Cómo configurar la UART de una SmartFusion2

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/12/03/como-configurar-la-uart-de-una-smartfusion2/

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

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

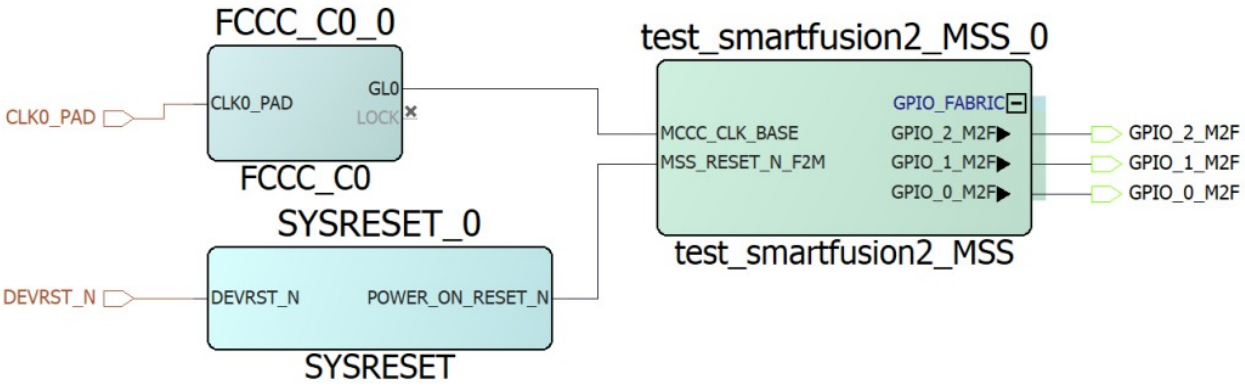
Este proyecto está basado en una entrada previa que utilizaremos como referencia para centrarnos en los conceptos más importantes de configuración de la UART para una SmartFusion2.

La entrada como referencia es esta, por lo que mucha de la configuración está aquí explicada.

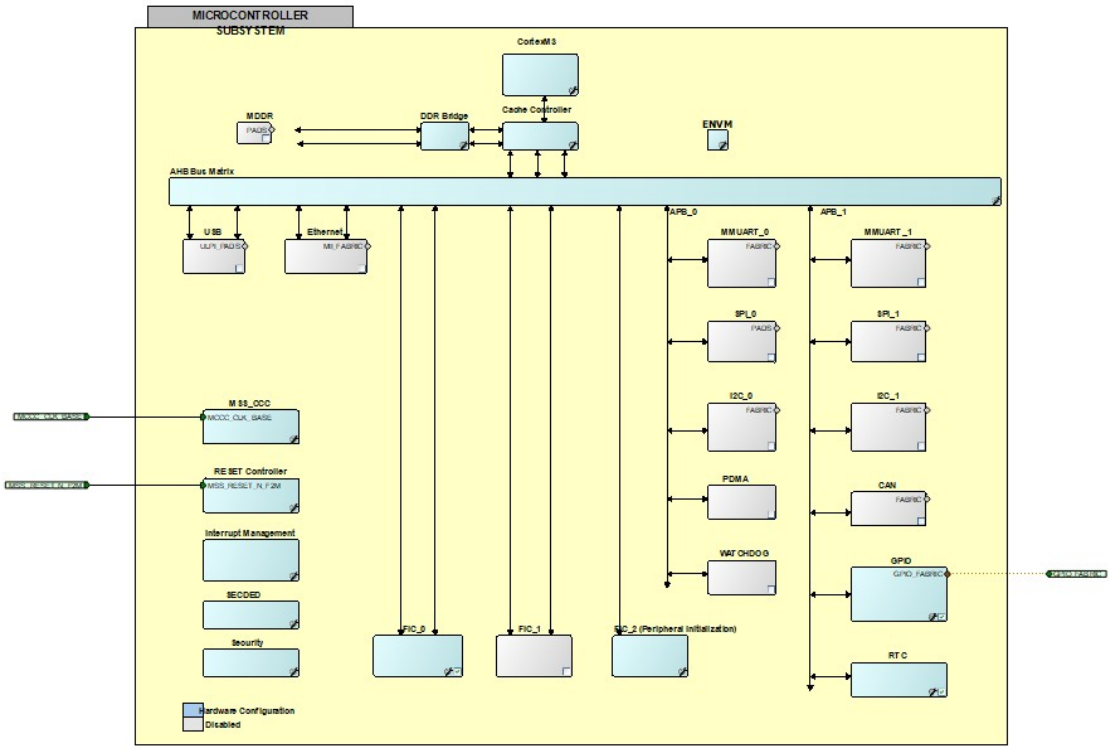
https://soceame.wordpress.com/2024/12/02/como-crear-un-proyecto-para-una-smartfusion2/

Configuración en Libero

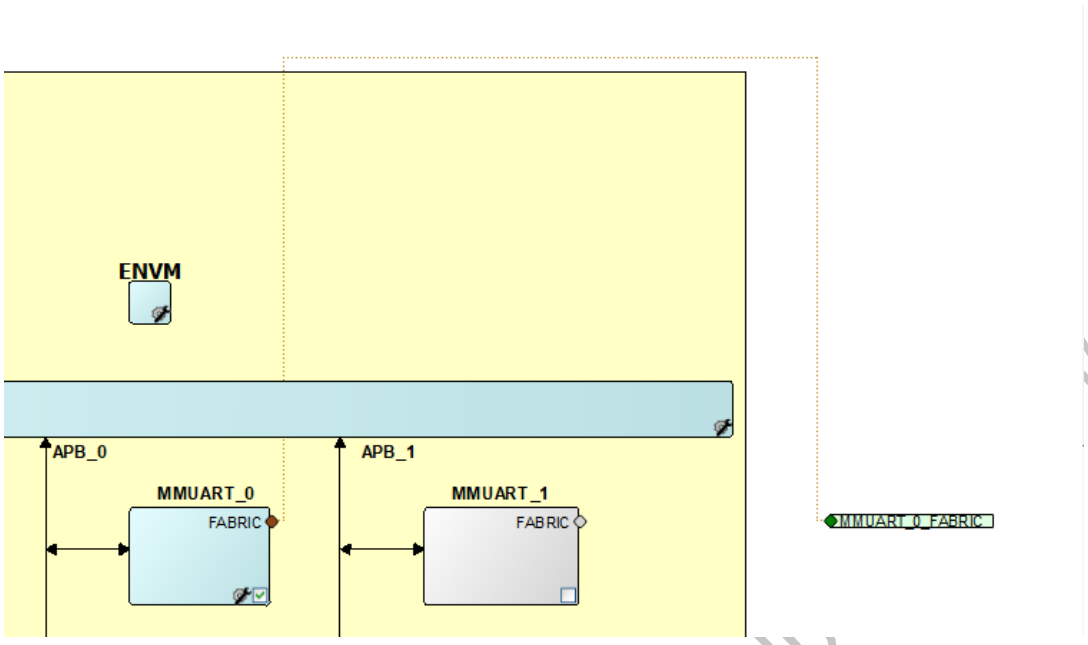
Para configurar una UART nos basamos en la segunda forma de configuración que explica la entrada anterior, que es la que genera el siguiente diagrama.



Este diagrama solo tiene los GPIOs configurados.



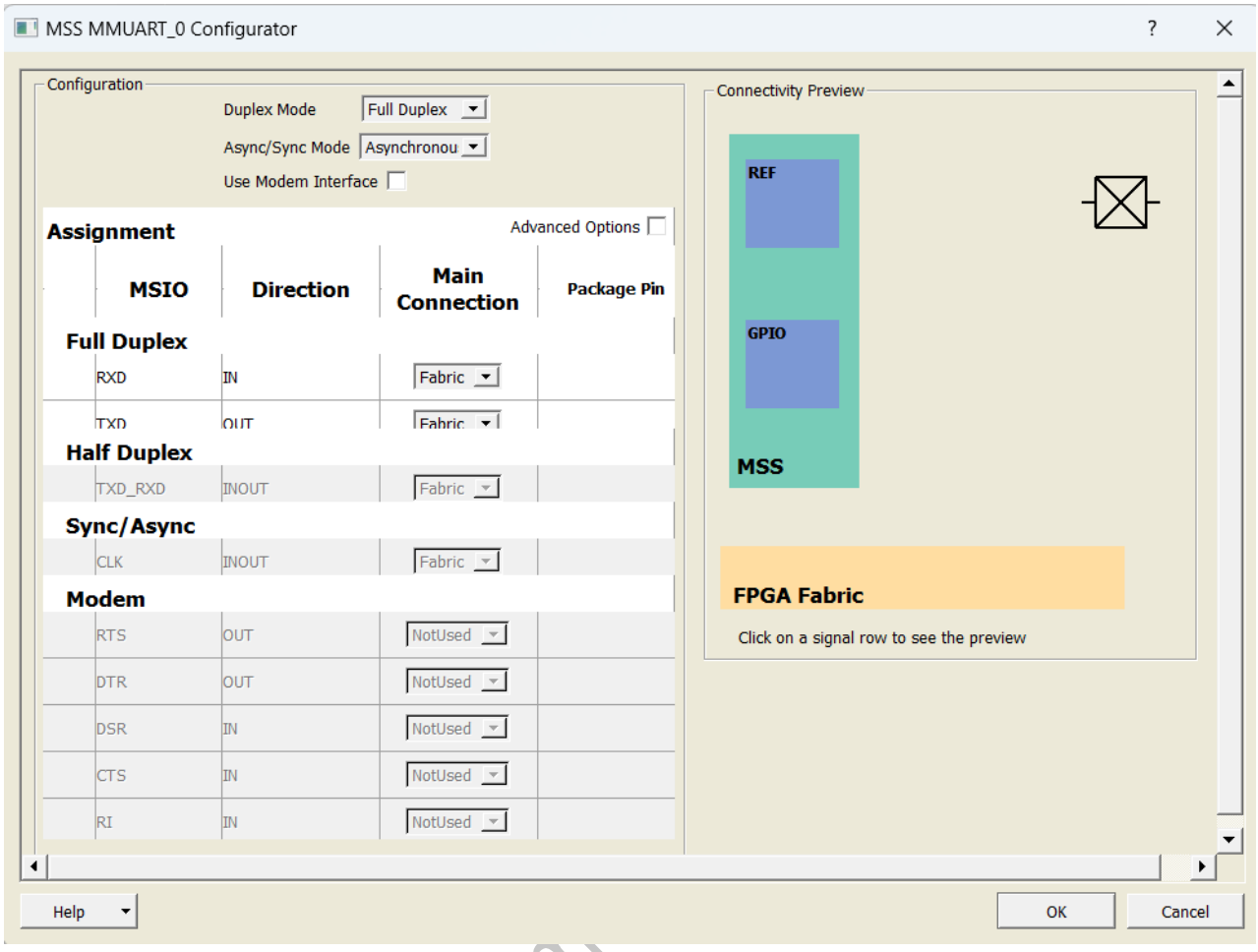
En este caso nos vamos a centrar en configurar la UART0, *(la UART1 se configura igual)*.
Para ello solo hay que marca la casilla de la UART0.



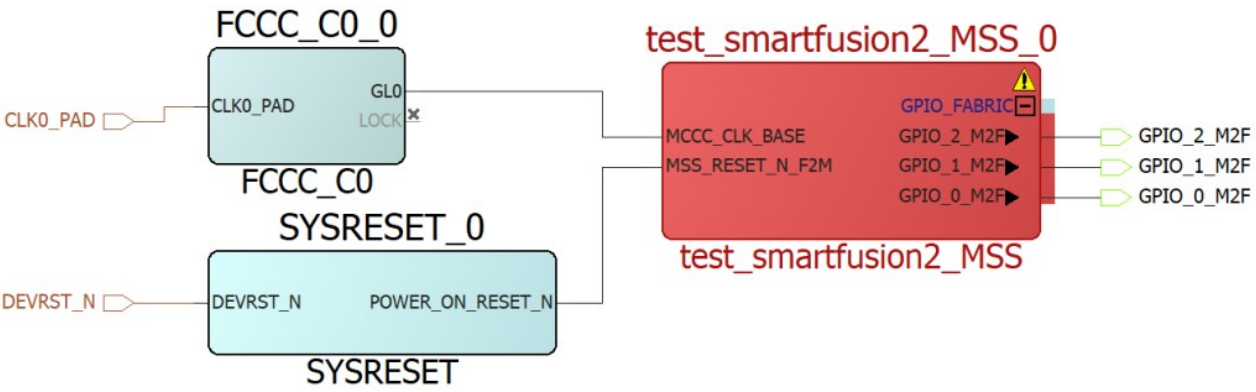
Al activarse genera una etiqueta, *MMUART_0_FABRIC*, que nos indica que hay una UART hacia el exterior por el *Fabric* (lógica programable), por lo que vamos a poder seleccionar los pines de salida de esa UART que queramos.

NOTA: Para configurar la UART para que no salga por el *Fabric*, se abre la UART que se ha activado previamente, y en la columna *Main Connection*, se selecciona la salida por los pines específicos de esa UART que tenga el chip *(suponiendo que tenga pines específicos, si no al Fabric)*.

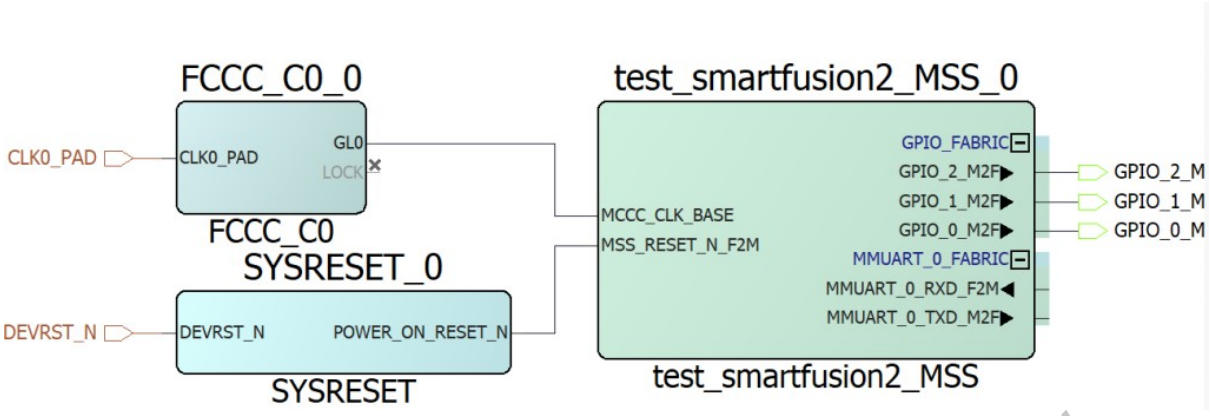
https://soceame.wordpress.com/2024/12/03/como-configurar-la-uart-de-una-smartfusion2/



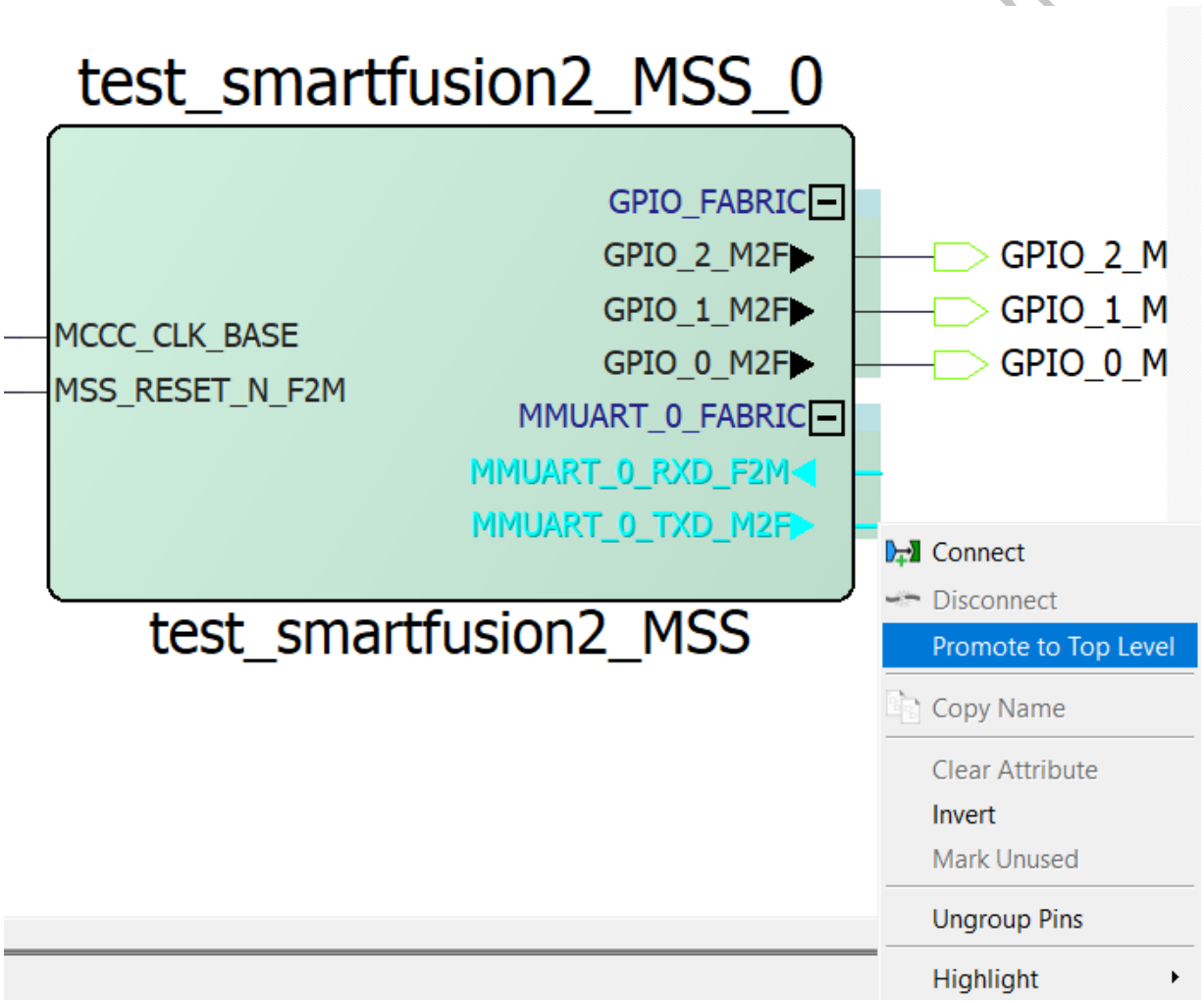
Una vez configurado (dándole al icono amarillo con el engranaje), nos genera los dos pines exteriores de la UART, Tx y Rx. Para verlo hace falta actualizar el bloque (clic derecho, *Update Instance*)



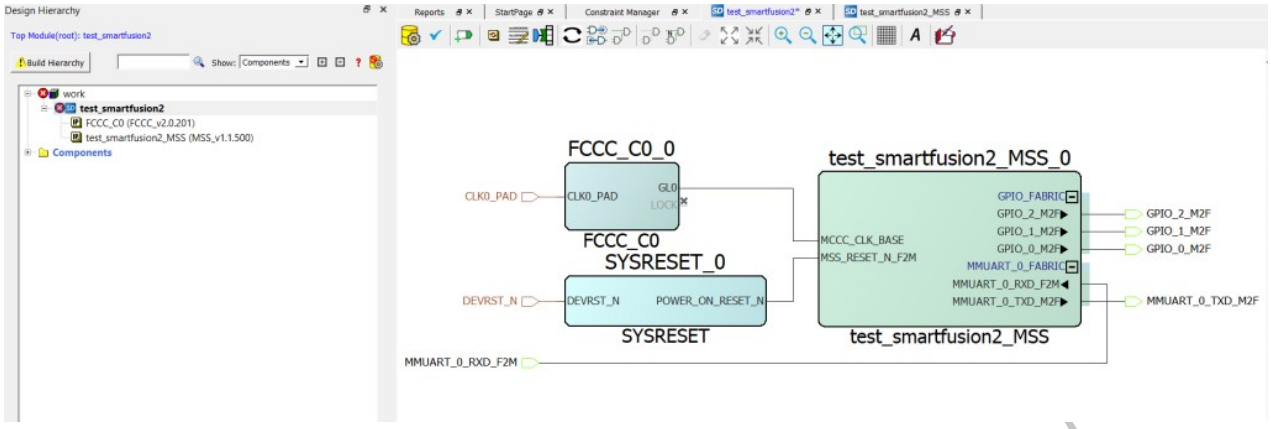
Una vez actualizado, aparecen los dos nuevos pines.



Para sacarlos al exterior del SoC se marca los pines y se selecciona la opción *Promote to Top Level*.



Esto nos genera las etiquetas exteriores de la UART. Ahora solo hay que actualizar el diagrama, para ello se le da clic al icono amarillo con el engranaje y después a *Build Hierarchy* en *Design Hierarchy.*



Una vez esté todo el diagrama configurado, sintetizamos el proyecto. Una vez sintetizado configuramos los pines en *Manage Contraints*, y le damos a *Edit* con el *I/O Editor.*

Port View [active] | Pin View | Package View | Floorplanner View | Netlist Viewer - Hier | Netlist Viewer - Flat

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode
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	MMUART_0_RXD_F2M	INPUT	LVCMOS25		☐	INBUF	--	TRISTATE
7	MMUART_0_TXD_M2F	OUTPUT	LVCMOS25		☐	OUTBUF	--	TRISTATE

Ahora le damos lo pines que vamos a utilizar para la UART.

NOTA: RX es la entrada del SoC y TX es la salida del SoC. Lo digo porque la UART a veces puede ser un lio.

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode
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	MMUART_0_RXD_F2M	INPUT	LVCMOS33	1	☑	INBUF	Bank7	TRISTATE
7	MMUART_0_TXD_M2F	OUTPUT	LVCMOS33	2	☑	OUTBUF	Bank7	TRISTATE

Una vez se han seleccionado los pines, se genera el bitstream *(recordad, generar el mapa de memoria en Generate Memory Map).*

Con el bitstream generado nos vamos a estas dos opciones.

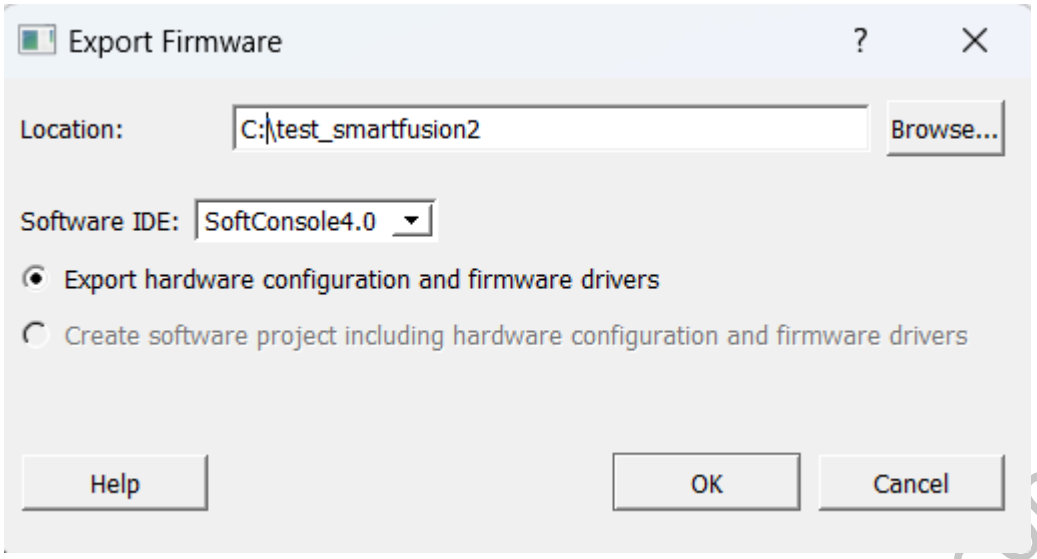
La primera nos indica que drivers vamos a exportar (*Configure Firmware Cores*). En caso de no haber exportado nunca los drivers de la UART, tenemos que descargarlos.

	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_MMUART_Driver*	*2.1.1(*	*test_smartfusion2_MSS:MMUART_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

Una vez descargados, ya los tenemos.

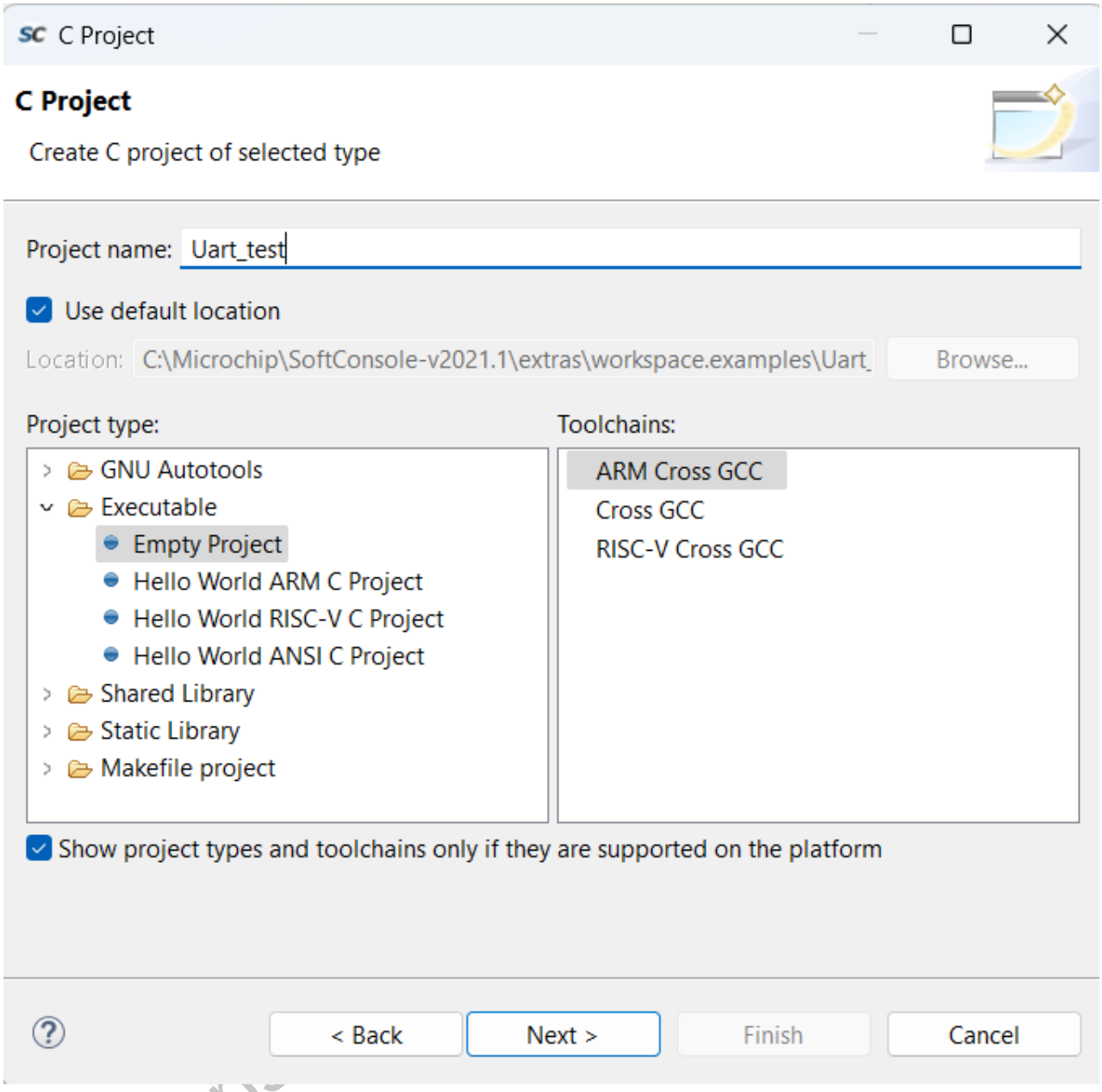
	Generate		Instance Name	Core Type	Version	Compatible Hardware Instance
1	☑		SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.1(	test_smartfusion2_MSS
2	☑		SmartFusion2_MSS_GPIO_Driver_0	SmartFusion2_MSS_GPIO_Driver	2.1.1(	test_smartfusion2_MSS:GPIO
3	☑		SmartFusion2_MSS_HPDMA_Driver_0	SmartFusion2_MSS_HPDMA_Driver	2.2.1(	test_smartfusion2_MSS
4	☑		SmartFusion2_MSS_MMUART_Driver_0	SmartFusion2_MSS_MMUART_Driver	2.1.1(	test_smartfusion2_MSS:MMUART_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

Ahora en la segunda opción (*Export Firmware*) nos va a preguntar dónde queremos exportar los drivers.

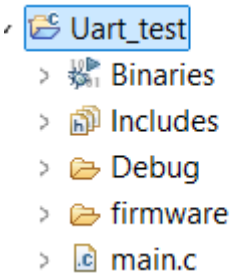


Configuración en SoftConsole

Una vez tenemos los drivers de la aplicación, creamos un proyecto en C en el SoftConsole *(todo el procedimiento está descrito en la entrada referenciada del principio)*.



Una vez creado el proyecto, importamos los drivers (carpeta *firmware*) y creamos un fichero principal (*main.c*).



Proyecto

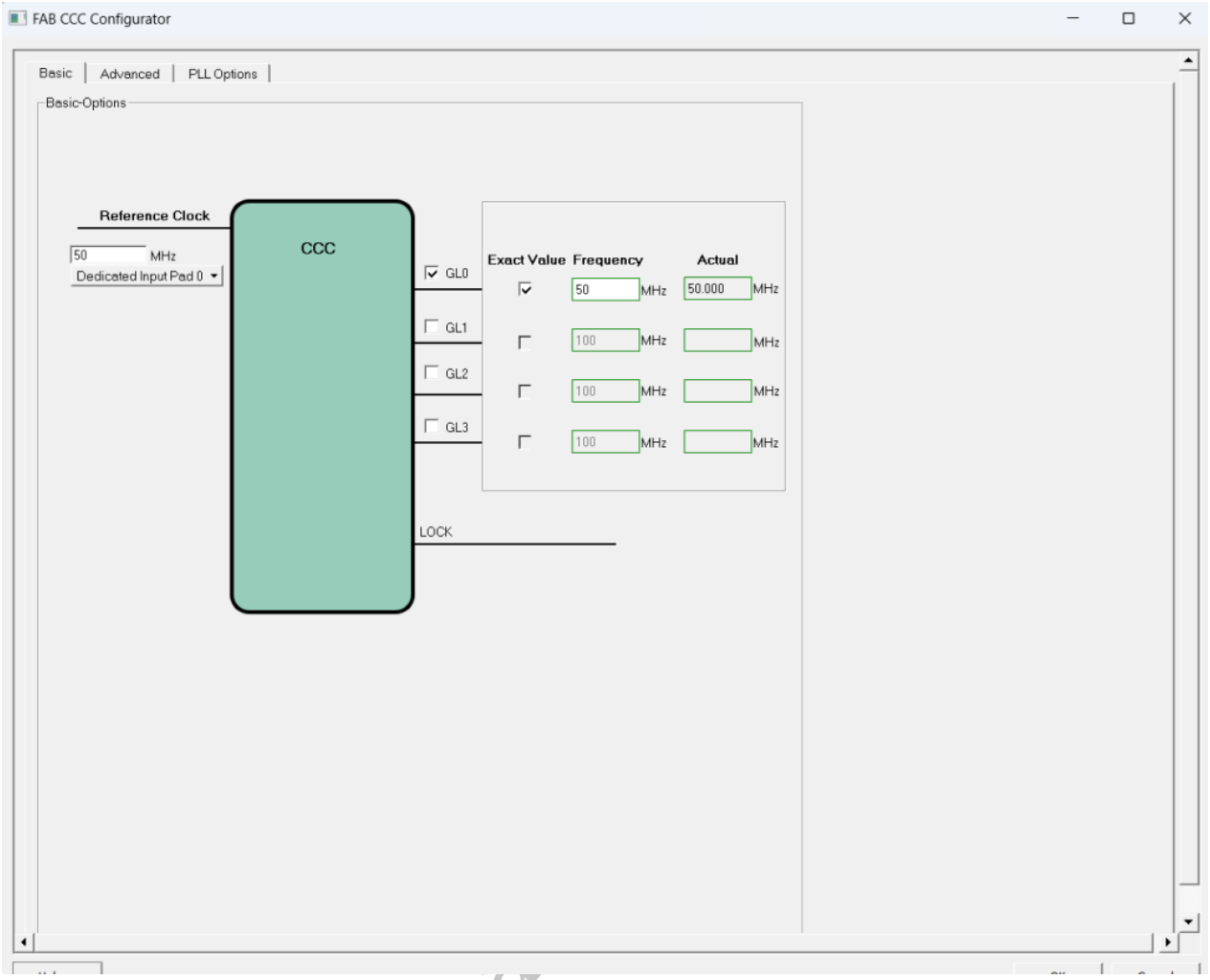
El proyecto que vamos a crear para configurar la UART es muy simple.

El proyecto es un simple loopback que devuelve los bytes recibidos por la UART por la misma UART.

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

#define RX_BUFF_SIZE    1

uint8_t g_rx_buff[RX_BUFF_SIZE];

void uart0_rx_handler(mss_uart_instance_t * this_uart){
    MSS_UART_get_rx(this_uart, &g_rx_buff[0], sizeof(g_rx_buff));

    MSS_UART_polled_tx(this_uart, g_rx_buff, 1);
}


void main(){
    MSS_UART_init(&g_mss_uart0,
            MSS_UART_115200_BAUD,
            MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

    MSS_UART_set_rx_handler(&g_mss_uart0,
                            uart0_rx_handler,
                            MSS_UART_FIFO_SINGLE_BYTE);

    while(1);

}
```

Para ello hay una primera parte que es la que configura la UART a 115200 baudios (*MSS_UAR_115200_BAUD*) y que también configura la función de interrupción (*uart0_rx_handler*) que saltará cuando llega un dato por la UART.

NOTA: debido a un fallo que tiene Libero a la hora de exportar los drivers. Para que las constantes de frecuencia de la UART (MSS_UAR_115200_BAUD, etc) se cumplan, el reloj de entrada al SoC tiene que ser de 50MHz. Si el reloj no es de esta frecuencia, la UART se ve alterada y pero las constantes no, lo cuál es un fallo de diseño por parte de Libero.
Ejemplo del fallo, si el reloj en vez de 50MHz es de 100MHz, lo que se hace es doblar la frecuencia de la UART, por lo que al utilizar la constante *MSS_UART_115200_BAUD* realmente la UART iría al doble de velocidad (230400 baudios). Este detalle es importante a la hora de configurar la UART con los baudios correctos.



Y luego la otra parte es la función de interrupción, que lo único que hace es recoge el dato que entra por la UART (*MSS_UART_get_rx*) y lo mete en un buffer de 1 byte (g_rx_buff), y luego envía el dato de este buffer por la UART de vuelta (*MSS_UART_polled_tx*).

NOTA: los drivers importados nos permiten trabajar con las dos UARTs, porque son drivers genéricos, por lo que si quieres utilizar la *UART1* los drivers te lo van a permitir, pero al no estar configurada en Libero, la *UART1* no va a funcionar.

NOTA 2: dentro de los drivers de la UART hay más funciones de configuración.

Código

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

#define RX_BUFF_SIZE    1

uint8_t g_rx_buff[RX_BUFF_SIZE];

void uart0_rx_handler(mss_uart_instance_t * this_uart){
    MSS_UART_get_rx(this_uart, &g_rx_buff[0], sizeof(g_rx_buff));
```

```
    MSS_UART_polled_tx(this_uart, g_rx_buff, 1);
}


void main(){
    MSS_UART_init(&g_mss_uart0,
            MSS_UART_115200_BAUD,
            MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

    MSS_UART_set_rx_handler(&g_mss_uart0,
                            uart0_rx_handler,
                            MSS_UART_FIFO_SINGLE_BYTE);

    while(1);

}
```

Para poder compilar y depurar este código hay que seguir las indicaciones de la entrada que utilizamos como referencia.

Una vez configurado todo, y con el bitstream grabado en el SoC, podemos depurar el código de la UART.