Tutorial de una PolarFire SoC de Microchip

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2025/01/07/tutorial-de-una-polarfire-soc-de-microchip/

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

GitHub: https://github.com/DRubioG

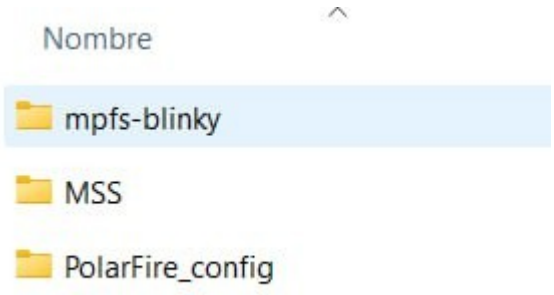
Fecha última modificación: 24/02/2025

Antes de empezar

Si también trabajas con un PolarFire SoC Discovery Kit te recomiendo que te descargues este repositorio de GitHub. Donde tiene la configuración base para poder: habilitar el procesador MSS y también tienes un proyecto de ejemplo que es con el que trabajaremos, que también incluye todos los drivers que puedes necesitar para una PolarFire.
Si no trabajas con este kit también te recomiendo descargarlo porque el proyecto base del SoftConsole es común a todas las placas.

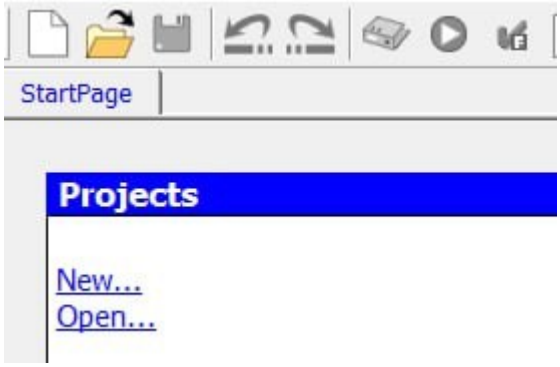
https://github.com/DRubioG/Polarfire_basic_project

Solo vamos a necesitar el mpfs-blinky y el PolarFire_config. El otro fichero es como el PolarFire_config pero aplicado al Discovery Kit completo.

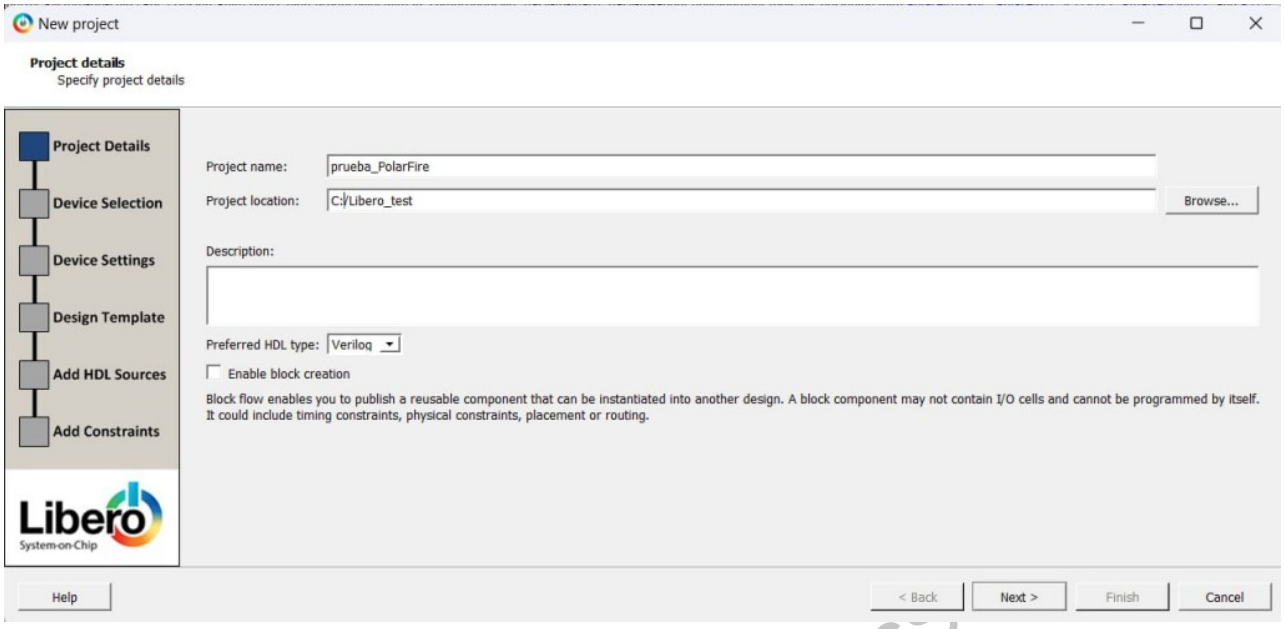


Libero

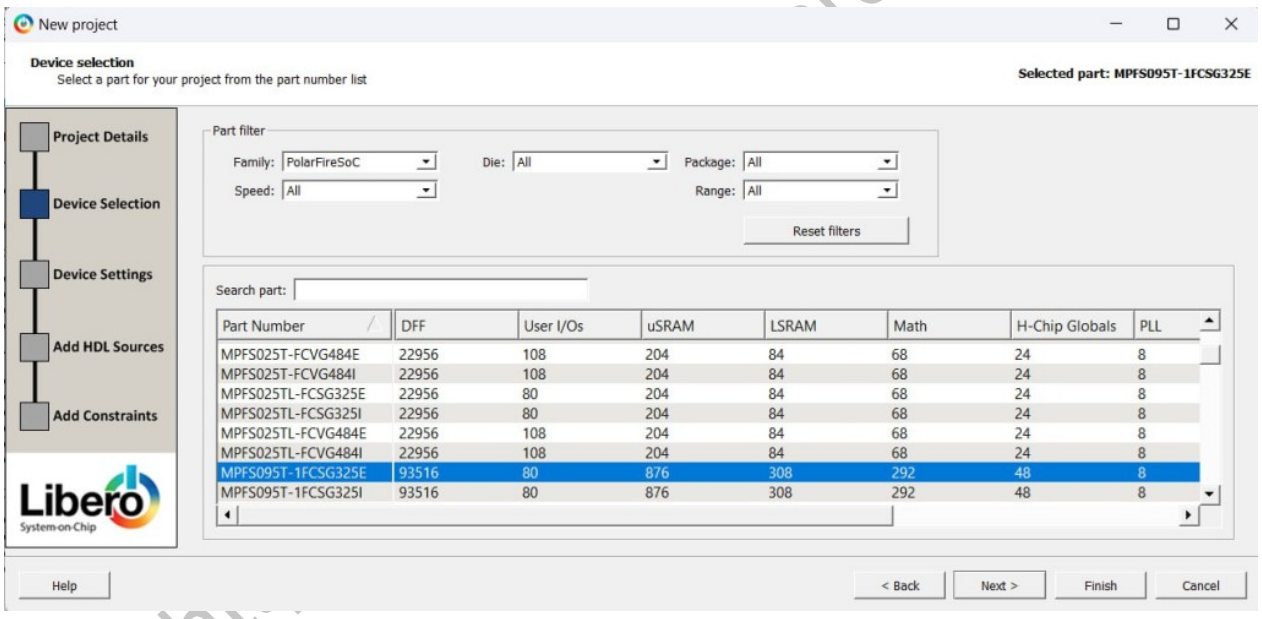
Primer paso es crear un nuevo proyecto en Libero.



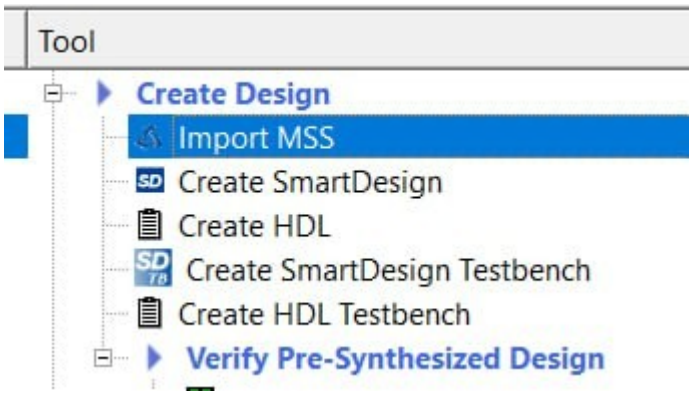
Le damos un nombre y una ubicación al proyecto.



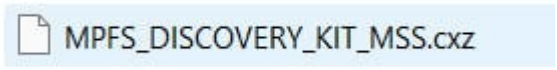
Después elegimos el chip y le damos a *Finish*.



Le damos a *Import MSS* e importamos un .cxz que es el del repositorio.



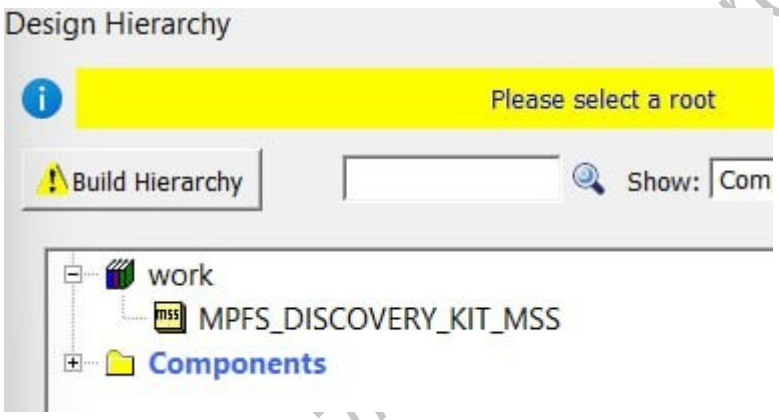
El que hay que importar.



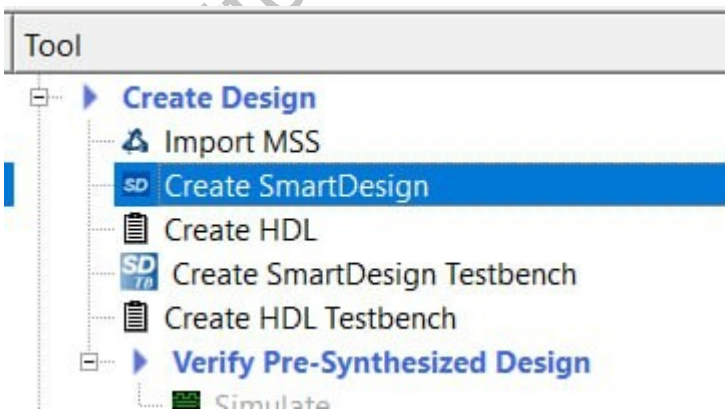
Para crear el tuyo propio tienes este tutorial.

https://soceame.wordpress.com/2025/01/07/configurar-los-perifericos-de-un-polarfire-soc/

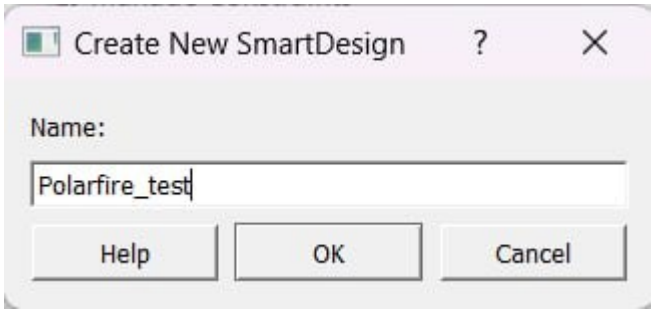
En la jerarquía tenemos lo siguiente.



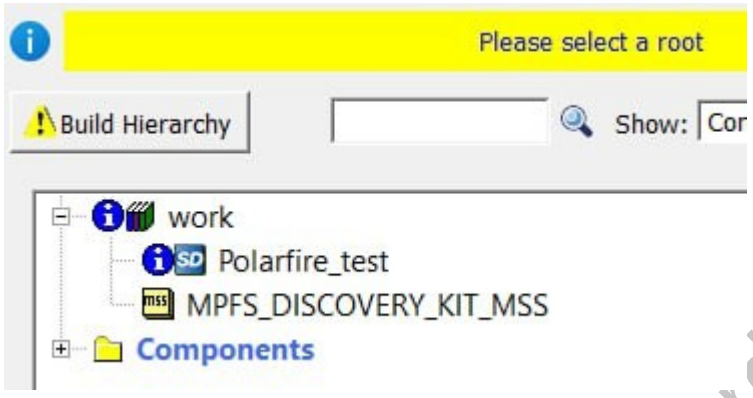
Ahora creamos un SmartDesign.



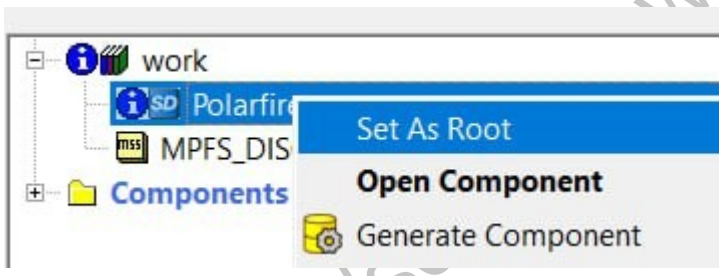
Lo llamamos *Polarfire_test*.



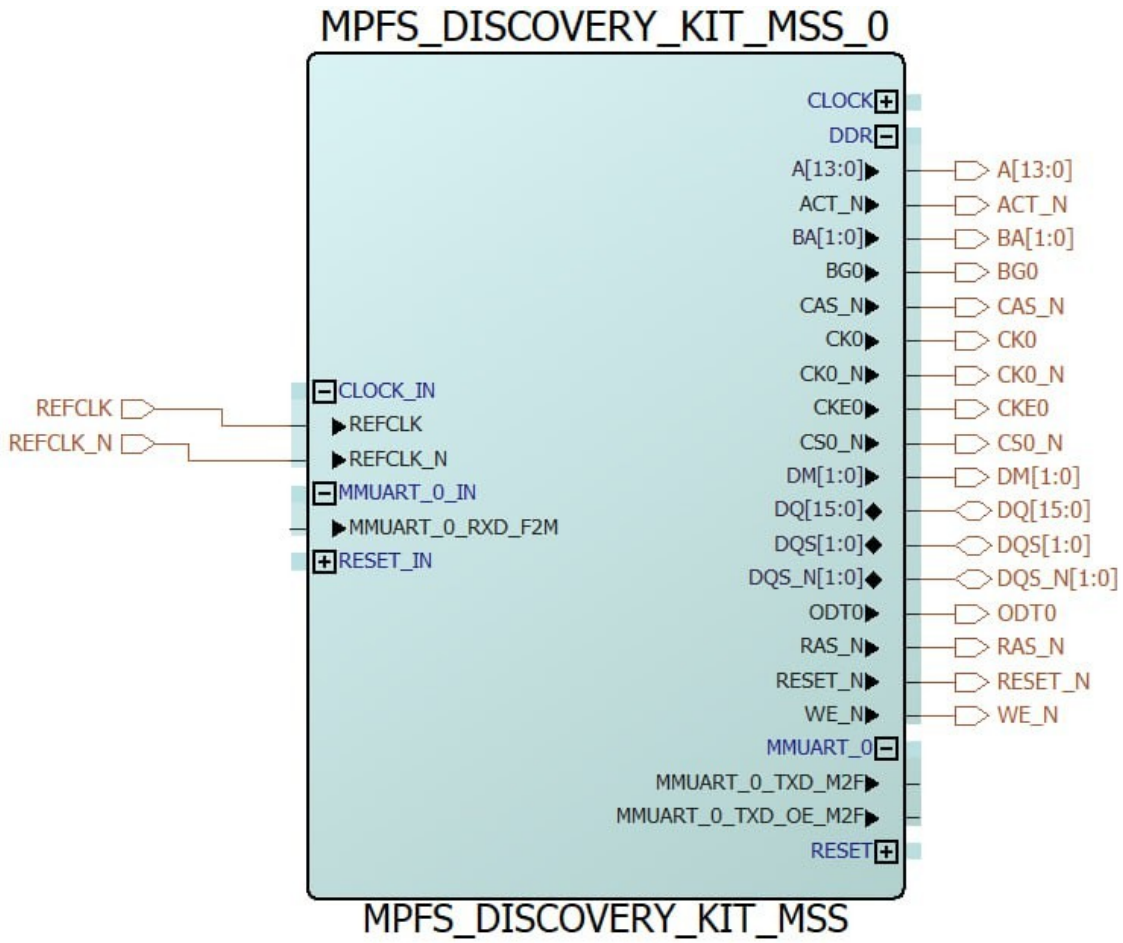
Se queda una estructura así.



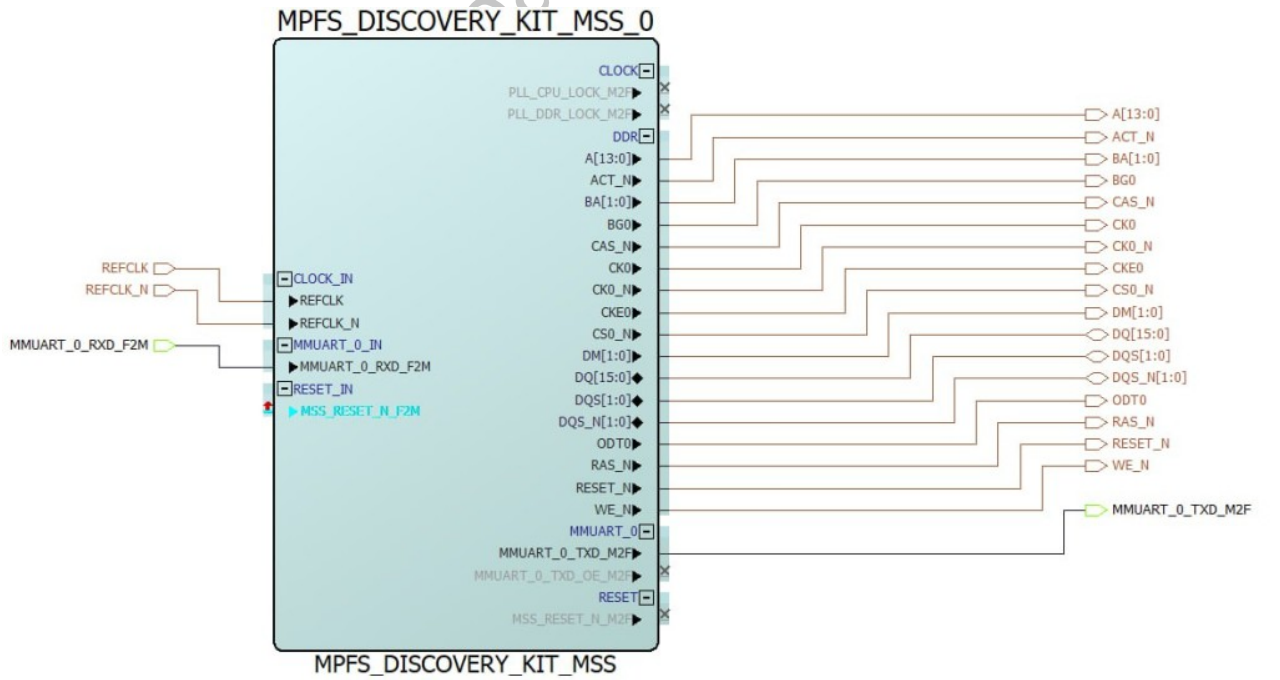
Seleccionamos como *Root* el *SmartDesign*.



Ahora arrastramos al SmartDesign el modelo de PolarFire SoC.



Ahora le conectamos solo los puertos de la UART que están como Fabric.

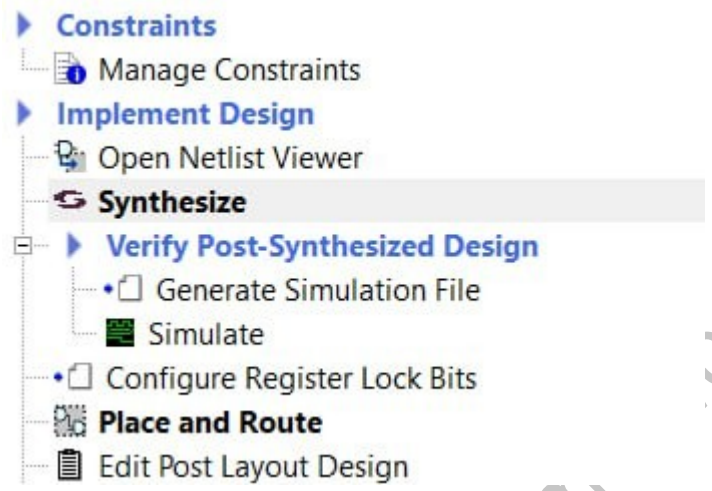


Ahora nos queda una estructura como la siguiente.

Si quieres conectar otros bloques IP lo puedes hacer siguiendo esto.

https://soceame.wordpress.com/2025/01/07/como-conectar-un-bloque-ip-en-libero/

Cuando lo tengamos todo, sintetizamos.



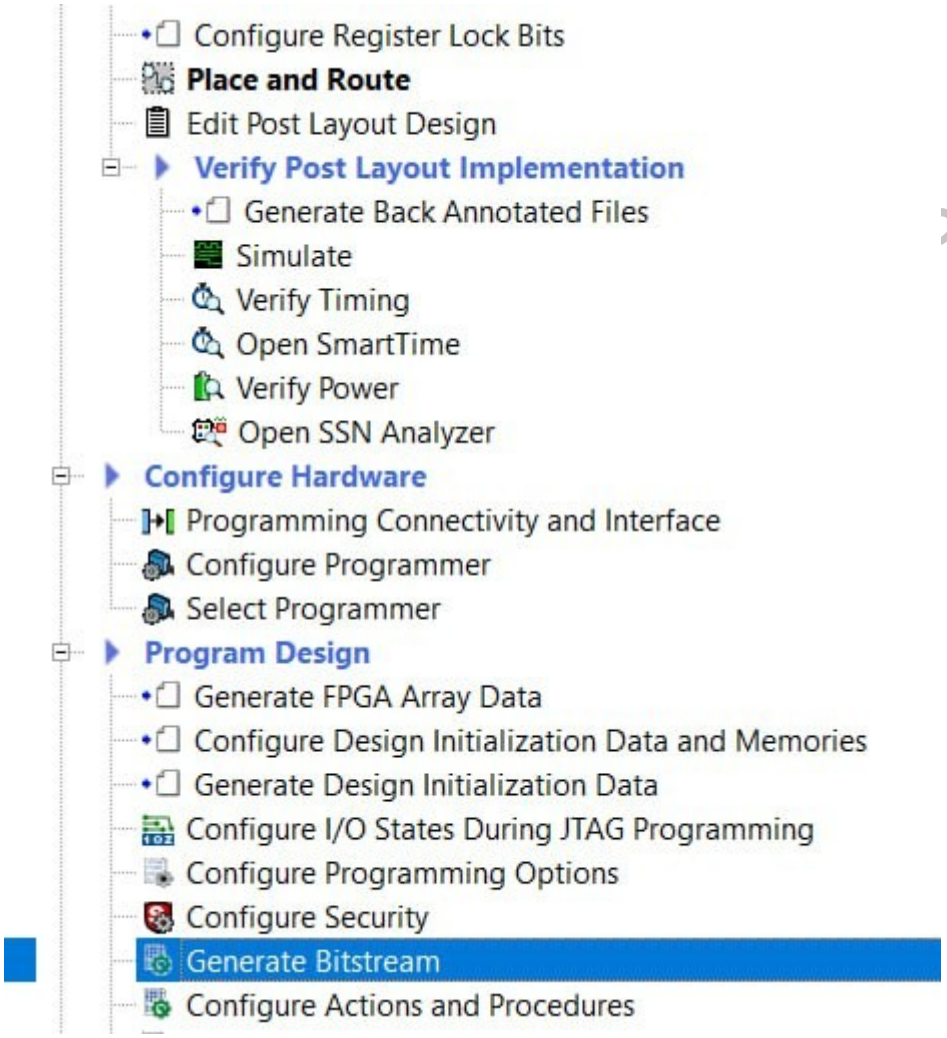
Ahora en *Manage Constraints* podemos configurar los pines de la UART como Fabric.

47	DQS_N[0]	INOUT	POD12I	Y10	☑	BIBUF_DIFF	
48	DQS[1]	INOUT	POD12I	V12	☑	BIBUF_DIFF	
49	DQS_N[1]	INOUT	POD12I	U12	☑	BIBUF_DIFF	
50	MMUART_0_RXD_F2M	INPUT	LVCMOS18		☐	INBUF	
51	MMUART_0_TXD_M2F	OUTPUT	LVCMOS18		☐	OUTBUF	
52	ODT0	OUTPUT	HSTL12I	T2	☑	IOPAD_TRI	
53	RAS_N	OUTPUT	HSTL12I	AA4	☑	IOPAD_TRI	
54	REFCLK	INPUT	LVDS25	P5	☑	INBUF_DIFF	
55	REFCLK_N	INPUT	LVDS25	P4	☑	INBUF_DIFF	

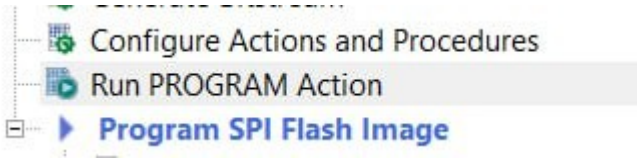
Ahora los seleccionamos.

47	DQS_N[0]	INOUT	POD12I	Y10
48	DQS[1]	INOUT	POD12I	V12
49	DQS_N[1]	INOUT	POD12I	U12
50	MMUART_0_RXD_F2M	INPUT	LVCMOS18	W21
51	MMUART_0_TXD_M2F	OUTPUT	LVCMOS18	Y21
52	ODT0	OUTPUT	HSTL12I	T2

Una vez seleccionados podemos generar el bitstream.



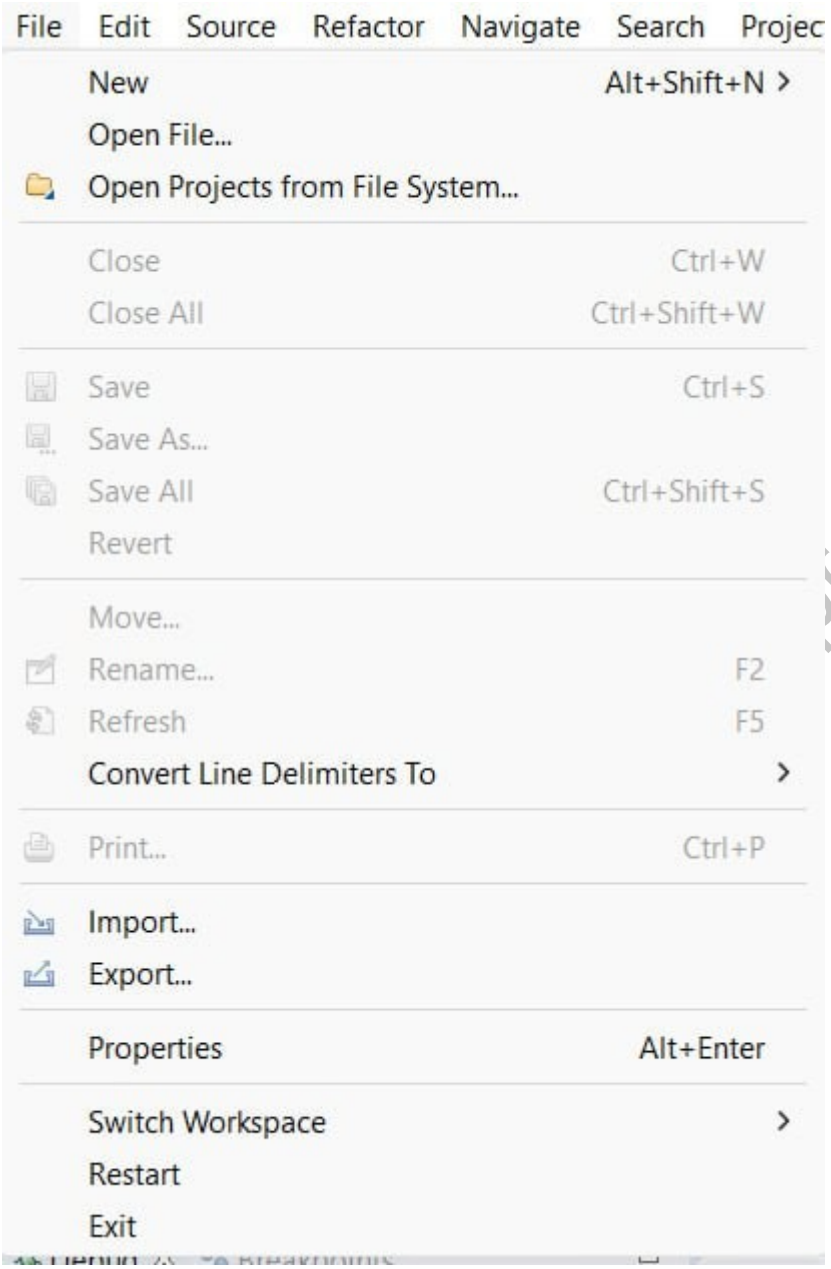
Una vez lo tengamos se lo podemos grabar al SoC.



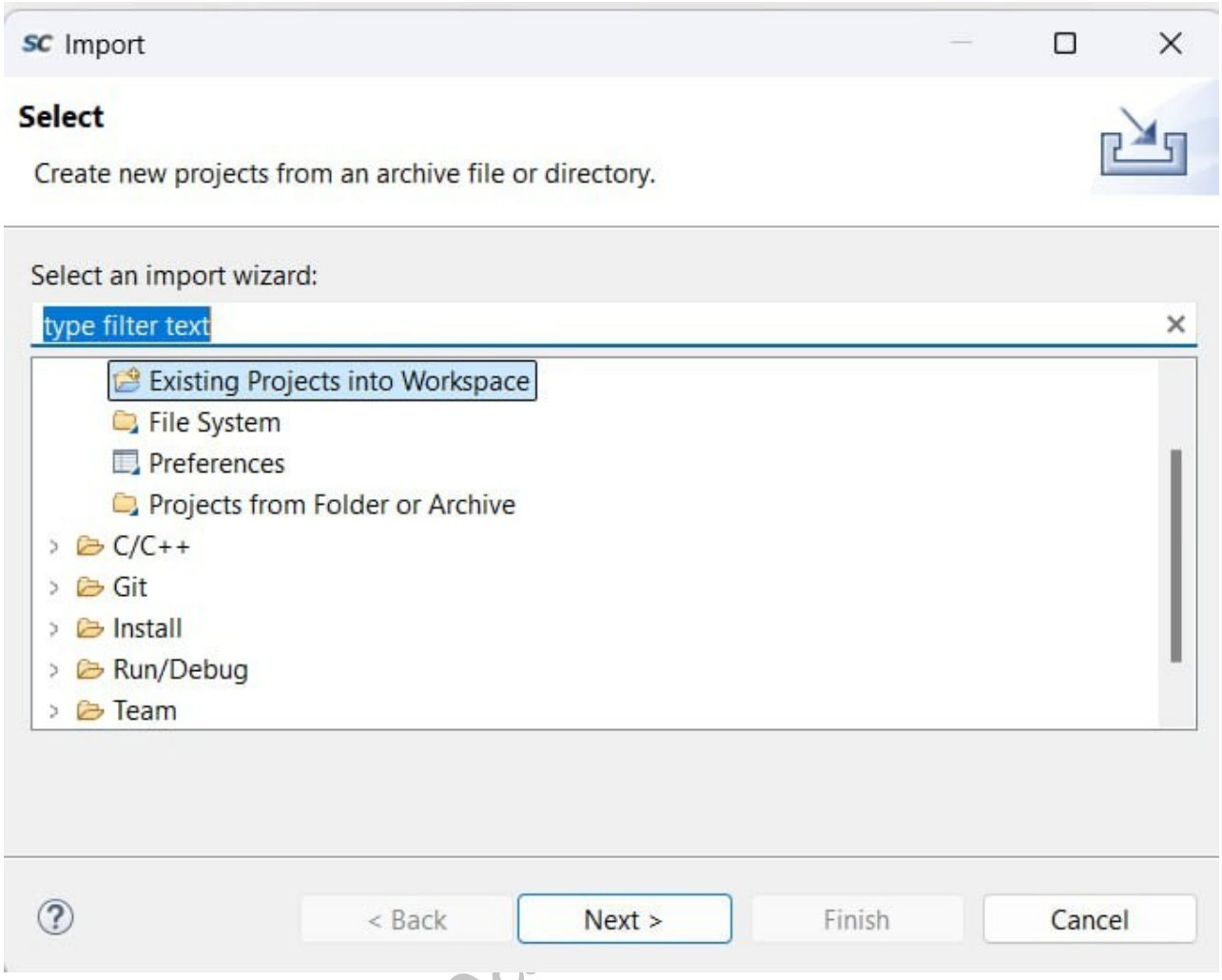
Ahora nos vamos al SoftConsole.

SoftConsole

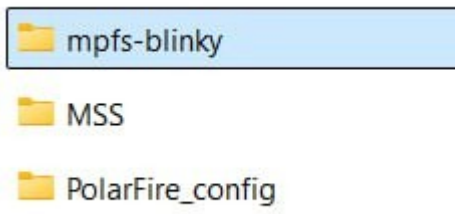
Ahora lo que hacemos es importar un proyecto, para ello le damos a *Import…*



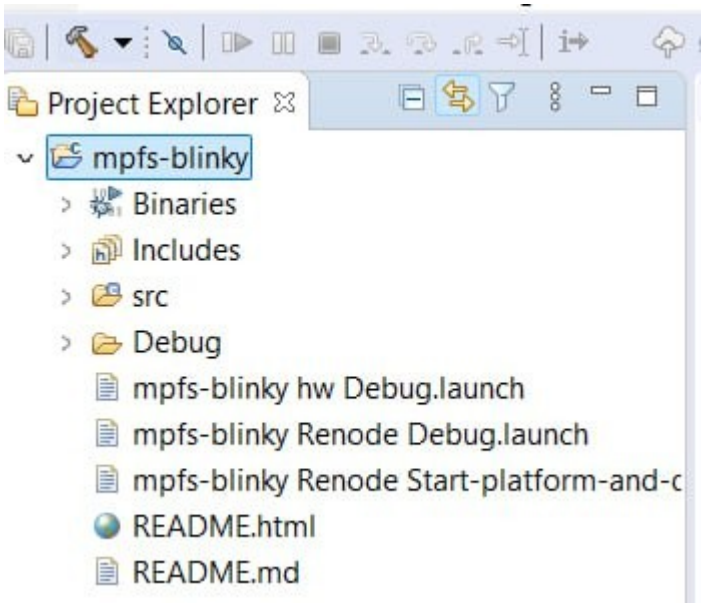
Después le demos a importar un *Existing Projects into Workspace*.



Ahora buscamos el mpfs-blinky, y lo importamos (a lo mejor tienes que bajar un directorio más para que te coja el proyecto de SoftConsole).



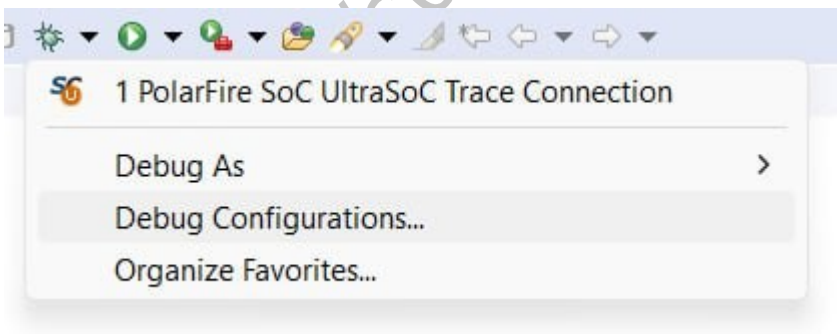
Una vez ya lo hemos importado, lo compilamos.



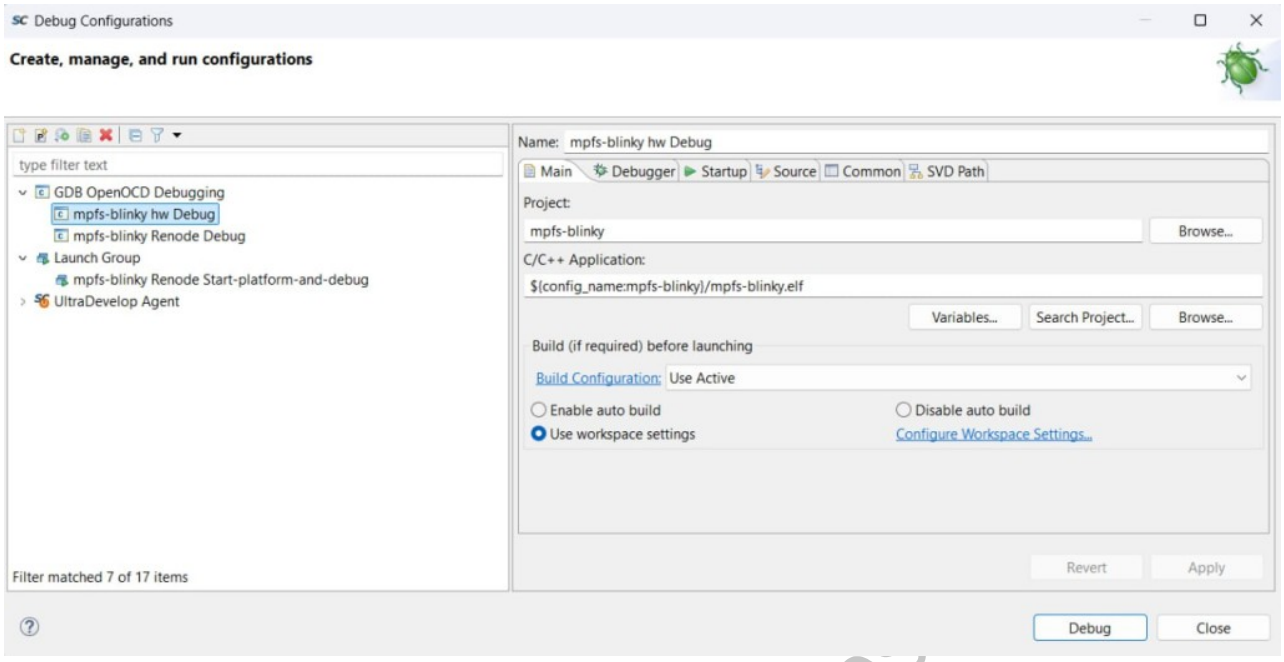
Cuando lo tengamos compilado ya lo podemos depurar.

```
Invoking: GNU RISC-V Cross Print Size
riscv64-unknown-elf-size --format=berkeley "mpfs-blinky.elf"
   text    data     bss     dec     hex filename
  19056    3648   12560   35264    89c0 mpfs-blinky.elf
Finished building: mpfs-blinky.siz


23:03:50 Build Finished. 0 errors, 0 warnings. (took 5s.689ms)
```

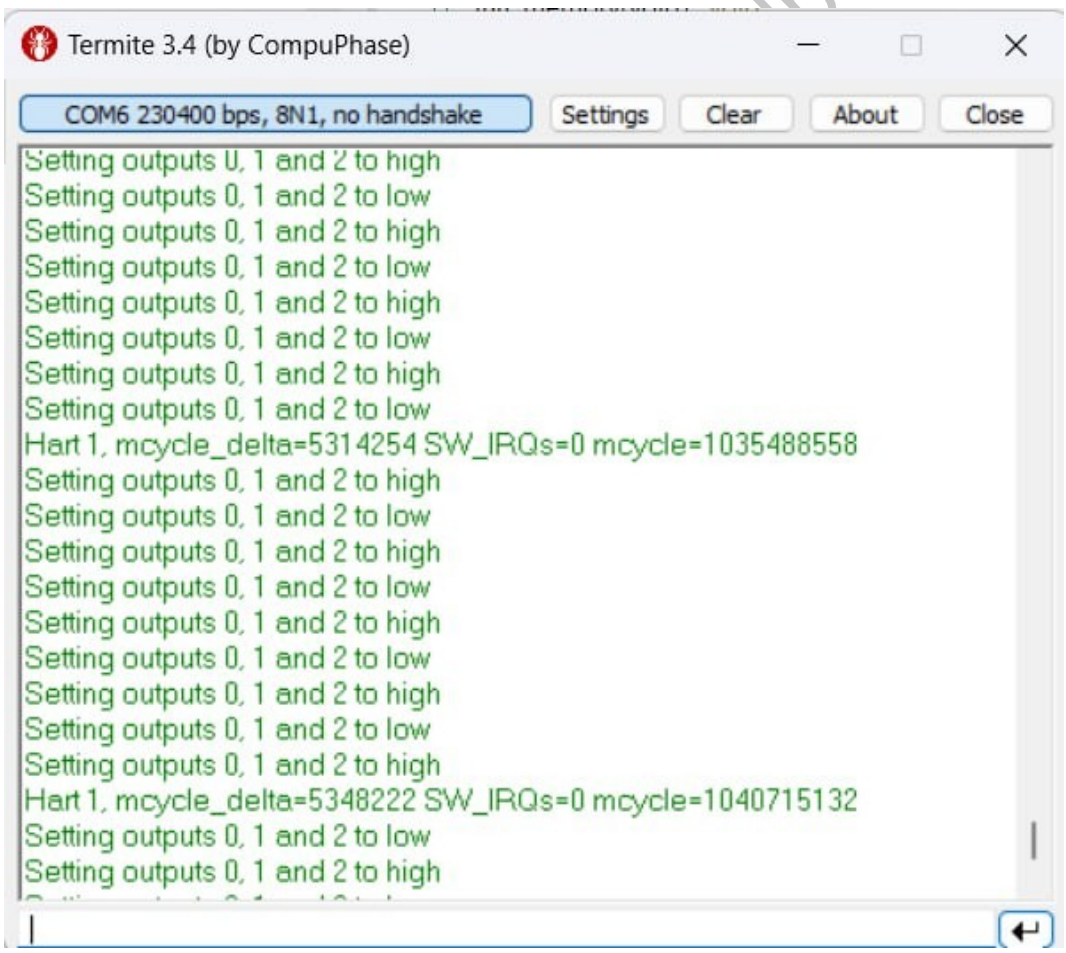
Para ello nos vamos a *Debug Configurations*…



Ahora buscamos el *mpfs-blinky hw Debug*, y lo lanzamos.



Si abrimos el terminal podemos ver que la aplicación se ejecuta correctamente.



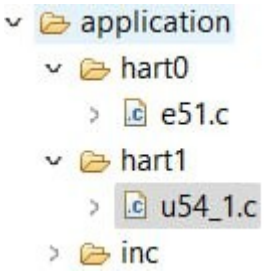
Si analizamos el código, podemos ver que hay una UART y unos GPIOs en el ejemplo. Los GPIO no funcionan *(no he conseguido que una PolarFire sea capaz de ejecutar, porque asignar sí que deja, «algo» con los GPIO desde la PolarFire)*.

```
safe_MSS_UART0_polled_tx_string("Hello World from e51 (hart 0).\r\n");

while (1)
{
    // Stay in the infinite loop, never return from main

    const uint64_t delay_loop_max = 10000;
    volatile uint64_t delay_loop_sum = 0;

    for (uint64_t i = 0; i< delay_loop_max; i++) {
        delay_loop_sum = delay_loop_sum + i;
    }

    safe_MSS_UART0_polled_tx_string("Setting outputs 0, 1 and 2 to high\r\n");

    MSS_GPIO_set_output(GPIO1_LO, MSS_GPIO_0, 1);
    MSS_GPIO_set_output(GPIO1_LO, MSS_GPIO_1, 1);
    MSS_GPIO_set_output(GPIO2_LO, MSS_GPIO_9, 1);

    for (uint64_t i = 0; i< delay_loop_max; i++) {
        delay_loop_sum = delay_loop_sum + i;
    }

    safe_MSS_UART0_polled_tx_string("Setting outputs 0, 1 and 2 to low\r\n");
    MSS_GPIO_set_output(GPIO1_LO, MSS_GPIO_0, 0);
    MSS_GPIO_set_output(GPIO1_LO, MSS_GPIO_1, 0);
    MSS_GPIO_set_output(GPIO2_LO, MSS_GPIO_9, 0);
}
```

El funcionamiento de la UART se ve fácil, pero aparece una UART intermedia. Eso proviene de aquí.

```c
void u54_1_application(void)
{
    time_benchmark_t mcycle         = { .start = 0, .end = 0, .delta = 0};
    volatile uint64_t loop_count_h1 = 0;
    const uint64_t num_loops        = 100000;
    uint64_t hartid                 = read_csr(mhartid);
    char uart_buf[100];

    while (1)
    {
        mcycle.start = readmcycle();

        for (uint64_t i = 0; i < num_loops; i++) {
            dummy_h1 = i;
        }

        sprintf(uart_buf, "Hart %ld, mcycle_delta=%ld SW_IRQs=%ld mcycle=%ld\r\n",
                hartid, mcycle.delta, count_sw_ints_h1, readmcycle());

        safe_MSS_UART0_polled_tx_string(uart_buf);

        hartid       = read_csr(mhartid);
        mcycle.end   = readmcycle();
        mcycle.delta = mcycle.end - mcycle.start;

        loop_count_h1++;
    }
}
```

Esto último que se ha visto es la ejecución en un segundo core, por lo que la aplicación solo se puede ejecutar con dos cores, e51 y e54_1,(si intentas quitar uno de los dos cores, cosa que parece fácil pero no lo es ni de lejos, corres el riesgo de que todo deje de funcionar, y que la placa se ponga «tonta», y tengas que salir de todo y retornar).



NOTA: si miras la frecuencia de los baudios de la UART, puedes ver que se ha configurado a 115200 baudios, pero el Termite está puesto a 230400, esto es debido a que este SoC a igual que las SmartFusion2 tienen un problema de diseño, que teóricamente se soluciona a nivel driver, pero que Microchip no ha querido solucionar. Este problema es que la frecuencia del core tiene que ser una específica que Microchip nunca dice, por lo que si la frecuencia que necesita el core es de 50MHz, pero lo alimentas a 100MHz, la placa dobla su frecuencia a nivel de periféricos haciendo que una UART de 115200 pase a 230400 (esto pasa en todos lo baudios de la UART). Esto se supone que se

tiene que solucionar a nivel de drivers, pero no lo hace, entonces, es importante tener esto en cuenta.

NOTA FINAL

Puede que trabajar con una PolarFire SoC se parezca a una SmartFusion2, pero no es así. Las PolarFire en el momento de escribir esto son extremadamente jodidas de manejar. Por lo que si sabes manejar una SmartFusion2 quédate con ella.