



Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica



Carrera: Ingeniería en Computación

Curso: Laboratorio de Organización y Arquitectura de
Computadoras
(Clave: 6867)

Práctica 5.

Profesor: Jorge Ferat Toscano

Alumno: García Sánchez Luis Manuel

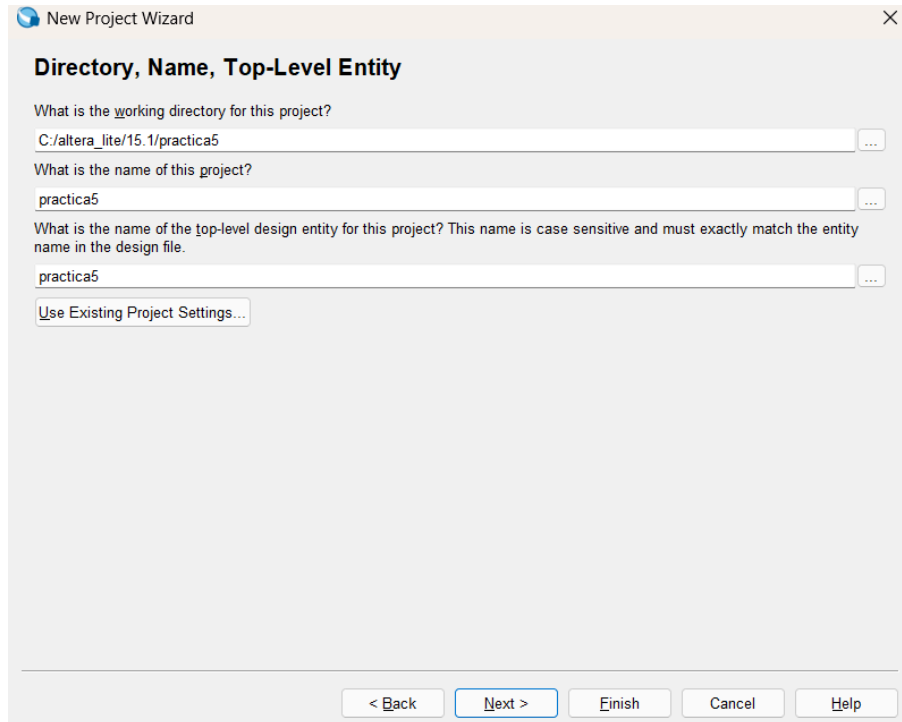
Fecha de entrega: 05/Mayo/25

Semestre: 2025-2

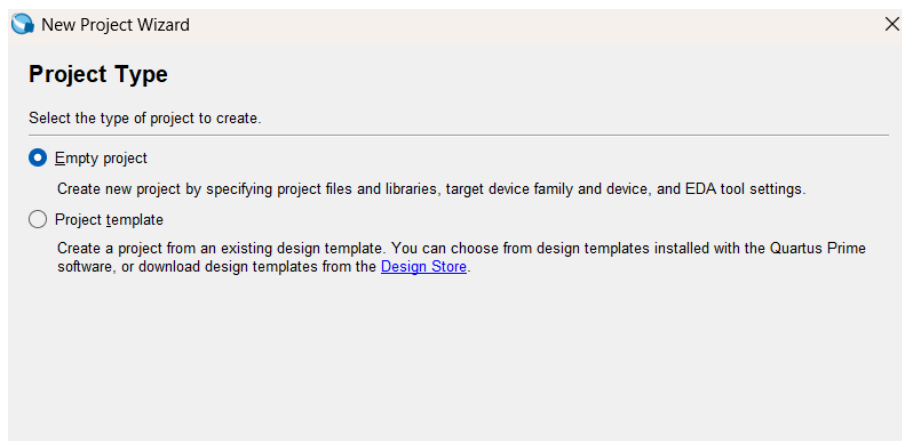
Objetivo

Generar de diferentes formas un multiplexor (MUX) por medio del lenguaje VHDL utilizando el software Quartus II .

1) Crear un proyecto con nombre practica 5, utilizando la misma forma de generarlo conforme a lo realizado en la anterior práctica 1



The screenshot shows the 'New Project Wizard' dialog box with the title 'New Project Wizard'. The main heading is 'Directory, Name, Top-Level Entity'. There are three text input fields with '...' buttons to the right. The first field is labeled 'What is the working directory for this project?' and contains 'C:/altera_lite/15.1/practica5'. The second field is labeled 'What is the name of this project?' and contains 'practica5'. The third field is labeled 'What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.' and contains 'practica5'. Below these fields is a button labeled 'Use Existing Project Settings...'. At the bottom of the dialog are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted with a blue border.



The screenshot shows the 'New Project Wizard' dialog box with the title 'New Project Wizard'. The main heading is 'Project Type'. Below the heading is the text 'Select the type of project to create.' followed by a horizontal line. There are two radio button options. The first option is 'Empty project', which is selected (indicated by a blue dot). Below it is the text 'Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.' The second option is 'Project template', which is not selected. Below it is the text 'Create a project from an existing design template. You can choose from design templates installed with the Quartus Prime software, or download design templates from the [Design Store](#).' The 'Design Store' is a blue hyperlink.

New Project Wizard

Family & Device Settings

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.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone IV E

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

Core Speed grade: Any

Name filter: EP4CE22F17C6

☒ Show advanced devices

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit el
EP4CE22F17C6	1.2V	22320	154	154	608256	132

< Back Next > Finish Cancel Help

2) El tipo de dispositivo a utilizar considere Auto

New Project Wizard

Summary

When you click Finish, the project will be created with the following settings:

Project directory: C:/altera_lite/15.1/practica5

Project name: practica5

Top-level design entity: practica5

Number of files added: 0

Number of user libraries added: 0

Device assignments:

Design template: n/a

Family name: Cyclone IV E

Device: EP4CE22F17C6

EDA tools:

Design entry/synthesis: <None> (<None>)

Simulation: ModelSim-Altera (VHDL)

Timing analysis: ()

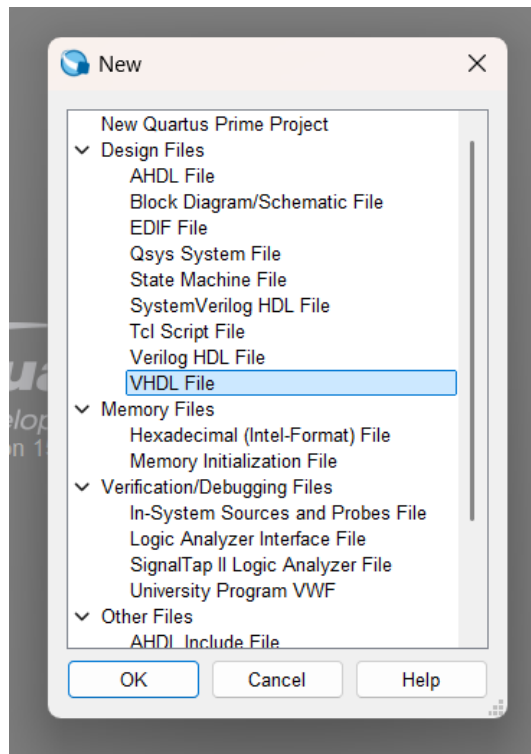
Operating conditions:

VCCINT voltage: 1.2V

Junction temperature range: 0-85 °C

< Back Next > **Finish** Cancel Help

3) Crear un archivo nuevo de diseño de tipo VHDL



4) Una vez que esté generado teclear en el editor el código

```
library ieee;
use ieee.std_logic_1164.all;

entity practica5 is
    port (
        pto0  : in std_logic;
        pto1  : in std_logic;
        sel   : in std_logic;
        ptops : out std_logic
    );
end practica5;

-- Opción mediante compuertas
--architecture multiplexor of practica5 is
--begin
--    ptops <= (pto0 and (not sel)) or (pto1 and sel);
--end multiplexor;
```

```

-- Opción mediante compuertas con señales intermedias
--architecture multiplexor of practica5 is
--    signal sig0, sig1: std_logic;
--begin
--    sig0 <= pto0 and (not sel);
--    sig1 <= pto1 and sel;
--    ptops <= sig0 or sig1;
--end multiplexor;

-- Opción mediante sentencia concurrente utilizando when
--architecture multiplexor of practica5 is
--begin
--    ptops <= pto0 when sel = '0' else pto1;
--end multiplexor;

-- Opción mediante proceso secuencial
--architecture multiplexor of practica5 is
--begin
--    proceso: process (sel, pto0, pto1)
--    begin
--        if sel = '0' then
--            ptops <= pto0;
--        else
--            ptops <= pto1;
--        end if;
--    end process;
--end multiplexor;

```

1. Opción mediante compuertas

Código VHDL de un multiplexor 2 a 1 implementado mediante compuertas lógicas. Se definen dos entradas (pto0 y pto1), una señal de selección (sel) y una salida (ptops). La lógica usada en la arquitectura permite que ptops tome el valor de pto0 cuando sel es 0, y de pto1 cuando sel es 1, usando una expresión lógica con AND, OR y NOT.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity practica5 is
5  port (
6      pto0 : in std_logic;
7      pto1 : in std_logic;
8      sel  : in std_logic;
9      ptops : out std_logic
10 );
11 end practica5;
12
13 -- Opción mediante compuertas
14 architecture multiplexor of practica5 is
15 begin
16     ptops <= (pto0 and (not sel)) or (pto1 and sel);
17 end multiplexor;
18

```

2. Opción mediante compuertas con señales intermedias

La imagen muestra una segunda versión del código VHDL de un multiplexor 2 a 1, ahora implementado con señales intermedias (sig0 y sig1). Estas señales almacenan el resultado parcial de las operaciones lógicas: sig0 toma el valor de pto0 cuando sel es 0, y sig1 toma el valor de pto1 cuando sel es 1. Finalmente, ptops se define como la suma lógica (or) de ambas señales. Esta versión es funcionalmente igual a la anterior, pero mejora la legibilidad y facilita la depuración.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity practica5 is
5  port (
6      pto0 : in std_logic;
7      pto1 : in std_logic;
8      sel  : in std_logic;
9      ptops : out std_logic
10 );
11 end practica5;
12
13 -- Opción mediante compuertas
14 --architecture multiplexor of practica5 is
15 --begin
16 --    ptops <= (pto0 and (not sel)) or (pto1 and sel);
17 --end multiplexor;
18
19 -- Opción mediante compuertas con señales intermedias
20 architecture multiplexor of practica5 is
21     signal sig0, sig1: std_logic;
22 begin
23     sig0 <= pto0 and (not sel);
24     sig1 <= pto1 and sel;
25     ptops <= sig0 or sig1;
26 end multiplexor;
```

3. Opción mediante sentencia concurrente utilizando when

La imagen muestra tres formas distintas de implementar un multiplexor 2 a 1 en VHDL dentro del mismo archivo. La primera opción (comentada) usa una expresión lógica directa con compuertas AND, OR y NOT. La segunda opción, también comentada, realiza la misma operación pero separando la lógica en señales intermedias (sig0 y sig1) para mejorar claridad. Finalmente, la tercera opción activa utiliza una sentencia concurrente con la estructura when-else, que asigna pto0 a la salida si sel es '0', y pto1 en caso contrario. Esta última forma es la más legible y compacta, y cumple con el mismo comportamiento funcional del multiplexor.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity practica5 is
5  port (
6      pto0 : in std_logic;
7      pto1 : in std_logic;
8      sel   : in std_logic;
9      ptops : out std_logic
10 );
11 end practica5;
12
13 -- Opción mediante compuertas
14 architecture multiplexor of practica5 is
15 --begin
16 --    ptops <= (pto0 and (not sel)) or (pto1 and sel);
17 --end multiplexor;
18
19 -- Opción mediante compuertas con señales intermedias
20 architecture multiplexor of practica5 is
21 --    signal sig0, sig1: std_logic;
22 --begin
23 --    sig0 <= pto0 and (not sel);
24 --    sig1 <= pto1 and sel;
25 --    ptops <= sig0 or sig1;
26 --end multiplexor;
27
28 -- Opción mediante sentencia concurrente utilizando when
29 architecture multiplexor of practica5 is
30 begin
31     ptops <= pto0 when sel = '0' else pto1;
32 end multiplexor;

```

4. Opción mediante proceso secuencial

La imagen presenta la cuarta opción de implementación de un multiplexor 2 a 1 en VHDL, utilizando un proceso secuencial. En este enfoque, se define un bloque process que depende de las señales sel, pto0 y pto1. Dentro del proceso se emplea una estructura if-else para asignar el valor de salida ptops: si sel es igual a '0', se asigna pto0; de lo contrario, se asigna pto1. Esta forma de descripción es útil cuando se busca una representación secuencial del comportamiento del circuito, facilitando su comprensión en diseños más complejos.

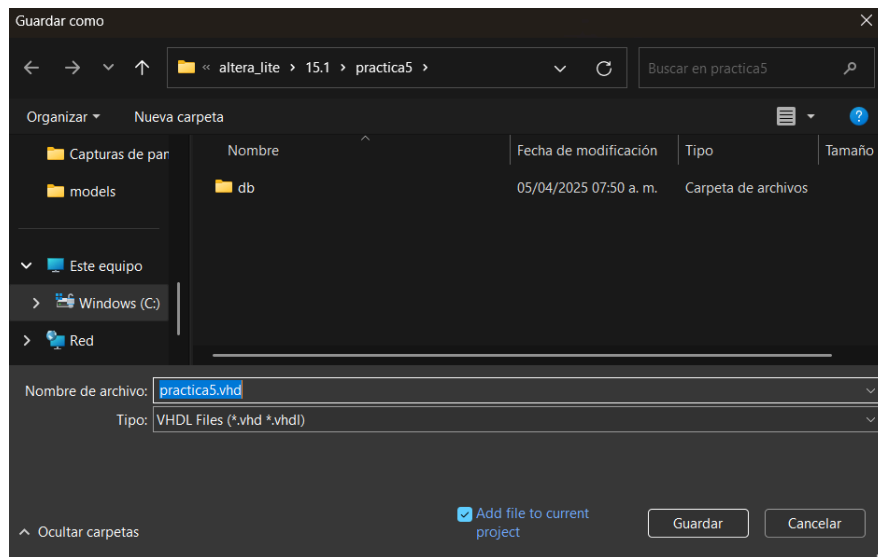
```

33
34 -- Opción mediante proceso secuencial
35 architecture multiplexor of practica5 is
36 begin
37     proceso: process (sel, pto0, pto1)
38     begin
39         if sel = '0' then
40             ptops <= pto0;
41         else
42             ptops <= pto1;
43         end if;
44     end process;
45 end multiplexor;
46

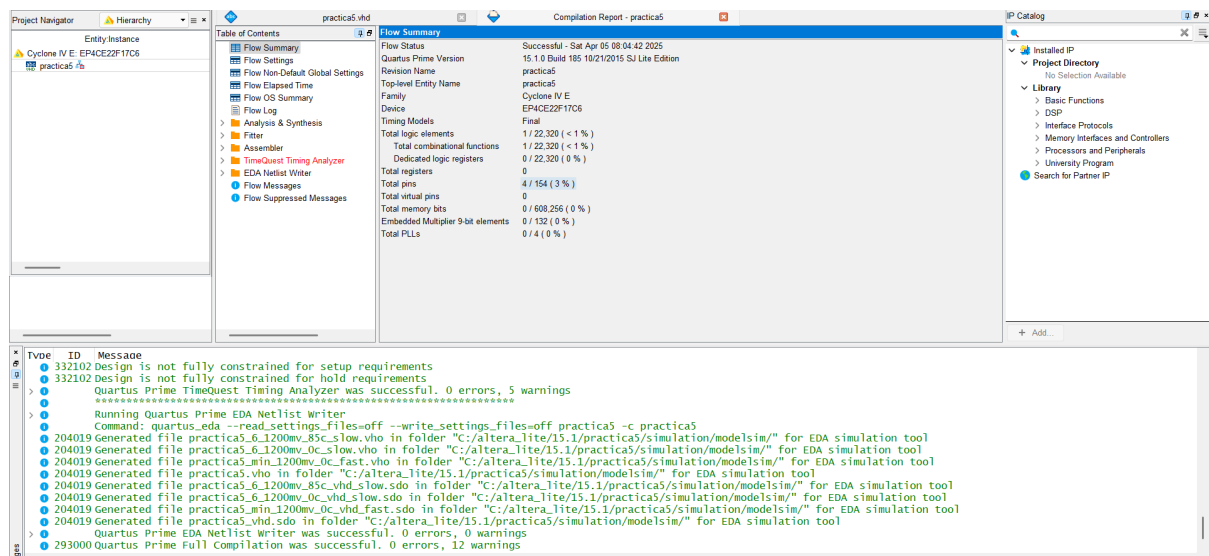
```

5) Compilar el programa Revisar que la compilación fue de manera exitosa

Muestra el proceso de compilación del archivo practica5.vhd en el entorno de desarrollo Quartus Prime. En la parte superior se observa la ventana donde se guarda el archivo con extensión .vhd, indicando que se está trabajando en un diseño en VHDL.



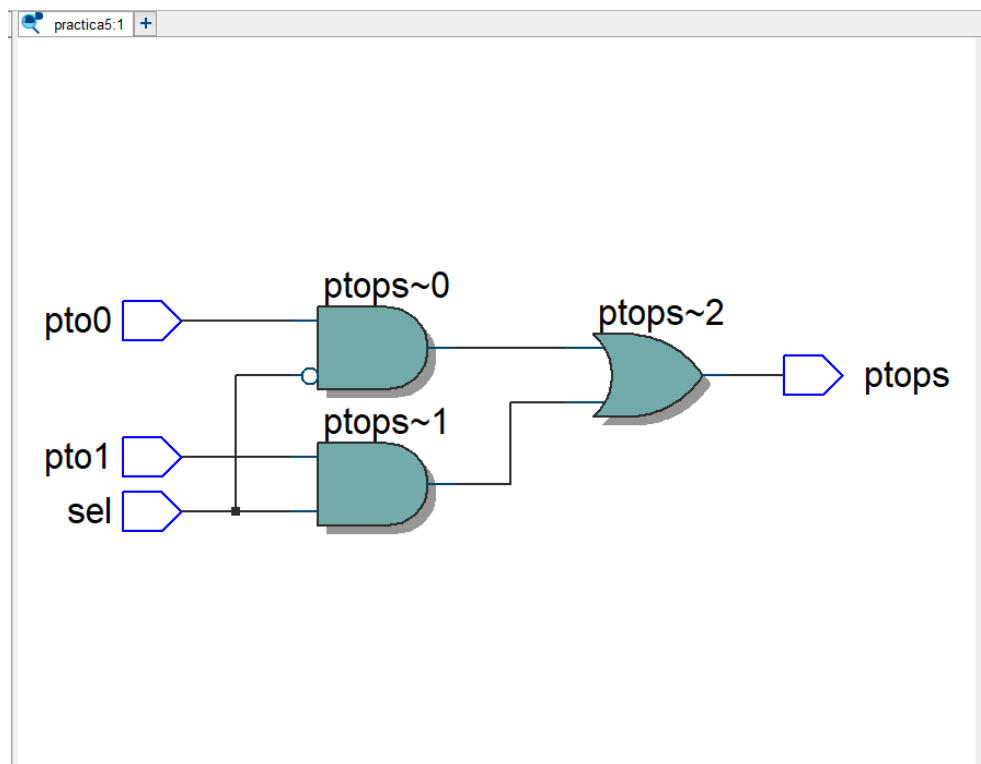
Se presenta el reporte de compilación, el cual confirma que el proceso fue exitoso, sin errores y con algunas advertencias mínimas. Esto indica que el código del multiplexor fue correctamente interpretado y sintetizado por el software, lo que permite continuar con la simulación y pruebas del diseño.



6) Procedemos a revisar el circuito sintetizado (RTL Viewer) Se observará en la pantalla el diagrama correspondiente al circuito

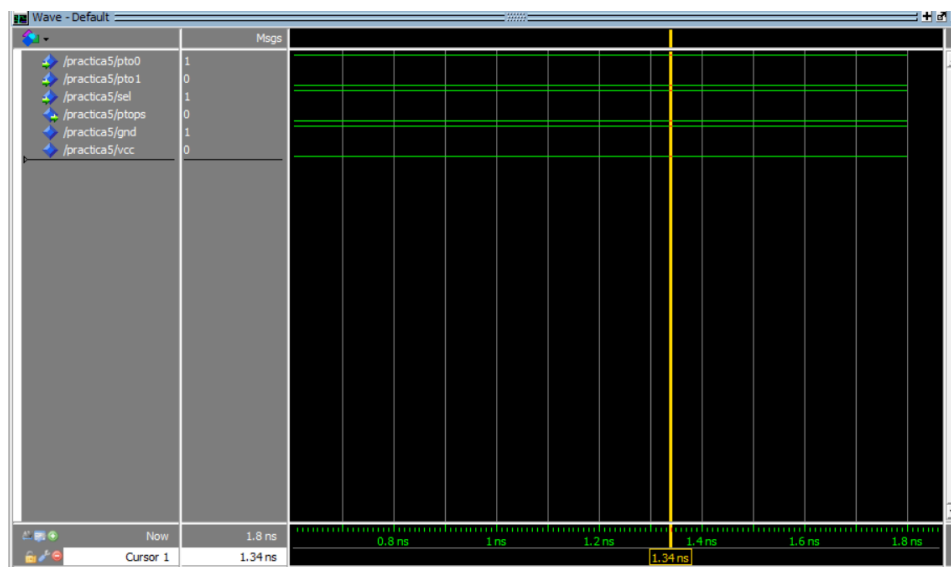
1. Opción mediante compuertas

La imagen muestra el circuito lógico sintetizado del multiplexor 2 a 1 a partir de compuertas lógicas básicas. Se pueden identificar dos compuertas AND y una OR que implementan la lógica combinacional del multiplexor. La primera AND combina pto0 con el inverso de sel, mientras que la segunda AND combina pto1 con sel. Ambas salidas se conectan a una compuerta OR, cuya salida final es ptops. Esta implementación refleja la ecuación $ptops \leq (pto0 \text{ AND NOT sel}) \text{ OR } (pto1 \text{ AND sel})$, y confirma visualmente que la síntesis del diseño en VHDL se realizó correctamente.



Simulador 1

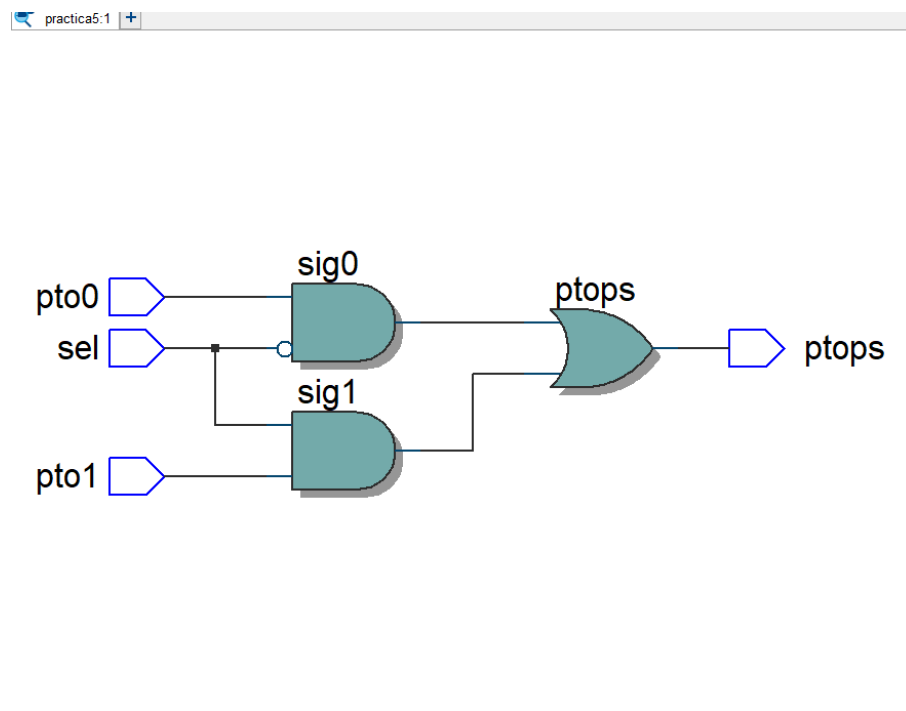
Se observa el comportamiento del multiplexor 2 a 1 con un nuevo conjunto de entradas. En este caso, pto0 está en 0, pto1 está en 1, y la señal de selección sel está en 0. De acuerdo con la lógica del multiplexor, cuando sel es 0, la salida ptops debe tomar el valor de pto0, por lo tanto, el resultado correcto es que ptops sea 0, tal como se muestra en la gráfica. Esta simulación confirma que el diseño responde adecuadamente a los cambios de entrada y cumple con la lógica esperada para este tipo de circuito.



Simulador 2

2. Opción mediante compuertas con señales intermedias

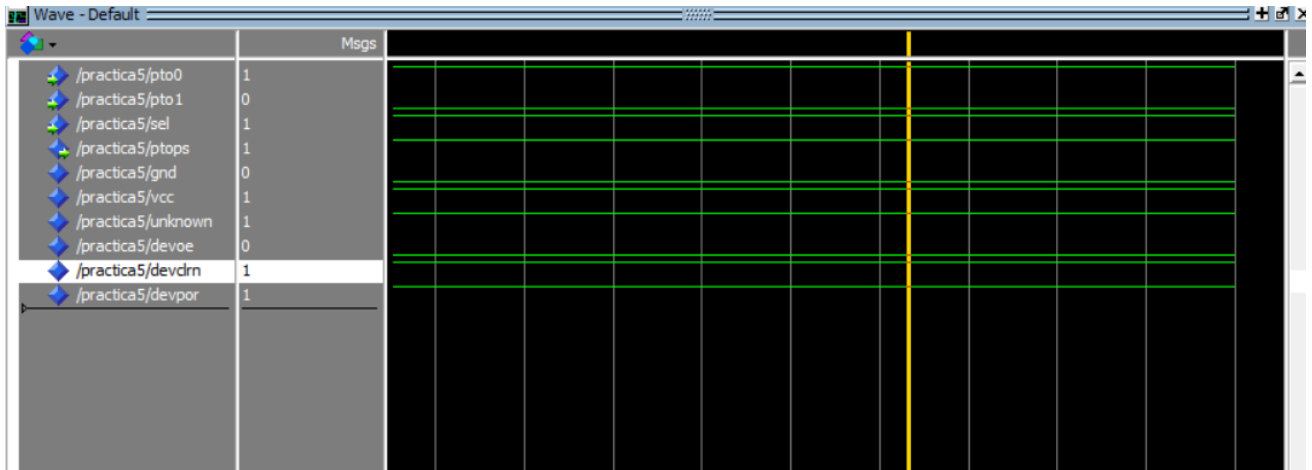
La imagen muestra el circuito lógico sintetizado de un multiplexor 2 a 1 utilizando **señales intermedias**, en este caso llamadas sig0 y sig1. Se observan dos compuertas AND: la primera activa sig0 cuando pto0 está en 1 y sel en 0 (gracias al inversor), mientras que la segunda activa sig1 cuando pto1 y sel están ambos en 1. Las salidas de ambas AND se combinan mediante una compuerta OR, generando el valor final de ptops. Esta estructura corresponde exactamente a la arquitectura vista en VHDL que utiliza señales intermedias, dividiendo el cálculo en partes más legibles sin alterar la lógica funcional del multiplexor.



Simulador 1

La imagen muestra una simulación en ModelSim donde se observa el comportamiento del multiplexor 2 a 1 en tiempo real. En el instante captado, las señales pto0, pto1 y sel están en alto (1), por lo tanto, la salida ptops también es 1, lo cual es correcto según la lógica del multiplexor, ya que cuando sel = 1, la salida debe tomar el valor de pto1. Además, se muestran

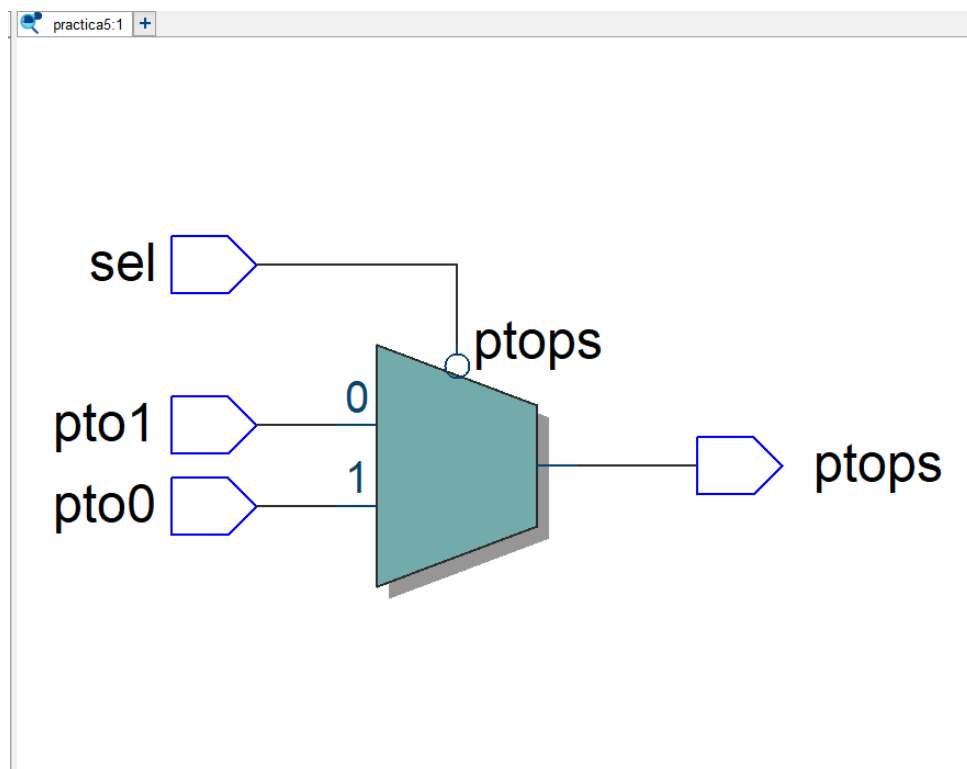
otras señales relacionadas con la simulación (vcc, gnd, devoe, devclrn, etc.), que forman parte del entorno de ejecución del dispositivo, pero no afectan directamente el comportamiento lógico del multiplexor. La línea amarilla indica el punto exacto del tiempo en el que se analizan los valores de las señales, y se confirma que el circuito responde adecuadamente según las entradas establecidas.



Simulador 2

3. Opción mediante sentencia concurrente utilizando when

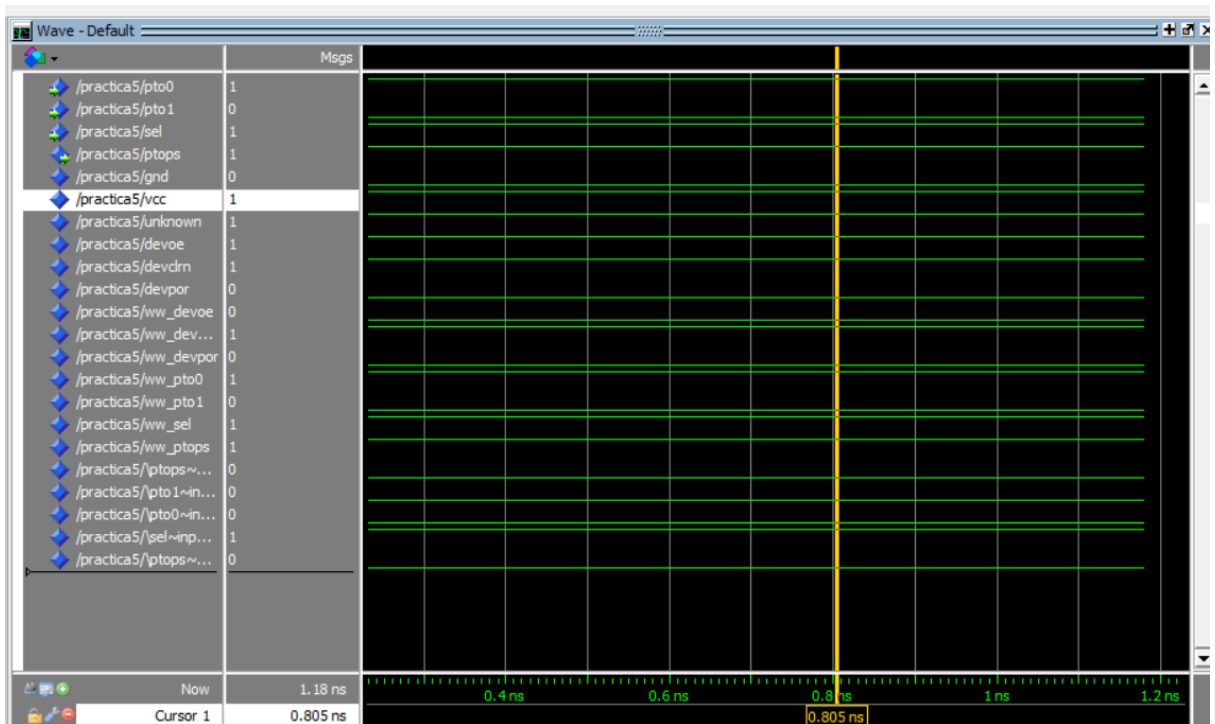
La imagen muestra el símbolo gráfico de un multiplexor 2 a 1 utilizado en el diseño del circuito. El componente tiene tres entradas: pto0, pto1 y sel, y una salida ptops. La señal sel determina cuál de las dos entradas se conecta a la salida: si sel es 0, se selecciona pto1; si es 1, se selecciona pto0. Este símbolo resume de manera clara y directa el comportamiento lógico del multiplexor, sin mostrar los detalles internos de compuertas, lo cual es útil para representar el diseño de manera más abstracta y legible en esquemas de nivel superior.



Simulador 1

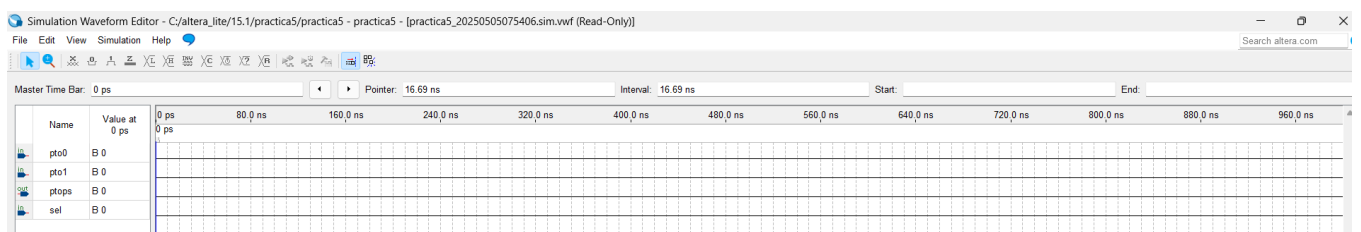
La imagen muestra una simulación temporal del multiplexor 2 a 1, capturada en el instante 0.805 ns. En este punto, pto0 y pto1 están en alto (1), al igual que la señal de selección sel. Según la lógica del multiplexor, cuando sel = 1, la salida ptops debe tomar el valor de pto0. La gráfica incluye también otras

señales internas y auxiliares del entorno de simulación, lo que permite un análisis más detallado del flujo de datos.



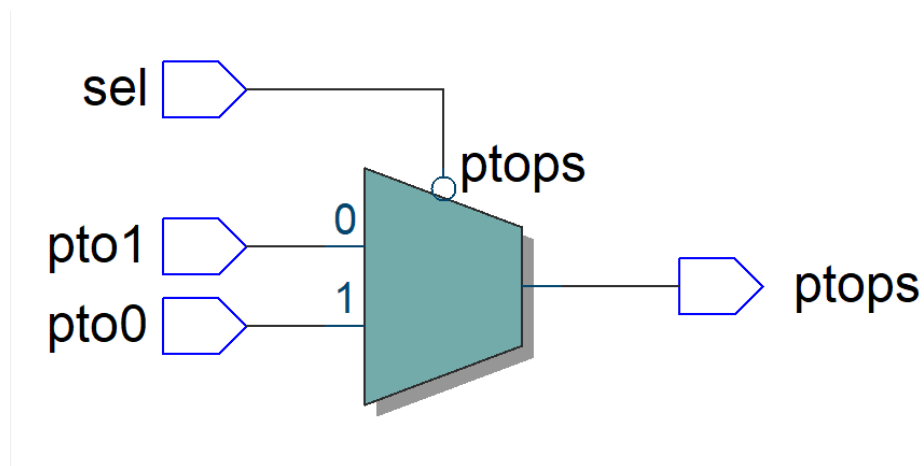
Simulador 2

Se observa el estado inicial de las señales pto0, pto1, sel y ptops, todas en bajo (B 0)



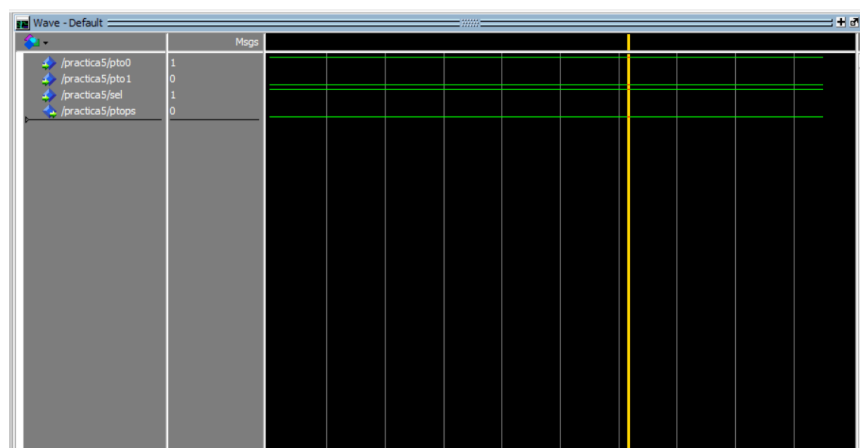
4. Opción mediante proceso secuencial

La imagen representa un multiplexor 2 a 1, un circuito digital que permite seleccionar entre dos señales de entrada (pto0 y pto1) para enviarlas a una única salida (ptops). La elección de la entrada que se dirige a la salida depende del valor de una señal de control llamada sel. Cuando esta señal vale 0, la salida toma el valor de pto1, y cuando vale 1, toma el valor de pto0. Este tipo de componente es esencial en sistemas digitales para gestionar la transmisión de datos, ya que permite redirigir señales de manera controlada mediante una entrada selectora.



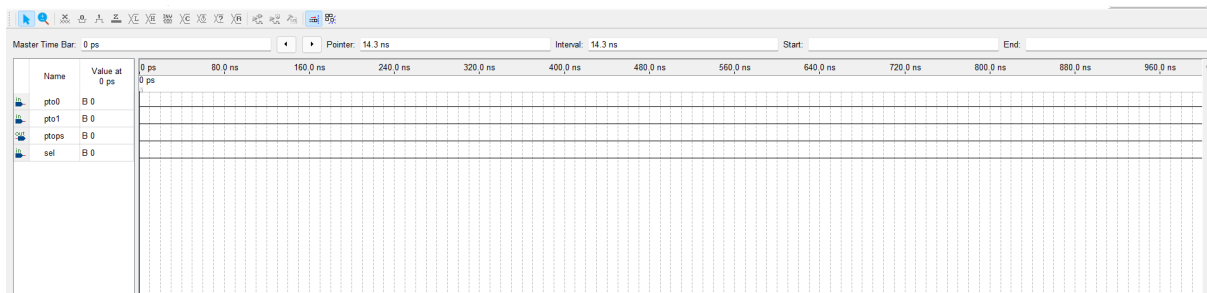
Simulador 1

Esta imagen muestra el análisis temporal de un multiplexor 2 a 1, donde se evalúa el comportamiento lógico de las señales pto0, pto1, sel y la salida ptops. En este momento de la simulación, la entrada pto0 está en alto (1), pto1 en bajo (0) y la señal de selección sel también está en alto (1). Según la lógica del multiplexor, esto implica que la salida ptops debe seguir el valor de pto0, es decir, debe colocarse en 1.



Simulador 2

Esta imagen muestra el inicio de una simulación temporal de un circuito digital en el que se analiza nuevamente el comportamiento de un multiplexor 2 a 1. Las señales observadas son pto0, pto1, sel y la salida ptops. En el instante inicial (0 ps), todas las señales se encuentran en bajo (B 0), lo que significa que ambas entradas del multiplexor, la señal de selección y la salida están en 0.



Conclusiones

Esta práctica fue una gran oportunidad para entender y aplicar distintas formas de crear un multiplexor (MUX) usando VHDL dentro del entorno de Quartus II. A lo largo del desarrollo, probamos varias maneras de describir el mismo circuito: desde compuertas lógicas básicas, pasando por el uso de señales intermedias, hasta estructuras más elegantes como when-else y bloques secuenciales con process. Esto no solo nos ayudó a ver que hay muchas formas de llegar al mismo resultado, sino que también nos permitió comparar cuál es más clara, más organizada o más útil según el contexto del diseño.

Cada versión del multiplexor fue compilada correctamente, lo cual validó nuestras descripciones y nos permitió avanzar con la simulación. Las gráficas obtenidas en ModelSim y en el Simulation Waveform Editor nos dieron evidencia clara de que los

MUX funcionaban tal como lo esperábamos: seleccionando entre pto0 y pto1 según el valor de sel, y reflejándose directamente en la salida ptops. Además, las representaciones gráficas del circuito sintetizado ayudaron a visualizar cómo ese código se traduce en compuertas reales, reforzando nuestro entendimiento de lo que realmente ocurre dentro del chip.

En resumen, esta práctica nos dio una visión completa: desde cómo se escribe el código en VHDL, hasta cómo se traduce y se comporta dentro del hardware. Fue una experiencia muy valiosa para afianzar conocimientos de diseño digital, y sobre todo, para darnos cuenta de que programar hardware no es solo escribir líneas de código, sino entender lo que hay detrás de cada instrucción y cómo se comporta en el tiempo.