

Práctica 01: Nombre de la práctica**Alumno 1:** Flores Oropeza Osvaldo**Alumno 2:** Ramirez Aguilar Victor Saul**Alumno 3:** Reyes Lira Alejandro**Grupo:** 2MV11

Resumen

La práctica consiste en el uso de lenguajes de descripción de hardware (VHDL y Verilog) para implementar y simular circuitos digitales en una tarjeta de desarrollo FPGA. Se trabajan tanto circuitos combinacionales como secuenciales: funciones lógicas básicas, un registro de 8 bits con flip-flops tipo D, un contador binario ascendente/descendente de 4 bits con salida a display de 7 segmentos, y generadores de sonido. Finalmente, se integran los distintos módulos en un Top Level Design (TLD), que concentra todas las funciones en un solo proyecto. El objetivo es que el estudiante se familiarice con el proceso completo de diseño digital: programación, simulación, implementación y prueba en hardware real.

Abstract

The practice focuses on the use of Hardware Description Languages (VHDL and Verilog) to implement and simulate digital circuits on an FPGA development board. Both combinational and sequential circuits are developed: basic logic functions, an 8-bit register based on D flip-flops, a 4-bit up/down binary counter with a 7-segment display output, and sound generators. Finally, all the modules are integrated into a Top Level Design (TLD), which combines every function into a single project. The goal is to help students become familiar with the complete digital design flow: programming, simulation, implementation, and testing on real hardware.

Resumo

A prática consiste no uso de linguagens de descrição de hardware (VHDL e Verilog) para implementar e simular circuitos digitais em uma placa de desenvolvimento FPGA. São trabalhados circuitos combinacionais e sequenciais: funções lógicas básicas, um registrador de 8 bits com flip-flops tipo D, um contador binário crescente/decrescente de 4 bits com saída em display de 7 segmentos e geradores de som. Por fim, todos os módulos são integrados em um Top Level Design (TLD), que reúne todas as funções em um único projeto. O objetivo é que o aluno se familiarize com todo o processo de projeto digital: programação, simulação, implementação e teste em hardware real.

Tabla de Contenido.

Puntos	Codigo VHDL	Codigo Verilog	Fotos	Simulación
1. Funciones				
2. Registro 8 bits				
3. Contador				
4. Sirenas				
5. TLD				
6. Encoder, Pmod, RS232				
CH				

Cuadro 1: Tabla de contenido

Pre-reporté

Como parte del pre-reporté solicitado, se presentan los siguientes puntos investigados:

Lenguajes de Descripción de Hardware (HDL)

Los lenguajes de descripción de hardware (HDL, por sus siglas en inglés) permiten describir el comportamiento y la estructura de sistemas digitales. Los más utilizados son:

- **VHDL:** Lenguaje robusto, fuertemente tipado y muy usado en la industria aeroespacial y militar. Facilita el modelado en distintos niveles de abstracción (comportamental, RTL, estructural).
- **Verilog:** Lenguaje más simple y con sintaxis similar a C. Es ampliamente empleado en aplicaciones industriales y académicas, especialmente en diseño de lógica digital.

Ambos lenguajes permiten la simulación, verificación, síntesis e implementación de circuitos digitales en CPLDs y FPGAs.

CPLD y FPGA

- **CPLD (Complex Programmable Logic Device):** Son dispositivos lógicos programables de mediana capacidad, con arquitectura basada en macroceldas. Son adecuados para implementar lógica combinacional y secuencial de menor escala.
- **FPGA (Field Programmable Gate Array):** Dispositivos programables con gran cantidad de bloques lógicos configurables, memoria interna y recursos adicionales (DSPs, PLLs, etc.). Permiten implementar desde sistemas digitales simples hasta procesadores completos.

Circuitos Externos Requeridos

Para el desarrollo de la práctica se requieren algunos circuitos adicionales:

- **Etapa de potencia de audio:** Amplificador de al menos 3W para excitar una bocina de 4 u 8 ohms.
- **Fuente de alimentación externa:** Regulada para garantizar un funcionamiento estable de los módulos adicionales.
- **Periféricos:** PmodALS (sensor de luz), encoder rotatorio y conexiones para RS232, dependiendo de las actividades a desarrollar.

Metodología General

El flujo de trabajo típico en la práctica consiste en:

1. Creación del proyecto en el software de desarrollo (ISE, Vivado o Quartus).
2. Escritura del código HDL en VHDL y Verilog.
3. Simulación y verificación funcional en el simulador integrado.
4. Síntesis e implementación del diseño.
5. Generación del archivo de programación (*.bit o *.jed).
6. Programación de la FPGA y verificación física de los resultados.

Introducción

En esta práctica se busca que el estudiante se familiarice con el uso de lenguajes de descripción de hardware (HDL), principalmente VHDL y Verilog, para implementar y simular circuitos digitales dentro de una tarjeta de desarrollo FPGA.

Los circuitos a desarrollar incluyen tanto sistemas combinacionales como secuenciales: funciones lógicas básicas a partir de una tabla de verdad, un registro de 8 bits basado en flip-flops tipo D, un contador binario ascendente/descendente de 4 bits con salida a un display de 7 segmentos, así como generadores de sonido (sirenas).

Finalmente, se integrarán los módulos en un *Top Level Design* (TLD), el cual permite concentrar todas las funciones en un único diseño jerárquico. De esta forma, el alumno recorre todo el flujo de diseño digital: codificación, simulación, síntesis, implementación y prueba en hardware real, aplicando conocimientos teóricos a aplicaciones prácticas.

Registro 8 bits

Un registro de 8 bits es un circuito electrónico para el almacenamiento y proceso de datos de 8 bits.

Para el desarrollo del registro de 8 bits se va a tomar como base el uso de un flip flop tipo D (Figura: 1) el cual funciona para guardar un bit de la entrada D en el momento en el que exista un flanco de subida en la entrada del CLK, y este se va a mostrar en la salida Q, y en la salida

Q' se va a mostrar el estado en el que se encontraba la entrada D antes del cambio del flanco de subida en el CLK.

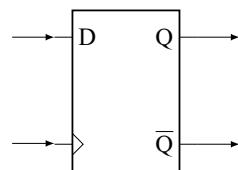


Figura 1: Flip-Flop D

Contador 4 bits Display de 7 segmentos

Para la visualización del contador de 4 bits vamos a tener apoyo de un display de 7 segmentos de 4 dígitos (Figura: 2) que viene incluido en la tarjeta de desarrollo cyclone 4.

El display de 7 segmentos es un elemento que nos permite visualizar números y letras dependiendo una combinación adecuada en los pines de entrada (a-f) y dado que esta construido por didos leds necesita una corriente máxima la cual es limitada por una resistencia que ya viene incluida de igual forma en la tarjeta de desarrollo, y al ser un display de 4 dígitos tiene 4 pines extras para definir que dígito es que vamos a mostrar.

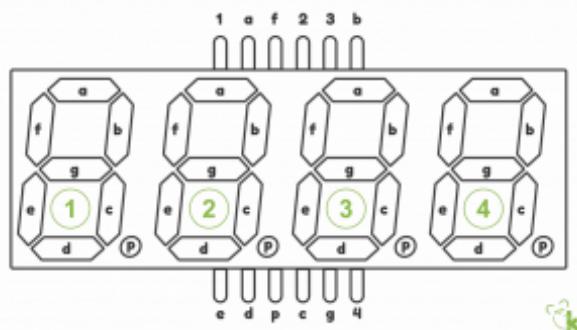


Figura 2: Display de 7 segmentos 4 dígitos cátodo común

Modulo de Voz.

El módulo ISD1820 es un grabador y reproductor de voz basado en el chip ISD1820. Su funcionamiento se centra en capturar, almacenar y reproducir mensajes de audio.

Utiliza un micrófono integrado para capturar el sonido. Al presionar el botón REC, el chip almacena la señal de audio de forma analógica en una memoria no volátil, donde la duración de la grabación es de aproximadamente 10 segundos.

Existen 2 modos para su reproducción. PLAYE (Playback, Edge-activated): Al presionarlo una vez, reproduce el mensaje completo grabado.

PLAYL (Playback, Level-activated): Mientras se mantiene presionado, reproduce el mensaje de forma continua. Al soltarlo, la reproducción se detiene.

Control: Puede operarse manualmente con sus botones o de forma automática de forma externa enviando señales a sus pines de control (REC, PLAYE, PLAYL).

El chip incluye un amplificador de audio interno para manejar altavoces pequeños de aproximadamente 8 ohms y 0.5 Watts.

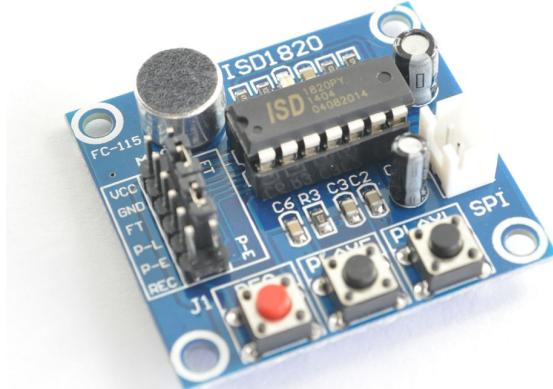


Figura 3: Modulo ISD1820

Comunicación RS232

RS232 es una forma de comunicación en serie, lo que significa que la información se envía entre dispositivos un bit a la vez. Es una forma de comunicación asincrónica, lo que significa que los dispositivos de envío y recepción pueden operar en diferentes momentos.

El estándar RS232 define los detalles de cómo se envían y reciben los datos entre dispositivos. Las principales características del estándar RS232 son las siguientes:

Niveles de voltaje: La comunicación RS232 usa señales de voltaje para representar datos binarios. Un voltaje entre -3 V y -15 V representa un "1" binario (también conocido como "Marca"), mientras que un voltaje entre +3 V y +15 V representa un "0" binario (también conocido como "Espacio"). Transmisión de datos: RS232 admite comunicación dúplex completo, lo que significa que los datos se pueden enviar y recibir en ambas direcciones al mismo tiempo. Enmarcado: Cada paquete de datos enviado a través de RS232 contiene un bit de inicio, de 5 a 9 bits de datos, un bit de paridad opcional para la detección de errores y uno o dos bits de parada. Esta estructura se denomina "marco". Tasa de baudios: La tasa de baudios es el número de cambios de señal por segundo que se envían o reciben a través de la línea. En RS232, la velocidad en baudios generalmente se especifica en bits por segundo (bps). Líneas de control: RS232 utiliza varias líneas de control para gestionar el flujo de datos entre el transmisor y el receptor. Estos incluyen líneas como "Terminal de datos listo" (DTR), "Conjunto de datos listo" (DSR), "Solicitud de envío" (RTS) y "Listo para enviar" (CTS). Comprobación de paridad: Este es un método para detectar errores en los datos transmitidos. La verificación de paridad agrega un bit adicional (el bit de paridad) a cada palabra de datos (generalmente un byte), que se establece para garantizar que el número total de 1 bits en la palabra (incluido el bit de paridad) sea siempre par o impar, según dependiendo de si se usa paridad par o impar.

Desarrollo

Actividad 1. Funciones de Salida

Escrutura, simulación e implementación de los códigos en VHDL y Verilog para programar las funciones de la tabla 1.1, repetida enseguida para su fácil implementación. Reportar diagrama de la entidad, códigos (VHDL, Verilog UCF), simulación y fotos editadas con texto explicativo.

Writing, simulation and implementation of the codes in VHDL and Verilog to program the functions of table 1.1, repeated immediately for easy implementation. Report codes, simulation and photographs.

Interpretación física de la tabla de verdad

Para dar un significado práctico a la tabla de verdad mostrada, se propone un escenario de **sistema de control de riego en un invernadero**, donde las variables de entrada y salida corresponden a sensores y actuadores reales.

Entradas (sensores)

- A : Sensor de humedad del suelo (1 = suelo seco, 0 = suelo húmedo).
- B : Sensor de radiación solar (1 = soleado, 0 = nublado).
- C : Sensor de nivel en tanque de agua (1 = tanque lleno, 0 = tanque vacío).

Salidas (actuadores/indicadores rediseñados)

- **S1: Indicador de Alerta / Modo de Acción (XOR)**

Expresión (SOP mínima): $S1 = \overline{A} \overline{B} C + \overline{A} B \overline{C} + A \overline{B} \overline{C} + ABC = A \oplus B \oplus C$.

Qué hace y cómo funciona: S1 se enciende únicamente cuando un número *impar* de sensores está activo (solo uno de ellos, o los tres a la vez). Funciona como un selector de modo lógico.

- *Razón física:* Indica que el sistema no está en un estado de reposo simple (como 000) ni en un estado de transición balanceado. Representa una condición que requiere una *acción específica* o una *alerta*, como cuando solo un sensor se activa (ej. solo el suelo está seco) o cuando todas las condiciones están al máximo (listo para regar bajo el sol).

- **S2: Indicador General de Actividad (OR)**

Expresión (SOP mínima): $S2 = A + B + C$.

Qué hace y cómo funciona: S2 se enciende si *cualquiera* de los sensores (A, B o C) está activo. Solo permanece apagado en el estado de reposo total (000).

- *Razón física:* Es el indicador más simple y fundamental del sistema. Sirve como una luz de "Sistema Ocupado" o "Condición Detectada", informando al operador que al menos una variable del invernadero no está en su estado base (húmedo, nublado y tanque vacío).

- **S3: Control de Ventilación por Coherencia Ambiental (XNOR)**

Expresión (SOP mínima): $S3 = \overline{A} \overline{B} + AB = A \odot B$.

Qué hace y cómo funciona: S3 activa un actuador cuando las condiciones de humedad del suelo (A) y sol (B) son *coherentes*; es decir, ambas están inactivas (húmedo y nublado) o ambas están activas (seco y soleado).

- *Razón física:* Es ideal para manejar la ventilación. Si está húmedo y nublado ($A = 0, B = 0$), las compuertas se cierran para conservar calor. Si está seco y soleado ($A = 1, B = 1$), las compuertas se abren para disipar el calor y prepararse para el riego. Si las condiciones son mixtas, la ventilación permanece en un estado neutro.

■ S4: Ventilador de Circulación

Expresión (SOP mínima): $S4 = \overline{A}BC + AB\overline{C} = B \cdot (A \oplus C)$.

Qué hace y cómo funciona: S4 enciende el ventilador únicamente cuando está *soleado* ($B = 1$) y las condiciones de agua presentan un contraste: suelo húmedo y tanque lleno ($A = 0, C = 1$) o suelo seco y tanque vacío ($A = 1, C = 0$).

- *Razón física:* Actúa para regular la temperatura y humedad en momentos de alta radiación solar. Si hay agua y humedad, previene la condensación. Si no hay agua y el suelo está seco, intenta mitigar el estrés térmico en las plantas.

■ S5: Sistema de Calefacción/Luz Artificial

Expresión (SOP mínima): $S5 = \overline{A}\overline{B}\overline{C}$.

Qué hace y cómo funciona: S5 activa un sistema de soporte (calefactor o luz de crecimiento) solo en el escenario más pasivo y frío: *suelo húmedo* ($A = 0$), *nublado* ($B = 0$) y *tanque vacío* ($C = 0$).

- *Razón física:* Compensa la falta de energía (sol) y el riesgo de bajas temperaturas en la condición de inactividad total, protegiendo a las plantas de un ambiente adverso y manteniendo un microclima estable.

■ S6: Alarma Crítica por Falta de Agua

Expresión (SOP mínima): $S6 = A \cdot \overline{C}$.

Qué hace y cómo funciona: S6 dispara una alarma sonora o visual únicamente si el suelo está *seco* ($A = 1$) y, al mismo tiempo, el tanque está *vacío* ($C = 0$).

- *Razón física:* Señaliza la condición más peligrosa para el cultivo. El sistema no puede actuar (regar) y necesita intervención humana inmediata para llenar el tanque y evitar la pérdida de las plantas.

Interpretación de casos

1. Caso 000: Suelo húmedo, sin sol, tanque vacío.

- Entradas: $A = 0, B = 0, C = 0$.
- Interpretación: Escenario de inactividad total. Ambiente frío, oscuro y sin recursos hídricos.
- Salidas:

- S3=1 (compuerta en modo conservación, ej. cerrada).
- S5=1 (sistema de calefacción activo para proteger del frío).

- *Razón:* El sistema entra en modo de protección pasiva, conservando calor y aportando energía artificialmente.

2. Caso 001: Suelo húmedo, sin sol, tanque lleno.

- Entradas: $A = 0, B = 0, C = 1$.
- Interpretación: Condiciones estables y con agua disponible. No se requiere acción de riego o ventilación.
- Salidas:

- S1=1 (alerta de estado, hay un sensor activo).
- S2=1 (indicador general activo).
- S3=1 (compuerta en modo conservación).

- *Razón:* El sistema está en espera, monitorizando una condición normal y con recursos.

3. Caso 010: Suelo húmedo, con sol, tanque vacío.

- Entradas: $A = 0, B = 1, C = 0$.
- Interpretación: Día soleado, el suelo aún no necesita agua, pero no hay reservas.
- Salidas:

- S1=1 (alerta de estado).
- S2=1 (indicador general activo).

- *Razón:* El sistema se mantiene en alerta por la combinación de sol (demanda de evaporación) y tanque vacío (sin capacidad de respuesta).

4. Caso 011: Suelo húmedo, con sol, tanque lleno.

- Entradas: $A = 0, B = 1, C = 1$.
- Interpretación: Condición ideal. Hay sol, el suelo está húmedo y hay agua de reserva.
- Salidas:

- S2=1 (indicador general activo).
- S4=1 (ventilador activo para circular aire y controlar la temperatura/humedad).

- *Razón:* Se realiza una gestión proactiva del clima para mantener las condiciones óptimas.

5. Caso 100: Suelo seco, sin sol, tanque vacío.

- Entradas: $A = 1, B = 0, C = 0$.
- Interpretación: ¡Problema! El suelo necesita agua, pero no hay ni sol ni agua en el tanque.
- Salidas:
 - S1=1 (alerta de estado).
 - S2=1 (indicador general activo).
 - S6=1 (alarma crítica por falta de agua).

- *Razón:* El sistema no puede solucionar el problema y escala la situación a una alarma para el operario.

6. Caso 101: Suelo seco, sin sol, tanque lleno.

- Entradas: $A = 1, B = 0, C = 1$.
- Interpretación: Se necesita regar y hay agua disponible, aunque esté nublado.
- Salidas:
 - S2=1 (indicador general activo).

- *Razón:* Sistema en espera para activar el riego (que sería una función lógica de $A \cdot C$).

7. Caso 110: Suelo seco, con sol, tanque vacío.

- Entradas: $A = 1, B = 1, C = 0$.
- Interpretación: ¡Emergencia! El suelo está seco, el sol acelera la deshidratación y no hay agua.
- Salidas:

- $S_2=1$ (indicador general activo).
- $S_3=1$ (compuerta en modo coherencia, ej. abierta para ventilar).
- $S_4=1$ (ventilador activo para intentar reducir el estrés por calor).
- $S_6=1$ (alarma crítica activa, es la máxima prioridad).

- Razón: El sistema activa todas las medidas paliativas posibles (ventilación) mientras alerta de la condición crítica.

8. Caso 111: Suelo seco, con sol, tanque lleno.

- Entradas: $A = 1, B = 1, C = 1$.
- Interpretación: Momento clave para el riego. El suelo lo necesita y las condiciones son de alta demanda energética.
- Salidas:

- $S_1=1, S_2=1, S_3=1$ (todos los indicadores y la ventilación están activos).

- Razón: El sistema está en su estado de máxima actividad, listo para desplegar todas sus funciones (riego, ventilación, etc.).

Tabla de verdad

Cuadro 2: Tabla de verdad completa del sistema de control

Entradas			Salidas					
A	B	C	S1	S2	S3	S4	S5	S6
0	0	0	0	0	1	0	1	0
0	0	1	1	1	1	0	0	0
0	1	0	1	1	0	0	0	0
0	1	1	0	1	0	1	0	0
1	0	0	1	1	0	0	0	1
1	0	1	0	1	0	0	0	0
1	1	0	0	1	1	1	0	1
1	1	1	1	1	1	0	0	0

Código VHDL

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

entity Ejercicio1_VH is
  Port (
    -- Entradas
```

```

A : in STD_LOGIC;
B : in STD_LOGIC;
C : in STD_LOGIC;

-- Salidas
S1 : out STD_LOGIC;
S2 : out STD_LOGIC;
S3 : out STD_LOGIC;
S4 : out STD_LOGIC;
S5 : out STD_LOGIC;
S6 : out STD_LOGIC
);

end Ejercicio1_VH;

architecture Behavioral of Ejercicio1_VH is
-- Señales internas (entradas invertidas)
signal na, nb, nc : STD_LOGIC;
begin
    -- Invertir entradas
    na <= not A;
    nb <= not B;
    nc <= not C;

    process(na, nb, nc)
    begin
        -- Lógica para S1 (001, 010, 100, 111)
        S1 <= ((not na and not nb and nc) or
                (not na and nb and not nc) or
                (na and not nb and not nc) or
                (na and nb and nc));

        -- Lógica para S2 (todas menos 000)
        S2 <= (na or nb or nc);

        -- Lógica para S3
        S3 <= ((not na and not nb) or (na and nb));

        -- Lógica para S4 (001, 011, 101)
        S4 <= ((not na and nb and nc) or (na and nb and not nc));

        -- Lógica para S5 (000, 011, 100, 111)
        S5 <= (not na and not nb and not nc);

        -- Lógica para S6 (010, 100, 101)
        S6 <= ((na and not nc));
    end process;
end Behavioral;

```

Listing 1: Código completo para el sistema de tres entradas y 6 salidas con interpretación física.

Código Verilog

```

module Ejercicio1(
    // Entradas
    input A,
    input B,
    input C,

```

```

// Salidas
output reg S1,
output reg S2,
output reg S3,
output reg S4,
output reg S5,
output reg S6
);

// Invertir entradas
wire a, b, c;
assign a = ~A;
assign b = ~B;
assign c = ~C;

always @(*) begin
    // Ahora toda la lógica usa las señales invertidas (a_in, b_in, c_in)

    // Lógica para S1 (001, 010, 100, 111)
    S1 = (~a & ~b & c) | (~a & b & ~c) | (a & ~b & ~c) | (a & b & c);

    // Lógica para S2 (todas menos 000)
    S2 = a | b | c;

    // Lógica para S3
    S3 = (~a & ~b) | (a & b);

    // Lógica para S4 (001, 011, 101)
    S4 = (~a & b & c) | (a & b & ~c) ;

    // Lógica para S5 (000, 011, 100, 111)
    S5 = ~a & ~b & ~c;

    // Lógica para S6 (010, 100, 101)
    S6 = a & ~c;
end

endmodule

```

Listing 2: Código completo para el sistema de tres entradas y 6 salidas con interpretación física.

Simulación

Por la configuración en Pull-Up de los switches en la tarjeta de desarrollo en la simulación se visualizan los valores de la tabla invertidos, ejemplo cuando la entrada es 111, la simulación muestra los valores a la salida de un 000.

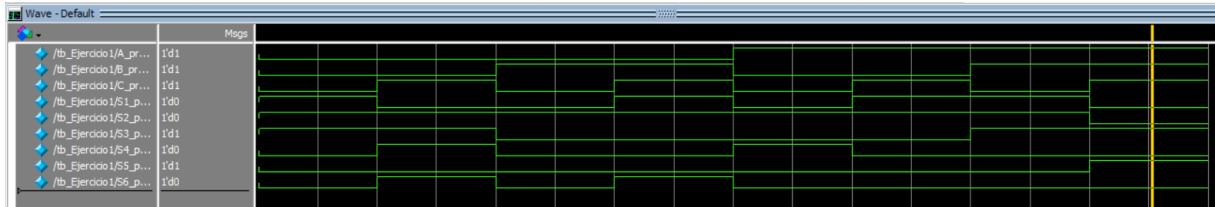
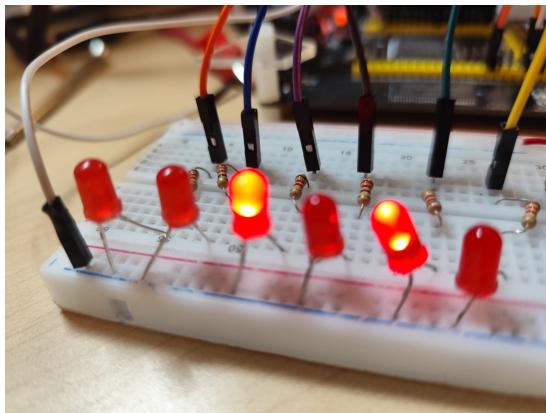
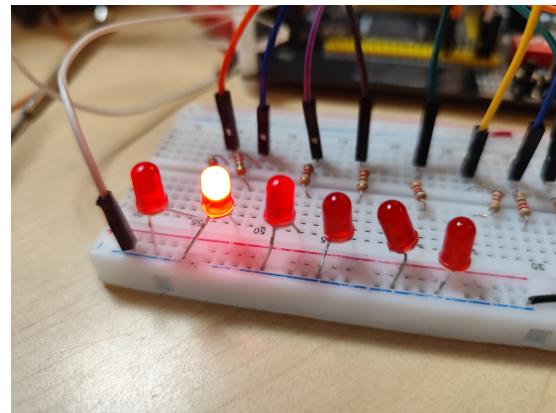


Figura 4: Simulación 1 en Questa

Fotos



(a) Leds de prueba con entrada 000



(b) Leds de prueba con entrada 101

Figura 5: Pruebas de los leds con distintas entradas

Actividad 2. Registro 8 bits

Escritura, simulación e implementación de los códigos en VHDL y Verilog para programar un registro de 8 bits basado en el FF-D. Reportar códigos, simulación y fotos.

Writing, simulation, and implementation of the codes in VHDL and Verilog to program a 4-bit register based on the FF-D. Report codes, simulation and photographs.

En este punto se realizó un registro de 8 bits donde tenemos 8 entradas y 8 salidas físicas para poder visualizar que los datos del registro se guarden correctamente al presionar un push button. Notemos que el retardo de dos ciclos se debe a que tu circuito realiza dos tareas de verificación antes de guardar los datos, cada una tomando un ciclo de reloj:

Ciclo 1: Sincronizar. El sistema primero "sincroniza" la señal del botón con el reloj interno para evitar errores. Es el primer paso para "escuchar" limpiamente tu pulsación.

Ciclo 2: Detectar el Borde. Luego, el circuito compara el estado actual del botón (presionado) con el anterior (no presionado) para confirmar que es una acción nueva. Al detectar este cambio, genera el pulso_captura que dura exactamente un ciclo.

Solo después de estos dos pasos de preparación, el pulso_captura le da la orden a los LEDs para que capturen el valor.

Código VHDL

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

```

use IEEE.NUMERIC_STD.ALL;

entity Ejercicio2_VH is
    Port (
        clk      : in  std_logic;           -- reloj de la FPGA (ej. 50 MHz)
        dipsw   : in  std_logic_vector(7 downto 0); -- DIP switches de 8 bits
        leds    : out std_logic_vector(7 downto 0)  -- LEDs de 8 bits
    );
end Ejercicio2_VH;

architecture Behavioral of Ejercicio2_VH is

    signal clk_div  : unsigned(23 downto 0) := (others => '0'); -- contador 24 bits
    signal slow_clk : std_logic := '0';

    signal reg_d : std_logic_vector(7 downto 0) := (others => '0');

begin

    -- Divisor de reloj: cada flanco de clk incrementa clk_div
    process(clk)
    begin
        if rising_edge(clk) then
            clk_div <= clk_div + 1;
            slow_clk <= clk_div(23); -- bit más alto del contador
        end if;
    end process;

    -- Registro tipo D: captura los DIP switches al flanco de slow_clk
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            reg_d <= dipsw;
        end if;
    end process;

    -- Salida a LEDs
    leds <= reg_d;

end Behavioral;

```

Listing 3: Código completo para el registro de 8 Bits

Verilog

```

module Ejercicio2(
    input wire      clk,           // reloj de la FPGA (50 MHz)
    input wire      btn_captura, // NUEVA entrada para el botón
    input wire [7:0] dipsw,       // DIP switches de 8 bits
    output reg [7:0] leds = 8'd0 // LEDs de 8 bits, inicializados en 0
);

    reg [1:0] btn_sincronizador;
    wire      pulso_captura;

```



Figura 6: Simulación 2 en Questa

```

always @(posedge clk) begin
    // Sincronizamos el botón con el reloj del sistema para evitar errores
    btn_sincronizador <= {btn_sincronizador[0], btn_captura};
end

// Genera un pulso cuando el botón pasa de '0' a '1' (se presiona)
assign pulso_captura = btn_sincronizador[1] & ~btn_sincronizador[0];

always @(posedge clk) begin
    if (pulso_captura) begin
        leds <= dipsw; // Capturamos el valor de los switches en los LEDs.
    end
    // Si no hay pulso, 'leds' mantiene su valor anterior (comportamiento de memoria).
    end
endmodule

```

Listing 4: Código completo para el registro de 8 Bits.

Simulación

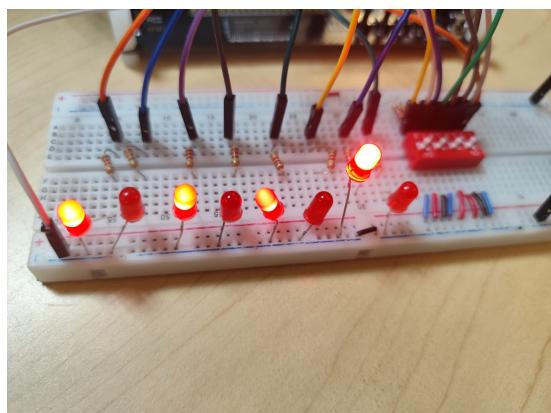
El retardo de dos ciclos se debe a que tu circuito realiza dos tareas de verificación antes de guardar los datos, cada una tomando un ciclo de reloj:

Ciclo 1: Sincronizar. El sistema primero "sincroniza" la señal del botón con el reloj interno para evitar errores. Es el primer paso para "escuchar" limpiamente tu pulsación.

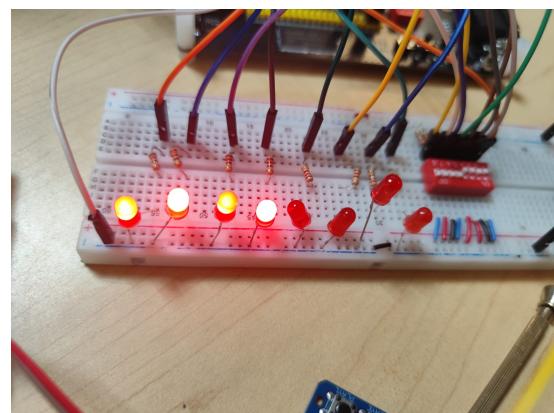
Ciclo 2: Detectar el Borte. Luego, el circuito compara el estado actual del botón (presionado) con el anterior (no presionado) para confirmar que es una acción nueva. Al detectar este cambio, genera el pulso_captura que dura exactamente un ciclo.

Solo después de estos dos pasos de preparación, el pulso_captura le da la orden a los LEDs para que capturen el valor.

Fotos



(a) Registro 1



(b) Registro 2

Figura 7: Pruebas de los leds para registro de 8 bits

Actividad 3. Contador de 4 bits

Escritura, simulación e implementación de los códigos en VHDL y Verilog para programar un contador binario de 4 bits ascendente-descendente ciclado, con interruptor de dirección (1 cuenta ascendente, 0 cuenta descendente), botón de reset en bajo y un botón de habilitación (inicio), con salida a display de 7 segmentos (0–9, A–F). Reportar códigos, simulación y fotos.
Writing, simulation, and implementation of the codes in VHDL and Verilog to program a cycled up-down binary 4-bit counter with address switch (1 up count, 0 counts down), reset button on bass and an enable button (start) with 7 seg display (0–9, A–F). Report codes, simulations, and photographs.

Para este punto se realizo un contador de 4 bits que muestra los datos de conteo en un display de 7 segmentos del 0 al 9 y de A-F de forma ascendente y descendente, al igual que cuenta con un switch para detener el conteo, un switch para cambiar la dirección de conteo y un push button para reiniciar el conteo.

Codigo VHDL

```

architecture Behavioral of P3 is

    signal hex_counter : integer range 0 to 15 := 0;
    signal clk_div      : unsigned(23 downto 0) := (others => '0');
    signal slow_clk     : std_logic := '0';

    -- Tabla de segmentos
    function to_segments(val : integer) return std_logic_vector is
    begin
        case val is
            when 0  => return "01000000";
            when 1  => return "01111001";
            when 2  => return "00100100";
            when 3  => return "00110000";
            when 4  => return "00011001";
            when 5  => return "00010010";
            when 6  => return "00000010";
            when 7  => return "01111000";
            when 8  => return "00000000";
            when 9  => return "00010000";
            when 10 => return "00001000"; -- A
            when 11 => return "00000011"; -- B
            when 12 => return "01000110"; -- C
            when 13 => return "00100001"; -- D
            when 14 => return "00000110"; -- E
            when 15 => return "00001110"; -- F
            when others => return "11111111";
        end case;
    end function;

begin

    -- Activar solo el primer dígito (ánodo común)
    digit_en <= "1110";

    -- Divisor de reloj para generar reloj lento (~0.5s si clk = 50MHz)
    process(clk)
    begin
        if rising_edge(clk) then
            clk_div <= clk_div + 1;
            if clk_div = 0 then
                slow_clk <= not slow_clk;
            end if;
        end if;
    end process;

    -- Proceso principal con control por switches
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            if reset_btn = '0' then
                hex_counter <= 0;
            elsif start_sw = '0' then -- Solo cuenta si el switch está
aktivado
                if dir_sw = '1' then
                    -- Contador ascendente

```

```

        if hex_counter = 15 then
            hex_counter <= 0;
        else
            hex_counter <= hex_counter + 1;
        end if;
    else
        -- Contador descendente
        if hex_counter = 0 then
            hex_counter <= 15;
        else
            hex_counter <= hex_counter - 1;
        end if;
    end if;
end if;
end process;

-- Salida a segmentos
segments <= to_segments(hex_counter);

end Behavioral;

```

Listing 5: Código completo para el contador de 4 Bits.

Verilog

```

module Ejercicio3_Veri(
    input wire clk,
    input wire reset_btn, // botón (activo en bajo)
    input wire start_sw, // switch: 1 = contar, 0 = pausa
    input wire dir_sw, // switch: 1 = ascendente, 0 = descendente
    output reg [7:0] segments, // A-G + DP
    output wire [3:0] digit_en // activación de dígito
);

// Contador hexadecimal
reg [3:0] hex_counter = 4'd0;

// Divisor de reloj
reg [23:0] clk_div = 24'd0;
reg slow_clk = 1'b0;

// Activar solo el primer dígito (ánodo común, activo en bajo)
assign digit_en = 4'b1110;

// Divisor de reloj (~0.5s si clk = 50 MHz)
always @(posedge clk) begin
    clk_div <= clk_div + 1;
    if (clk_div == 24'd0)
        slow_clk <= ~slow_clk;
end

// Proceso principal: contador con control de reset/start/dir
always @(posedge slow_clk) begin
    if (reset_btn == 1'b0) begin
        hex_counter <= 4'd0;
    end else if (start_sw == 1'b0) begin // 0 = contar
        if (dir_sw == 1'b1) begin

```

```

// Ascendente
if (hex_counter == 4'd15)
    hex_counter <= 4'd0;
else
    hex_counter <= hex_counter + 1;
end else begin
// Descendente
if (hex_counter == 4'd0)
    hex_counter <= 4'd15;
else
    hex_counter <= hex_counter - 1;
end
end

// Tabla de segmentos
always @(*) begin
    case (hex_counter)
        4'd0: segments = 8'b01000000;
        4'd1: segments = 8'b01111001;
        4'd2: segments = 8'b00100100;
        4'd3: segments = 8'b00110000;
        4'd4: segments = 8'b00011001;
        4'd5: segments = 8'b00010010;
        4'd6: segments = 8'b00000010;
        4'd7: segments = 8'b01111000;
        4'd8: segments = 8'b00000000;
        4'd9: segments = 8'b00010000;
        4'd10: segments = 8'b00001000; // A
        4'd11: segments = 8'b00000011; // B
        4'd12: segments = 8'b01000110; // C
        4'd13: segments = 8'b00100001; // D
        4'd14: segments = 8'b00000110; // E
        4'd15: segments = 8'b00001110; // F
        default: segments = 8'b11111111;
    endcase
end

endmodule

```

Listing 6: Código completo para el contador de 4 Bits.

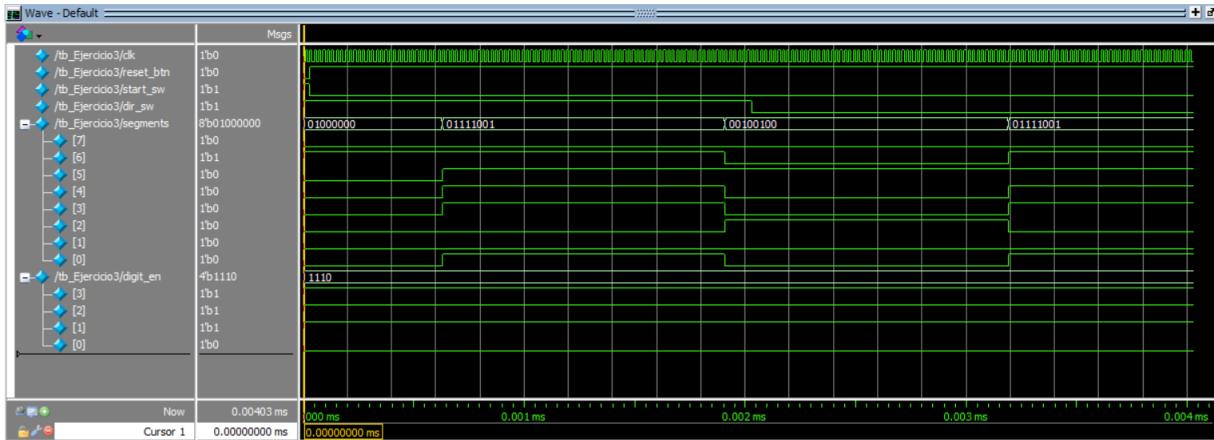
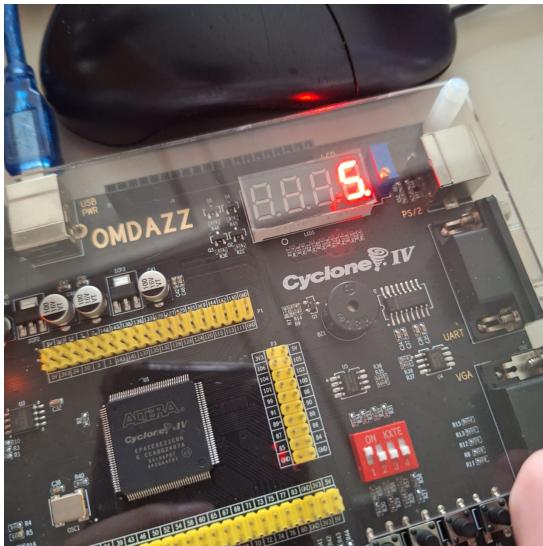


Figura 8: Simulación 3 en Questa

Simulación

Fotos



(a) Contador 4 Bits ascendente.



(b) Contador 4 Bits descendente.

Figura 9: Implementación física en un display de 7 segmentos incorporado en la tarjeta de desarrollo.

Actividad 4. Sirenas

Implementar un circuito para tocar la sirena (a) de policía y (b) de una ambulancia con distintos barridos de tonos, controlando el encendido y apagado con un interruptor (SW0). Recuerde que la salida se conecta a una bocina con su etapa de potencia de audio. Reportar códigos, fotos y videos.

Implement a circuit to play the siren of (i) police and (ii) an ambulance with different sweeps of tones, controlling them on and off with a switch (SW0). Connect the output to a speaker with audio stage power. Report code, photos and video.

Para el punto 4 se desarrollaron una sirena de policía y una sirena de ambulancia usando un buzzer incluido en la tarjeta de desarrollo y un switch para hacer el cambio entre sirenas.

Código VHDL

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

entity Ejercicio4_VH is
    generic (
        N           : integer := 24;
        POLICE_FREQ1: integer := 30000;
        POLICE_FREQ2: integer := 50000;
        AMB_MIN     : integer := 20000;
        AMB_MAX     : integer := 60000;
        STEP         : integer := 50;
        TOGGLE_TIME : integer := 50000000    -- clk=50 MHz -> 2 s
    );
    port(
        clk   : in  std_logic;
        enc   : in  std_logic;
        des   : in  std_logic;    -- 1 = policía, 0 = ambulancia
        sal   : out std_logic
    );
end Ejercicio4_VH;

architecture Behavioral of Ejercicio4_VH is

    signal div_cnt      : unsigned(N-1 downto 0) := (others => '0');
    signal time_cnt     : unsigned(31 downto 0) := (others => '0');
    signal tone_freq    : integer := POLICE_FREQ1;
    signal dir          : std_logic := '1';           -- Dirección del barrido
ambulancia
    signal police_sel   : std_logic := '0';           -- Alterna tono policía
    signal sal_reg      : std_logic := '0';

begin

    sal <= sal_reg;

    process(clk)
    begin
        if rising_edge(clk) then
            if enc = '0' then
                sal_reg      <= '0';
                div_cnt      <= (others => '0');
                time_cnt     <= (others => '0');
                tone_freq    <= POLICE_FREQ1;
                police_sel   <= '0';
                dir          <= '1';
            else
                if des = '1' then
                    -- Sirena de policía
                    if div_cnt >= to_unsigned(tone_freq, N) then
                        sal_reg <= not sal_reg;
                    end if;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

```

        div_cnt <= (others => '0');
    else
        div_cnt <= div_cnt + 1;
    end if;

    if time_cnt >= to_unsigned(TOGGLE_TIME, 32) then
        time_cnt <= (others => '0');
        police_sel <= not police_sel;
        if police_sel = '1' then
            tone_freq <= POLICE_FREQ1;
        else
            tone_freq <= POLICE_FREQ2;
        end if;
    else
        time_cnt <= time_cnt + 1;
    end if;

else
    -- Sirena de ambulancia (barrido ascendente/descendente
)
    if div_cnt >= to_unsigned(tone_freq, N) then
        sal_reg <= not sal_reg;
        div_cnt <= (others => '0');
    else
        div_cnt <= div_cnt + 1;
    end if;

    if div_cnt = 0 then
        if dir = '1' then
            tone_freq <= tone_freq - STEP;
        else
            tone_freq <= tone_freq + STEP;
        end if;

        if tone_freq < AMB_MIN then
            dir <= '0';
        elsif tone_freq > AMB_MAX then
            dir <= '1';
        end if;
        end if;
    end if;
end if;
end process;

end Behavioral;

```

Listing 7: Código completo para el sistema de alarma para una patrulla de policía y una ambulancia.

Verilog

```

module Ejercicio4 #(parameter N = 24)(
    input clk,           // Reloj del FPGA (ej: 50 MHz)
    input enc,           // Encendido/apagado de la sirena
    input des,           // Selector: 1 = policía, 0 = ambulancia
    output reg sal      // Salida a la bocina
);

```

```

parameter POLICE_FREQ1 = 30000;      // tono 1 policía
parameter POLICE_FREQ2 = 50000;      // tono 2 policía
parameter AMB_MIN = 20000;          // rango ambulancia
parameter AMB_MAX = 60000;
parameter STEP     = 50;           // paso de barrido ambulancia
parameter TOGGLE_TIME = 50_000_000;
// Con clk=50 MHz → 100M ciclos 2 s

reg [N-1:0] div_cnt = 0;
reg [31:0] time_cnt = 0;          // contador para intervalos largos
reg [15:0] tone_freq = POLICE_FREQ1;
reg dir = 1;                     // Dirección barrido ambulancia
reg police_sel = 0;              // Alterna tono policía

always @(posedge clk) begin
    if (!enc) begin
        sal <= 0;
        div_cnt <= 0;
        time_cnt <= 0;
        tone_freq <= POLICE_FREQ1;
        police_sel <= 0;
    end else begin
        if (des) begin
            // Sirena de policía (dos tonos conmutados cada 2 s)
            if (div_cnt >= tone_freq) begin
                sal <= ~sal;
                div_cnt <= 0;
            end else begin
                div_cnt <= div_cnt + 1;
            end

            // Contador de 2 segundos
            if (time_cnt >= TOGGLE_TIME) begin
                time_cnt <= 0;
                police_sel <= ~police_sel;
                if (police_sel)
                    tone_freq <= POLICE_FREQ1;
                else
                    tone_freq <= POLICE_FREQ2;
            end else begin
                time_cnt <= time_cnt + 1;
            end
        end else begin
            // Sirena de ambulancia (barrido asc/desc)
            if (div_cnt >= tone_freq) begin
                sal <= ~sal;
                div_cnt <= 0;
            end else begin
                div_cnt <= div_cnt + 1;
            end

            if (div_cnt == 0) begin
                if (dir)
                    tone_freq <= tone_freq - STEP; // sube
                else
            end
        end
    end
end

```

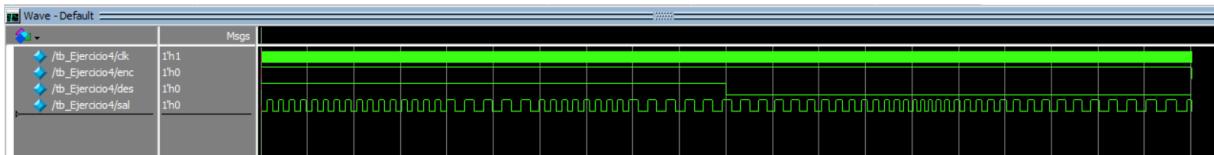


Figura 10: Simulación 4 en Questa

```

        tone_freq <= tone_freq + STEP; // baja

        if (tone_freq < AMB_MIN) dir <= 0;
        if (tone_freq > AMB_MAX) dir <= 1;
    end
end
end
endmodule

```

Listing 8: Código completo para el sistema de alarma para una patrulla de policía y una ambulancia.

Simulación

Fotos

Al ser una actividad auditiva no agregamos fotografía de la implementación, en su lugar adjuntamos un video en la asignación de Classroom junto con este pdf.

Actividad 5. Modulo de Voz

Implementar el diseño de dos circuitos utilizando alto nivel (Top Level Design, figura 1.16), uno con VHDL y uno con Verilog. Dentro de cada diseño activar un mensaje de audio utilizando un módulo reproductor comercial (ver figura 1.17), con una bocina de 4–8 Ω/Ωmega y 3–4 W con su etapa de potencia de audio. Reportar códigos y fotos. Nota: deben ser distintos a los otros puntos entregados.

Implement the design of two circuits using high level (Top Level Design, figure 1.16), one with VHDL and one with Verilog. In each design activate a voice message using a commercial module (see figure 1.17) and 4–8 Ω, 2–3 W speaker with audio power output. Report codes and photos. Note: must be different to another point reviewed.

Código Verilog

```

module Ejercicio6(
    // --- Entradas de Control ---
    input wire record_switch,      // Interruptor para activar la grabación
    input wire play_switch,        // Interruptor para activar la
    reproducción

    // --- Salidas de Estado ---
    // Deben ser 'reg' porque se asignan dentro de un bloque 'always'
    output reg record_output,
    output reg play_output
);

```

```

// Bloque que se ejecuta siempre que una de las entradas cambie
always @(*) begin
    // Lógica para la salida de grabación
    if (record_switch == 0) begin
        record_output = 1;
    end else begin
        record_output = 0;
    end

    // Lógica para la salida de reproducción
    if (play_switch == 0) begin
        play_output = 1;
    end else begin
        play_output = 0;
    end
end

endmodule

```

Listing 9: Código en para grabación y reproducción de audio.

Actividad 6. Encoder y protocolo RS232

Implementar con TLD un circuito de prueba del giro de un encoder mecánico rotatorio (ver figura 1.18), mostrado en las referencias, uno para el PmodALS, o el del módulo RS232 con una separación de por lo menos 1m (ver figuras 1.19 y 1.20), mostrado en el documento (PDLP 05 RS232.pdf), u otro módulo externo similar. Reportar códigos y fotos.

Implement using TLD a test circuit for rotatory mechanical encoder (see figure 1.18), showing in references, one for the PmodALS, or RS232 module with at least 1m between the boards (see figures 1.19 and 1.20) o showed in the document (PDLP 05 RS232.pdf) or another similar external module. Report codes and photos.

En el punto 6 realizamos un Top Level Designed para realizar la comunicación entre una CPLD con un chip MAX II, y una tarjeta de desarrollo cyclone 4, por medio del protocolo de comunicación RS232, haciendo que la cyclone 4 lea los datos del encoder mecánico y mande los datos por el protocolo de comunicación hacia la CPLD y esta la muestre visualmente por medio de LEDS.

Códigos VHDL

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity TXRXtop is
Port (
    clk,rst: in STD_LOGIC;
    clk_enc_pin : in std_logic;          -- Pin CLK (A) del encoder
    dt_enc_pin  : in std_logic;          -- Pin DT (B) del encoder
    sw_enc_pin  : in std_logic;          -- Pin SW (Botón) del encoder
    leds: out std_logic_vector(7 downto 0); -- LD (Muestra el valor RX o
                                            -- TX)
    LDtx : out STD_LOGIC;                -- LED indicador de TX
    tx : out STD_LOGIC;                  -- Salida RS232 TX
    rx : in std_logic
);

```

```

end TXRXtop;

architecture Arq_TXRX of TXRXtop is

-- Señal interna para el valor del contador del encoder
signal encoder_data : std_logic_vector(7 downto 0);

component Encoder_VHDL
Port (
    clk      : in  std_logic;
    reset    : in  std_logic;
    clk_pin : in  std_logic;
    dt_pin   : in  std_logic;
    sw_pin   : in  std_logic;
    cout     : out std_logic_vector(7 downto 0)
);
end component;

component Trans
Port (
    clk: in STD_LOGIC;
    rst: in STD_LOGIC;
    sw : in STD_LOGIC_VECTOR(7 downto 0);
    LDtx : out STD_LOGIC;
    tx : out STD_LOGIC
);
end component;

-- componente U2 RX
component Recep
Port (
    clk: in std_logic;
    rst: in std_logic;
    rx: in std_logic;
    leds: out std_logic_vector(7 downto 0)
);
end component;

begin

-- INSTANCIA 0: Módulo del Encoder
U0 : Encoder_VHDL
port map(
    clk => clk,
    reset => rst,
    clk_pin => clk_enc_pin,
    dt_pin => dt_enc_pin,
    sw_pin => sw_enc_pin,
    cout => encoder_data        -- Salida del encoder
);

-- INSTANCIA 1: Módulo Trans
U1 : Trans
port map(
    clk => clk,
    rst => rst,
    sw => encoder_data,          -- ¡CONEXIÓN CLAVE! Ahora envía el valor

```

```

    del encoder.
      LDtx => LDtx,
      tx => tx
  );

-- INSTANCIA 2: Módulo Recep
U2 : Recep
port map(
  clk => clk,
  rst => rst,
  rx => rx,
  leds => leds
);
end Arq_TXRX;

```

Listing 10: Código TLD.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Encoder_VHDL is
  port (
    -- Entradas de Reloj y Reset
    clk      : in std_logic;
    reset    : in std_logic;

    -- Pines del Encoder
    clk_pin  : in std_logic;                      -- Pin CLK (A) del encoder
    dt_pin   : in std_logic;                      -- Pin DT (B) del encoder
    sw_pin   : in std_logic;                      -- Pin SW (Switch/Botón) del
encoder

    -- Salidas
    cout     : out std_logic_vector(7 downto 0) -- Salida del contador
  );
end entity Encoder_VHDL;

architecture Behavioral of Encoder_VHDL is

  -- Señales internas para Sincronización y Antirrebote (Debouncing)
  signal clk_sync : std_logic_vector(1 downto 0) := "00";
  signal dt_sync  : std_logic_vector(1 downto 0) := "00";
  signal sw_sync  : std_logic_vector(1 downto 0) := "00";

  -- Contador principal
  signal count_value : unsigned(7 downto 0) := (others => '0');

begin

  -- P1: Proceso de Sincronización y Antirrebote
  process (clk) is
  begin
    if rising_edge(clk) then
      clk_sync <= clk_sync(0) & clk_pin;
      dt_sync  <= dt_sync(0) & dt_pin;
      sw_sync  <= sw_sync(0) & sw_pin;
    end if;
  end process;

```

```

    end process;

    -- P2: Proceso de Detección de Giro y Control del Contador
process (clk) is
begin
    if rising_edge(clk) then
        if reset = '1' then
            count_value <= (others => '0');
        else
            -- Detección de flanco de bajada en CLK
            if clk_sync(1) = '1' and clk_sync(0) = '0' then
                -- Lógica de giro
                if dt_sync(0) /= clk_sync(0) then
                    count_value <= count_value + 1;
                else
                    count_value <= count_value - 1;
                end if;
            end if;

            -- Lógica del Pulsador SW (Reset del contador)
            if sw_sync(1) = '1' and sw_sync(0) = '0' then
                count_value <= (others => '0');
            end if;
        end if;
    end process;

    -- Asignación de la Salida
    cout <= std_logic_vector(count_value);
end architecture Behavioral;

```

Listing 11: Código Para lectura de encoder mecánico.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Trans is
Port (
    clk,rst: in STD_LOGIC;
    sw : in STD_LOGIC_VECTOR(7 downto 0);
    LDtx : out STD_LOGIC;
    tx : out STD_LOGIC
);
end Trans;

architecture Arq_Trans of Trans is

signal cnttx_us : UNSIGNED(15 downto 0):= (others => '0');
signal ttx_us     : UNSIGNED(7 downto 0) := (others => '0');

constant BAUDTX_C : STD_LOGIC_VECTOR(15 downto 0) := "0001010001011000"
; -- 5208 en binario
constant BAUDTX_US : UNSIGNED(15 downto 0) := to_unsigned(5207, 16); --
Para la comparación, 5208-1

```

```

-- Señales para el 'with select'
signal ttx : STD_LOGIC_VECTOR(7 downto 0);
begin

    ttx <= STD_LOGIC_VECTOR(ttx_us);

    -- Reloj de transmisión
    process (clk, rst)
    begin
        if (rst='1') then
            cnttx_us <= (others=>'0');
            ttx_us <= (others=>'0');
        elsif (rising_edge(clk)) then
            if (cnttx_us = BAUDTX_US) then -- si alcanzó 5208-1 = 5207
                ttx_us <= ttx_us + 1; -- contador para el transmisor
                cnttx_us <= (others=>'0');
            else
                cnttx_us <= cnttx_us + 1; -- contador del TX
            end if;
        end if;
    end process;

    -- Protocolo de transmisión
    with ttx select
    tx <= '1' when X"00",
    '0' when X"01", -- bit de start
    sw(0) when X"02",
    sw(1) when X"03",
    sw(2) when X"04",
    sw(3) when X"05",
    sw(4) when X"06",
    sw(5) when X"07",
    sw(6) when X"08",
    sw(7) when X"09",
    '1' when X"0A", -- bit de stop
    '1' when others;

    -- Protocolo de transmisión led testigo
    with ttx select
    LDtx <= '1' when X"00",
    '0' when X"01", -- bit de start
    sw(0) when X"02",
    sw(1) when X"03",
    sw(2) when X"04",
    sw(3) when X"05",
    sw(4) when X"06",
    sw(5) when X"07",
    sw(6) when X"08",
    sw(7) when X"09",
    '1' when X"0A", -- bit de stop
    '0' when others;

end Arq_Trans;

```

Listing 12: Código Para transmitir datos.

```
library IEEE;
```

```

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Recep is
Port (
    clk,rst,rx: in std_logic;
    leds: out std_logic_vector(7 downto 0)
);
end Recep;

architecture Arq_Recep of Recep is
    signal erx: std_logic:='0';
    signal trx_us: UNSIGNED(4 downto 0) := (others => '0');
    signal regrx: std_logic_vector(7 downto 0):="00000000";

    signal cntrx_us: UNSIGNED(10 downto 0) := (others => '0');

    constant BAUD_3X_C : STD_LOGIC_VECTOR(10 downto 0):="11011001000"; --
1736 en binario
    constant BAUD_3X_US : UNSIGNED(10 downto 0) := to_unsigned(1735, 11); --
1736-1 para la comparación

    signal trx: std_logic_vector(4 downto 0); -- Se mantiene para las
condiciones de captura
begin
    -- Asignación para las condiciones de captura
    trx <= STD_LOGIC_VECTOR(trx_us);

    -- Habilita la recepción al detectar el bit de inicio
process (rst, rx)
begin
    if rst='1' then
        erx <= '0';
    elsif rx='0' and erx = '0' then
        erx <= '1';
    elsif trx_us = to_unsigned(27, 5) then
        erx <= '0';
    else null;
    end if;
end process;

    -- Conteo para el tiempo del receptor
process (clk, rst)
begin
    if (rst='1' or erx='0') then
        cntrx_us <= (others=>'0');
        trx_us <= (others=>'0');
    elsif (rising_edge(clk) and erx='1') then
        if (cntrx_us = BAUD_3X_US) then
            trx_us <= trx_us + 1;
            cntrx_us <= (others=>'0');
        else
            cntrx_us <= cntrx_us + 1;
        end if;
    end if;
end process;

```

```

end process;

-- Recibe los datos en el registro del receptor regrx
process (clk, rst)
begin
    if rst='1' then
        regrx <= (others=>'0');
        leds <= (others=>'0');
    elsif (rising_edge(clk)) then
        case (trx) is -- captura en el centro del bit:
            when "00100" => regrx(0) <= rx; -- 4 (inicio del bit 0)
            when "00111" => regrx(1) <= rx; -- 7
            when "01010" => regrx(2) <= rx; -- 10
            when "01101" => regrx(3) <= rx; -- 13
            when "10000" => regrx(4) <= rx; -- 16
            when "10011" => regrx(5) <= rx; -- 19
            when "10110" => regrx(6) <= rx; -- 22
            when "11001" => regrx(7) <= rx; -- 25
            when "11010" => leds <= regrx; -- 26 Carga el dato a leds
            when others => null;
        end case;
    end if;
end process;
end Arq_Recep;

```

Listing 13: Código Para recibir datos.

Fotos

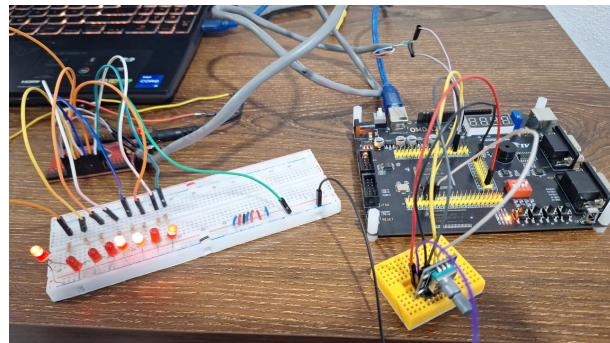


Figura 11: Comunicación con protocolo RS232

Conclusiones

Flores Oropeza Osvaldo

Durante la práctica note lo complejo que es diseñar etapas de potencia, ya sean de alimentación o de audio. Igualmente entendí la importancia de preparar con antelación los módulos necesarios para el correcto desarrollo de la práctica.

En la primera actividad tuve problemas al darle significado físico a las salidas del sistema, esto debido a que lo ideal es que las salidas sean consecuencias del sistema diseñado. Esto es muy útil a la hora de diseñar sistemas de control, lo que nos ayudará durante la carrera.

Frase: El **esfuerzo** sin objetividad y realidad no sirve.

Metáfora: La metáfora habla de aceptación, cada persona es quien es no debería intentar ser alguien más, pero si debería intentar ser mejor.

Ramírez Aguilar Víctor Saul

En esta práctica se vio a simplicidad que puede llegar a tener el realizar algunas funciones basadas en circuitos de lógica secuencial y combinatoria con el uso de lenguajes VHDL y verilog para la programación de tarjetas FPGA, ya que no es muy diferente a otros lenguajes de programación donde podemos tener funciones para realizar tareas específicas, o el uso de vectores para el uso de líneas de datos como fue el caso del registro de 8 bits donde la función se realizó en pocas líneas de código lo que en un diseño con lógica combinacional física pudo haber requerido el uso de más componentes.

Frase: El **esfuerzo** trae buenas recompensas

Lectura: Al ser estudiante de mecatronica creo que lo que quisiera transmitir al mundo sería compartir mi capacidad para tratar de resolver problemas de la vida cotidiana con soluciones tecnológicas.

Reyes Lira Alejandro

La práctica evidenció la capacidad y flexibilidad de los PLD para diseñar sistemas digitales diversos. Se inició con la descripción de una tabla de verdad en HDL, mostrando la eficiencia en la abstracción de lógica combinacional. Luego, se implementó un registro de 8 bits con flip-flops tipo D, destacando su importancia en el almacenamiento de datos. Posteriormente, se desarrolló un contador con salida a display de 7 segmentos, integrando conteo, decodificación e interfaz visual, esenciales en instrumentación. Finalmente, se aplicó el control de señales con la generación de sonidos de sirena, demostrando la interacción directa del PLD con el entorno. En conjunto, la práctica consolidó cómo estos dispositivos abarcan desde almacenamiento hasta generación de señales dinámicas, reafirmando su papel central en la tecnología digital.

Frase: Enfoca tu **energía**, no en luchar contra lo que fuiste, sino en construir la persona que quieras llegar a ser.

Metáfora: la metáfora nos enseña que la felicidad no proviene de lograr ser una buena imitación de otros, sino de un acto de valentía: mirar hacia nuestro interior, descubrir nuestra naturaleza única y abrazar nuestro propio y auténtico propósito, que es tan valioso como cualquier otro.

Referencias

- [1] D. Kokkinos and D. Kokkinos, “Learn How a 4-Digit 7-Segment LED Display Works and how to control it using an Arduino - Software Particles,” 10 2023. [Online]. Available: <https://softwareparticles.com/learn-how-a-4-digit-7-segment-led-display-works-and-how-to-control-it-using-an-arduino/>
- [2] Administrador, “Flip-flop tipo D. Descripción, símbolo, tabla de verdad - Electrónica Unicrom,” 8 2025. [Online]. Available: <https://unicrom.com/flip-flop-tipo-d-descripcion-y-simbolo/>
- [3] “Protocolo RS232: la guía para principiantes,” 7 2023. [Online]. Available: <https://opencircuit.es/blog/rs232-protocol-de-gids-voor-beginners>