# How to configure the UART of a SmartFusion2



Created by: David Rubio G.

Blog post: https://soceame.wordpress.com/2025/03/10/how-to-configure-the-uart-of-a-smartfusion2/

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

GitHub: https://github.com/DRubioG

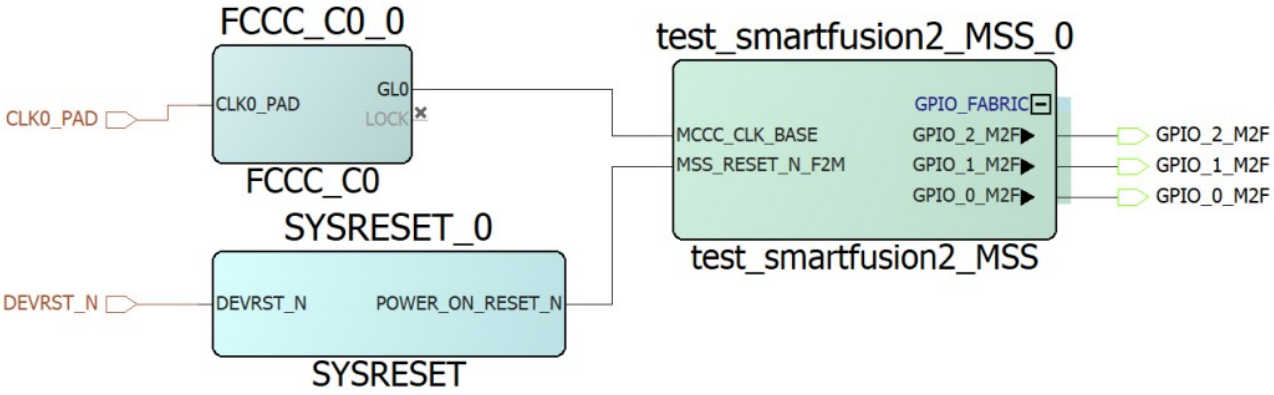
Last modification date: 10/03/25

This project is based on a previous post that we will use as a reference to focus on the most important concepts of UART configuration for a SmartFusion2.

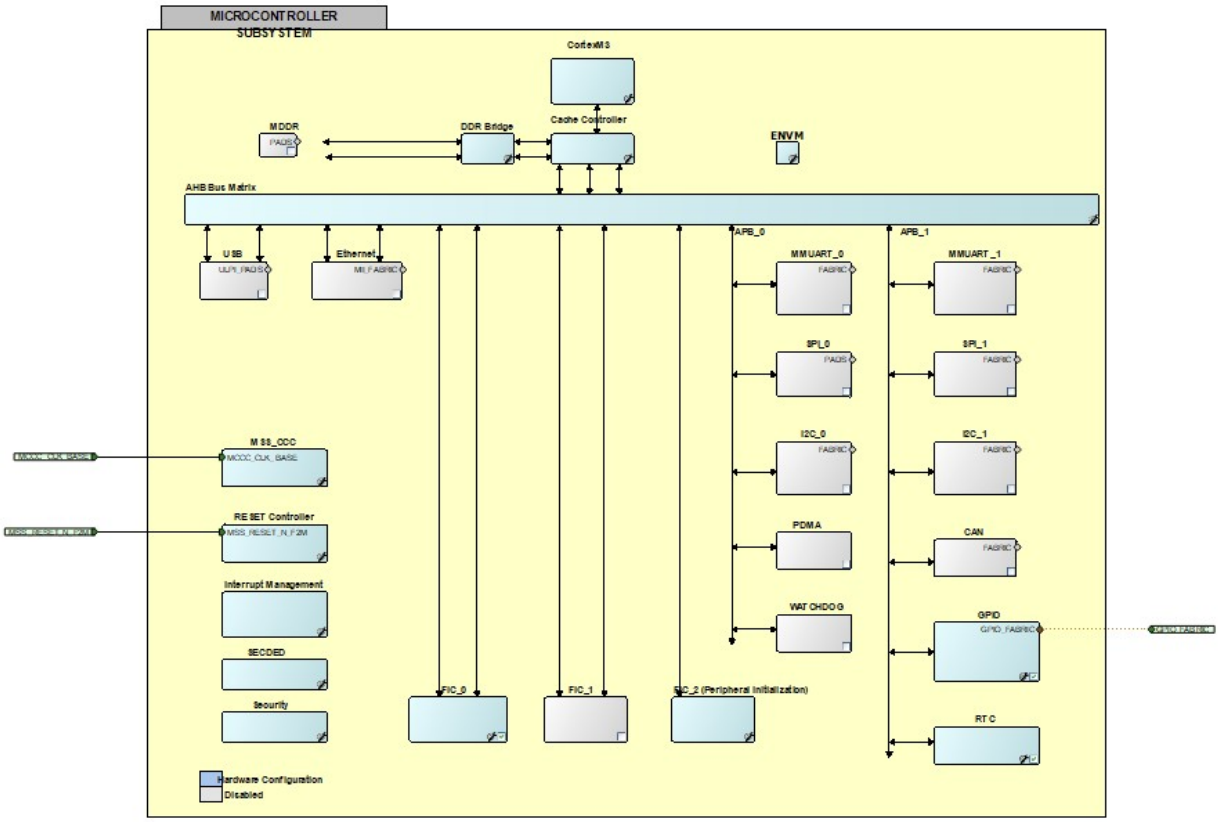
The post is used as a reference, so much of the configuration is explained here.
Configuration in Libero

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

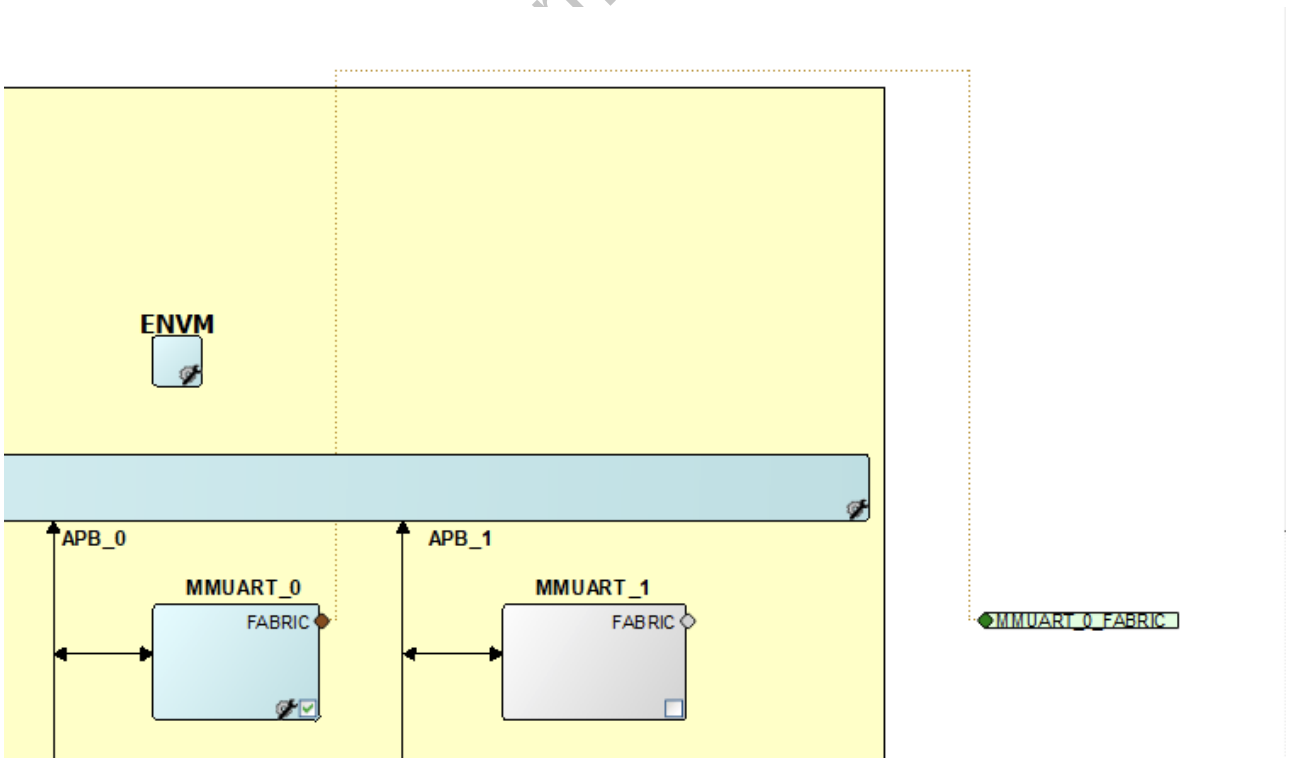
To configure a UART we rely on the second form of configuration explained in the previous post, which is the one that generates the following diagram.




This diagram only has the GPIOs configured.

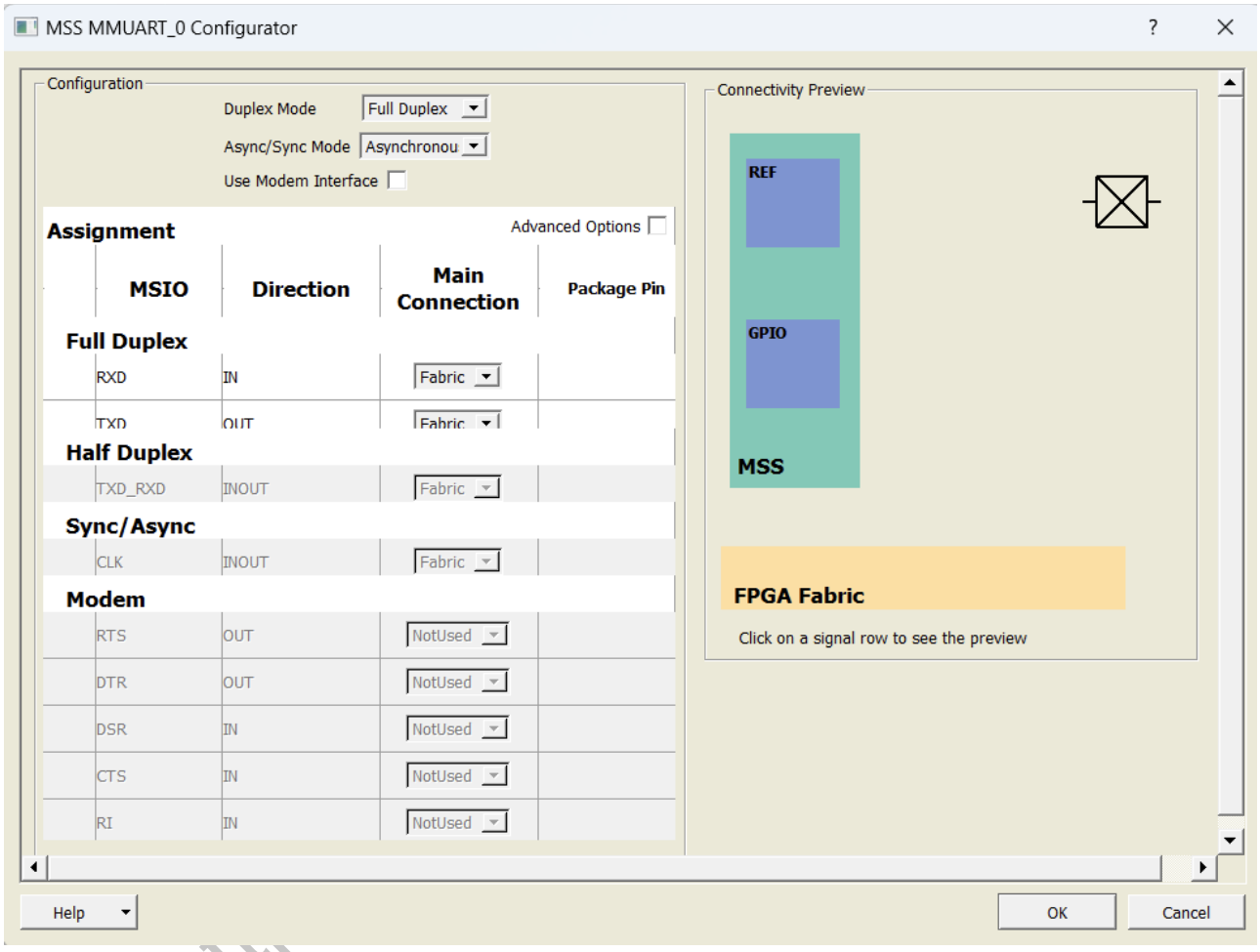



In this case we are going to focus on configuring UART0, *(UART1 is configured the same way)*. To do this, you only have to check the UART0 box.

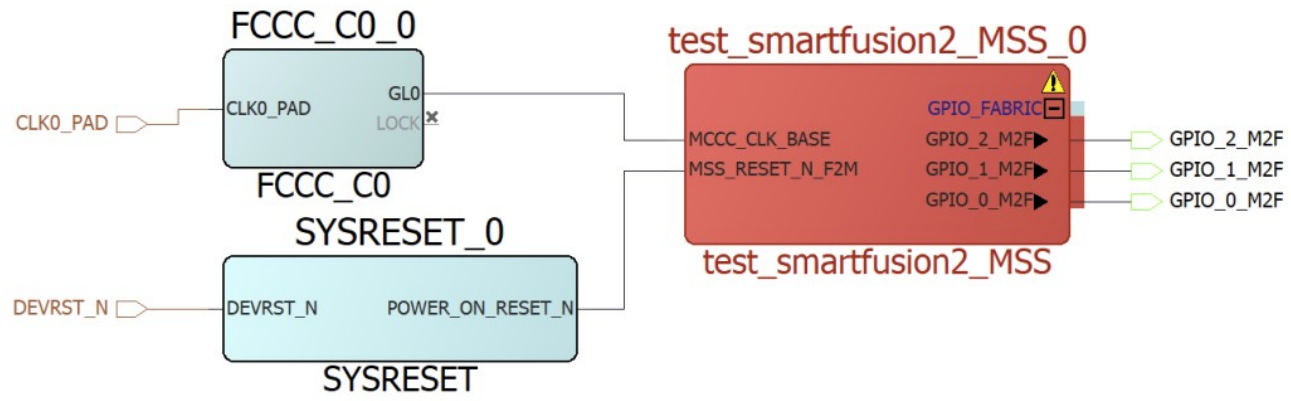


When activated, it generates a label, *MMUART_0_FABRIC*, which tells us that there is a UART to the outside through the *Fabric* (programmable logic), so we will be able to select the output pins of that UART that we want.

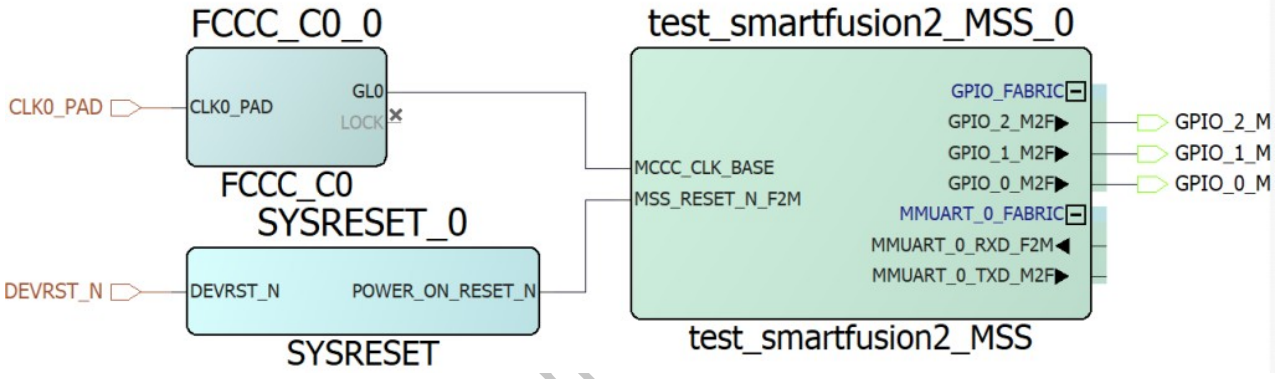
**NOTE**: To configure the UART so that it does not go out through the *Fabric*, open the UART that has been previously activated, and in the *Main Connection* column, select the output through the specific pins of that UART that the chip has *(assuming that it has specific pins, if not, to the Fabric).*



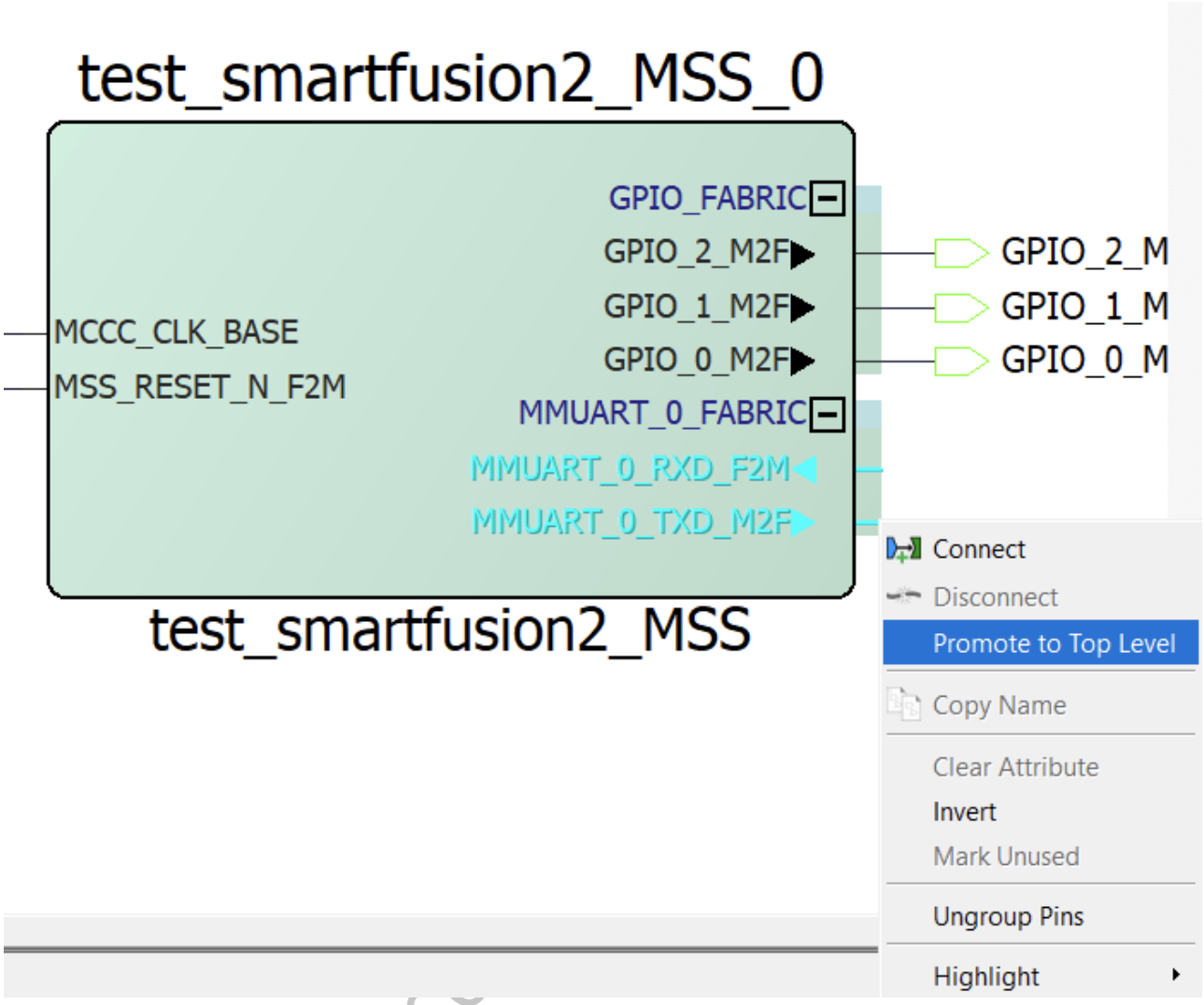

Once configured (by clicking on the yellow icon with the gear), it generates the two external pins of the UART, Tx and Rx. To see it, you need to update the block (right click, *Update Instance*)



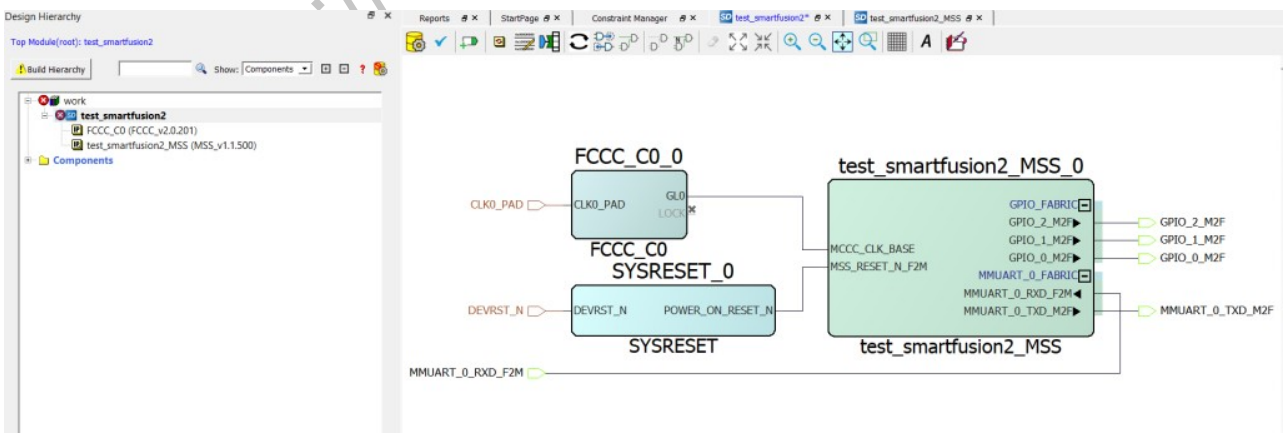

Once updated, the two new pins appear.




To take them out of the SoC, mark the pins and select the *Promote to Top Level* option.




This generates the external labels of the UART. Now you just need to update the diagram, to do this, click on the yellow icon with the gear and then on *Build Hierarchy* in *Design Hierarchy.*




Once the entire diagram is configured, we synthesize the project. Once synthesized, we configure the pins in *Manage Contracts*, and click *Edit* with the *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 |

Now we give the pins that we are going to use for the UART.

**NOTE**: RX is the SoC input and TX is the SoC output. I say this because the UART can sometimes be a mess.

| | 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 |

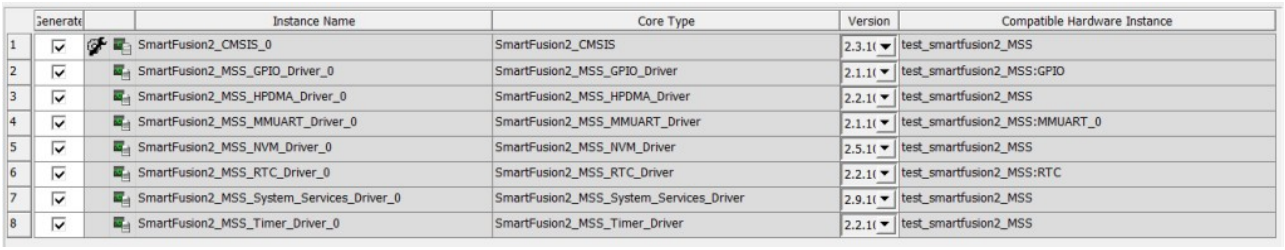
Once the pins have been selected, the bitstream is generated *(remember, generate the memory map in Generate Memory Map)*.

With the generated bitstream we go to these two options.


The first one tells us which drivers we are going to export *(Configure Firmware Cores)*. If we have never exported the UART drivers, we have to download them.

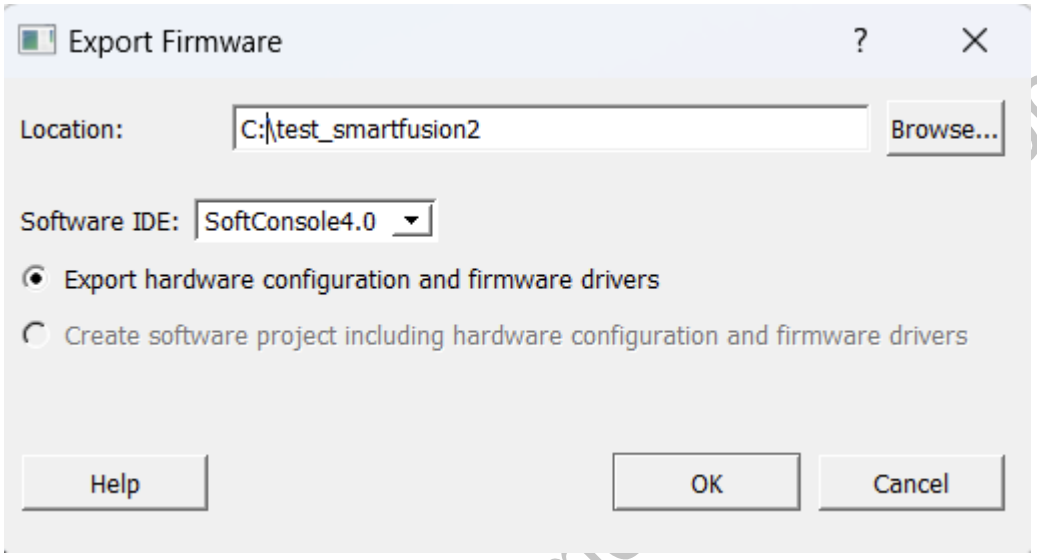
| | Generate | | Instance Name | Core Type | Version | Compatible Hardware Instance |
|---|---|---|---|---|---|---|
| 1 | ☑ | | SmartFusion2_CMSIS_0 | SmartFusion2_CMSIS | 2.3.1( | test_smartfusion2_MSS |
| 2 | ☑ | | SmartFusion2_MSS_GPIO_Driver_0 | SmartFusion2_MSS_GPIO_Driver | 2.1.1( | test_smartfusion2_MSS:GPIO |
| 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 |

Once downloaded, we already have them.



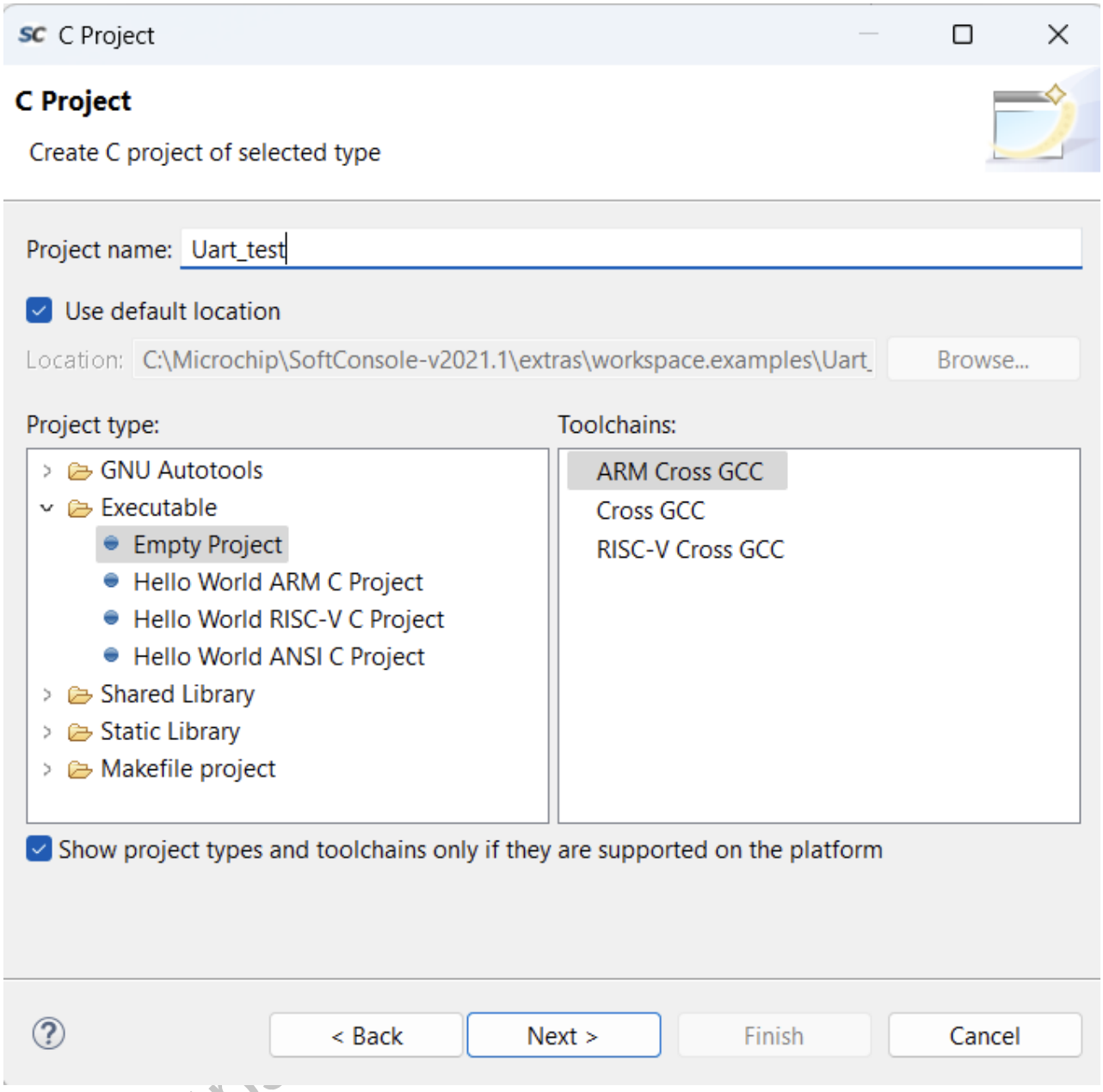
| | Generate | | Instance Name | Core Type | Version | Compatible Hardware Instance |
|---|---|---|---|---|---|---|
| 1 | ☑ | | SmartFusion2_CMSIS_0 | SmartFusion2_CMSIS | 2.3.1( | test_smartfusion2_MSS |
| 2 | ☑ | | SmartFusion2_MSS_GPIO_Driver_0 | SmartFusion2_MSS_GPIO_Driver | 2.1.1( | test_smartfusion2_MSS:GPIO |
| 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 |

Now in the second option *(Export Firmware)* it will ask us where we want to export the drivers.

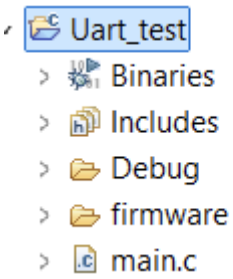



# Configuration in SoftConsole

Once we have the application drivers, we create a C project in SoftConsole *(the entire procedure is described in the referenced entry at the beginning)*.




Once the project is created, we import the drivers (firmware folder) and create a main file (*main.c*).



# Project

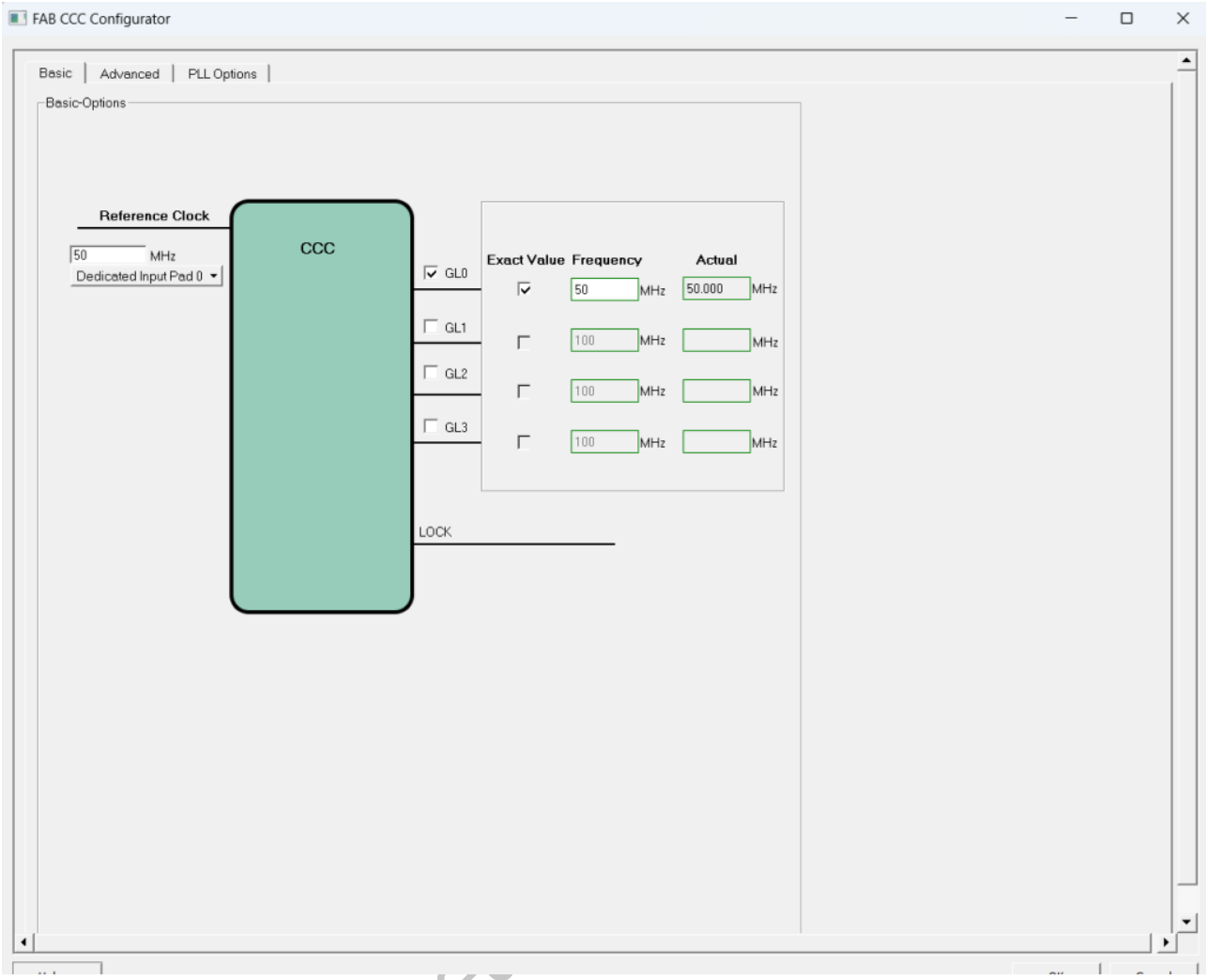
The project that we are going to create to configure the UART is very simple.

The project is a simple loopback that returns the bytes received by the UART through the same 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);

}
```

For this there is a first part that configures the UART to 115200 baud *(MSS_UAR_115200_BAUD)* and that also configures the interrupt function *(uart0_rx_handler)* that will be triggered when data arrives through the UART.

**NOTE**: due to a fault that Libero has when exporting the drivers. In order for the UART frequency constants (MSS_UAR_115200_BAUD, etc.) to be met, the input clock to the SoC has to be 50MHz. If the clock is not of this frequency, the UART is altered but the constants are not, which is a design flaw on Libero's part.
**Example of the flaw**, if the clock is 100MHz instead of 50MHz, what is done is double the frequency of the UART, so when using the constant *MSS_UART_115200_BAUD* the UART would actually go at double speed (230400 baud). This detail is important when configuring the UART with the correct baud rate.




And then the other part is the interrupt function, which only collects the data coming in through the UART (*MSS_UART_get_rx*) and puts it into a 1-byte buffer (g_rx_buff), and then sends the data from this buffer back through the UART (*MSS_UART_polled_tx*).

**NOTE**: the imported drivers allow us to work with both UARTs, because they are generic drivers, so if you want to use *UART1* the drivers will allow you to do so, but since it is not configured in Libero, *UART1* will not work.

**NOTE 2**: within the UART drivers there are more configuration functions.

# Code

```
#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);

}
```

To compile and debug this code, you must follow the instructions in the entry we used as a reference.

Once everything is configured, and with the bitstream recorded in the SoC, we can debug the UART code.