

Practica 5 - Christian Lara

Introducción:

En el desarrollo de sistemas digitales, la integración de entidades es un proceso esencial que permite la construcción de diseños más complejos a partir de módulos más simples. Esta práctica tiene como propósito analizar las ventajas y desventajas que surgen al integrar múltiples entidades dentro de un mismo proyecto, utilizando el entorno de desarrollo Quartus de Altera.

La integración puede llevarse a cabo de diferentes maneras: diseñando cada bloque lógico por separado y uniéndolos posteriormente mediante un módulo principal, o evaluando el sistema de forma global a través de sus entradas y salidas. Ambos enfoques son válidos y funcionales; sin embargo, es importante comprender sus implicaciones prácticas y conceptuales.

Asimismo, durante esta práctica se identificarán y aplicarán nuevas palabras reservadas del lenguaje de descripción de hardware (VHDL o Verilog), las cuales son fundamentales para mejorar la estructura y funcionalidad de los diseños digitales.

1. Creamos el proyecto en donde se ubicará nuestros dos archivos VHDL y la configuración de nuestro proyecto en pin planner, así que debemos asignar correctamente la configuración al proyecto, integrando el modelo de la FPGA, el tipo de la simulación.

New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

Device family

Family: Cyclone III
Devices: All

Target device

☐ Auto device selected by the Fitter
☒ Specific device selected in 'Available devices' list
☐ Other: n/a

Show in 'Available devices' list

Package: Any
Pin count: Any
Speed grade: Any
Name filter: EP3C16F48
☒ Show advanced devices ☐ HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL	Global
EP3C16F484C6	1.2V	15408	347	516096	112	4	20
EP3C16F484C7	1.2V	15408	347	516096	112	4	20
EP3C16F484C8	1.2V	15408	347	516096	112	4	20
EP3C16F484I7	1.2V	15408	347	516096	112	4	20

Companion device

HardCopy:
☐ Limit DSP & RAM to HardCopy device resources

< Back Next > Finish Cancel Help

New Project Wizard

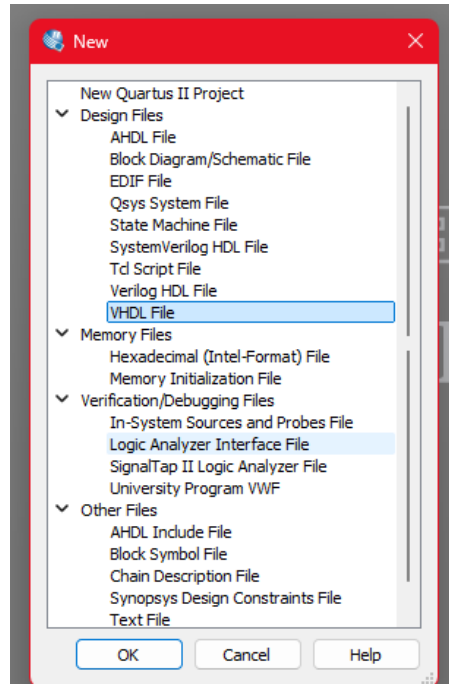
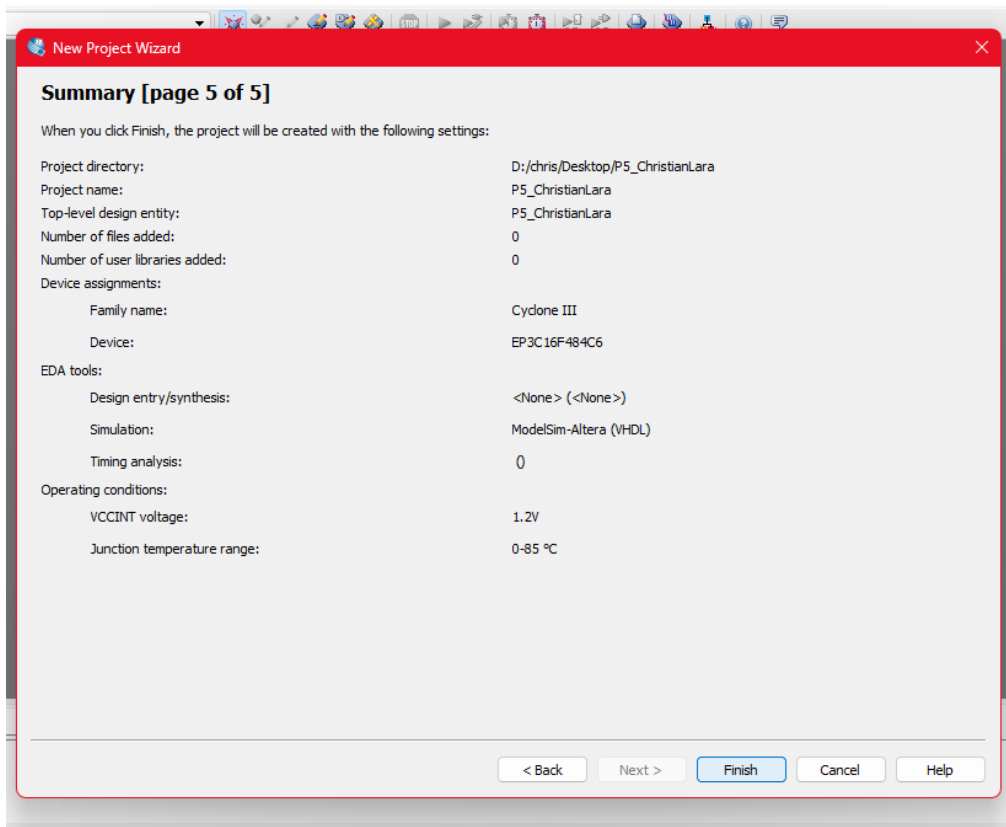
EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

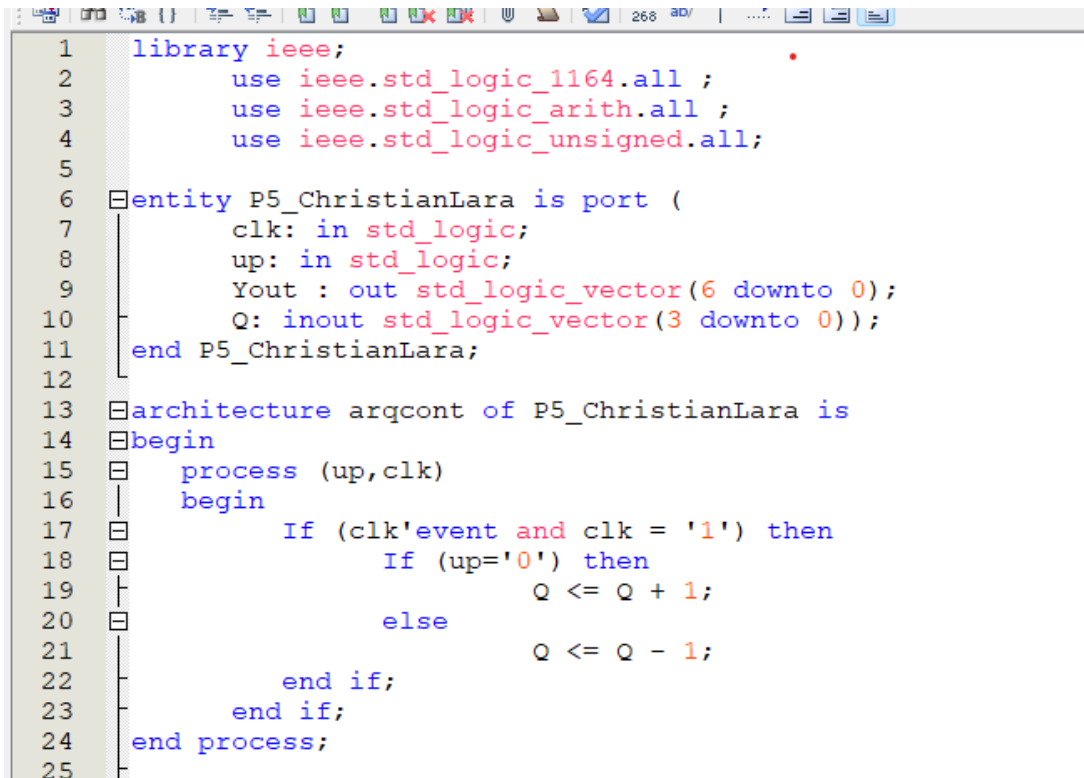
Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Formal Verification	<None>	<None>	
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

< Back Next > Finish Cancel Help



1. Esquema básico de integración de entidades

Como lo vimos en clase, la integración de entidades puede hacerse de dos formas: diseñando cada parte del sistema de manera individual y luego uniéndolas con un programa común, o analizando todo el sistema como un conjunto, enfocándose solo en sus entradas y salidas. Ambas opciones son válidas; sin embargo, es importante analizar las ventajas y desventajas de cada método para elegir la mejor solución.



```
1  library ieee;
2      use ieee.std_logic_1164.all ;
3      use ieee.std_logic_arith.all ;
4      use ieee.std_logic_unsigned.all;
5
6  entity P5_ChristianLara is port (
7      clk: in std_logic;
8      up: in std_logic;
9      Yout : out std_logic_vector(6 downto 0);
10     Q: inout std_logic_vector(3 downto 0));
11  end P5_ChristianLara;
12
13  architecture arqcont of P5_ChristianLara is
14  begin
15      process (up,clk)
16      begin
17          If (clk'event and clk = '1') then
18              If (up='0') then
19                  Q <= Q + 1;
20              else
21                  Q <= Q - 1;
22              end if;
23          end if;
24      end process;
25  end
```

Estas líneas importan librerías estándar de VHDL necesarias para manejar tipos de datos como std_logic y operaciones aritméticas entre vectores.

Define los puertos de entrada y salida:

- clk: entrada de reloj.
- up: define si el contador sube (up = 0) o baja (up = 1).
- Q: contador de 4 bits (inout para permitir lectura y escritura).
- Yout: salida de 7 bits que controla un display de 7 segmentos.

Este proceso se activa con el flanco de subida del reloj (clk).

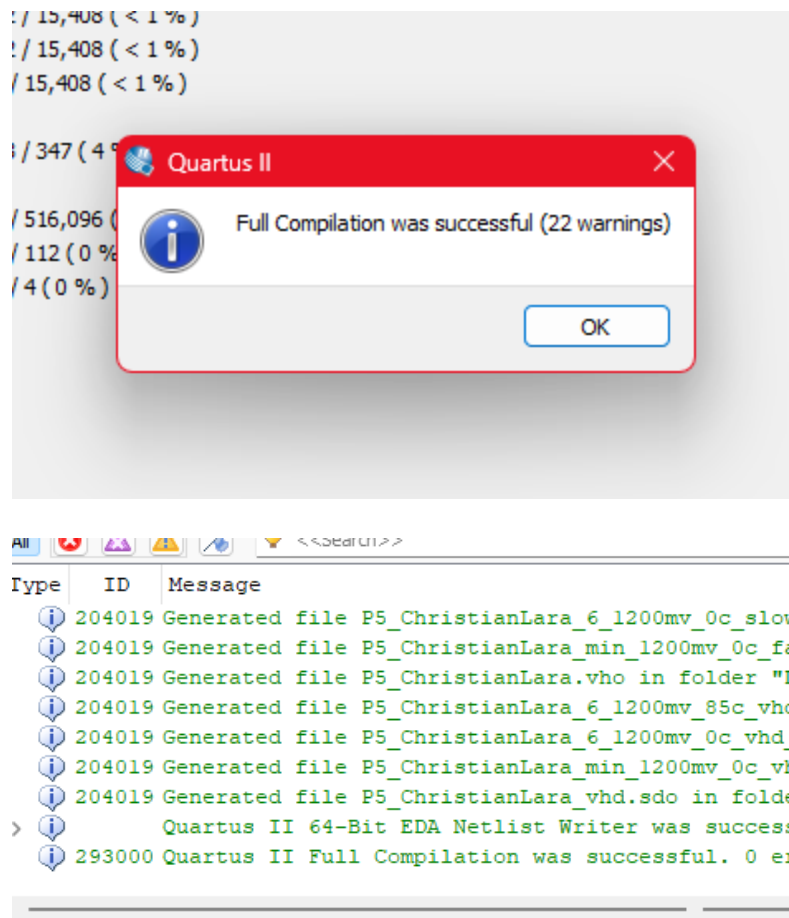
Según el valor de up, incrementa (up = 0) o decrementa (up = 1) el contador Q.

```
26 process (Q) begin
27     case Q is
28         -- dcba abcdefg
29         when "0000" => Yout <= "0000001";
30         when "0001" => Yout <= "1001111";
31         when "0010" => Yout <= "0010010";
32         when "0011" => Yout <= "0000110";
33         when "0100" => Yout <= "1001100";
34         when "0101" => Yout <= "0100100";
35         when "0110" => Yout <= "0100000";
36         when "0111" => Yout <= "0001111";
37         when "1000" => Yout <= "0000000";
38         when "1001" => Yout <= "0000100";
39         when others => Yout <= "1111111";
40     end case;
41 end process;
42 end arqcont;
```

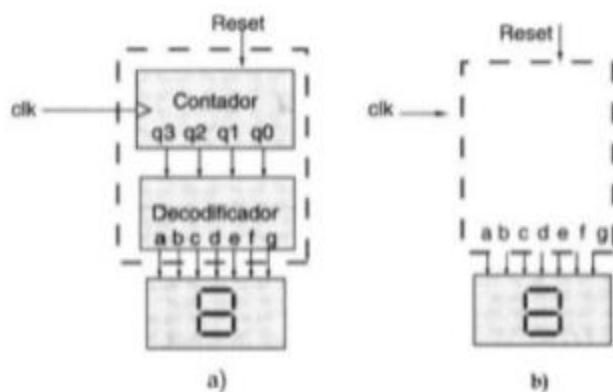
Este proceso traduce el valor binario de Q en el patrón correspondiente para un display de 7 segmentos.

Cada combinación de Q en binario activa los segmentos adecuados para mostrar los números del 0 al 9.

Este código implementa un **contador ascendente o descendente** controlado por la señal up, y **muestra el valor en un display de 7 segmentos** a través de la señal Yout. Cuando se presiona un botón o se activa up, el contador incrementa o decrementa su valor en cada pulso de reloj.



Finalmente compilamos el código que se encuentra en nuestro archivo de VHDL en el que verificamos que no tenga ningún error, ya sea de sintaxis, o de lógica.



En esta imagen se puede observar la interpretación física de nuestro código en donde en el inciso a) se puede mostrar el diseño de entidades individuales y en el inciso b) se observa el diseño mediante la relación de entradas y salidas.

Relación entradas y salidas.

El contador deberá llegar al 9 y reiniciar en 0 y si va en orden descendente al llegar a 0 debe iniciar nuevamente en 9; además Agregar una entrada llamada reset, si RESET = 1, las salidas Q toman el valor de 0; si no es igual a 1 continúa la cuenta.

```
1  library ieee;
2      use ieee.std_logic_1164.all ;
3      use ieee.std_logic_arith.all ;
4      use ieee.std_logic_unsigned.all;
5
6  entity P5_ChristianLara is port (
7      clk: in std_logic;
8      up: in std_logic;
9      Yout : out std_logic_vector(6 downto 0);
10     Q: inout std_logic_vector(3 downto 0));
11  end P5_ChristianLara;
12
13  architecture arqcont of P5_ChristianLara is
14  begin
15      process (up,clk)
16      begin
17          If (clk'event and clk = '1') then
18              If (up='0') then
19                  Q <= Q + 1;
20              else
21                  Q <= Q - 1;
22              end if;
```

Estas líneas importan librerías necesarias para usar tipos de datos estándar (como std_logic, operaciones aritméticas (+, -), y operaciones con vectores de bits sin signo).

Aquí defino el módulo llamado P5p2_ChristianLara con:

- **clk**: señal de reloj (cuando hay flanco de subida, se actualiza el contador).
- **up**: control de dirección (si es 0 cuenta hacia arriba, si es 1 cuenta hacia abajo).
- **Yout**: salida de 7 bits que representa el número en un display de 7 segmentos.

Defino un **registro interno Q** de 4 bits que almacenará el valor del contador.

Aquí haces que **cada vez que el reloj tenga un flanco de subida** (`clk'event and clk = '1'`), el contador:

- **Sume 1** si `up = 0`.

- **Reste 1** si up = 1.

```

23     end if;
24 end process;
25
26 process (Q) begin
27     case Q is
28         -- dcba abcdefg
29         when "0000" => Yout <= "0000001";
30         when "0001" => Yout <= "1001111";
31         when "0010" => Yout <= "0010010";
32         when "0011" => Yout <= "0000110";
33         when "0100" => Yout <= "1001100";
34         when "0101" => Yout <= "0100100";
35         when "0110" => Yout <= "0100000";
36         when "0111" => Yout <= "0001111";
37         when "1000" => Yout <= "0000000";
38         when "1001" => Yout <= "0000100";
39         when others => Yout <= "1111111";
40     end case;
41 end process;
42 end arqcont;

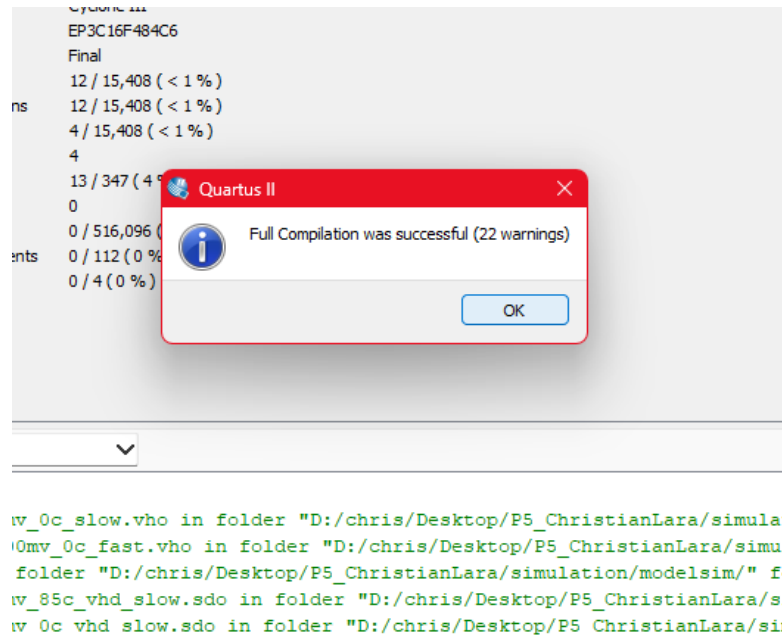
```

Este bloque traduce el valor de Q a un patrón para un display de 7 segmentos:

- Por ejemplo, si Q = "0000" (0 en binario), se activa el patrón "0000001", que prende los segmentos correctos para mostrar el número 0.

el código hace un contador de 0 a 9 o de 9 a 0 dependiendo de la señal up, y muestra ese número en un display de 7 segmentos.

- up = 0 → cuenta hacia arriba (0, 1, 2, 3, ..., 9).
- up = 1 → cuenta hacia abajo (9, 8, 7, ..., 0).
- En cada cambio del contador, se actualiza la salida Yout para mostrar el número correspondiente en un display.



Nuevamente compilamos nuestras líneas de código para que no existan errores y podamos ahora si a pasara a asignar los pines en nuestra FPGA con ayuda de pin planner, se debe tener en mente que los pines los verificamos en el manual de la FPGA y en base a eso hacemos una asignación rápida.

¿Para qué sirve asignar pines?

Asignar los pines sirve para que el hardware de la FPGA sepa por qué pin debe recibir o enviar cada señal. Sin esta asignación, el diseño no podría interactuar físicamente con el mundo exterior (botones, switches, displays, etc.).

¿Qué muestra el Pin Planner?

El Pin Planner muestra:

Nombre de la señal (las entradas y salidas definidas en VHDL).

Pin físico asignado a cada señal.

Dirección (entrada o salida).

Opciones de configuración como activar resistencias de pull-up o características eléctricas especiales.

También podemos ver una representación gráfica del chip, para visualizar dónde están ubicados los pines que estamos utilizando.

Los pines son los siguientes ahora, solo nos queda compilar y visualizar nuestra practica exitosa en nuestra FPGA, se adjuntan las imágenes y también el video que se encuentra dentro de la carpeta comprimida.

Pin Planner - D:\chris\Desktop\P5_ChristianLara\P5_ChristianLara - P5_ChristianLara

File Edit View Processing Tools Window Help

Report Report not available

Groups Report

Tasks

- Run Anal
- Early Pin
- Early
- Run
- Expo
- Change
- Show
- Show
- Show

Top View - Wire Bond
Cyclone III - EP3C16F484C6

I/O Bank...

I/O	Ass	Use	Avail
1	33	7	26
2	48	0	48
3	46	0	46
4	41	0	41
5	46	0	46
6	43	0	43
7	47	6	41
8	43	0	43

Named: H

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
in clk	Input	PIN_H2	1	B1_N1	PIN_G2	2.5 V (default)		8mA (default)		
io Q[3]	Bidir	PIN_H1	1	B1_N1	PIN_R7	2.5 V (default)		8mA (default)	2 (default)	
io Q[2]	Bidir	PIN_J3	1	B1_N1	PIN_R8	2.5 V (default)		8mA (default)	2 (default)	
io Q[1]	Bidir	PIN_J2	1	B1_N1	PIN_V4	2.5 V (default)		8mA (default)	2 (default)	
io Q[0]	Bidir	PIN_J1	1	B1_N1	PIN_T4	2.5 V (default)		8mA (default)	2 (default)	
in up	Input	PIN_J6	1	B1_N0	PIN_T9	2.5 V (default)		8mA (default)		
out Yout[6]	Output	PIN_H5	1	B1_N0	PIN_T5	2.5 V (default)		8mA (default)	2 (default)	
out Yout[5]	Output	PIN_E11	7	B7_N1	PIN_T8	2.5 V (default)		8mA (default)	2 (default)	
out Yout[4]	Output	PIN_F11	7	B7_N1	PIN_T7	2.5 V (default)		8mA (default)	2 (default)	
out Yout[3]	Output	PIN_H12	7	B7_N1	PIN_R6	2.5 V (default)		8mA (default)	2 (default)	

Filter: Pins: all

0% 00:00:00

Named: H

(ROW I/O)

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
in clk	Input	PIN_H2	1	B1_N1	PIN_G2	2.5 V (default)
io Q[3]	Bidir	PIN_H1	1	B1_N1	PIN_R7	2.5 V (default)
io Q[2]	Bidir	PIN_J3	1	B1_N1	PIN_R8	2.5 V (default)
io Q[1]	Bidir	PIN_J2	1	B1_N1	PIN_V4	2.5 V (default)
io Q[0]	Bidir	PIN_J1	1	B1_N1	PIN_T4	2.5 V (default)
in up	Input	PIN_J6	1	B1_N0	PIN_T9	2.5 V (default)
out Yout[6]	Output	PIN_H5	1	B1_N0	PIN_T5	2.5 V (default)
out Yout[5]	Output	PIN_E11	7	B7_N1	PIN_T8	2.5 V (default)
out Yout[4]	Output	PIN_F11	7	B7_N1	PIN_T7	2.5 V (default)
out Yout[3]	Output	PIN_H12	7	B7_N1	PIN_R6	2.5 V (default)

Conclusión

En esta práctica aprendí a diseñar y programar un contador que puede funcionar tanto en forma ascendente como descendente, además de integrar una señal de **reset** para reiniciar el conteo a cero en cualquier momento. Me llamó mucho la atención cómo, a través de una combinación de procesos y control de señales, podemos lograr un comportamiento dinámico en un circuito digital.

Entre los conceptos clave que reforcé están el manejo de **flancos de reloj**, el uso de señales de control como **up** y **reset**, y la importancia de asignar correctamente las salidas a un **display de 7 segmentos** usando un decodificador. También entendí mejor cómo funciona el **Pin Planner**, ya que sin la correcta asignación de pines sería imposible que nuestro diseño interactuara físicamente con los componentes de la tarjeta FPGA.

Algo que me gustó mucho fue ver cómo el contador no solo se mueve en un solo sentido, sino que responde de manera inteligente dependiendo de la entrada up, lo que hace el diseño mucho más versátil. Esta práctica me ayudó a comprender de una forma más real el paso de la teoría a la implementación, y a valorar la precisión que se necesita tanto en la programación como en el manejo del hardware.

