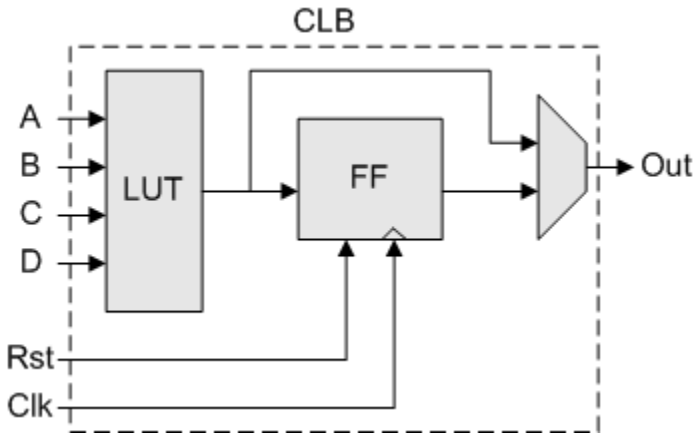
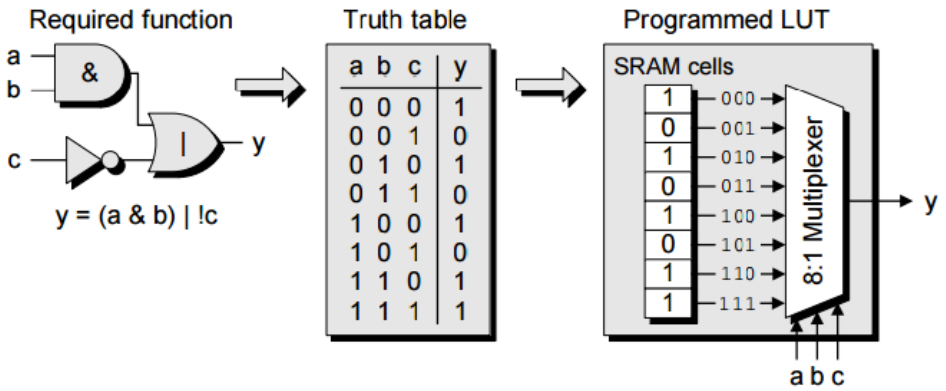
Tutorial de Iniciación a FPGAs, SystemVerilog y Vivado

Que es una FPGA

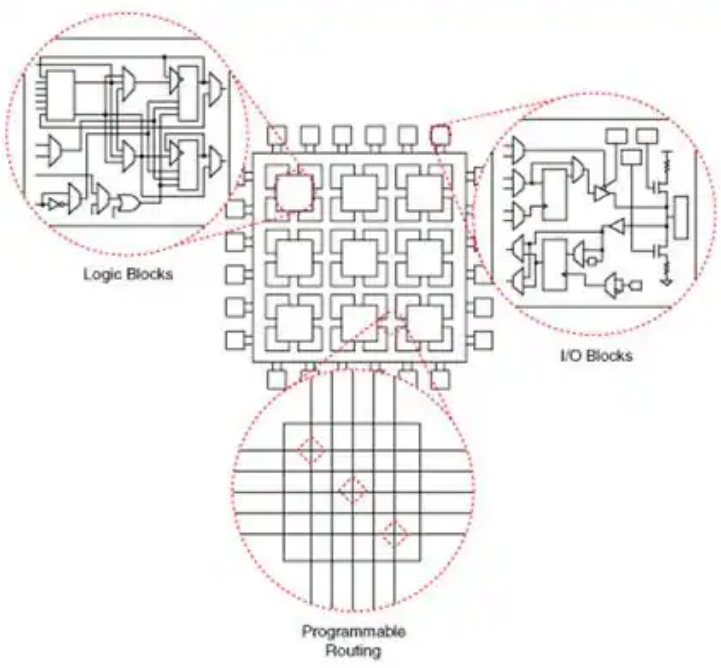
Una FPGA (Field-Programmable Gate Array) es un circuito integrado (o “chip”) que contiene una matriz de bloques lógicos configurables (CLBs) capaces de implementar funciones lógicas combinacionales (de una determinado número de entradas como máximo) y secuenciales con la posible conexión de un flip-flop D. Además, cuenta con una jerarquía de interconexiones reconfigurables que puede interconectar estos bloques o conectarlos con otros recursos disponibles en el dispositivo (como puertos, bloques dsp, memorias, etc.). El CLB está conformado por una LUT que implementa funciones lógicas combinacionales y un flip flop D como se muestra en la siguiente imagen:



Una LUT es fundamentalmente una memoria SRAM de 32 palabras de 1 bit, donde cada bit está conectado a la entrada de un multiplexor de 4x32. Las entradas del circuito se conectan a las entradas de selección del multiplexor y en la SRAM se almacenan las salidas que, según la tabla de verdad del circuito a implementar, deben tomar las distintas combinaciones de entradas al circuito.



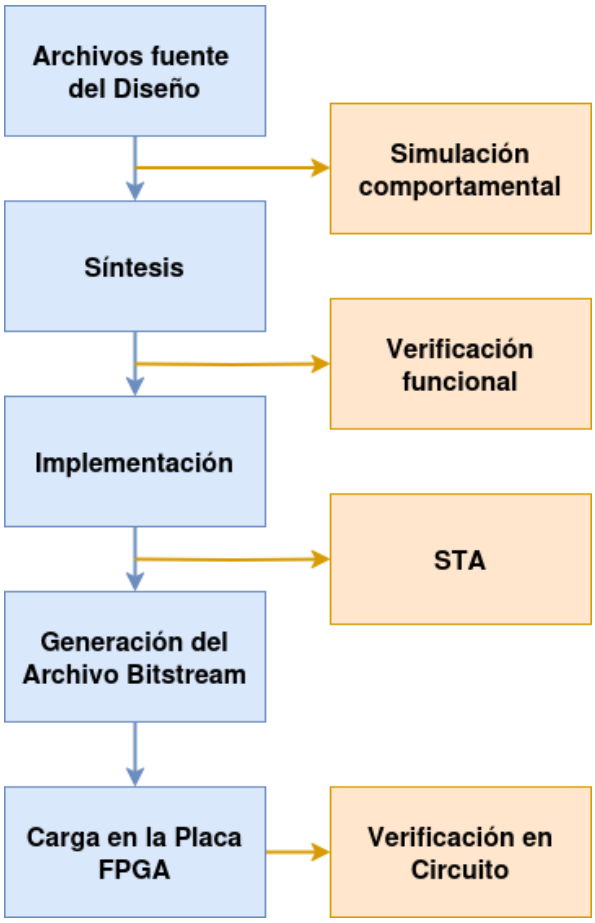
Esta estructura básica se repite un número muy grande de veces permitiendo que en las FPGAs puedan implementarse circuitos considerablemente grandes.



La configuración de estos bloques y sus interconexiones se define a través de un bitstream cargado en la memoria de configuración del dispositivo. Este bitstream se genera mediante el uso herramientas de síntesis y place-and-route (Vivado en nuestro caso) a partir de la descripción de un circuito. Para esta descripción es necesario acudir al uso de lenguajes de descripción de hardware (HDLs) como VHDL, Verilog o System Verilog.

Flujo de diseño

El proceso de diseño comienza con la creación de los archivos fuente del diseño mediante la descripción del comportamiento y la estructura utilizando un Lenguaje de Descripción de Hardware (HDL), en este caso system verilog. Es importante verificar que esta descripción lógica sea correcta antes de continuar, lo cual se realiza mediante la simulación comportamental. Utilizando un testbench, se simula el código HDL generando señales de entrada a la ALU y analizando si se producen las salidas esperadas, independientemente del hardware final. Una vez validado el comportamiento, se pasa a la síntesis del diseño. Aquí, una herramienta de software especializada convierte el código HDL a una representación a nivel de compuertas lógicas genéricas, conocida como *netlist*, la cual describe cómo se interconectan los elementos lógicos básicos. Opcionalmente, se puede realizar una verificación funcional (o simulación post-síntesis) sobre esta netlist para confirmar que la funcionalidad se ha preservado durante la traducción.



El siguiente paso crucial es la **implementación del diseño**, donde la netlist genérica se mapea a los recursos físicos específicos de la FPGA elegida. Este proceso incluye el *placement* (asignar cada elemento lógico a una ubicación física dentro del chip) y el *routing* (conectar estos elementos usando las pistas de interconexión disponibles). Durante o después de la implementación, se realiza el **análisis estático de tiempos (STA)**. Esta herramienta es vital porque calcula los retardos de las señales a través de las rutas implementadas y verifica si el diseño cumplirá con las restricciones de tiempo (como la frecuencia de reloj máxima) sin necesidad de simular con datos. Si el STA falla, indica que el diseño podría no funcionar a la velocidad deseada.

Superada la implementación y la verificación de tiempos, se procede a la **generación del bitstream**. Este es el archivo binario final que contiene toda la información de configuración necesaria para programar cada elemento configurable dentro de la FPGA y así materializar el diseño de la ALU. Finalmente, este archivo bitstream se **carga en la placa FPGA**, con la FPGA configurada, llega el momento de la **verificación en circuito**. Esta es la prueba real, donde se interactúa con la ALU implementada en el hardware (por ejemplo, usando interruptores y LEDs de la placa) para comprobar su correcto funcionamiento en condiciones reales. Esta última etapa confirma que todo el flujo, desde el código hasta el hardware, ha sido exitoso.

Recursos de System Verilog básicos necesarios para el laboratorio

En system verilog, es posible declarar cables del tipo logic que permite interconectar módulos, esto se hace mediante la palabra reservada *logic* y el nombre de la señal. Por ejemplo:

```
logic and_out_0, and_out_1, and_out_2, and_out_3;
```

Las compuertas lógicas dadas en el archivo basic_gates.sv se utilizan al instanciarlas, en system verilog esto se hace con la siguiente sintaxis:

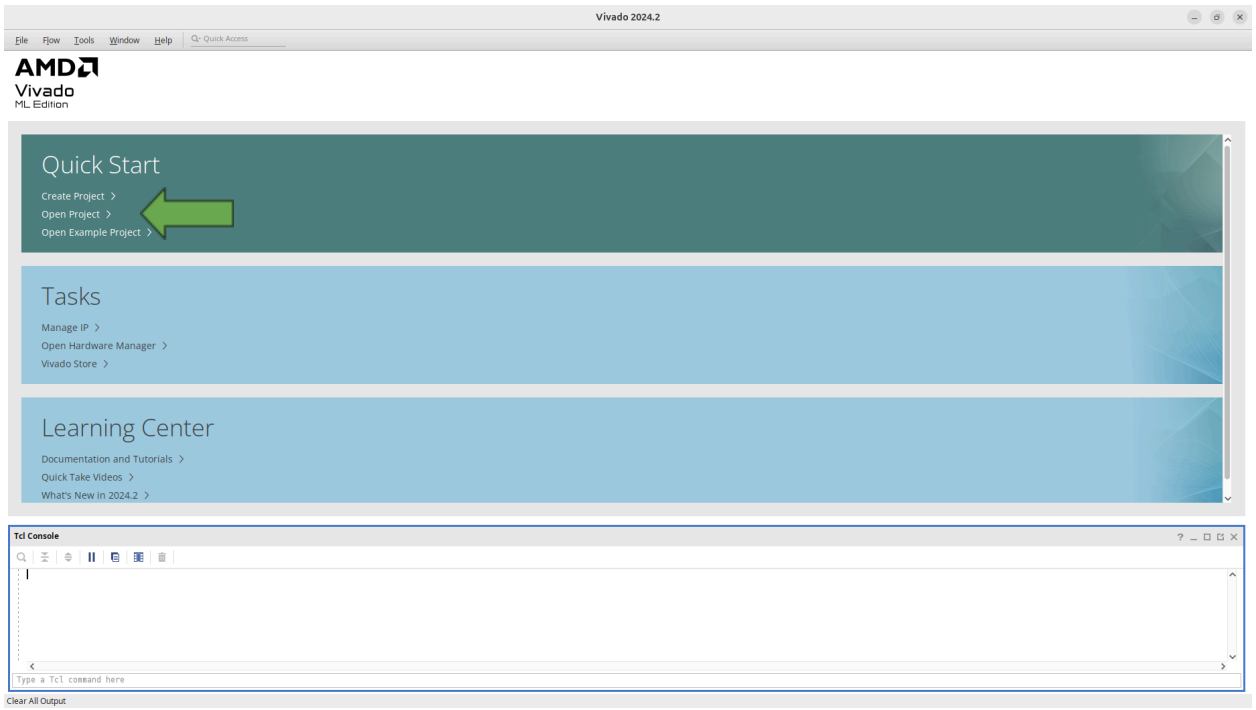
```
nombreDelMódulo nombreDeInstacia (
.nombreEntrada1Modulo(cableAConectar),
.nombreEntrada2Modulo(cableAConectar),
.nombreSalida1Modulo(cableAConectar),
.nombreSalida2Modulo(cableAConectar)
);
```

Por ejemplo, para instanciar una compuerta and:

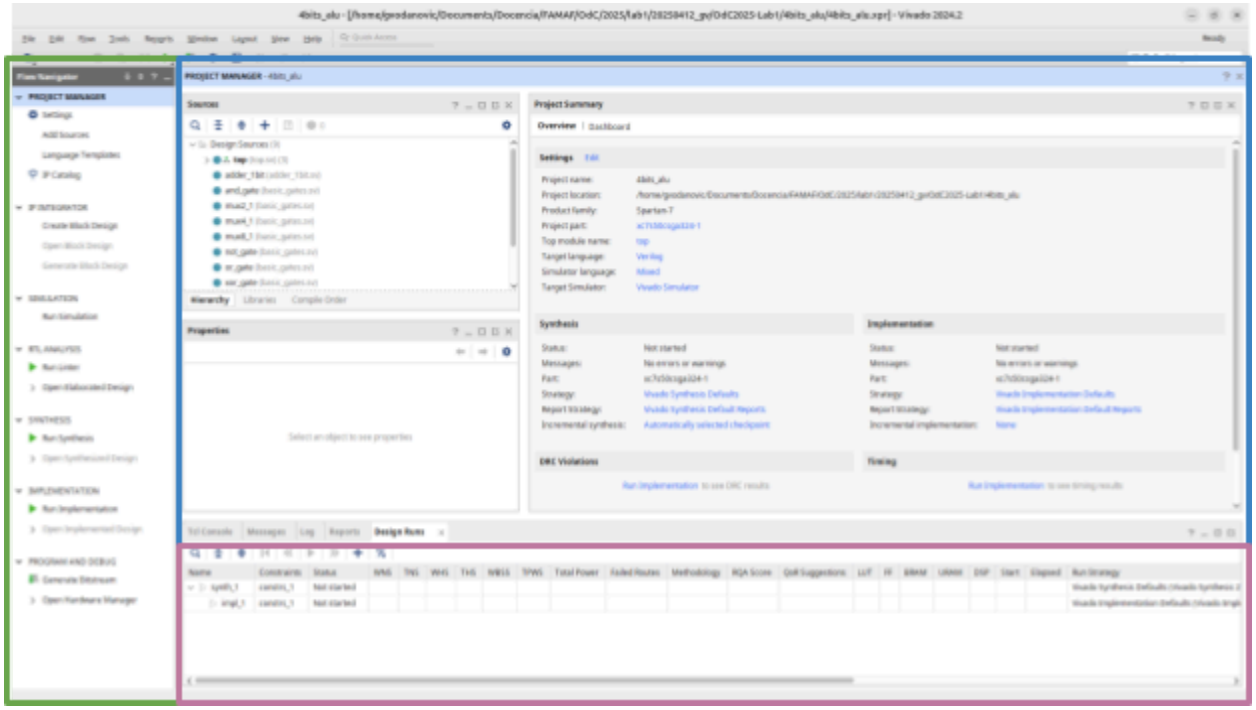
```
and_gate u_and_0 (.a(a_in_0), .b(b_in_0), .y(and_out_0));
```

Interfaz gráfica de vivado

Bajar el proyecto inicial del repositorio asignado, abrir vivado y abrir el proyecto. Se debe seleccionar el archivo llamado **4bits_alu.xpr**



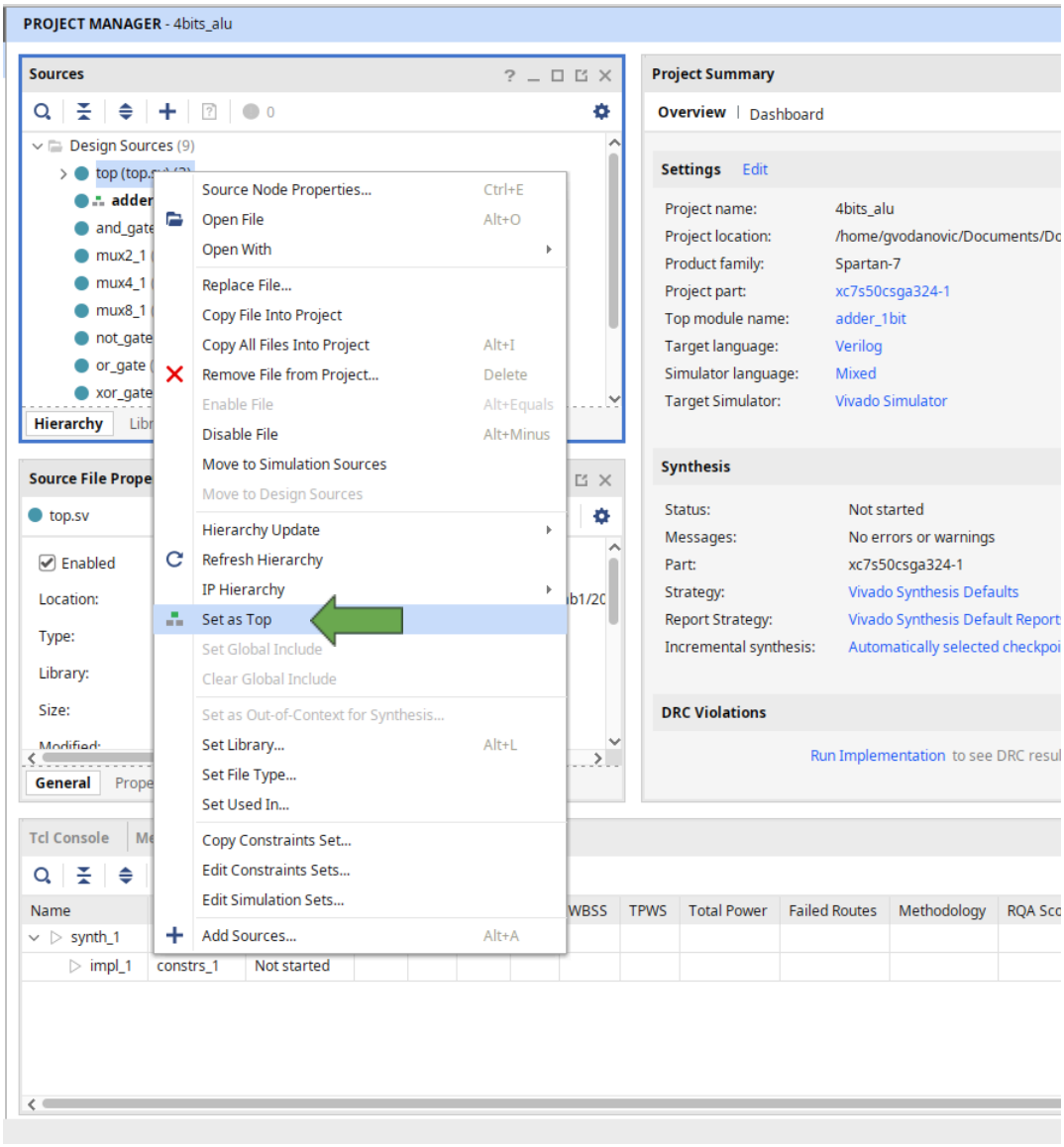
La interfaz de vivado puede separarse principalmente en 3 partes, en verde se marca el control de flujo donde se selecciona en qué etapa del flujo de diseño se encuentra, esto puede modificar el contenido del recuadro celeste o abrir pestañas nuevas. Finalmente en el recuadro rosa se muestran los mensajes, errores obtenidos como resultado.



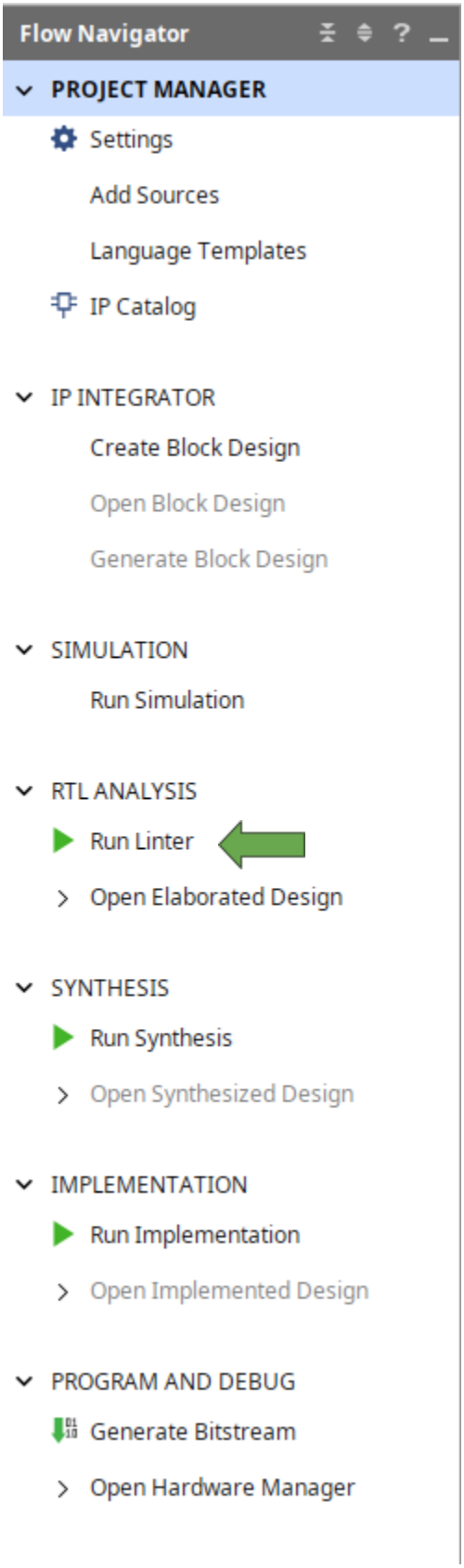
Las etapas de diseño son:

- RTL Analysis: La herramienta Linter verifica que no existan errores en la descripción del módulo, como errores de sintaxis, etc.
- Simulación: Mediante un archivo especial es posible describir señales de entrada al módulo para simular su comportamiento y verificar si se corresponde con el esperado. En nuestro caso siempre haremos simulaciones del tipo comportamental (Behavoral simulation).
- Synthesis: Esta herramienta genera un circuito a partir de una descripción, en nuestro caso a partir de lenguaje system verilog.
- Implementación: En esta etapa se utilizan distintas herramientas que ubican las compuertas lógicas de nuestro circuito en las disponibles en el dispositivo FPGA y las conecta entre ellas y con los puertos de entrada y salida.
- Program and debug: A partir de la implementación, esta herramienta genera un archivo llamado bitstream que contiene la configuración de la FPGA y programa el dispositivo.

Es muy importante considerar que las descripción de hardware es un proceso jerárquico. Por lo general, se inicia en los módulos más simples y se hace crecer. Es necesario para la herramienta indicarle cuál es el módulo de mayor jerarquía que estamos diseñando. En nuestro caso, cuando simulemos debemos elegir el módulo alu.sv pero cuando vamos a codificar la FPGA debemos seleccionar el módulo top.sv. Esto se hace con botón derecho sobre el módulo y la opción “Set as Top”

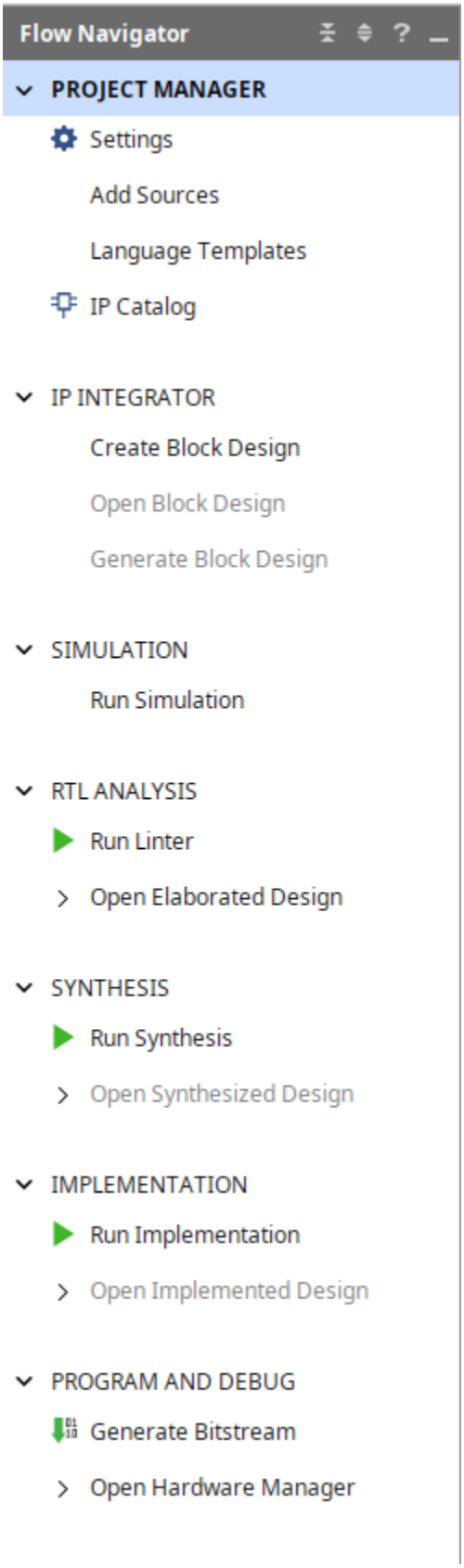


A continuación se describe cómo ejecutar cada etapa:

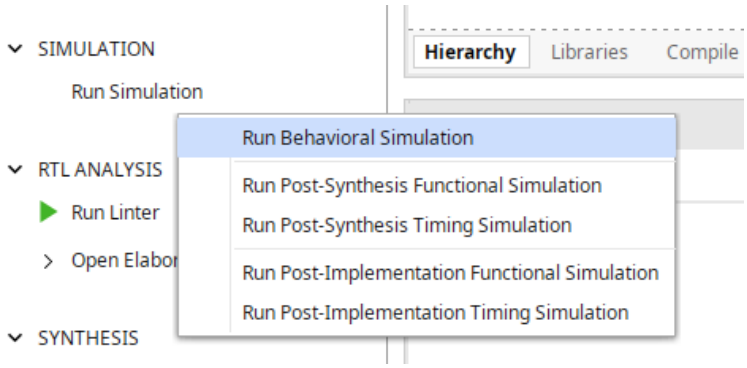


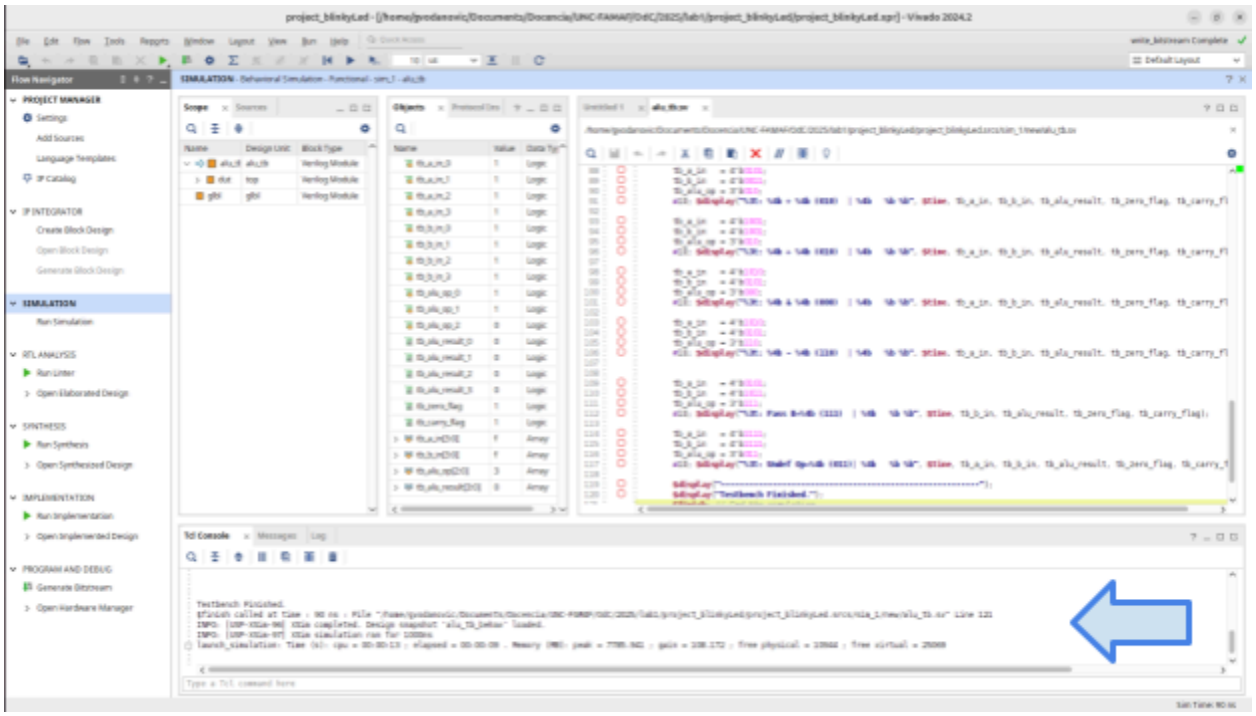
1) RTL Analysis: Elegir la opción “Run Linter”. En caso de encontrarse errores, se detallaran en la parte de abajo en la ventana de información, en la pestaña de Linter.

A continuación se describe cómo ejecutar cada etapa:



Simulación: Elegir la opción “Run Simulation” y luego “Run Behavioral Simulation”. Esto abrirá una serie de ventanas en la parte superior derecha de la interfaz pero lo que nos interesa está en la parte inferior donde se dicen los mensajes. Ahí buscar la tabla mostrada en la Guia del laboratorio.

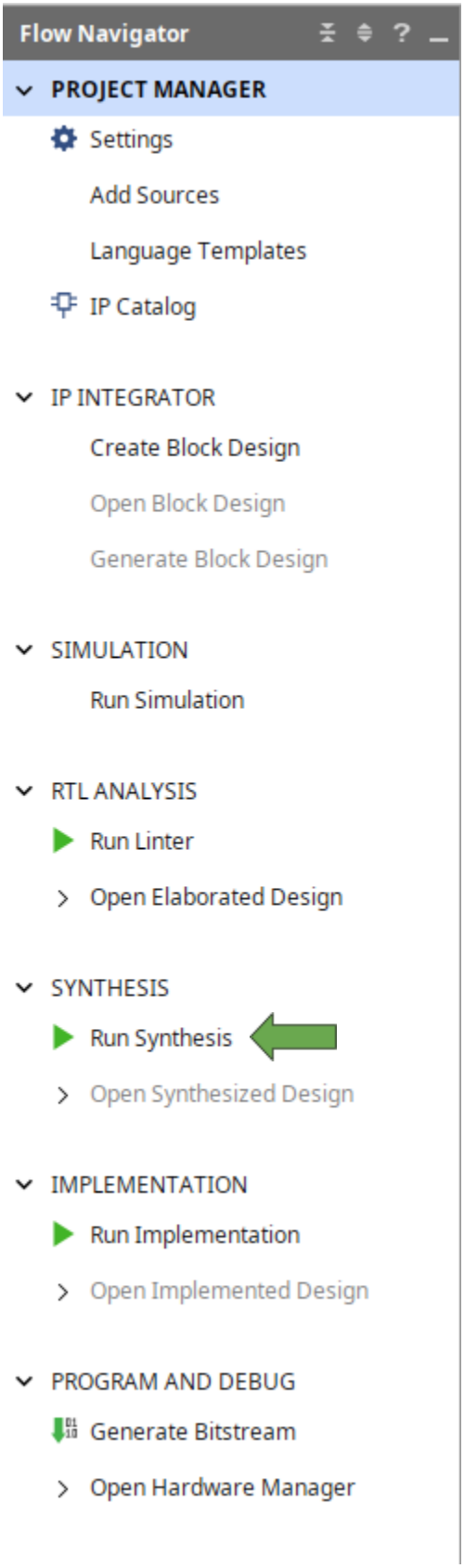




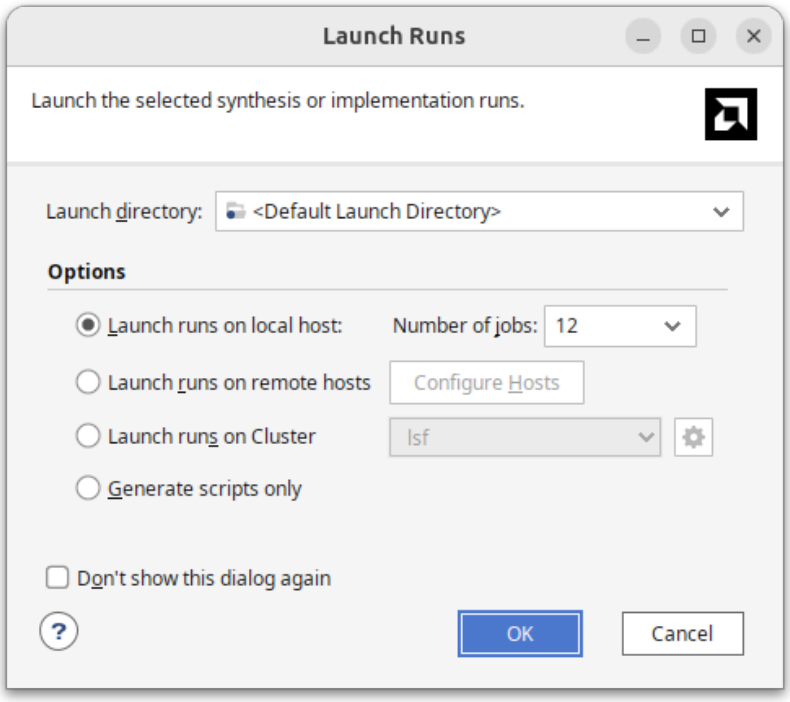
Tcl Console | Messages | Log

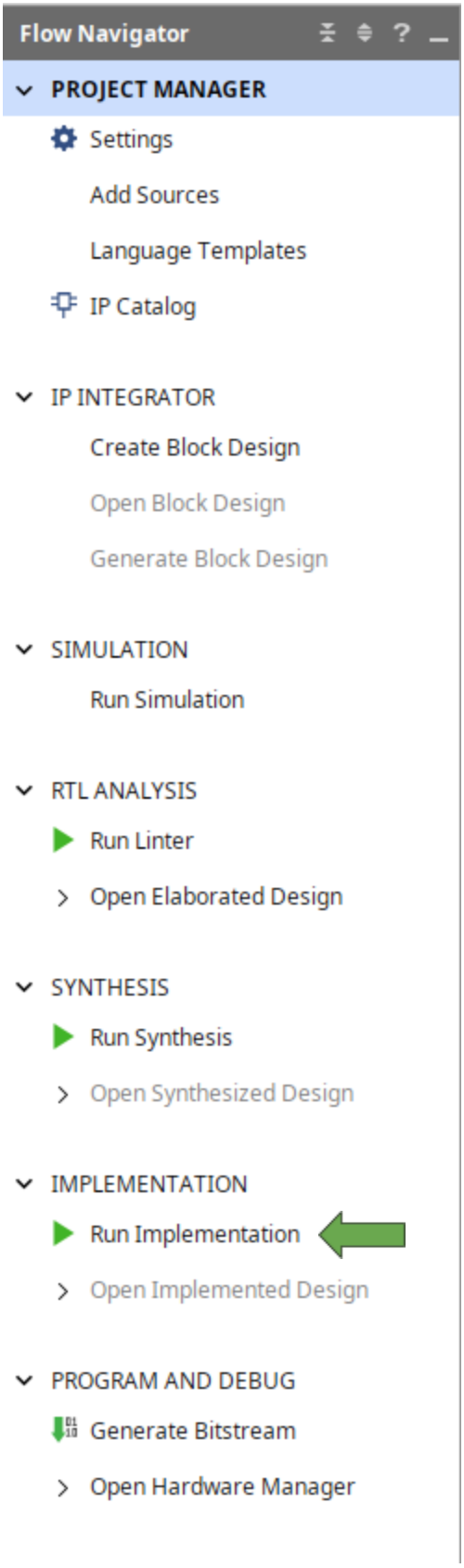
```
# }
# run 1000ns
              Starting ALU Testbench...
              Time   A      B      Op   | Result Z C
-----------------------------------------------------
              20000: 0101 & 0011 (000)  | 0001   0 0
              30000: 0101 | 0011 (001)  | 0111   0 0
              40000: 0101 + 0011 (010)  | 1000   0 0
              50000: 1001 + 1001 (010)  | 0010   0 1
              60000: 1010 & 0101 (000)  | 0000   1 0
              70000: 1010 - 0101 (110)  | 0101   0 0
              80000: Pass B=1011 (111)  | 1011   0 0
              90000: Undef Op=1111 (011)| xxxx   x x
-----------------------------------------------------
Testbench Finished.
$finish called at time : 90 ns : File "/home/gvodanovic/Documents/Docencia/UNC-FAMAF/OdC/2025/lab1/project_blinkyLed/project_blinkyLed.srcs/sim_1/new/alu_tb.sv" Line 121
INFO: [USF-XSim-96] XSim completed. Design snapshot 'alu_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:13 ; elapsed = 00:00:09 . Memory (MB): peak = 7785.941 ; gain = 108.172 ; free physical = 10944 ; free virtual = 25069
```

Type a Tcl command here

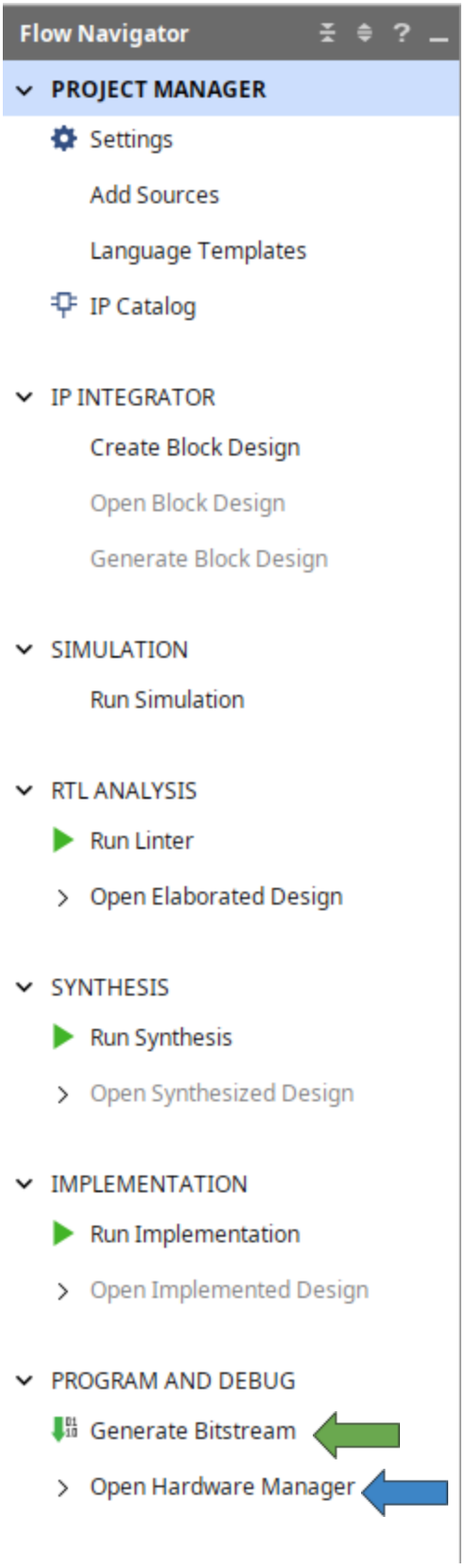


2) Sintesis: De debe elegir la opción de “Run Synthesis”, en la ventana nueva que se abre se debe elegir la cantidad de cores (jobs) a utilizar y luego el botón OK:



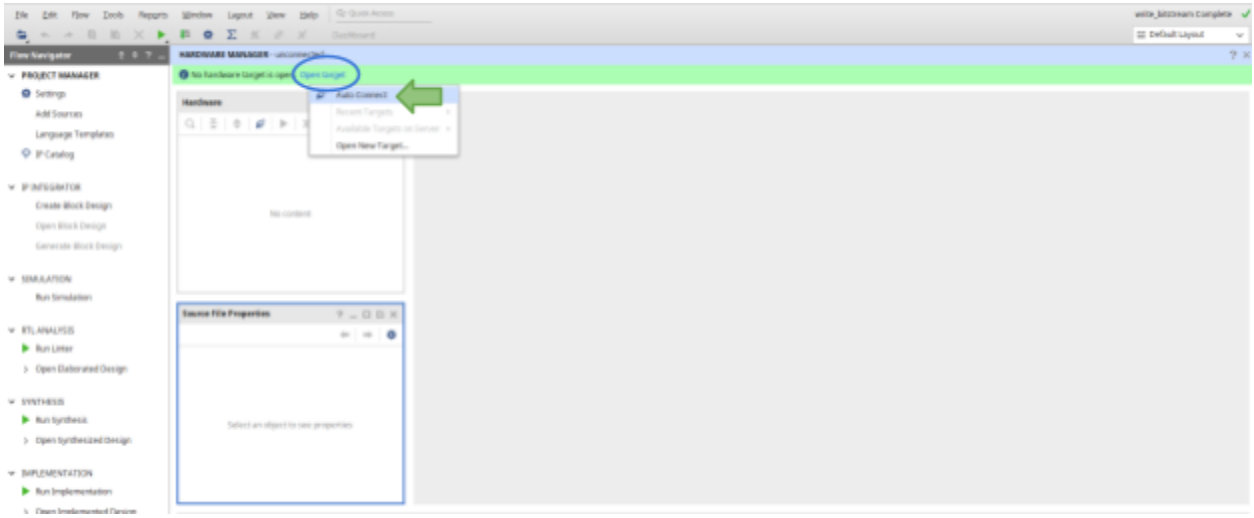


3) Implementación: En forma similar, se debe elegir la opción “Run Implementation”, luego elegir la cantidad de cores y darle al botón OK. Dependiendo de los recursos computacionales disponibles esta etapa puede demorar algunos segundos o minutos.

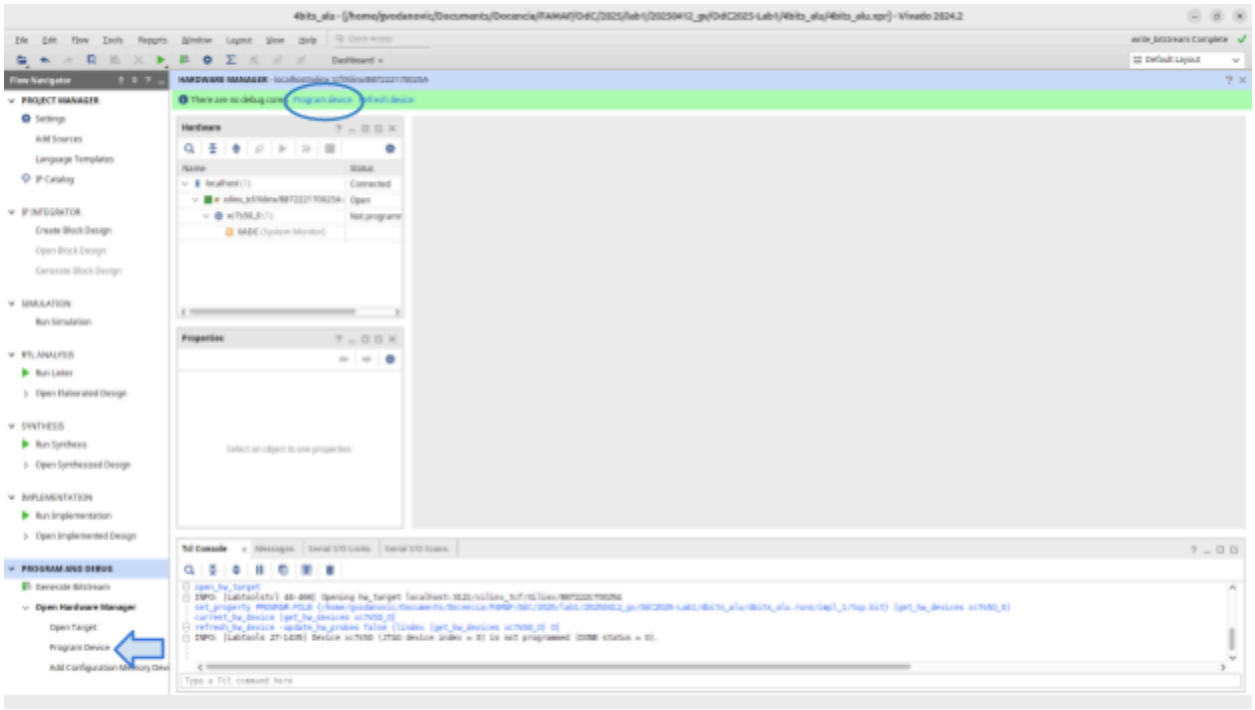


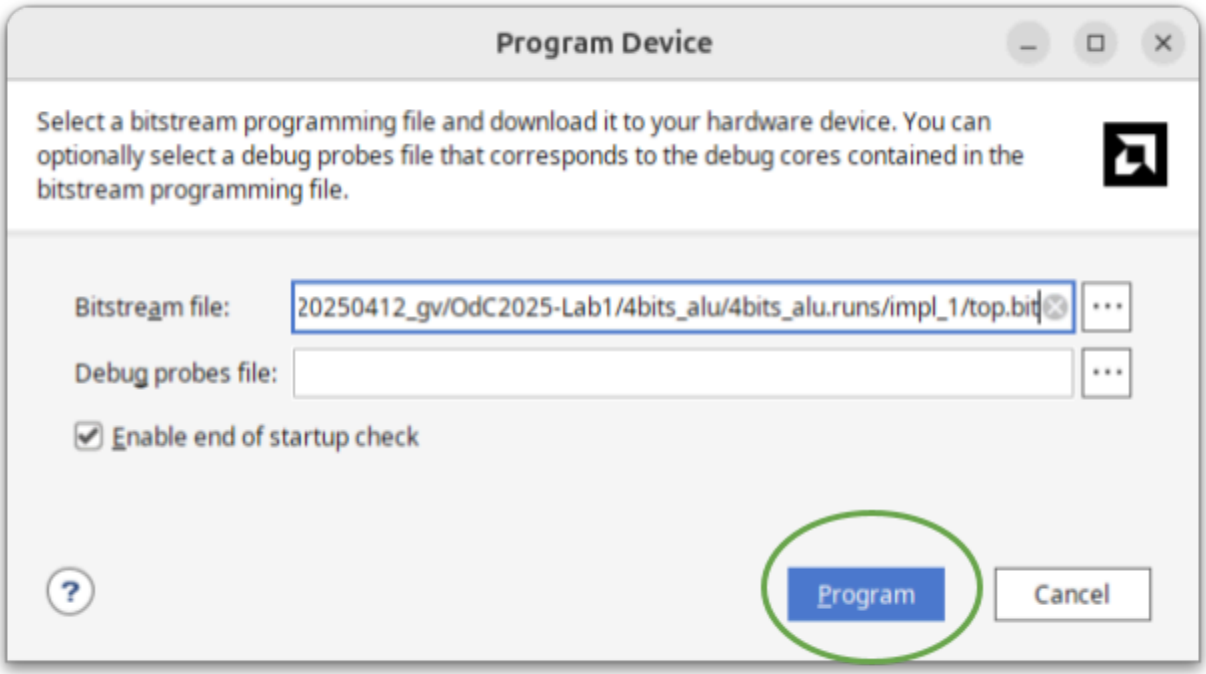
4) Configurar (programar) la FPGA: en primer lugar es necesario generar el archivo bitstream (flecha verde). Luego abrir el manejador de hardware con la opción "Open Hardware Manager" (flecha celeste).

Una vez en la ventana del hardware manager se debe abrir el “target” (circulo celeste) y elegir la opción “Auto Connect” (flecha verde):



Si la placa está conectada y encendida, se verá como se muestra en la siguiente imagen y se podrá elegir la opción “program device”:




Program Device
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.
Bitstream file:
20250412_gv/OdC2025-Lab1/4bits_alu/4bits_alu.runs/impl_1/top.bit
Debug probes file:
Enable end of startup check
Program
Cancel