

# **Cómo crear un proyecto para una SmartFusion2**

Creador: David Rubio G.

Entrada: <https://soceame.wordpress.com/2024/12/02/como-crear-un-proyecto-para-una-smartfusion2/>

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

GitHub: <https://github.com/DRubioG>

Fecha última modificación: 23/02/2025

Para este proyecto se utiliza como base para ir más rápido esta otra entrada.

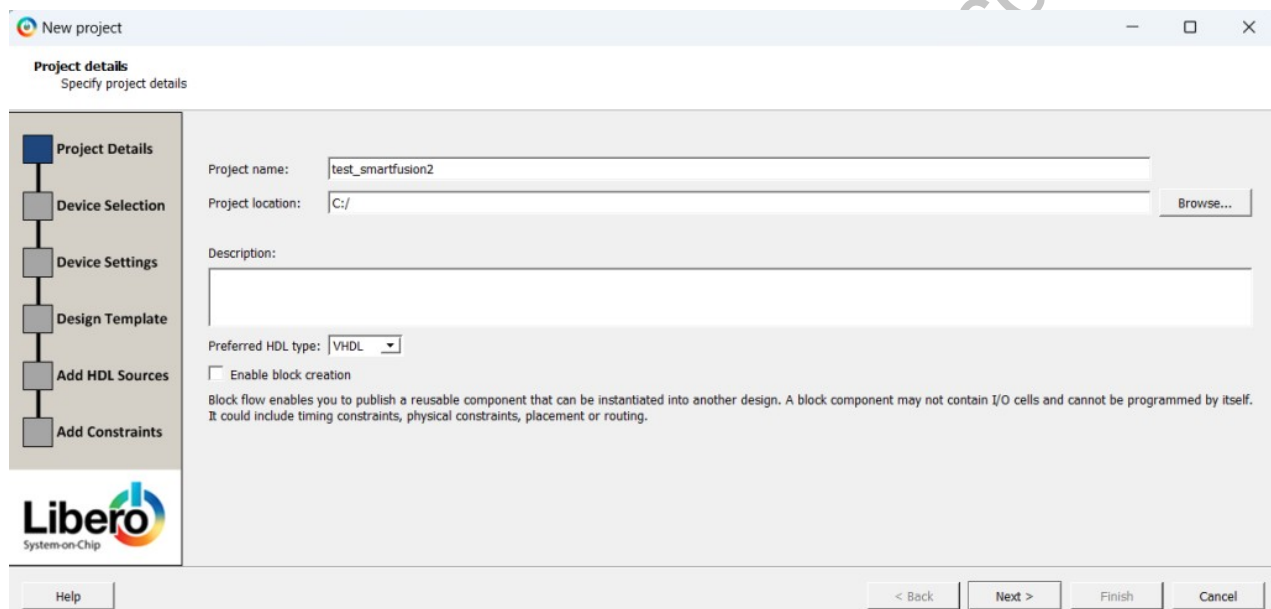
<https://soceame.wordpress.com/2024/12/01/proyecto-basico-con-libero/>

En esta entrada voy a explicar cómo crear un proyecto para una SmartFusion2 utilizando, tanto Libero como el SoftConsole.

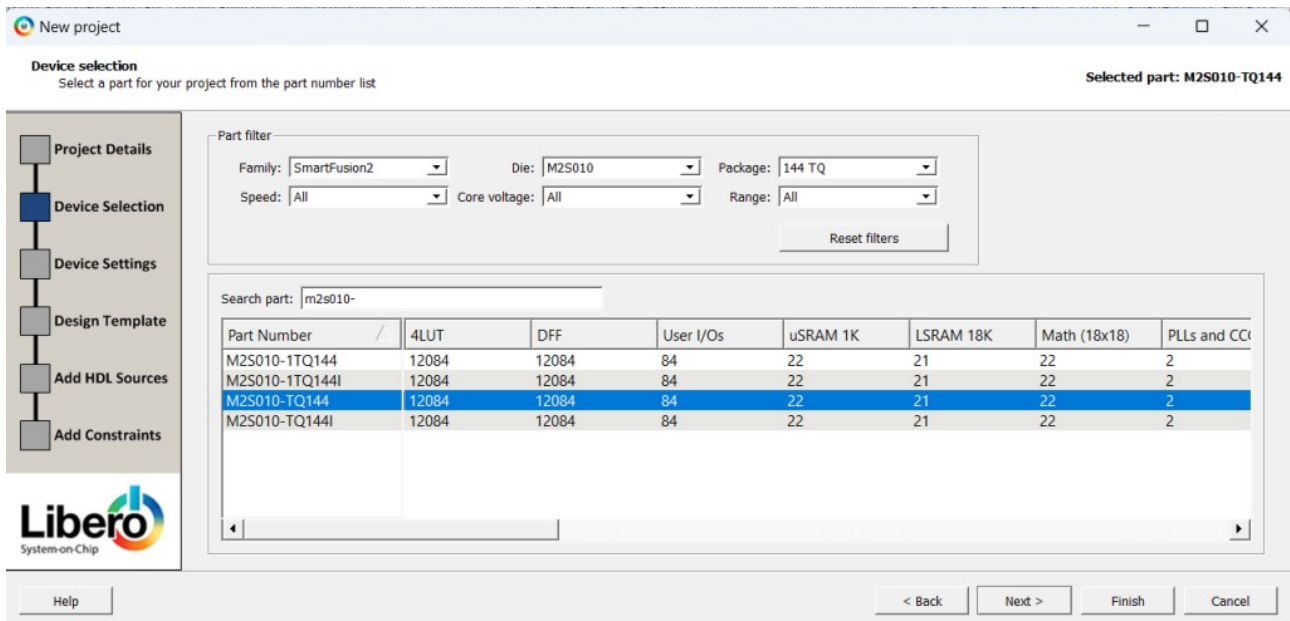
Para ello el proyecto es muy simple, vamos a configurar unos GPIO para que enciendan y apaguen unos leds desde el SoftConsole.

## Proyecto de Libero

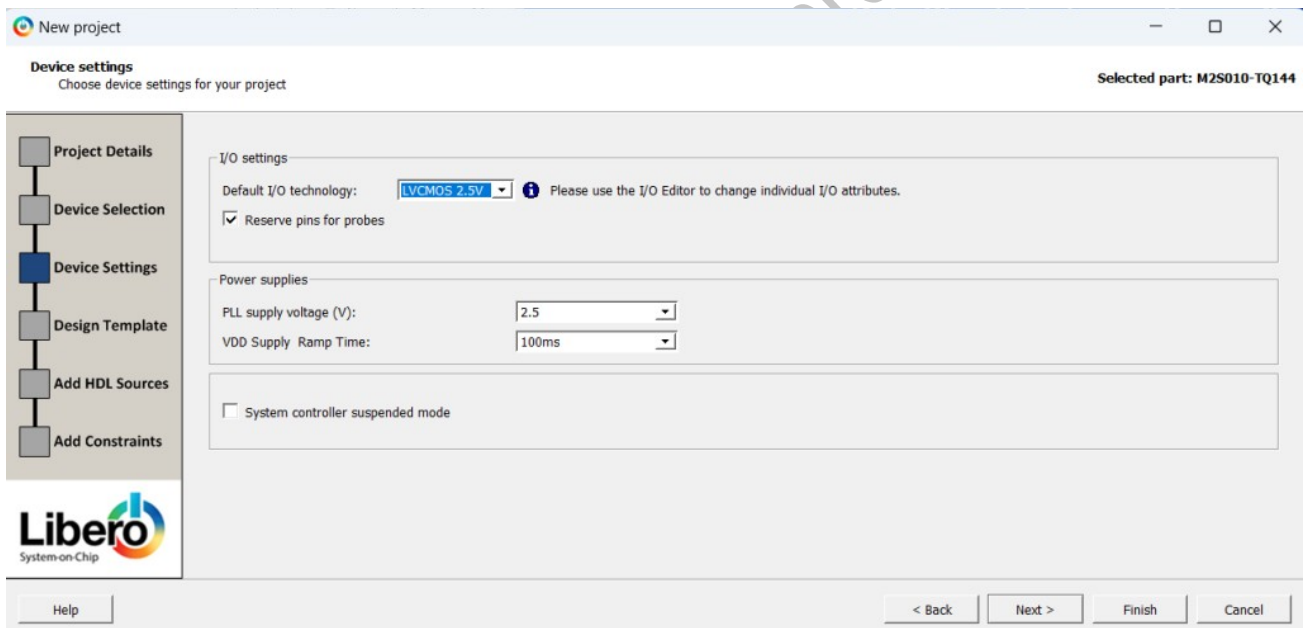
Lo primero es crear un proyecto en Libero.



Después elegimos una SmartFusion2 con la que vayamos a trabajar.

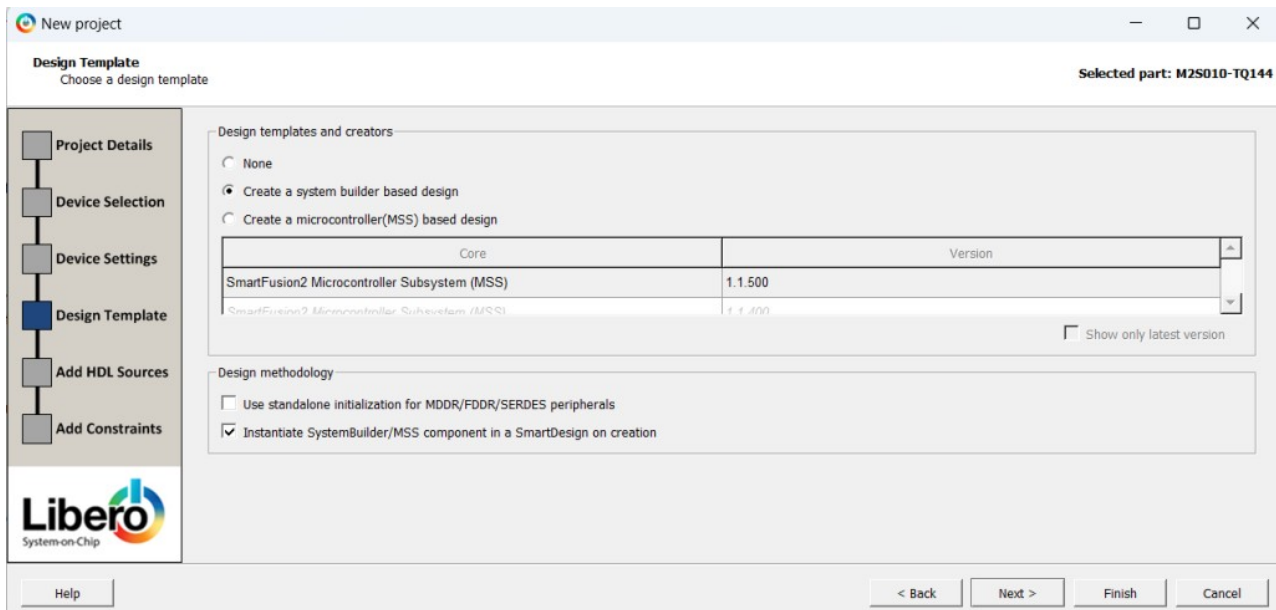


El siguiente paso es la selección de la tensión por defecto de los pines.

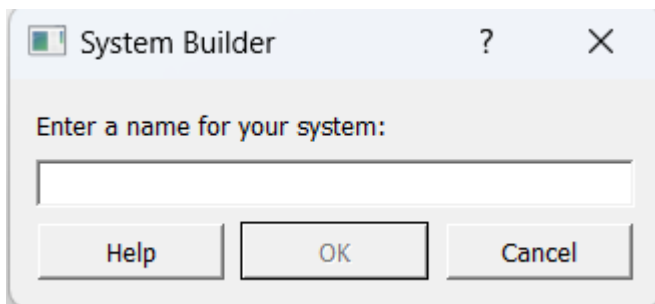


El siguiente paso es el importante, porque va a decidir cómo vamos a trabajar con el SoC. Voy a describir las dos formas que hay:

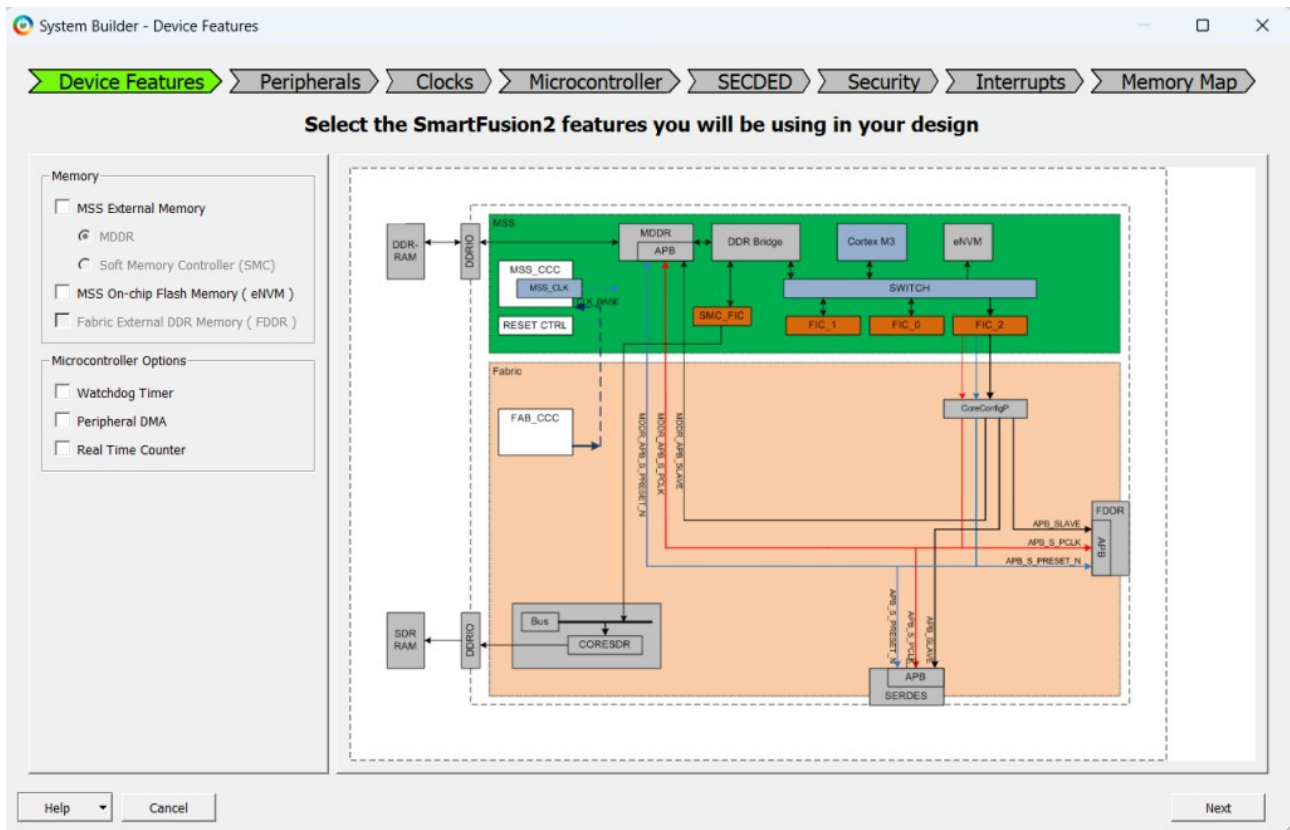
- Este primero lo que va a hacer es preguntarnos cómo queremos activar el microcontrolador.



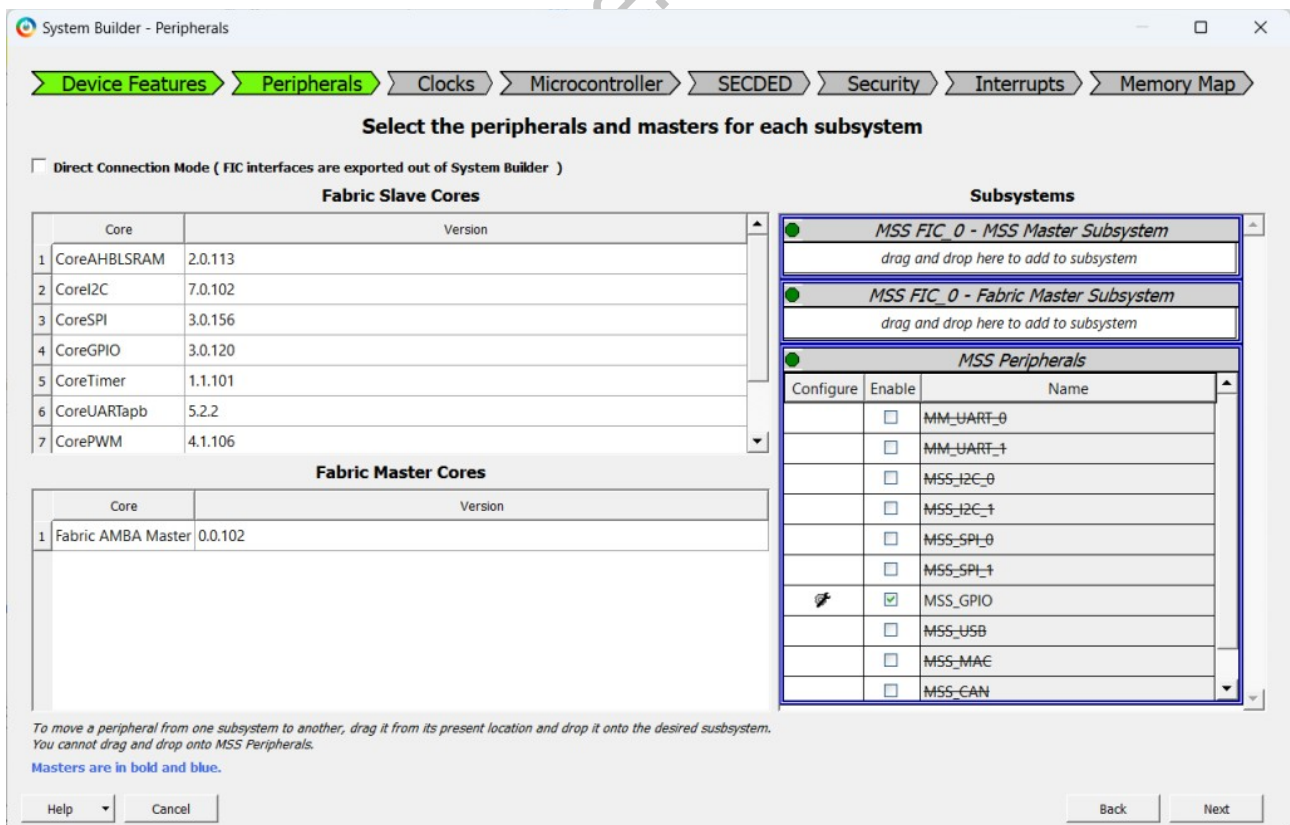
Ahora podemos decirle a Libero que pase termine la configuración. Al terminar la configuración nos pide un nombre para el System Builder.



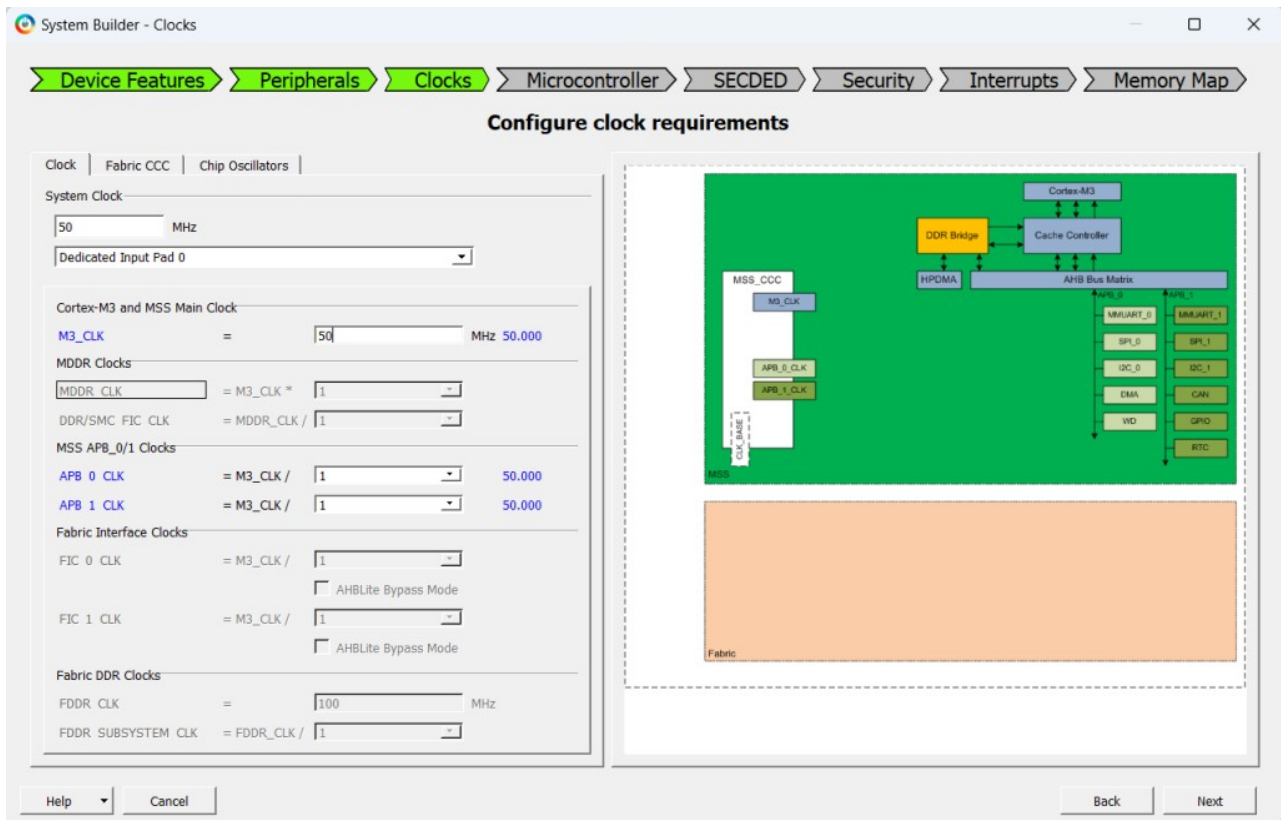
Al darle un nombre comienza la activación. Se nos abre una pestaña donde nos pregunta qué cosas queremos activar.



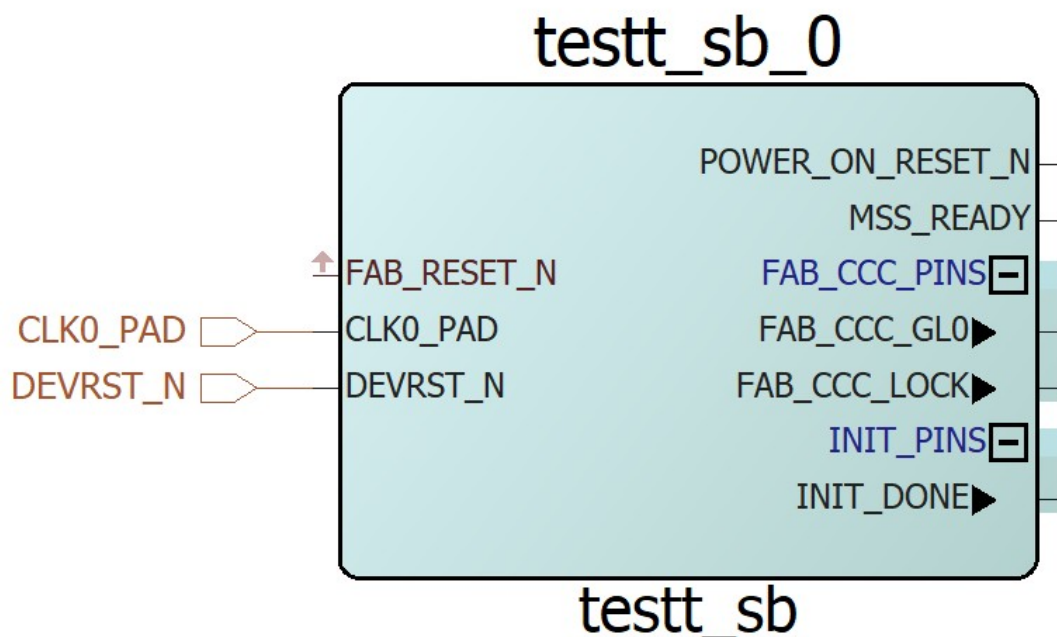
Después los periféricos que queremos activar.



Lo siguiente que pide son los relojes, la placa que yo manejo tiene un reloj de 50MHz.

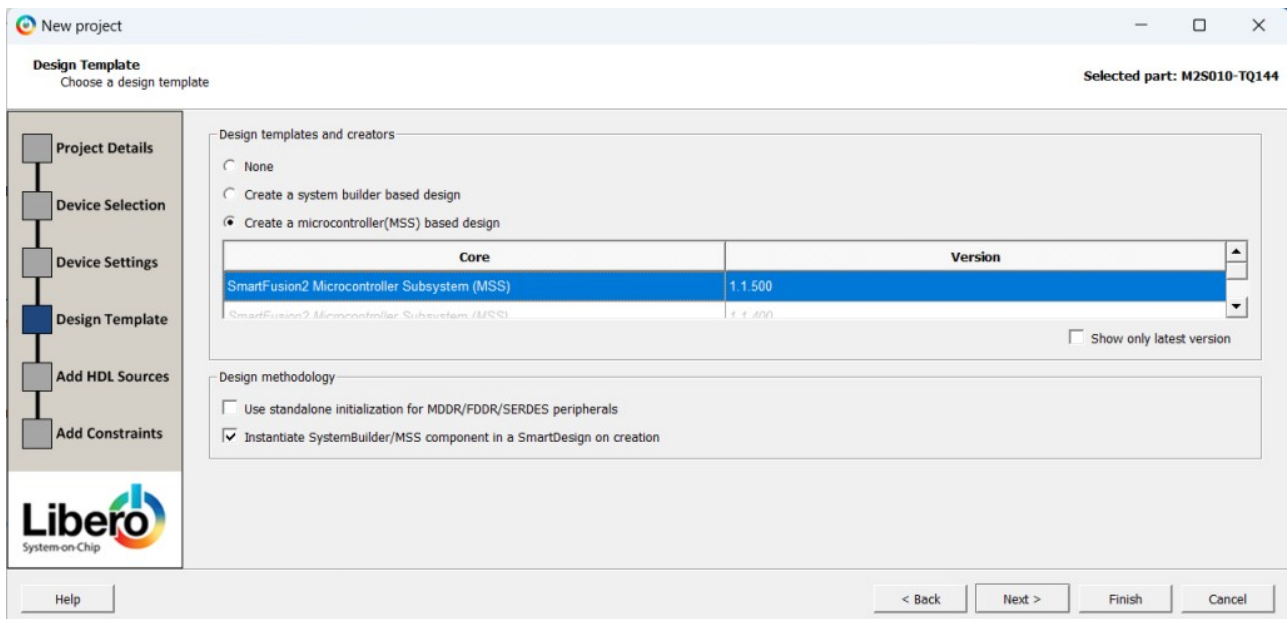


El resto de configuración es sobrante para la aplicación que quiero realizar. Entonces, en Libero se nos crea un bloque como el siguiente, que al abrirlo vemos que nos lleva a las pestañas de configuración.

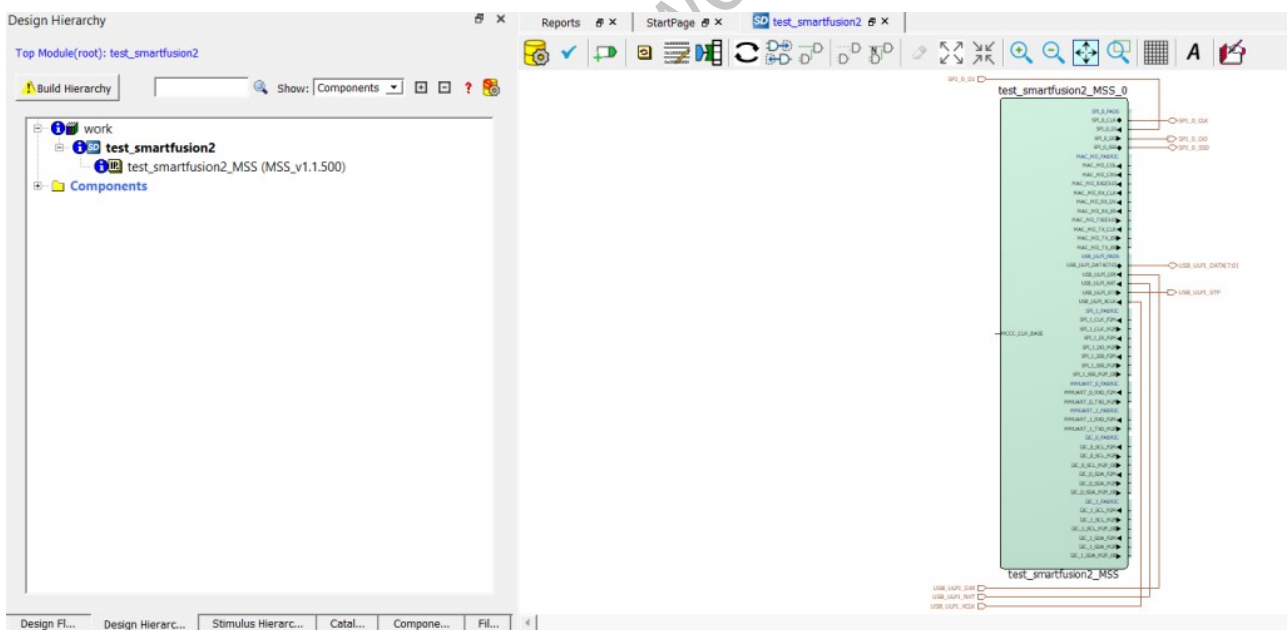


<https://soceame.wordpress.com/2024/12/02/como-crear-un-proyecto-para-una-smartfusion2/>

- Esta segunda forma lo que va a hacer es crear un microcontrolador directamente en Libero con todos los periféricos activados en un SmartDesign. **(recomiendo esta segunda forma)**

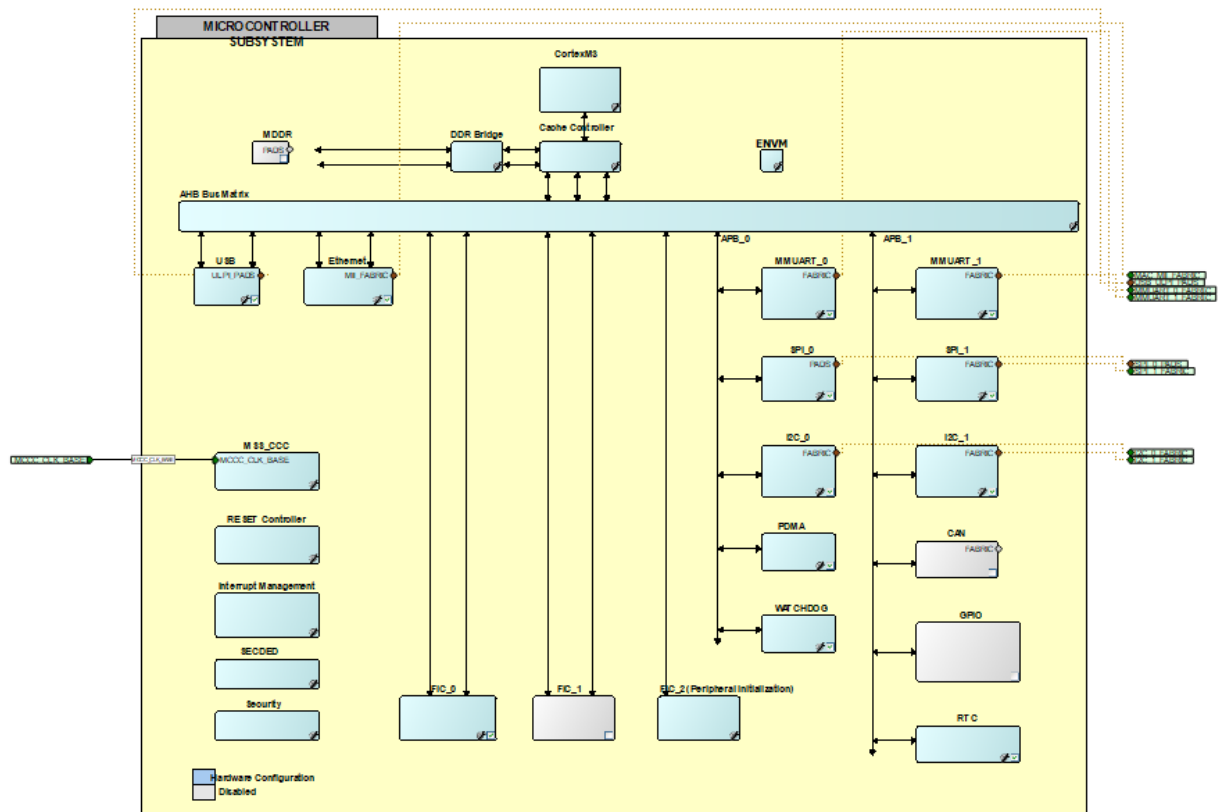


Ahora le podemos dar a finalizar la configuración, y lo que hace es crear automáticamente un bloque en un SmartDesign de Libero.



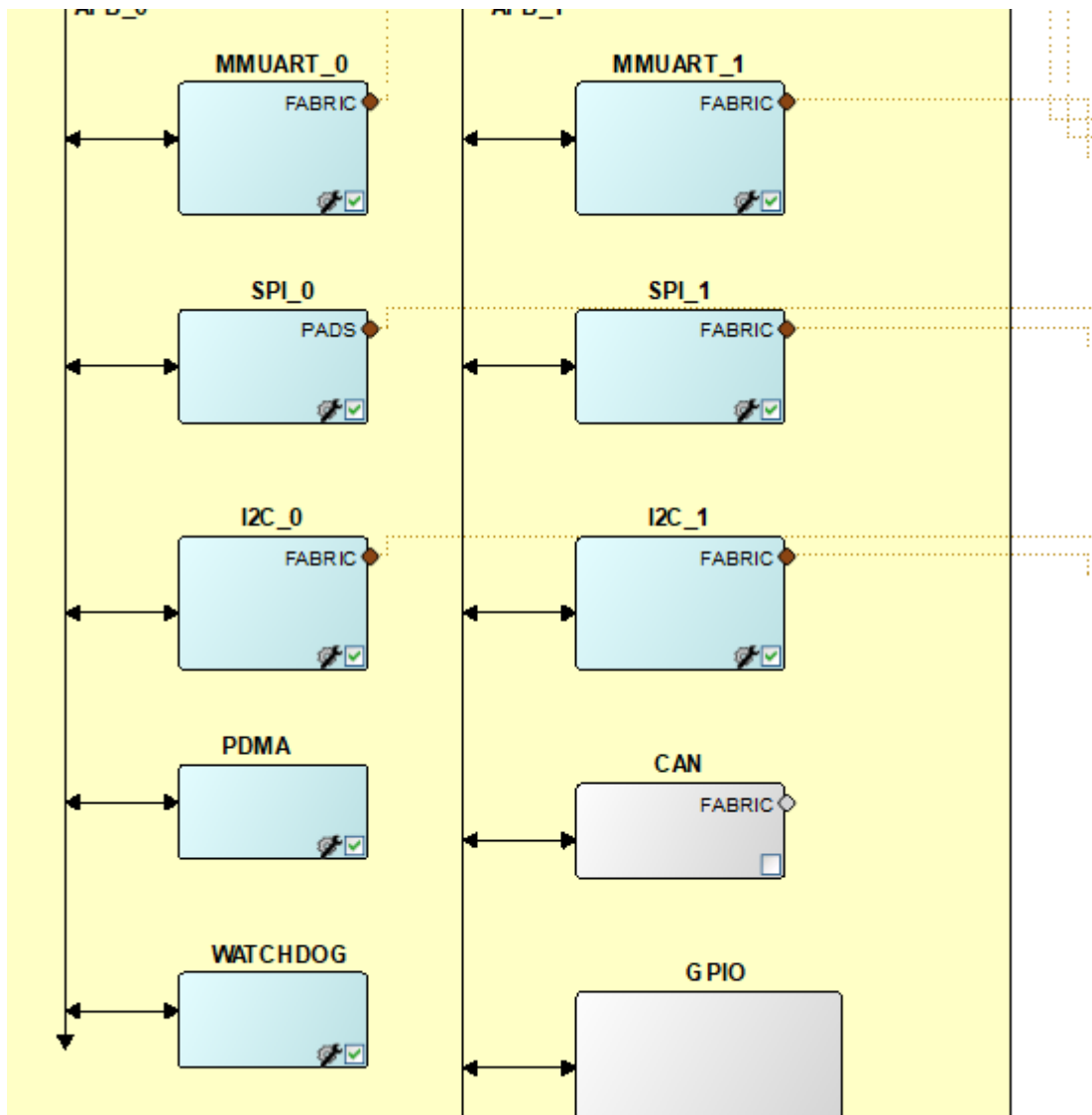
Al abrirlo tenemos una pestaña como la siguiente con casi todos los periféricos activados.



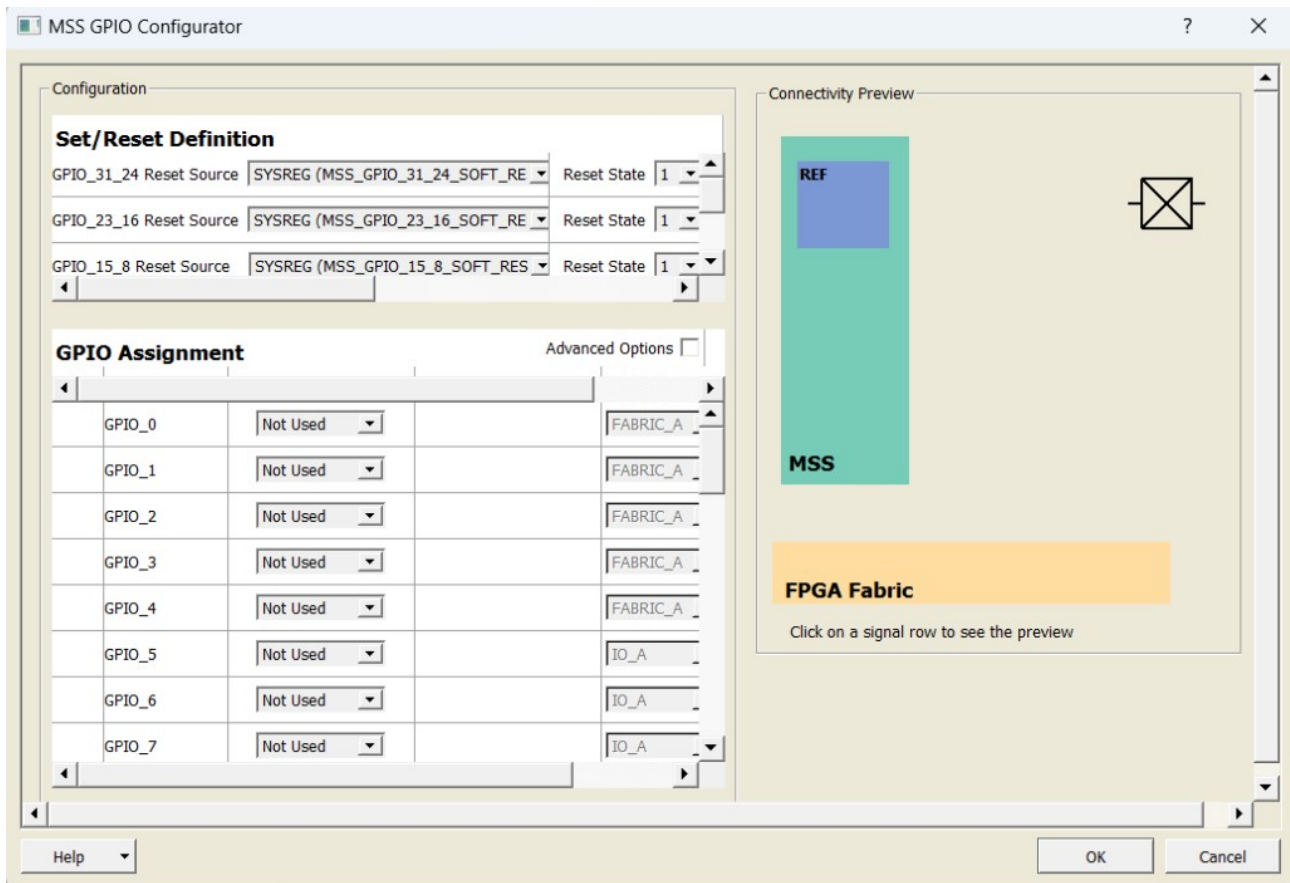


Si no acercamos podemos ver que tiene 2 UART, 2 I2C, 2 SPI, etc.

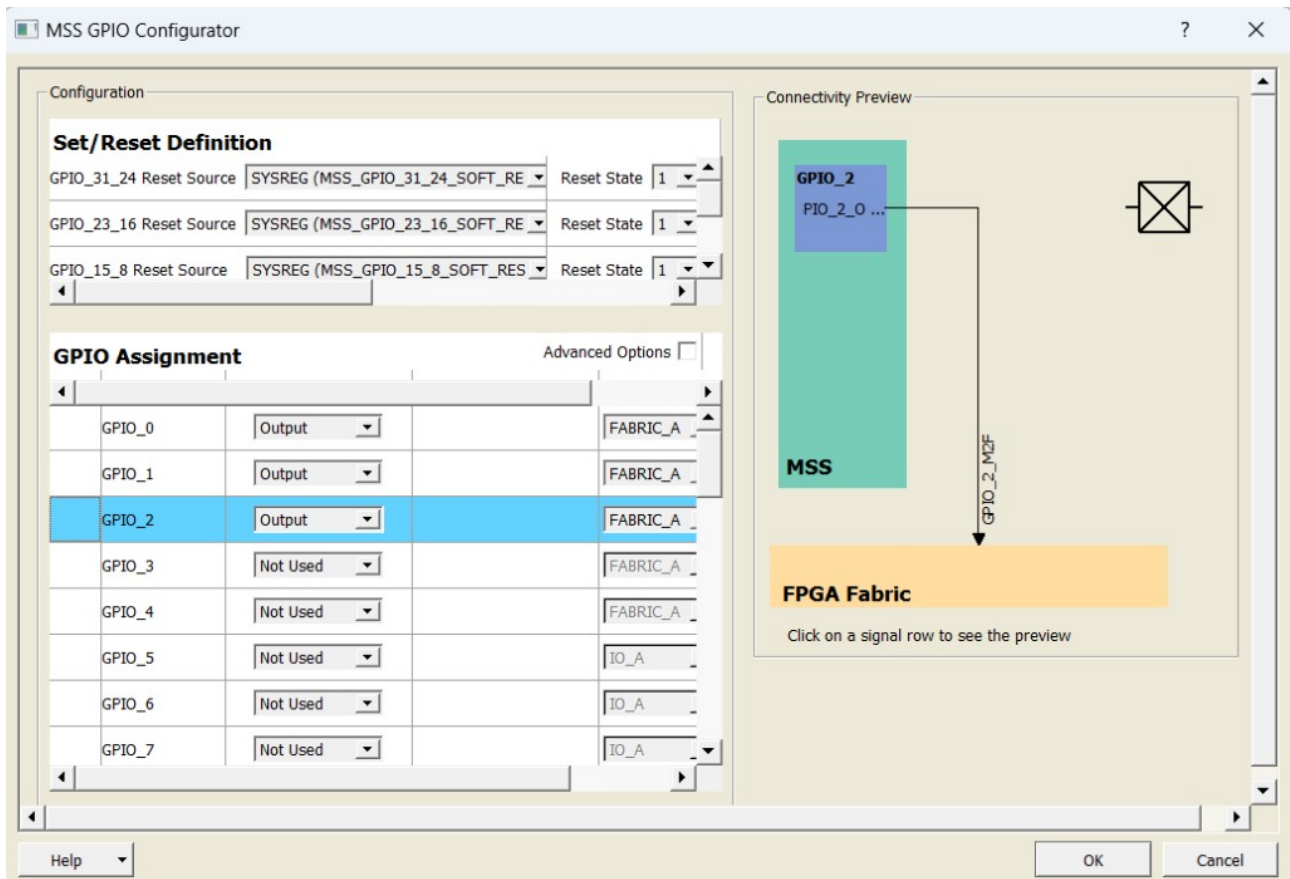




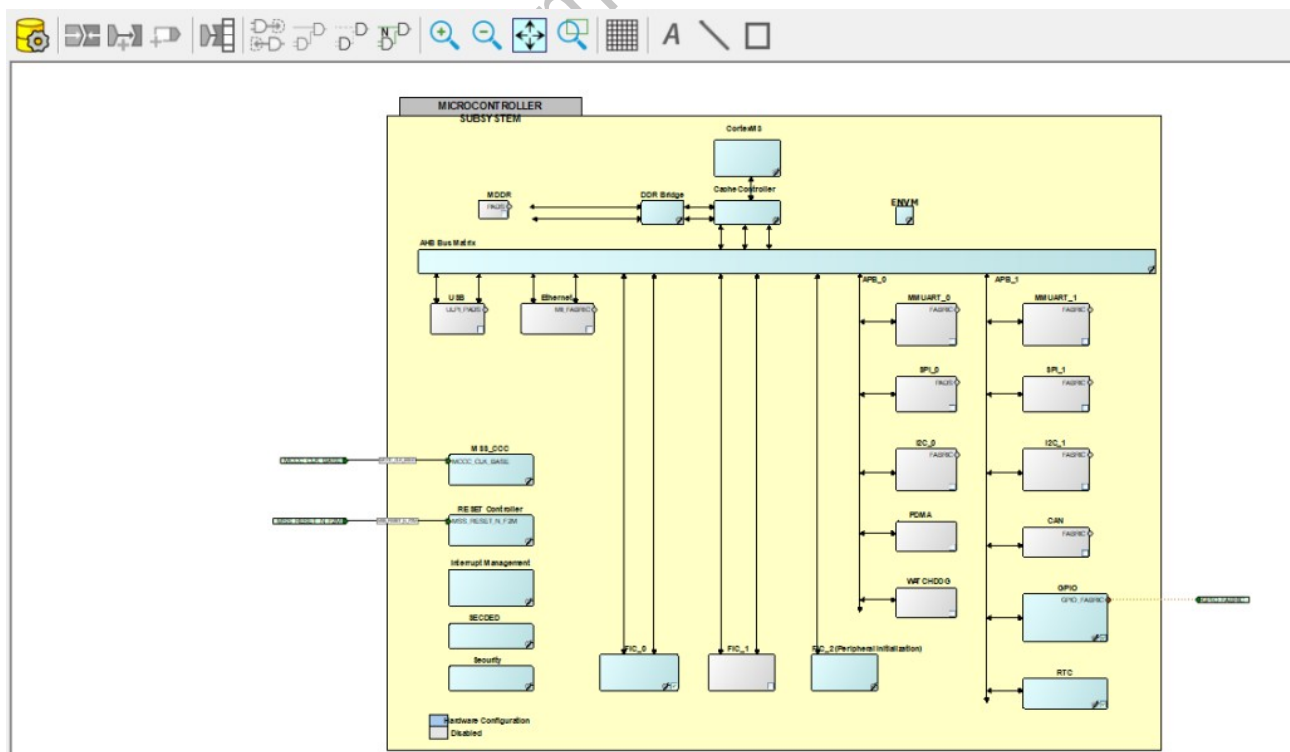
En nuestro caso queremos configurar solo los GPIO, por ello nos metemos en el bloque de los GPIOs que al entrar esta desactivado.



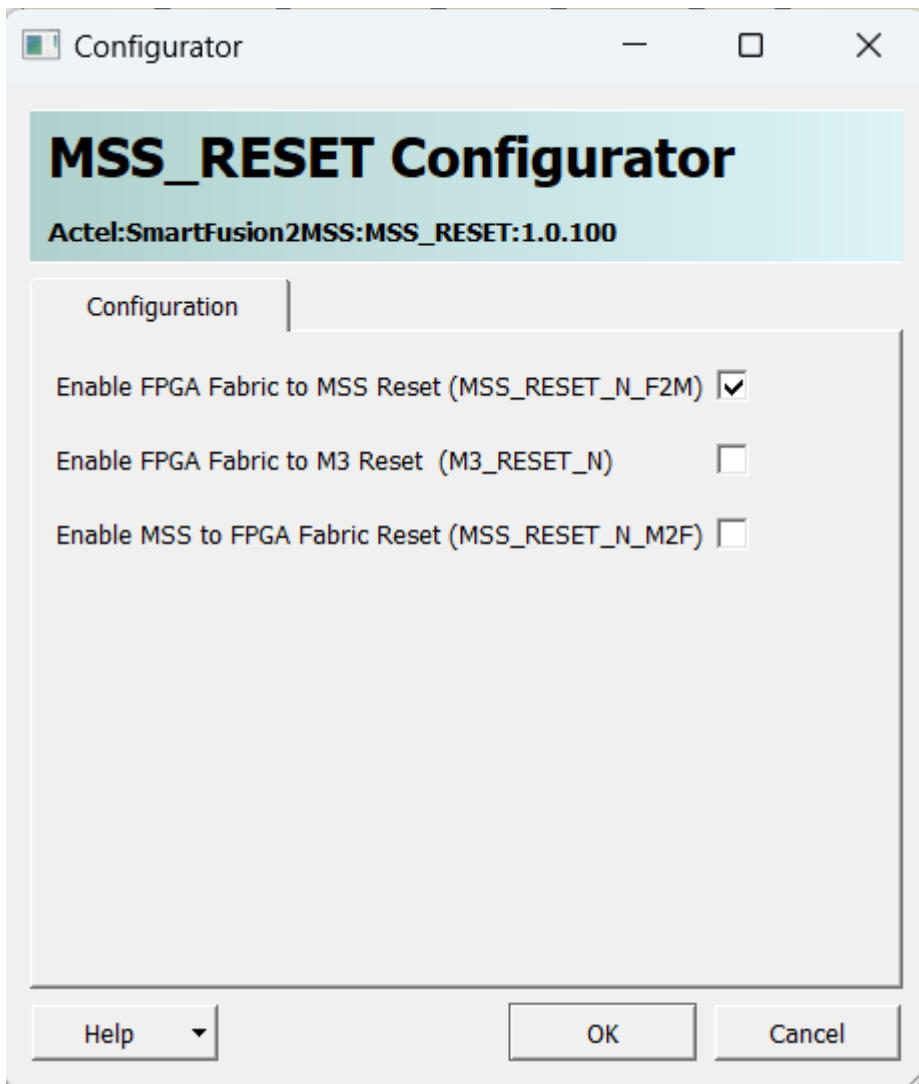
Y activamos los tres primeros GPIOs como salidas y al FABRIC (FABRIC significa que van a la lógica programable de la FPGA, por lo que se puede seleccionar el pin por el que se quiere sacar el GPIO. La otra opción es llevar esa salida a un pin específico del SoC, si estuviera disponible).



Antes de salir desactivamos todo, pero dejamos activado el reloj y el reset.

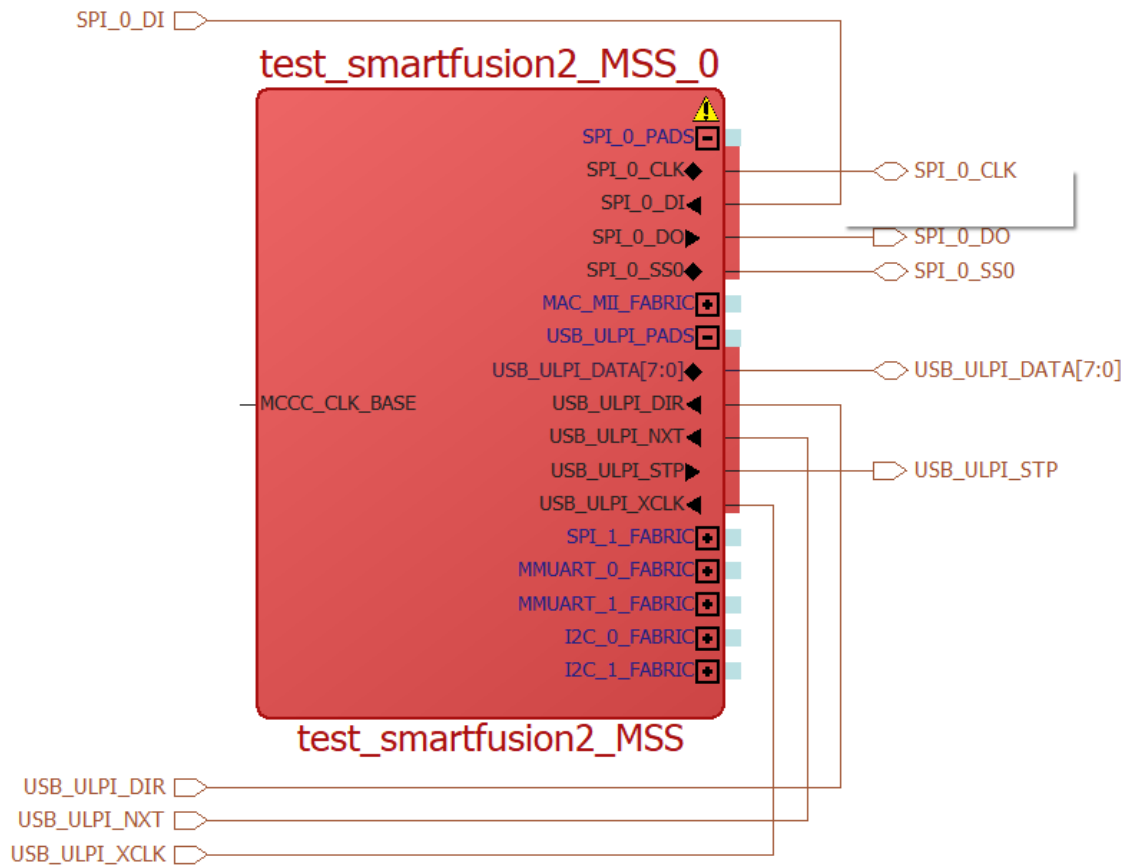


Para activar el reset elegimos la opción de que el reset vaya de la FPGA al microcontrolador (*FABRIC to MSS*).

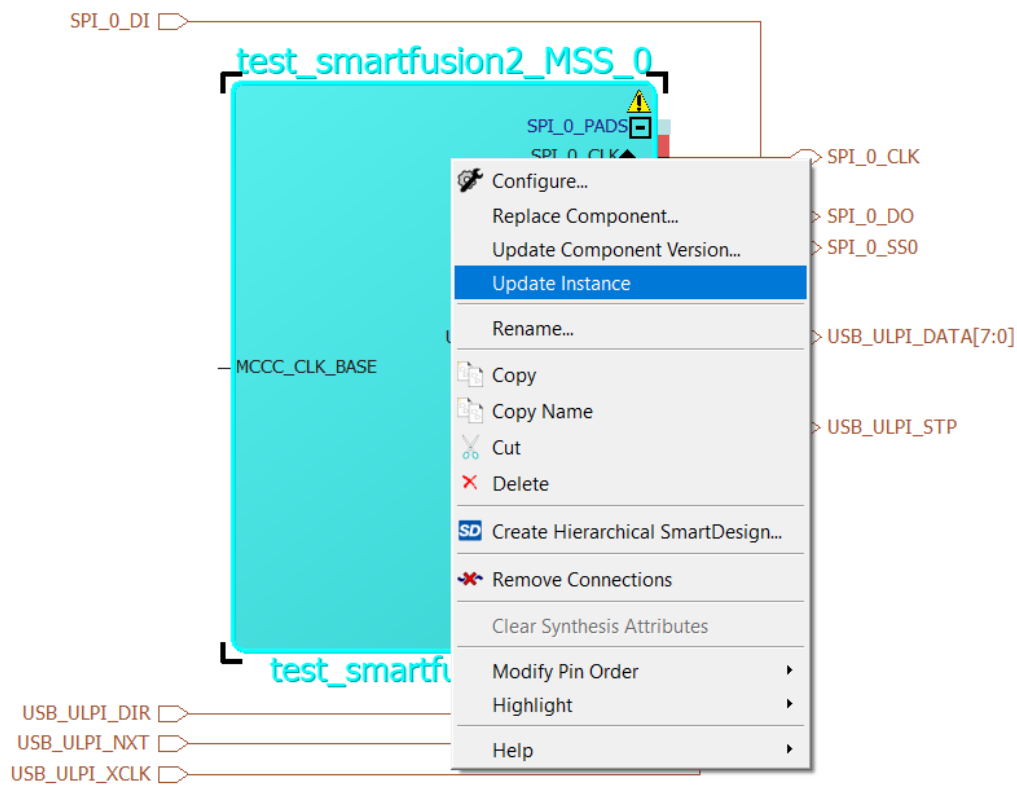


Una vez tengamos todo configurado le damos al símbolo amarillo con el engranaje.

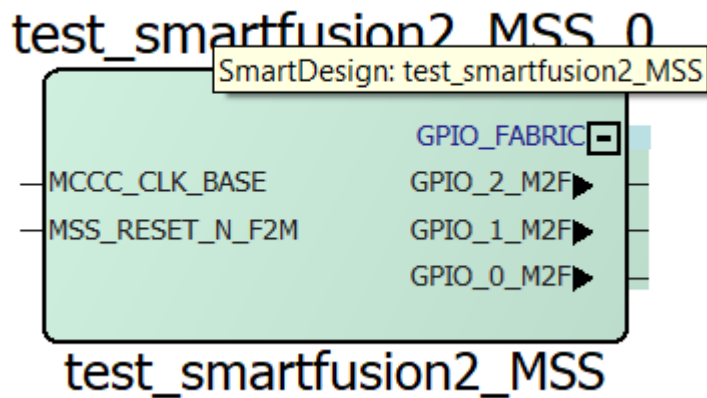
Si volvemos a Libero vemos el bloque marcado en rojo.



Para actualizar el bloque le damos clic derecho y a *Update Instance*.

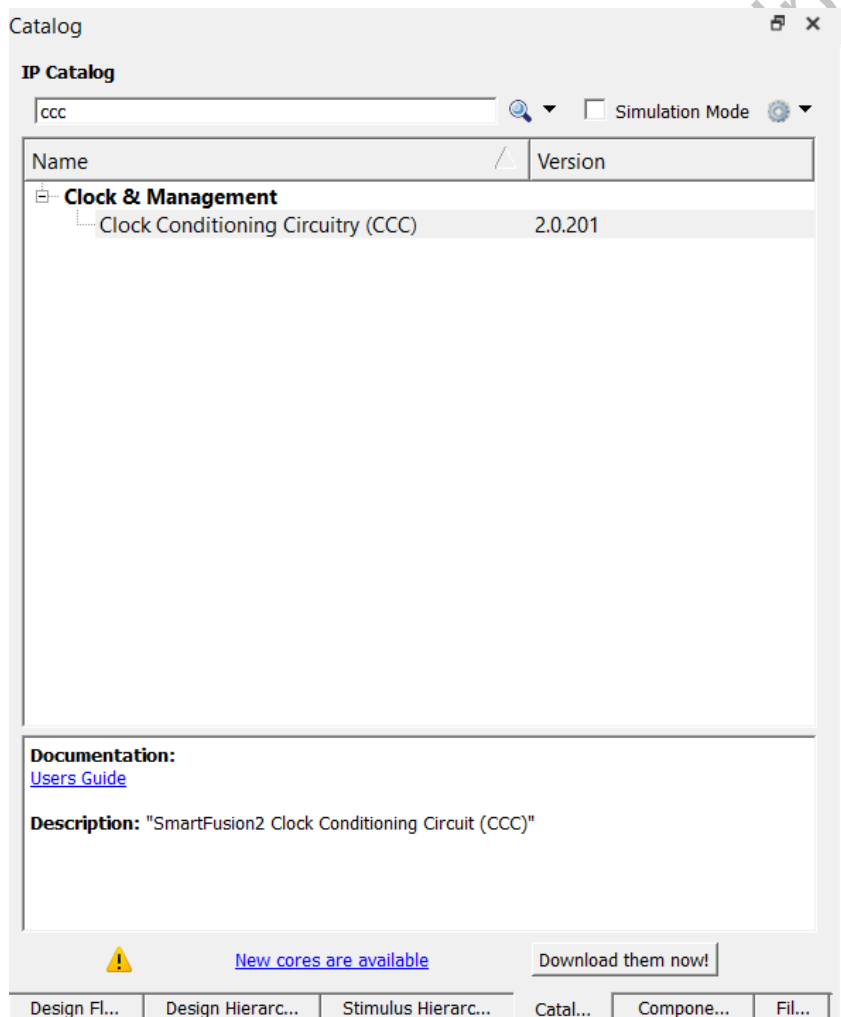


Al actualizarse queda algo así.

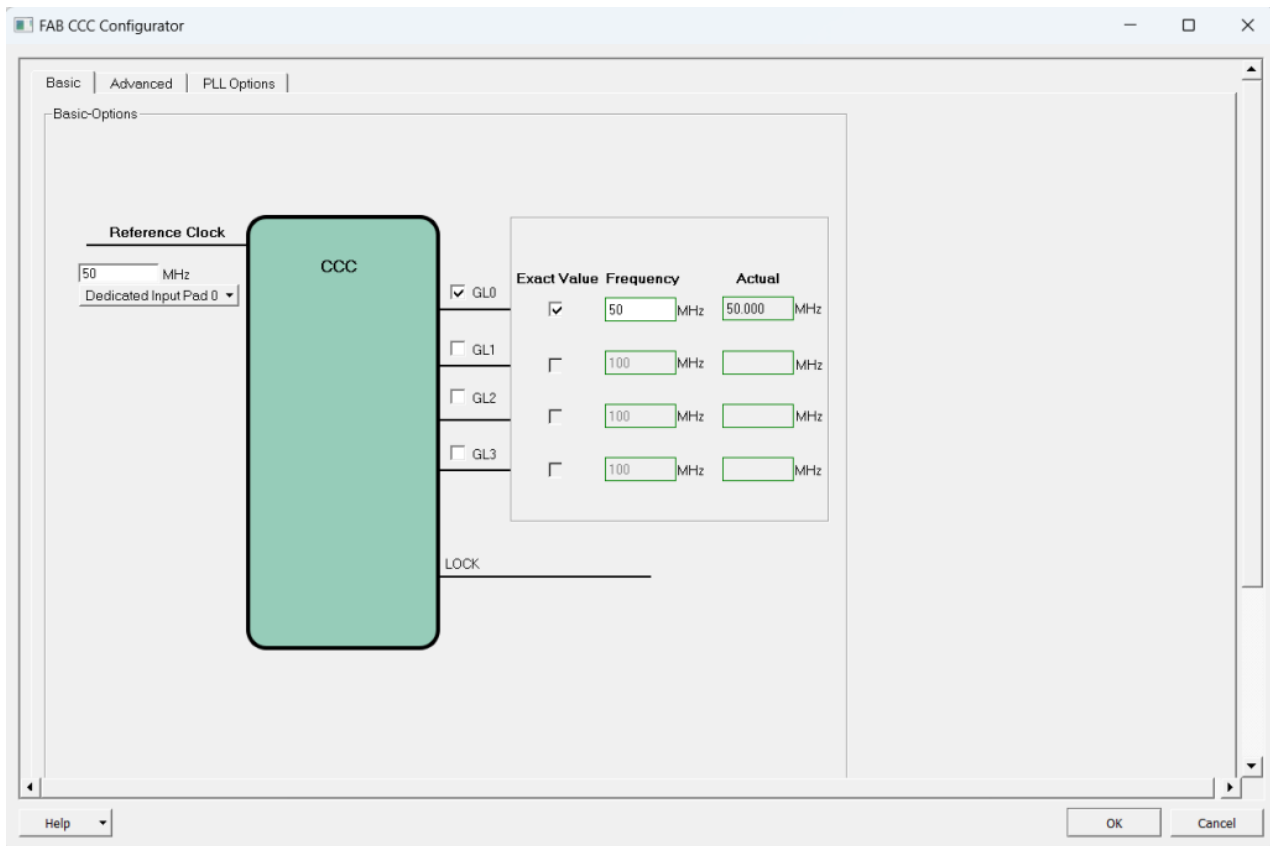


Ahora nos falta el reloj y el reset.

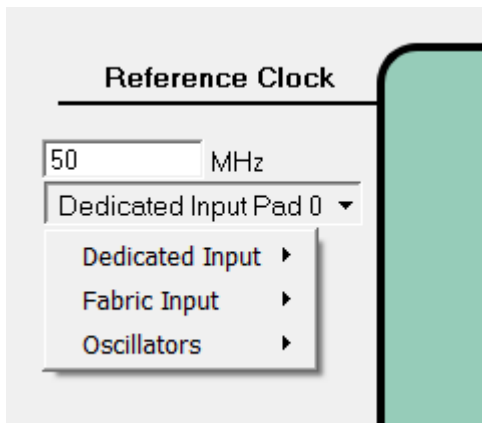
Para el reloj utilizamos un PLL interno de Libero llamado *CCC*, para buscar este bloque se hace desde la pestaña *Catalog*, si no está descargado te permite descargarlo.



A este PLL le decimos que tenemos una entrada de 50MHz y que la salida sea de 50MHz.

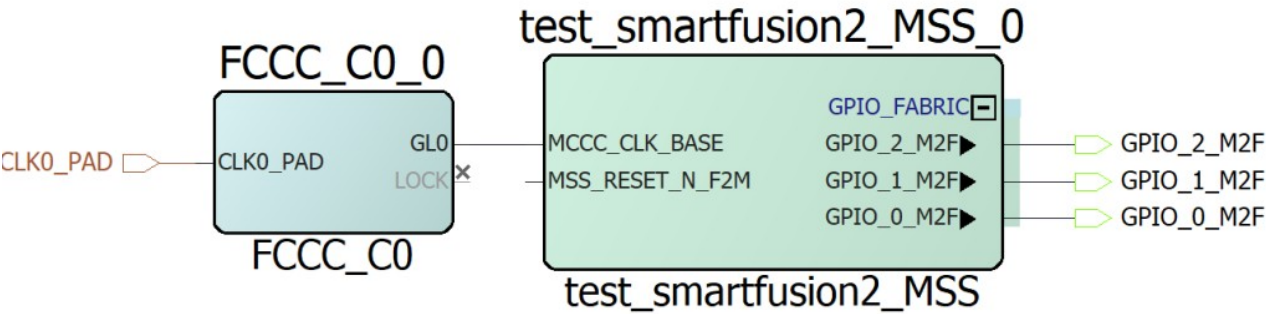


Existen diferentes formas de introducir un reloj al PLL que explicaré en otra entrada.



Al añadir el reloj unimos el reloj con la entrada de reloj del MSS. El pin de *LOCK* lo dejamos al sin utilizar.

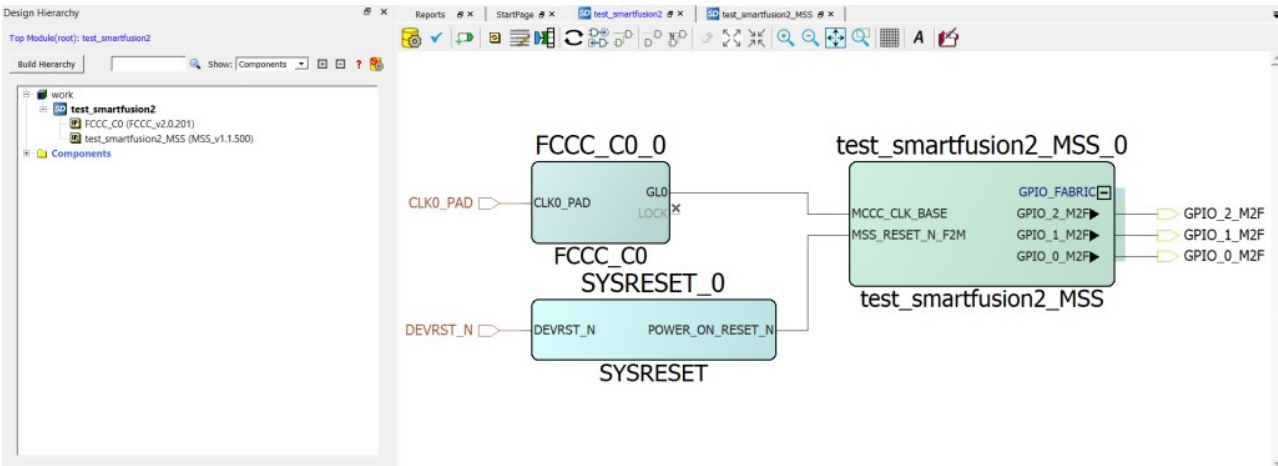




Ahora para el reset utilizamos el bloque `SYSRESET` que da la pestaña de *Catalog*.

Name	Version
<b>Macro Library</b>	
SYSCTRL_RESET_STATUS	1.0
SYSRESET	1.0
<b>Peripherals</b>	
CoreResetP	7.1.100
CoreReset_PF	2.1.100
CoreSF2Reset	3.0.100
CoreWatchdog	1.1.101
<b>Solutions-MotorControl</b>	
Rate Limiter	4.2.0
Rate Limiter	4.1.0

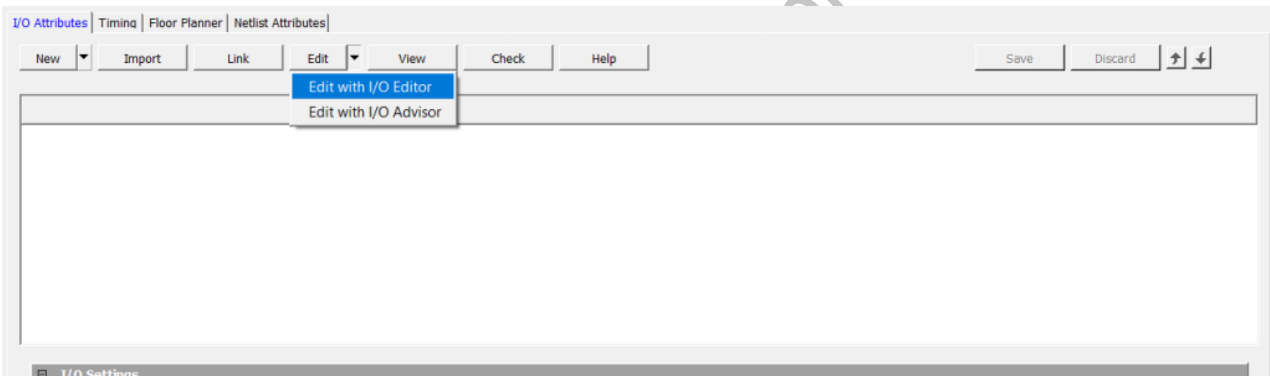
Al añadirlo nos queda algo así.



Para terminar la configuración le damos al icono amarillo con el engranaje.

(A partir de aquí las dos configuraciones del SoC se juntan en un solo desarrollo)

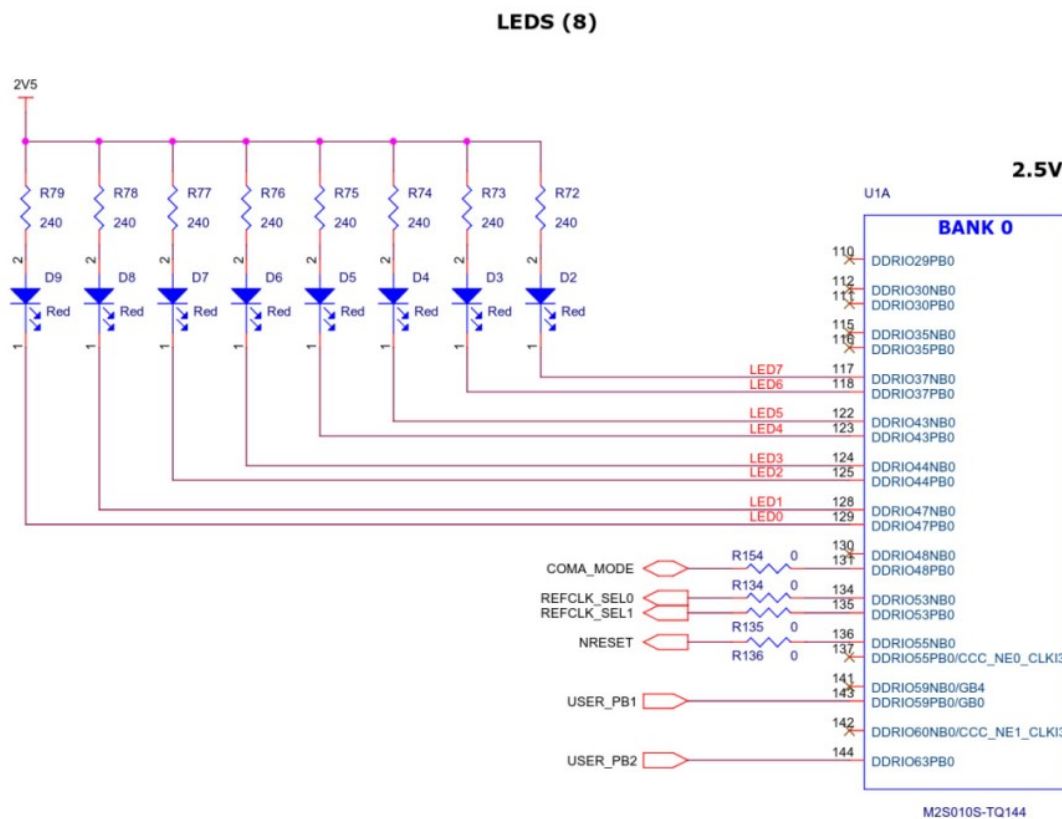
Ahora sintetizamos el modelo y configuramos los pines en el *Manage Constraints*, después en la opción de *Edit* le damos a aditar con *I/O Editor*.



Cuando se nos abre la pestaña, el único pin que nos aparece colocado es el pin del reset, porque es el pin de reset de la placa, el *DEVIRST\_N* es un pin específico y de un solo uso.

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Rank Name	I/O state in Flash*Freeze mode	Resistor Pull
1	CLK0_PAD	INPUT	LVC MOS25		<input type="checkbox"/>	INBUF	--	TRISTATE	None
2	DEVIRST_N	INPUT	--	72	<input checked="" type="checkbox"/>	SYSRESET	--	--	--
3	GPIO_0_M2F	OUTPUT	LVC MOS25		<input type="checkbox"/>	OUTBUF	--	TRISTATE	None
4	GPIO_1_M2F	OUTPUT	LVC MOS25		<input type="checkbox"/>	OUTBUF	--	TRISTATE	None
5	GPIO_2_M2F	OUTPUT	LVC MOS25		<input type="checkbox"/>	OUTBUF	--	TRISTATE	None

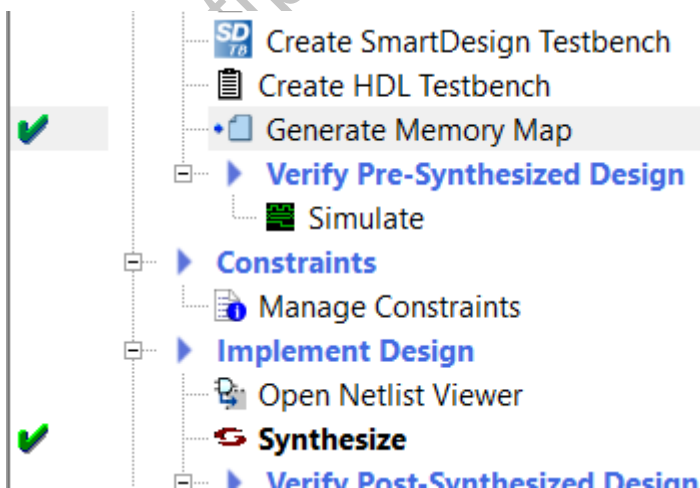
Los leds que queremos configurar son los siguientes.



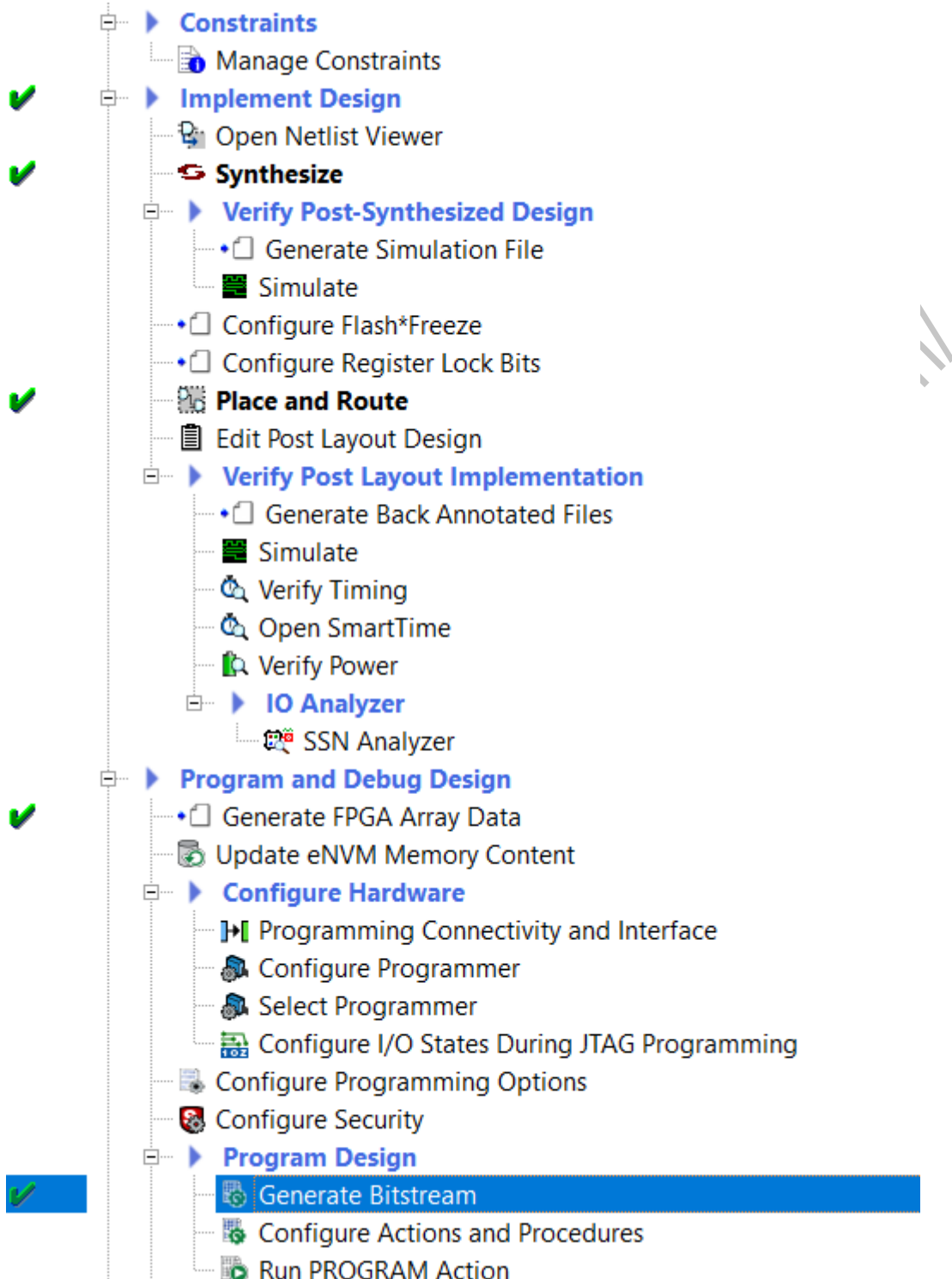
Para ello actualizamos la tabla para que coincidan.

Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode	Resistor Pull
CLK0_PAD	INPUT	LVC MOS25	23	<input checked="" type="checkbox"/>	INBUF	Bank6	TRISTATE	None
DEVIRST_N	INPUT	--	72	<input checked="" type="checkbox"/>	SYSRESET	--	--	--
GPIO_0_M2F	OUTPUT	LVC MOS25	129	<input checked="" type="checkbox"/>	OUTBUF	Bank0	TRISTATE	None
GPIO_1_M2F	OUTPUT	LVC MOS25	128	<input checked="" type="checkbox"/>	OUTBUF	Bank0	TRISTATE	None
GPIO_2_M2F	OUTPUT	LVC MOS25	125	<input checked="" type="checkbox"/>	OUTBUF	Bank0	TRISTATE	None

Una vez tenemos los pines establecidos, ya podemos crear el bitstream. Pero primero generamos el mapa de memoria que se va a utilizar.



Después, ya podemos continuar con el bitstream.



Si queremos depurar el SoC cuando nos vayamos a Libero es necesario dejar el SoC grabado, que al ser de memoria Flash se queda grabado con el bitstream, por lo que no hay perdida posible.

El siguiente paso antes de irse al SoftConsole es conseguir los drivers de los periféricos que se quieren utilizar. Para ello accedemos a la opción *Configure Firmware Cores*.



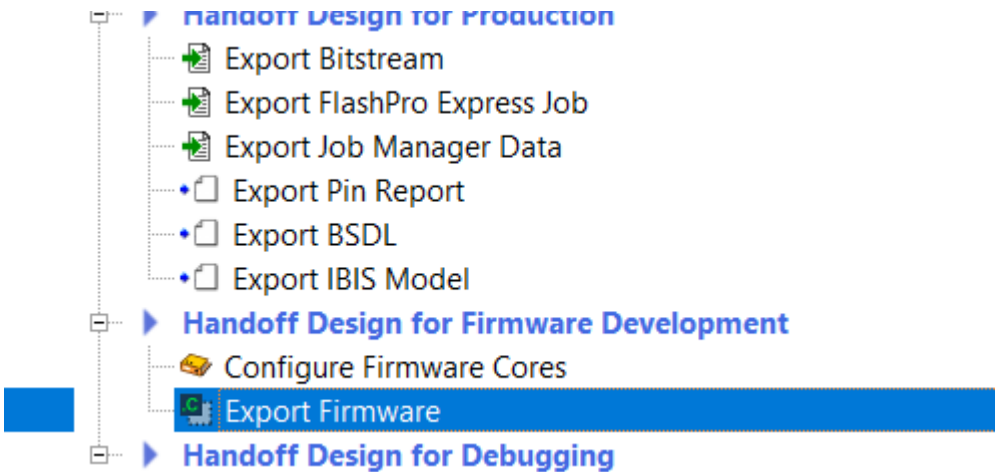
Al abrirse se puede ver que no hay ningún driver instalado. Para descargarlos hay dos opciones: marcar la casilla de la izquierda para instalar uno a uno el driver o darle al símbolo de descarga de arriba.

	Generate	Instance Name	Core Type	Version	Compatible Hardware Instance
1	<input type="checkbox"/>		SmartFusion2_CMSIS	2.3.1	test_smartfusion2_MSS
2	<input type="checkbox"/>		SmartFusion2_MSS_GPIO_Driver	2.1.1	test_smartfusion2_MSS:GPIO
3	<input type="checkbox"/>		SmartFusion2_MSS_HPDMMA_Driver	2.2.1	test_smartfusion2_MSS
4	<input type="checkbox"/>		SmartFusion2_MSS_NVM_Driver	2.5.1	test_smartfusion2_MSS
5	<input type="checkbox"/>		SmartFusion2_MSS_RTC_Driver	2.2.1	test_smartfusion2_MSS:RTC
6	<input type="checkbox"/>		SmartFusion2_MSS_System_Services_Driver	2.9.1	test_smartfusion2_MSS
7	<input type="checkbox"/>		SmartFusion2_MSS_Timer_Driver	2.2.1	test_smartfusion2_MSS

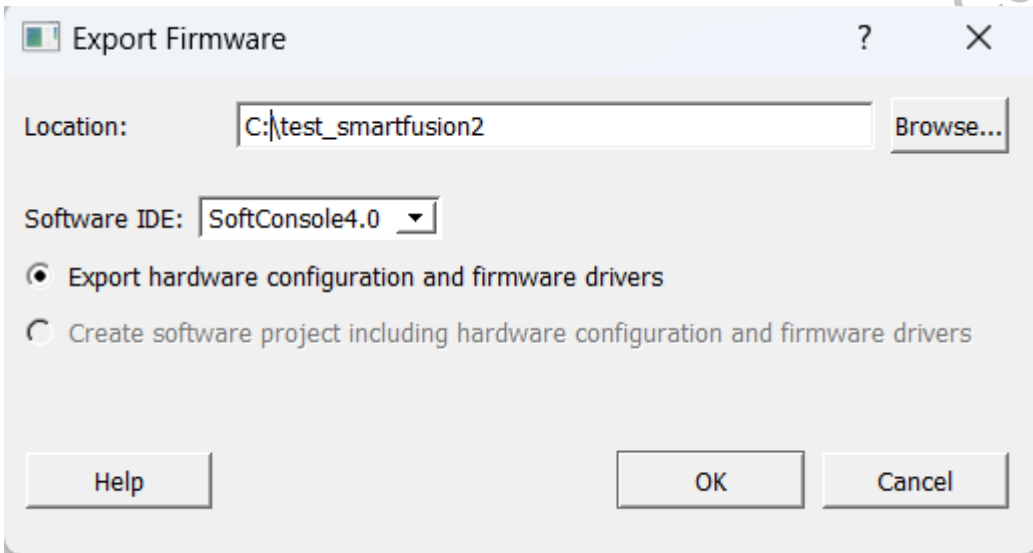
Una vez descargados, podemos pasar al paso siguiente.

	Generate	Instance Name	Core Type	Version	Compatible Hardware Instance
1	<input checked="" type="checkbox"/>	SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.1	test_smartfusion2_MSS
2	<input checked="" type="checkbox"/>	SmartFusion2_MSS_GPIO_Driver_0	SmartFusion2_MSS_GPIO_Driver	2.1.1	test_smartfusion2_MSS:GPIO
3	<input checked="" type="checkbox"/>	SmartFusion2_MSS_HPDMMA_Driver_0	SmartFusion2_MSS_HPDMMA_Driver	2.2.1	test_smartfusion2_MSS
4	<input checked="" type="checkbox"/>	SmartFusion2_MSS_NVM_Driver_0	SmartFusion2_MSS_NVM_Driver	2.5.1	test_smartfusion2_MSS
5	<input checked="" type="checkbox"/>	SmartFusion2_MSS_RTC_Driver_0	SmartFusion2_MSS_RTC_Driver	2.2.1	test_smartfusion2_MSS:RTC
6	<input checked="" type="checkbox"/>	SmartFusion2_MSS_System_Services_Driver_0	SmartFusion2_MSS_System_Services_Driver	2.9.1	test_smartfusion2_MSS
7	<input checked="" type="checkbox"/>	SmartFusion2_MSS_Timer_Driver_0	SmartFusion2_MSS_Timer_Driver	2.2.1	test_smartfusion2_MSS

Ahora exportamos los drivers, para ello marcamos la opción *Export Firmware*.



Que nos pregunta donde queremos bajarlos.



Si abrimos la carpeta *firmware* que nos ha generado tiene los drivers para nuestro proyecto.

Nombre	Fecha de modificación	Tipo	Tamaño
CMSIS	01/12/2024 15:58	Carpeta de archivos	
drivers	01/12/2024 15:59	Carpeta de archivos	
drivers_config	01/12/2024 15:58	Carpeta de archivos	
filelist	01/12/2024 15:59	Carpeta de archivos	
hal	01/12/2024 15:58	Carpeta de archivos	
README.txt	01/12/2024 15:59	Documento de tex...	1 KB
test_smartfusion2_cmsis_svd.xml	01/12/2024 15:59	Archivo XML	120 KB
test_smartfusion2_hw_platform.h	01/12/2024 15:59	Archivo H	1 KB

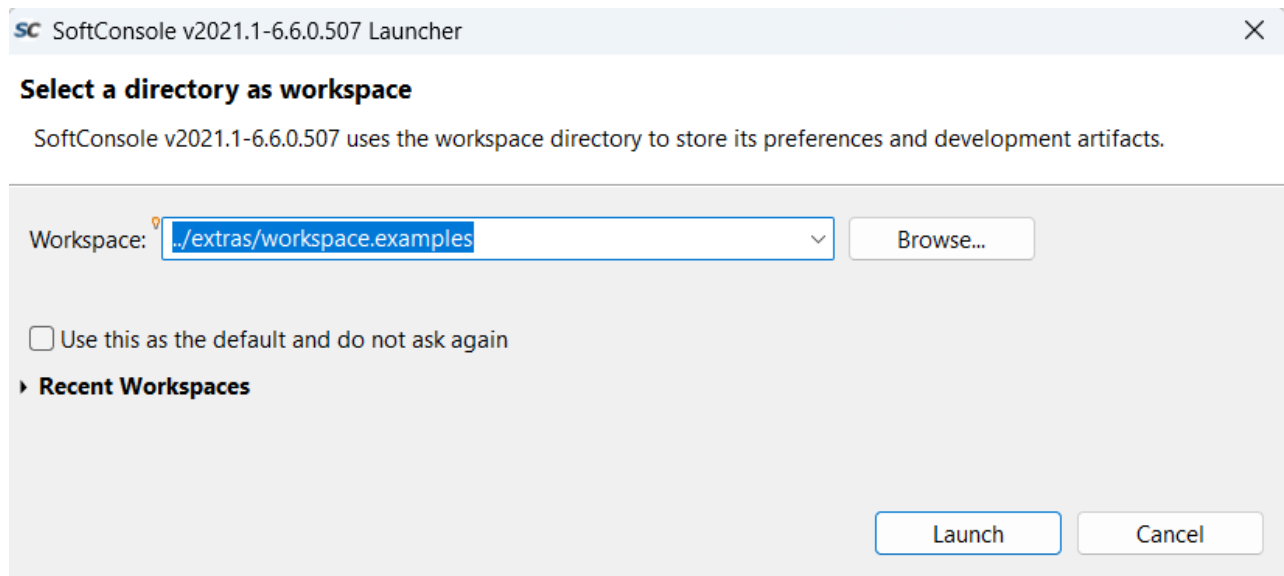
Con esto el proyecto de Libero queda concluido, ahora vamos al SoftConsole.

## Proyecto de SoftConsole

Debido a que queremos crear un proyecto para una SmartFusion2 que lleva dentro un Cortex-M3, es necesario tener instalado una versión de SoftConsole anterior a 2022.

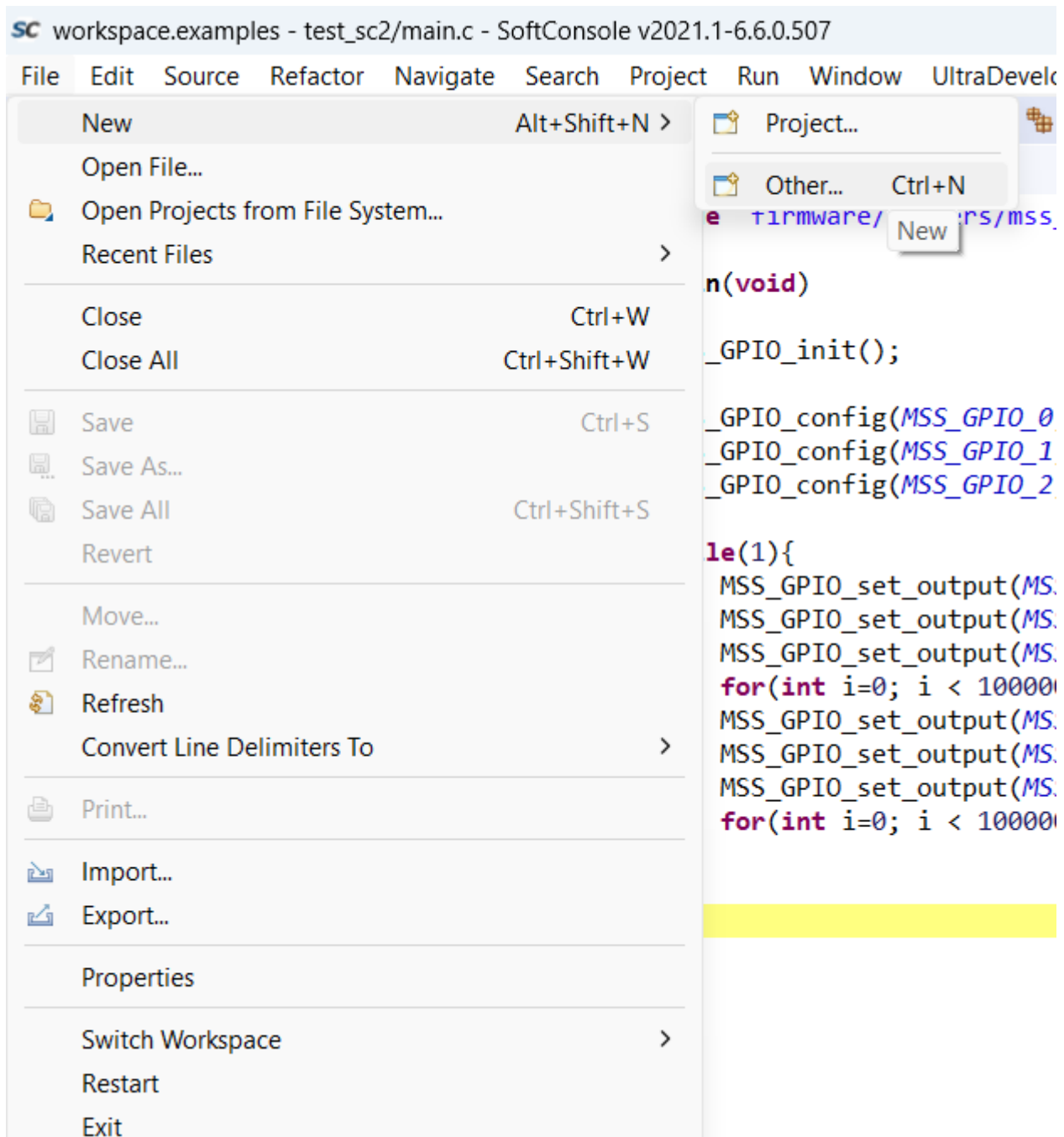
**NOTA:** el SoftConsole está basado en Eclipse, por lo que se asemeja bastante a Vitis o al Nios II editor.

Para empezar, primero abrimos el SoftConsole, que nos pregunta por dónde queremos abrir el proyecto.

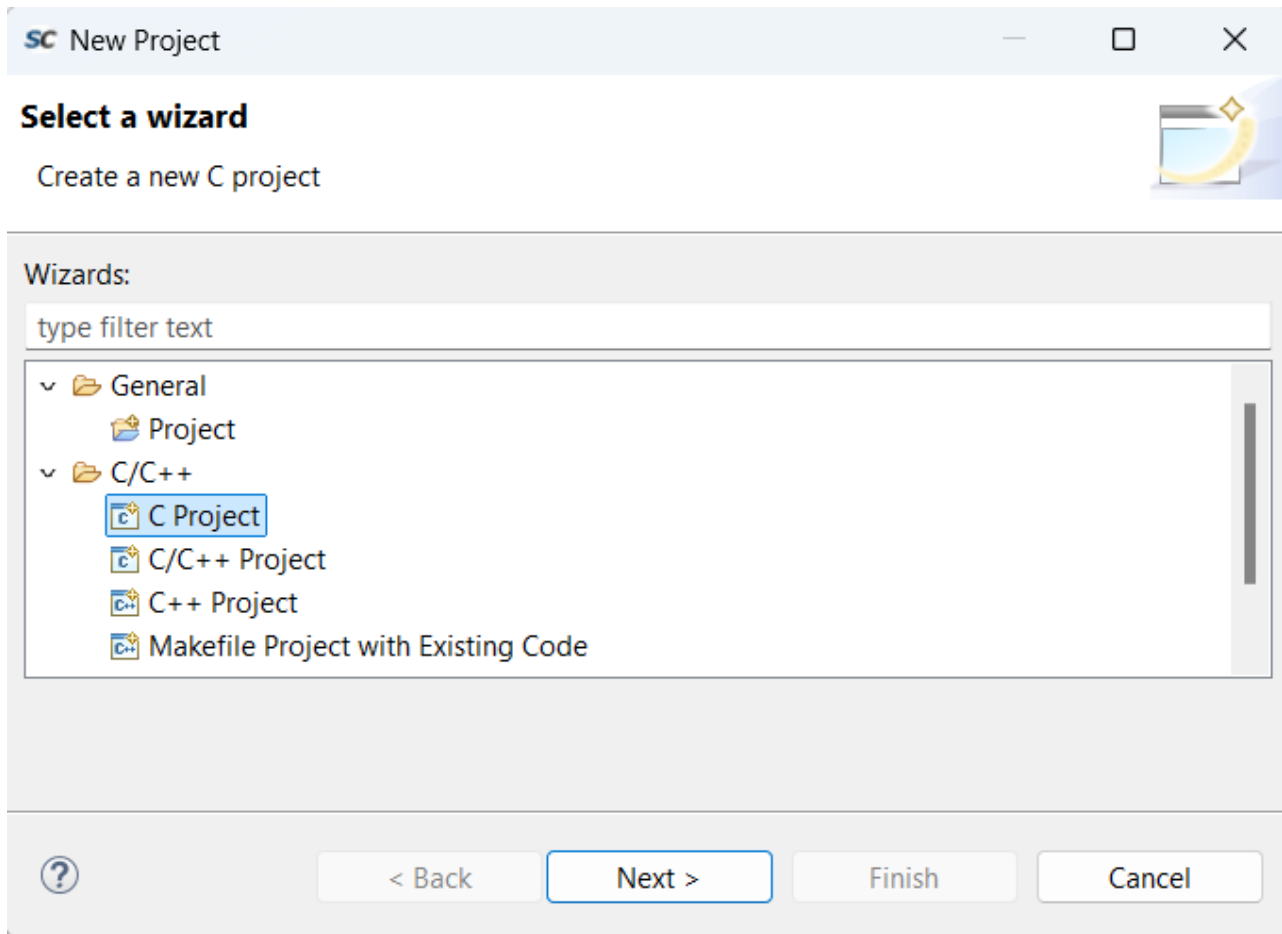


Una vez abierto, creamos una proyecto.





En nuestro caso creamos un proyecto en C, esto significa que al crearlo nos crea la carpeta *includes*.



Ahora creamos un proyecto vacío basado en un ARM.

**C Project**

Create C project of selected type

Project name:

☒ Use default location

Location:

Project type:

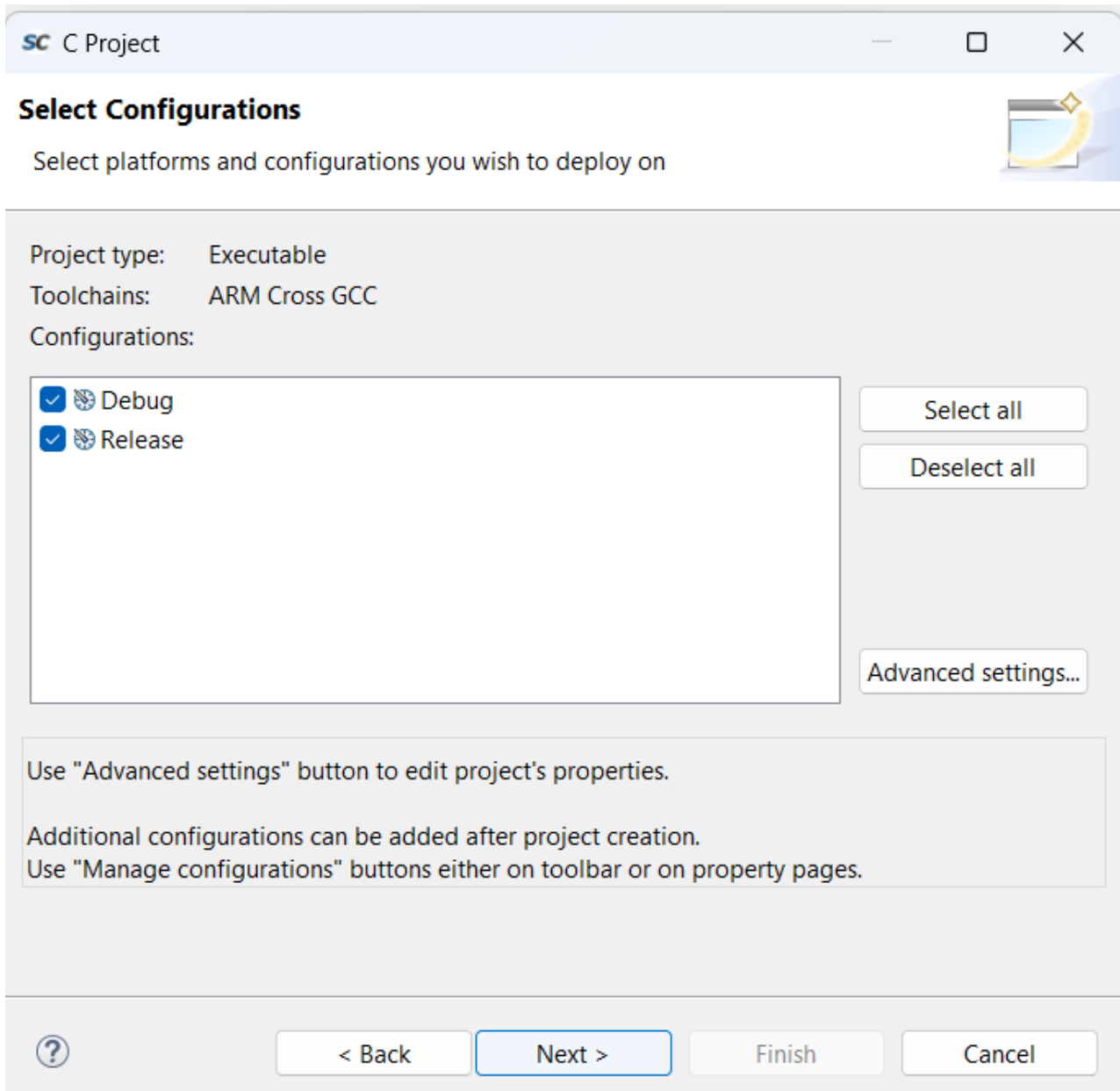
- > GNU Autotools
  - ▼ Executable
    - Empty Project
    - Hello World ARM C Project
    - Hello World RISC-V C Project
    - Hello World ANSI C Project
  - > Shared Library
  - > Static Library
  - > Makefile project

Toolchains:

- ARM Cross GCC
- Cross GCC
- RISC-V Cross GCC

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

Luego le indicamos que va a ser para depuración y para *release*.



The screenshot shows the 'C Project' dialog box in the SmartFusion2 IDE. The title bar reads 'SC C Project'. The main heading is 'Select Configurations', followed by the instruction 'Select platforms and configurations you wish to deploy on'. The project settings are listed as: Project type: Executable, Toolchains: ARM Cross GCC, and Configurations: (empty). A list box contains two items: 'Debug' and 'Release', both with checked checkboxes. To the right of the list box are three buttons: 'Select all', 'Deselect all', and 'Advanced settings...'. Below the list box, a text box contains the following instructions: 'Use "Advanced settings" button to edit project's properties.', 'Additional configurations can be added after project creation.', and 'Use "Manage configurations" buttons either on toolbar or on property pages.' At the bottom of the dialog, there is a row of buttons: a help button (question mark icon), '< Back', 'Next >' (highlighted with a blue border), 'Finish', and 'Cancel'.

SC C Project

### Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable  
Toolchains: ARM Cross GCC  
Configurations:

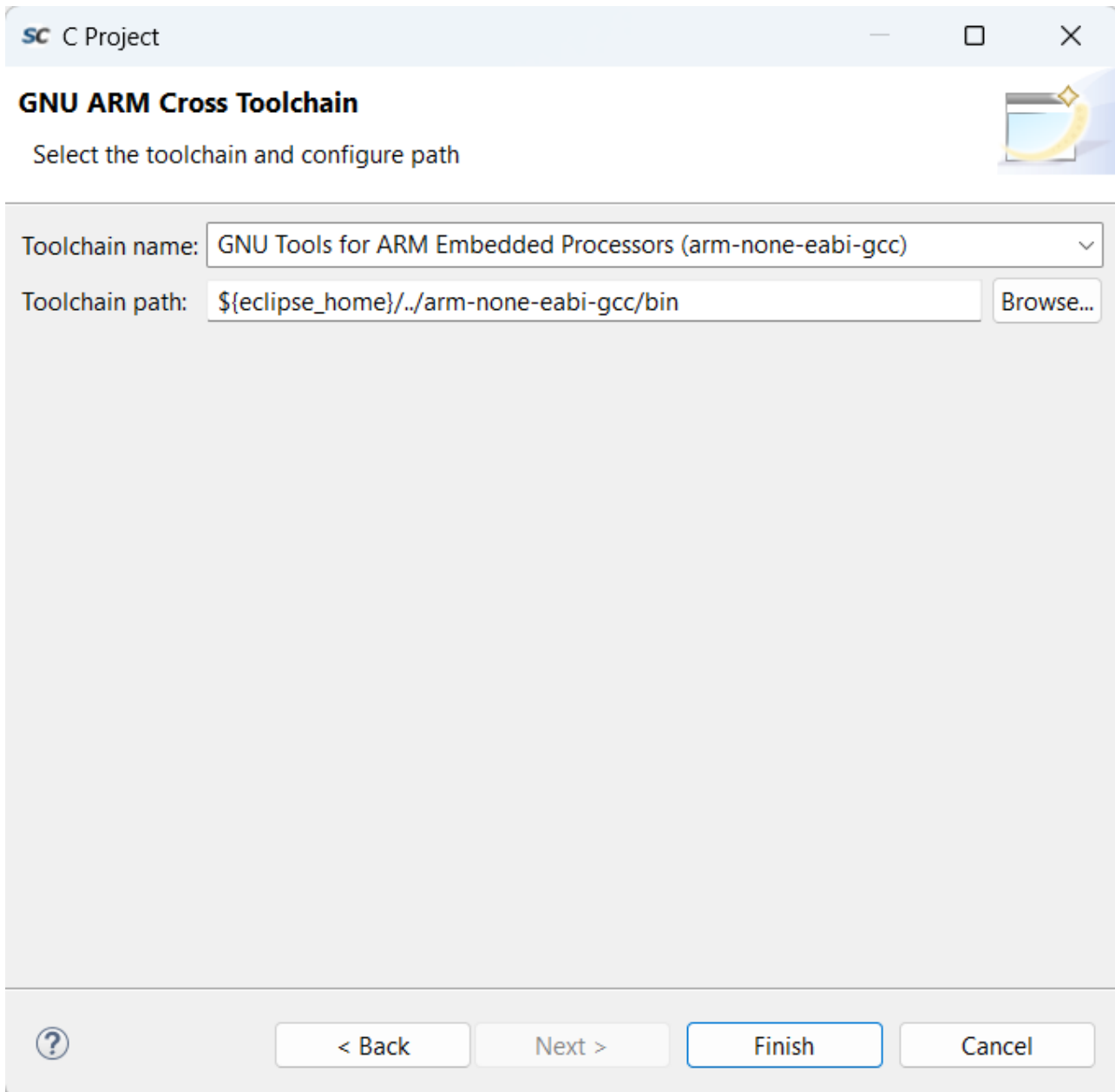
- ☒ Debug
- ☒ Release

Select all  
Deselect all  
Advanced settings...

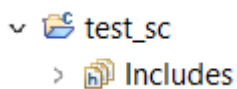
Use "Advanced settings" button to edit project's properties.  
Additional configurations can be added after project creation.  
Use "Manage configurations" buttons either on toolbar or on property pages.

? < Back Next > Finish Cancel

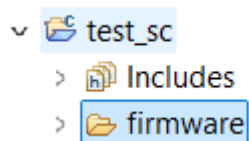
Luego le decimos que utilice GCC para ARM.



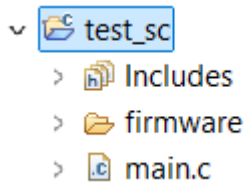
Ahora ya podemos crear un proyecto. Al crear el proyecto el SoftConsole nos crea esto.



Primero lo que hacemos es importar la carpeta *firmware* creada por Libero con los drivers.



Ahora podemos crear nuestro proyecto en un fichero *main.c*



Dentro del *main.c* programamos el funcionamiento de los leds.

Para ello primero llamamos al fichero de los drivers de los GPIO, que está dentro de la carpeta *firmware* que hemos añadido, después configuramos los GPIO como salida, luego les adjudicamos un valor de 0 y de 1 cada 1000000 de cuentas de reloj, lo que hace que los leds parpadeen.

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

```
int main(void)
{
    MSS_GPIO_init();

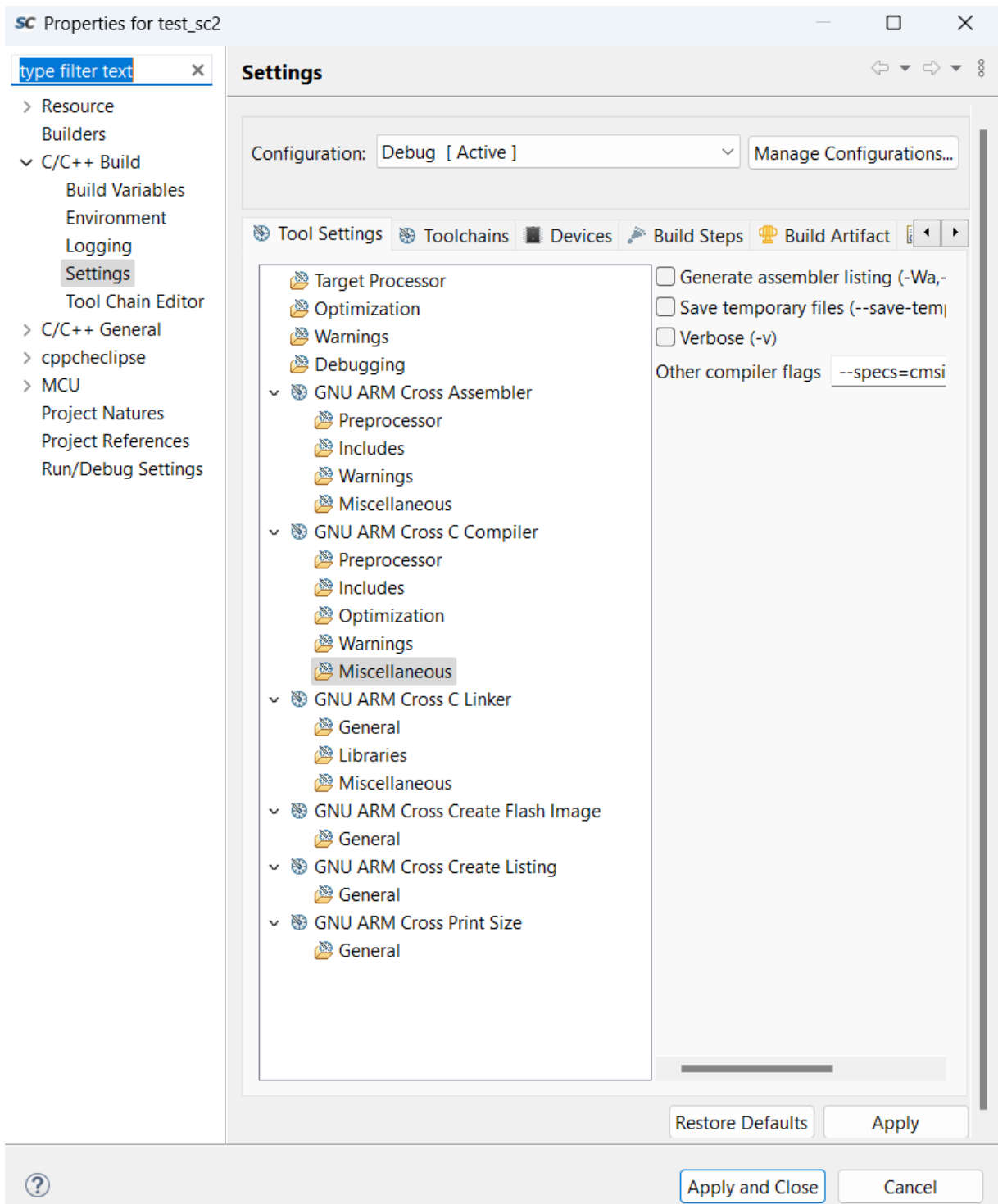
    MSS_GPIO_config(MSS_GPIO_0, MSS_GPIO_OUTPUT_MODE);
    MSS_GPIO_config(MSS_GPIO_1, MSS_GPIO_OUTPUT_MODE);
    MSS_GPIO_config(MSS_GPIO_2, MSS_GPIO_OUTPUT_MODE);

    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++);
    }
}
```

Ahora si lo intentamos compilar dará error, esto es debido a que hay que configurar variables internas del compilador.

## Compilación

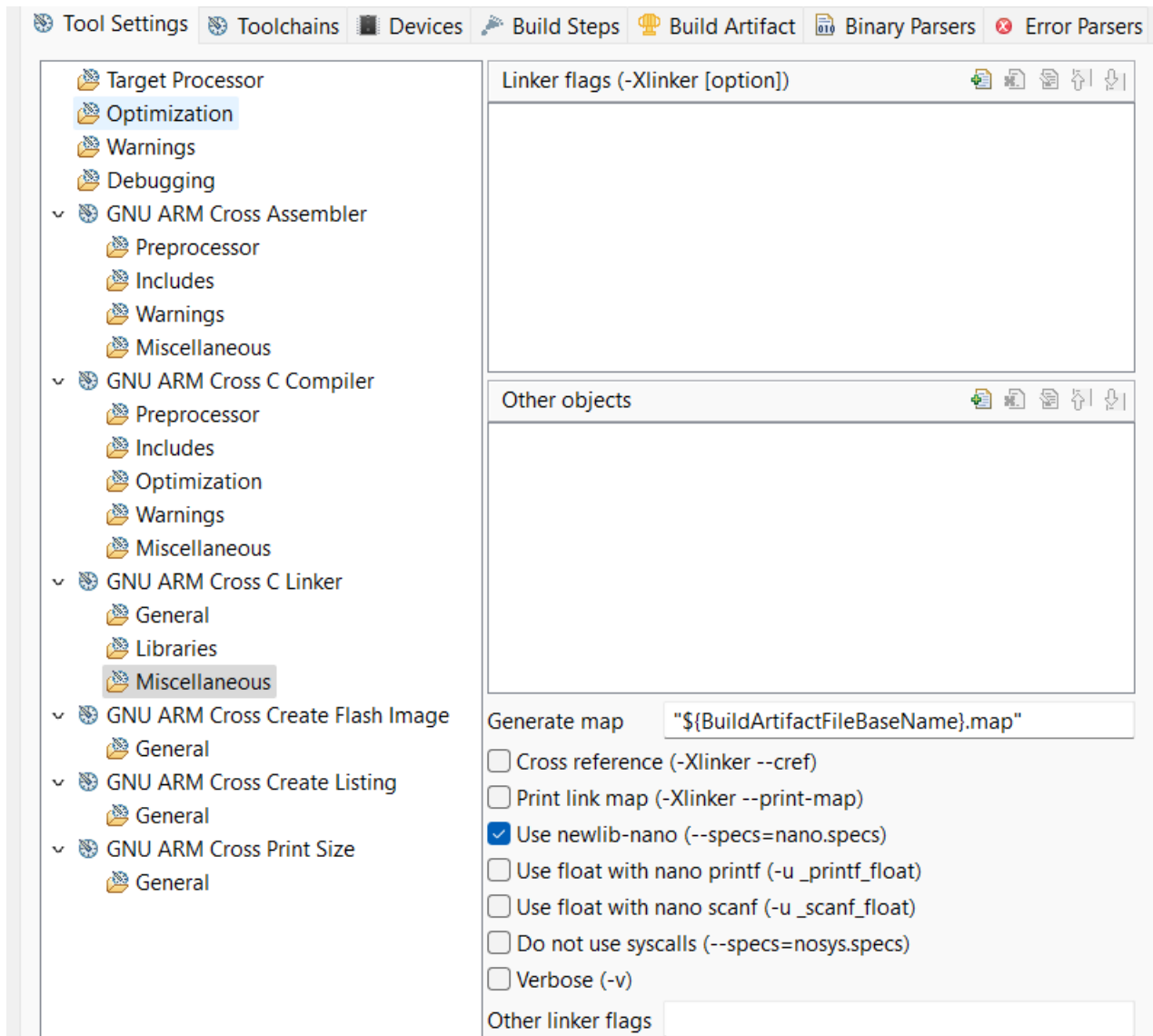
Para acceder a las variables, se hace dando clic derecho al proyecto que hemos creado y dándole a *properties*. Una vez en la pestaña se tiene que ir a *C/C++ Build* en *Settings*.



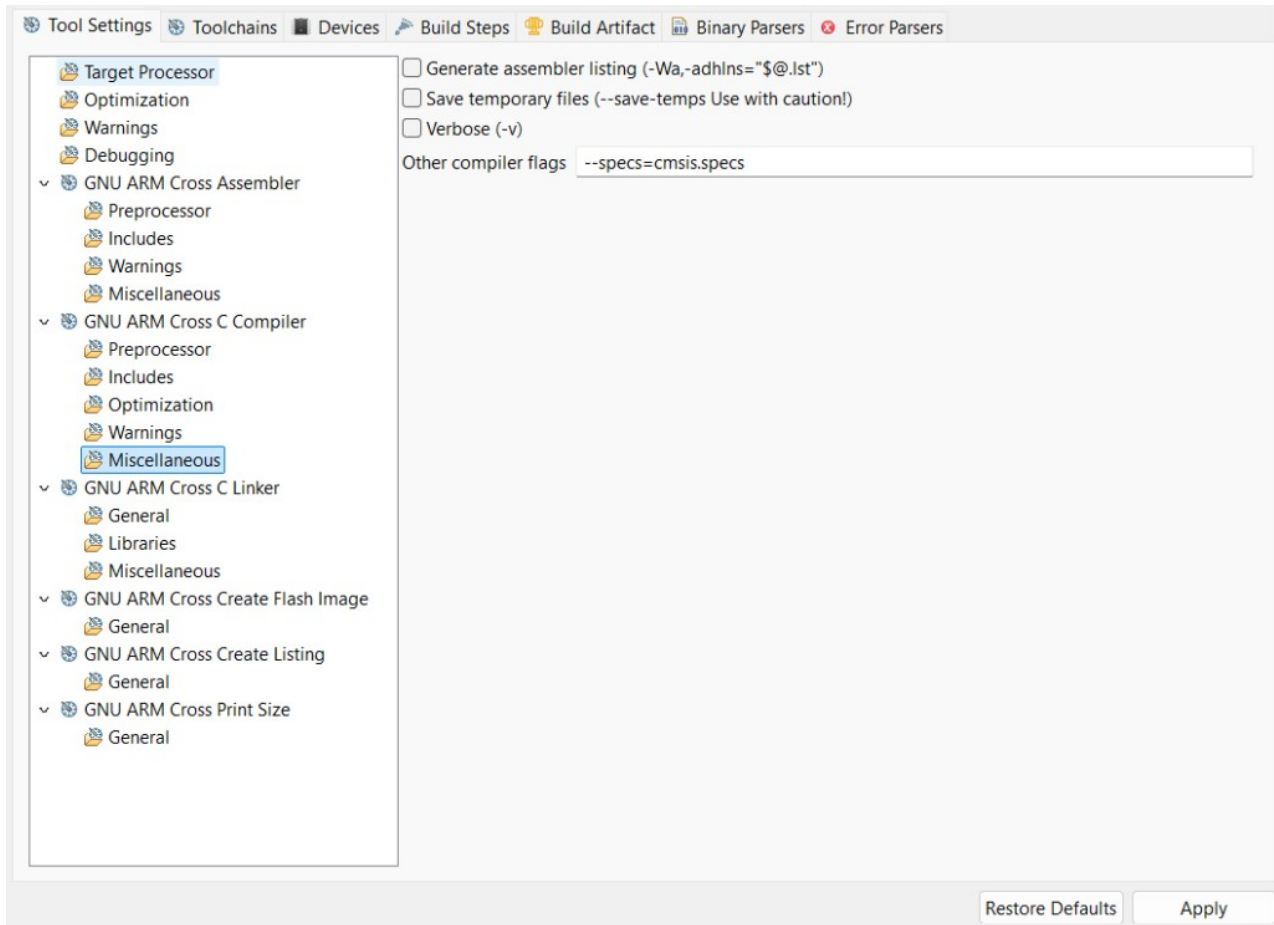
Estas variables son las que hay que configurar:

- Marcar la casilla `--specs=nano.specs` en *Miscellaneous* del *GNU ARM Cross C Linker*.

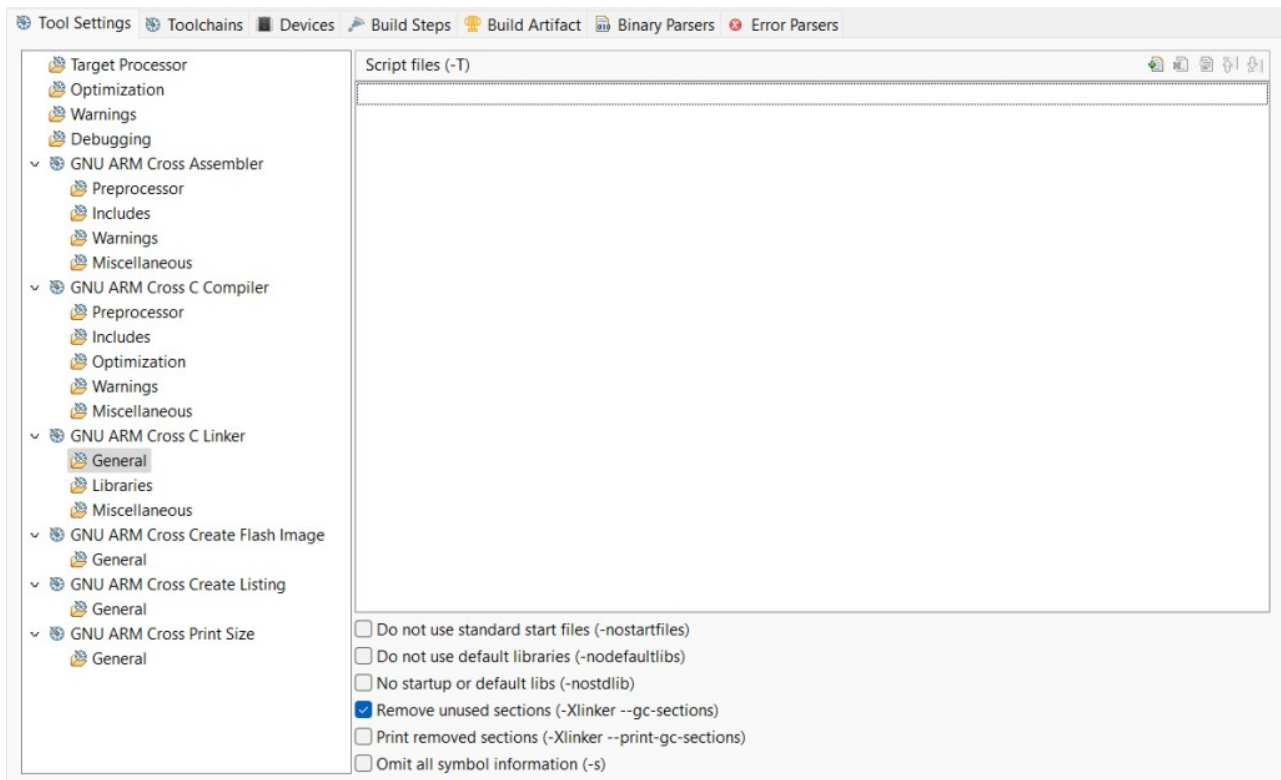




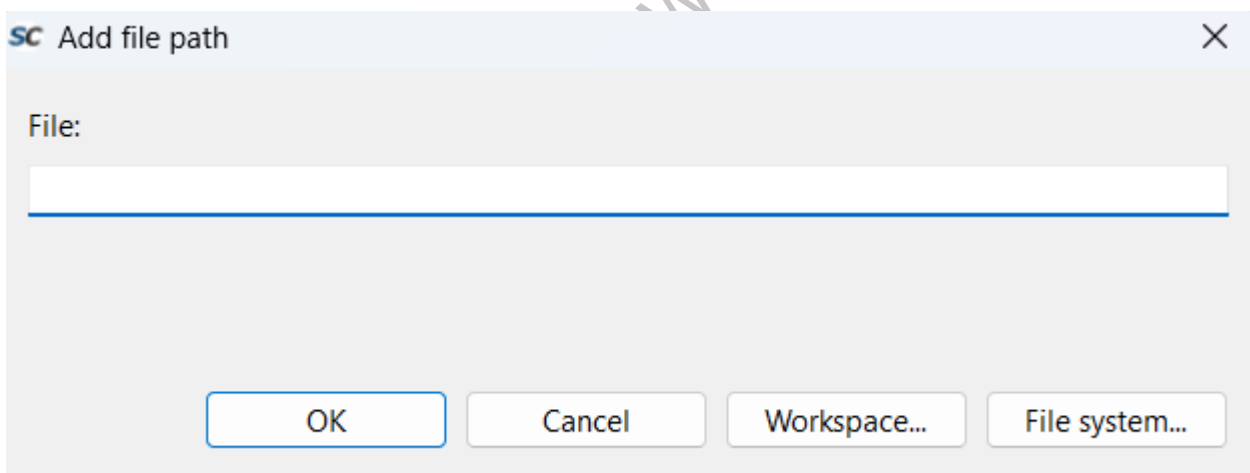
- Añadir el flag `--specs=cmsis.specs` en *Miscellaneous* del *GNU ARM Cross C Compiler*.



- Y por último, al tratarse de un FPGA basada en memoria Flash hay que indicar si se quiere guardar el programa al apagar la FPGA.  
Para ello en *General* del *GNU ARM Cross Compiler* hay que añadir un fichero de tipo *.ld*.

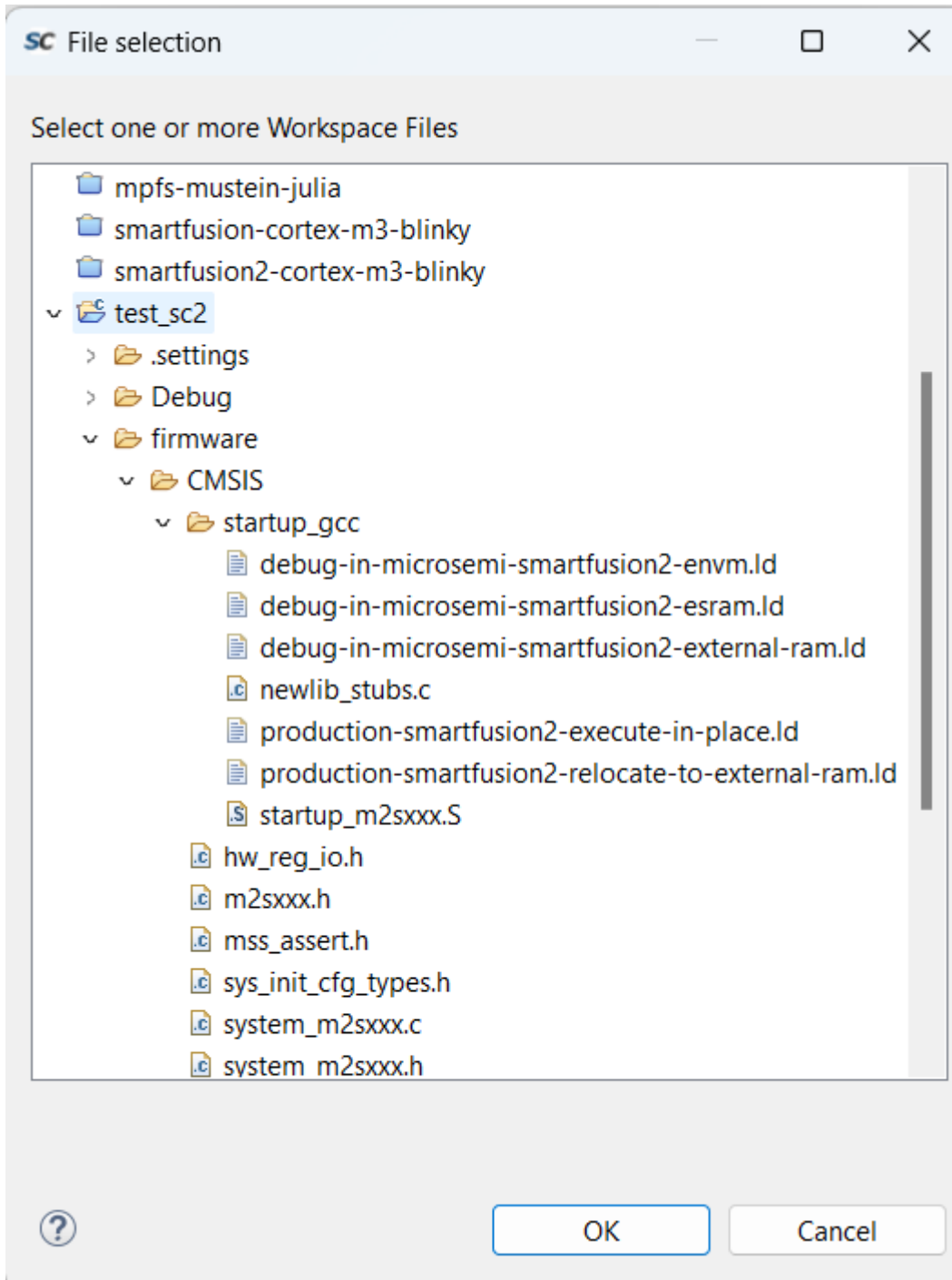


Para ello se le da al botón de añadir fichero y nos pregunta de dónde vamos a añadir el fichero.

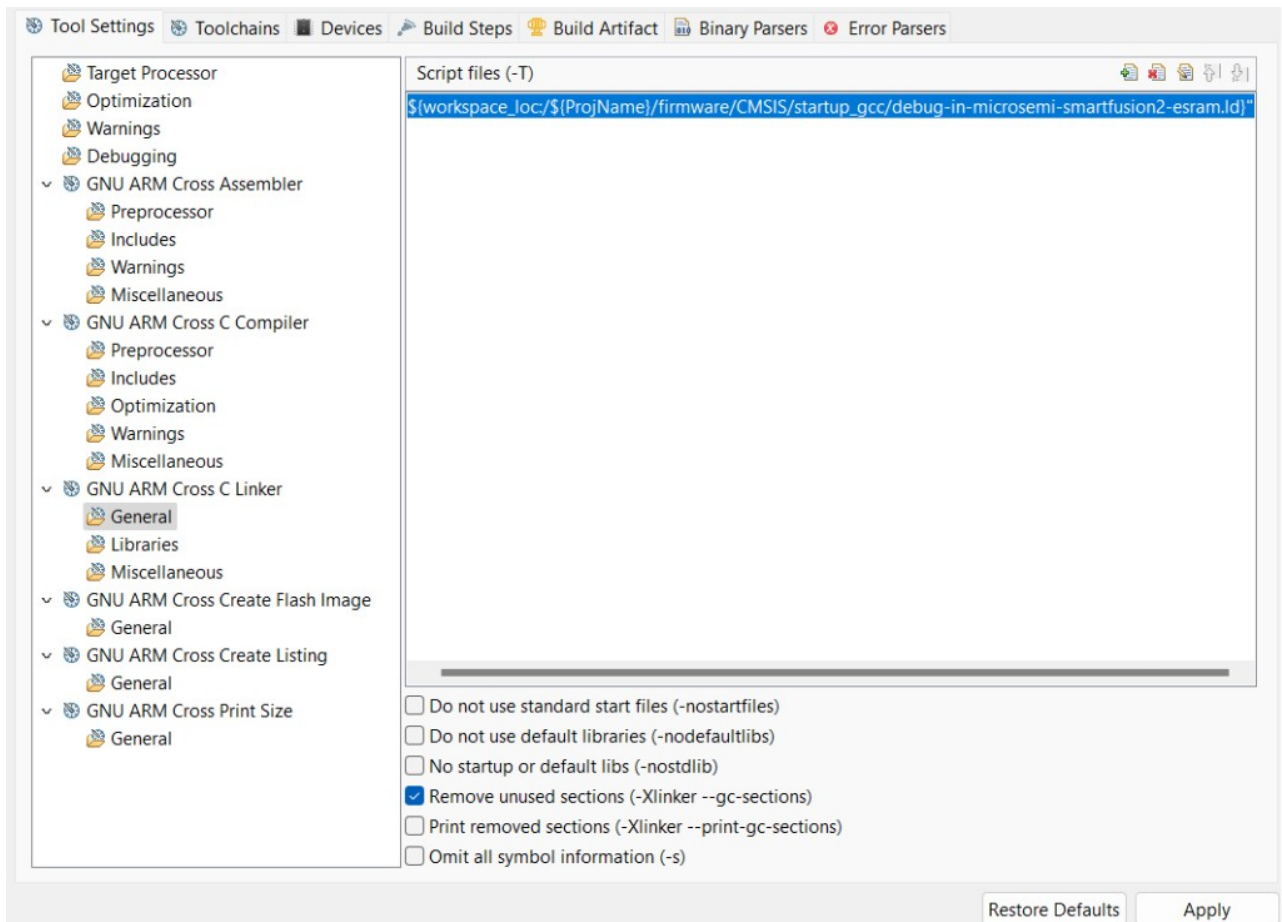


Pinchamos en *workspace* y se nos abre una pestaña, los ficheros que estamos buscando están en el proyecto que hemos creado. En el *firmware* > *CMSIS* > *startup\_gcc*. Aquí se presentan diferentes opciones.

- Grabar en la memoria no volátil el ejecutable (...*envm.ld*). Esto hace que no haya que grabar la FPGA más de una vez.
- Grabar en la memoria SRAM el ejecutable (...*esram.ld*). Esto borra el ejecutable al apagar la placa (pero el bitstream de Libero sigue dentro).
- Grabar en una RAM externa el ejecutable (...*external-ram.ld*). Este desconozco su funcionamiento.



Una vez elegido quedan todas las variables de compilación programadas.



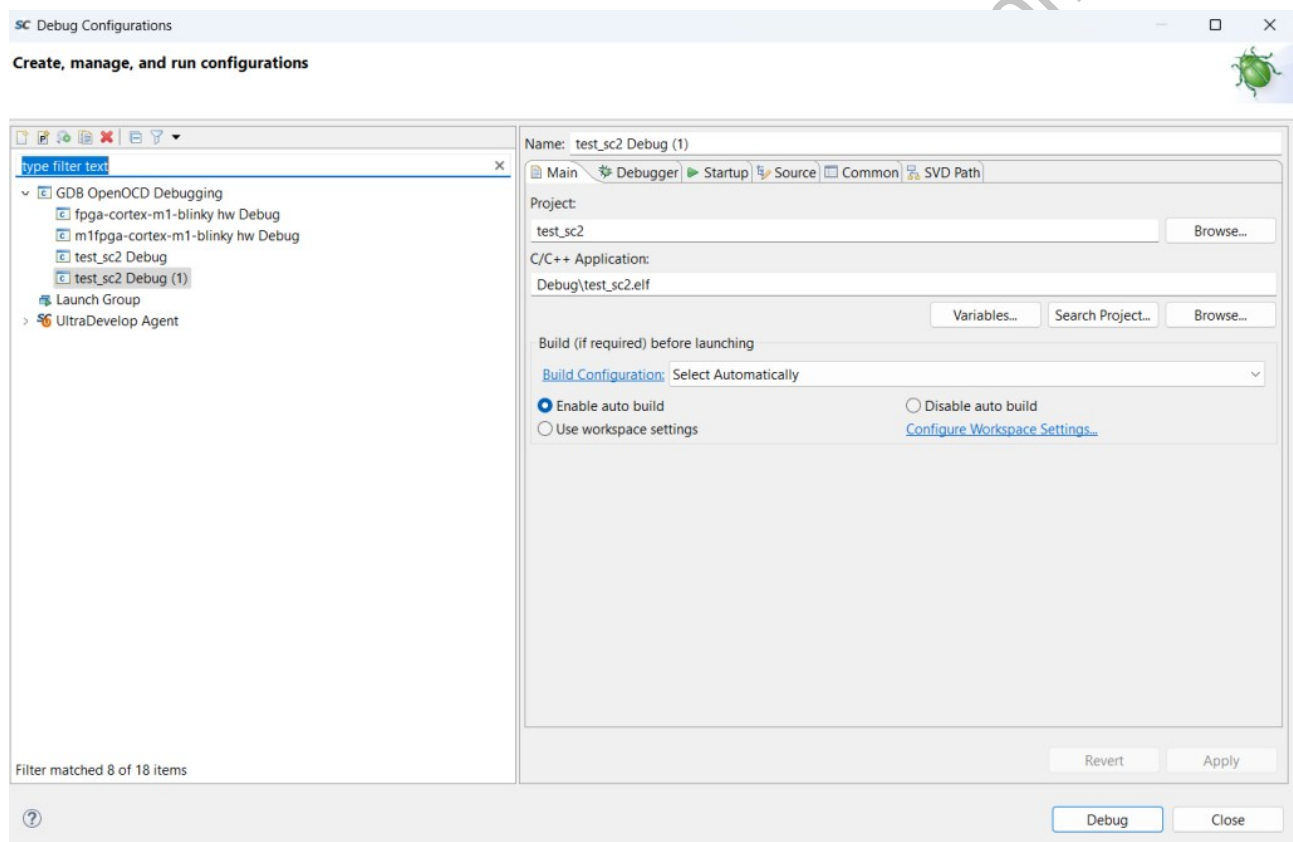
Ahora ya se puede compilar. Al compilar aparecen hasta tres carpetas (dependiendo de cómo se haga la compilación, si es de depuración aparece la carpeta *Debug*, si es de versión funcional aparece la carpeta *Release*).

- *Binaries*: que es una carpeta donde el SoftConsole mete el .elf del proyecto para que sea más fácil de localizar.
- *Debug*: que es la carpeta donde generar el .elf de depuración. También se genera el .hex que necesitan los cores de ARM para programarse de forma independiente, o sea, utilizando el J-Link.
- *Release*: que es la carpeta donde se genera el .elf de depuración. También se genera el .hex que necesitan los cores de ARM para programarse de forma independiente, o sea, utilizando el J-Link, este .hex también se puede añadir al bitstream que genera Libero para crear un único bitstream con toda la funcionalidad del SoC, que se puede programar desde el propio Libero o desde el FlashPro Express.

- > Binaries
- > Includes
- > Debug
- > firmware
- > Release
- > main.c

## Depuración/Ejecución

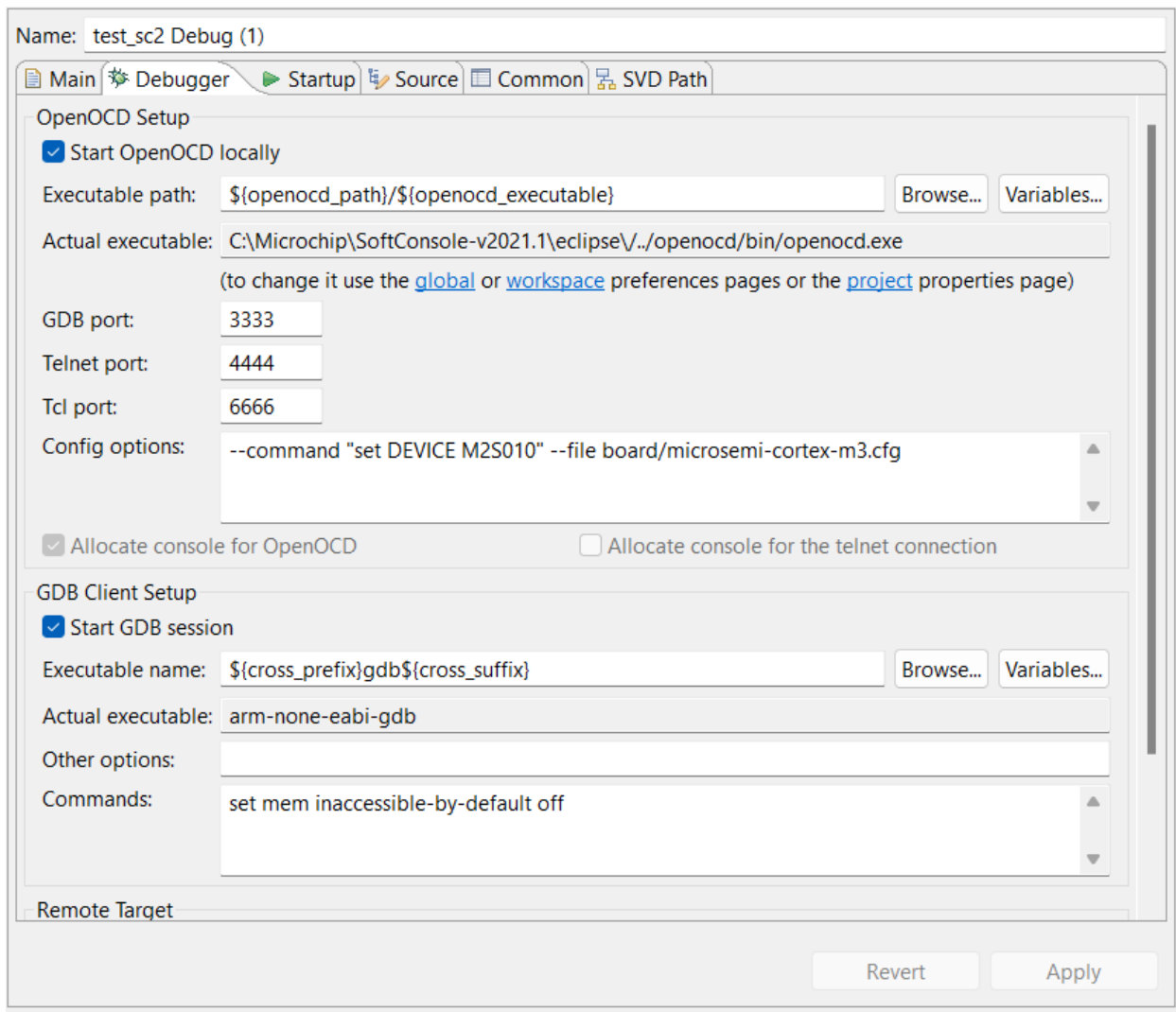
Para la depuración/ejecución se tiene que crear un perfil de depuración/ejecución. Para ello en la opción de *Configurations*, primero se crea un perfil, con el .elf que se quiera probar.



En la pestaña *Debugger* se tiene que configurar de la siguiente manera. Y en *Config options* se tiene que añadir el siguiente texto:

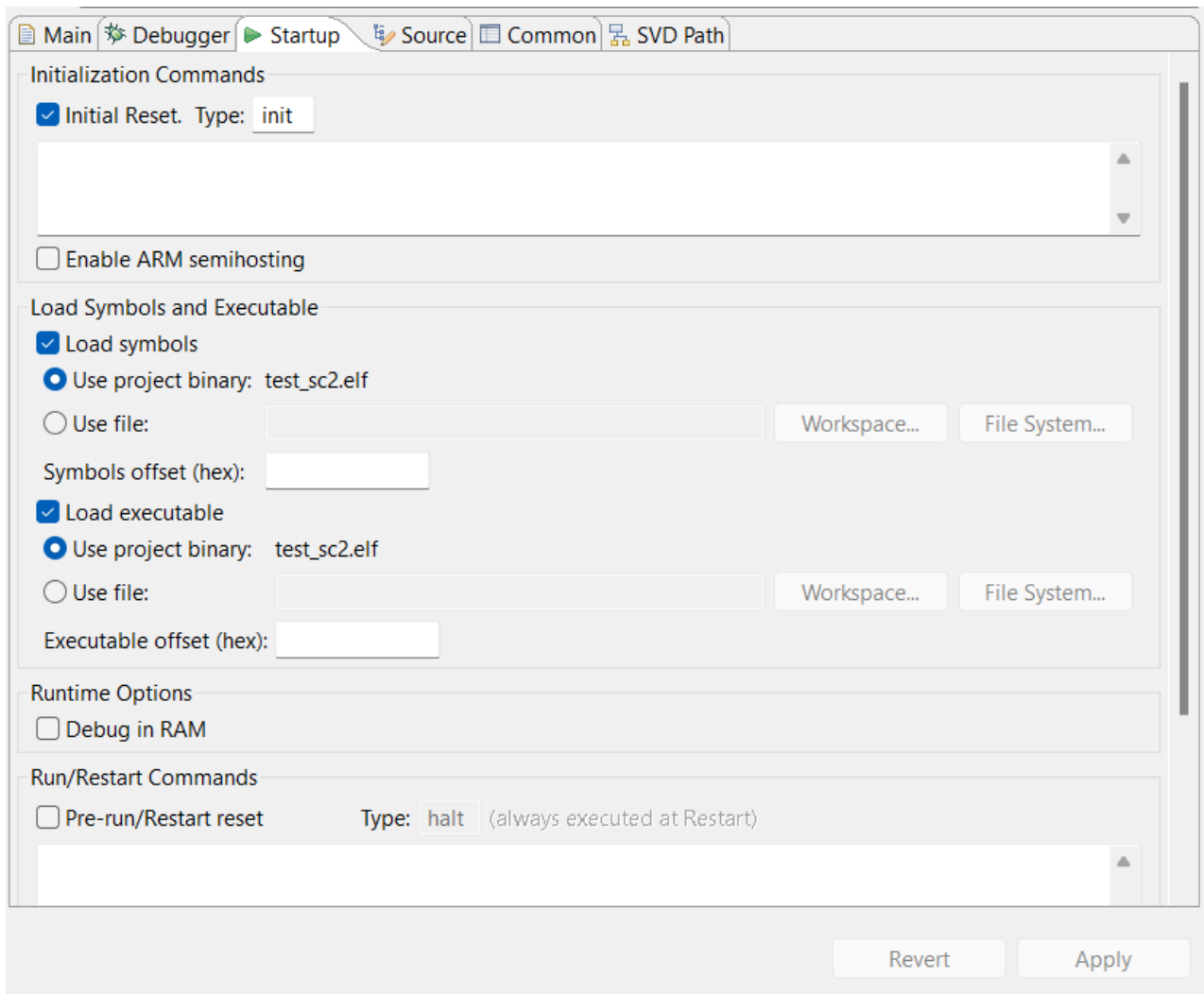
`-command «set DEVICE <dispositivo a programar>» -file board/microsemi-cortex-m3.cfg`

(En mi caso voy a programar un SmartFusion2 de tipo M2S010). Y abajo se añade el comando: **set mem inaccessible-by-default off**



Y por último en la pestaña *Startup*, se desmarca la casilla *Pre-run/Restart reset*.

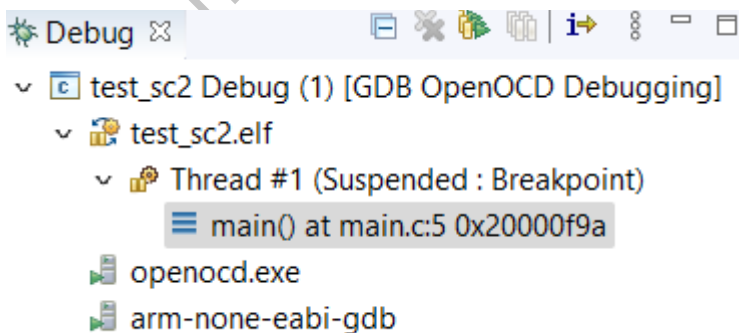




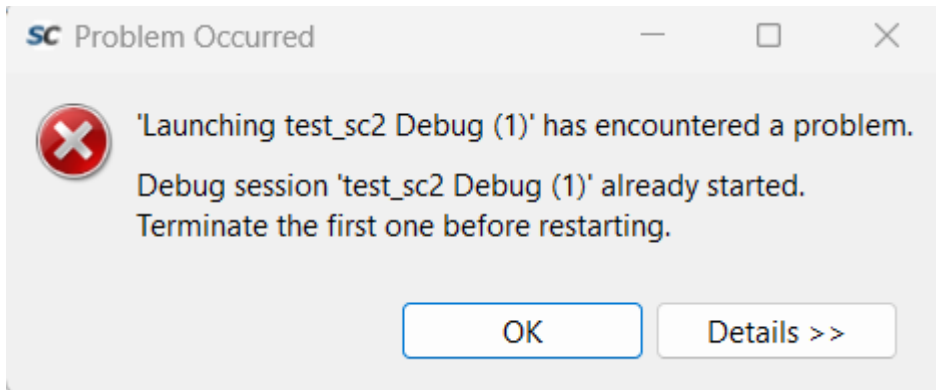
Con todo esto ya se puede depurar la placa. **Recordad que primero hay que grabar el bitstream desde Libero, que no se pierde entre reprogramaciones.**

Cuando empieza a depurar suele tener un breakpoint en la primera línea para indicarlo.

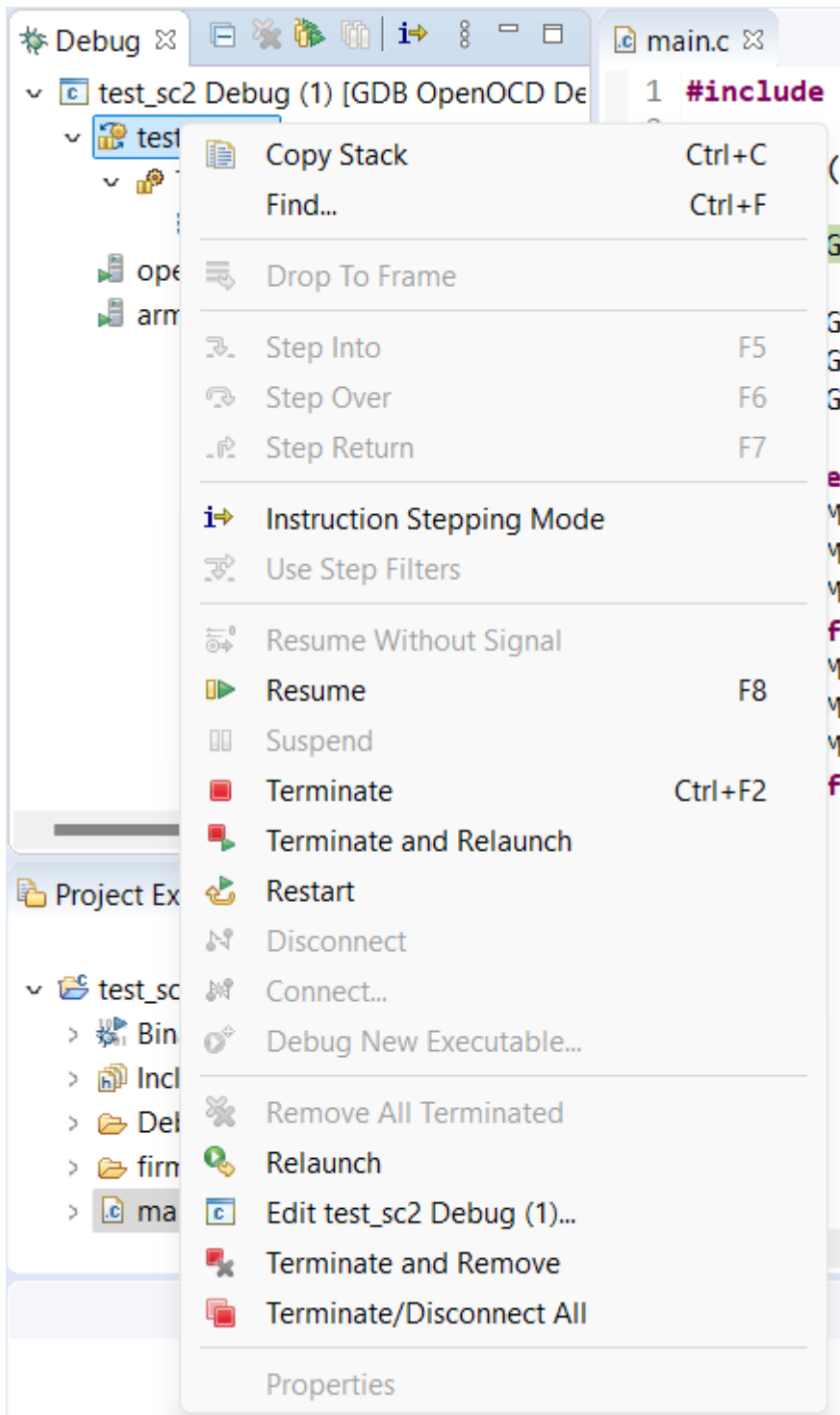
Para saber si se está depurando correctamente, en el perfil de depuración aparece una pestaña llamada *Debug*, ahí aparece si estás depurando un core.



Nota: cada vez que quieras reprogramar el SoC, te va a saltar un error, diciendo te que ya hay un programa ejecutándose, y que al intentar reprogramarlo se acaba la depuración anterior.



Para evitar este mensaje le puedes dar clic derecho en Debug y a *Terminate and Relaunch*.



A partir de esta entrada se comentarán otros periféricos que tienen las SmartFusion2, sobre cómo se trabaja con ellos.