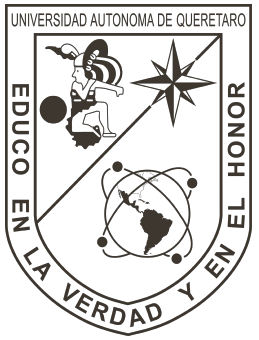
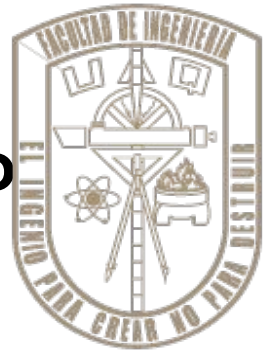


Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería en Automatización



Sistemas Digitales con Lógica Reconfigurable

M en C. Marcos Romo Avilés

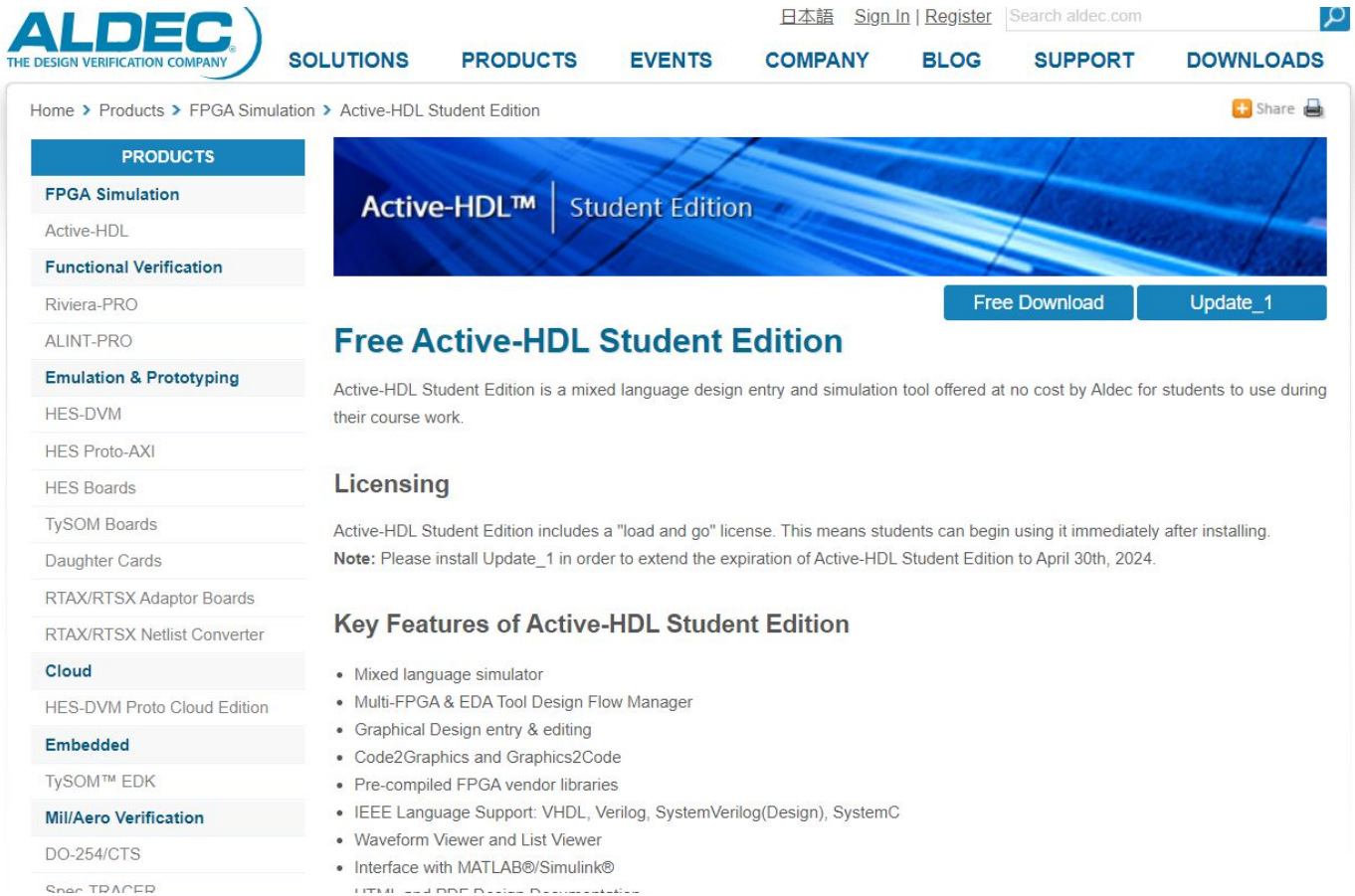
Practica 1: Entorno gráfico de active HDL

Descripción: Mostrar los comandos, entorno de descripción de Hardware y uso de compuertas para generar código.

Entregables: Implementar en simulación solo con compuertas las ecuaciones proporcionadas.

Practica 1: Entorno gráfico de active HDL

Buscar en su navegador Active HDL Student Edition



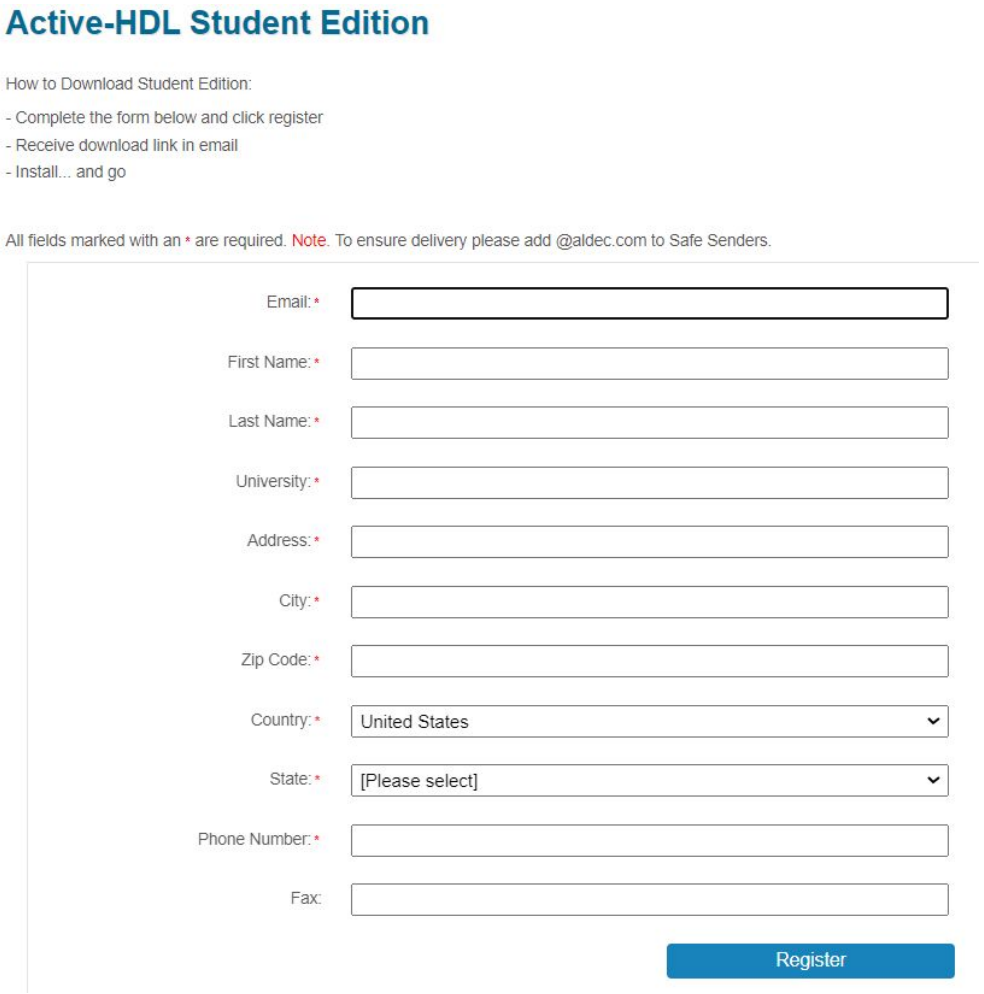
- Active-HDL es una herramienta de diseño y verificación integrada para FPGAs. Ofrece un entorno de desarrollo completo para diseñadores de circuitos digitales, facilitando la creación, simulación, depuración y documentación de proyectos. Active-HDL incluye soporte para lenguajes de descripción de hardware como VHDL, Verilog y SystemVerilog

Practica 1: Entorno gráfico de active HDL

Dar clic en Free Download

Practica 1: Entorno gráfico de active HDL

FACULTAD DE INGENIERÍA



Active-HDL Student Edition

How to Download Student Edition:

- Complete the form below and click register
- Receive download link in email
- Install... and go

All fields marked with an * are required. Note. To ensure delivery please add @aldec.com to Safe Senders.

Email:*

First Name:*

Last Name:*

University:*

Address:*

City:*

Zip Code:*

Country:* United States

State:* [Please select]

Phone Number:*

Fax:

Register

Llenar el formulario de solicitud y registrarse

Practica 1: Entorno gráfico de active HDL

- A su correo llegara un mensaje con los links de descarga.
- Se debe iniciar la descarga del archivo Active-HDL_SE-2022 y Active-HDL-Student-2022_Update1

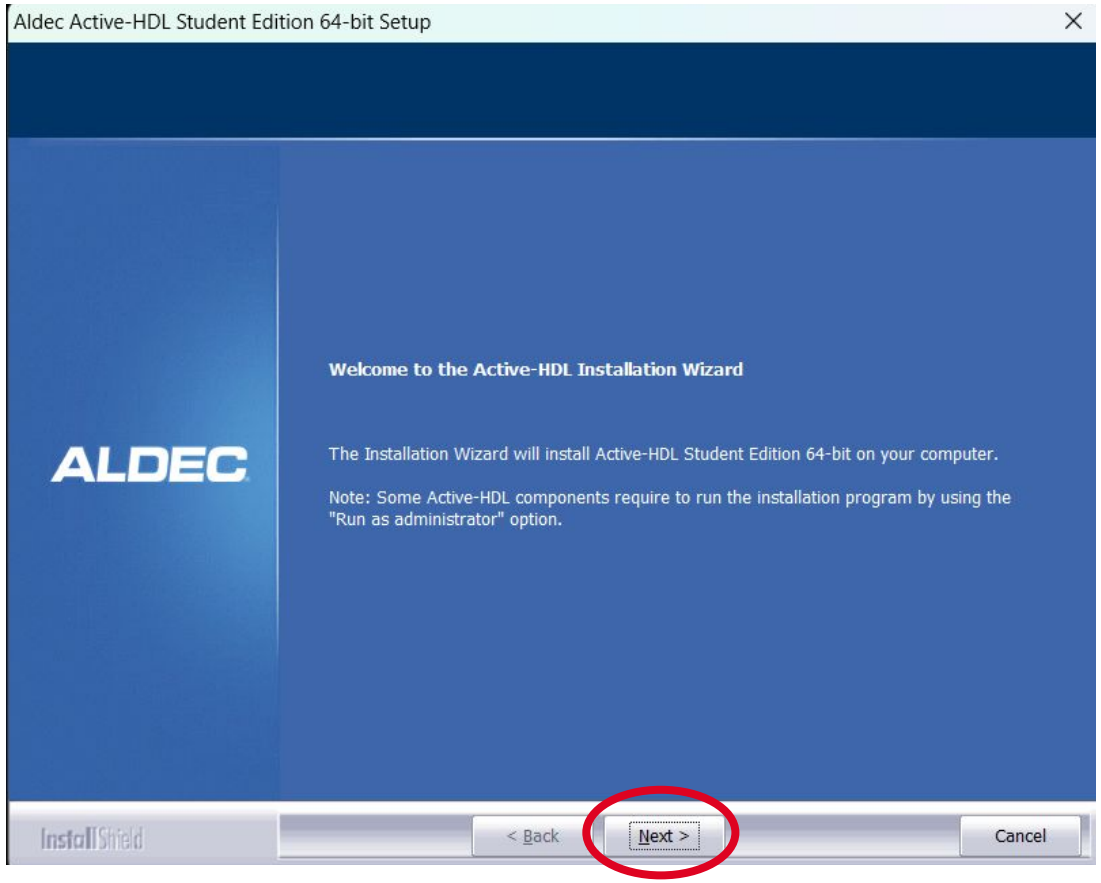
Download Information
==========
Click the links below to download Active-HDL Student Edition.

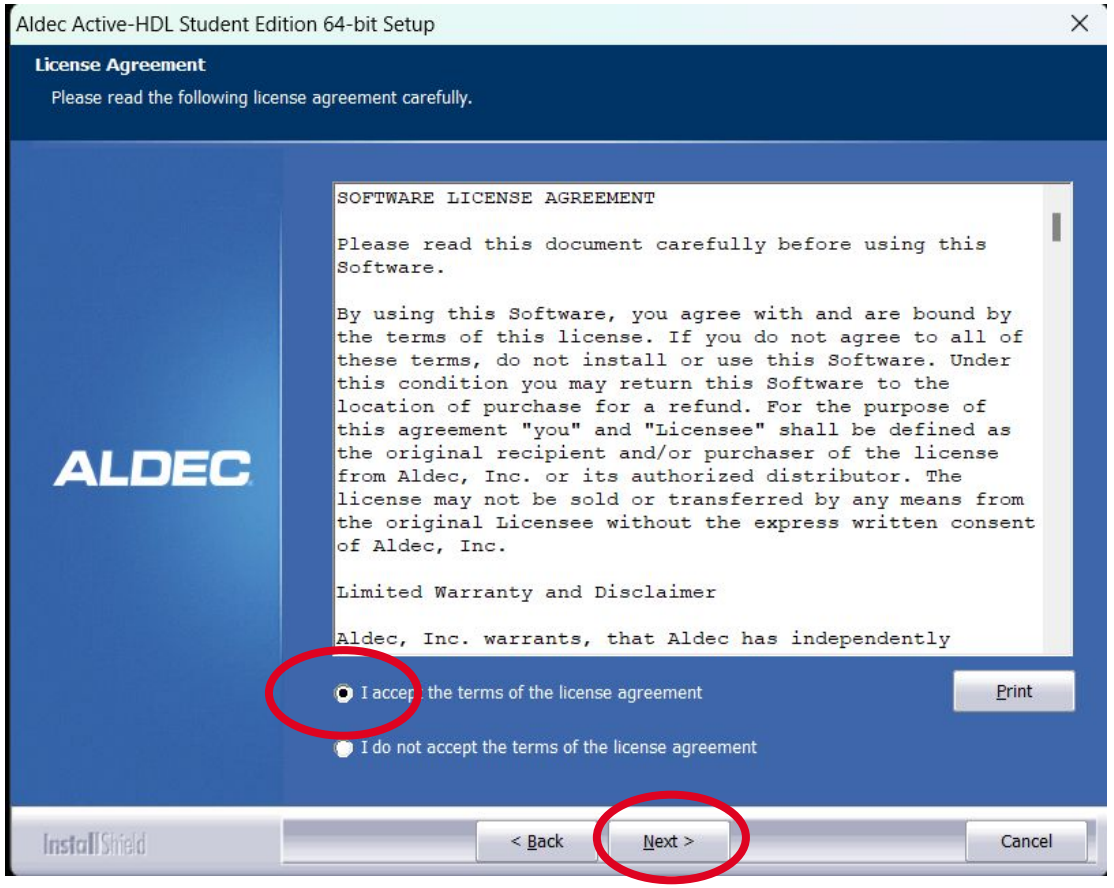
Important: All files need to be downloaded to install the tool (Do not change file names or extensions, save files in one place).
This link(s) will expire in 24 hour.
If the link has expired you are required to re-register.

Main installation file(s):
Active-HDL_SE-2022_x64_main_setup.exe: <https://www.aldec.com/students/student_download.php?i=192778&p=41&e=DNZDMUUCUMJFTJBDRTAZODDDCVGAHUGQ>

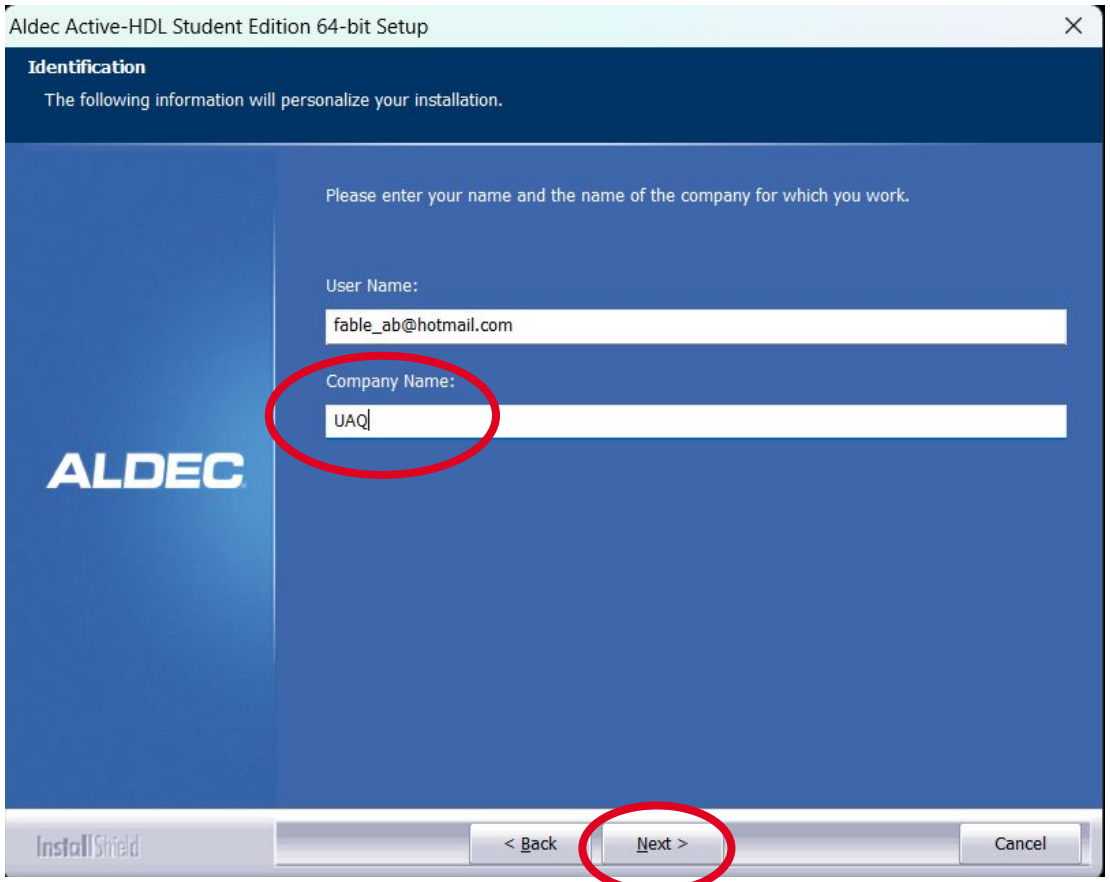
Update File(s):
Active-HDL-Student-Edition-2022_Update1.exe: <https://www.aldec.com/students/student_download.php?i=192778&p=55&e=DNZDMUUCUMJFTJBDRTAZODDDCVGAHUGQ>

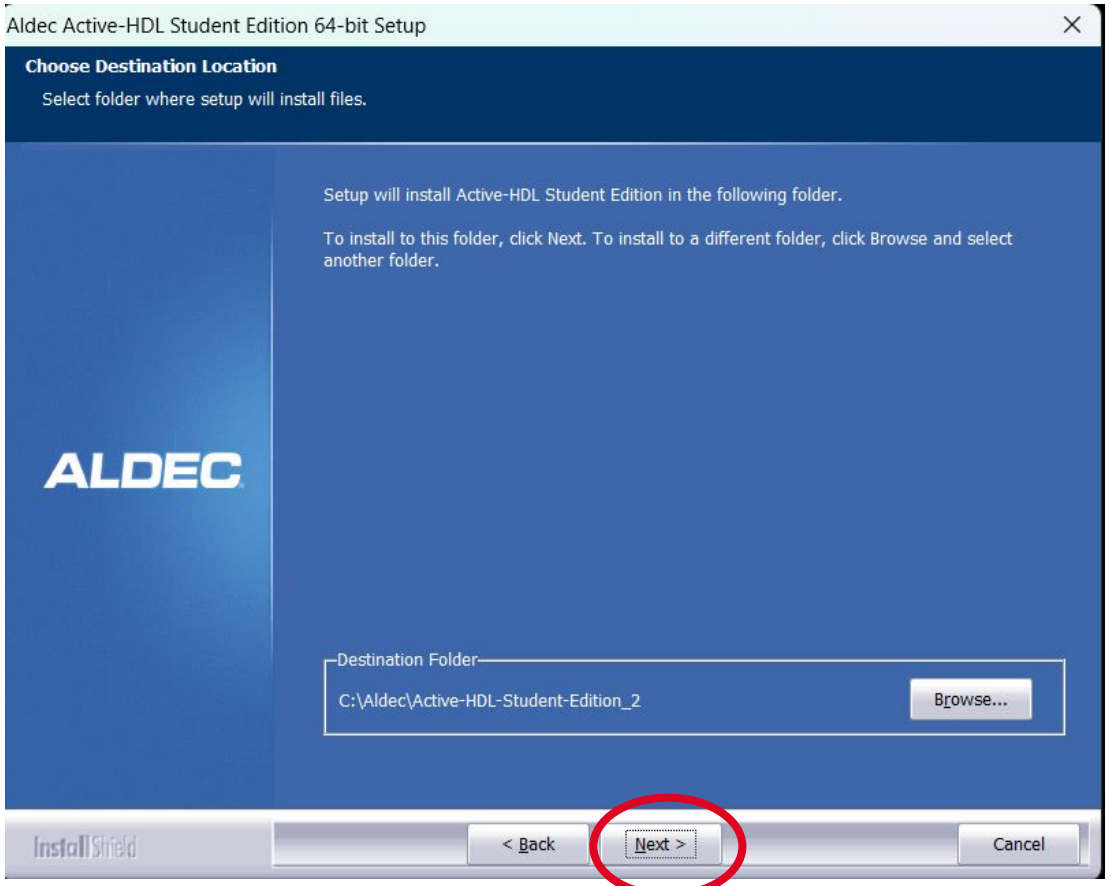
Practica 1: Entorno gráfico de active HDL



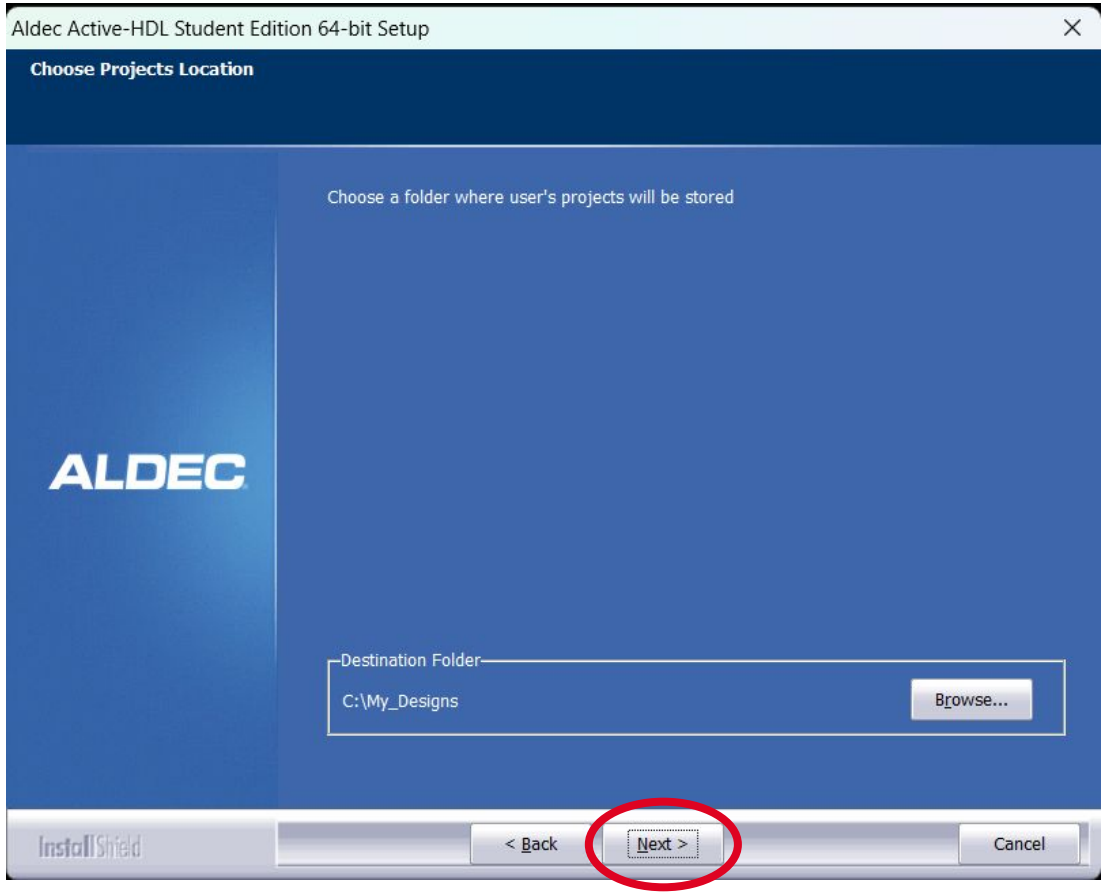


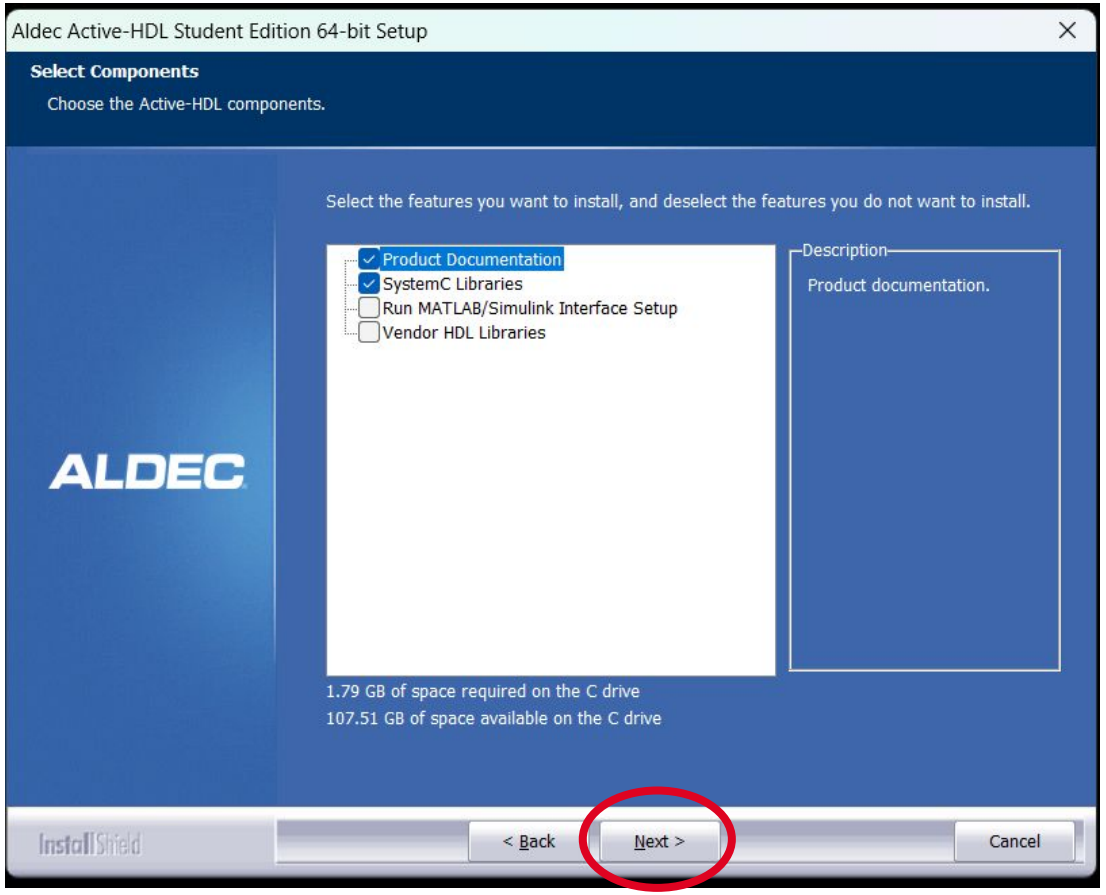
Practica 1: Entorno gráfico de active HDL



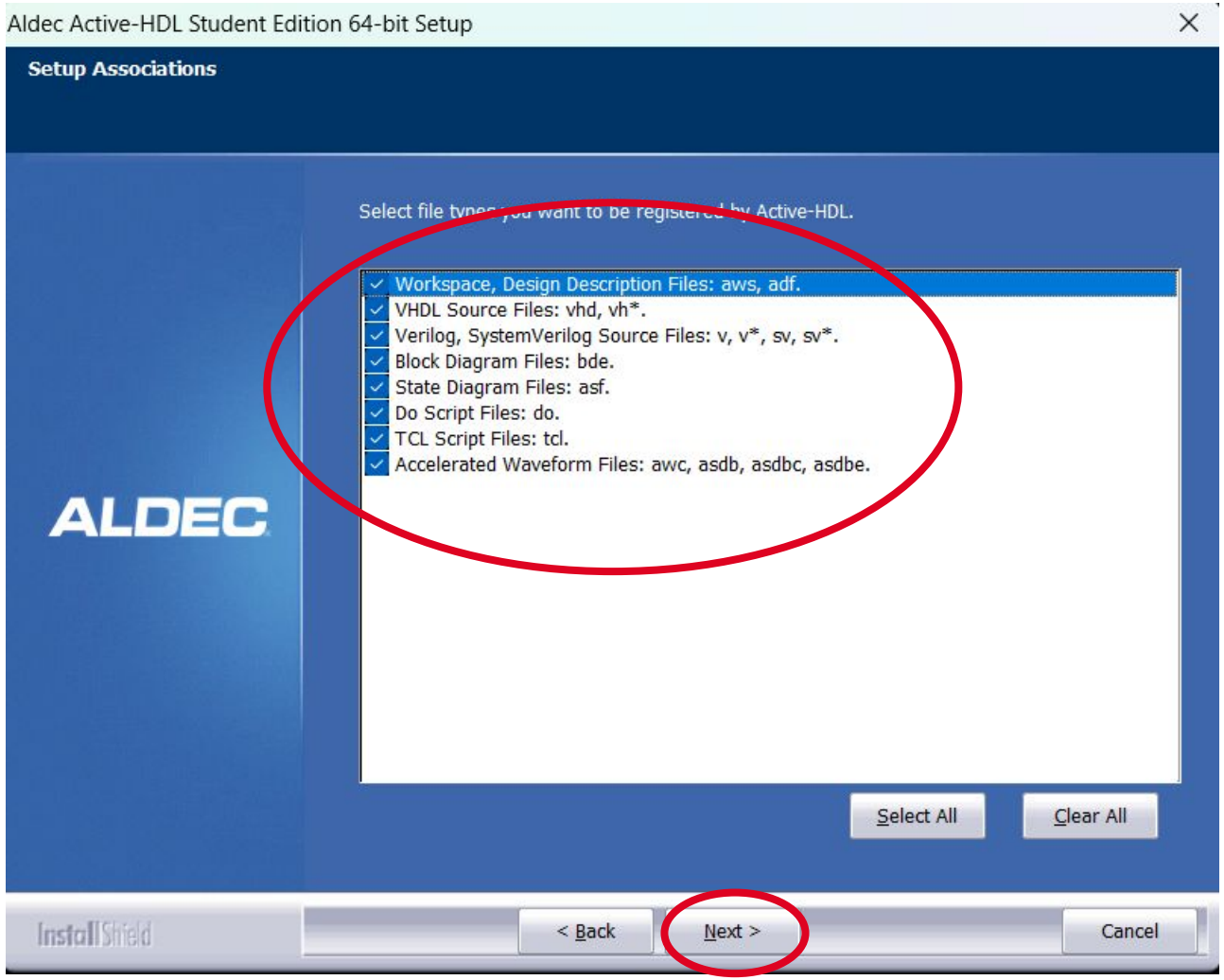


Practica 1: Entorno gráfico de active HDL

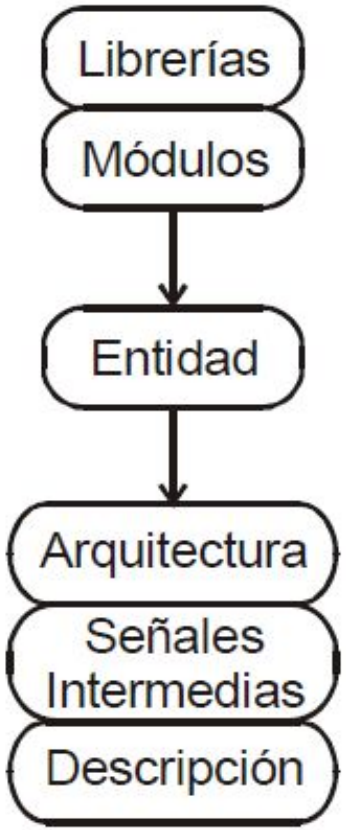




Practica 1: Entorno gráfico de active HDL



Practica 1: Introducción



- **Librería**

Se especifica el tipo de librería a utilizar, donde se puede utilizar la desarrollada por el IEEE, estándar, o alguna en particular desarrollada por algún fabricante o proveedor.

```
library IEEE;
```

- **Módulos**

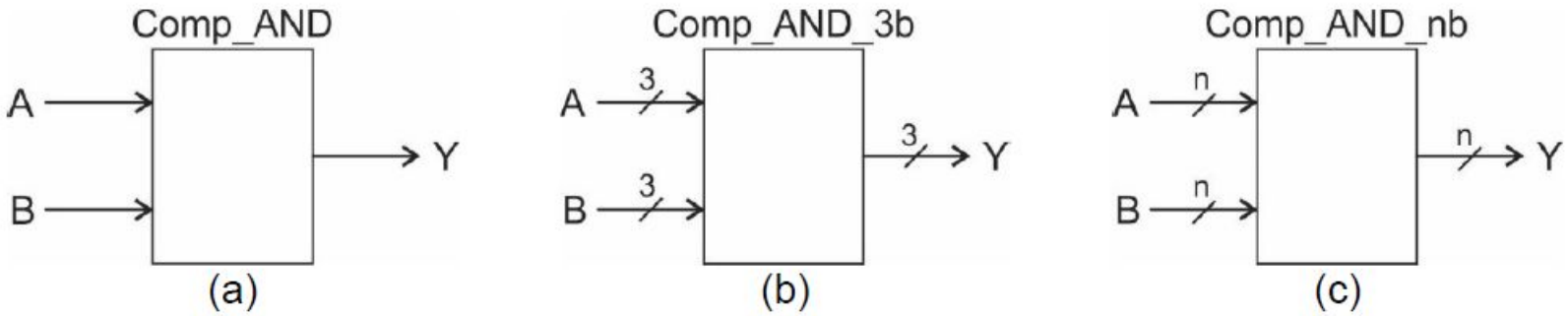
Las librerías están divididas en módulos, de tal forma que solamente usaremos los necesarios. Los módulos sirven para especificar lo que se desea utilizar de la librería.

```
Use IEEE.std_logic_1164.all;
```

Practica 1: Introducción

- **Entidad**

Se define la caja negra del componente. La definición se realiza indicando el tipo y nombre de las terminales del componente. En la figura se puede observar 3 tipos diferentes de cajas negras, para una compuerta AND. Para este propósito, se utilizan Bloques Descriptivos de Hardware (BDH).



3 tipos de entidades utilizando BDH. (a) Compuerta AND de 1 bit. (b) Compuerta AND de 3 bits. (c) Compuerta AND genérica de n bits.

Practica 1: Introducción

- **Entidad**

Se define la caja negra del componente. La definición se realiza indicando el tipo y nombre de las terminales del componente. En la figura se puede observar 3 tipos diferentes de cajas negras, para una compuerta AND. Para este propósito, se utilizan Bloques Descriptivos de Hardware (BDH).

```
Entity nombre_entidad is

-- Declaración de valores genéricos
Generic(
Nombre de la variable : tipo := valor;
Nombre de la variable : tipo := valor;
Nombre de la variable : tipo := valor
);
-- Declaración de terminales o señales externas
Port(
Nombre de la señal 1: modo tipo; -- Señal externa 1
Nombre de la señal 2: modo tipo; -- Señal externa 2
Nombre de la señal k: modo tipo -- Señal externa k
);
end nombre_entidad;
```

Practica 1: Introducción

- El “**nombre_entidad**” representa el nombre de la entidad que se va a crear. El “nombre de la señal” se puede formar bajo las mismas reglas de la declaración de variables en un lenguaje de programación, con la excepción de que no debe haber dos caracteres ‘_’ consecutivos. Así mismo, se debe considerar que el lenguaje es insensible a mayúsculas/minúsculas.
- **La sección Generic** dentro de una entidad se utiliza para definir parámetros que pueden ser configurados externamente cuando se llama la entidad en un diseño. Estos parámetros son valores constantes que se pueden usar para generalizar y configurar aspectos del diseño de un componente, como tamaños de buses, temporizaciones, configuraciones funcionales, entre otros, sin tener que modificar el código interno del componente.

Practica 1: Introducción

- El “**modo**” indica si la señal es entrada (in), salida (out) o entrada/salida (inout).
- El “**tipo**” se refiere al tipo de señal. Las señales pueden ser de 1 bit de lógica estándar (std_logic), o bien, un bus de n bits de lógica estándar, el cual se representa como std_logic_vector (n-1 downto 0), donde el valor n-1 corresponde al bit más significativo y el 0 al bit menos significativo, y en total representa n bits. También se puede expresar como std_logic_vector (0 upto n-1), donde el valor 0 corresponde al bit más significativo y el n-1 al bit menos significativo, y en total representa n bits.

```
Entity Comp_AND is
-- Declaración de terminales ó señales externas
Port(
A : in std_logic; -- Señal de entrada A
B : in std_logic; -- Señal de entrada B
Y : out std_logic -- Señal de salida Y
);
end Comp_AND;
```

Practica 1: Introducción

- Sintaxis para la declaración de la entidad de la compuerta AND de 2 bits

```
Entity Comp_AND_3b is
-- Declaración de terminales ó señales externas
Port(
A : in std_logic_vector(2 downto 0); -- Señal de entrada A
B : in std_logic_vector(2 downto 0); -- Señal de entrada B
Y : out std_logic_vector(2 downto 0) -- Señal de salida Y
);
end Comp_AND_3b;
```

- Sintaxis para la declaración de la entidad de la compuerta AND genérica

```
Entity Comp_AND_nb is
Generic(
n : integer := 4 -- Se define el valor genérico de n
);
-- Declaración de terminales ó señales externas
Port(
A : in std_logic_vector(n-1 downto 0); -- Señal de entrada A
B : in std_logic_vector(n-1 downto 0); -- Señal de entrada B
Y : out std_logic_vector(n-1 downto 0) -- Señal de salida Y
);
end Comp_AND_nb;
```

Practica 1: Introducción

- **Arquitectura**

Es la parte más importante, pues es donde se define la interacción de las señales de entrada para obtener la señal de salida deseada. En la definición de la arquitectura se manejan los siguientes términos:

- **Comentarios:** Se marca el inicio de un comentario con dos caracteres de signo menos, '--'; cualquier cosa que esté escrito después de estos caracteres, es ignorado por el compilador, hasta terminar el renglón.
- **Palabras reservadas:** Son palabras que no pueden ser usadas con fines distintos a los que están asignados. Por ejemplo, entity, port, begin, end, architecture, in, out, entre otros.
- **Nombre_arquitectura:** Es el nombre asignado a la arquitectura. Este nombre puede ser igual o diferente al nombre de la entidad.
- **Signal:** Se utiliza para declarar una señal interna. Las señales internas NO manejan dirección, por lo que se pueden utilizar para realizar conexiones internas, sin importar si entran o salen a un proceso específico.

Practica 1: Introducción

```
-- Se define la arquitectura de conexiones del sistema digital
architecture nombre_arquitectura of nombre_entidad is
-- Se declaran las señales internas de la arquitectura
signal nombre de la señal 1 : tipo;
signal nombre de la señal 2 : tipo;
begin
-- Se escribe la descripción del hardware,
-- donde se pueden tener
-- combinacional y/o secuencial.
End nombre_arquitectura;
```

Practica 1: Introducción

1. A + (B + C)
2. A(BC)
3. A(B + C)
4. $\bar{c}$d + a$\bar{c}$ + bd + a$\bar{c}$
5. $\bar{a}$+ $\bar{b}$ $\bar{c}$

Practica 1: Test bench

- **Librerías y uso:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Estas líneas indican que el testbench usa la biblioteca IEEE y específicamente el paquete STD_LOGIC_1164, que proporciona definiciones para manejar señales lógicas estándar como '0', '1', 'Z' (alta impedancia), etc.

- **Declaración de la entidad:**

```
entity LogicFunctions_tb is
-- El testbench no tiene puertos.
end LogicFunctions_tb;
```

Aquí se declara la entidad del testbench, LogicFunctions_tb. En los testbenches, normalmente no hay puertos porque sirven para simular el entorno externo de la entidad a probar.

Practica 1: Test bench

- **Declaración de la arquitectura:**

```
architecture Behavioral of LogicFunctions_tb is
```

Esta línea inicia la definición de la arquitectura Behavioral para LogicFunctions_tb.

- **Componente a probar:**

Dentro de la arquitectura, se declara un componente llamado LogicFunctions, que es la entidad que se va a probar.

```
component LogicFunctions
        Port (
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            C : in STD_LOGIC;
            D : in STD_LOGIC;
            F1 : out STD_LOGIC;
            F2 : out STD_LOGIC;
            F3 : out STD_LOGIC;
            F4 : out STD_LOGIC;
            F5 : out STD_LOGIC
        );
end component
```

Practica 1: Test bench

- **Señales para conectar con el componente:**
- Se definen señales A, B, C, D y F1 a F5. Estas señales se usarán para conectar con los puertos del componente LogicFunctions.

```
signal A, B, C, D : STD_LOGIC := '0';
signal F1, F2, F3, F4, F5 : STD_LOGIC;
```

- **Instanciación del componente:**

```
begin
uut: LogicFunctions Port Map (...);
```

- Aquí, se crea una instancia del componente LogicFunctions (a menudo denominada UUT, Unidad Bajo Prueba) y se mapean las señales del testbench a los puertos del componente.

Practica 1: Test bench

- **Proceso de estímulo:**

```
 stimulus: process
    begin
    A <= '0'; B <= '0'; C <= '0'; D <= '0';
        wait for 10 ns;
        A <= '1'; B <= '0'; C <= '0'; D <= '0';
        wait for 10 ns;
        A <= '0'; B <= '1'; C <= '0'; D <= '0';
        wait for 10 ns;
        A <= '1'; B <= '1'; C <= '0'; D <= '0';
        wait for 10 ns;
        -- Termina la simulación
        wait;
    end process stimulus
end Behavioral;
```

Este es un proceso que define cómo cambiarán las señales de entrada (A, B, C, D) con el tiempo. En este caso, se aplican diferentes combinaciones de entradas y se espera un tiempo determinado (wait for 10 ns;) para observar cómo reaccionan las salidas (F1 a F5).