How to create a project for a SmartFusion2 board

Created by: David Rubio G.

Blog post: https://soceame.wordpress.com/2025/03/09/how-to-create-a-project-for-a-smartfusion2-board/

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

GitHub: https://github.com/DRubioG

Last modification date: 09/03/25

For this project, this other post is used as a base to go faster.

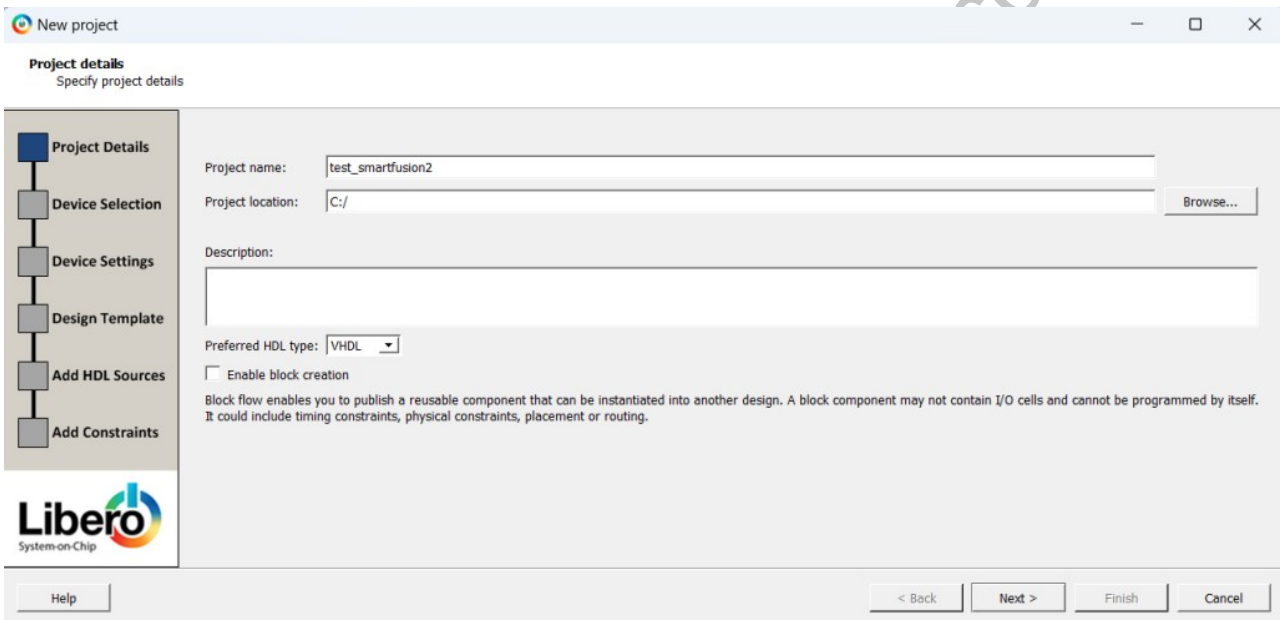
https://soceame.wordpress.com/2025/03/09/basic-project-with-libero/

In this entry I will explain how to create a project for a SmartFusion2 using both Libero and the SoftConsole.

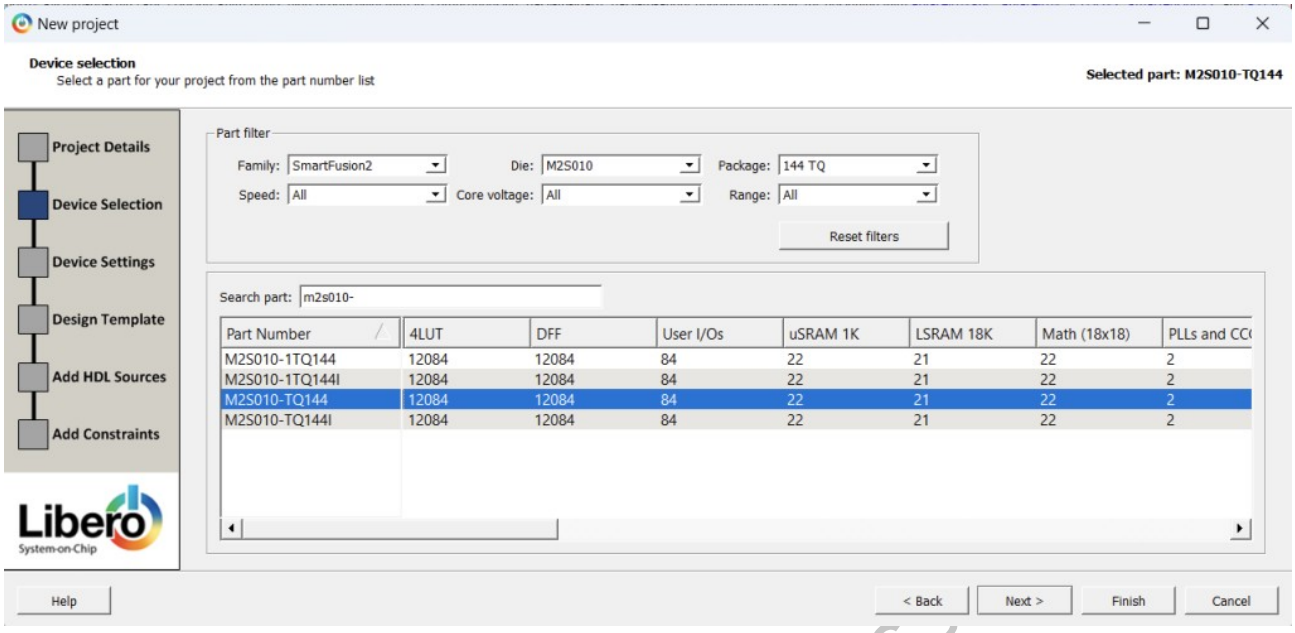
To do this, the project is very simple, we will configure some GPIOs to turn on and off some LEDs from the SoftConsole.

Libero project

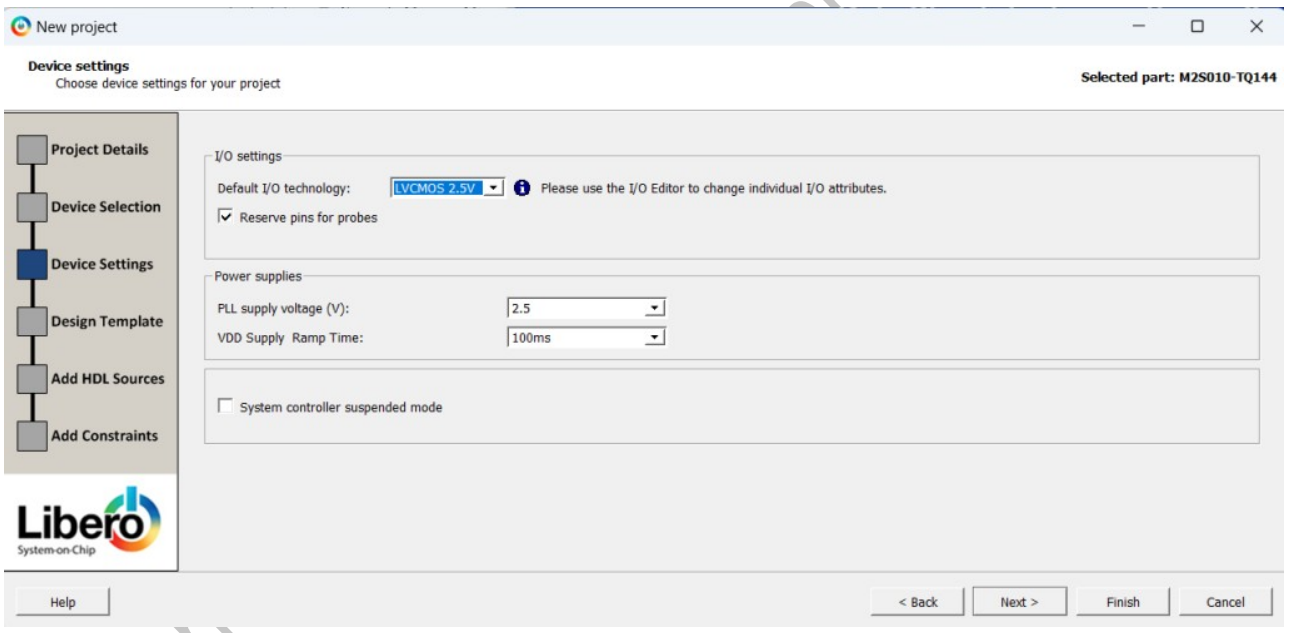
The first thing is to create a project in Libero.



Then we choose a SmartFusion2 with which we are going to work.

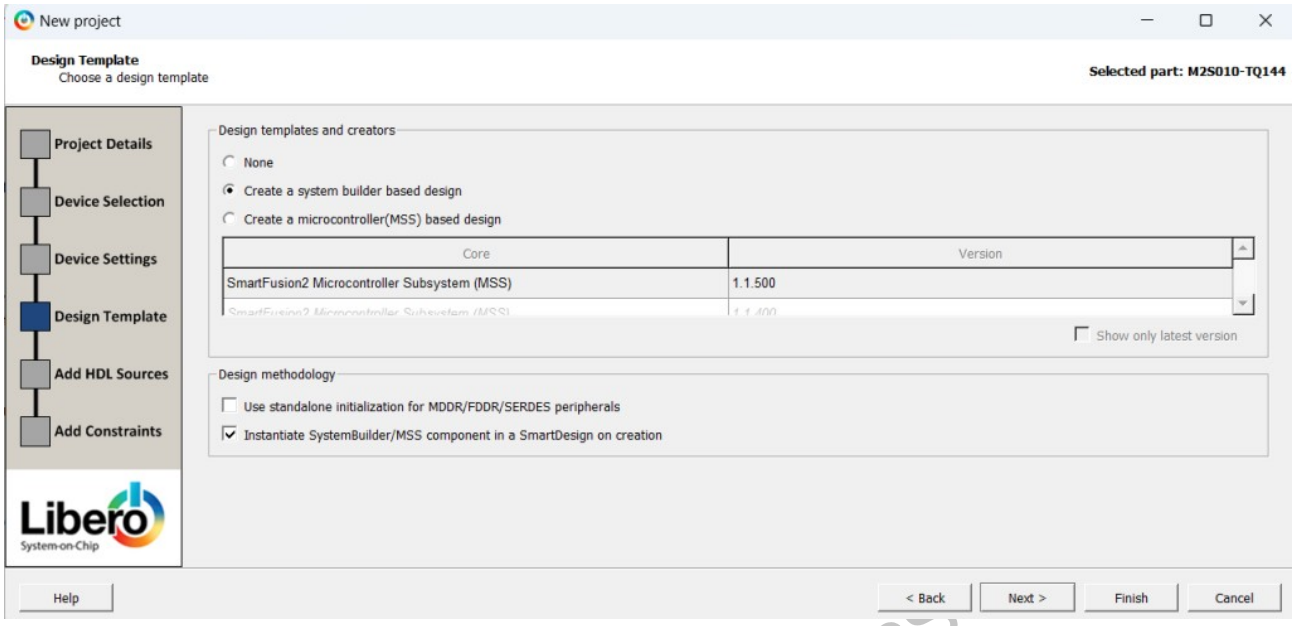


The next step is the selection of the default voltage of the pins.

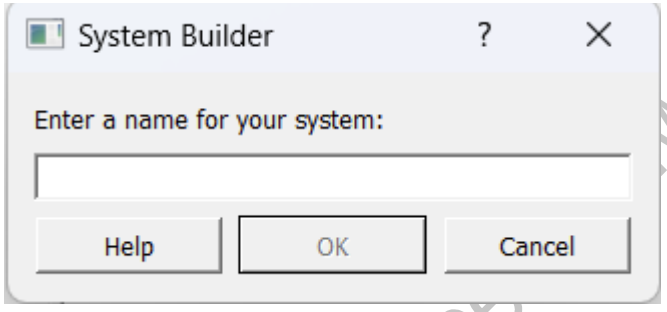


The next step is the important one, because it will decide how we are going to work with the SoC. I will describe the two ways there are:

- This first one will ask us how we want to activate the microcontroller.

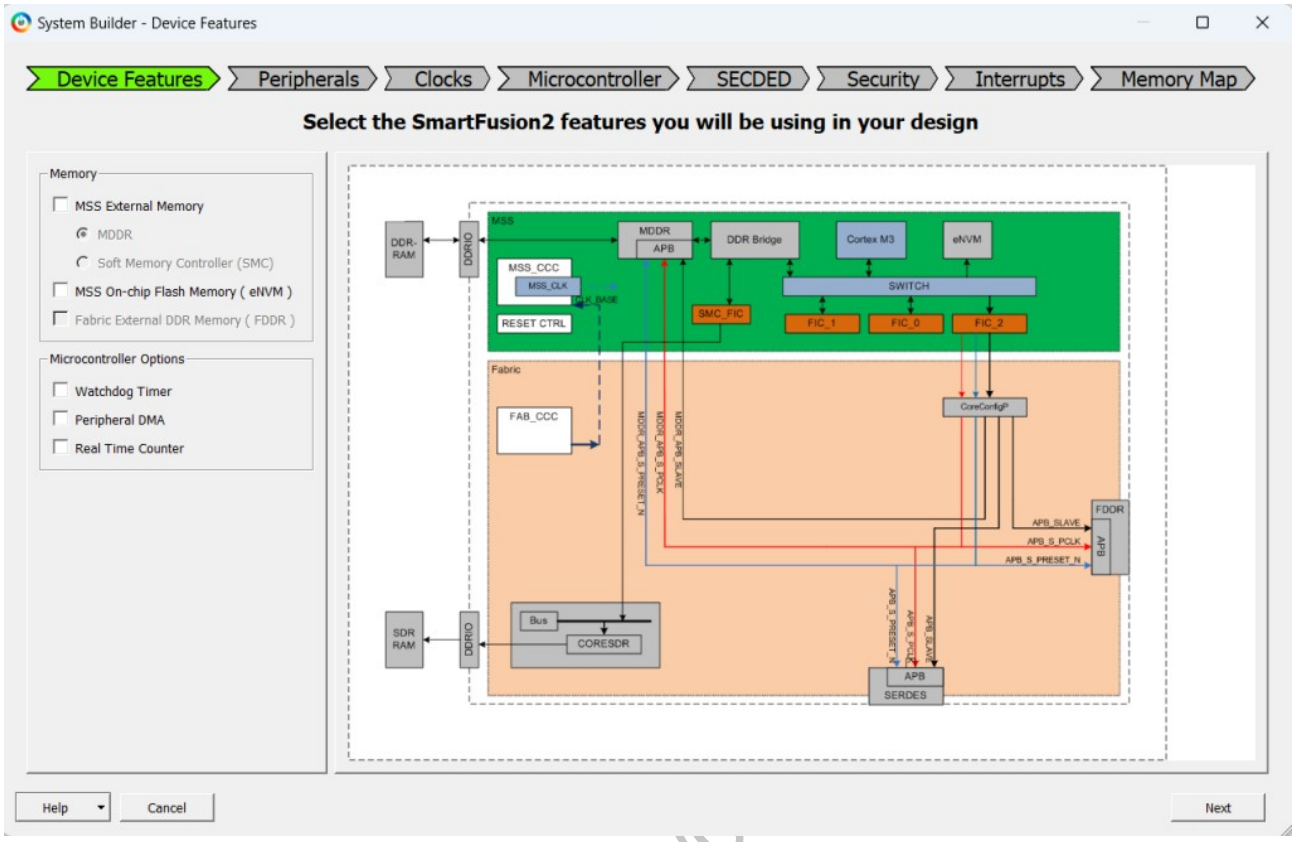


Now we can tell Libero to finish the configuration. When the configuration is finished, it will ask us for a name for the System Builder.

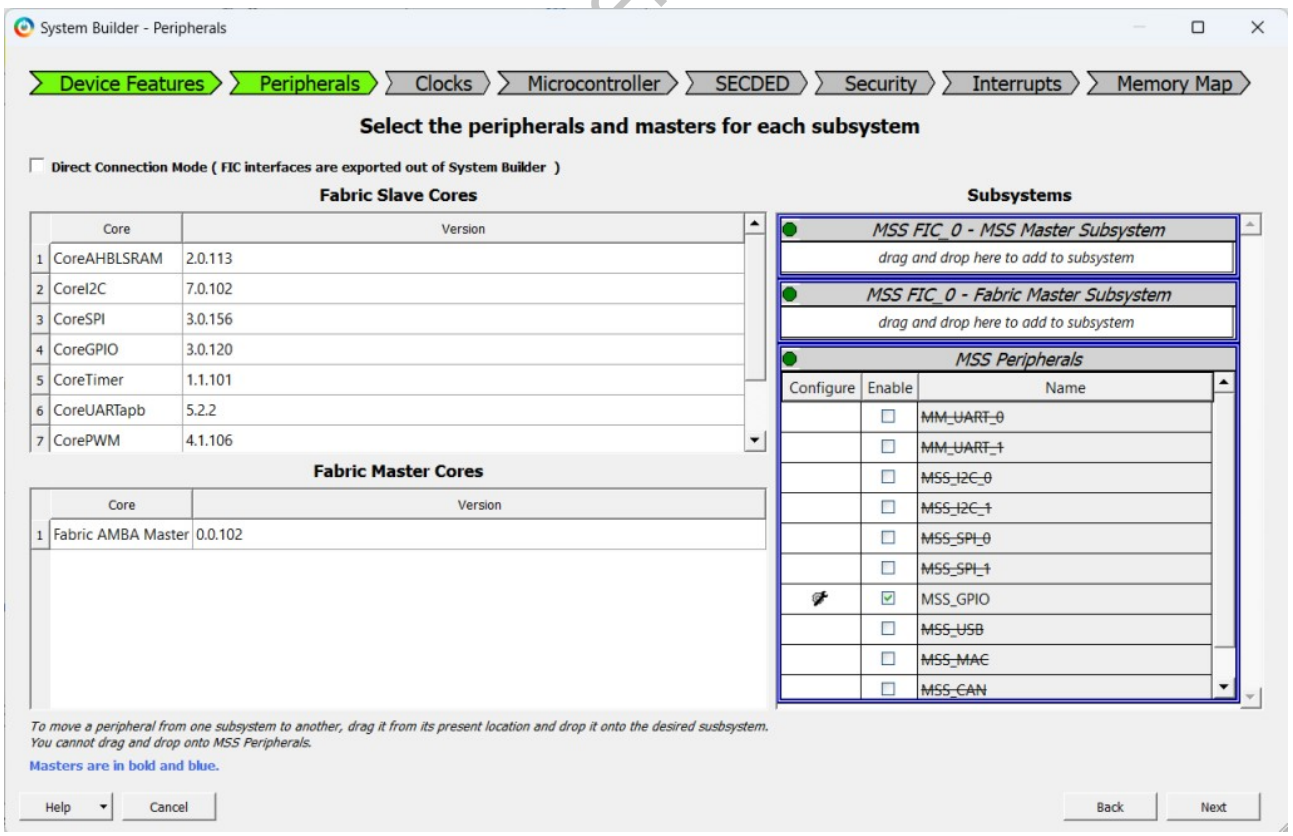


When it gives it a name, the activation begins. A tab opens asking us what things we want to activate.

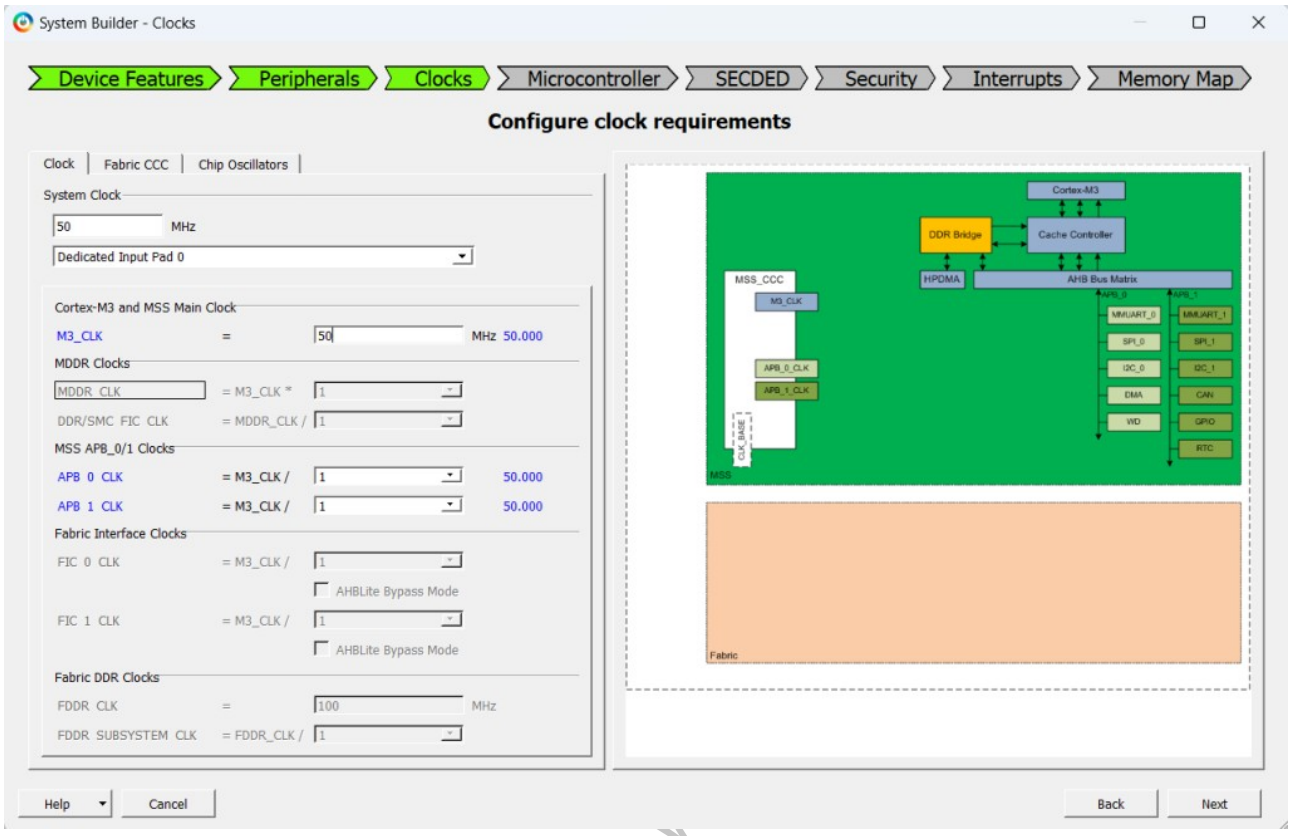
https://soceame.wordpress.com/2025/03/09/how-to-create-a-project-for-a-smartfusion2-board/



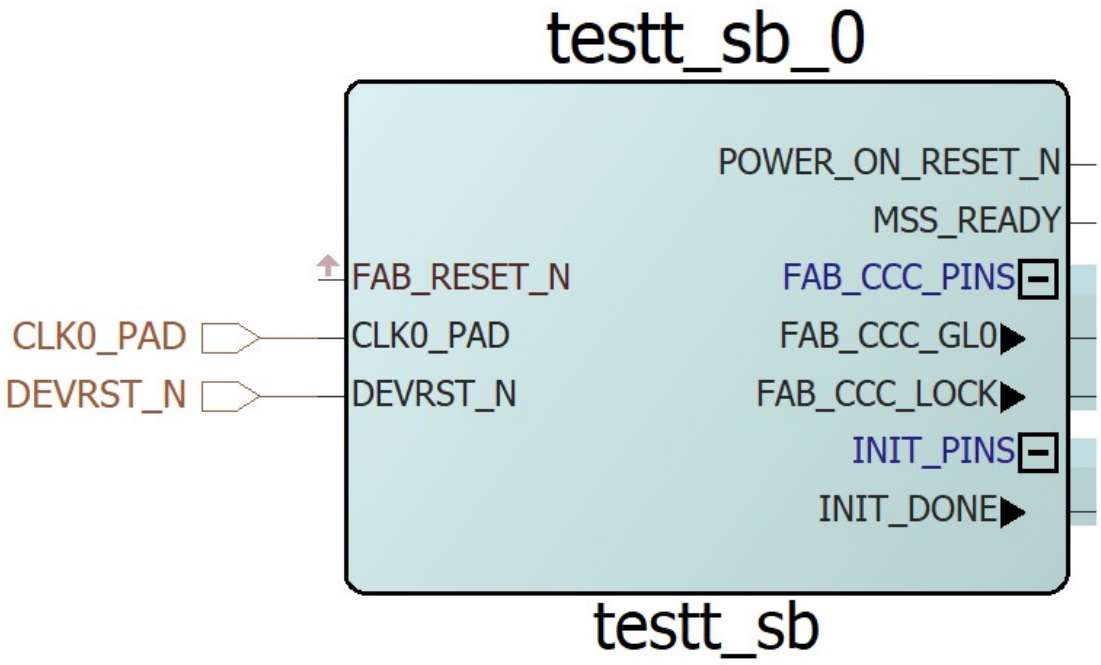
Then the peripherals we want to activate.



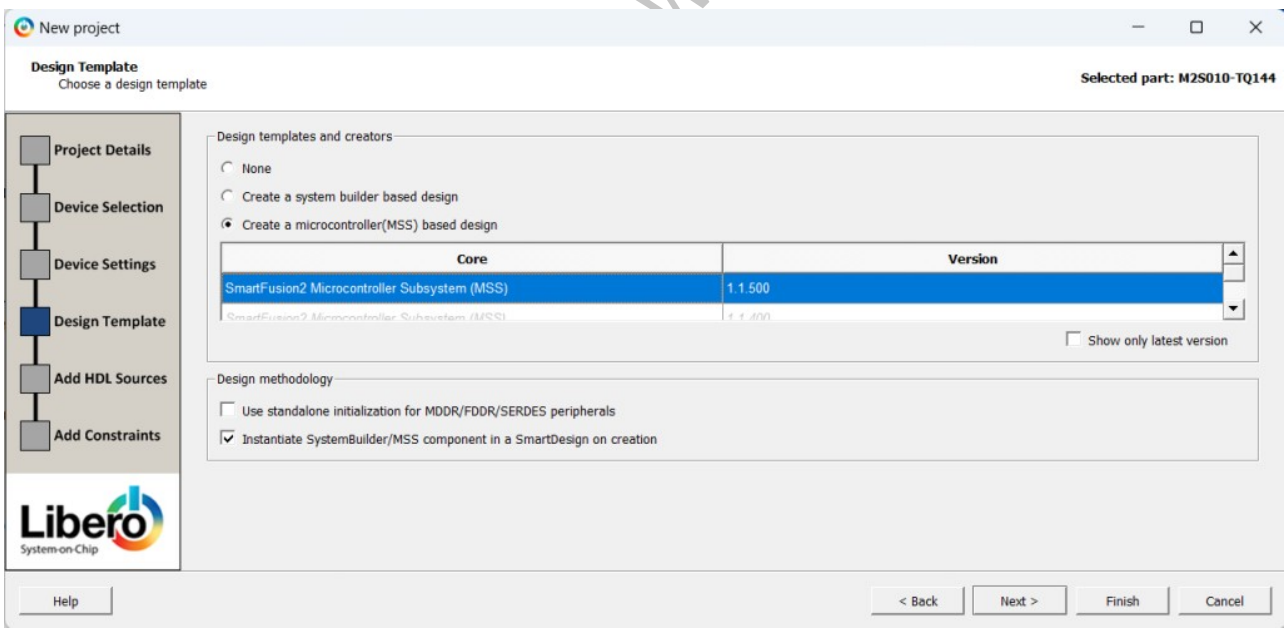
The next thing it asks for are the clocks, the board I use has a 50MHz clock.



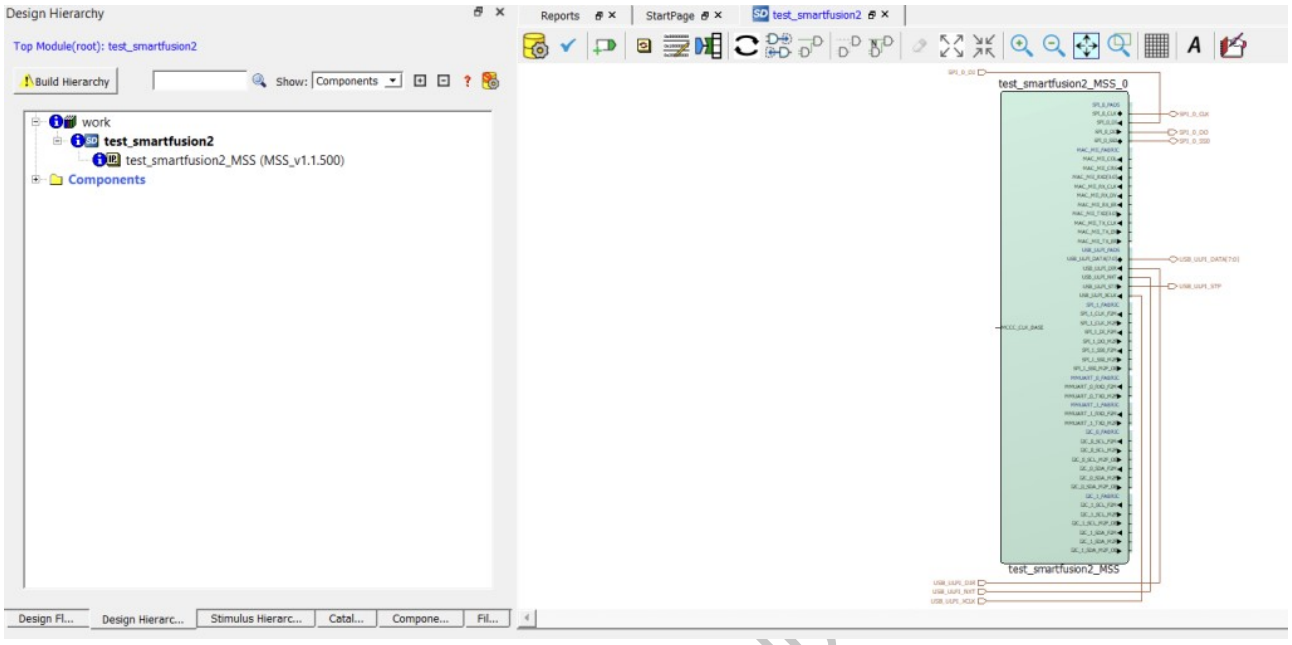
The rest of the configuration is unnecessary for the application I want to create. Then, in Libero a block like the following is created, which when opened we see that it takes us to the configuration tabs.



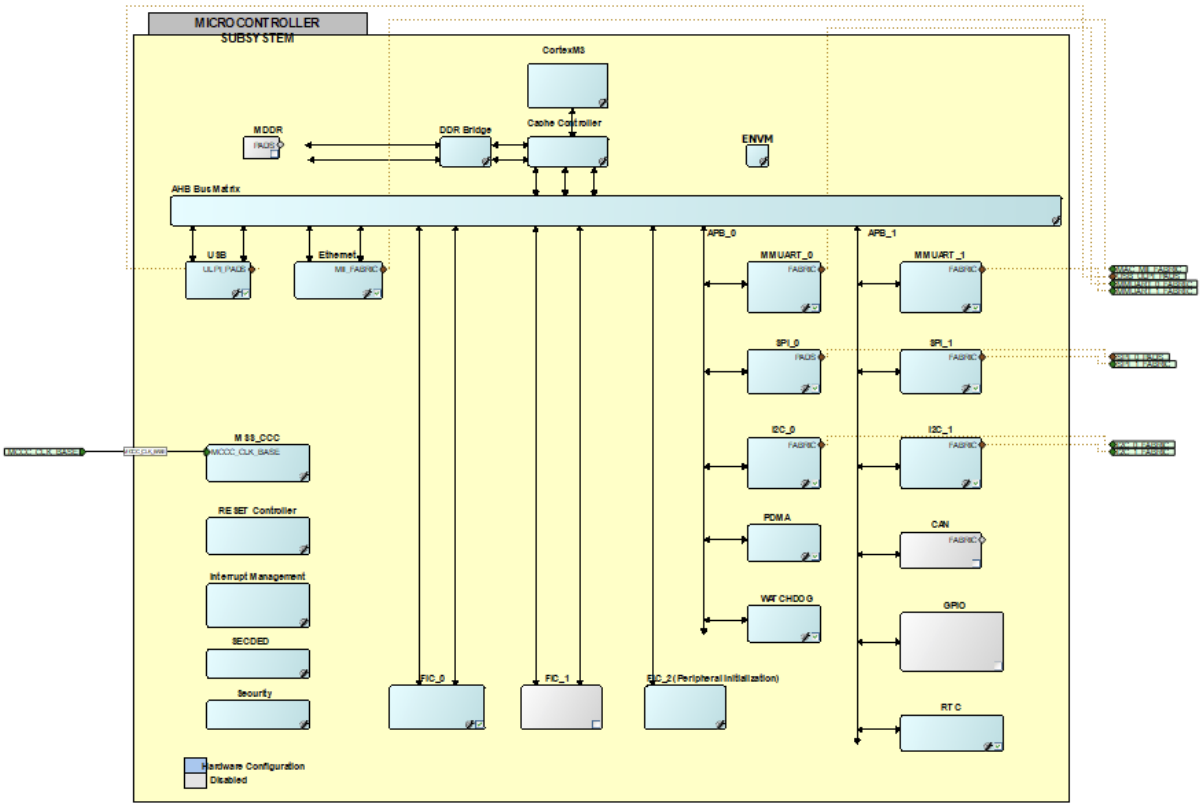
- This second way is going to create a microcontroller directly in Libero with all the peripherals activated in a SmartDesign. **(I recommend this second way)**



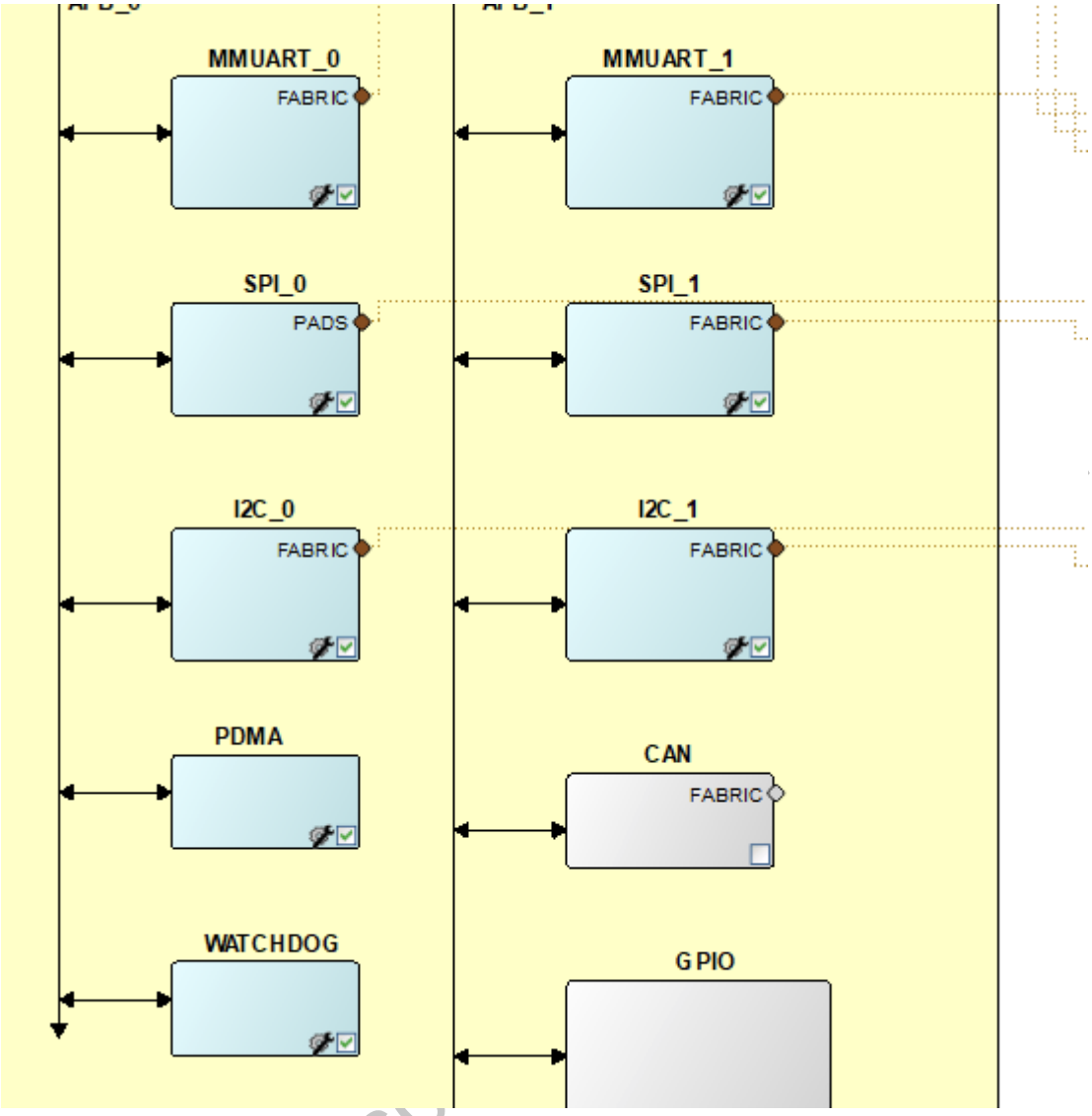
Now we can click on finish the configuration, and what it does is automatically create a block in a Libero SmartDesign.



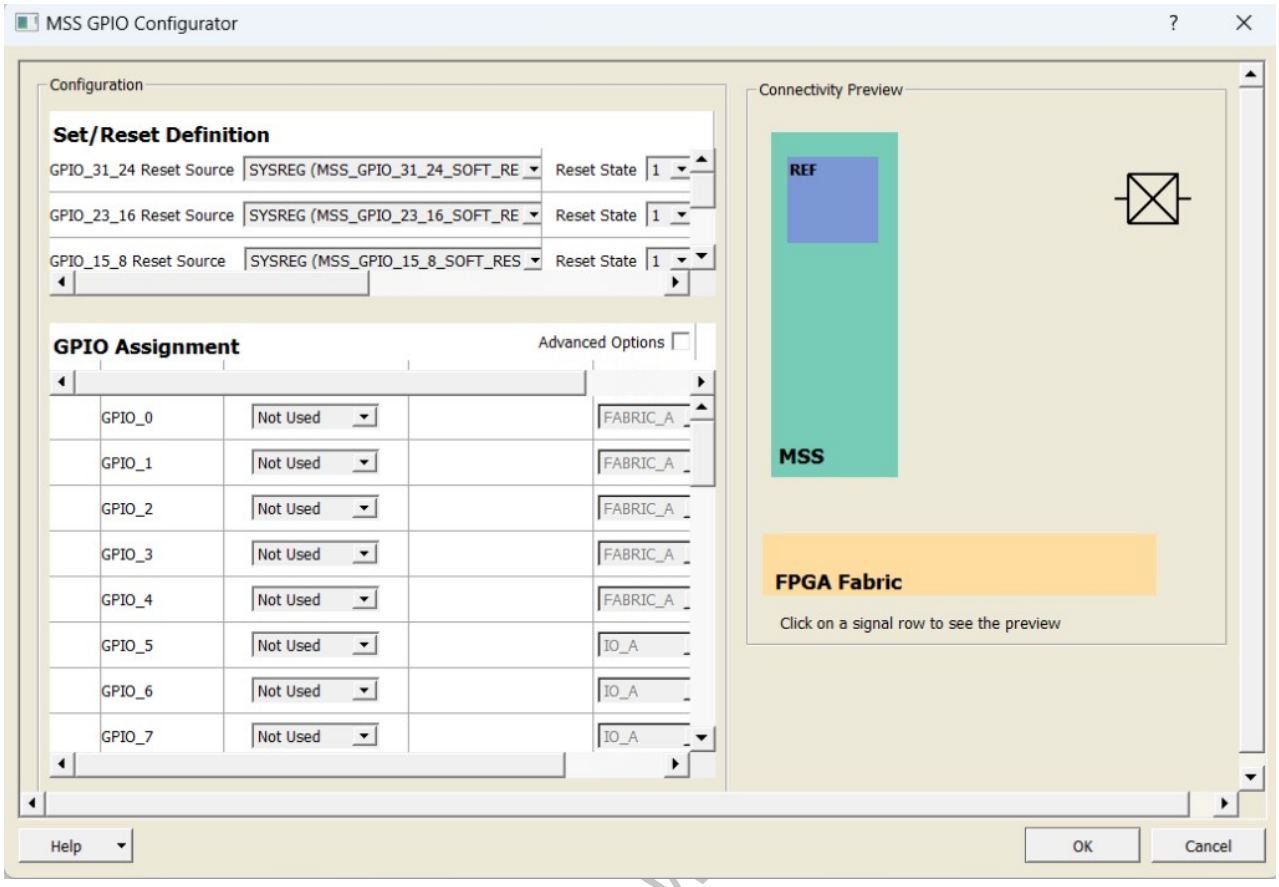
When we open it we have a tab like the following with almost all the peripherals activated.



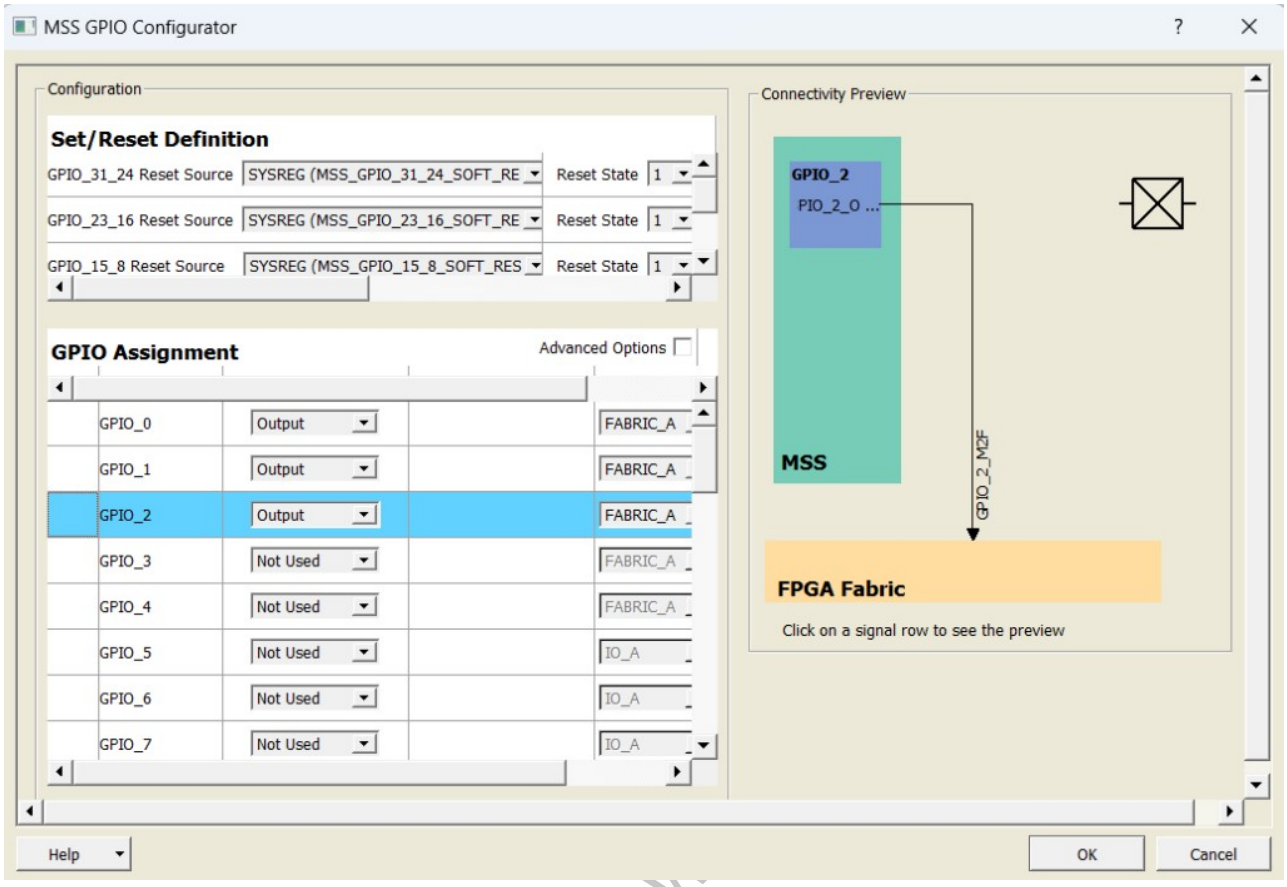
If we do not zoom in we can see that it has 2 UART, 2 I2C, 2 SPI, etc.



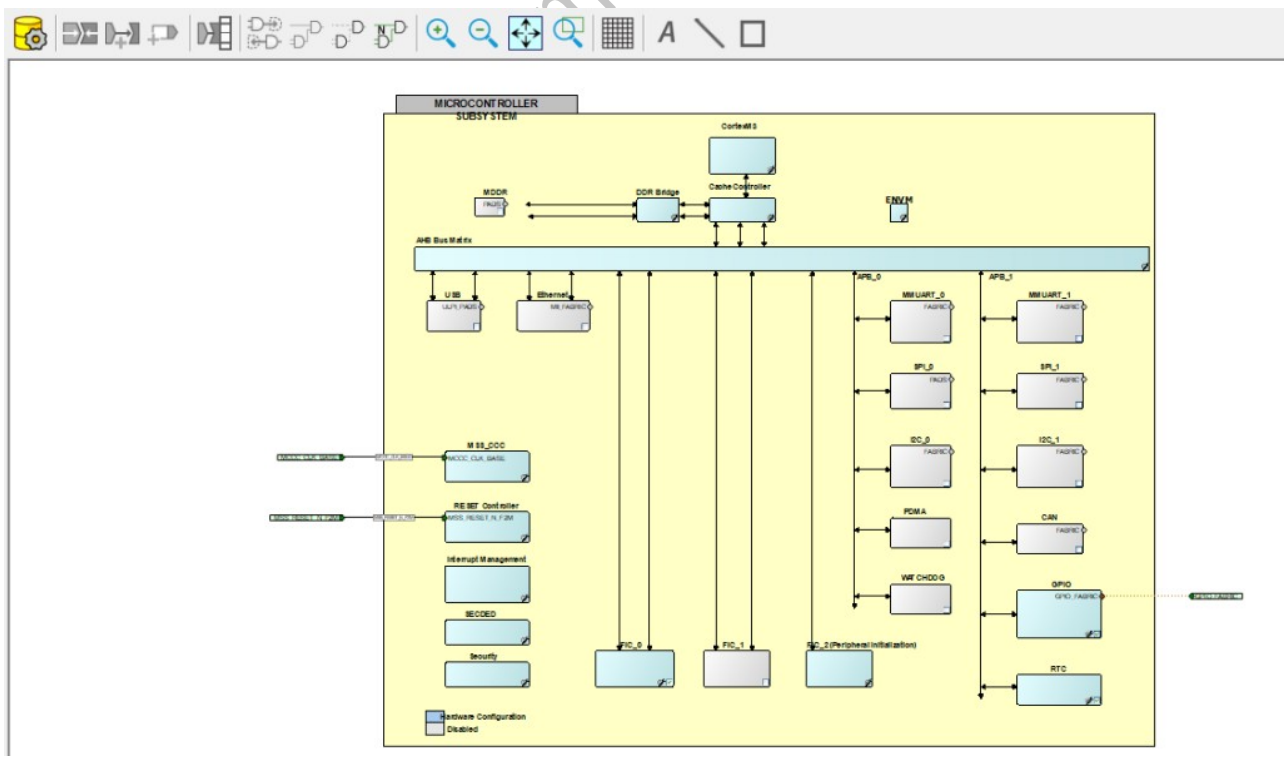
In our case we want to configure only the GPIOs, so we go into the GPIOs block which is disabled when we enter.



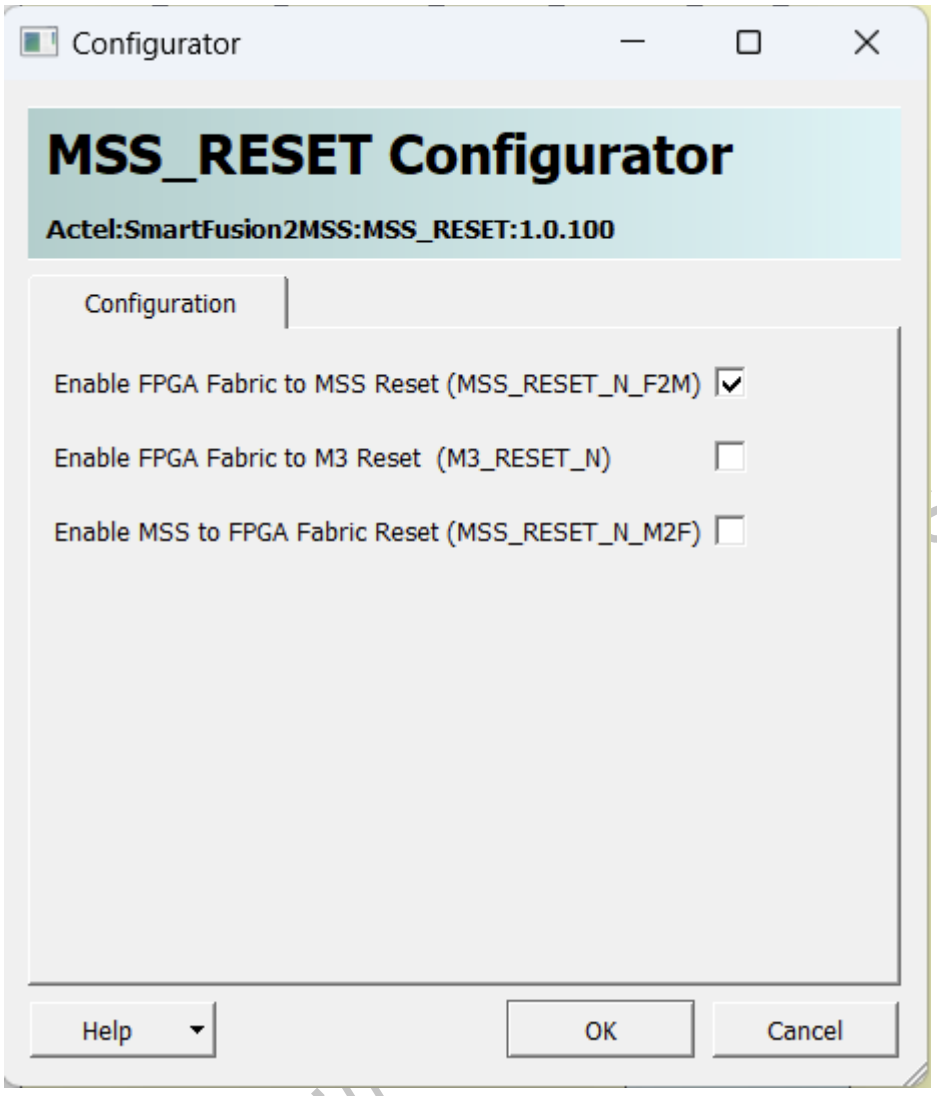
And we activate the first three GPIOs as outputs and to the FABRIC *(FABRIC means that they go to the programmable logic of the FPGA, so you can select the pin through which you want to output the GPIO. The other option is to take that output to a specific pin of the SoC, if available).*



Before leaving we deactivate everything, but we leave the clock and reset activated.

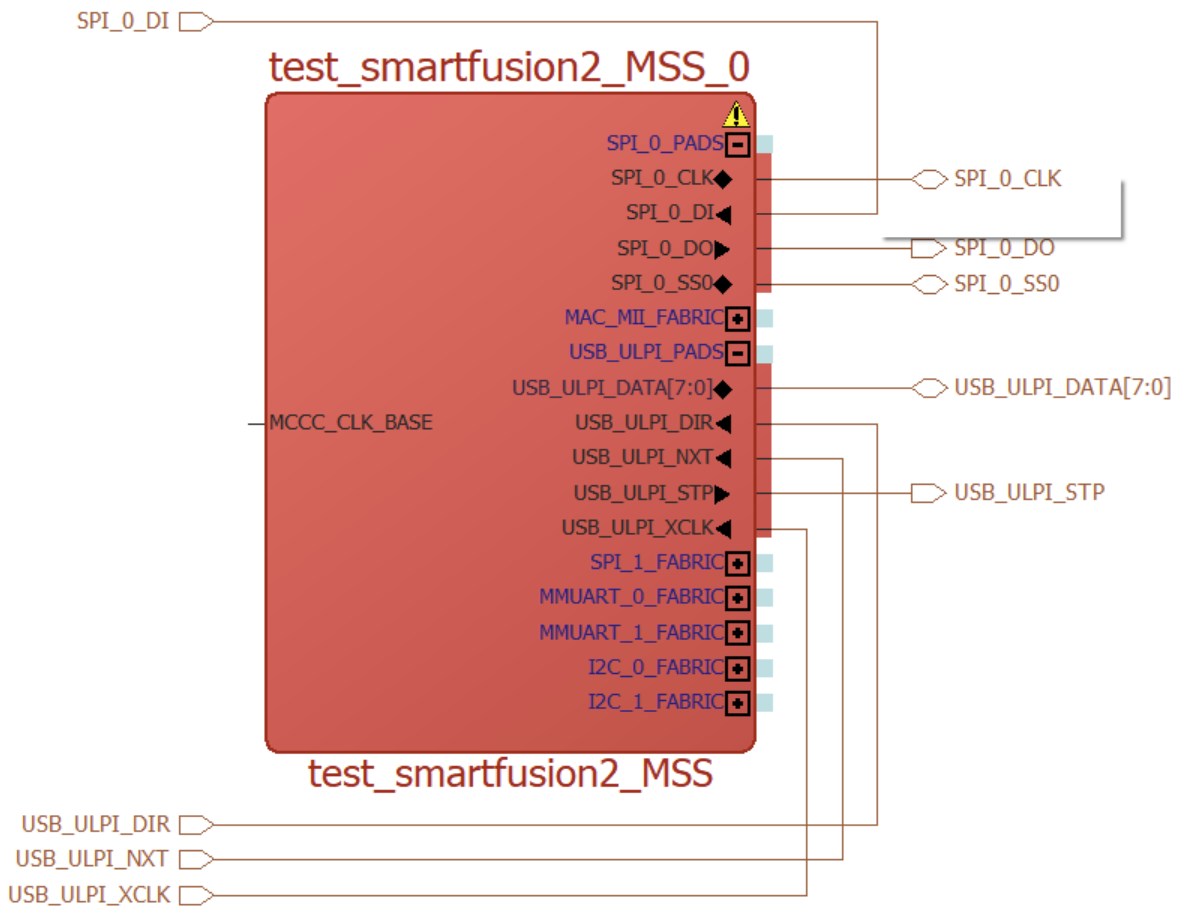


To activate the reset we choose the option that the reset goes from the FPGA to the microcontroller *(FABRIC to MSS)*.

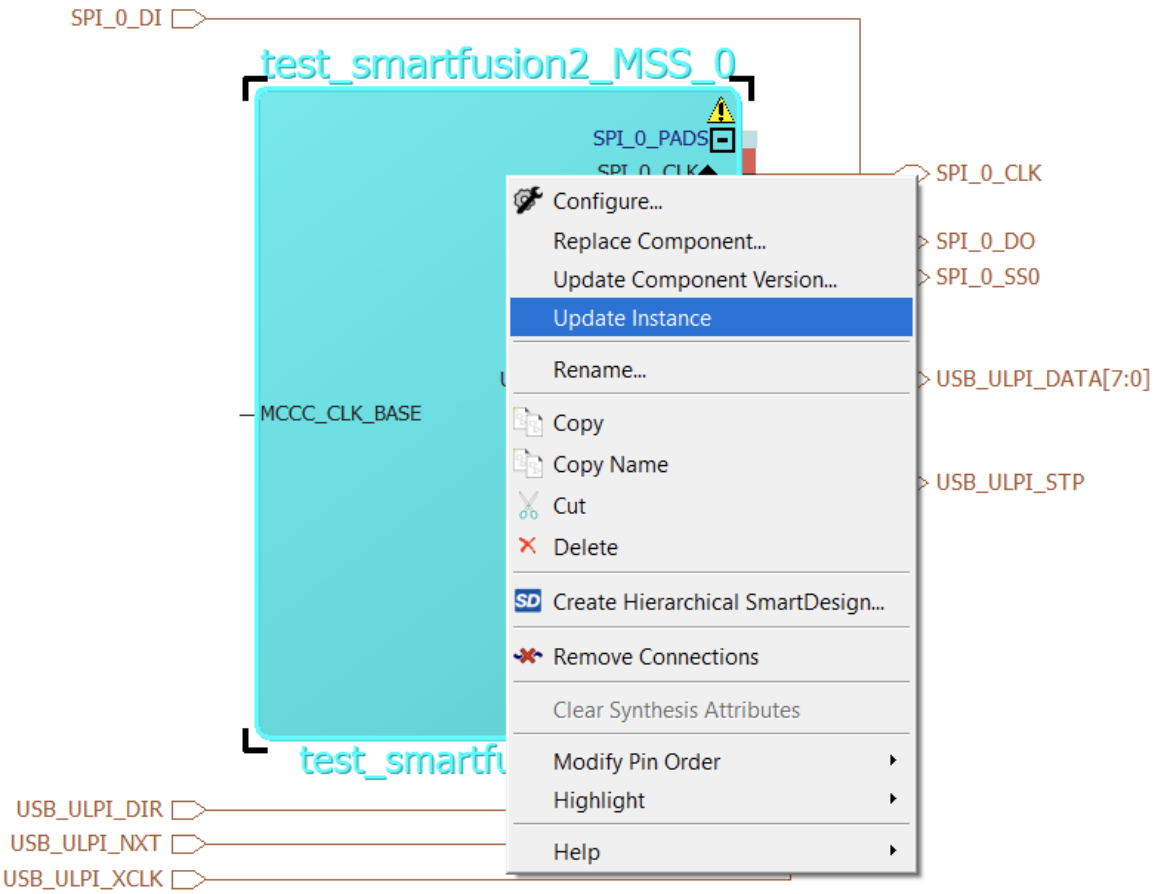


Once we have everything configured we click on the yellow symbol with the gear.

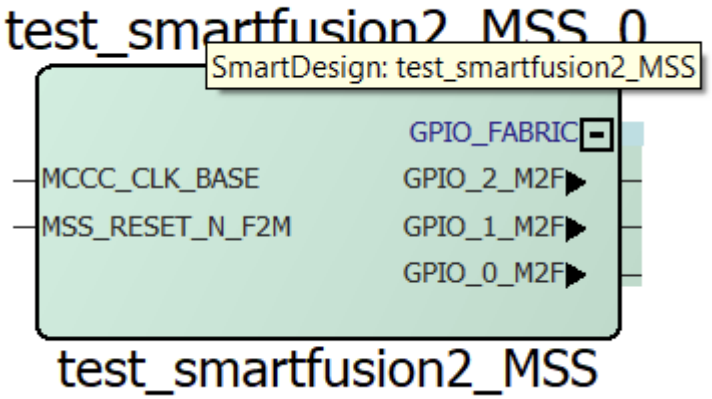
If we return to Libero we see the block marked in red.



To update the block we right click on it and click *Update Instance*.

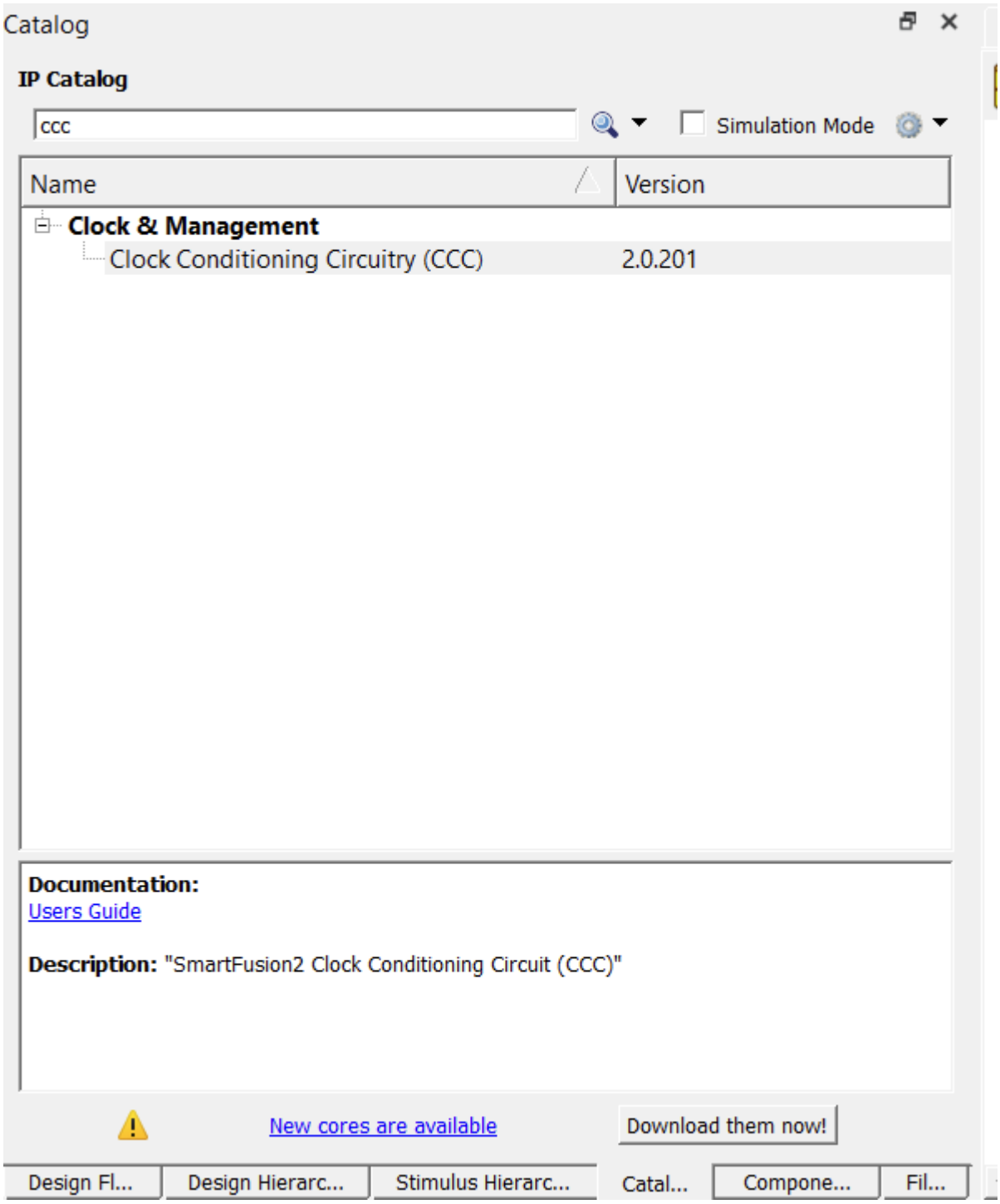


When it is updated it looks something like this.

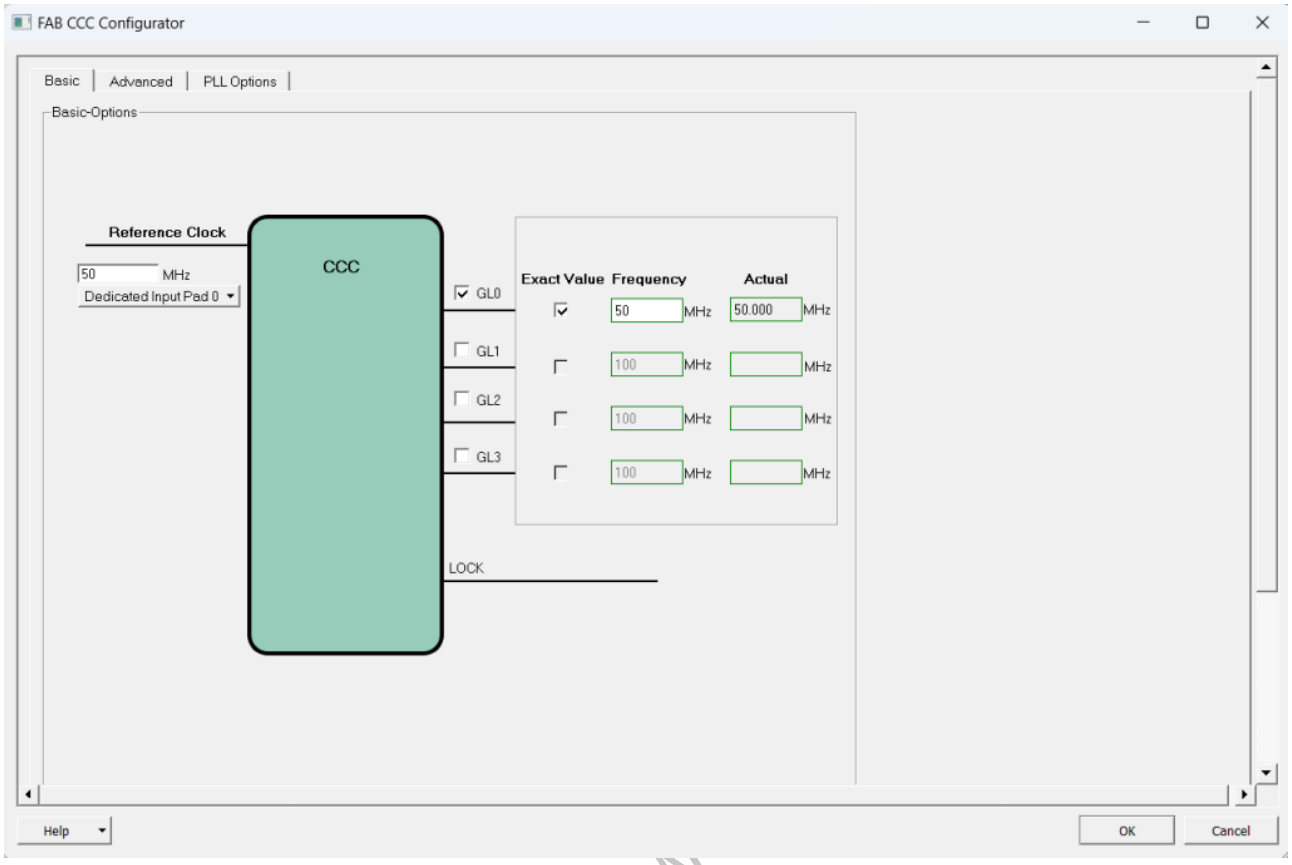


Now we need the clock and the reset.

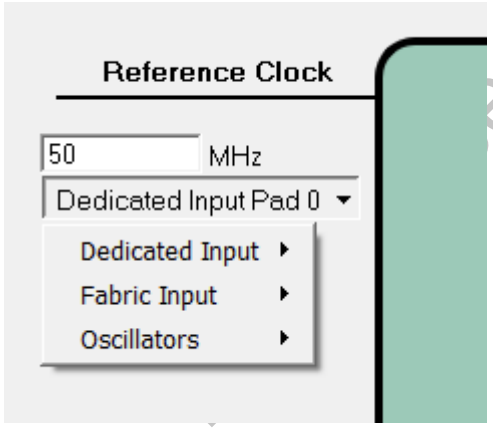
For the clock we use an internal PLL of Libero called *CCC*, to search for this block it is done from the *Catalog* tab, if it is not downloaded it allows you to download it.



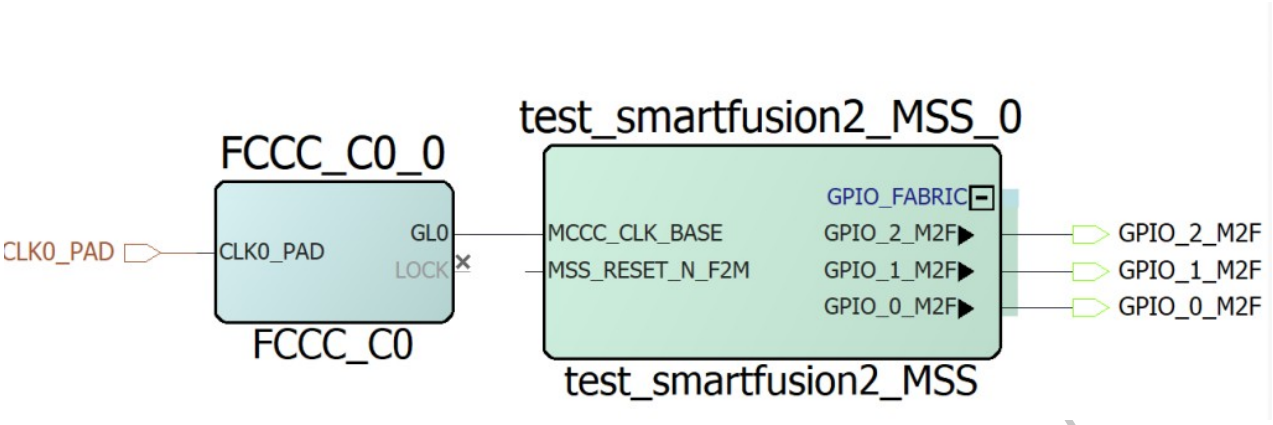
To this PLL we tell that we have a 50MHz input and that the output is 50MHz.



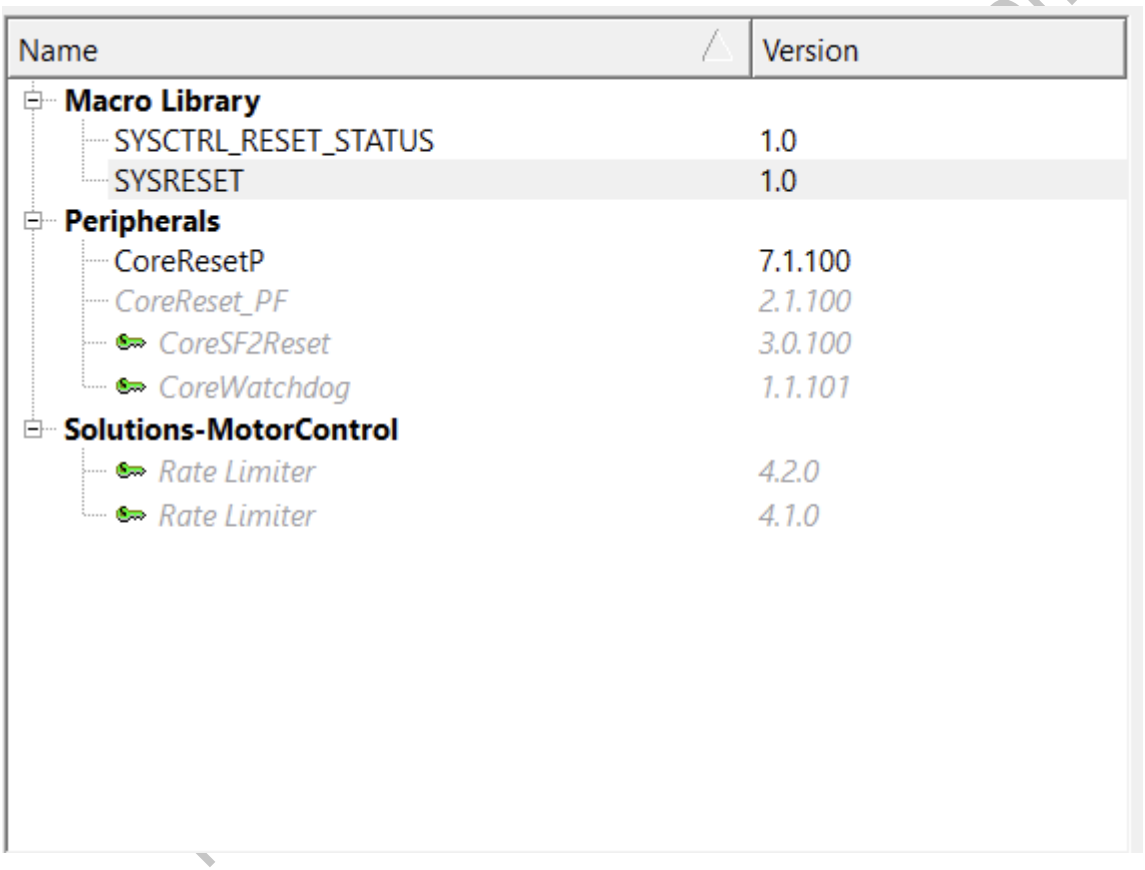
There are different ways to introduce a clock to the PLL that I will explain in another entry.



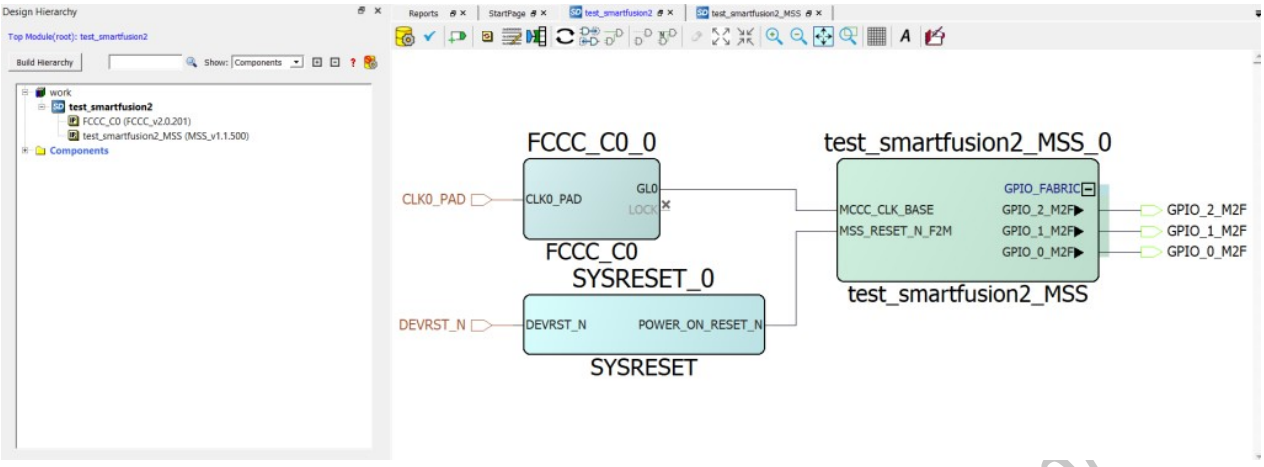
When adding the clock we join the clock with the clock input of the MSS. We leave the *LOCK* pin unused.



Now for the reset we use the SYSRESET block that the Catalog tab provides.



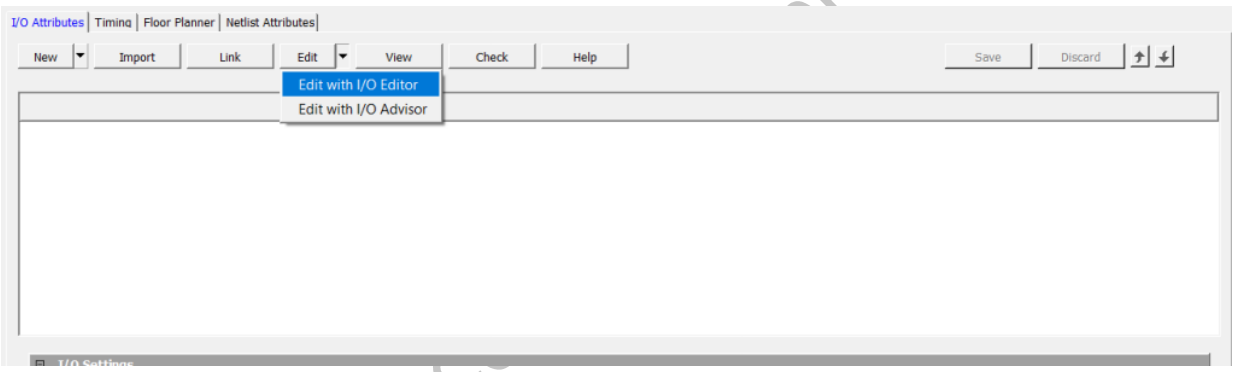
When we add it we get something like this.



To finish the configuration we click on the yellow icon with the gear.

(From here the two SoC configurations are combined into a single development)

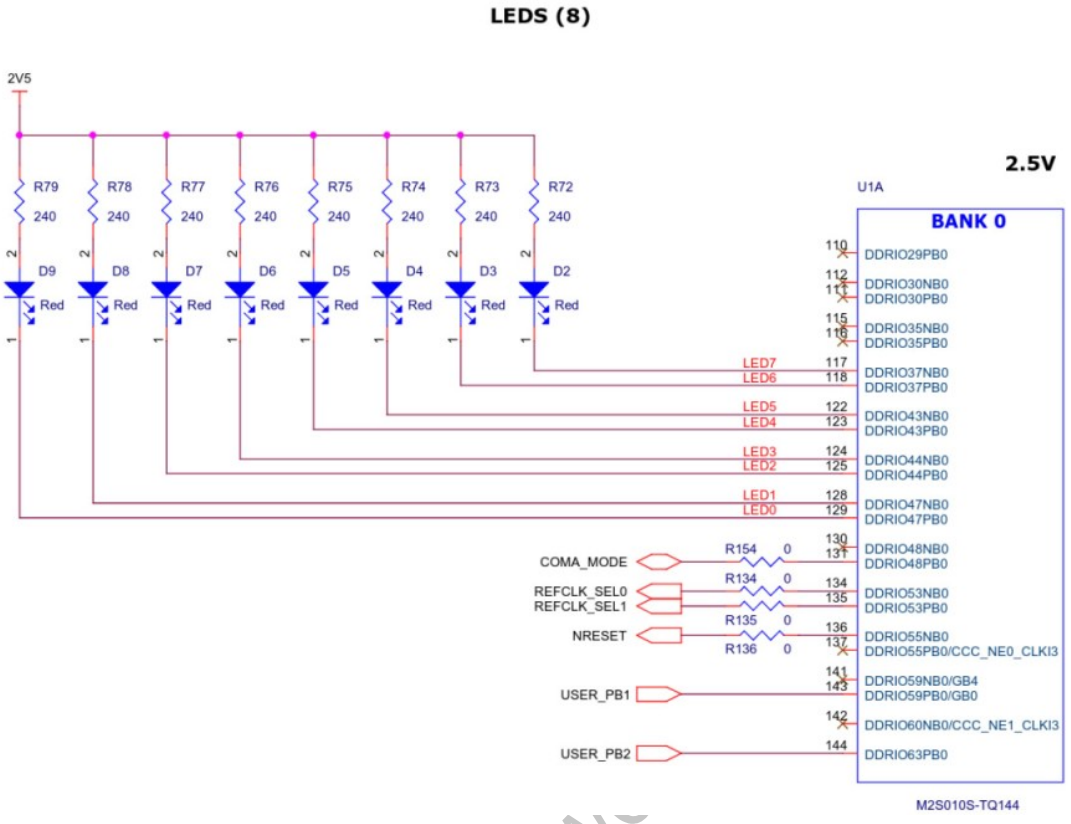
Now we synthesize the model and configure the pins in the *Manage Constraints*, then in the Edit option we click on *Edit with I/O Editor.*



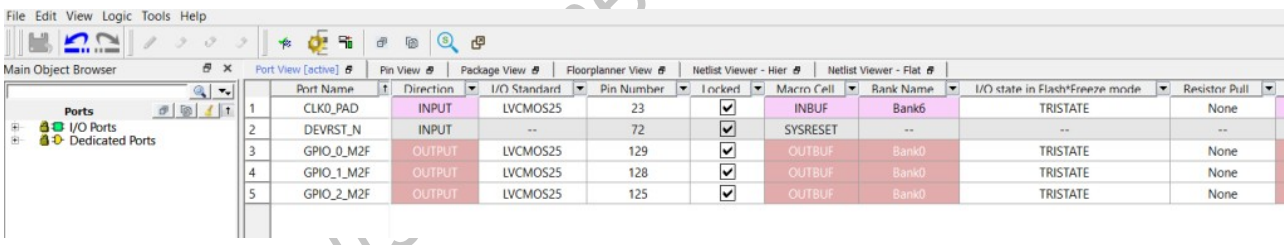
When the tab opens, the only pin that appears placed is the reset pin, because it is the reset pin of the board, the *DEVRST_N* is a specific and single-use pin.

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode	Resistor Pull
1	CLK0_PAD	INPUT	LVCMOS25		☐	INBUF	--	TRISTATE	None
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE	None
4	GPIO_1_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE	None
5	GPIO_2_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE	None

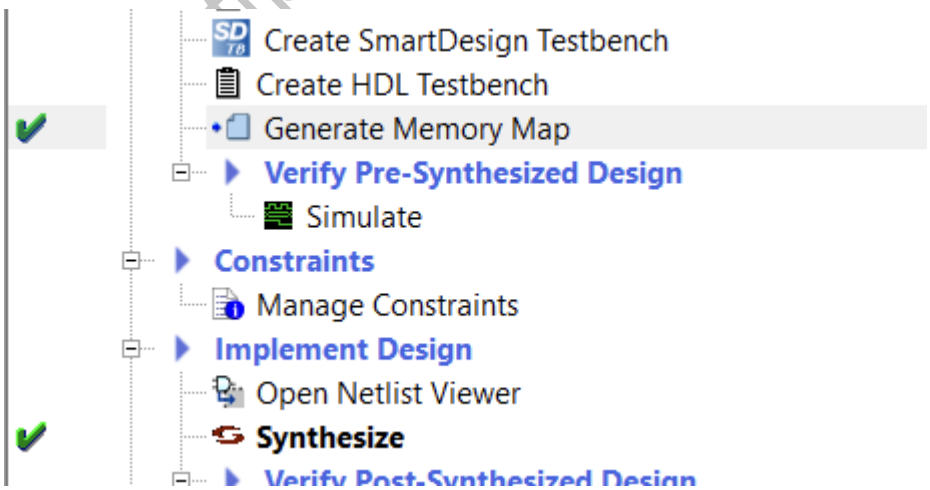
The LEDs that we want to configure are the following.



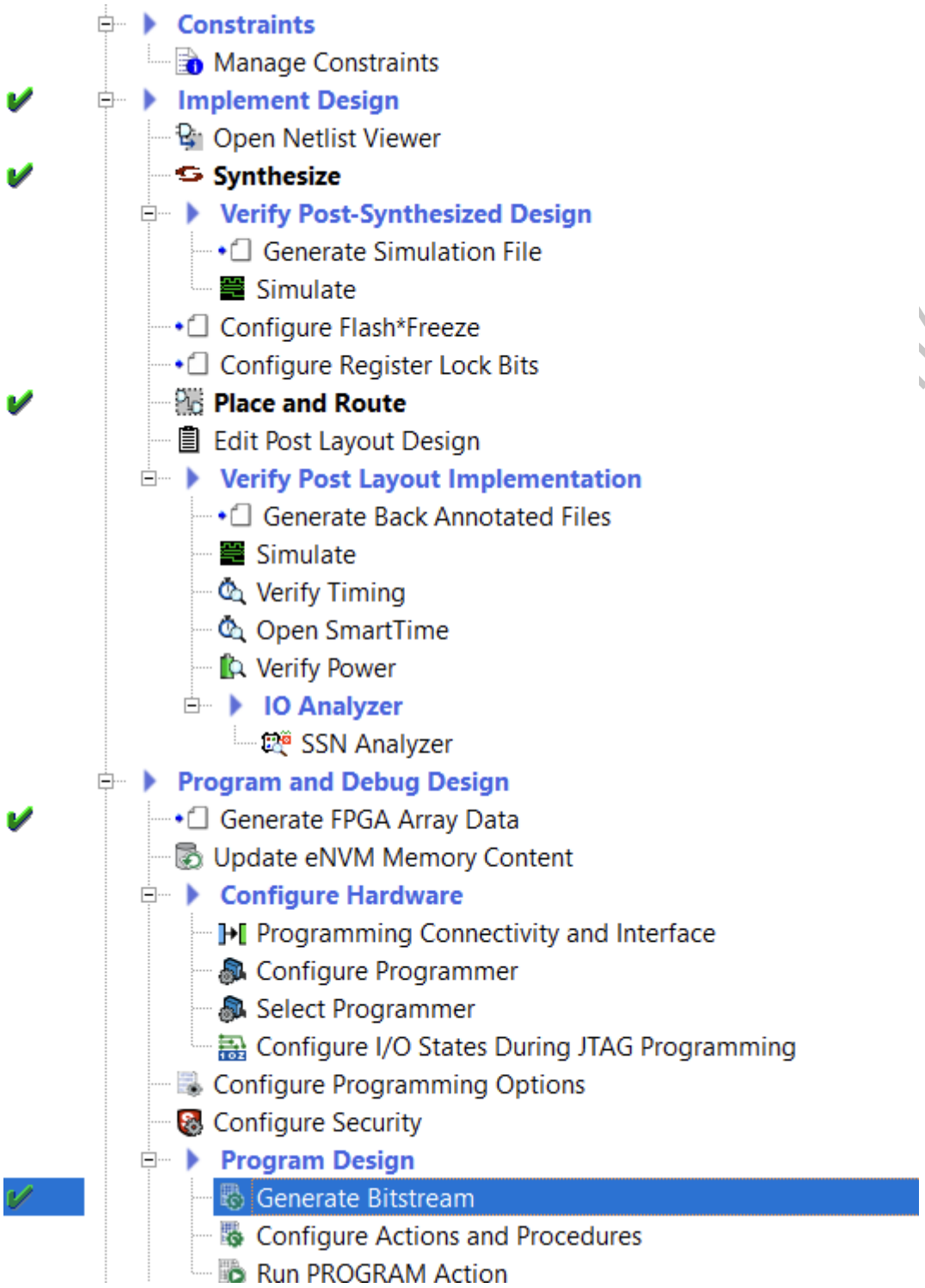
To do this we update the table so that they match.



Once we have the pins established, we can now create the bitstream. But first we generate the memory map that is going to be used.



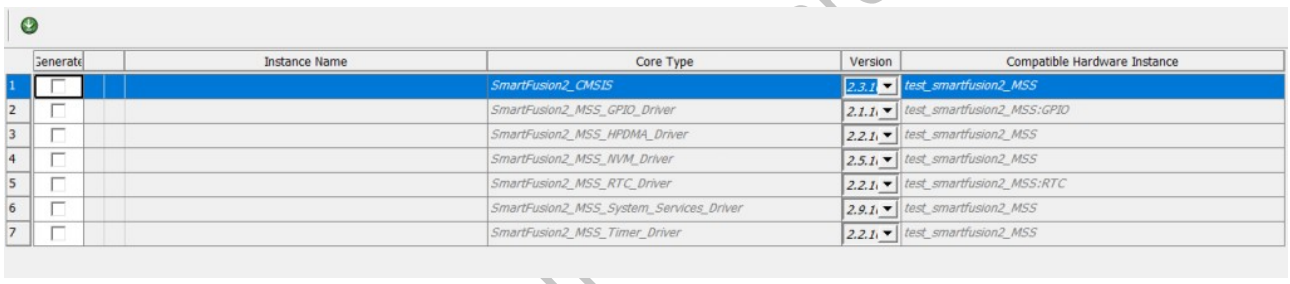
Then, we can continue with the bitstream.



If we want to debug the SoC when we go to Libero it is necessary to leave the SoC recorded, which being a Flash memory is recorded with the bitstream, so there is no possible loss.

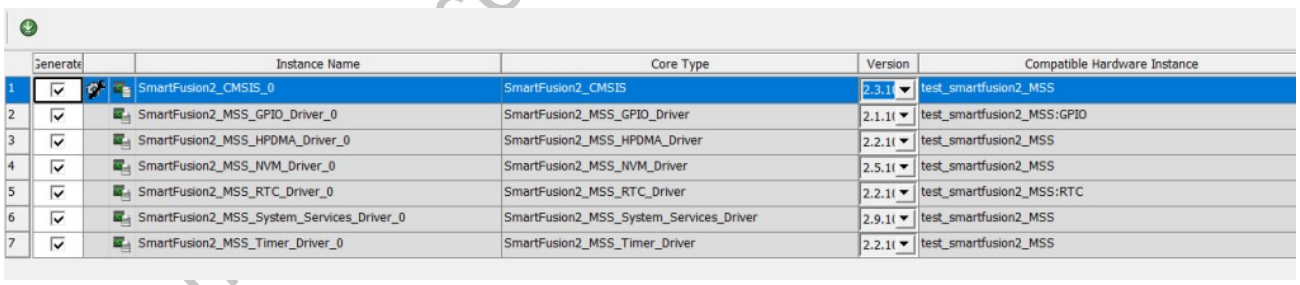
The next step before going to the SoftConsole is to get the drivers for the peripherals that we want to use. To do this we access the *Configure Firmware Cores* option.

When it opens you can see that there are no drivers installed. To download them there are two options: check the box on the left to install the driver one by one or click on the download symbol above.



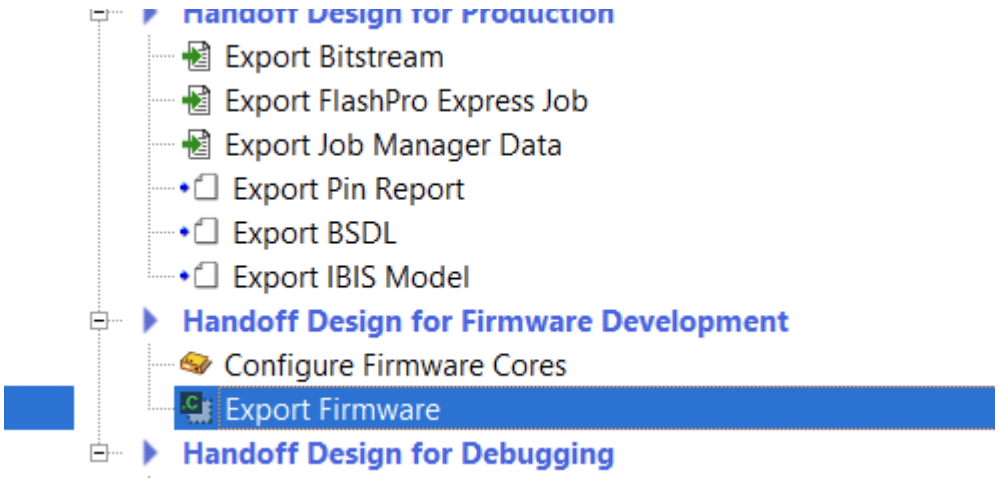
	Generate		Instance Name	Core Type	Version	Compatible Hardware Instance
1	☐			SmartFusion2_CMSIS	2.3.1	test_smartfusion2_MSS
2	☐			SmartFusion2_MSS_GPIO_Driver	2.1.1	test_smartfusion2_MSS:GPIO
3	☐			SmartFusion2_MSS_HPDMA_Driver	2.2.1	test_smartfusion2_MSS
4	☐			SmartFusion2_MSS_NVM_Driver	2.5.1	test_smartfusion2_MSS
5	☐			SmartFusion2_MSS_RTC_Driver	2.2.1	test_smartfusion2_MSS:RTC
6	☐			SmartFusion2_MSS_System_Services_Driver	2.9.1	test_smartfusion2_MSS
7	☐			SmartFusion2_MSS_Timer_Driver	2.2.1	test_smartfusion2_MSS

Once downloaded, we can move on to the next step.

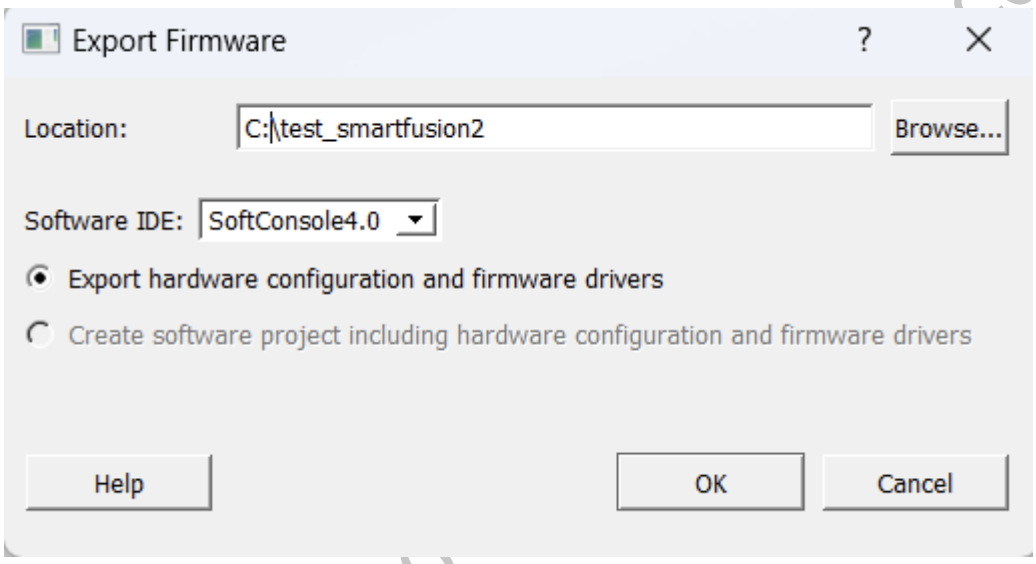


	Generate		Instance Name	Core Type	Version	Compatible Hardware Instance
1	☑		SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.1	test_smartfusion2_MSS
2	☑		SmartFusion2_MSS_GPIO_Driver_0	SmartFusion2_MSS_GPIO_Driver	2.1.1	test_smartfusion2_MSS:GPIO
3	☑		SmartFusion2_MSS_HPDMA_Driver_0	SmartFusion2_MSS_HPDMA_Driver	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_System_Services_Driver_0	SmartFusion2_MSS_System_Services_Driver	2.9.1	test_smartfusion2_MSS
7	☑		SmartFusion2_MSS_Timer_Driver_0	SmartFusion2_MSS_Timer_Driver	2.2.1	test_smartfusion2_MSS

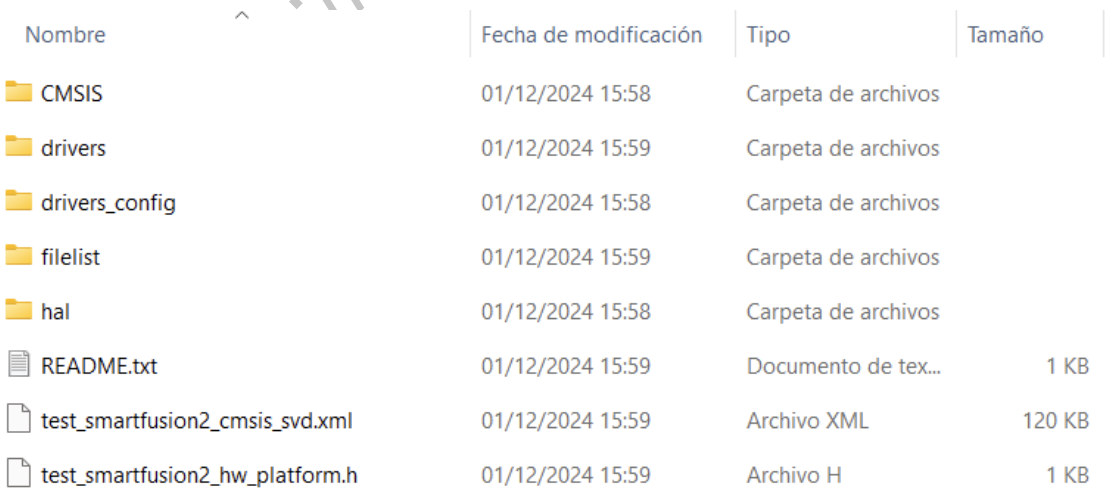
Now we export the drivers, to do this we check the *Export Firmware* option.



It asks us where we want to download them.



If we open the *firmware* folder that has been generated, it has the drivers for our project.



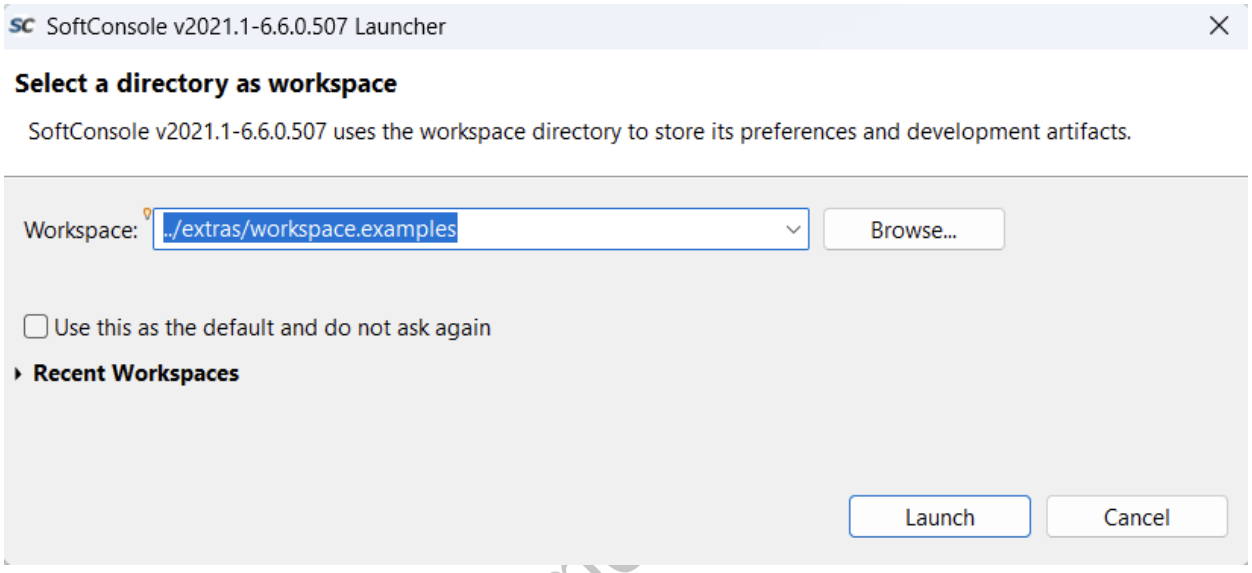
With this the Libero project is finished, now we go to the SoftConsole.

SoftConsole project

Because we want to create a project for a SmartFusion2 that has a Cortex-M3 inside, it is necessary to have a version of SoftConsole installed prior to 2022.

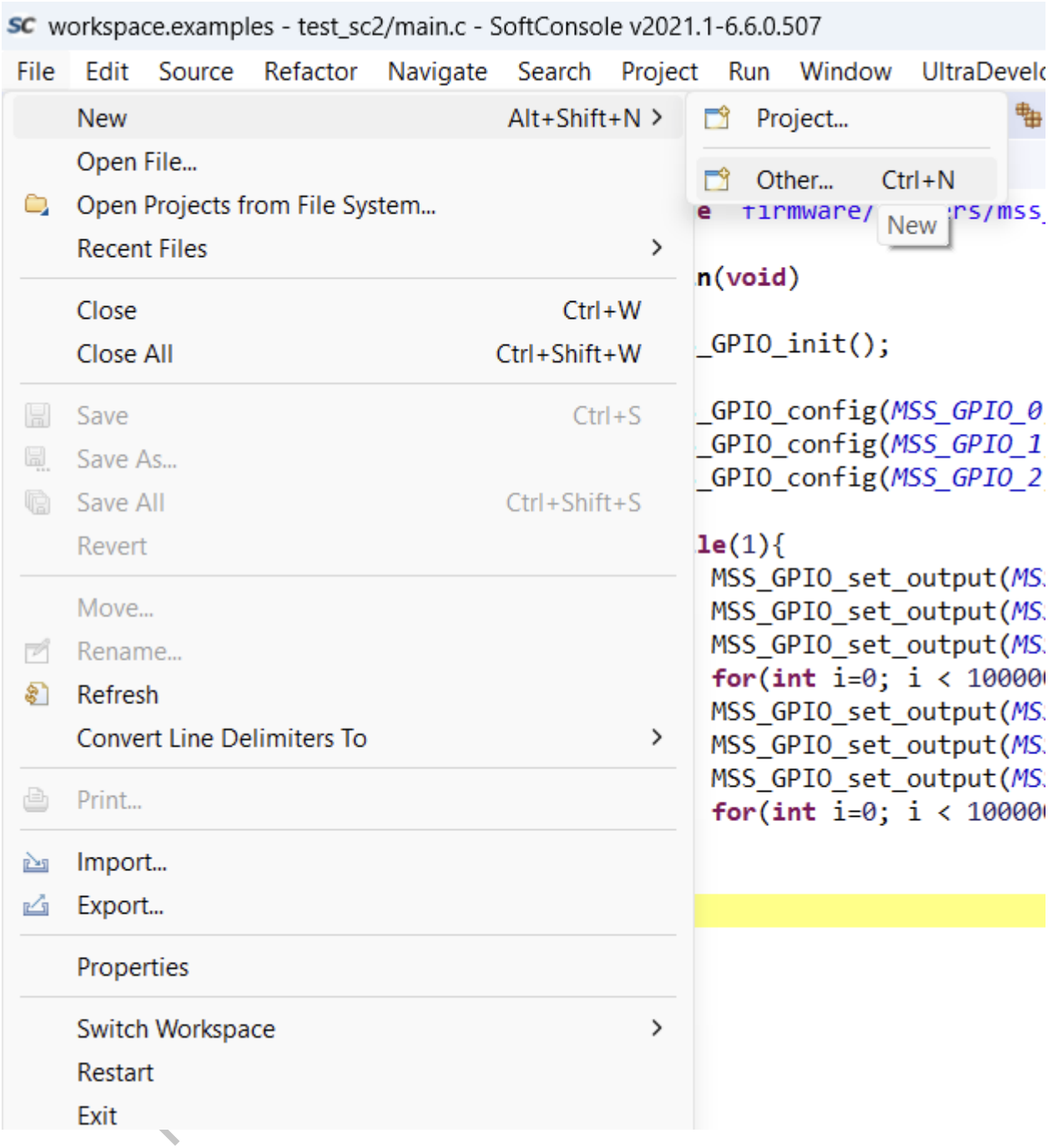
NOTE: the SoftConsole is based on Eclipse, so it is quite similar to Vitis or the Nios II editor.

To start, we first open the SoftConsole, which asks us where we want to open the project.

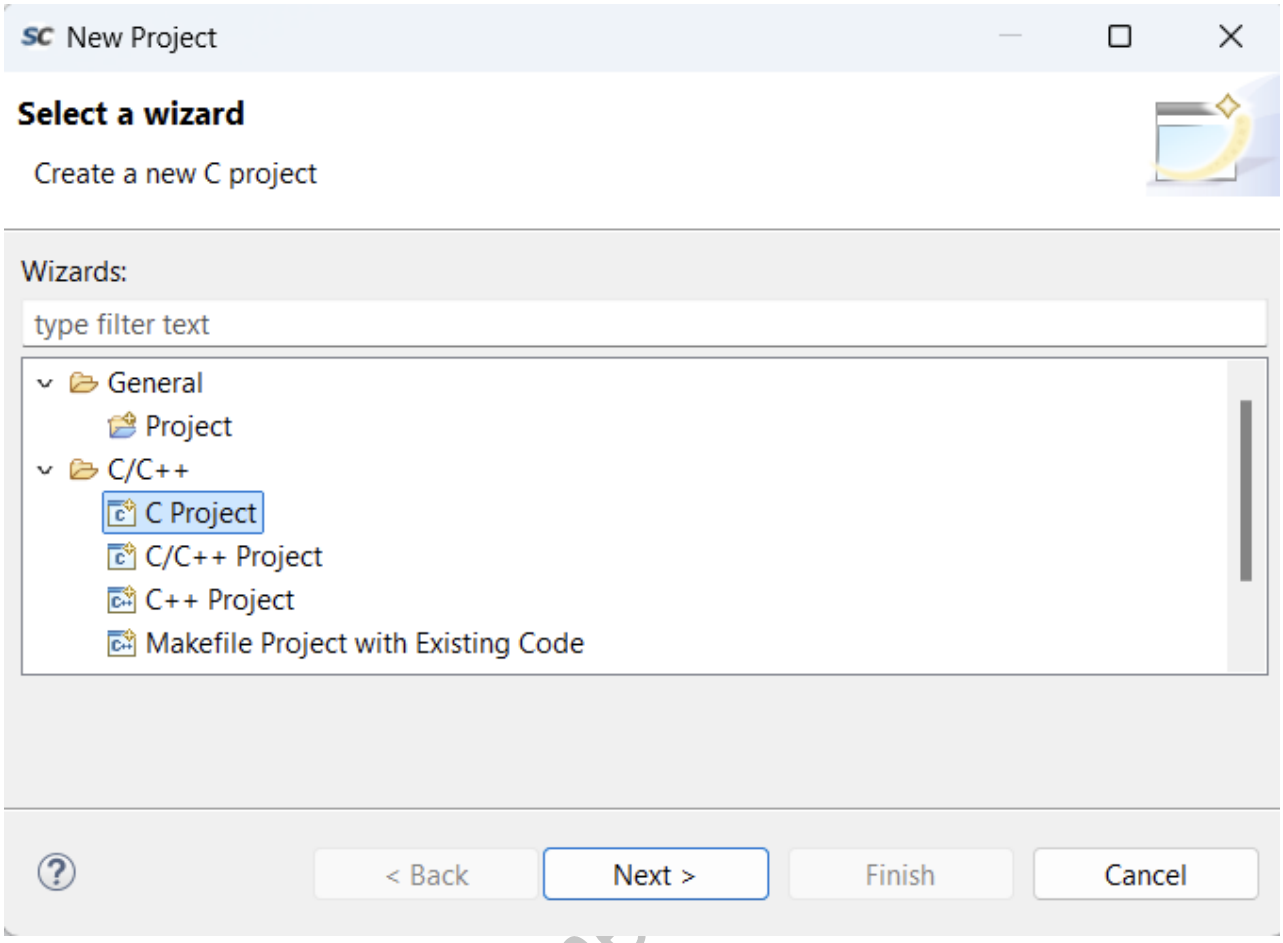


Once opened, we create a project.

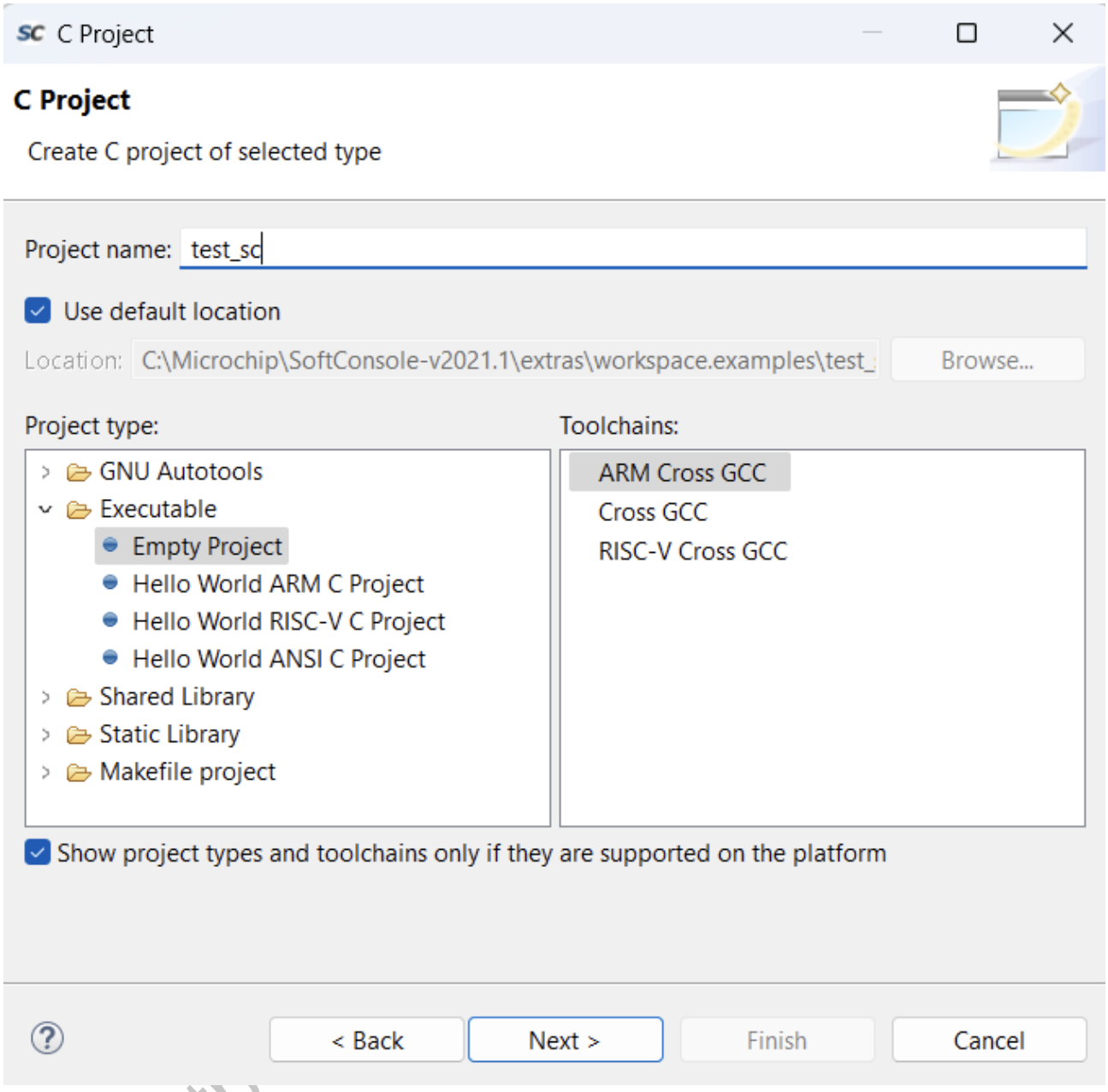
https://soceame.wordpress.com/2025/03/09/how-to-create-a-project-for-a-smartfusion2-board/



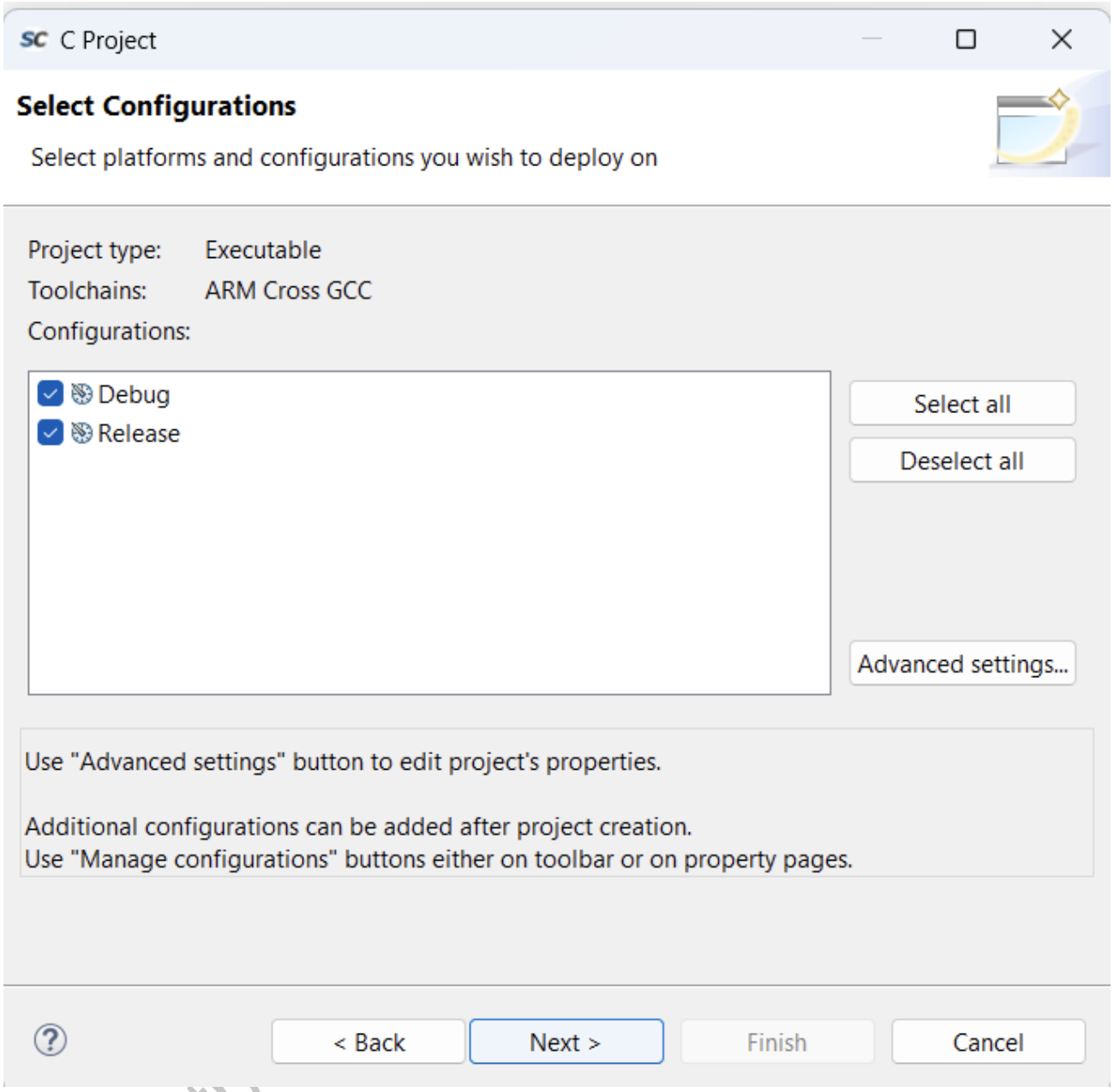
In our case we create a project in C, this means that when we create it, it creates the *includes* folder.



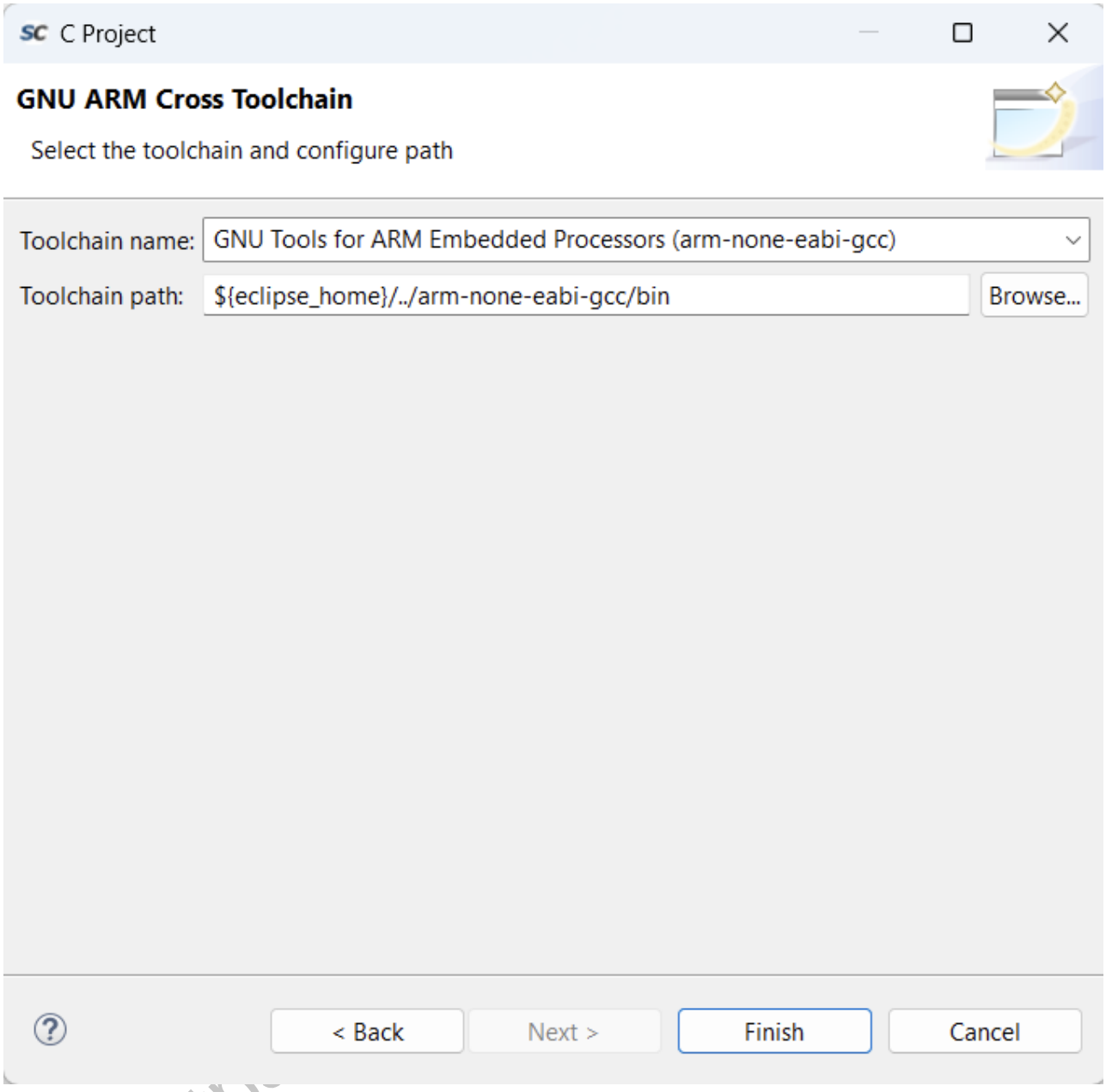
Now we create an empty project based on an ARM.



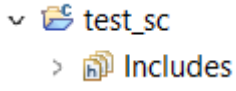
Then we tell it that it will be for debugging and for *release*.



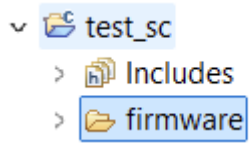
Then we tell it to use GCC for ARM.



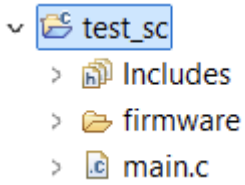
Now we can create a project. When creating the project, the SoftConsole creates this for us.



First we import the firmware folder created by Libero with the drivers.



Now we can create our project in a main.c file



Inside the main.c we program the operation of the LEDs.

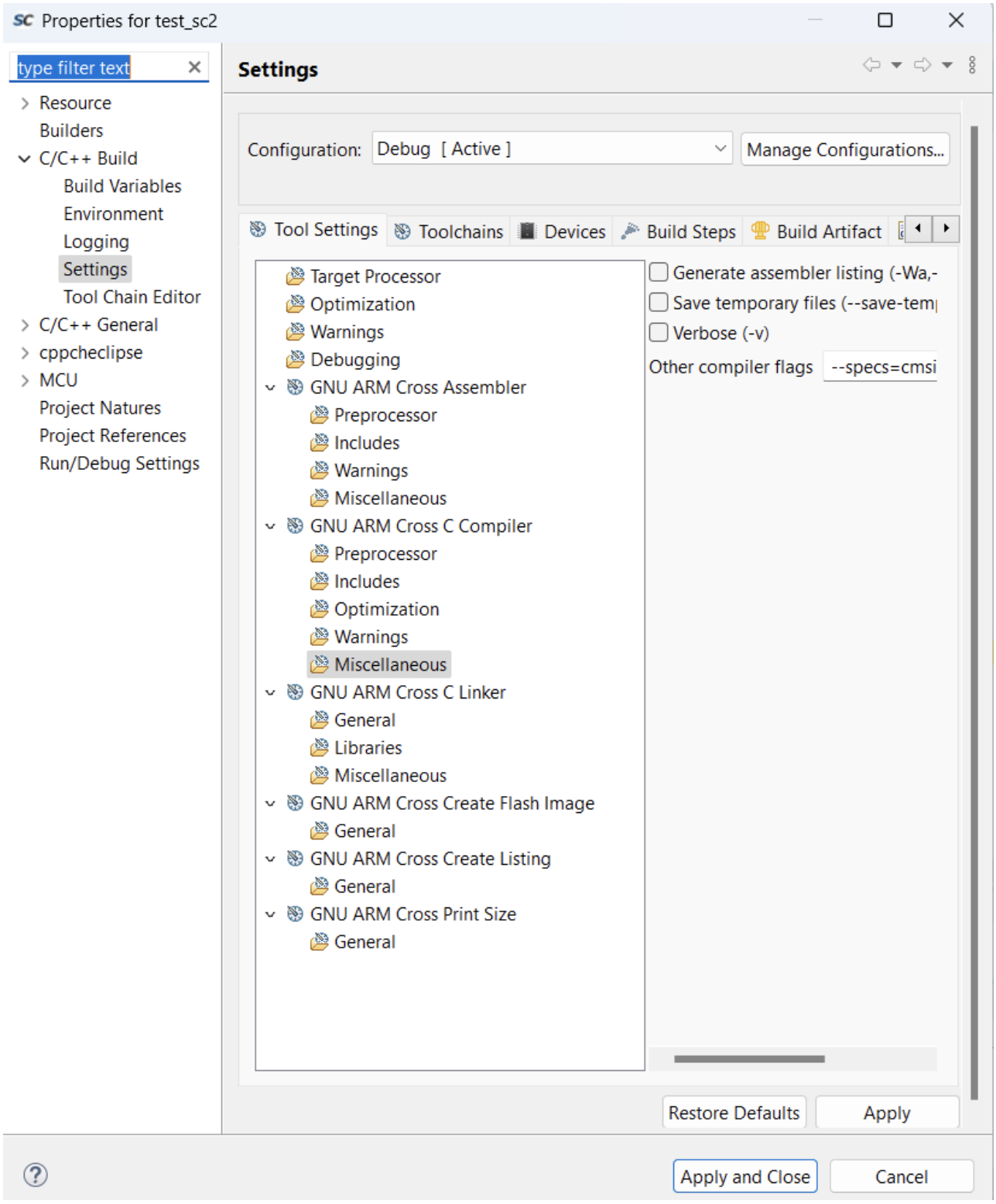
To do this we first call the GPIO drivers file, which is inside the firmware folder we added, then we configure the GPIOs as output, then we assign them a value of 0 and 1 every 1000000 clock counts, which makes the LEDs blink.

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

int main(void)
{
    MSS_GPIO_init();

    MSS_GPIO_config(MSS_GPIO_0, MSS_GPIO_OUTPUT_MODE);
    MSS_GPIO_config(MSS_GPIO_1, MSS_GPIO_OUTPUT_MODE);
    MSS_GPIO_config(MSS_GPIO_2, MSS_GPIO_OUTPUT_MODE);

    while(1){
        MSS_GPIO_set_output(MSS_GPIO_0, 0);
        MSS_GPIO_set_output(MSS_GPIO_1, 0);
        MSS_GPIO_set_output(MSS_GPIO_2, 0);
        for(int i=0; i < 1000000; i++);
        MSS_GPIO_set_output(MSS_GPIO_0, 1);
        MSS_GPIO_set_output(MSS_GPIO_1, 1);
        MSS_GPIO_set_output(MSS_GPIO_2, 1);
        for(int i=0; i < 1000000; i++);
    }
}
```

Now if we try to compile it, it will give an error, this is because we have to configure internal variables of the compiler.

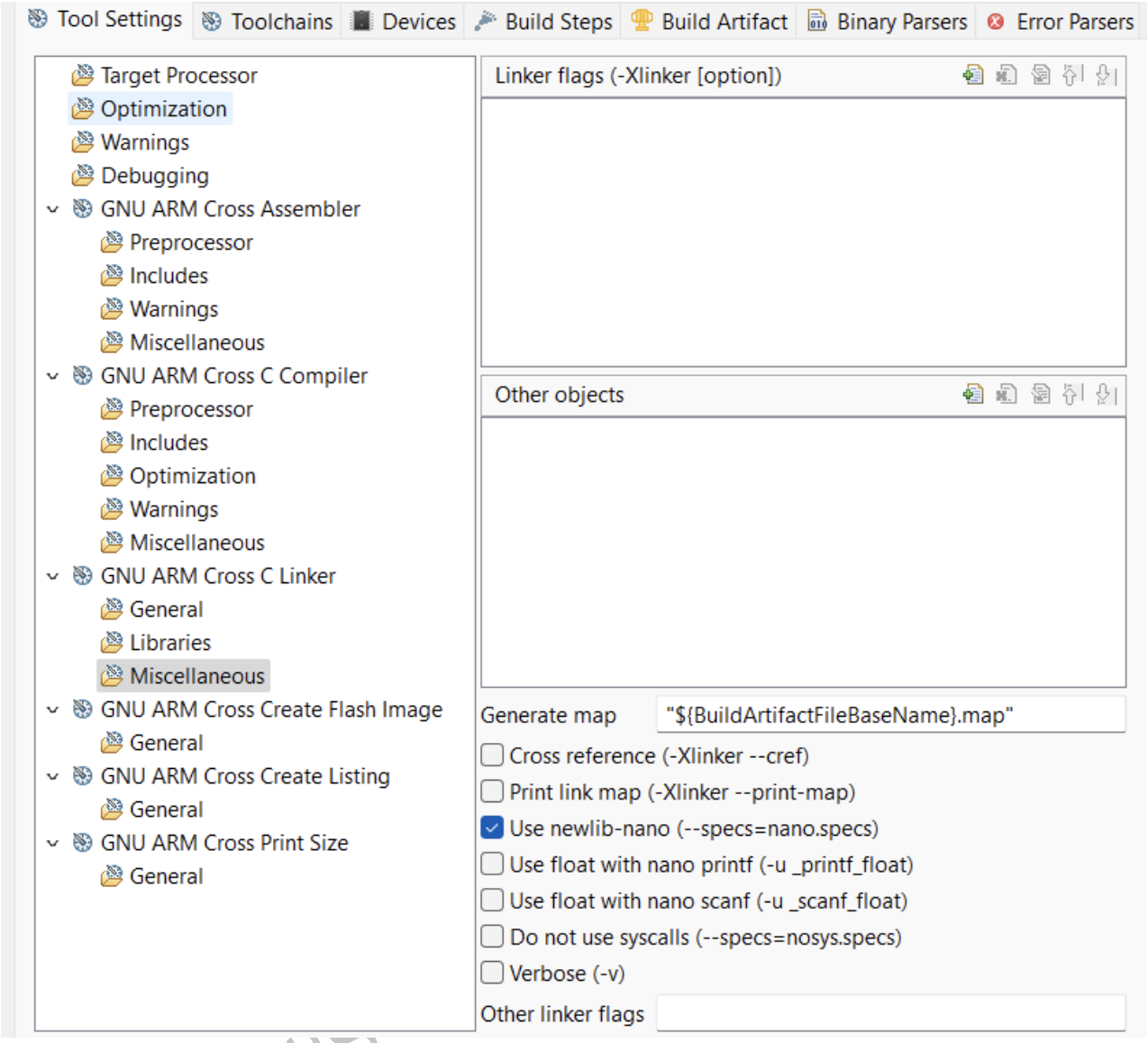
Compilation

To access the variables, we do it by right-clicking on the project we have created and clicking on *properties*. Once in the tab we have to go to *C/C++ Build in Settings.*

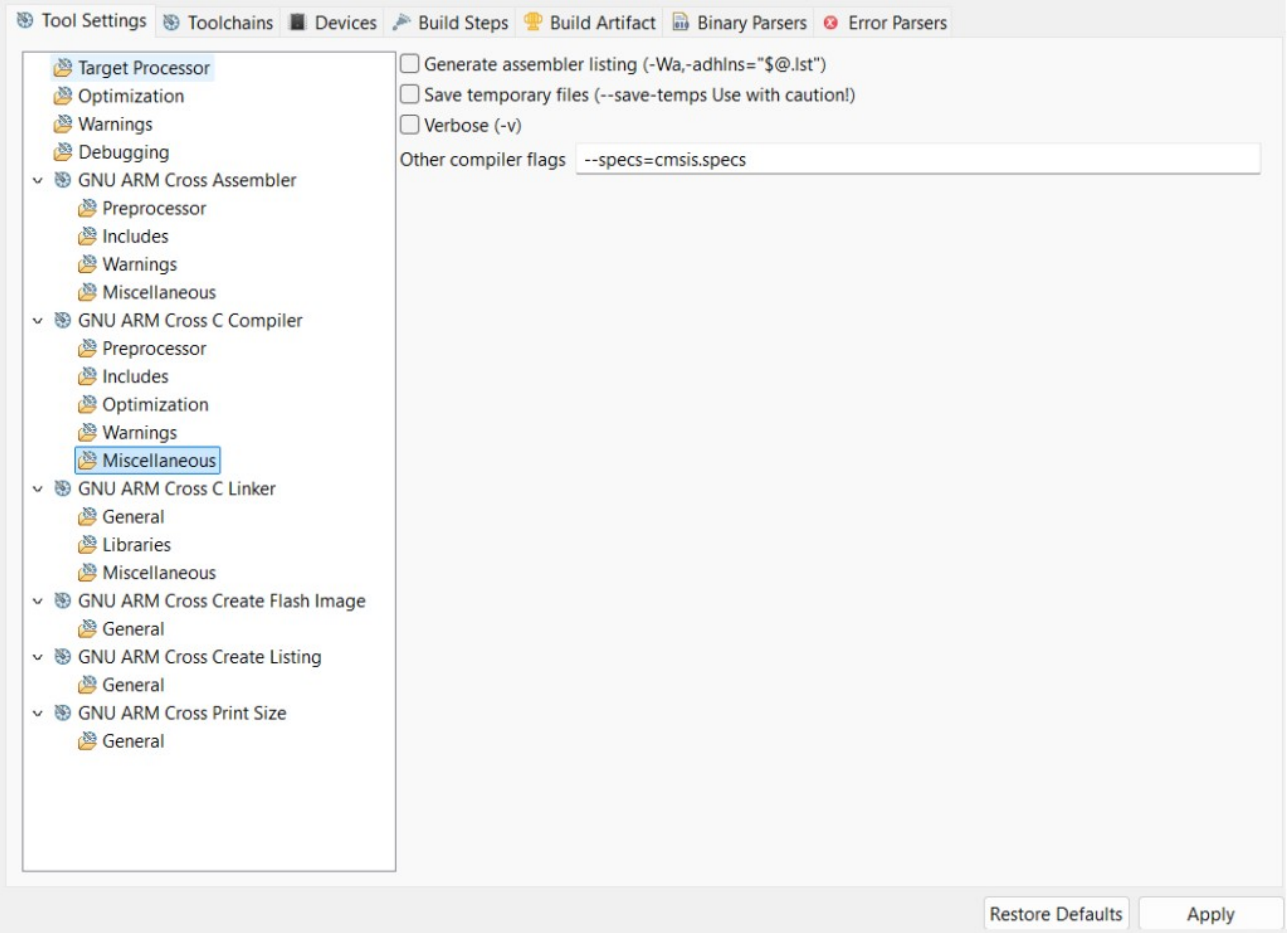


These variables are the ones that have to be configured:

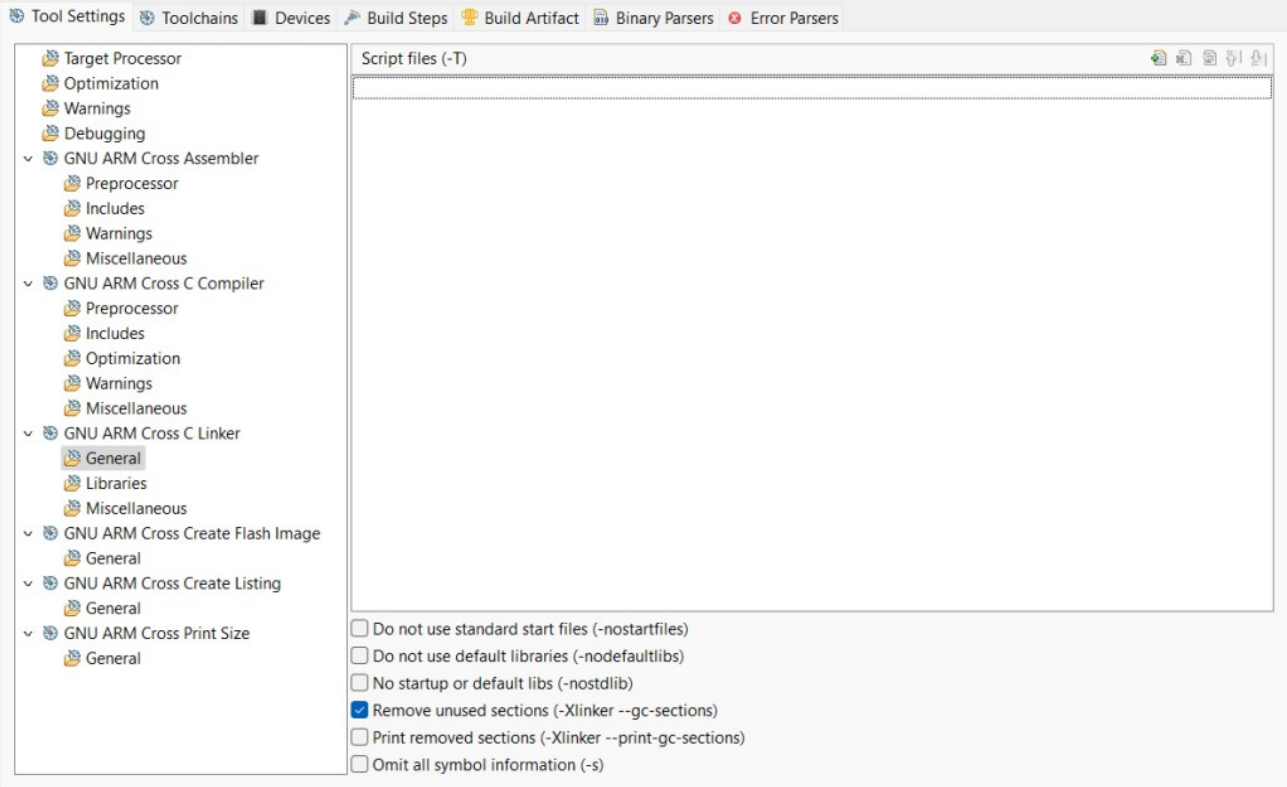
- Check the box *--specs=nano.specs* in *Miscellaneous* of the *GNU ARM Cross C Linker.*



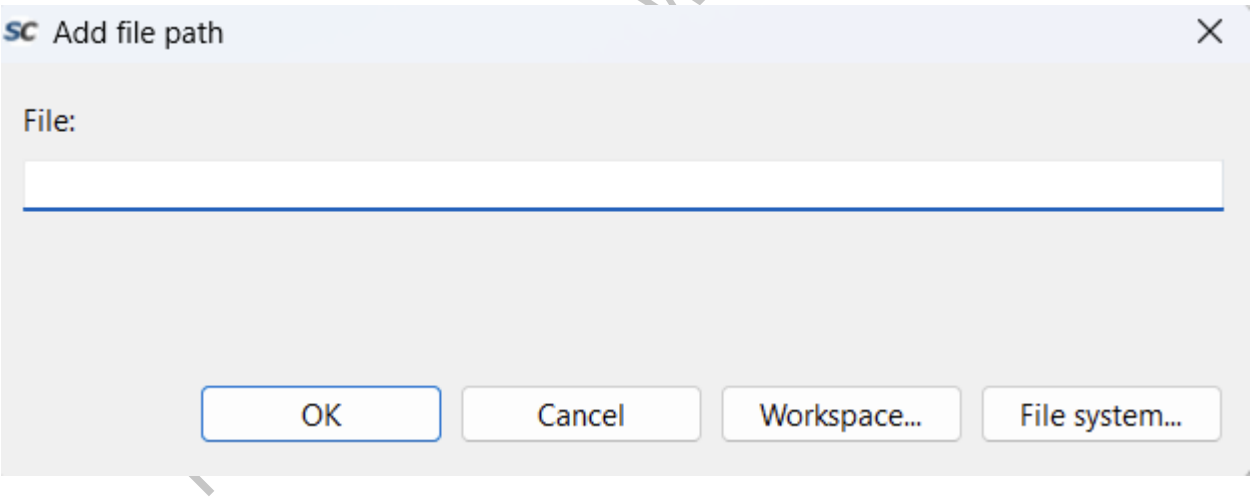
- Add the flag *--specs=cmsis.specs* in *Miscellaneous* of the *GNU ARM Cross C Compiler.*



- And finally, since it is a Flash-based FPGA, we have to indicate whether we want to save the program when the FPGA is turned off.
  To do this, in *General* of the *GNU ARM Cross Compiler* we have to add a file of type .ld.

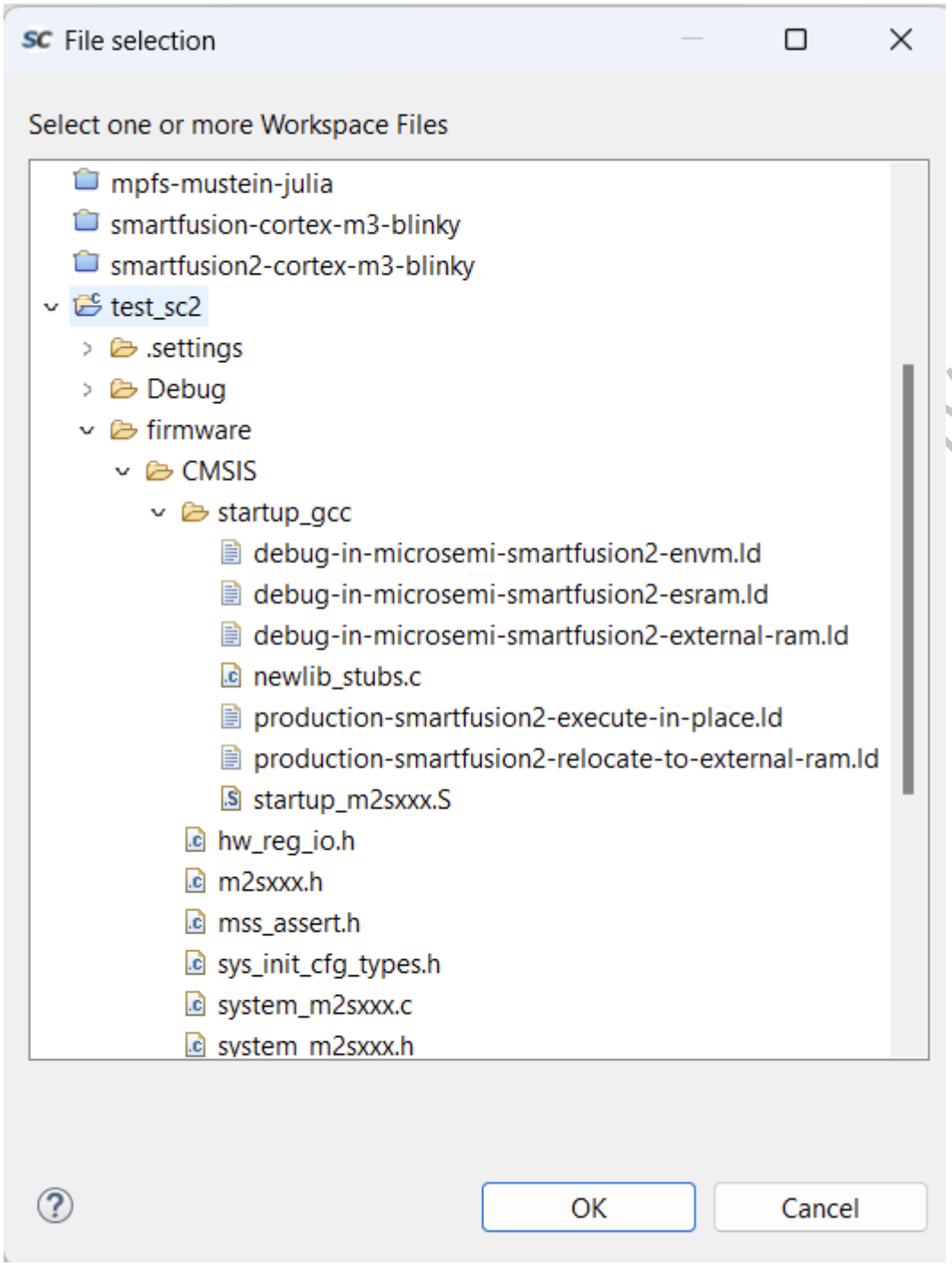


To do this, we click on the add file button and it asks us where we are going to add the file from.

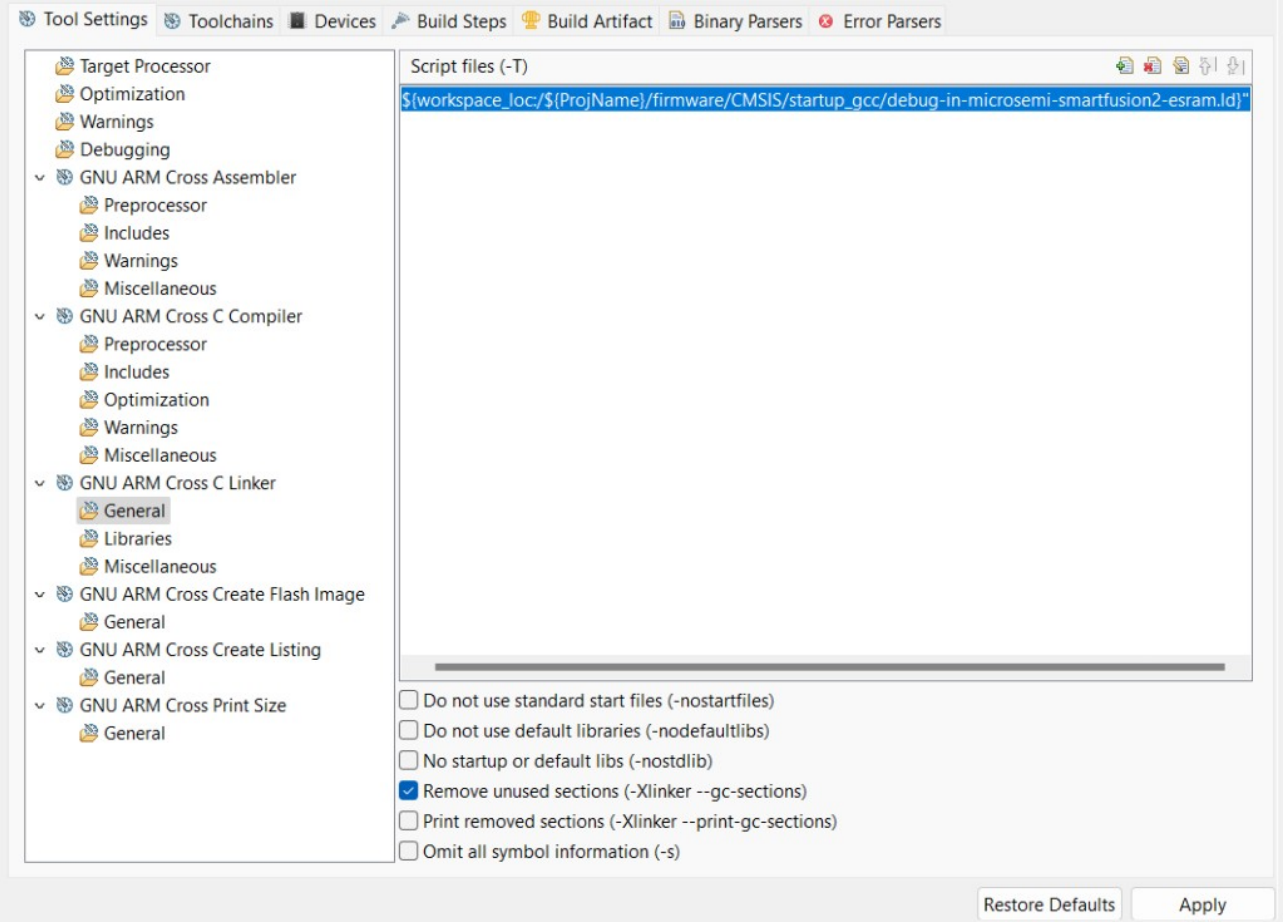


We click on *workspace* and a tab opens, the files we are looking for are in the project we have created. In *firmware* > *CMSIS* > *startup_gcc*. Here we have different options.

- Write the executable in non-volatile memory (…*envm.ld*). This means that we do not have to write the FPGA more than once.
- Write the executable in SRAM memory (…*esram.ld*). This deletes the executable when the board is turned off (but the Libero bitstream remains inside).
- Write the executable in external RAM (…*external-ram.ld*). I do not know how this works.

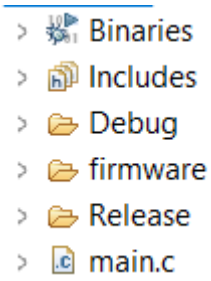


Once chosen, all the compilation variables are programmed.



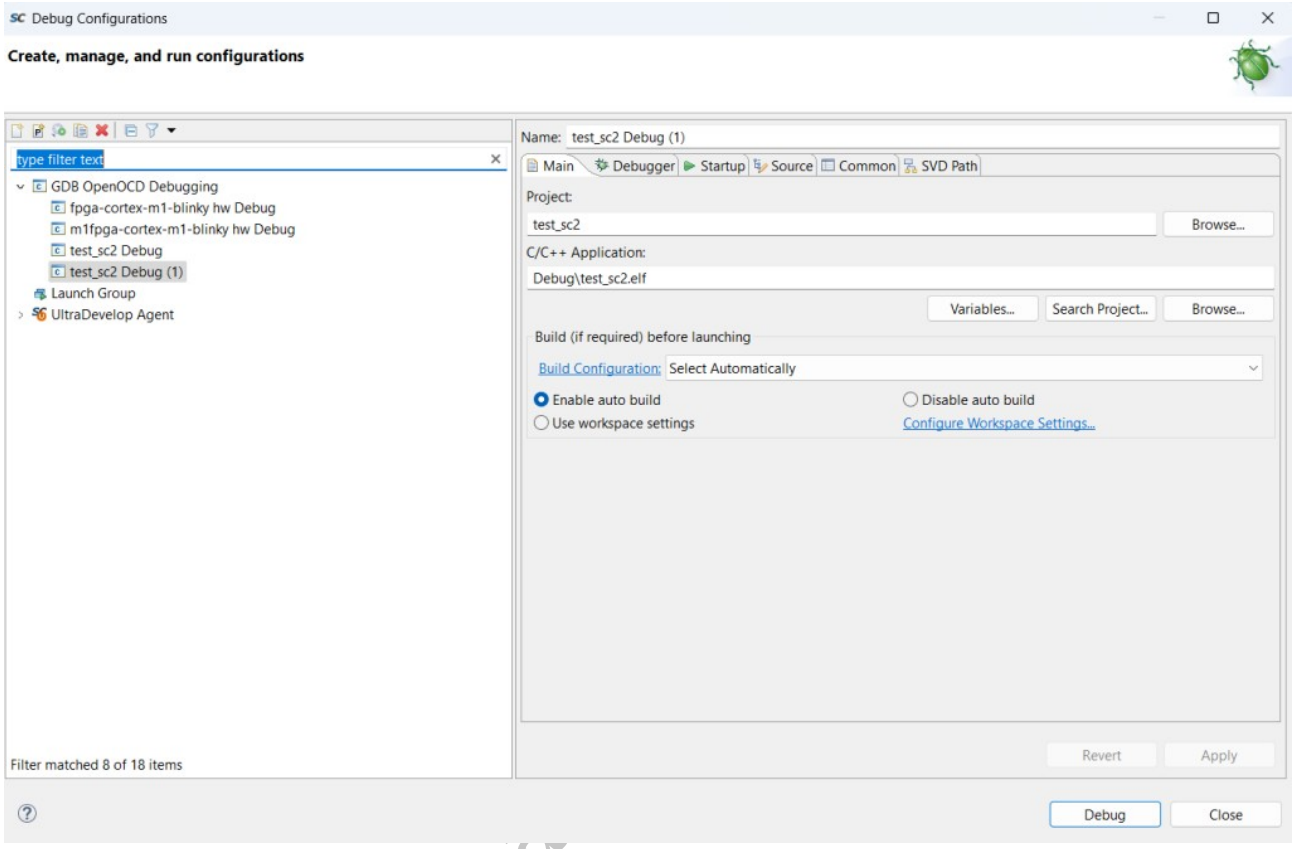
Now you can compile. When compiling, up to three folders appear (depending on how the compilation is done, if it is a debugging version, the *Debug* folder appears, if it is a functional version, the *Release* folder appears).

- *Binaries*: This is a folder where SoftConsole puts the project's .elf so that it is easier to locate.
- *Debug*: This is the folder where the debug .elf is generated. The .hex that ARM cores need to be programmed independently, that is, using J-Link, is also generated.
- *Release*: This is the folder where the debug .elf is generated. The .hex that ARM cores need to be programmed independently, that is, using J-Link, is also generated. This .hex can also be added to the bitstream generated by Libero to create a single bitstream with all the functionality of the SoC, which can be programmed from Libero itself or from FlashPro Express.



Debugging/Execution

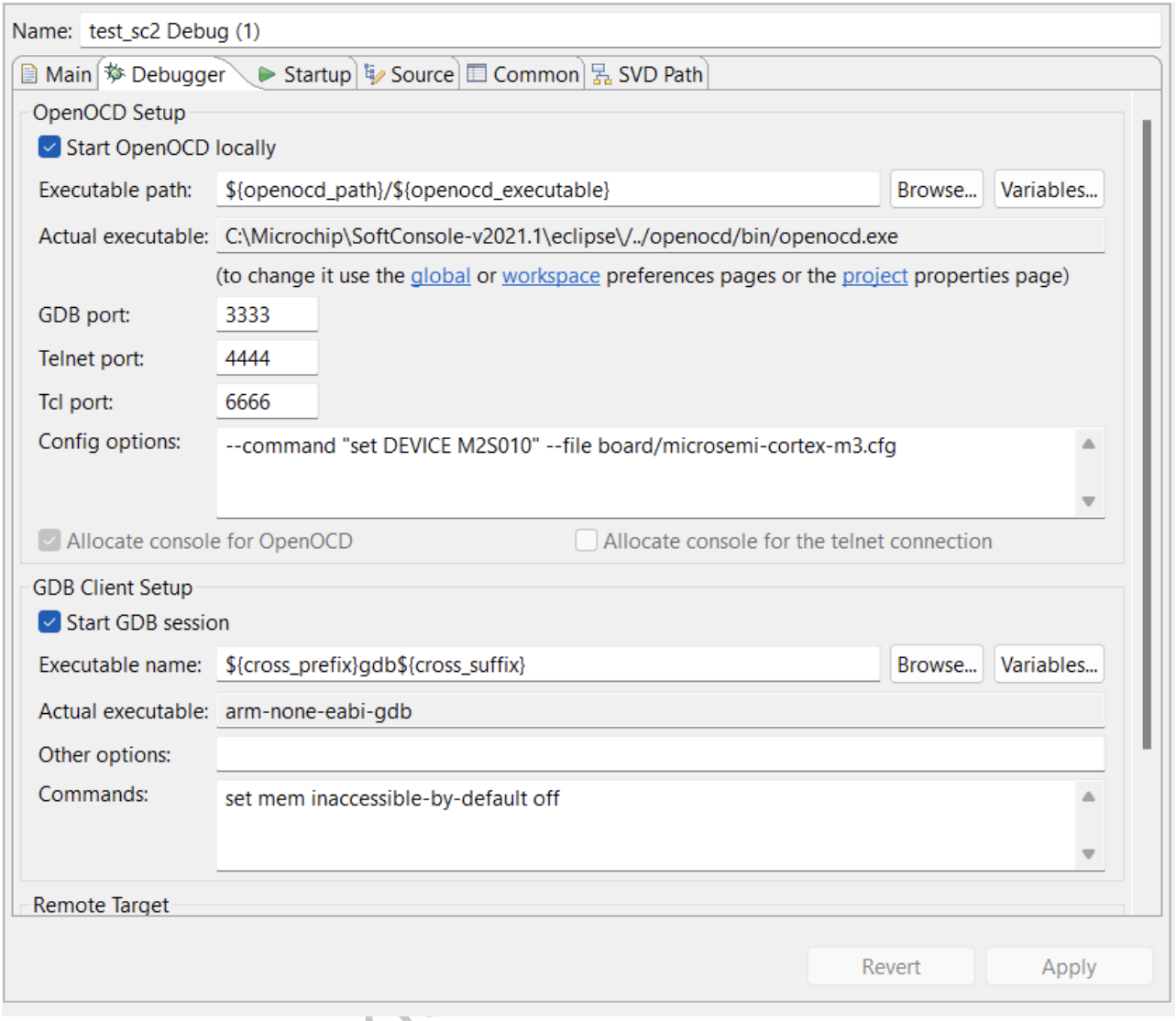
For debugging/execution, a debugging/execution profile must be created. To do this, in the *Configurations* option, first create a profile with the .elf that you want to test.



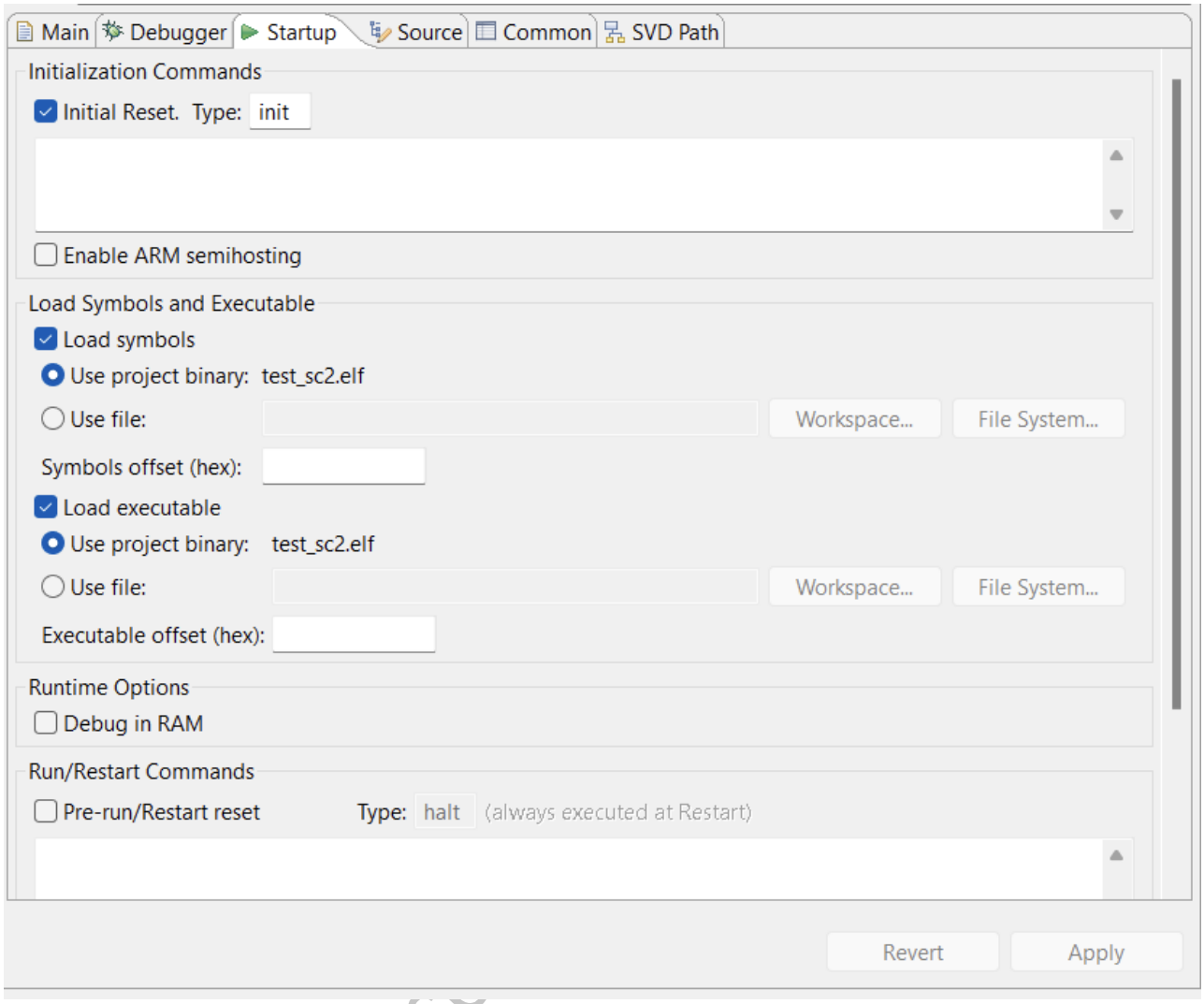
In the *Debugger* tab you have to configure it as follows. And in Config options you have to add the following text:

–command «set DEVICE <Device to program>» –file board/microsemi-cortex-m3.cfg

(In my case I am going to program a SmartFusion2 of type M2S010). And below you add the command: **set mem inaccessible-by-default off**



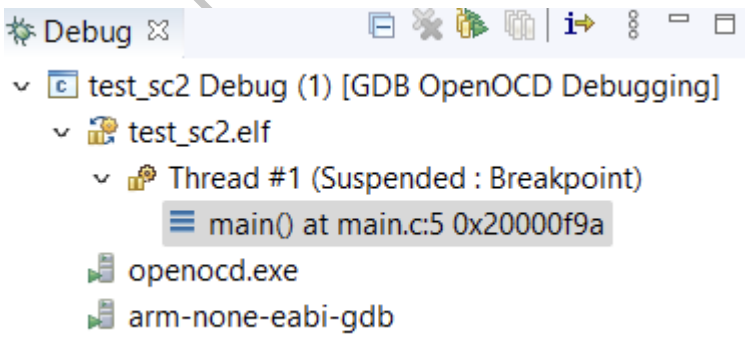
And finally, in the *Startup* tab, uncheck the *Pre-run/Restart reset* box.



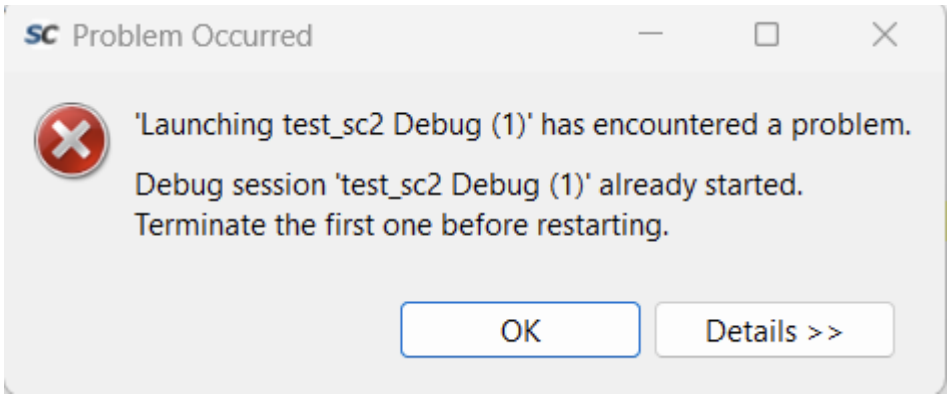
With all this, you can now debug the board. **Remember that you must first record the bitstream from Libero, which is not lost between reprogramming.**

When it starts debugging, it usually has a breakpoint on the first line to indicate this.

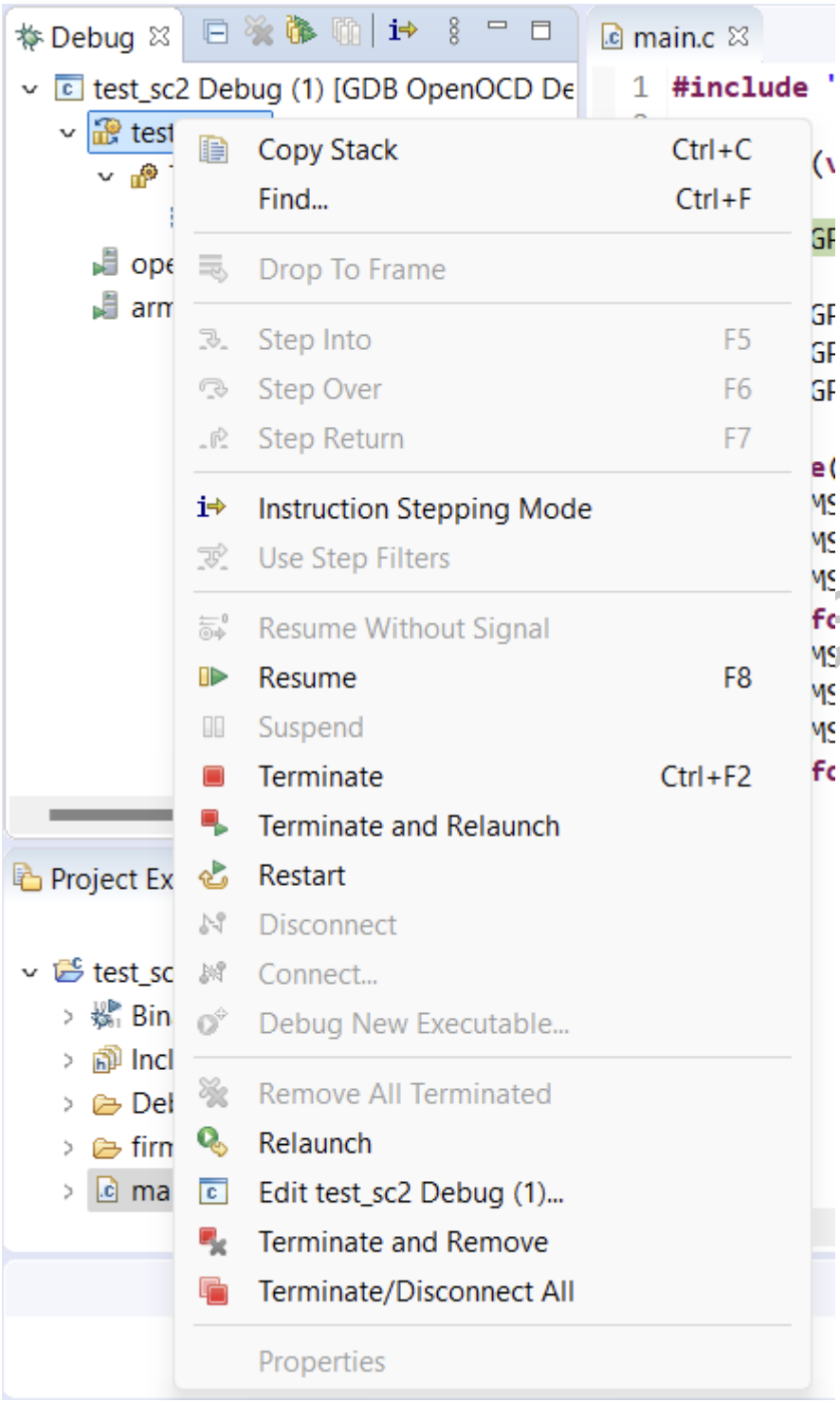
To know if it is being debugged correctly, in the debug profile there is a tab called *Debug*, which appears there if you are debugging a core.



Note: every time you want to reprogram the SoC, you will get an error, telling you that there is already a program running, and that trying to reprogram it will end the previous debugging.



To avoid this message, you can right-click on Debug and *Terminate and Relaunch.*



From this entry on, other peripherals that the SmartFusion2 have will be discussed, on how to work with them.