Cómo configurar el I2C de una SmartFusion2

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/12/04/como-configurar-el-i2c-de-una-smartfusion2/

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

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

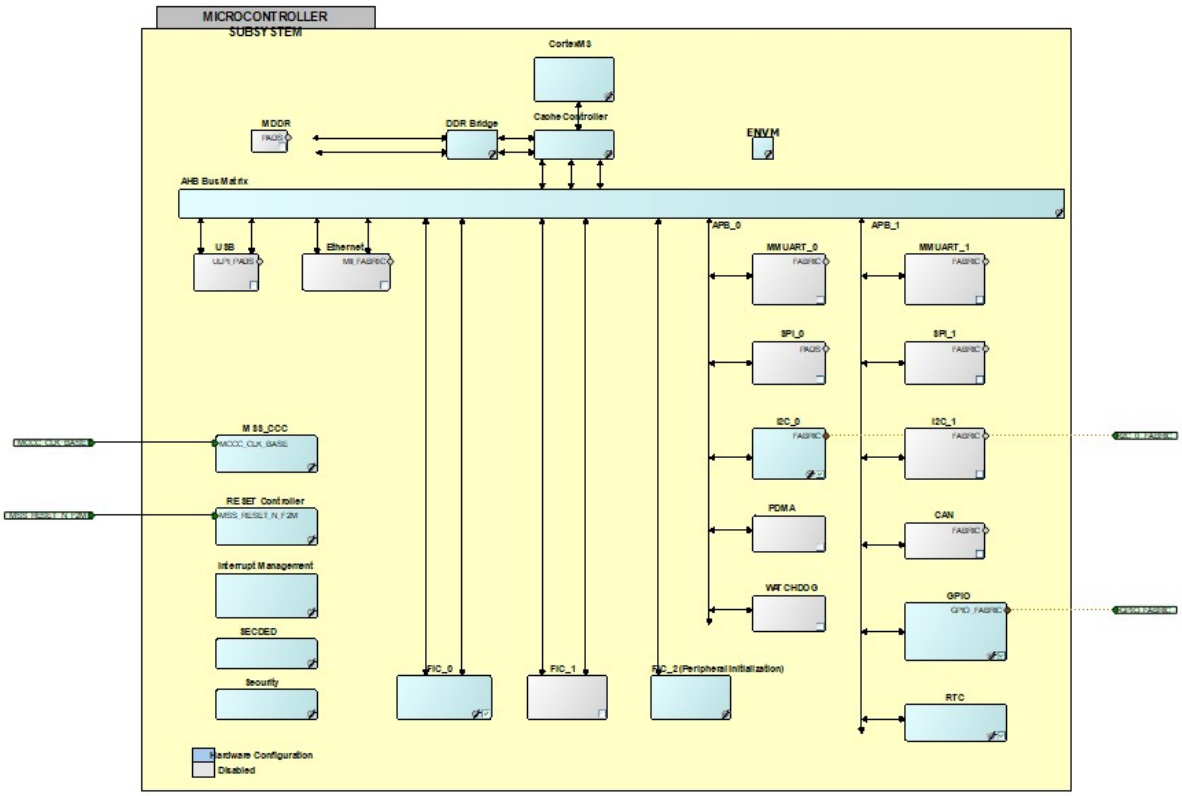
Para esta entrada nos vamos a basar estas dos entrada anteriores.

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/

Proyecto en Libero

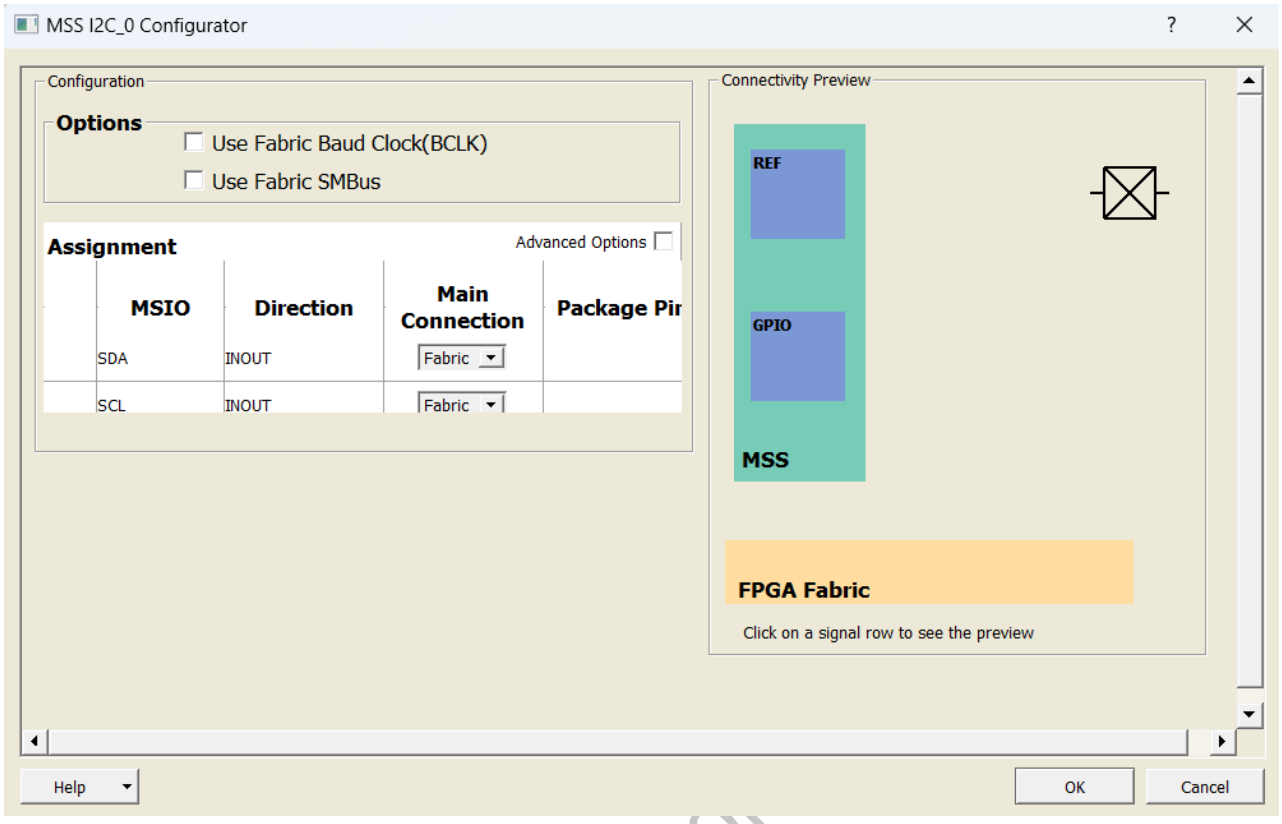
Lo primero que hay que hacer es crear un proyecto en Libero, utilizando como referencia las entradas anteriores.

Entonces, seleccionamos el I2C0 para utilizarlo como salida.

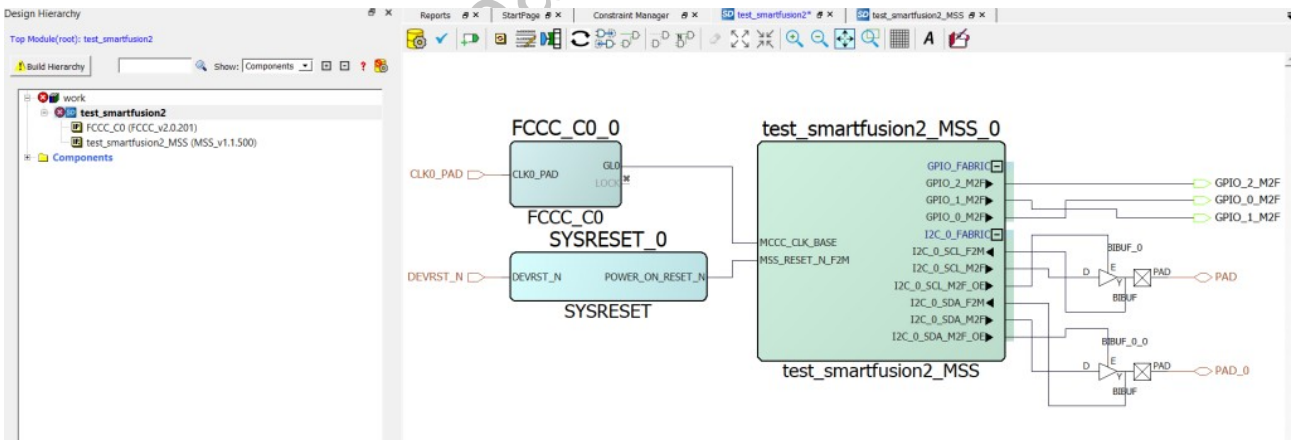


Si entramos dentro del I2C para ver cómo se configura, podemos elegir si queremos utilizar cualquier pin del SoC o los pines específicos para el I2C del SoC. En nuestro caso utilizamos cualquier pin de la FPGA (*Fabric*).

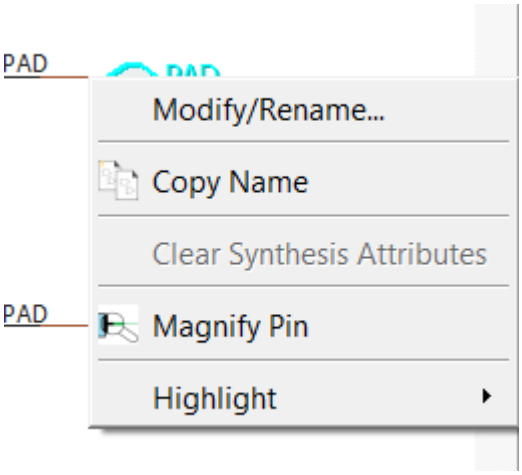
https://soceame.wordpress.com/2024/12/04/como-configurar-el-i2c-de-una-smartfusion2/



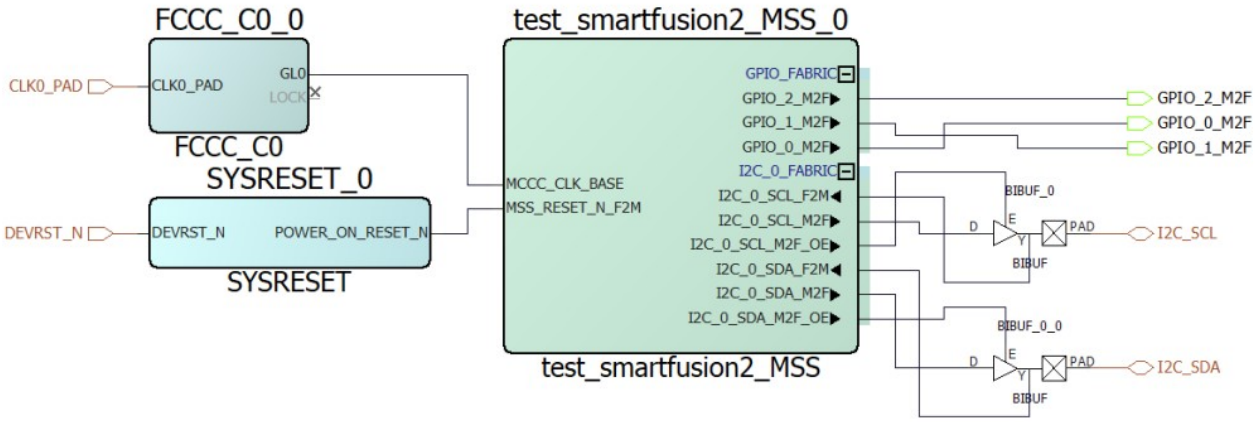
Una vez hemos generado las salidas del I2C del core, tenemos que configurarlas. Para configurar las salidas debido a que el I2C es un protocolo bidireccional, tenemos que utilizar bloques *BIBUF*, que permiten seleccionar la direccionalidad del pin con la salida OE del core. (El bloque BIBUF está en el Catalog)



Luego lo que hacemos es cambiarle el nombre al pin, para ello clic derecho en la salida *Modify/Rename* y elegimos el nuevo nombre.



Una vez le hemos cambiado el nombre a los pines, queda de la siguiente forma.



Ahora lo que tenemos que hacer es terminar de configurar el modelo, dándole clic a icono amarillo con el engranaje, y después a *Build Hierarchy* para que nos añada los *BIBUF* a la jerarquía. Después sintetizamos el proyecto, *recordad de generar el Memory Map*.

Después, de sintetizar elegimos los nuevos *constraints* para los pines. Abrimos el *Edit* para los 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

Después, elegimos los pines del I2C.

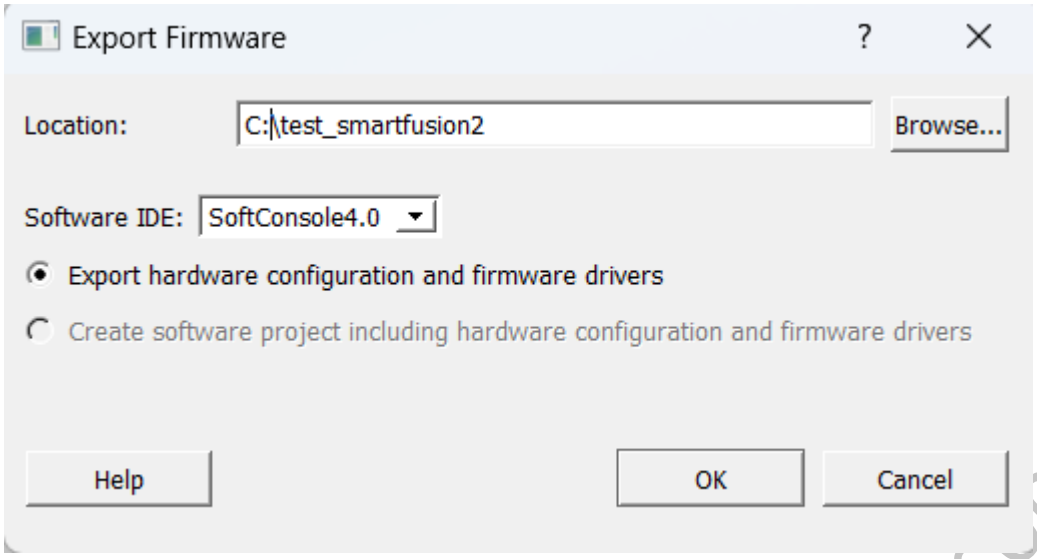
	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze
1	CLK0_PAD	INPUT	LVCMOS25	23	☑	INBUF	Bank6	TRISTATE
2	DEVRST_N	INPUT	--	72	☑	SYSRESET	--	--
3	GPIO_0_M2F	OUTPUT	LVCMOS25	129	☑	OUTBUF	Bank0	TRISTATE
4	GPIO_1_M2F	OUTPUT	LVCMOS25	128	☑	OUTBUF	Bank0	TRISTATE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

Después, terminamos generando el bitstream del proyecto. Cuando lo tengamos, programamos el SoC. Y ahora extraemos los drivers del proyecto, para ello pinchamos en *Configure Firmware Cores*.

Tenemos que tener seleccionado los drivers del I2C.

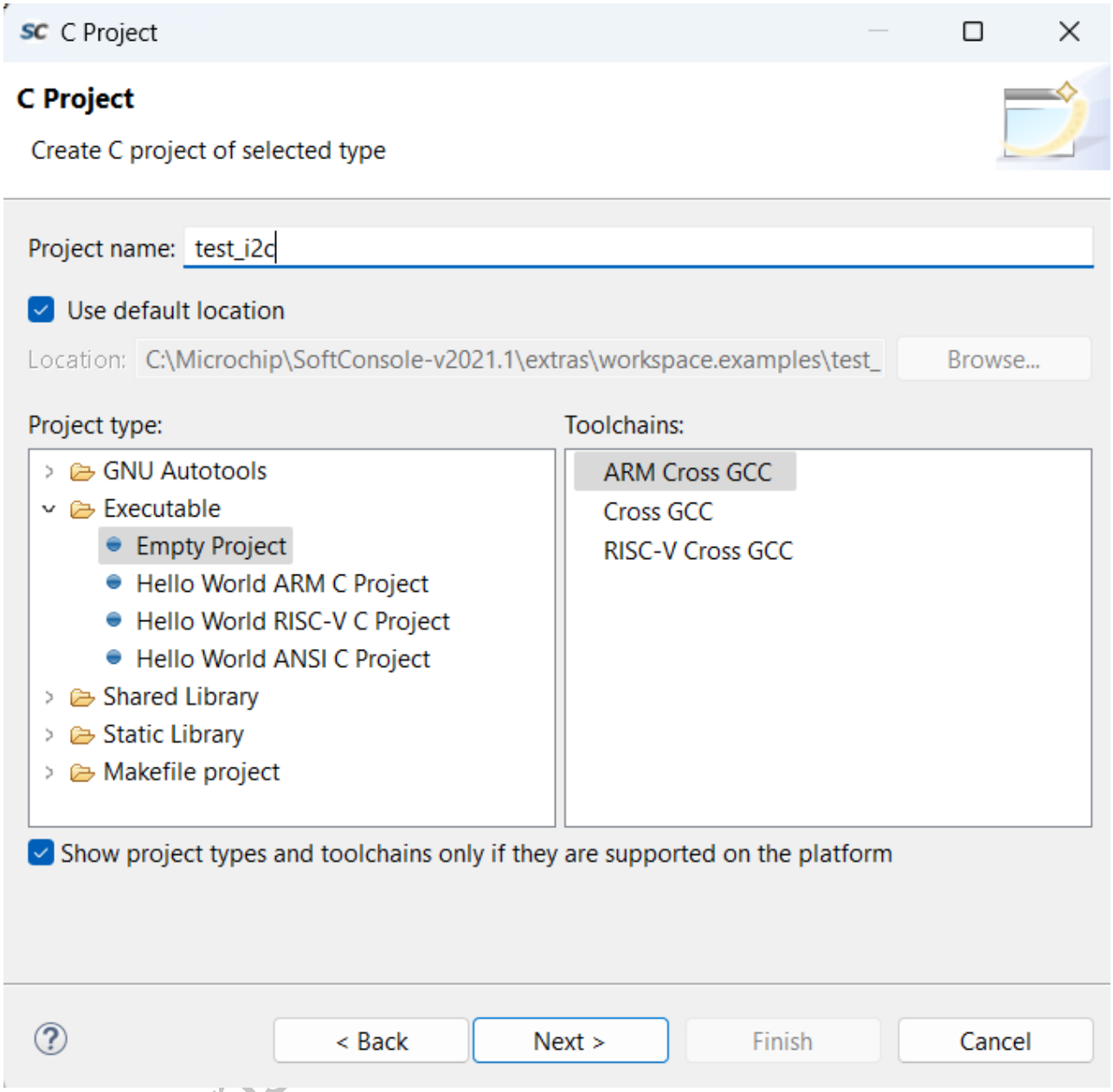
	Generate		Instance Name	Core Type	Version	Compatible Har
1	☑		SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.1(	test_smartfusion2_MSS
2	☑		SmartFusion2_MSS_GPIO_Driver_0	SmartFusion2_MSS_GPIO_Driver	2.1.1(	test_smartfusion2_MSS:GPIO
3	☑		SmartFusion2_MSS_HPDMA_Driver_0	SmartFusion2_MSS_HPDMA_Driver	2.2.1(	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

Luego los exportamos.

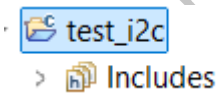


Proyecto en SoftConsole

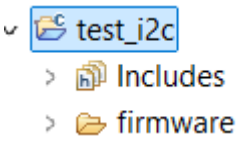
Lo primero que hacemos es crear un proyecto en C vacío



Cuando se crea el proyecto, tenemos la carpeta con los *includes*.



Ahora añadimos los drivers que acabamos de exportar y creamos un fichero main.c.



Ahora tenemos dos opciones de creación con el I2C, que sea master o que sea slave.

- **master**

Si lo definimos como maestro, lo primero que tenemos que hacer es configurar el I2C como maestro (*MSS_I2C_init*) y después decirle que escriba (*MSS_I2C_write*) los datos que están en el buffer de escritura (*tx_buffer*). También hay un buffer que es el de escritura que es el que utilizan los esclavos para mandar datos al maestro.

NOTA: es importante entender que el maestro de una comunicación le tiene que decir a los esclavos cuando vaya a leer, cuántos datos quiere leer, en este ejemplo 16 datos.

```
#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**

Como esclavo es necesario entender que el esclavo de una comunicación I2C es totalmente reactivo, o sea, que no va a emitir datos por su propio interés, si no que tiene que ser el maestro es que le solicite los datos.

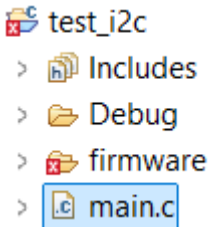
El I2C de las SmartFusion2 está compuesto de **dos buffers**, uno de lectura, que es el que lee el maestro cuando manda la orden de lectura, y uno de escritura que es el que utiliza el maestro para comunicarse con el esclavo.

Además, tal y como está configurado el I2C en las SmartFusion2, no se genera una interrupción cuando llega un dato nuevo, por lo que la única manera de saber si ha llegado un dato es leyendo el buffer de escritura de forma recurrente.

```
#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 );
}
```

Si se intenta compilar el proyecto dará fallo, porque hay que configurar las directivas del compilador que necesita el SoftConsole, y que son siempre las mismas.



Estas directivas de compilación están definidas en las entradas anteriores.