



Universidad Nacional Autónoma de
México



Facultad de Ingeniería
Ingeniería en computación

Diseño Digital VLSI

Profesor: Gerardo Ariel Castillo García

Semestre 2025-1

Proyecto 1. Integración PWM y relojes para control de motores mediante ultrasónico

Grupo: 02 Brigada:05

Integrantes del Equipo:

- Bravo Luna Ivonne Monserrat
- Diaz González Rivas Ángel Iñaqui
- García Gallegos Samantha
- Siliano Haller Rodrigo

Índice o contenido

1. Objetivo.....	2
2. Metodología	2
3. Metas	5
4. Alcance.....	5
5. Introducción	7
6. Marco Teórico.....	8
6.1. Sensor ultrasónico.....	8
6.2. PWM.....	8
6.3. Servomotor	9
6.4. Tarjeta DE10-Lite.....	10
7. Frontera.....	11
8. Desarrollo	12
8.1. Requerimientos	12
8.2. Especificaciones.....	14
8.2.1. Especificaciones técnicas.....	14
8.2.2. Especificaciones medibles	15
8.3. Modelos y algoritmos.....	17
8.3.1. Máquina de estados	17
8.4. Generación del código VHDL	18
8.4.1. Código PWM	18
8.4.2. Código divf	21
8.4.3. Código para el movimiento	23
8.4.4. Código para la señal	26
8.4.5. Código sensor ultrasónico	28
8.5. Pines utilizados en Quartus.....	32
9. Resultados	32
10. Conclusiones.....	34
11. Referencias bibliográficas	28
12. Lista de Imágenes	39

1. Objetivo

El alumno aprenderá a desarrollar un sistema en VHDL para integrar un sensor ultrasónico que mida la distancia de un objeto, con visualización en tiempo real de la distancia en el display de la tarjeta D10Lite. Además, el sistema controlará la velocidad de un motor en función de la distancia: a mayor distancia entre el sensor y el objeto, la velocidad del motor se incrementará gradualmente; mientras que, a menor distancia, la velocidad se reducirá de forma proporcional.

2. Metodología

1. Se planificaron los objetivos específicos, las metas y el alcance del proyecto con el fin de asegurar la dirección que tendría el proyecto.

En esta planificación analizamos los requerimientos que veríamos en el desarrollo del proyecto, establecimos mediante cálculos y un análisis gráfico el alcance y recursos necesarios tanto del sensor, como del motor y de la tarjeta.

Así mismo determinamos un cronograma de actividades y delegamos las mismas en dos divisiones; el proyecto físico y la documentación del proyecto, con el fin de volver más eficiente la elaboración del trabajo y llegar al objetivo deseado.

2. Realización del análisis de los requerimientos del proyecto, en el que se establecen la distancia máxima, la cual se configura de forma dinámica por medio de un dipswitch en la FPGA



3. Elaboración y planteamiento del modelo matemático, con el fin de definir el comportamiento de la velocidad y el estado gráfico del proyecto.

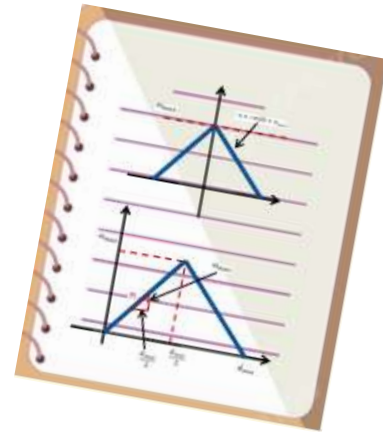
4. Se realiza la implementación en FPGA usando VHDL, en dónde inicialmente se configura el sensor ultrasónico, en el trigger el sensor se activa enviando un pulso de 10 microsegundos a la línea de este, mientras que la línea echo devuelve un pulso cuyo ancho es proporcional al tiempo que tarda el ultrasonido en regresar.

El t se toma como el tiempo del eco, en el que se utiliza un divisor de frecuencia en el FPGA con el fin de gestionar el tiempo sin la necesidad de otros factores.

También se establece la frecuencia base para la señal PWM, que sea muy común para los motores servo, así como también, se controla la velocidad del motor por medio del ajuste del ciclo de trabajo, el cual cambiará en tiempo real según la distancia detectada por el sensor.

5. Realización del código VHDL

Respecto al análisis tanto matemático como teórico de los requerimientos, especificaciones y modelos del proyecto se implementa el módulo de medición del sensor ultrasónico, el módulo de visualización en los



displays de 7 segmentos. El código se divide en bloques funcionales independientes para facilitar la depuración y pruebas modulares.

Las divisiones de los módulos principales son el código para el sensor ultrasónico, el módulo de control PWM y el módulo que nos permite la visualización en los displays de 7 segmentos.

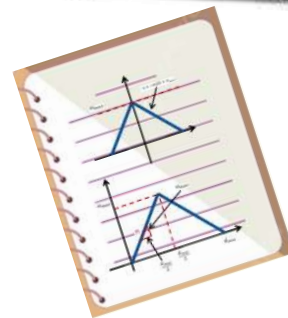
6. Realizar las pruebas para cada uno de los módulos implementados en la FPGA con el fin de garantizar que trabajen correctamente relacionados entre sí.

En el módulo del sensor ultrasónico se debe verificar que el sensor entregue los valores correctos o consistentes de tiempo eco, que representaría que la distancia está correctamente medida.

Para el módulo PWM se debe observar si el motor gira adecuadamente y a velocidades diferentes en función de la distancia medida, utilizando diferentes valores de ciclo de trabajo.

7. Realizar los ajustes finales del código VHDL con el fin de que se mejore la eficiencia del sistema, para ello se simplifica un poco el código y se describe a detalle la función de cada línea de este, así como también, si se detectan inestabilidades o incoherencias en el servomotor se realizarán correcciones.

8. Someter al proyecto físico a pruebas de mayor duración y con diferentes condiciones o distancias, esto con el fin de asegurar que no existe algún daño o falla.



3. Metas

Para lograr el objetivo anterior, se han establecido metas claras y un alcance definido que guiarán el desarrollo y la implementación del sistema.

1. Desarrollar un sistema de control automatizado utilizando una FPGA programada en VHDL que permita gestionar la medición y visualización de distancia en tiempo real, integrando un sensor ultrasónico. Este sistema proporcionará una solución efectiva para aplicaciones donde la medición precisa de distancias es crucial, como en robótica o automatización industrial.
2. Implementar un control dinámico y preciso de la velocidad de un motor en función de la distancia detectada, optimizando su respuesta en relación con la cercanía o lejanía del objeto. Esto permitirá que el motor ajuste su operación de manera eficiente, mejorando el rendimiento del sistema y adaptándose a diferentes escenarios de uso.
3. Establecer un sistema seguro y eficiente para la protección de los componentes, asegurando la estabilidad en el control del motor y evitando condiciones que puedan causar daños, como sobrecalentamiento o paradas bruscas. La implementación de medidas de seguridad contribuirá a la fiabilidad y durabilidad del sistema.



4. Alcance

El alcance de este proyecto se define claramente para garantizar que todos los componentes y funciones necesarias se integren de manera efectiva en el sistema. Este enfoque permite concentrarse en las funcionalidades esenciales que cumplen con las metas establecidas, al tiempo que se evitan desviaciones que podrían

complicar el desarrollo o extender innecesariamente el tiempo de implementación. A continuación, se detallan las áreas específicas que se abordarán dentro del alcance del proyecto.

1. El sistema capturará la distancia de un objeto mediante la emisión de ondas ultrasónicas, calculando el tiempo que tardan en regresar. Esta funcionalidad será fundamental para la operativa del sistema, garantizando mediciones precisas en diversas condiciones.
2. Se configurará el sistema para mostrar la distancia en centímetros en tiempo real en el display de la tarjeta D10Lite. Esta representación visual permitirá al usuario tener información inmediata sobre la distancia medida, facilitando la interacción con el sistema.
3. Se implementará el control de un motor a través de señales PWM, ajustando su velocidad de manera progresiva y proporcional según la distancia medida. Este método de control garantiza un funcionamiento suave y controlado del motor, evitando movimientos bruscos.
4. Se incluirá un mecanismo de retroalimentación visual adicional, como LEDs, que indicará el estado de distancia y velocidad del motor. Esto mejorará la experiencia del usuario al proporcionar información visual clara y accesible sobre el funcionamiento del sistema.
5. Se establecerán límites de velocidad máxima y se programará el sistema para evitar paradas completas o cambios bruscos. Estas medidas de seguridad son cruciales para proteger los componentes del sistema y asegurar un funcionamiento confiable y prolongado.



5. Introducción

El presente proyecto implementa un sistema de control automatizado basado en un sensor ultrasónico y un servomotor, utilizando la tarjeta FPGA DE10-Lite y codificación en VHDL. Este sistema simula a un robot que permite medir en tiempo real la distancia de un objeto mediante ondas ultrasónicas y visualizar dicha distancia en un display de 7 segmentos. Además de controlar la velocidad de un servomotor en función de la proximidad del objeto, en el cual se establece que a mayor distancia, la velocidad del motor aumentará progresivamente y a menor distancia, la velocidad disminuirá proporcionalmente hasta detenerse.

El control de la velocidad del motor se gestiona mediante señales PWM o modulación por ancho de pulso, las cuales son generadas desde la FPGA.

A medida que la distancia entre el sensor y el objeto varíe, la tarjeta FPGA ajustará el comportamiento del motor, mostrando además la distancia medida en los displays de siete segmentos. Además, se establecerán límites de velocidad y energía para proteger los componentes del sistema, con el fin de garantizar un buen funcionamiento.

Finalmente, los sistemas que integran sensores ultrasónicos y control de motores tienen diversas aplicaciones en áreas como la robótica para la navegación autónoma, vehículos de control remoto y automatización industrial para el manejo de materiales. La capacidad de adaptar la velocidad del motor en función de la distancia medida no solo mejora la eficacia operativa, sino que también contribuye a la seguridad y fiabilidad del sistema.



Imagen del proyecto físico, durante la prueba de funcionamiento.

6. Marco Teórico

6.1. Sensor Ultrasónico

Los sensores ultrasónicos son dispositivos que emplean ondas sonoras para determinar distancias. Funcionan emitiendo pulsos de sonido en la frecuencia ultrasónica, generalmente por encima de 20 kHz, y midiendo el tiempo que tardan en regresar tras rebotar en un objeto. Así mismo, es capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 2 a 450 cm. El funcionamiento de los sensores mediante ultrasonido y cuenta con la electrónica encargada de realizar la medición, debido a sus pines de ECHO y TRIGGER.

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. Su funcionamiento consiste con el emisor piezoeléctrico cuando emite 8 pulsos de ultrasonido y luego de recibir la orden en el pin



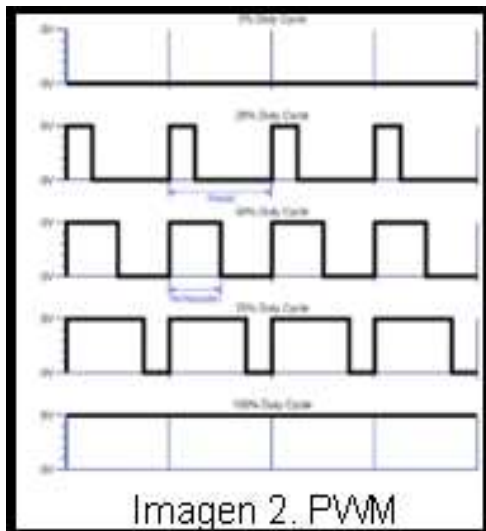
Imagen 1. Sensor ultrasónico

TRIG, las ondas de sonido viajan en el aire y rebotan al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico, luego el pin ECHO cambia a Alto por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto.

6.2. PWM

PWM son las siglas de Pulse Width Modulation o en su traducción al español, Modulación por ancho de pulso. Para transmitir una señal, ya sea analógica o digital, se debe modular para que sea transmitida sin perder potencia o sufrir distorsión por interferencias.

PWM es una técnica que se usa para transmitir señales analógicas cuya señal portadora será digital. En esta técnica se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga. El ciclo de trabajo de una señal periódica es el ancho de su parte positiva, en relación con el período. Está expresado en porcentaje, por tanto, un duty cycle de 10% indica que está 10 de 100 a nivel alto.



ciclo de 10% indica que está 10 de 100 a nivel alto.

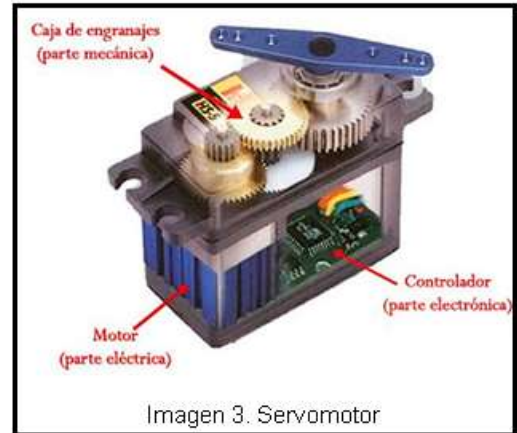
Esta modulación es muy usada para controlar la cantidad de energía que se envía a una carga, es una técnica utilizada para regular la velocidad de giro de los motores, regulación de intensidad luminosa, controles de elementos termoelectrónicos o controlar fuentes conmutadas, entre otros usos.

6.3. Servomotor

Un servomotor es un actuador rotativo o lineal que permite lograr un control preciso en cuanto a posición angular, aceleración y velocidad del eje, capacidades que un motor normal no tiene. Está conformado por un motor eléctrico convencional que a su vez integra un sistema de retroalimentación.

Un servomotor es básicamente un tipo de motor eléctrico de alto rendimiento, cuyo sistema convierte la electricidad en un movimiento controlado con gran precisión. El servomotor es considerado el esclavo del servodrive debido a que le informa y ejecuta las órdenes que este proporciona acerca del eje.

Del mismo modo, un servomotor es un motor altamente especializado diseñado para el control preciso de movimientos rotativos o lineales. Se trata de un motor de rotación o traslación que emplea un mecanismo de realimentación para garantizar un posicionamiento exacto, normalmente mediante una señal de control que dicta el movimiento del motor hasta una posición deseada.



6.4. Tarjeta DE10-Lite

De10 Lite es una placa FPGA que utiliza la capacidad máxima de FPGA MAX 10, proporciona la solución perfecta de creación de prototipos a nivel de sistema para aplicaciones industriales, automotrices, de consumo y muchas otras aplicaciones del mercado.

Esta tarjeta destaca por combinar potencia de procesamiento, flexibilidad y un



conjunto diverso de periféricos, permitiendo que los usuarios diseñen e implementen sistemas digitales personalizados. Su uso es ideal para aplicaciones como control de motores, sistemas de sensores, procesamiento de señales, e interfaces de usuario, gracias a los displays de 7 segmentos, interruptores, y LEDs integrados.

7. Frontera

La frontera del presente proyecto delimita algunos elementos considerados incluidos y excluidos, con el fin de garantizar un desarrollo enfocado, adecuado y acorde a los objetivos planteados en un principio.

El proyecto incluye:

1. La lectura de la distancia mediante el sensor ultrasónico, que incluye la emisión y la recepción de ondas ultrasónicas.
2. La visualización de la distancia medida en centímetros en los displays de la tarjeta DE10-Lite.
3. El control de velocidad del motor mediante PWM, que incluye el ajuste de velocidad del motor respecto a la distancia medida y la señal PWM generada en ciclos de trabajo que varían gradualmente.

El proyecto no incluye:

1. El sistema no podrá interaccionar con otros sistemas externos, pues el sistema será autónomo.
2. El proyecto únicamente se centrará en el control de un solo motor y de un solo sensor ultrasónico y no soportará más de uno en cada caso.
3. Solo se implementará un control básico proporcional de la velocidad.
4. La interacción del proyecto se limitará a los displays de 7 segmentos presentes en la tarjeta DE10-Lite.
5. En la integración de funciones adicionales estará presente el ajuste dinámico de velocidad basado en la medición de la distancia de un único objeto.

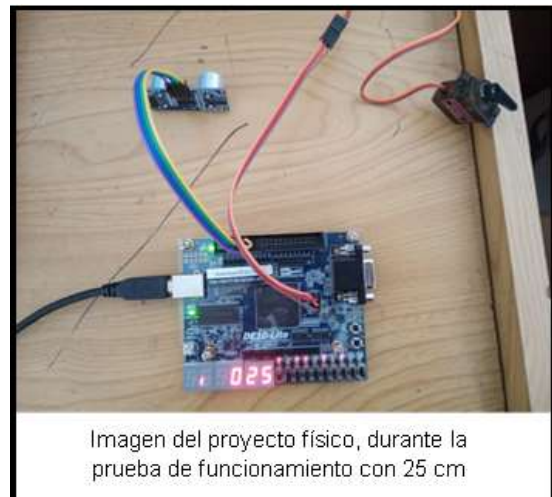
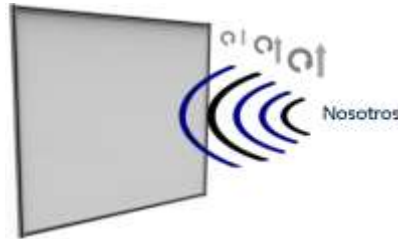


Imagen del proyecto físico, durante la prueba de funcionamiento con 25 cm

8. Desarrollo

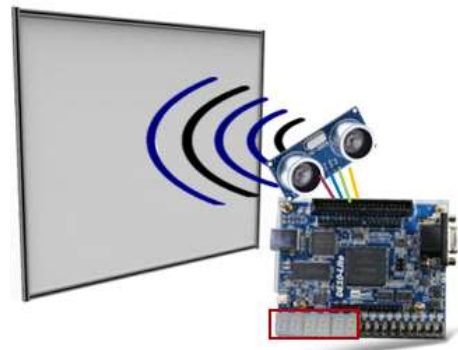
8.1. Requerimientos

1. El sistema va a emplear un sensor ultrasónico con el fin de medir la distancia entre el objeto detectado y el sensor en tiempo real.



2. La distancia máxima que se va a permitir va a ser definida en el tiempo de compilación y podrá configurarse de forma dinámica mediante el dipswitch de la tarjeta FPGA.

3. El proyecto físico va a ser capaz de poder actualizar de forma continua las distancias detectadas y las mostrará en los displays de 7 segmentos de la tarjeta DE10-Lite.



4. La velocidad del servomotor se representará por la letra griega omega (ω) y será directamente proporcional a la distancia detectada por el sensor.

mayor distancia = mayor velocidad del servomotor
menor distancia = menor velocidad del servomotor

Cálculo de ω_{motor}

$$\omega_{motor} = k_1 * \frac{V_{cc}}{T_{motor}}$$

$$\omega_{motor} = 1 * \frac{5 [V]}{1 [s]} = 5 \text{ rad/s}$$

Cálculo de d

$$d = \frac{V_{us\ echo}}{2}$$

$$d = \frac{340 [\frac{m}{s}]}{2} = 170 [m]$$

El tiempo de encendido es t_{on}

$t_{on} = cont_{on}$ (que pertenece a 0 ... 100)

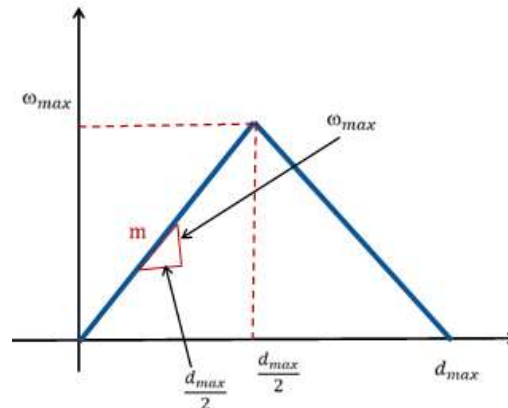
$cont_{echo}$ (que pertenece a 0)

La velocidad del servomotor se ajustará en función a la distancia medida

Si $d > \frac{d_{max}}{2}$ ω_{motor} = La función se incrementa

Si $d \leq \frac{d_{max}}{2}$ ω_{motor} = La función disminuye

5. El sistema va a permitir que el motor comience en reposo, es decir en la posición cero (0) y aumente su velocidad gradualmente conforme el objeto se acerque o se aleje, al punto de alcanzar su velocidad máxima a la mitad de la distancia máxima configurada



6. La distancia máxima definida como d_{max} se va a definir como una constante basada en el tiempo de compilación menos el tiempo de ejecución y se podrá ajustar mediante un dipswitch en la tarjeta FPGA.

$$V - V_{\max} = C(d_{\max} - d)$$



La velocidad del servomotor se ajusta mediante un ciclo de trabajo de la señal PWM

$$PWM = \frac{t_{on}}{T} * 100$$

8.2. Especificaciones

En esta sección, explicaremos los elementos clave necesarios para alcanzar nuestro objetivo y estableceremos una guía con criterios que faciliten el seguimiento del progreso y el cumplimiento del proyecto.

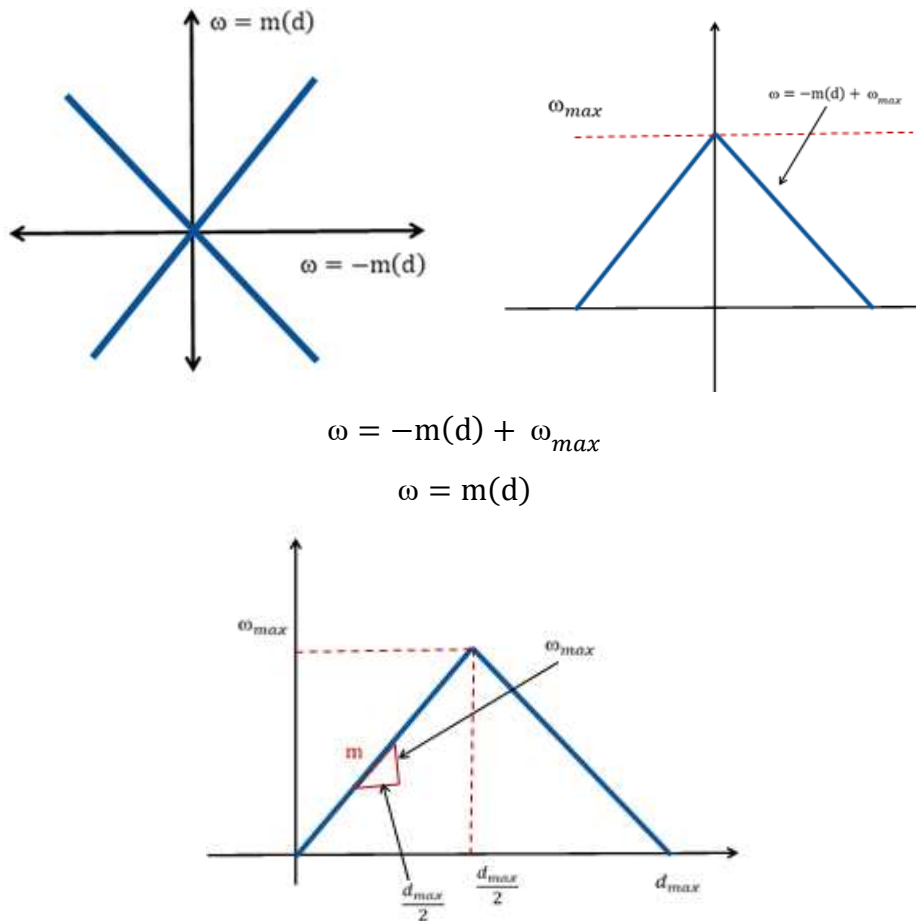
8.2.1. Especificaciones técnicas

Material	Características
<p>1. Tarjeta D10-Lite</p> 	<p>Pantalla integrada para la visualización de la distancia medida.</p> <p>Capacidad para la generación de señales PWM para el control del servomotor y el sensor ultrasónico.</p> <p>Entrada y salida digitales compatibles con el sensor HC-SR04.</p>
<p>2. Servomotor MG60</p> 	<p>Rango de operación de voltaje 5-12</p>

<p>3. Sensor ultrasónico <i>HC-SR04</i></p> 	<p>Rango de medición: 2 cm a 400 cm. Precisión: ± 0.3 cm. Frecuencia ultrasónica: 40 kHz. Tiempo de respuesta: Menor a 15 ms. Alimentación: 5V, que se toma directamente de la tarjeta D10Lite. Comunicación: Dos pines de control (Trigger y Echo) para la medición de la distancia.</p>
---	---

8.2.2. Especificaciones medibles

<p>1. El sistema debe monitorear constantemente la distancia entre el sensor y el objeto dentro de su rango de operación, ajustando la velocidad del servomotor según sea necesario.</p>	<p>-Rango de operación del sensor: 0 a 100 cm -Actualización mínima de medición: 2Hz -Precisión del sensor 1 cm</p>
<p>2. La medición de la distancia debe ser precisa y en tiempo real, permitiendo que cualquier cambio se refleje inmediatamente en el comportamiento del servomotor.</p>	<p>-Tiempo de procesamiento: 0.5s -Actualización de la visualización en pantalla: 1s como máx</p>
<p>3. La velocidad del servomotor debe variar de forma gradual y suave conforme a la distancia medida, evitando movimientos bruscos o repentinos entre distintos niveles de distancia.</p>	<p>-Tasa de medición continua: 2Hz -Ajuste de velocidad en respuesta a cambios de distancia menor a 0.5s</p>
<p>4. Es esencial que el sistema no permita que el motor se detenga por completo.</p>	<p>-Transición en línea recta entre velocidades</p>



$$\omega = -m(d) + \omega_{max}$$

$$\omega = m(d)$$

5. El sistema debe incluir un mecanismo de retroalimentación visual, como un LED o una pantalla, para mostrar las lecturas de distancia al usuario.

-Actualización de los displays de 7 segmentos cada 0.5s

6. Para asegurar un funcionamiento seguro, el sistema debe evitar que el motor se sobrecaliente o se dañe, independientemente de las variaciones de distancia.

-Temperatura máxima motor: 40°C
-Protección contra corriente
-Límite de corriente de 2A

7. La velocidad del servomotor debe controlarse de acuerdo con la distancia

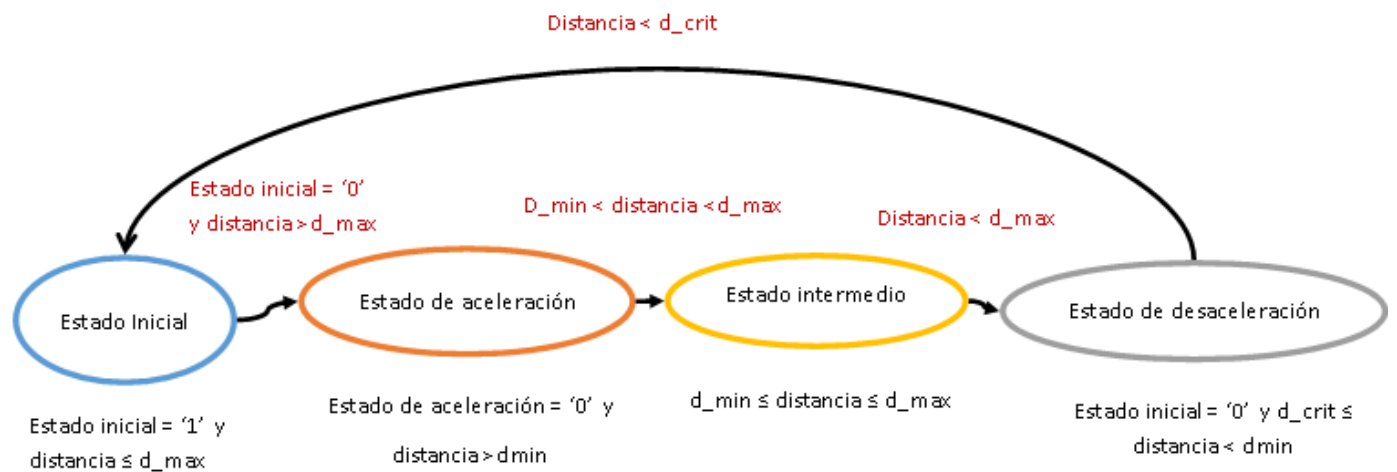
-Niveles de velocidad
Distancia corta: 0-65 cm
Distancia larga: 65-100cm

detectada, asegurando una respuesta progresiva y proporcional.

Distancia máx: 100cm
Tiempo de cambio de velocidad: 0.5s

8.3. Modelos y algoritmos

8.3.1. Máquina de estados



Estado	Condición	Siguiente estado
Estado Inicial	La distancia es mayor que un umbral máximo (distancia $> d_{\max}$). Se requiere que el sistema acelere porque el objeto está lejos del punto objetivo.	Estado de aceleración
Estado de aceleración	La distancia ha entrado en un rango normal seguro	Estado Intermedio

	$(d_{min} < distancia < d_{max})$ El sistema deja de acelerar y entra en un estado intermedio donde se mantiene estable.	
Estado Intermedio	La distancia es menor que un umbral mínimo ($distancia < d_{min}$). El objeto está demasiado cerca, por lo que es necesario reducir la velocidad para evitar una colisión.	Estado de desaceleración
Estado de desaceleración	La distancia es crítica ($distancia < d_{crit}$) o se recibe una señal de reinicio ($rst = 1$). El sistema necesita reiniciar porque el objeto está peligrosamente cerca o se ha forzado un reinicio.	Estado Inicial

8.4. Generación del código VHDL

8.4.1. Código PWM

El código PWM en VHDL implementa un sistema de control PWM para gestionar la velocidad de un motor en función de la distancia medida por un sensor ultrasónico. El sistema se divide en varios módulos: *divf* reduce la frecuencia del reloj para sincronizar el resto de las operaciones; *senal* controla la señal PWM en función del duty cycle (relación de

encendido/apagado del motor); sonicos gestiona el sensor ultrasónico, calculando la distancia y mostrando la información en los displays de unidades, decenas y centenas. Además, se utilizan LEDs para indicar si el objeto está a 20 cm o 5 cm del sensor. Finalmente, movimiento controla el motor ajustando el duty cycle según la distancia medida, permitiendo que la velocidad varíe de acuerdo con la cercanía del objeto detectado

```

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 PWM is
7  port (
8      clk, pi, pf, rst, clkU, sensor_eco : in std_logic;
9      displayunidades: out std_logic_vector(6 downto 0);
10     displaydecenas: out std_logic_vector(6 downto 0);
11     displaycentenas: out std_logic_vector(6 downto 0);
12     centimetros_out: out unsigned(15 downto 0);
13     control, sensor_disp, led_20cm, led_5cm : out std_logic
14 );
15 end entity;
16
17 architecture Behavioral of PWM is
18     signal clk1: std_logic;
19     signal duty : integer range 0 to 200 := 85;
20     signal centimetros_sig: std_logic_vector(15 downto 0);
21
22     -- Señales intermedias para los displays
23     signal displayunidades_sig: std_logic_vector(6 downto 0);
24     signal displaydecenas_sig: std_logic_vector(6 downto 0);
25     signal displaycentenas_sig: std_logic_vector(6 downto 0);
26
27 begin
28     u1: entity work.divf(arqdivf)
29         generic map(500)
30         port map (clk => clk, clk1 => clk1);
31
32     u2: entity work.senal(arqsnl)
33         port map (clk1, duty, control);
34
35     u3: entity work.sonicos(arqsonicos)
36         port map(
37             clkU => clkU,
38             sensor_disp => sensor_disp,
39             sensor_eco => sensor_eco,
40             led_20cm => led_20cm,
41             led_5cm => led_5cm,
42             displayunidades => displayunidades_sig,
43             displaydecenas => displaydecenas_sig,
44             displaycentenas => displaycentenas_sig,
45             centimetros_out => centimetros_sig
46         );
47
48     -- Asignación de las señales intermedias a las salidas de la entidad
49     displayunidades <= displayunidades_sig;
50     displaydecenas <= displaydecenas_sig;
51     displaycentenas <= displaycentenas_sig;
52
53     u4: entity work.movimiento(arqmov)
54         port map(clk1, pi, pf, rst, centimetros_sig, duty);
55
56 end Behavioral;

```

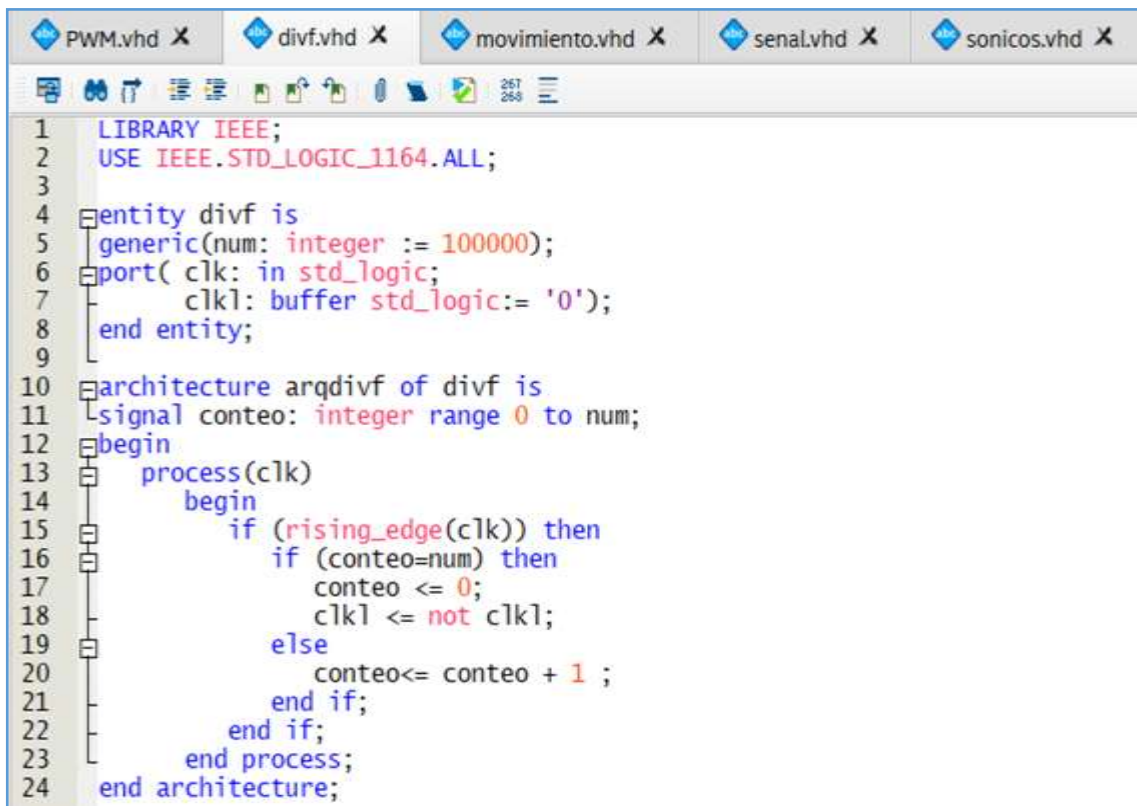
```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL; --Proporciona tipos y operaciones lógicas estándar
3  use IEEE.STD_LOGIC_ARITH.ALL; --Proporcionan soporte para operaciones aritméticas y de comparaciones sin signo.
4  use IEEE.STD_LOGIC_UNSIGNED.ALL; --Proporcionan soporte para operaciones aritméticas y de comparaciones sin signo.
5
6  entity PWM is
7  port ( --Inicia la declaración de puertos (entradas y salidas).
8        clk, pi, pf, rst, clkU, sensor_eco : in std_logic; --clk: Señal de reloj principal.
9                                                    --pi, pf: Señales de control de posición
10                                                    --rst: Señal de reinicio para el sistema.
11                                                    --clkU: Señal de reloj para el sensor ultrasónico.
12                                                    --sensor_eco: Señal de eco recibida del sensor ultrasónico.
13        displayunidades: out std_logic_vector(6 downto 0); --Salida de 7 bits para el display de las unidades.
14        displaydecenas: out std_logic_vector(6 downto 0); --Salida de 7 bits para el display de las decenas.
15        displaycentenas: out std_logic_vector(6 downto 0); --Salida de 7 bits para el display de las centenas.
16        centimetros_out: out unsigned(15 downto 0); --Salida de 16 bits para la distancia medida en centímetros, en formato sin signo.
17        control, sensor_disp, led_20cm, led_5cm : out std_logic --control: Señal de control general.
18                                                    --sensor_disp: Señal de activación del sensor ultrasónico.
19                                                    --led_20cm: LED que indica si la distancia es menor o igual a 20 cm.
20                                                    --led_5cm: LED que indica si la distancia es menor o igual a 5 cm.
21
22    );
23  end entity;
24
25  architecture Behavioral of PWM is
26    signal clk1: std_logic; --Señal de reloj generada por un divisor de frecuencia (divf).
27    signal duty : integer range 0 to 200 := 85; --Señal duty, usada para controlar el ciclo de trabajo de la señal PWM, con valor inicial de 85.
28    signal centimetros_sig: std_logic_vector(15 downto 0); -- Señal intermedia que almacena la distancia en centímetros (16 bits).
29
30  begin
31    u1: entity work.divf(arqdivf) --Instancia el componente divf (divisor de frecuencia).
32    generic map(500) --Configura un parámetro genérico con el valor 500 (ajusta la frecuencia de clk1).
33    port map (clk => clk, clk1 => clk1); --Mapea el reloj de entrada (clk) a la señal de salida de reloj dividida (clk1).
34
35    u2: entity work.senal(arqsnl) -- Instancia el componente senal (generador de señal PWM).
36    port map (clk1, duty, control); --Mapea las señales de entrada clk1 y duty, y la señal de salida control.
37
38    u3: entity work.sonicos(arqsonicos) --Instancia el componente sonicos para gestionar el sensor ultrasónico.
39    port map( --Mapea las señales de entrada y salida del sensor:
40        clkU => clkU, --Reloj para el sensor.
41        sensor_disp => sensor_disp, --Señal de activación del sensor.
42        sensor_eco => sensor_eco, --Señal de eco del sensor.
43        led_20cm => led_20cm, --Salidas a los LEDs.
44        led_5cm => led_5cm, --Salidas a los LEDs.
45        displaydecenas => displaydecenas_sig, --Salidas intermedias a los displays.
46        displaycentenas => displaycentenas_sig, --Salidas intermedias a los displays.
47        centimetros_out => centimetros_sig
48    );
49
50    -- Asignación de las señales intermedias a las salidas de la entidad
51    displayunidades <= displayunidades_sig;
52    displaydecenas <= displaydecenas_sig;
53    displaycentenas <= displaycentenas_sig;
54
55    u4: entity work.movimiento(arqmov) --Instancia el componente movimiento.
56    port map(clk1, pi, pf, rst, centimetros_sig, duty); --Mapea las señales de entrada clk1, pi, pf, rst y la señal intermedia centimetros_sig
57                                                    --al componente movimiento, que controla el valor de duty en función de la distancia
58                                                    --detectada.
59
60  end Behavioral;

```


8.4.2. Código *divf*

El código **divf** en VHDL define un módulo de división de frecuencia llamado **divf**, que toma una señal de reloj de entrada (**clk**) y genera una señal de reloj de menor frecuencia (**ckl**). La frecuencia de salida se determina por el valor del parámetro genérico **num**, que establece el número de ciclos de **clk** necesarios para que **ckl** cambie de estado (de 0 a 1 o de 1 a 0). Dentro del proceso, cada vez que **clk** detecta un flanco ascendente, el contador (**conteo**) se incrementa; cuando alcanza el valor de **num**, se restablece a 0 y se invierte **ckl**, produciendo así una señal de reloj con una frecuencia más baja, útil para sincronizar módulos que requieren operaciones a menor velocidad.



```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  entity divf is
5    generic(num: integer := 100000);
6    port( clk: in std_logic;
7          ckl: buffer std_logic:= '0');
8  end entity;
9
10 architecture arqdivf of divf is
11   signal conteo: integer range 0 to num;
12 begin
13   process(clk)
14   begin
15     if (rising_edge(clk)) then
16       if (conteo=num) then
17         conteo <= 0;
18         ckl <= not ckl;
19       else
20         conteo<= conteo + 1 ;
21       end if;
22     end if;
23   end process;
24 end architecture;
```



```
PWM.vhd X divf.vhd X movimiento.vhd X senal.vhd X sonicos.vhd X
1  LIBRARY IEEE; --Importa la biblioteca IEEE, que contiene definiciones estándar utilizadas en VHDL.
2  USE IEEE.STD_LOGIC_1164.ALL; --Importa el paquete STD_LOGIC_1164, que proporciona tipos de datos como std_logic y std_logic_vector,
3  --esenciales para la manipulación de señales lógicas.
4
5  entity divf is -- Declara la entidad divf, que es el módulo divisor de frecuencia.
6  generic(num: integer := 100000); --Define un parámetro genérico num, que es el valor de cuenta máxima para dividir la frecuencia del reloj.
7  --Su valor predeterminado es 100,000.
8  port( clk: in std_logic; --Define clk como una señal de entrada de tipo std_logic. Representa el reloj de entrada que se desea dividir.
9  clk1: buffer std_logic:= '0'); --Define clk1 como una señal de tipo std_logic de tipo buffer que actúa como reloj dividido, con un valor
10 --inicial de '0'.
11 end entity; -- Finaliza la declaración de la entidad divf.
12
13 architecture arqdivf of divf is --Inicia la arquitectura arqdivf para la entidad divf.
14 signal conteo: integer range 0 to num; --Declara una señal llamada conteo de tipo integer, con un rango de valores de 0 a num. Esta señal actúa
15 --como contador para dividir la frecuencia del reloj de entrada.
16 begin
17 process(clk) --Inicia un proceso que se ejecuta en cada flanco de subida de clk.
18 begin
19 if (rising_edge(clk)) then --Comprueba si hay un flanco de subida en la señal de reloj clk.
20 if (conteo=num) then --Si el valor del contador conteo ha alcanzado el valor num, realiza las siguientes acciones:
21 conteo <= 0; --Reinicia el contador conteo a 0.
22 clk1 <= not clk1; --Cambia el estado de clk1 (se invierte). Esto divide la frecuencia del reloj clk porque clk1 cambia de estado
23 --solo cuando conteo alcanza el valor máximo num.
24 else
25 conteo<= conteo + 1 ; --Incrementa el valor de conteo en 1.
26 end if;
27 end if;
28 end process;
29 end architecture;
```

8.4.3. Código para el movimiento

El código **movimiento** en VHDL implementa un módulo de división de frecuencia llamado **divf**. A partir de una señal de reloj de entrada (**clk**), genera una señal de reloj de menor frecuencia (**ckl**). El parámetro **num** define cuántos ciclos de **clk** se necesitan para que **ckl** cambie de estado (de 0 a 1 o de 1 a 0), lo que disminuye la frecuencia de salida. En cada flanco ascendente de **clk**, el contador (**conteo**) se incrementa; cuando llega al valor de **num**, el contador se reinicia y se invierte **ckl**, generando así un reloj más lento que sincroniza módulos que operan a menor velocidad.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity movimiento is
6 port(
7     clk, pi, pf, rst: in std_logic;
8     centimetros_in: in std_logic_vector(15 downto 0); -- New input for the distance from 'sonicos'
9     ancho: out integer
10 );
11 end entity;
12
13 architecture arquiv of movimiento is
14     signal valor: integer range 0 to 200 := 75;
15     signal counter: integer range 0 to 50000 := 0; -- Contador para generar el retardo
16     signal enable: std_logic := '0'; -- Señal de habilitación para controlar el incremento/decremento
17     signal direccion: std_logic := '1'; -- Señal para controlar la dirección: '1' para incrementar, '0' para decrementar
18     signal counter_multiplier: integer := 0; -- Multiplier for the counter
19 begin
20     process (clk, rst)
21         variable distancia: integer;
22     begin
23         if rst = '1' then
24             valor <= 75;
25             counter <= 0;
26             enable <= '0';
27             direccion <= '1'; -- Inicialmente configurado para incrementar
28             counter_multiplier <= 500;
29         elsif rising_edge(clk) then
30             distancia := to_integer(unsigned(centimetros_in));
31             -- Convert and use a temporary variable for the calculation
32             distancia := to_integer(unsigned(centimetros_in));
33
34             -- Calculate the counter_multiplier based on temp_distancia
35             if distancia < 90 then
36                 distancia := 100 - distancia;
37                 counter_multiplier <= distancia * 50;
38             else
39                 counter_multiplier <= 500;
40             end if;
41
42             -- Control the increment/decrement
43             if counter = counter_multiplier then -- Use counter_multiplier here
44                 enable <= '1';
45                 counter <= 0;
46             else
47                 counter <= counter + 1;
48                 enable <= '0';
49             end if;
50
51             if enable = '1' then
52                 if pi = '1' then
53                     valor <= 55; -- 5.5% ~1 ms
54                     direccion <= '1'; -- Cambia dirección a incrementar
55                 elsif pf = '1' then
56                     valor <= 95; -- 9.5% ~2ms
57                     direccion <= '0'; -- Cambia dirección a decrementar
58                 elsif direccion = '1' and valor < 95 then
59                     valor <= valor + 1;
60                     if valor = 94 then
61                         direccion <= '0'; -- Cambia dirección a decrementar
62                     end if;
63                 elsif direccion = '0' and valor > 55 then
64                     valor <= valor - 1;
65                     if valor = 56 then
66                         direccion <= '1'; -- Cambia dirección a incrementar
67                     end if;
68                 end if;
69             end if;
70         end if;
71     end process;
72
73     ancho <= valor; -- Asignación continua de 'ancho' al valor calculado
74 end architecture;
```

```

1 library ieee; -- Importa la biblioteca IEEE, que contiene las definiciones estándar utilizadas en VHDL.
2 use ieee.std_logic_1164.all; -- Importa el paquete STD_LOGIC_1164, que proporciona tipos de datos como std_logic.
3 use ieee.numeric_std.all; -- Importa numeric_std, que ofrece operaciones para tipos aritméticos como integer y unsigned, necesarios para manejar conv
4
5 entity movimiento is -- Declara la entidad movimiento, que ajusta el ancho de una señal PWM en función de la distancia (centímetros_in) y
6     -- otras entradas de control.
7     port(
8         clk, pi, pf, rst: in std_logic; -- clk, pi, pf, rst: in std_logic; Define las señales de entrada clk (reloj), pi, pf, y rst (reinicio),
9         -- todas de tipo std_logic.
10        centímetros_in: in std_logic_vector(15 downto 0); -- New input for the distance from 'sonicos'
11        ancho: out integer -- Señal de salida de tipo integer que representa el ancho de la señal PWM.
12    );
13 end entity;
14
15 architecture argmov of movimiento is -- Define la arquitectura argmov para movimiento
16     signal valor: integer range 0 to 200 := 75; -- Define valor como un integer dentro del rango 0 a 200, que inicialmente es 75.
17     -- Este representa el ancho del pulso PWM.
18     signal counter: integer range 0 to 50000 := 0; -- Contador para generar el retardo
19     signal enable: std_logic := '0'; -- Señal de habilitación para controlar el incremento/decremento
20     signal direccion: std_logic := '1'; -- Señal para controlar la dirección: '1' para incrementar, '0' para decrementar
21     signal counter_multiplier: integer := 0; -- Multiplier for the counter
22 begin
23     process (clk, rst) -- Inicia un proceso controlado por clk y rst.
24         variable distancia: integer; -- Declara una variable distancia para almacenar la distancia convertida de centímetros_in.
25     begin
26         if rst = '1' then -- Si rst está activado (en '1'), reinicia el sistema
27             valor <= 75; -- Establece valor en 75.
28             counter <= 0; -- Reinicia el contador.
29             enable <= '0'; -- Deshabilita cambios.
30             direccion <= '1'; -- Inicialmente configurado para incrementar
31             counter_multiplier <= 500; -- Establece el multiplicador predeterminado en 500.
32         elsif rising_edge(clk) then
33             distancia := to_integer(unsigned(centímetros_in));
34             -- Convert and use a temporary variable for the calculation
35             distancia := to_integer(unsigned(centímetros_in));
36
37             -- Calculate the counter_multiplier based on temp_distancia
38             if distancia < 90 then -- Si distancia es menor que 90, ajusta el multiplicador de acuerdo con la proximidad (a menor distancia,
39                 -- mayor multiplicador para incrementar la frecuencia de ajuste de valor).
40                 distancia := 100 - distancia;
41                 counter_multiplier <= distancia * 50;
42             else
43                 counter_multiplier <= 500;
44             end if;
45
46             -- Control the increment/decrement
47             if counter = counter_multiplier then -- Si counter alcanza counter_multiplier, activa enable y reinicia el contador.
48                 enable <= '1';
49                 counter <= 0;
50             else -- Si counter no alcanza counter_multiplier, incrementa el contador y desactiva enable.
51                 counter <= counter + 1;
52                 enable <= '0';
53             end if;
54
55             if enable = '1' then
56                 if pi = '1' then
57                     valor <= 55; -- 5.5% ~1 ms
58                     direccion <= '1'; -- Cambia dirección a incrementar
59                 elsif pf = '1' then
60                     valor <= 95; -- 9.5% ~2ms
61                     direccion <= '0'; -- Cambia dirección a decrementar
62                 elsif direccion = '1' and valor < 95 then
63                     valor <= valor + 1;
64                     if valor = 94 then
65                         direccion <= '0'; -- Cambia dirección a decrementar
66

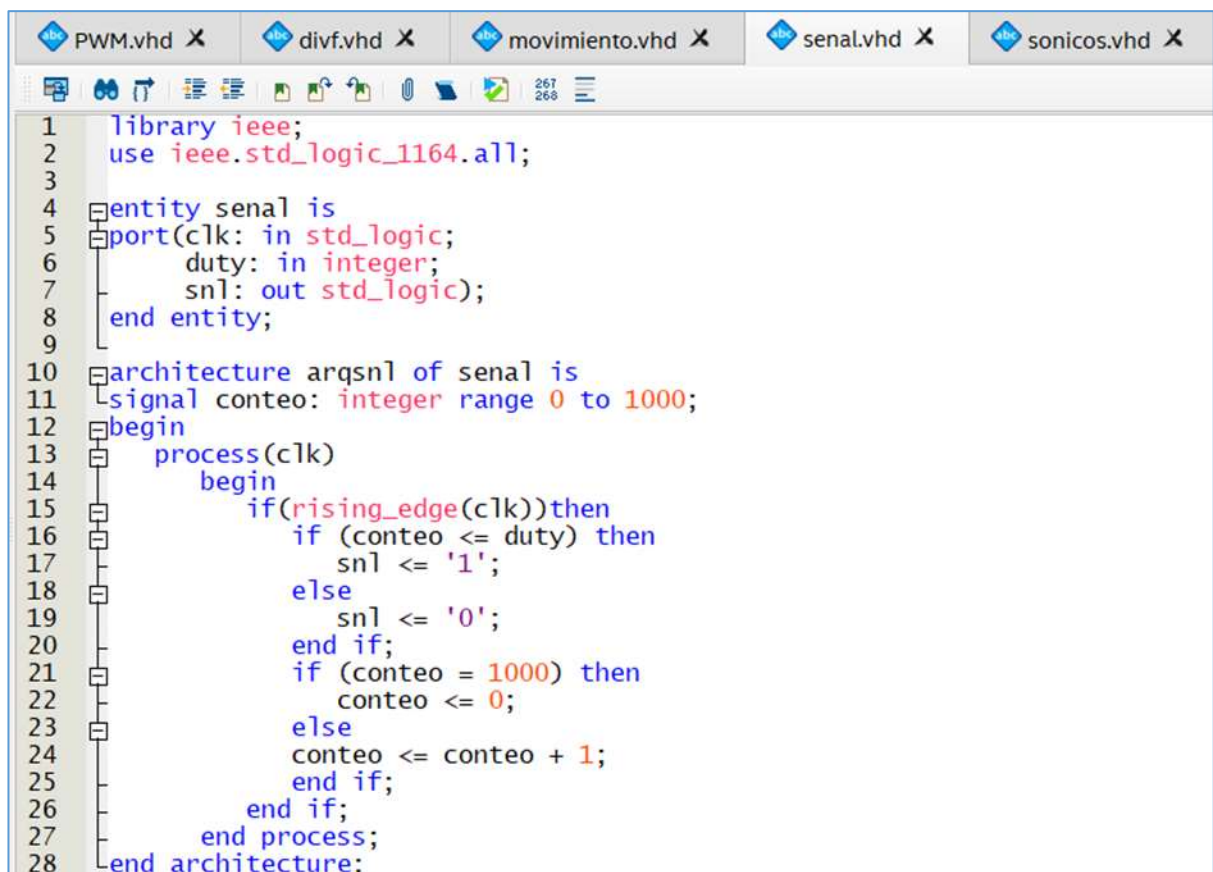
```



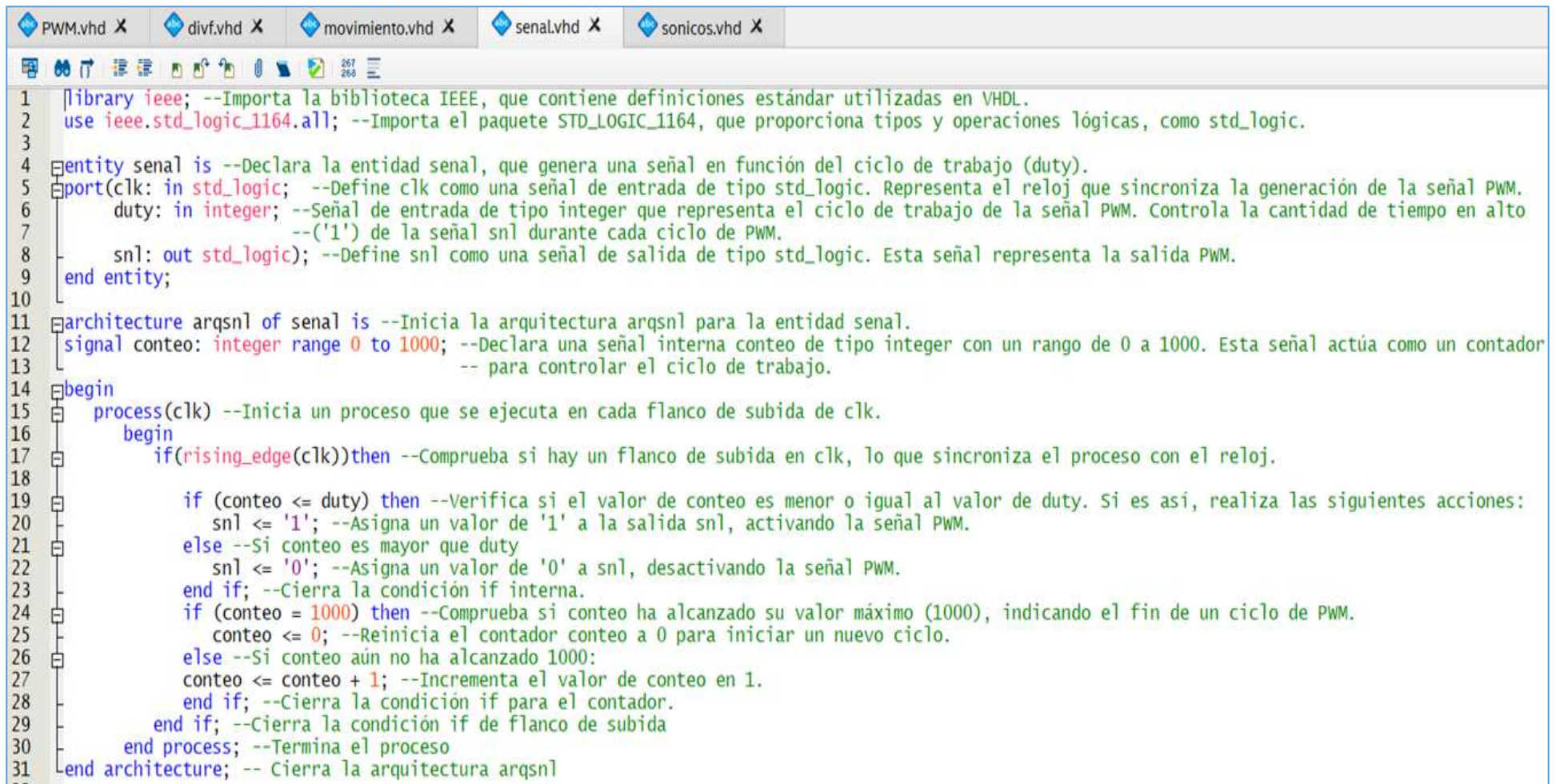
```
PWM.vhd X divf.vhd X movimiento.vhd X senal.vhd X sonicos.vhd X
48 if counter = counter_multiplier then -- Si counter alcanza counter_multiplier, activa enable y reinicia el contador.
49     enable <= '1';
50     counter <= 0;
51 else --Si counter no alcanza counter_multiplier, incrementa el contador y desactiva enable.
52     counter <= counter + 1;
53     enable <= '0';
54 end if;
55
56 if enable = '1' then
57     if pi = '1' then
58         valor <= 55; -- 5.5% ~1 ms
59         direccion <= '1'; -- Cambia dirección a incrementar
60     elsif pf = '1' then
61         valor <= 95; -- 9.5% ~2ms
62         direccion <= '0'; -- Cambia dirección a decrementar
63     elsif direccion = '1' and valor < 95 then
64         valor <= valor + 1;
65         if valor = 94 then
66             direccion <= '0'; -- Cambia dirección a decrementar
67         end if;
68     elsif direccion = '0' and valor > 55 then
69         valor <= valor - 1;
70         if valor = 56 then
71             direccion <= '1'; -- Cambia dirección a incrementar
72         end if;
73     end if;
74 end if;
75 end process;
76
77 ancho <= valor; -- Asignación continua de 'ancho' al valor calculado
78 end architecture;
79
80
```

8.4.4. Código para la señal

Este código **senal** en VHDL implementa un generador de señal PWM (modulación por ancho de pulso) llamado senal. A partir de una señal de reloj de entrada (clk) y un valor de duty cycle (duty), controla la salida (snl) que activa o desactiva el pulso en función del valor de duty. Dentro del proceso, en cada flanco ascendente de clk, el contador (conteo) se incrementa. Cuando conteo es menor o igual a duty, snl se establece en '1' (encendido); cuando conteo supera duty, se establece en '0' (apagado). El contador se reinicia a cero al llegar a 1000, generando una señal con un duty cycle configurable y repetitivo, útil para controlar la potencia o velocidad de dispositivos como motores.



```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity senal is
5  port(clk: in std_logic;
6        duty: in integer;
7        snl: out std_logic);
8  end entity;
9
10 architecture arqsnl of senal is
11 signal conteo: integer range 0 to 1000;
12 begin
13     process(clk)
14     begin
15         if(rising_edge(clk))then
16             if (conteo <= duty) then
17                 snl <= '1';
18             else
19                 snl <= '0';
20             end if;
21             if (conteo = 1000) then
22                 conteo <= 0;
23             else
24                 conteo <= conteo + 1;
25             end if;
26         end if;
27     end process;
28 end architecture;
```



The image shows a screenshot of a VHDL code editor with five tabs open: PWM.vhd, divf.vhd, movimiento.vhd, senal.vhd, and sonicos.vhd. The 'senal.vhd' tab is active, displaying the following code:

```

1  library ieee; --Importa la biblioteca IEEE, que contiene definiciones estándar utilizadas en VHDL.
2  use ieee.std_logic_1164.all; --Importa el paquete STD_LOGIC_1164, que proporciona tipos y operaciones lógicas, como std_logic.
3
4  entity senal is --Declara la entidad senal, que genera una señal en función del ciclo de trabajo (duty).
5  port(clk: in std_logic; --Define clk como una señal de entrada de tipo std_logic. Representa el reloj que sincroniza la generación de la señal PWM.
6        duty: in integer; --Señal de entrada de tipo integer que representa el ciclo de trabajo de la señal PWM. Controla la cantidad de tiempo en alto
7          --('1') de la señal snl durante cada ciclo de PWM.
8        snl: out std_logic); --Define snl como una señal de salida de tipo std_logic. Esta señal representa la salida PWM.
9  end entity;
10
11 architecture arqsnl of senal is --Inicia la arquitectura arqsnl para la entidad senal.
12  signal conteo: integer range 0 to 1000; --Declara una señal interna conteo de tipo integer con un rango de 0 a 1000. Esta señal actúa como un contador
13    -- para controlar el ciclo de trabajo.
14  begin
15    process(clk) --Inicia un proceso que se ejecuta en cada flanco de subida de clk.
16    begin
17      if(rising_edge(clk))then --Comprueba si hay un flanco de subida en clk, lo que sincroniza el proceso con el reloj.
18
19        if (conteo <= duty) then --Verifica si el valor de conteo es menor o igual al valor de duty. Si es así, realiza las siguientes acciones:
20          snl <= '1'; --Asigna un valor de '1' a la salida snl, activando la señal PWM.
21        else --Si conteo es mayor que duty
22          snl <= '0'; --Asigna un valor de '0' a snl, desactivando la señal PWM.
23        end if; --Cierra la condición if interna.
24        if (conteo = 1000) then --Comprueba si conteo ha alcanzado su valor máximo (1000), indicando el fin de un ciclo de PWM.
25          conteo <= 0; --Reinicia el contador conteo a 0 para iniciar un nuevo ciclo.
26        else --Si conteo aún no ha alcanzado 1000:
27          conteo <= conteo + 1; --Incrementa el valor de conteo en 1.
28        end if; --Cierra la condición if para el contador.
29      end if; --Cierra la condición if de flanco de subida
30    end process; --Termina el proceso
31  end architecture; -- Cierra la arquitectura arqsnl

```

8.5.4. Código sensor ultrasónico

Este código sonicos en VHDL implementa un módulo denominado sonicos que maneja la medición de distancia utilizando un sensor ultrasónico y muestra los resultados en displays de siete segmentos. La arquitectura se basa en un contador que mide el tiempo entre la emisión y la recepción de una señal (eco), permitiendo calcular la distancia en centímetros. La distancia calculada se divide en unidades, decenas y centenas, para luego ser representada en displays de siete segmentos.

El código también controla dos LEDs, encendiendo uno cuando el objeto se encuentra a 20 cm y el otro a 5 cm, proporcionando una indicación visual rápida de la proximidad. La señal sensor_disp activa el sensor para la medición, mientras que sensor_eco recibe la señal de respuesta. Cuando la distancia se mide, se muestra en los displays y se almacena en centimetros_out.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity sonicos is
6  Port (
7      clk: in STD_LOGIC;
8      -- sensor
9      sensor_disp: out STD_LOGIC;
10     sensor_eco: in STD_LOGIC;
11
12     -- distancia
13     displayunidades: out std_logic_vector(6 downto 0);
14     displaydecenas: out std_logic_vector(6 downto 0);
15     displaycentenas: out std_logic_vector(6 downto 0); -- Nuevo puerto para centenas
16
17     -- LEDs
18     led_20cm: out std_logic; -- LED que se enciende a 20 cm
19     led_5cm: out std_logic; -- LED que se enciende a 5 cm
20
21     centimetros_out: out std_logic_vector(15 downto 0)
22 );
23 end sonicos;
24
25 architecture arqsonicos of sonicos is
26     signal cuenta: unsigned(16 downto 0) := (others => '0');
27
28     -- conversión de unidades
29     signal centimetros: unsigned(15 downto 0) := (others => '0');
30     signal centimetros_unid: unsigned(3 downto 0) := (others => '0');
31     signal centimetros_dece: unsigned(3 downto 0) := (others => '0');
32     signal centimetros_cent: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para centenas
33     signal sal_unid: unsigned(3 downto 0) := (others => '0');
34     signal sal_dece: unsigned(3 downto 0) := (others => '0');
35     signal sal_cent: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para salida de centenas
36     signal digitounidad: unsigned(3 downto 0) := (others => '0');
37     signal digitodecena: unsigned(3 downto 0) := (others => '0');
38     signal digitocentena: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para dígito de centenas
39
40     -- señales del sensor
41     signal eco_pasado: std_logic := '0';
42     signal eco_sinc: std_logic := '0';
43     signal eco_nsinc: std_logic := '0';
44
45     signal espera: std_logic := '0';
46     signal siete_seg_cuenta: unsigned(15 downto 0) := (others => '0');
47
```



```

PWM.vhd x divf.vhd x movimiento.vhd x senal.vhd x sonicos.vhd x
1 library IEEE; --Importa la biblioteca IEEE.
2 use IEEE.STD_LOGIC_1164.ALL; --Permite el uso de tipos de señal como std_logic.
3 use IEEE.NUMERIC_STD.ALL; --Permite operaciones aritméticas y conversiones en tipos de datos numéricos como unsigned.
4
5 entity sonicos is --Define la entidad sonicos, que maneja la señal del sensor, la distancia y los LEDs de advertencia.
6   Port (
7     clk_u: in STD_LOGIC; --Señal de reloj de entrada clk_u.
8     -- sensor
9     sensor_disp: out STD_LOGIC; --Señal de disparo para el sensor ultrasónico.
10    sensor_eco: in STD_LOGIC; --Entrada del pulso de eco.
11
12    -- distancia
13    displayunidades: out std_logic_vector(6 downto 0); -- Nuevo puerto para unidades
14    displaydecenas: out std_logic_vector(6 downto 0); -- Nuevo puerto para decenas
15    displaycentenas: out std_logic_vector(6 downto 0); -- Nuevo puerto para centenas
16
17    -- LEDs
18    led_20cm: out std_logic; -- LED que se enciende a 20 cm
19    led_5cm: out std_logic; -- LED que se enciende a 5 cm
20
21    centimetros_out: out std_logic_vector(15 downto 0) --Salida que muestra el valor de la distancia en centímetros en formato std_logic_vector
22  );
23 end sonicos;
24
25 architecture arqsonicos of sonicos is --Define la arquitectura arqsonicos
26   signal cuenta: unsigned(16 downto 0) := (others => '0'); --cuenta: Un contador para temporización de disparo del sensor.
27
28   -- conversión de unidades
29   signal centimetros: unsigned(15 downto 0) := (others => '0'); --centimetros: Almacena la distancia medida en centímetros.
30   signal centimetros_unid: unsigned(3 downto 0) := (others => '0'); --Señales que almacenan las unidades de la distancia medida.
31   signal centimetros_dece: unsigned(3 downto 0) := (others => '0'); --Señales que almacenan las decenas de la distancia medida.
32
33   signal centimetros_cent: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para centenas
34   signal sal_unid: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para salida de unidades
35   signal sal_dece: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para salida de decenas
36   signal sal_cent: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para salida de centenas
37   signal digitounidad: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para dígito de unidades
38   signal digitodecena: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para dígito de decenas
39   signal digitocentena: unsigned(3 downto 0) := (others => '0'); -- Nueva señal para dígito de centenas
40
41   -- señales del sensor
42   signal eco_pasado: std_logic := '0'; --Señales de sincronización para el eco.
43   signal eco_sinc: std_logic := '0'; --Señales de sincronización para el eco.
44   signal eco_nsinc: std_logic := '0'; --Señales de sincronización para el eco.
45   signal espera: std_logic := '0'; --Controla el estado de espera entre disparos del sensor.
46   signal siete_seg_cuenta: unsigned(15 downto 0) := (others => '0'); --siete_seg_cuenta: Controla la duración del pulso de eco para la medición
47   --distancia.
48
49 begin
50   -- trabajo del sensor
51   Trigger: process(clk_u) --Proceso que controla el sensor y la medición de distancia, activado en cada flanco de subida de clk_u.
52   begin
53     if rising_edge(clk_u) then
54       if espera = '0' then --Si no está en espera, inicia el disparo del sensor.
55         if cuenta = 500 then --Envía una señal de disparo durante un tiempo específico.
56           sensor_disp <= '0';
57           espera <= '1';
58           cuenta <= (others => '0');
59         else
60           sensor_disp <= '1';
61           cuenta <= cuenta + 1;
62         end if;
63       end if;
64     end if;
65   end process;
66

```

```

PWM.vhd X divt.vhd X movimiento.vhd X senal.vhd X sonicos.vhd X
67
68 -- distancia
69 elsif eco_pasado = '0' and eco_sinc = '1' then
70     cuenta <= (others => '0');
71     centimetros <= (others => '0');
72     centimetros_unid <= (others => '0'); -- Reset de unidades
73     centimetros_dece <= (others => '0'); -- Reset de decenas
74     centimetros_cent <= (others => '0'); -- Reset de centenas
75
76 elsif eco_pasado = '1' and eco_sinc = '0' then
77     sal_unid <= centimetros_unid; -- Salida de unidades
78     sal_dece <= centimetros_dece; -- Salida de decenas
79     sal_cent <= centimetros_cent; -- Salida de centenas
80     digitounidad <= sal_unid; -- Asignación del dígito de unidades
81     digitodecena <= sal_dece; -- Asignación del dígito de decenas
82     digitocentena <= sal_cent; -- Asignación del dígito de centenas
83
84 -- Control de los LEDs
85 centimetros_out <= std_logic_vector(centimetros);
86 if centimetros <= 5 then
87     led_5cm <= '1';
88     led_20cm <= '0'; -- Apagar LED de 20 cm cuando esté a 5 cm
89 elsif centimetros <= 20 then
90     led_20cm <= '1';
91     led_5cm <= '0'; -- Apagar LED de 5 cm cuando esté a 20 cm
92 else
93     led_5cm <= '0';
94     led_20cm <= '0'; -- Apagar ambos LEDs si está a más de 20 cm
95 end if;
96
97 elsif cuenta = 2900 - 1 then
98     if centimetros_unid = 9 then
99         centimetros_unid <= (others => '0');
100         if centimetros_dece = 9 then
101             centimetros_dece <= (others => '0');
102             centimetros_cent <= centimetros_cent + 1; -- Incremento de centenas
103         else
104             centimetros_dece <= centimetros_dece + 1; -- Incremento de decenas
105         end if;
106     else
107         centimetros_unid <= centimetros_unid + 1; -- Incremento de unidades
108     end if;
109     centimetros <= centimetros + 1;
110     cuenta <= (others => '0');
111
112     if centimetros = 3448 then
113         espera <= '0';
114     end if;
115
116 else
117     cuenta <= cuenta + 1;
118 end if;
119
120 eco_pasado <= eco_sinc;
121 eco_sinc <= eco_nsinc;
122 eco_nsinc <= sensor_eco;
123 end if;
124 end process;
125
126 -- Display unidades
127 Unidades: process(digitounidad)
128 begin
129     case digitounidad is
130         when "0000" => displayunidades <= "1000000";
131         when "0001" => displayunidades <= "1111001";
132         when "0010" => displayunidades <= "0100100";
133         when "0011" => displayunidades <= "0110000";

```

```

PWM.vhd x divf.vhd x movimiento.vhd x senal.vhd x sonicos.vhd x
133 when "0100" => displayunidades <= "0011001";
134 when "0101" => displayunidades <= "0010010";
135 when "0110" => displayunidades <= "0000010";
136 when "0111" => displayunidades <= "1111000";
137 when "1000" => displayunidades <= "0000000";
138 when "1001" => displayunidades <= "0011000";
139 when others => displayunidades <= "1111111";
140 end case;
141 end process;
142
143 -- Display decenas
144 Decenas: process(digitodecena)
145 begin
146     case digitodecena is
147     when "0000" => displaydecenas <= "1000000";
148     when "0001" => displaydecenas <= "1111001";
149     when "0010" => displaydecenas <= "0100100";
150     when "0011" => displaydecenas <= "0110000";
151     when "0100" => displaydecenas <= "0011001";
152     when "0101" => displaydecenas <= "0010010";
153     when "0110" => displaydecenas <= "0000010";
154     when "0111" => displaydecenas <= "1111000";
155     when "1000" => displaydecenas <= "0000000";
156     when "1001" => displaydecenas <= "0011000";
157     when others => displaydecenas <= "1111111";
158     end case;
159 end process;
160
161 -- Display centenas
162 Centenas: process(digitocentena)
163 begin
164     case digitocentena is
165     when "0000" => displaycentenas <= "1000000";
166     when "0001" => displaycentenas <= "1111001";
167     when "0010" => displaycentenas <= "0100100";
168     when "0011" => displaycentenas <= "0110000";
169     when "0100" => displaycentenas <= "0011001";
170     when "0101" => displaycentenas <= "0010010";
171     when "0110" => displaycentenas <= "0000010";
172     when "0111" => displaycentenas <= "1111000";
173     when "1000" => displaycentenas <= "0000000";
174     when "1001" => displaycentenas <= "0011000";
175     when others => displaycentenas <= "1111111";
176     end case;
177 end process;
178
179 end;
180

```


8.6. Pines utilizados en Quartus

Node Name	Direction	Location	I/O Bank	VREF Group	Filter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
centimetros_out[15]	Output				PIN_W9	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[14]	Output	PIN_AB2	3	B3_NO	PIN_AB2	2.5 V		12mA (default)	2 (default)		
centimetros_out[13]	Output				PIN_P12	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[12]	Output				PIN_AA8	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[11]	Output				PIN_V8	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[10]	Output				PIN_AB3	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[9]	Output	PIN_Y6	3	B3_NO	PIN_Y6	2.5 V		12mA (default)	2 (default)		
centimetros_out[8]	Output				PIN_W16	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[7]	Output				PIN_AB13	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[6]	Output				PIN_A11	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[5]	Output				PIN_B12	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[4]	Output				PIN_C13	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[3]	Output				PIN_R11	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[2]	Output				PIN_E9	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[1]	Output				PIN_H19	2.5 V (default)		12mA (default)	2 (default)		
centimetros_out[0]	Output				PIN_B10	2.5 V (default)		12mA (default)	2 (default)		
clk	Input	PIN_P11	3	B3_NO	PIN_P11	2.5 V		12mA (default)			
clkU	Input	PIN_N14	6	B6_NO	PIN_N14	2.5 V		12mA (default)			
control	Output	PIN_AA19	4	B4_NO	PIN_AA19	2.5 V		12mA (default)	2 (default)		
displaycentenas[6]	Output	PIN_B22	6	B6_NO	PIN_B22	2.5 V		12mA (default)	2 (default)		
displaycentenas[5]	Output	PIN_C22	6	B6_NO	PIN_C22	2.5 V		12mA (default)	2 (default)		
displaycentenas[4]	Output	PIN_B21	6	B6_NO	PIN_B21	2.5 V		12mA (default)	2 (default)		
displaycentenas[3]	Output	PIN_A21	6	B6_NO	PIN_A21	2.5 V		12mA (default)	2 (default)		
displaycentenas[2]	Output	PIN_B19	7	B7_NO	PIN_B19	2.5 V		12mA (default)	2 (default)		
displaycentenas[1]	Output	PIN_A20	7	B7_NO	PIN_A20	2.5 V		12mA (default)	2 (default)		
displaycentenas[0]	Output	PIN_B20	6	B6_NO	PIN_B20	2.5 V		12mA (default)	2 (default)		
displaydecenas[6]	Output	PIN_B17	7	B7_NO	PIN_B17	2.5 V		12mA (default)	2 (default)		

9. Resultados

Los resultados obtenidos validan el diseño y la implementación del sistema propuesto. La combinación de la tarjeta DE10-Lite, el sensor ultrasónico, y el control



Imagen del proyecto físico, durante la prueba de funcionamiento con 23 cm

PWM demostró ser una solución efectiva para aplicaciones que requieren mediciones precisas y control dinámico de actuadores.

El sensor ultrasónico HC-SR04 proporcionó mediciones consistentes y confiables en el rango esperado, la implementación del módulo VHDL para gestionar los pulsos de trigger y eco funcionó

correctamente, calculando de manera precisa la distancia del objeto detectado. Las distancias fueron mostradas en tiempo real en los displays de 7 segmentos de la tarjeta DE10-Lite, facilitando al usuario la lectura inmediata de los datos.

El control de la velocidad del motor basado en la distancia detectada funcionó conforme al diseño. Se generó una señal PWM dinámica en VHDL con

ciclos de trabajo ajustables en función de la cercanía o lejanía del objeto. El motor respondió adecuadamente, aumentando gradualmente su velocidad cuando el objeto se alejaba y reduciéndola a medida que se acercaba.



El sistema se sometió a pruebas prolongadas y bajo diferentes condiciones y no se registraron fallas durante la operación continua, y el motor operó sin problemas de sobrecalentamiento o inconsistencias en la respuesta. Asimismo, el uso de LEDs para indicar estados y distancias proporcionó buenos resultados visuales.



Imagen del proyecto físico, durante la prueba de funcionamiento en clase

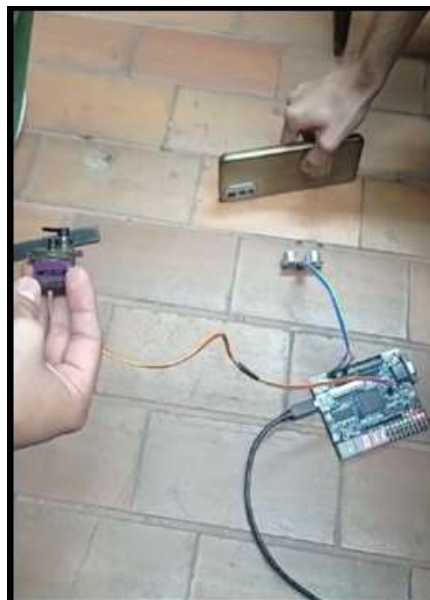


Imagen del proyecto físico, durante la prueba de funcionamiento en clase, con menor distancia de alejamiento

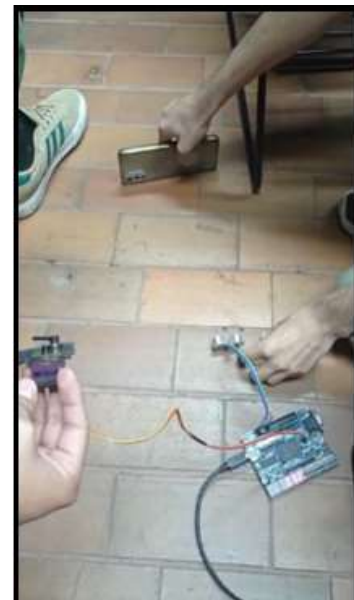


Imagen del proyecto físico, durante la prueba de funcionamiento en clase, con mayor distancia de alejamiento

10. Conclusiones

🚩 **Bravo Luna Ivonne Monserrat**

En conclusión, el presente proyecto ha sido una experiencia muy enriquecedora, pues me ha permitido profundizar en aspectos muy importantes sobre el control de sistemas automatizados, por medio de la implementación del control de velocidad de un servomotor mediante modulación por ancho de pulso, así como también por la integración de un sensor ultrasónico, que me ha permitido apreciar de manera práctica la importancia de la sincronización de los materiales en un sistema de control en tiempo real.

Así mismo, uno de los principales aprendizajes que tuve fue el manejo del PWM para el control de velocidad de un servomotor, pues a pesar de haberlo visto previamente en prácticas de laboratorio y teoría, su uso práctico me permitió desarrollar un sistema adaptativo capaz de responder de forma dinámica a un

entorno. La integración práctica de este proyecto requirió de una planificación minuciosa y pruebas exhaustivas, tanto en la búsqueda de resultados experimentales y matemáticos, como en el análisis preciso de los requerimientos, especificaciones y modelos de este sistema. Denotando la importancia que tienen no solo los materiales utilizados, sino también la integración de relojes, de contadores, de triggers, etc. para un funcionamiento eficiente y correcto.

Es importante destacar que el trabajo en equipo ha sido fundamental para alcanzar el objetivo propuesto, ya que la división del trabajo y la combinación de ideas diferentes han sido cruciales para poder superar los desafíos técnicos y resolver los problemas que surgieron durante el desarrollo del proyecto. Algunos de estos problemas surgieron al analizar y codificar los requerimientos solicitados. Por ejemplo, en la implementación del control PWM, hubo momentos en los que tuvimos que replantear nuestras ideas e investigar nuevamente el funcionamiento de cada material para poder cumplir con el objetivo deseado, enriqueciendo nuestro aprendizaje y aprendiendo a enfrentar este tipo de desafíos técnicos.

Díaz González Rivas Ángel Iñaqui

En términos de precisión, se observó que el control PWM proporcionó una respuesta bastante precisa para ajustar la velocidad del motor, aunque en algunos casos, el sensor ultrasónico mostró inconsistencias que afectaron la medición, especialmente en rangos cortos o con interferencias externas. En cuanto a eficiencia energética, la integración de relojes permitió una reducción significativa en el consumo de potencia. Comparativamente, el control por PWM tuvo un rendimiento más estable frente a las variaciones del entorno, mientras que el sensor ultrasónico podría beneficiarse de un refinamiento adicional para mejorar su confiabilidad. Las métricas de tiempo de respuesta se mantuvieron dentro de los límites aceptables, logrando un buen balance entre control y eficiencia. En general, este proyecto muestra que, aunque el sistema presenta áreas a mejorar, ofrece una solución viable y aceptable

García Gallegos Samantha

Este proyecto me permitió profundizar en el control de motores mediante PWM, donde aprendí a ajustar con precisión su velocidad, lo que resultó esencial para el funcionamiento eficaz del sistema. La integración de relojes presentó un desafío interesante, ya que fue necesario asegurar que todas las señales estuvieran correctamente sincronizadas, destacando la importancia de la sincronización en sistemas de control en tiempo real. La incorporación del sensor ultrasónico facilitó el diseño de un sistema adaptativo que ajusta automáticamente la velocidad del motor según la distancia detectada, brindando una comprensión práctica del uso de sensores en tiempo real y su papel en la interacción con el entorno. Integrar todos los componentes, como el motor, el sensor ultrasónico, PWM y los relojes, fue un proceso complejo que mejoró mis habilidades en la unión de hardware y software, resaltando la importancia de la planificación cuidadosa y las pruebas constantes en proyectos de este tipo. Además, esta experiencia me mostró el valor de la automatización en sistemas autónomos, donde los dispositivos pueden reaccionar de manera eficiente a los cambios en el entorno. Además de incorporar el proyecto tuvimos que aprender a trabajar en equipo, donde tuvimos que dividir el trabajo y lo más difícil de todo fue incorporar las ideas de cada uno en el trabajo, sin embargo, creo que logramos hacer un buen trabajo que cumple con cada una de las características que se solicitan. Aunque tengo que admitir que nos comió un poco el tiempo para realizar el archivo escrito, pero creo que logramos terminar el trabajo de una manera agradable y satisfactoria para el equipo. Tengo que decir que este proyecto se encargó de reforzar los temas anteriormente vistos, logrando que comprendamos de mejor forma los temas que abarca el temario.

Siliano Haller Rodrigo

A lo largo de este proyecto, he logrado entender de manera más profunda cómo funciona la modulación por ancho de pulso (PWM) y su importancia en el control de motores. Al implementar PWM, pude controlar de manera precisa la velocidad del motor, lo cual fue clave para el éxito del sistema.

Importancia de la sincronización con los relojes: Integrar el uso de relojes fue un desafío interesante, ya que fue necesario asegurar que todas las señales estuvieran sincronizadas correctamente. Esto me permitió ver la importancia de la sincronización en sistemas de control en tiempo real, especialmente cuando se trabaja con varios componentes como motores y sensores.

Aplicación práctica de sensores ultrasónicos: Trabajar con el sensor ultrasónico me permitió poner en práctica lo aprendido sobre sensores y su uso para obtener información del entorno en tiempo real. Pude observar cómo el sistema ajustaba automáticamente la velocidad del motor en función de las distancias detectadas, lo que me ayudó a comprender mejor cómo se pueden diseñar sistemas adaptativos.

Desafíos y aprendizaje en la integración de componentes: Uno de los aspectos más difíciles fue integrar todos los componentes del proyecto (el motor, el sensor ultrasónico, PWM y los relojes) de manera que funcionaran juntos de forma efectiva. Sin embargo, este desafío me ayudó a mejorar mis habilidades en la integración de hardware y software y a entender la importancia de la planificación y la prueba constante en este tipo de proyectos.

Valor de la automatización en sistemas de control: Gracias a este proyecto, he aprendido la relevancia de la automatización y cómo los sistemas pueden reaccionar de forma autónoma ante las condiciones del entorno, como los cambios en la distancia. Esto me ha motivado a seguir investigando sobre cómo mejorar la eficiencia y la capacidad de los sistemas autónomos en futuras aplicaciones. Este proyecto no solo me permitió aplicar los conocimientos teóricos adquiridos en clase, sino que también me brindó una visión práctica de cómo se diseñan y controlan sistemas automáticos.

11. *Referencias Bibliográficas*

- All About Circuits. (2023). Understanding Pulse Width Modulation (PWM). Recuperado de <https://www.allaboutcircuits.com/technical-articles/understanding-pulse-width-modulation/>
- Arduino. (2023). PWM: Control de velocidad de motores DC. Recuperado de <https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM>
- Chávez, M., Norma, R., Flores Olvera, E., Fonseca Chávez, V., Guevara, E., María, R., Carrillo, S., Prieto Meléndez, M., Ramírez Chavarría, R., & Giovanni, R. (n.d.). <http://profesores.fi-b.unam.mx/fpga/Practicas%20VLSI.pdf>
- Digi-Key Electronics. (2023). Using PWM to Control DC Motor Speed. Recuperado de <https://www.digikey.com/en/articles/using-pwm-to-control-dc-motor-speed>
- ¿Qué es PWM y cómo usarlo? (2020). Solectroshop.com. <https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38?srsId=AfmBOorDoxJvkIMBobMhPuAFmi-78m1HxvNhVI9gKpPJb1co41wWeM7Z>
- Marcos, S. M. (2014). Introducción a la programación en VHDL. <https://docta.ucm.es/entities/publication/3152080b-11fa-466c-88c7-61ed262828a6>
- Paguayo. (2024, Junio 3). FPGA (Field Programmable Gate Array) - MCI Educación. MCI Educación. Recuperado de <https://cursos.mcielectronics.cl/2019/06/18/fpga-field-programmable-gate-array/>
- Staff, A. (2024, January 23). Qué es un servomotor: Definición, orígenes, componentes, tipos y aplicaciones - ADVANCED Motion Controls. ADVANCED Controles de Movimiento. <https://www.a-m-c.com/es/servomotor/>

Texas Instruments. (2023). Fundamentals of PWM for motor control. Recuperado de <https://www.ti.com/lit/an/slva001e/slva001e.pdf>

(S/f). Recuperado el 6 de septiembre de 2024, de <http://chrome-extension://efaidnbmnnnibpcajpcgclefindmkaj/http://cimogsys.esPOCH.edu.ec/direccion-publicaciones/public/docs/books/2019-09-19-195937-96%20Introducci%C3%B3n%20al%20VHDL.pdf>

(S/f-b). Recuperado el 6 de septiembre de 2024, de <http://chrome-extension://efaidnbmnnnibpcajpcgclefindmkaj/https://catedras.facet.unt.edu.ar/de/wp-content/uploads/sites/61/2015/01/VLSI-1.pdf>

(S/f-c). Recuperado el 6 de Septiembre de 2024, de <http://chrome-extension://efaidnbmnnnibpcajpcgclefindmkaj/https://uvadoc.uva.es/bitstream/handle/10324/11863/TFG-P-204.pdf?sequence=1>

ULTRASONIC-HC-SR04 SENSOR DE DISTANCIA ULTRASONICO HC-SR04.
(n.d.). <https://agelectronica.lat/pdfs/textos/U/ULTRASONIC-HC-SR04.PDF>

12. Lista de Imágenes

Imagen 1. Imagen de un sensor ultrasónico obtenida de Preparación de Salvador, I. M. (2020, April 24). ¿Qué es un sensor ultrasónico HC-SR04 ? Un sensor ultrasónico HC-SR04 es un transductor que mide Leer más. Blog Arduino, LabVIEW Y Electrónica. <https://electronicamade.com/sensor-ultrasonico/>

Imagen 2. Imagen PWM obtenida de ¿Qué es PWM y cómo usarlo? (2020). Solectroshop.com. <https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38?srsId=AfmBOorDoxJvkIMBobMhPuAFmi-78m1HxvNhVI9gKpPJb1co41wWeM7Z>

Imagen 3. Imagen de un servomotor, obtenida de Servomotores de 180 y 360 grados de rotación. (2021). La Electrónica.
<https://laelectronica.com.gt/extras/servomotores-de-180-y-360-grados-de-rotacion>

Imagen 4. Imagen de una tarjeta DE10-Lite obtenida de Tarjeta DE10-Lite - SANDOROBOTICS. (2024, September 26). SANDOROBOTICS.
<https://sandorobotics.com.mx/producto/de10-lt/>