✓	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	✓	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	✓	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	✓	OUTBUF	Bank0	TRISTATE
5	GPIO_2_M2F	OUTPUT	LVCMOS25	125	✓	OUTBUF	Bank0	TRISTATE
6	I2C_SCL	INOUT	LVCMOS25		☐	BIBUF	--	TRISTATE
7	I2C_SDA	INOUT	LVCMOS25		☐	BIBUF	--	TRISTATE

Then, we choose the I2C pins.



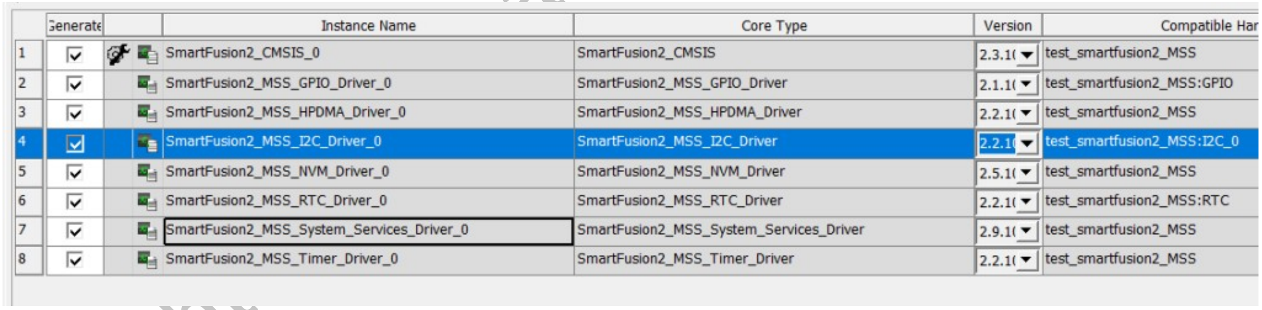
	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freez
1	CLK0_PAD	INPUT	LVCMOS25	23	✓	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	✓	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	✓	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	✓	OUTBUF	Bank0	TRISTATE
5	GPIO_2_M2F	OUTPUT	LVCMOS25	125	✓	OUTBUF	Bank0	TRISTATE
6	I2C_SCL	INOUT	LVCMOS33	81	✓	BIBUF	Bank2	TRISTATE
7	I2C_SDA	INOUT	LVCMOS33	93	✓	BIBUF	Bank2	TRISTATE

Then, we finish generating the project bitstream. When we have it, we program the SoC. And now we extract the project drivers, to do this we click on *Configure Firmware Cores*.



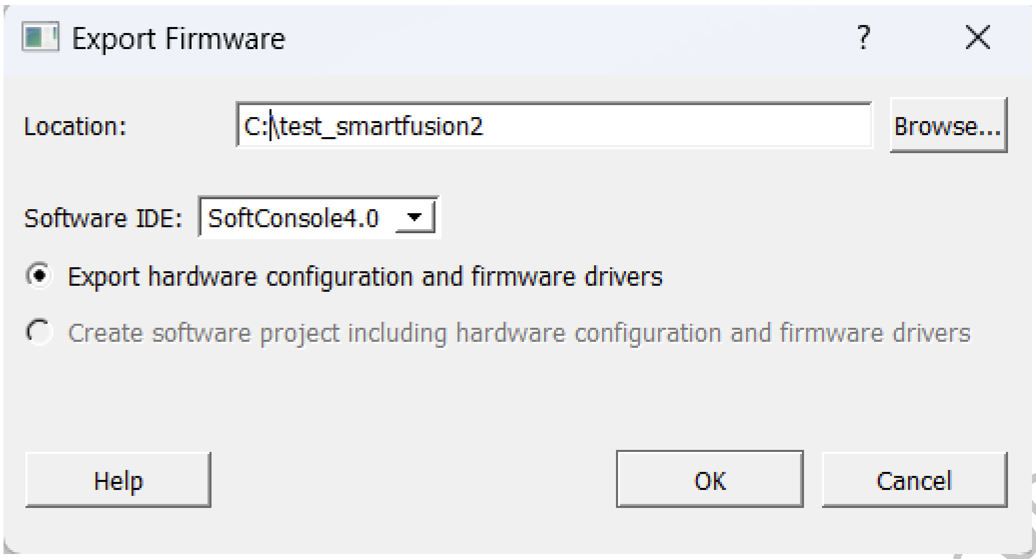
- **Handoff Design for Production**
  - Export Bitstream
  - Export FlashPro Express Job
  - Export Job Manager Data
  - Export Pin Report
  - Export BSDL
  - Export IBIS Model
- **Handoff Design for Firmware Development**
  - Configure Firmware Cores
  - Export Firmware
- **Handoff Design for Debugging**
  - Export SmartDebug Data

We have to have the I2C drivers selected.



	Generate	Instance Name	Core Type	Version	Compatible Har
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_I2C_Driver_0	SmartFusion2_MSS_I2C_Driver	2.2.1(	test_smartfusion2_MSS:I2C_0
5	✓	SmartFusion2_MSS_NVM_Driver_0	SmartFusion2_MSS_NVM_Driver	2.5.1(	test_smartfusion2_MSS
6	✓	SmartFusion2_MSS_RTC_Driver_0	SmartFusion2_MSS_RTC_Driver	2.2.1(	test_smartfusion2_MSS:RTC
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

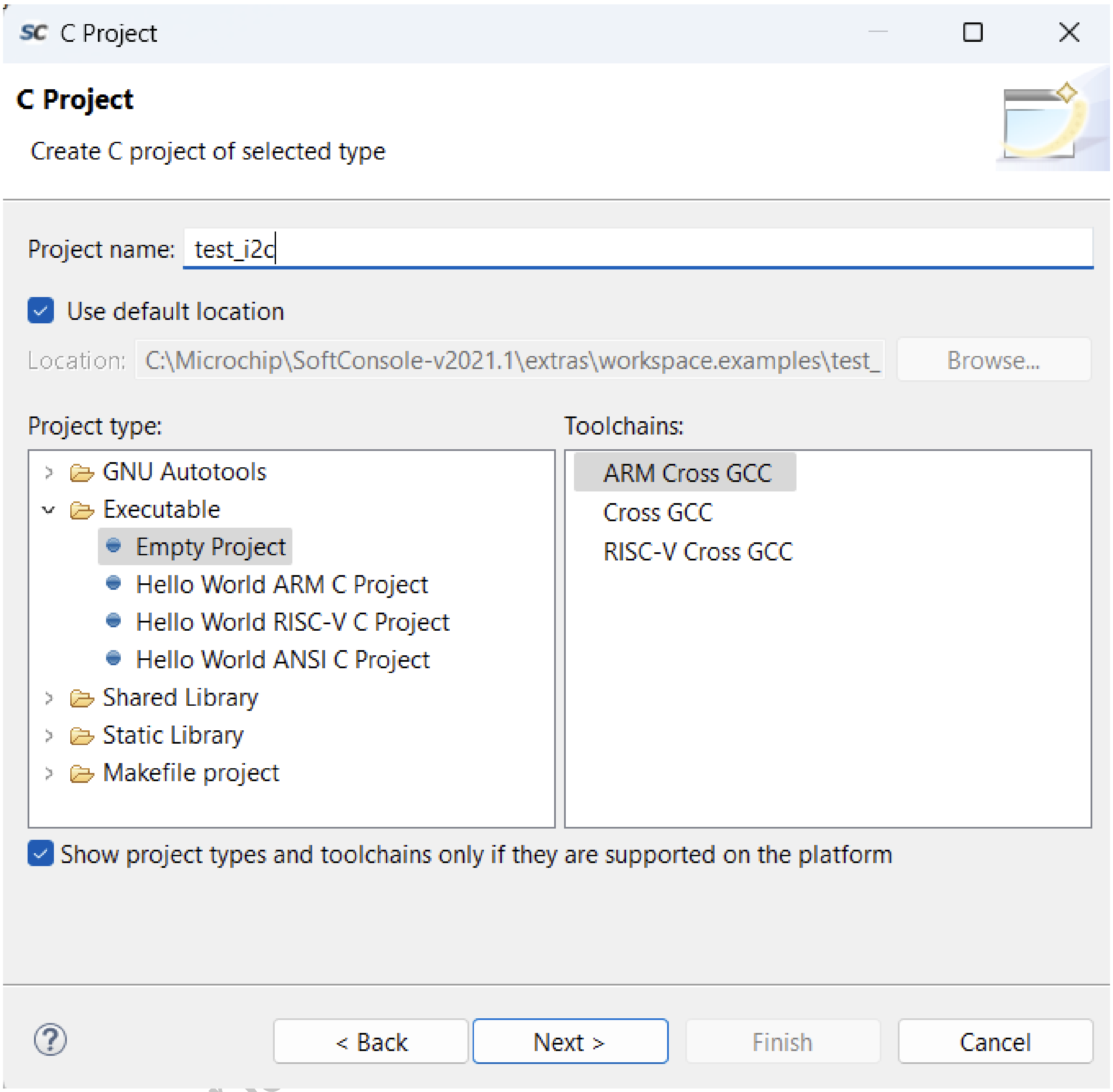
Software IDE: SoftConsole4.0

[✓ Export hardware configuration and firmware drivers]

Create software project including hardware configuration and firmware drivers

Project in SoftConsole

The first thing we do is create an empty C project



Project name: test_i2c

[✓] Use default location

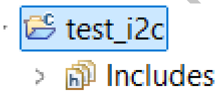
Location: C:\Microchip\SoftConsole-v2021.1\extras\workspace.examples\test_

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

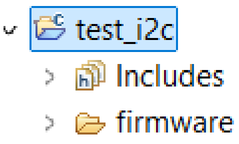
Toolchains:
- ARM Cross GCC
- Cross GCC
- RISC-V Cross GCC

[✓] Show project types and toolchains only if they are supported on the platform

When the project is created, we have the folder with the *includes*.



Now we add the drivers that we just exported and create a main.c file.



Now we have two options for creating the I2C, either as a master or as a slave.

- **master**

If we define it as a master, the first thing we have to do is configure the I2C as a master (*MSS_I2C_init*) and then tell it to write (*MSS_I2C_write*) the data that is in the write buffer (*tx_buffer*). There is also a buffer that is the write buffer that the slaves use to send data to the master.

NOTE: it is important to understand that the master of a communication has to tell the slaves when it is going to read, how much data it wants to read, in this example 16 data.

```
#include "firmware/drivers/mss_i2c/mss_i2c.h"

#define I2C_DUMMY_ADDR   0x10u
#define DATA_LENGTH      16u

uint8_t  tx_buffer[DATA_LENGTH];
uint8_t  write_length = DATA_LENGTH;

void main(){
    uint8_t  target_slave_addr = 0x12;
    mss_i2c_status_t status;

    // Initialize MSS I2C peripheral
    MSS_I2C_init( &g_mss_i2c0, I2C_DUMMY_ADDR, MSS_I2C_PCLK_DIV_256 );

    // Write data to slave.
    MSS_I2C_write( &g_mss_i2c0, target_slave_addr, tx_buffer, write_length,
                   MSS_I2C_RELEASE_BUS );

    // Wait for completion and record the outcome
    status = MSS_I2C_wait_complete( &g_mss_i2c0, MSS_I2C_NO_TIMEOUT );

}
```

- **slave**

As a slave, it is necessary to understand that the slave of an I2C communication is totally reactive, that is, it will not emit data for its own interest, but rather it has to be the master that requests the data.

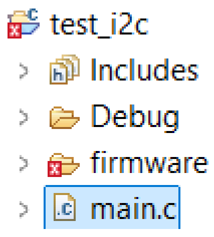
The I2C of the SmartFusion2 is composed of **two buffers**, one for reading, which is the one that the master reads when it sends the reading order, and one for writing, which is the one that the master uses to communicate with the slave.

In addition, as the I2C is configured in the SmartFusion2, an interruption is not generated when new data arrives, so the only way to know if data has arrived is by reading the writing buffer on a recurring basis.

```
#include "firmware/drivers/mss_i2c/mss_i2c.h"

#define SLAVE_SER_ADDR         0x10u
#define SLAVE_TX_BUFFER_SIZE   10u
#define SLAVE_RX_BUFFER_SIZE 10u
```

```
uint8_t g_slave_tx_buffer[SLAVE_TX_BUFFER_SIZE] = { 1, 2, 3, 4, 5,
                                                    6, 7, 8, 9, 10 };
uint8_t g_slave_rx_buffer[SLAVE_RX_BUFFER_SIZE];


void main( void )
{
    // Initialize the MSS I2C driver with its I2C serial address and serial
    // clock divider.
    MSS_I2C_init( &g_mss_i2c0, SLAVE_SER_ADDR, MSS_I2C_PCLK_DIV_256 );

    MSS_I2C_set_slave_tx_buffer( &g_mss_i2c0, g_slave_tx_buffer,
                                 sizeof(g_slave_tx_buffer) );


    MSS_I2C_set_slave_rx_buffer( &g_mss_i2c0, g_slave_rx_buffer,
                                         sizeof(g_slave_rx_buffer) );

    MSS_I2C_enable_slave( &g_mss_i2c0 );
}
```

If you try to compile the project it will fail, because you have to configure the compiler directives that SoftConsole needs, and they are always the same.



These compilation directives are defined in the previous entries.