Universidad Politécnica del Estado de Morelos

LEVITADOR ACÚSTICO UNIAXIAL DE FASES VARIABLES PARA LA GENERACIÓN DE VÓRTICES ACÚSTICOS.

T E S I S

Que para obtener el título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

P r e s e n t a

Said Israel Vara Tiempos

Directores de Tesis

M. C. E. Miguel Ángel Velasco Castillo

Dr. Víctor Ulises Lev Contreras Loera

Jiutepec, Morelos Mayo 2021

CONTENIDO

Agradecimientos:

Agradezco y dedico este proyecto de tesis a la persona que me dio la vida a mi madre, por su comprensión, motivación y apoyo que me han brindado para lograr todas y cada una de mis metas, así como me impulsa a lograr mis sueños y anhelos. Todo este trabajo ha sido posible gracias a ella ¡Gracias!

A mis hermanos estar presente no solo en esta etapa tan importante de mi vida si no en todo momento ofreciéndome y buscando lo mejor para mi persona.

A Andrea por que llego a ser una mujer muy importante en mi vida, que siempre me motivo a seguir adelante y por todo su apoyo incondicional.

Al Instituto de Ciencias Físicas por haber formado parte del grupo de trabajo del laboratorio de Óptica Aplicada.

A mi asesor Dr. Víctor Contreras, por brindarme su ayuda, paciencia y dedicación para la realización del presente trabajo, así como, por haber confiado y sugerido el tema de tesis. Gracias por su amistad y por compartir su experiencia y conocimiento.

A mi asesor Miguel Ángel Velasco Castillo por su gran apoyo para la culminación de mis estudios profesionales y para la elaboración de esta tesis.

A los miembros del jurado, Eduardo Vargas Zacarías y Alfredo Gil Velasco, por sus comentarios y sugerencias sobre la tesis.

Al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) IN109221 de DGAPA-UNAM por brindar los recursos para la adquisición del equipo y material utilizado en este trabajo.

Resumen:

El presente documento tiene como principal objetivo implementar y caracterizar un sistema de levitación acústico de fase variable. La levitación acústica no solo es un fenómeno físico interesante, sino también una herramienta para suspender y manipular objetos. La levitación acústica se puede combinar con sistemas de medición sin contacto, permitiendo el análisis y la caracterización de muestras en levitación sin tocarlas.

Con este trabajo se describen todos los conceptos físicos que hay detrás de este fenómeno, además de los conceptos básicos de electrónica y programación necesarios para construir un levitador de fase variable. La parte final de este trabajo muestra la caracterización experimental de las diferentes trampas acústicas que se pueden crear con este sistema de levitación acústica de fase variable.

Lista de abreviaturas

1. ICF: Instituto de Ciencias Físicas.
2. FPGA: Field Programmable Gate Array.
3. LIBS: Espectroscopía de rompimiento inducido por láser.
4. HCI: Human-Computer Interaction.
5. UART: Universal Asynchronous Receiver-Transmitter.
6. HDL: Hardware Description Language.
7. RISC: Reduced Instruction Set Computing.
8. VHSIC: Very High-Speed Integrated Circuit.
9. VHDL: VHSIC Hardware Description Language.
10. PLD: Programmable Logic Device.
11. PLA: Programmable Logic Array.
12. PAL: Programmable Array Logic.
13. ASIC: Application Specific Integrated Circuit.
14. JTAG: Joint Test Action Group.
15. GUI: Graphical User Interface.
16. PCB: Printed Circuit Board.
17. PLL: Phase-Locked Loop.
18. RAM: Random Access Memory.
19. JTAG: Joint Test Action Group.
20. FSM: Finite-State Machine.

Lista de Figuras

Lista de Tablas

1. Capítulo I:

INTRODUCCIÓN

1.1 Introducción

La levitación es el proceso de suspender objetos en el aire u otro fluido sin ningún soporte mecánico. El principio más común de un levitador acústico es la generación de una onda estacionaria entre un emisor, llamado transductor, y un reflector separado por una distancia igual a un múltiplo entero de media longitud de onda. Las ondas acústicas estacionarias ejercen fuerzas de radiación que forman trampas acústicas en los puntos donde estas fuerzas convergen. Estas trampas acústicas permiten la levitación de partículas de una amplia gama de materiales y tamaños a través de aire, agua o tejidos biológicos.

Los levitadores de un solo eje son la forma más común de generar trampas acústicas. Se componen de un transductor (o arreglo de transductores) acústico(s) y un reflector u otro transductor (o arreglo de transductores) formando una cavidad con simetría axial. Este arreglo produce interferencia entre todas las ondas generadas dentro de la cavidad, creando nodos y antinodos en los que hay una diferencia de presión. La partícula se sitúa en los nodos, que es donde generalmente se alcanza el equilibrio estático. A mayor amplitud de presión en los antinodos, mayor es el gradiente de presión en la trampa y como consecuencia, mayor es la fuerza de levitación. La amplitud de la onda estacionaria se controla con el voltaje suministrado a los transductores y la geometría de la cavidad. Sin embargo, en algunos levitadores también es posible controlar la fase de los transductores.

1.2 Antecedentes

La levitación acústica es una técnica que se ha desarrollado a lo largo de los últimos 40 años a partir de experimentos diseñados para generar ambientes de microgravedad (R. H. Morris, 2019). La investigación en levitación acústica se ha basado principalmente en sistemas uniaxiales que emplean transductores PZT tipo Langevin acoplados a amplificadores mecánicos. Sin embargo, en la última década se han desarrollado levitadores basados en arreglos de transductores compactos y de bajo costo que han permitido, por ejemplo, manipular la posición y orientación de los objetos levitados al variar de manera controlada la fase del arreglo de

transductores (L. Cox, 2018) (A. Marzo, 2017). Una forma de lograr esta manipulación de los objetos levitados es a través del control de las fases de los transductores compactos, lo que permite la generación de vórtices acústicos y la transferencia de momento angular.

En el Laboratorio de Óptica Aplicada (LOA) del Instituto de Ciencias Físicas (ICF) se está trabajando con levitadores acústicos para el análisis de contaminantes en líquidos para poder detectar la presencia de elementos de interés toxicológico como el caso de metales pesados en agua potable. Un sistema de levitación robusto y de bajo costo con la capacidad de generar vórtices acústicos permitirá, por ejemplo, levitar de manera estable objetos con geometrías diferentes a la esférica, ampliando sus potenciales aplicaciones tecnológicas. La levitación de partículas se ha utilizado ampliamente como un medio de manipulación de objetos libre de contacto, se tiene la ventaja de que es simple y puede levitar tanto materiales no magnéticos como no conductores.

1.3 Planteamiento del problema

La levitación acústica no solo es un fenómeno físico interesante, sino también una herramienta prometedora para múltiples disciplinas como la biología, la química o la farmacia. Las características únicas de la levitación acústica también tienen un gran potencial para realizar algunas tareas en la industria, como la manipulación sin contacto de objetos frágiles o el procesamiento de pequeños volúmenes de líquidos.

Los levitadores generalmente utilizan transductores PZT tipo Langevin y tienen la ventaja de soportar altos voltajes, sin embargo, tienen varias desventajas. Una de ellas es que son muy sensible a la variación de su frecuencia de operación y debido a su angosto ancho de banda, ligeras variaciones de temperatura o voltaje desestabilizan el sistema. En la actualidad se han desarrollado levitadores conformados por arreglos de transductores ultrasónicos. Estos levitadores son menos sensibles a los cambios de temperatura y humedad, funcionan con bajo voltaje, son fáciles de operar (A. Marzo, 2017) y son capaces de levitar objetos densos como los levitadores basados en transductores Langevin (Contreras, 2021). Los

levitadores uniaxiales de cavidad esférica son los más eficientes ya que permiten enfocar la presión acústica en el centro de la cavidad. Sin embargo, una limitación de estos levitadores es que en general solo levitan objetos milimétricos con geometría esférica. Una posible forma de resolver esta limitación y extender la levitación a objetos con diferentes geometrías es a través de vórtices acústicos que pueden generarse al controlar la fase de arreglos de transductores que forman la cavidad acústica.

Una onda, que además de propagarse, rota en el tiempo, se conoce como onda helicoidal o vórtice y se define por una función de onda de tipo (ecuación 1):

$$\psi_H = A_m(\rho, z)cos(\omega t - kz \mp m\varphi) \qquad (1.1)$$

El factor m se conoce como carga topológica y físicamente corresponde al número de hélices enroscadas en el frente de onda. Es decir, $m = 1$ representa un vórtice con una sola hélice y $m = 2$ representa un vórtice con 2 hélices enroscadas, así sucesivamente (A. O. Santillán and K. Volke-Sepúlveda, 2009).

En este trabajo se propone construir un levitador acústico con la capacidad de generar vórtices acústicos de carga topológica m = 1 y ondas estacionarias diversas al variar de manera controlada la fase de una cavidad acústica formada por dos arreglos de transductores ultrasónicos de bajo costo. Posterior a la implementación, que involucra el ensamblaje del levitador, el diseño electrónico, la programación de fases, y una interfaz de usuario, se caracteriza cualitativamente la distribución de presión acústica generada en la trampa acústica respecto al control de fases.

1.4 Objetivos

1.4.1 Objetivo general

Implementar y caracterizar un sistema de levitación acústico de fase variable.

1.4.2 Objetivos específicos

I. Estudiar y comprender la física que involucra la levitación acústica.

II. Desarrollar una programación que se adecue a las especificaciones físicas planteadas, haciendo uso de un dispositivo programable FPGA.
III. Diseñar una interfaz de usuario para poder manipular las fases.
IV. Elaborar un manual de usuario del dispositivo.

1.5 Justificación

La implementación de un levitador acústico de fase variable podría, por ejemplo:

I. Transferir de manera controlada momento angular a los objetos levitados.

II. Levitar de manera estable objetos con geometrías no-esféricas.

III. Generar distribuciones de presión diferentes a las de una onda estacionaria convencional.

Estas características son importantes desde el punto de vista de las aplicaciones prácticas de la levitación acústica, ya que, en general, permitirían la levitación de objetos independientemente de su simetría. Al igual que, se obtendría una levitación más estable al poder anular el momento angular de la partícula levitada de geometrías específicas.

1.6 Hipótesis

El desarrollo de un sistema de levitación acústica con fase variable permitirá generar diferentes distribuciones de presión, incluyendo las distribuciones asociadas a vórtices acústicos. Estas distribuciones permitirán levitar simultáneamente múltiples partículas de manera estable fuera del eje de simetría, además de levitar objetos de simetría no-esférica.

1.7 Alcances

I. Se manipularán las fases relativas entre los transductores que componen el levitador acústico y que opera con señales periódicas cuadradas con frecuencias de 40 KHz.

II. Se elaborará un Interfaz de usuario eficaz para la manipulación de las variables de la FPGA.
III. Crear vórtices y trampas acústicas.
IV. Levitar objetos de cualquier geometría.

1.8 Limitaciones

I. Inmersión en lenguaje nuevo de programación (Python).
II. Tiempo de desarrollo del proyecto.
III. Acceso restringido a laboratorio debido a la pandemia actual (covid 19)

1.9 Metodología

A continuación, se presenta la metodología que se utilizará para el desarrollo completo de esta tesis, como se observa en la figura 1.1, se muestra las etapas con las que se dará solución a la problemática teniendo su descripción general de cada una.

1. Planteamiento del problema: En esta sección se plantea el problema general dando una solución con los conocimientos adquiridos a lo largo de la carrera.

2. Investigación documental: Compilar todas las fuentes de información posibles, se revisarán y compararán teniendo una selección de información de acuerdo a las necesidades del proyecto.

3. Diseño: Se desarrollarán los diseños de cada una de las etapas que conforma el sistema de levitación acústica abarcando desde el diseño del prototipo hasta el diseño de PCB e interfaz gráfica del usuario.

4. Simulación: En base al análisis y desarrollo de las etapas se tendrá que corroborar antes de poder pasar a la implementación previniendo posibles fallas, haciendo simulaciones respectivas tanto de software como de hardware si los resultados de las simulaciones no son las esperadas se

tendrá que pasar a la etapa 2 teniendo una retroalimentación y de esta forma aplicar correcciones hasta que los resultados sean los deseados.

5. Implementación y pruebas de las diferentes etapas del sistema: Se realizará un plan de pruebas en donde se evalúe el desempeño de cada una de las etapas.

6. Documentación: Comprobando que el sistema es capaz de entregar resultados satisfactorios de acuerdo con la problemática a resolver se documentara el desarrollo de dicha solución, en caso de no ser los resultados esperados se reiniciara a la parte de diseño, con el fin de determinar la falla y aplicar correcciones hasta que los resultados sean satisfactorios.

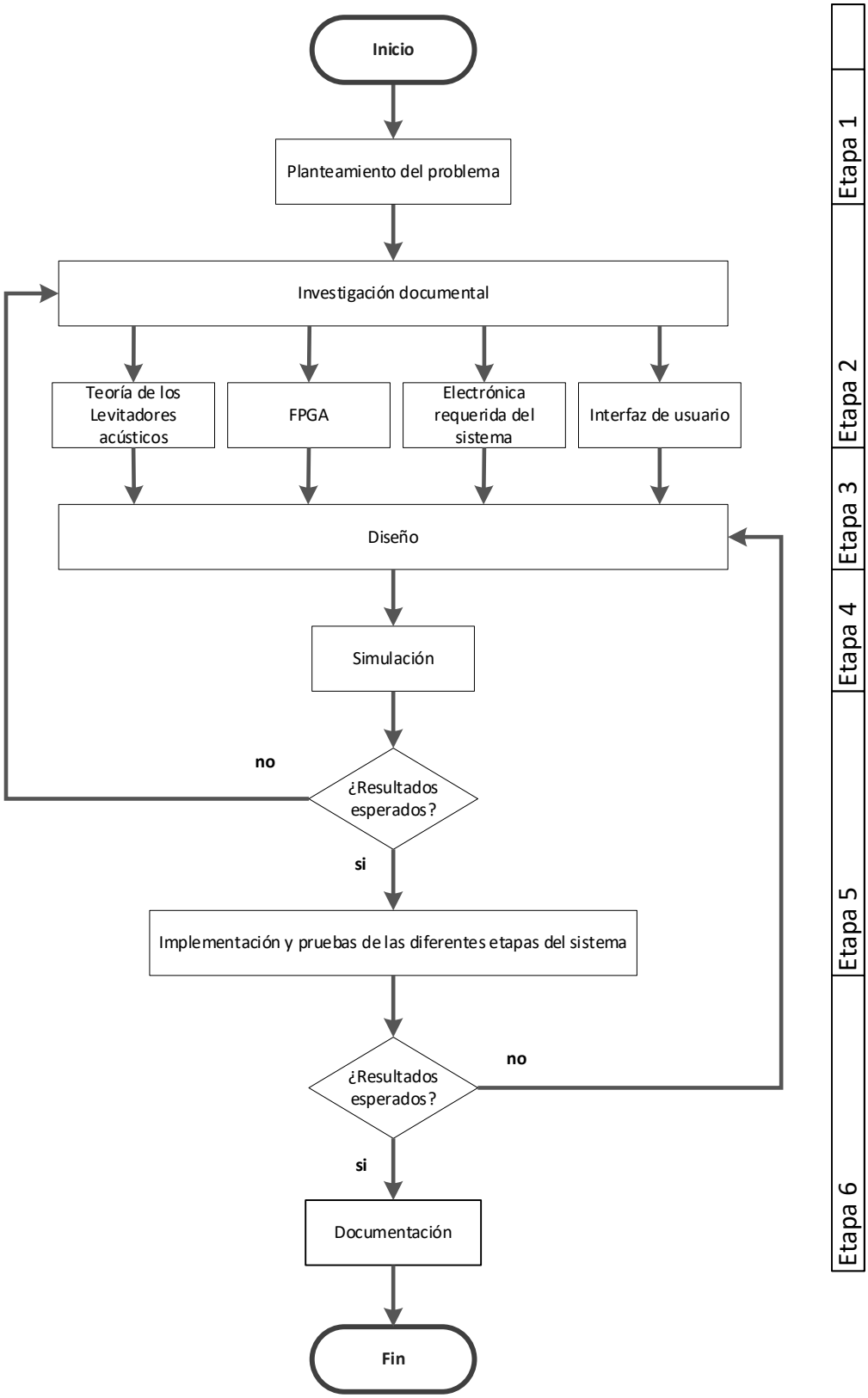


Figura 1.1 Diagrama de flujo de la metodología.

1.10 Organización de la tesis

La presente tesis se compone de seis capítulos:

- Capítulo uno: Introducción.
  Este capítulo uno se declaran las características del proyecto a desarrollar, así como los trabajos similares que se han publicado hasta el momento, la manera en que se abordará el problema.

- Capítulo dos: Marco teórico.
  En este segundo capítulo tiene el objetivo de exponer los conceptos fundamentales de la levitación acústica, aplicaciones, características, ventajas y desventajas, así como las herramientas a considerar para un correcto diseño de prototipo.

- Capítulo tres: Implantación de los modelos.
  A lo largo del capítulo se muestran los pasos a seguir durante el desarrollo del prototipo final.

- Capítulo cuatro: Pruebas y resultados.
  En el capítulo cuatro se muestra detalladamente los resultados del prototipo final, presentando el levitador acústico con su correspondiente manual de usuario.

- Capítulo cinco: Conclusiones y trabajos a futuro.
  En el último capítulo que conforma a la tesis se expresan las conclusiones a las que se llegaron, la comparación con la hipótesis planteada al comienzo de la misma y se exponen una serie de trabajos a futuro.

2. Capítulo II:

MARCO TEÓRICO

2.1 Introducción

En este capítulo se muestran los elementos teóricos necesarios para el desarrollo del presente proyecto. Es importante conocer cada una de las partes que conforman un sistema de levitación acústica con fase variable. Se conocerán algunos conceptos importantes, así como una breve historia de la levitación, además se presentan las características de las tecnologías aplicadas en este proyecto, desde software utilizado para el control de las señales generadas y para la implementación de la interfaz de usuario, hasta el hardware implementado.

2.2 Levitación

La levitación es el proceso de suspender objetos en un fluido, generalmente aire, sin ningún soporte o contacto mecánico. Existen diferentes técnicas de levitación, entre ellas, magnética, electrostática, aerodinámica, óptica y levitación acústica, Cada una de estas tecnologías presenta características únicas. Por ejemplo: La levitación óptica se basa en la transferencia del impulso de los fotones y es capaz de manipular células y átomos (Karen Volke Sepúlveda, 2007).

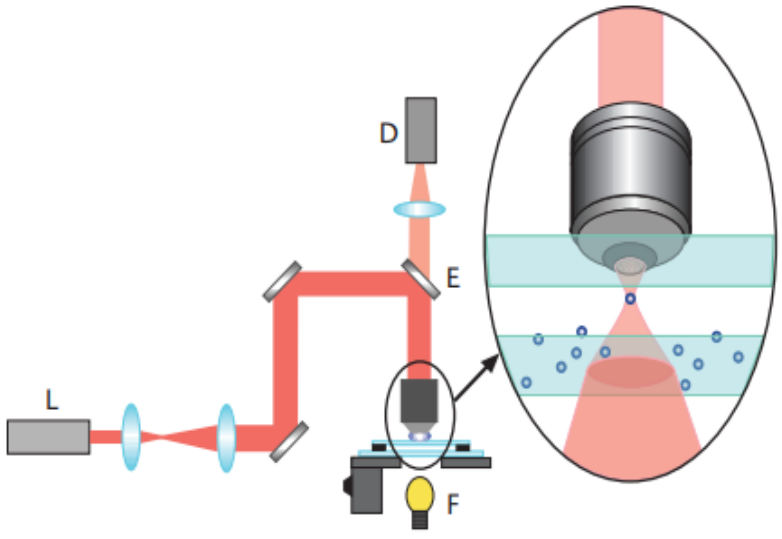


Figura 2.1 Arreglo experimental simple de pinzas ópticas, mostrado el sistema de levitación óptica.

La levitación magnética se ha utilizado en HCI y fue ejemplificada por el sistema llamado ZeroN (J. Lee, 2011)donde una esfera magnética de 3 cm de diámetro fue levitada en un espacio de interacción de 20 cm. Sin embargo, el control magnético es para una de las dimensiones, solo se puede controlar un objeto levitado y las propiedades materiales del objeto tangible son limitadas ya que debe ser magnético.

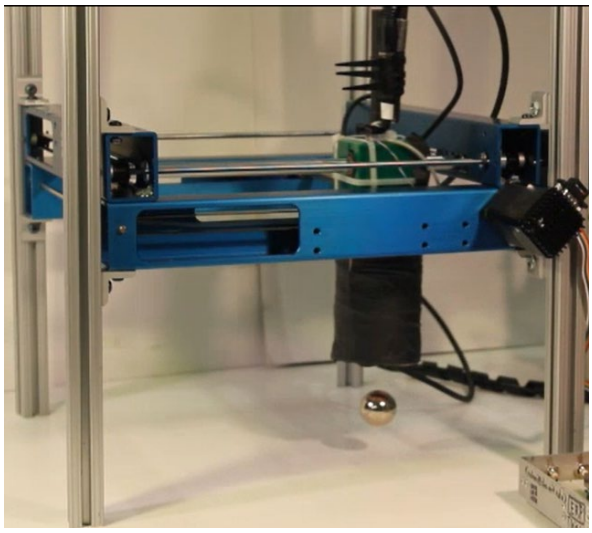


Figura 2.2 Sistema de levitación magnética (ZeroN).

2.3 Levitación acústica

El principio de un levitador acústico es la generación de una onda estacionaria desde un emisor, llamado transductor, hacia un reflector separados por una distancia múltiplo entero de media longitud de onda. Esto produce reflexiones e interacción entre todas las ondas generadas, creando máximos y mínimos de presión y puntos nulos de presión (nodos), la partícula situada en ese último punto levitará alcanzando el equilibrio estático.

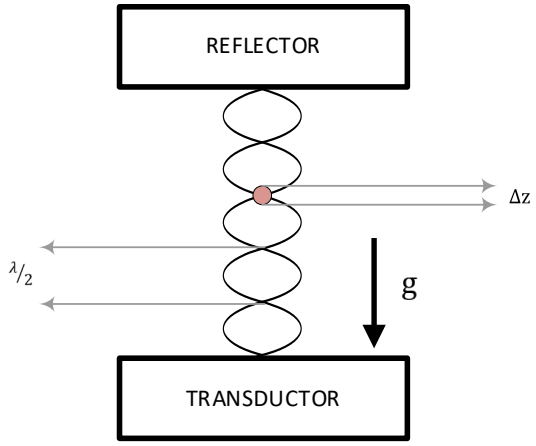


Figura 2.3 Esquema de un levitador de onda estacionaria entre un reflector y un transductor.

Donde:

g = la gravedad que actúa sobre la masa de la partícula

λ = longitud de onda

$\lambda/2$ = media longitud de onda, tamaño máximo de la partícula a levitar

Δz = es el eje el cual debería levitar la partícula

La distancia Δz puede ser calculada en función de las propiedades de la partícula y la onda estacionaria.

2.3.1 Topologías de levitadores acústicos

Los levitadores más comunes son los de eje único, la clasificación puede extenderse a cinco tipos (figura 2.6) (Andrade, 2017):

a) Levitación acústica de onda estacionaria
b) Levitación de campo lejano
c) Levitación de campo cercano
d) Levitación de campo cercano invertida
e) Levitación de un sólo haz

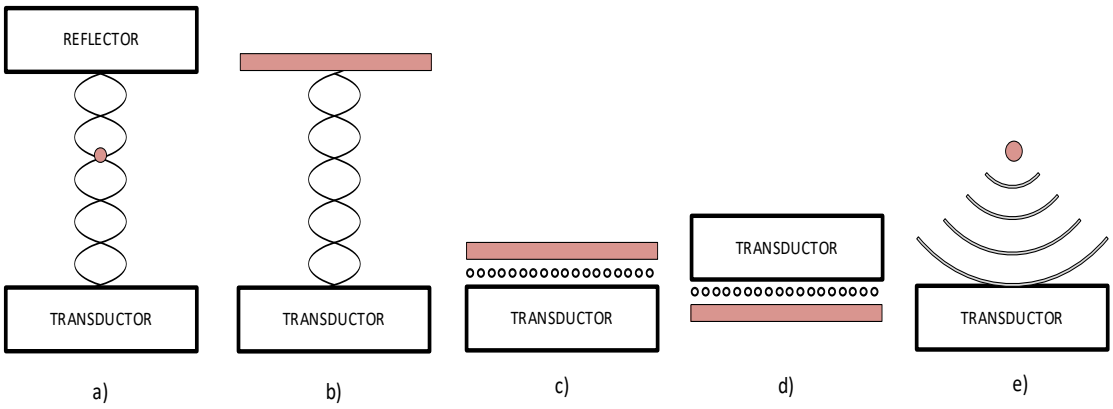


Figura 2.4 Clasificación de los levitadores acústicos.

2.3.2 Clasificación de levitadores uniaxiales por geometría

Los sistemas de levitación acústica uniaxiales se pueden clasificar por su geometría de la superficie de transductor y reflector como:

a) Plano-plano: en el cual ambos transductor y reflector tienen superficie con geometría plana.
b) Plano-cóncavo: donde transductor y reflector tienen superficie con geometría plana y cóncava, respectivamente.
c) Cóncavo – cóncavo: en esta configuración transductor y reflector, ambos tienen superficie con geometría cóncava.

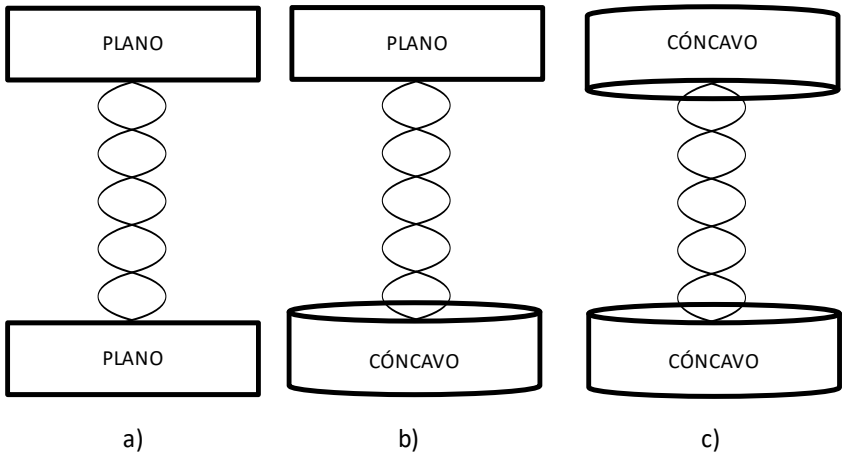


Figura 2.5 Clasificación de levitadores de acuerdo a su geometría.

En la actualidad, la configuración de levitadores acústicos más común consta de un transductor comercial de tipo Langevin, conectado a un amplificador mecánico cóncavo y un reflector cóncavo permite levitar partículas en el nodo de una onda acústica estacionaria producida por el levitador uniaxial, este tipo de levitador tiene varias limitantes una de ellas es que opera a altos voltajes y son muy sensibles a al cambio de temperatura debido al calentamiento del transductor tipo Langevin. (Lin, 1995), en la imagen siguiente se muestra un sistema de levitación acústica uniaxial con el que se trabaja en el Laboratorio de óptica aplicada (Instituto de Ciencias Físicas, UNAM, 2020)

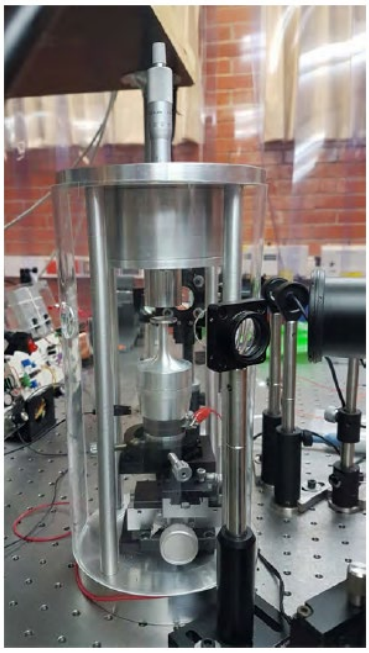


Figura 2.6 Sistema de levitación acústica con cavidad resonante tipo Langevin.

2.4 Potencial acústico

Lev Gor'kov determinó la fuerza promedio que actúa sobre una partícula en un campo acústico, cuando el tamaño de la partícula es mucho menor que la longitud de onda del sonido. Esta fuerza resulta de un flujo de momento no uniforme en la región de campo cercano alrededor de la partícula. Es el resultado de las ondas acústicas entrantes y su esparcimiento en la superficie de la partícula cuando las ondas acústicas se propagan a través de ella (Andrade, 2017).

La energía potencial acústica U se calcula de la siguiente manera:

$$U = V_o \left[\left(\frac{f_1}{3\rho_0 c_0^2} \right) \langle \left(p_1^{in} \right)^2 \rangle - \frac{f_2 \rho_0}{2} \langle \boldsymbol{u}_1^{in} \cdot \boldsymbol{u}_1^{in} \rangle \right] \quad (2.1)$$

Donde:

V_o = es el volumen de la partícula

p_1^{in} = es la presión acústica incidente de primer orden

$\boldsymbol{u}_1^{in}$ = es la velocidad de las partículas en la posición de la esfera

Los coeficientes $f1$ y $f2$ dependen, respectivamente de las propiedades de los resonadores monopolo y dipolo modelados como dos sistemas de resorte-masa. Dados por las siguientes formulas:

$$f1 = 1 - \frac{\rho_0 c_0^2}{\rho_p c_p^2} \quad (2.2)$$

$$f2 = 2 \cdot \left(\frac{\rho_p - \rho_0}{2\rho_p + \rho_0} \right) \quad (2.3)$$

Del potencial U (también llamado potencial Gor'kov), la fuerza sobre la esfera se calcula por:

$$\boldsymbol{F_{rad}} = -\nabla U \quad (2.4)$$

Para aplicar la expresión de Gor'kov para calcular la fuerza de radiación acústica en una partícula esférica rígida de radio R << λ, debemos encontrar la presión acústica de primer orden (p_1^{in}) y las distribuciones de velocidad de partícula entre el transductor y el reflector.

En una onda estacionaria acústica, la presión acústica de primer orden incidente (p_1^{in}) se puede describir mediante:

$$p_1^{in} = \cos(\omega t)\cos(kz) \quad (2.5)$$

y la velocidad de la partícula ($\boldsymbol{u}_1^{in}$):

$$\boldsymbol{u}_1^{in} = \frac{\rho_0}{\rho_0 c_0} \operatorname{sen}(\omega t) \operatorname{sen}(kz)\ \hat{k} \tag{2.6}$$

Considerando $f1 = f2 = 1$ y al considerar una presión descrita por una función senoidal (ecuación 2.5 y 2.6) se obtiene el potencial, dada por la siguiente formula:

$$U = \frac{\rho_0^2 \pi R^3}{\rho_0 c_0^2} \left[\frac{\cos^2(kz)}{3} - \frac{\operatorname{sen}^2(kz)}{2} \right] \tag{2.7}$$

Finalmente, la fuerza de radiación acústica es:

$$\boldsymbol{F_{rad}} = \frac{5\pi R^3 k \rho_0^2}{6\rho_0 c_0^2} \tag{2.8}$$

2.5 Ondas giratorias

Las ondas giratorias y helicoidales también se denominan vórtices de ondas porque sus distribuciones de flujo de energía y densidad de momento lineal en el plano transversal se asemejan al campo de velocidad de un fluido de vórtice.

Considerando ondas monocromáticas que se propagan en un medio lineal, caracterizado por su frecuencia angular ω, longitud de onda λ y la velocidad de propagación o fase:

$$v = \frac{\omega}{k} \tag{2.9}$$

Donde:

$k = \frac{2\pi}{\lambda}$ es el número de onda

$T = \frac{2\pi}{\omega}$ es el período

Las ondas monocromáticas no presentan dispersión y cuando se propagan en un medio lineal, tienen una velocidad constante que no depende de su amplitud. Todas estas condiciones pueden cumplirse para el sonido y las ondas de luz láser continuas que se propagan en el aire. Si una onda viaja hacia adelante al mismo tiempo que gira, la onda se conoce como helicoidal y su función de onda tiene la forma:

$$\psi_H = A_m(\rho, z)cos(\omega t - kz \mp m\varphi) \tag{2.10}$$

Donde:

$$\varphi = arctan\left(\frac{x}{y}\right) \tag{2.11}$$

$$\rho = (x^2 + y^2)^{1/2} \tag{2.12}$$

El factor m se conoce como carga topológica y físicamente corresponde al número de hélices enroscadas en el frente de onda. Es decir, $m = 1$ representa un vórtice con una sola hélice y $m = 2$ representa un vórtice con 2 hélices enroscadas, así sucesivamente (A. O. Santillán and K. Volke-Sepúlveda, 2009).

La superposición de dos ondas contrarrotantes de la misma amplitud y frecuencia da lugar a ondas estacionarias de la forma

$$\psi_S(x, y, t) = 2A_m(\rho)\, cosm\varphi\, cos\omega t \tag{2.13}$$

En este caso $m = 2$, donde hay nodos definidos por la condición:

$$m\varphi = \frac{(2n-1)\pi}{2} \tag{2.14}$$

Donde n es un número entero con valores de $m = 1, 2$.

La interferencia de dos ondas estacionarias ortogonales que oscilan alrededor de sus superficies nodales con un cambio de fase relativo de $\mp \frac{\pi}{2}$ produce una onda giratoria representada por:

$$\psi_R(x, y, t) = A_m(\rho) \left[cosm\varphi \, cos\omega t + senm\varphi \, cos\left(\omega t \mp \frac{\pi}{2}\right)\right] \quad (2.15)$$

$$\psi_R(x, y, t) = A_m(\rho) \, cos(\omega t \mp m\varphi) \quad (2.16)$$

El signo negativo del cambio de fase (retraso de fase) da lugar a una onda que gira en el sentido contrario a las agujas del reloj, mientras que un cambio de fase positivo da como resultado una rotación en el sentido de las agujas del reloj. (A. O. Santillán and K. Volke-Sepúlveda, 2009)

2.5.1 Momento angular

Una de las propiedades más interesantes de las ondas giratorias o vórtices es que llevan un momento angular que se puede transferir a la materia, al igual que el momento lineal de las ondas viajeras. Esta propiedad ha atraído una atención significativa debido a la posibilidad de controlar la rotación de partículas.

El momento angular de las ondas acústicas en rotación también se puede transferir a la materia. El componente z de la densidad del momento angular para una onda de sonido que gira alrededor de este eje es:

$$L_z = \mp \left(\frac{m}{\omega}\right) \; p^2 \, / \, \rho_0 v^2 \quad (2.17)$$

Donde:

$p^2 \, / \, \rho_0 v^2$ es el doble de la densidad de energía potencial acústica

p es la presión del sonido

ρ_0 es la densidad media del medio de propagación

v es la velocidad del sonido

Si la densidad de energía total se distribuye por igual entre las contribuciones potencial y cinética, podemos escribir el momento angular como:

$$L_z = \mp \left(\frac{m}{\omega}\right) \varepsilon \tag{2.18}$$

Esta es la misma expresión para las ondas en óptica y mecánica cuántica. (A. O. Santillán and K. Volke-Sepúlveda, 2009)

2.6 Field Programmable Gate Arrays (FPGAS)

Las Field Programmable Gate Arrays (FPGAS) son un tipo de PLD (Programmable Logic Device), es decir, son dispositivos programables de propósito general compuestos por un conjunto de elementos lógicos sobre los que el usuario puede implementar su circuito. Se diferencian de otros dispositivos programables como los PAL o los PLA en que no hacen uso de planos de compuertas AND u OR para implementar los circuitos, sino que en su lugar proporcionan bloques lógicos para esta tarea y además a diferencia del resto, permiten implementar circuitos que contienen un gran número de puertas lógicas.

A grandes rasgos, la arquitectura de una FPGA está compuesta por bloques lógicos, bloques de E/S y elementos de interconexión para la conexión entre bloques. Los bloques lógicos son organizados en forma de matriz bidimensional y los elementos de interconexión, compuestos de “cables” y switches, se organizan vertical y horizontalmente. (C. Beckhoff, 2012)

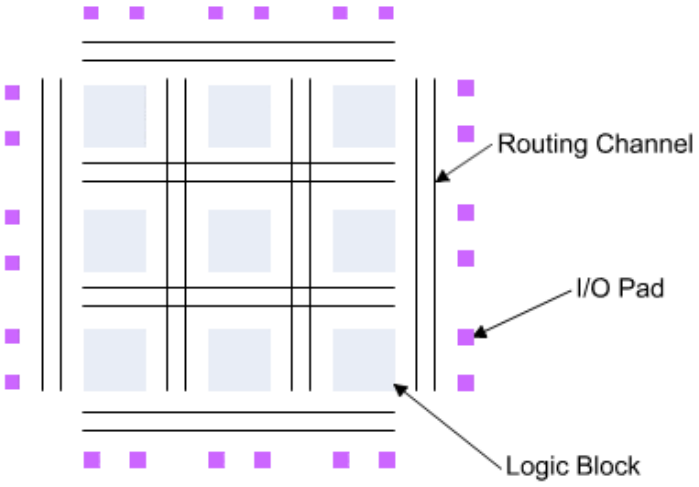


Figura 2.7 Estructura genérica de una FPGA.

2.6.1 FPGA ALTERA Cyclone IV

CoreEP4CE6 es una placa base FPGA que cuenta con un dispositivo EP4CE6E22C8N a bordo, admite una mayor expansión. (waveshare, 2015)

- Dispositivo de configuración serial integrado EPCS16SI8N.
- Circuito básico FPGA integrado, como circuito de reloj.
- Botón nCONFIG integrado, botón RESET, 4 x LED.
- Todos los puertos de E/S son accesibles en los encabezados de los pines
- Interfaz de programación / depuración JTAG integrada.
- Diseño de paso de cabezal de 2,54 mm, adecuado para enchufar su sistema de aplicación.

EP4CE6E22C8N: el dispositivo FPGA ALTERA Cyclone IV que cuenta con las siguientes especificaciones:

Tabla 1 Especificaciones FPGA ALTERA Cyclone IV.

Especificaciones	
Frecuencia de funcionamiento	50 MHz
Voltaje de funcionamiento	1.15 V ~ 3.465 V.
Paquete	QFP144
Entradas/Salidas	80
Elementos Lógicos	6K
RAM	270 Kb
PLL	2
Depuración / Programación	compatible con JTAG
Oscilador de cristal activo	50 MHz
Toma de corriente continua	5 V

Figura 2.8 FPGA Cyclone Core EP4CE6E22C8N.

2.7 Quartus II

Quartus II es una herramienta de software producida por Altera para el análisis y la síntesis de diseños realizados en HDL. Quartus II permite al desarrollador compilar

sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador.

2.7.1 Quartus II Edición Web

La Edición Web es una versión gratuita de Quartus II que puede ser descargada o enviada gratuitamente por correo electrónico. Esta edición permite la compilación y la programación de un número limitado de dispositivos Altera.

La familia de FPGAs de bajo coste Cyclone, está soportada por esta edición, por lo que los pequeños desarrolladores y desarrolladoras no tendrán problemas por el coste del desarrollo de software. (Intel, 2013)

Figura 2.9 Logo de software Quartus II.

2.8 Tipos de comunicación

En los canales de comunicación existen tres tipos de transmisión.

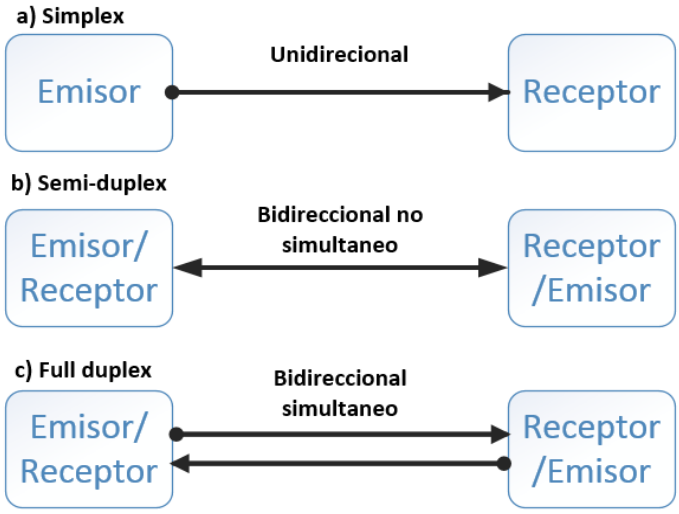


Figura 2.10 Diagrama de los tipos de comunicación.

- Simplex: En este caso el transmisor y el receptor están perfectamente definidos y la comunicación es unidireccional. Este tipo de comunicaciones se emplean usualmente en redes de radiodifusión, donde los receptores no necesitan enviar ningún tipo de dato al transmisor.
- Duplex o Semi-duplex: En este caso ambos extremos del sistema de comunicación cumplen funciones de transmisor y receptor y los datos se desplazan en ambos sentidos, pero no simultáneamente. Este tipo de comunicación se utiliza habitualmente en la interacción entre terminales y un computador central.
- Full Duplex: El sistema es similar al duplex, pero los datos se desplazan en ambos sentidos simultáneamente. Para ello ambos transmisores poseen diferentes frecuencias de transmisión o dos caminos de comunicación separados, mientras que la comunicación semi-duplex necesita normalmente uno solo.

Para el intercambio de datos entre computadores este tipo de comunicaciones son más eficientes que las transmisiones semi-duplex. (textoscientificos.com, s.f.)

2.9 Comunicación serial

Es un protocolo muy común (no hay que confundirlo con el Bus Serial de Comunicación, o USB) para comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora. El concepto de comunicación serial es sencillo. El puerto serial envía y recibe bytes de información un bit a la vez.

Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión:

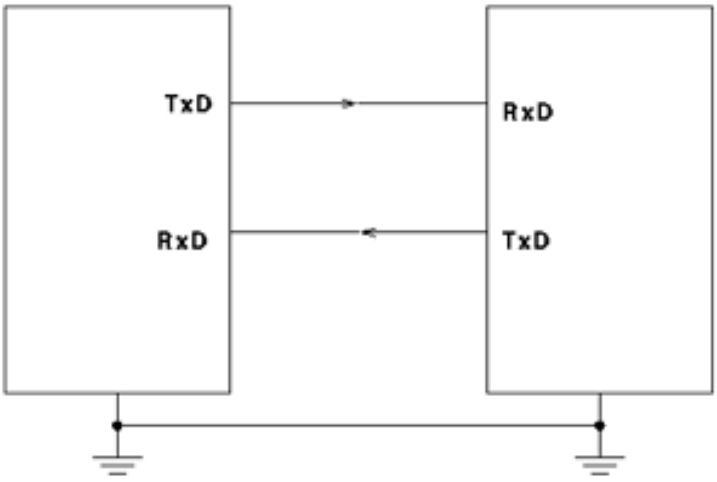


Figura 2.11 Diagrama a bloques de Comunicación asincrónica.

- GND (tierra o referencia)
- TX (transmitir)
- RX (Recibir)

Debido a que la transmisión es asincrónica, es posible enviar datos por una línea mientras se reciben datos por otra. Las características más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad. Para que dos puertos se puedan comunicar, es necesario que las características sean iguales.

- Velocidad de transmisión (baud rate): Indica el número de símbolos por segundo que se transfieren.
- Bits de datos: Se refiere a la cantidad de bits en la transmisión. Cuando la computadora envía un paquete de información, el tamaño de ese paquete no necesariamente será de 8 bits. Las cantidades más comunes de bits por paquete son 5, 7 y 8 bits. El número de bits que se envía depende en el tipo de información que se transfiere. Por ejemplo, el ASCII estándar tiene un rango de 0 a 127, es decir, utiliza 7 bits; para ASCII extendido es de 0 a 255, lo que utiliza 8 bits.
- Bits de parada: Usado para indicar el fin de la comunicación de un solo paquete. Los valores típicos son 1, 1.5 o 2 bits

- Paridad: Es una forma sencilla de verificar si hay errores en la transmisión serial. Existen cuatro tipos de paridad: par, impar, marcada y espaciada. La opción de no usar paridad alguna también está disponible. (Igoe, 2020)

En la actualidad existen diferentes ejemplos de puertos que comunican información de manera serial (un bit a la vez). El conocido como “puerto serial” ha sido gradualmente reemplazado por el puerto USB que permite mayor versatilidad en la conexión de múltiples dispositivos.

2.9.1 Conversor USB a Serial TTL CP2102

El conversor CP2102 facilita la comunicación entre una PC y un microcontrolador utilizando el protocolo USB. Funciona de forma similar a los conversores FTDI232 y PL2303HX, con la ventaja de tener un mejor precio y mayor soporte de drivers. Además, puede funcionar como "programador" de microcontroladores, pues incluye el pin DTR o RESET necesario para cargar fácilmente un programa al microcontrolador.

Al utilizar el conversor USB se facilita la integración de nuestros proyectos con programas de PC, desde el punto de vista del programador del microcontrolador el conversor es "transparente", solo necesitamos usar el clásico protocolo serial UART dejando la complejidad del protocolo USB.

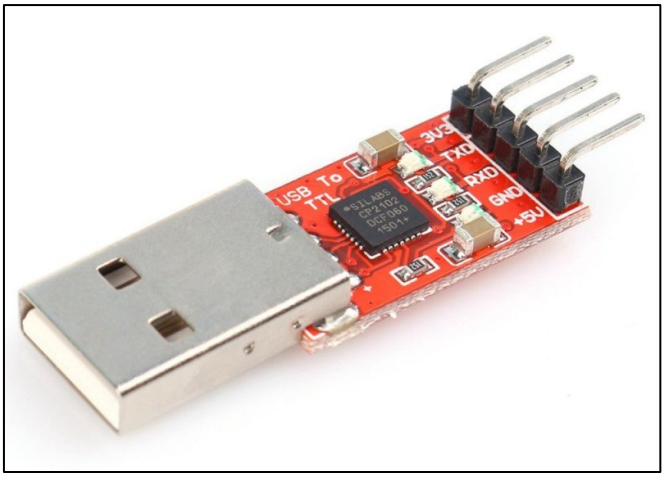


Figura 2.12 Conversor USB a Serial TTL CP2102.

Tabla 2 Especificaciones del conversor USB a Serial TTL CP2102.

Especificaciones	
Conector USB	USB tipo A
Pines salida (TTL)	+3.3V, RST, TXD, RXD, GND y + 5V
Transceiver USB	Integrado
Cristal oscilador	Integrado
Regulador de voltaje	3.3V interno
Buffer de recepción	576 Bytes
Buffer de transmisión	640 Bytes
Temperatura de trabajo	-40° a 80°C
Sistemas Operativos soportados	Windows 10, 8, Vista, 7, XP, 2000, 98SE y Linux 2.40 (en adelante)

2.10 Python

Python es un gran lenguaje de programación interactivo, orientado a objetos, interpretado. A menudo se compara (de forma favorable, por supuesto) con Lisp, Tcl, Perl, Ruby, C #, Visual Basic, Visual Fox Pro, Scheme o Java.

Python combina un poder notable con una sintaxis muy clara. Tiene módulos, clases, excepciones, tipos de datos dinámicos de muy alto nivel y escritura dinámica. Existen interfaces para muchas bibliotecas y llamadas de sistema, así como para varios sistemas de ventanas. Los nuevos módulos incorporados se escriben fácilmente en C o C ++ (u otros idiomas, según la implementación elegida). Python también se puede usar como un lenguaje de extensión para aplicaciones escritas en otros idiomas que necesitan scripts o interfaces de automatización fáciles de usar. (Python, 2021)

Figura 2.13 Logo de Python

2.10.1 PyQt5

Qt es un conjunto de bibliotecas C ++ multiplataforma que implementan API de alto nivel para acceder a muchos aspectos de los sistemas móviles y de escritorio modernos. Estos incluyen servicios de ubicación y posicionamiento, conectividad multimedia, NFC y Bluetooth, un navegador web basado en Chromium, así como el desarrollo de IU tradicional.

PyQt5 es un conjunto completo de enlaces de Python para Qt v5. Se implementa como más de 35 módulos de extensión y permite que Python se utilice como un lenguaje de desarrollo de aplicaciones alternativo a C ++ en todas las plataformas compatibles, incluidas iOS y Android.

PyQt5 también puede integrarse en aplicaciones basadas en C ++ para permitir a los usuarios de esas aplicaciones configurar o mejorar la funcionalidad de esas aplicaciones. (Limited, 2021)

2.11 Qt Designer

Qt Designer es la herramienta Qt para diseñar y construir interfaces gráficas de usuario (GUI) con Qt Widgets. Puede componer y personalizar sus ventanas o cuadros de diálogo en una forma de lo que ve es lo que obtiene y probarlos usando diferentes estilos y resoluciones.

Los widgets y formularios creados con Qt Designer se integran a la perfección con el código programado, utilizando el mecanismo de ranuras y señales de Qt, para que pueda asignar fácilmente el comportamiento a los elementos gráficos. Todas las propiedades establecidas en Qt Designer se pueden cambiar dinámicamente dentro del código. Además, las funciones como la promoción de widgets y los complementos personalizados le permiten usar sus propios componentes con Qt Designer. (Company, 2021)

Figura 2.14 Logo de Qt Designer.

3. Capítulo III: IMPLANTACIÓN DE LOS MODELOS

3.1 Introducción

En este apartado se describe los requisitos del sistema correspondiente al levitador acústico uniaxial de fase variable; además, se presenta la metodología usada para llevar a cabo las distintas etapas del proyecto abarcando desde la interfaz gráfica de usuario hasta los arreglos de transductores. Sin embargo, el diseño y algunas características principales se han adecuado de acuerdo con los requerimientos necesarios para su implementación.

3.2 Requisitos del sistema de levitación acústica

Los requisitos del sistema son:

3.2.1 Restricciones de diseño

RD1. Situación actual.

Debido los tiempos que estamos viviendo, la contingencia por el COVID-19 no se podrá asistir al laboratorio.

3.2.2 Requisitos funcionales nominales

FN1. Fuente de alimentación variable.

Debe operar desde un valor de 4 V hasta 35 V para proteger la integridad de los amplificadores, siendo el usuario capaz de manipular el voltaje de operación.

FN2. Menú de interacción con el usuario.

El usuario será capaz de cambiar las fases en cada uno de los segmentos que conforma el sistema.

FN3. Conexiones del dispositivo.

Utilizando conectores de fácil manipulación se podrá conectar los segmentos de los arreglos de transductores.

3.2.3 Requisitos funcionales no nominales

FF1. Interruptor con indicador de funcionamiento.

Indica si el sistema está en funcionamiento, permitiendo ver cuando esta operando o no el sistema.

3.2.4 Requisitos de interfaz

IN1. Interfaz de usuario.

Se desarrollará en Python una GUI (Interfaz Gráfica de Usuario) que muestre un menú donde el usuario pueda escoger el desfase y segmento deseado para operar el sistema, así como mostrara el desfase que tiene cada segmento en tiempo real.

IN2. UART.

La transmisión de datos estará utilizando un módulo UART externo al FPGA.

3.2.5 Requisitos de calidad

CA1. Señales a la frecuencia exacta.

Con el objetivo de tener un sistema en un correcto funcionamiento, se generarán señales exactas de 40 KHz.

CA2. Desfase correcto.

Se generarán los desfases de la señal de 40 KHz, teniendo una resolución de desfase de $\frac{\pi}{2}$ exacto para poder evitar problemas tanto en la interfaz y la implementación del prototipo.

3.2.6 Requisitos de soporte

SO1. El usuario no deberá remover el módulo UART, es decir el conversor USB a Serial TTL CP2102 siempre debe estar conectada a la PC.

3.2.7 Requisitos evolutivos

RE1. Implementar el sistema en PCB

El sistema deberá estar presentado en PCB para su implementación final, con su respectivo gabinete para facilitar su portabilidad.

RE2. Ejecutable de la GUI.

Atendiendo a la portabilidad, la interfaz deberá tener su archivo ejecutable evitando la instalación de software extra.

3.3 Etapas del sistema de levitación acústica

El sistema de levitación acústica de fase variable consta de 4 etapas, como se puede observar en la Figura 3.11. A continuación, se describen cada una de las etapas.

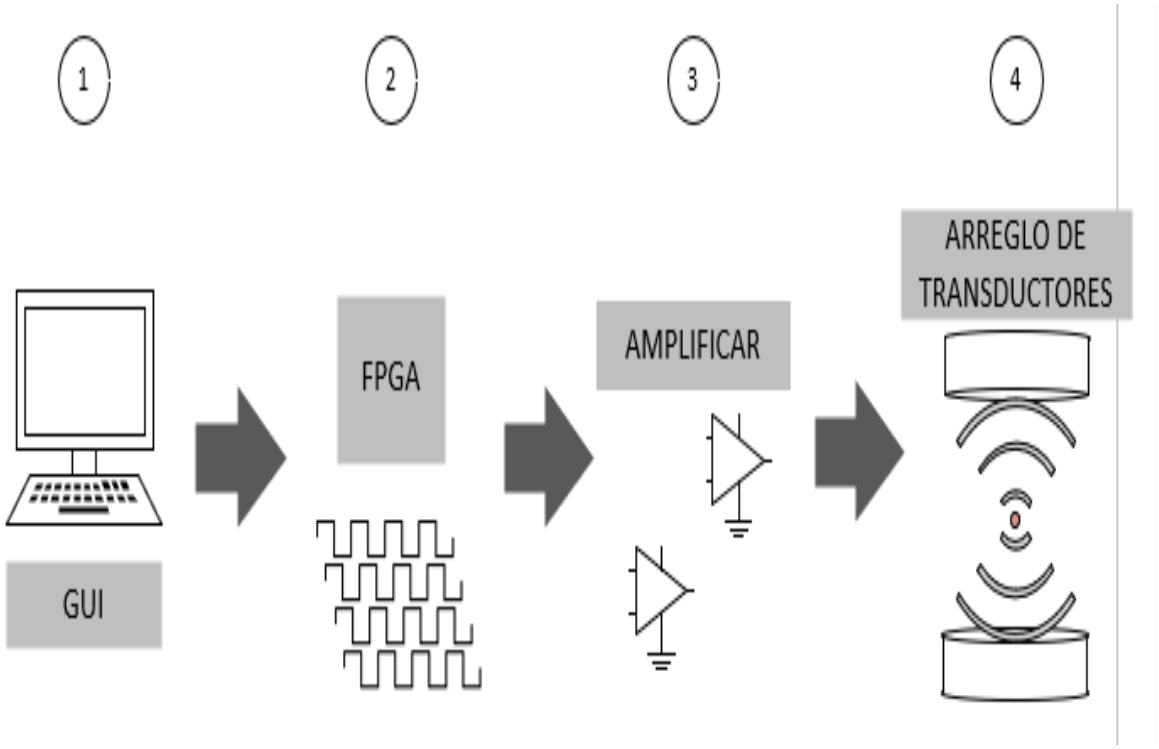


Figura 3.1 Diagrama general del proyecto

Etapa 1: Interfaz gráfica de usuario

Esta etapa es el medio por el cual el usuario podrá interactuar y manipular con los cambios de fase (desde un punto gráfico) de cada segmento de los arreglos de transductores teniendo una resolución de $\pi/2$ entre cada desfase.

Etapa 2: Control de señales

Dicha etapa se concentra en generar las señales necesarias para el sistema, así mismo de interpretar y procesar los datos recibidos desde la interfaz gráfica de usuario. Al igual que también se encarga de controlar las diferentes salidas que portaran las señales desfasadas.

Etapa 3: Amplificadores

Tiene la funcionalidad de amplificar las señales que proviene de la FPGA teniendo una potencia demasiada baja para poder operar en óptimas condiciones a los transductores.

Etapa 4: Arreglo de transductores

En este último apartado se describe y se caracterizan los arreglos de transductores que incorpora el sistema tanto inferior y superior, así también se analizan los transductores a utilizar.

3.3.1 Etapa 1: Interfaz gráfica de usuario

En la figura 3.2 se muestra la interfaz gráfica de usuario que se implementó para este proyecto, la cual permite una interacción intuitiva del usuario con un dispositivo a través de widgets. La GUI está diseñada pensando en el usuario por cuestiones estéticas, que sea clara, comprensible y fácil de utilizar.

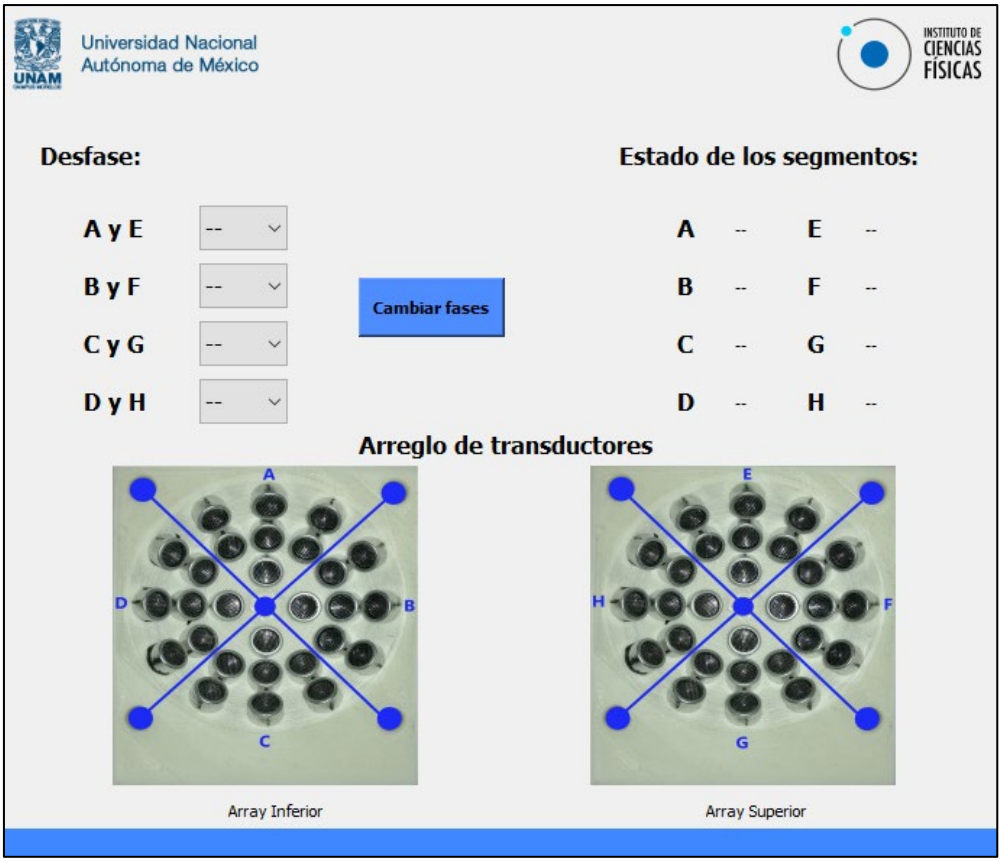


Figura 3.2 Interfaz Gráfica de Usuario.

El uso de dicha interfaz nos permitirá controlar las diferentes fases y poder manipular a conveniencia del usuario. La GUI consta de tres partes importantes:

1. Desfase: En esta sección se tiene los 8 segmentos de transductores a controlar (A-H) en el cual se puede seleccionar una fase de 0°, 90°, 180° y 270° como se puede observar en la Figura 3.3.

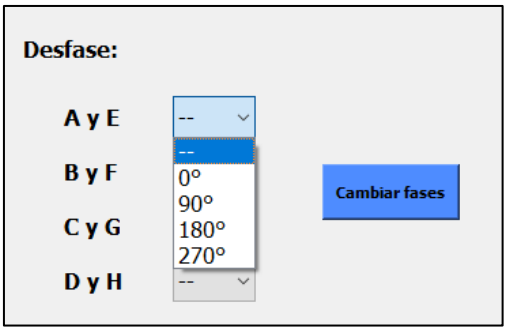
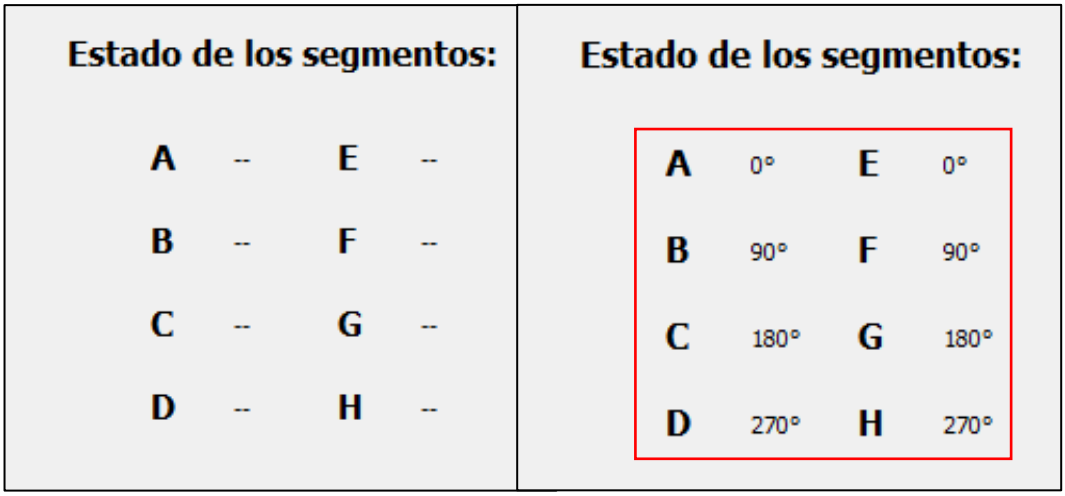


Figura 3.3 Selección de fases GUI.

2. Estado de los segmentos: se incluyen indicadores los cuales confirman que los datos se enviaron correctamente a la FPGA, arrojando la fase que se tiene en tipo real de cada segmento en una etiqueta.



a) *b)*

Figura 3.4 Indicadores de fase GUI, a) Etiquetas sin valor alguno, b) Etiquetas con la fase enviada.

3. Arreglo de transductores: en esta última sección se tiene dos imágenes de referencia de los arreglos de transductores, definiendo el arreglo superior e inferior y sus segmentos que los compone, como se observa de la imagen 3.4.

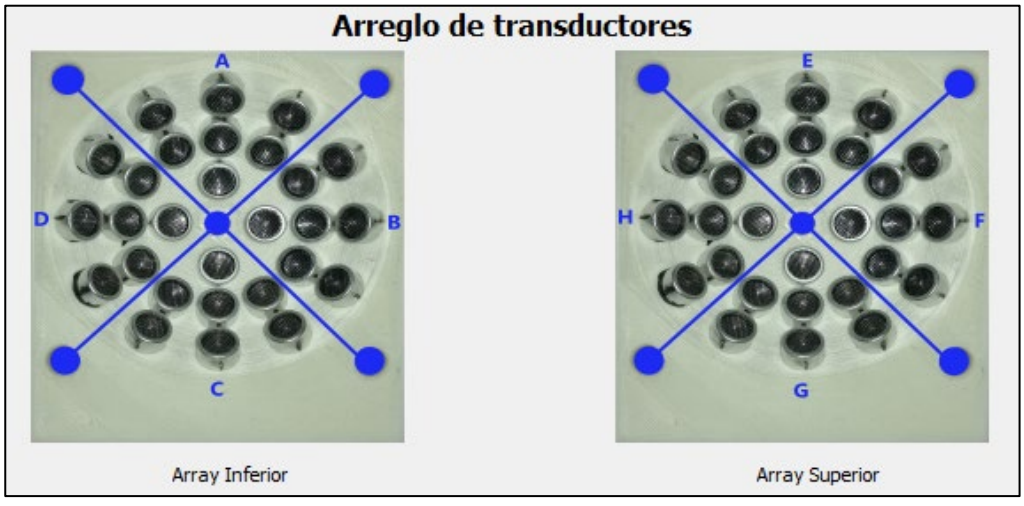


Figura 3.5 Arreglo de los transductores GUI.

3.3.1.1 Programación

En el desarrollo e implementación de la GUI se utilizó el lenguaje de programación Python y los paquetes de PyQt5 (conceptos vistos en el capítulo 2), librería principal para poder desarrollar interfaz graficas. Al igual que se utilizó el software Qt Designer para poder diseñar los gráficos.

Qt Designer nos permite y nos facilita diseñar los gráficos, es decir la vista estética de la interfaz agregando widgets, botones, etiquetas, imágenes etc. En la Figura 3.5 se muestra el menú principal del software.

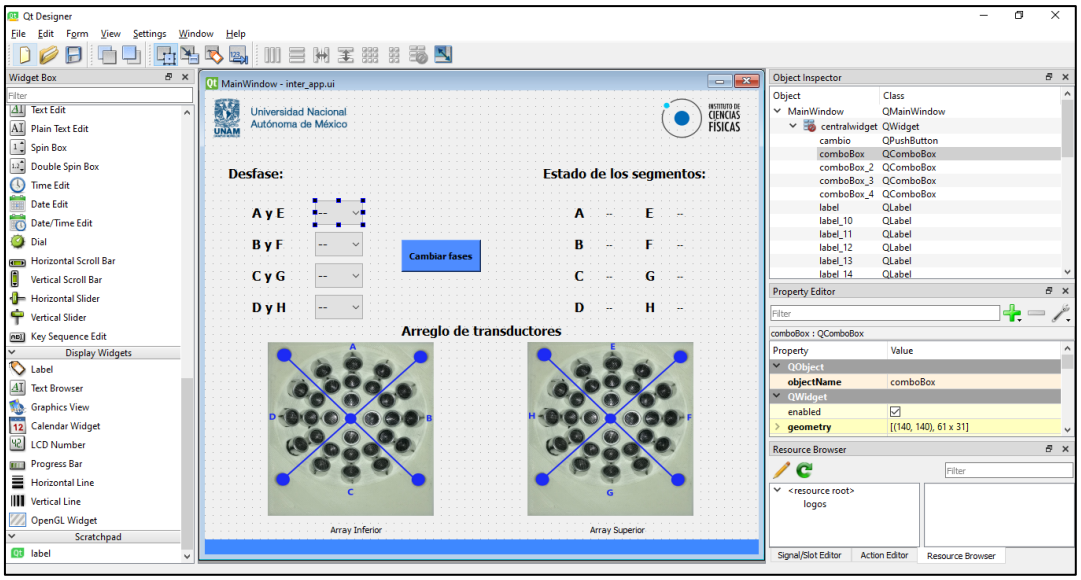


Figura 3.6 Menú principal del software Qt Designer.

Hasta este punto la interfaz no es funcional ya que falta crear el ciclo de eventos principal que nos permitirá enviar y recibir datos, activar nuestros menús desplegables y hacer cambios a nuestras etiquetas. Cabe mencionar que el software, al crear los gráficos, nos crea un código que contiene todo lo diseñado (Anexo A), todas las funciones que necesitará para crear la ventana de la aplicación. Este código lo podemos importar o llamar directamente a nuestro ciclo principal.

Para poder diseñar nuestro código funcional (principal) se necesita llevar a cabo el proceso descrito en el siguiente esquema:

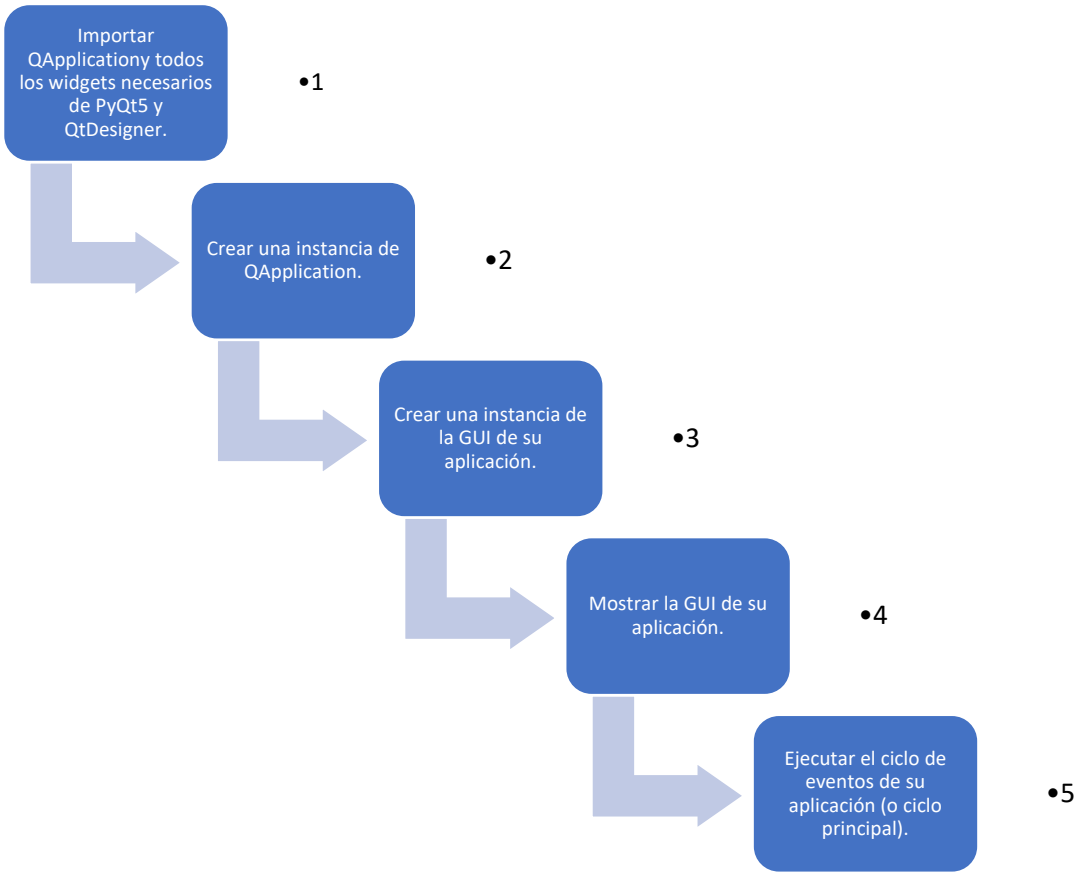


Figura 3.7 Esquema del desarrollo de la GUI.

El proceso descrito en la Figura 3.7 nos permite hacer el llamado de programa antes creado en Qt Designer y también nos permite interpretar los datos que el usuario elija y a su vez enviarlos por el puerto serial presionando el botón “Cambiar desfases”. En el Anexo B se muestra el código funcional comentado y completo de la GUI.

3.3.1.2 Adquisición de datos

La adquisición de datos lo realizamos con la ayuda de los ComboBox, que son básicamente los menús despegables y vienen con valores predeterminados. En la Figura 3.3 se observa que dichos valores son los desfases que selecciona el usuario (0°, 90°, 180° y 270°). A estos valores gráficos se les asigno un numero hexadecimal para poder enviarlos en bytes a través del puerto serial, haciendo uso del módulo USB-UART.

En la siguiente Figura se muestra un diagrama de flujo el cual muestra el proceso de la adquisición de datos, dicho diagrama es el mismo para las diferentes variables que utilizamos.

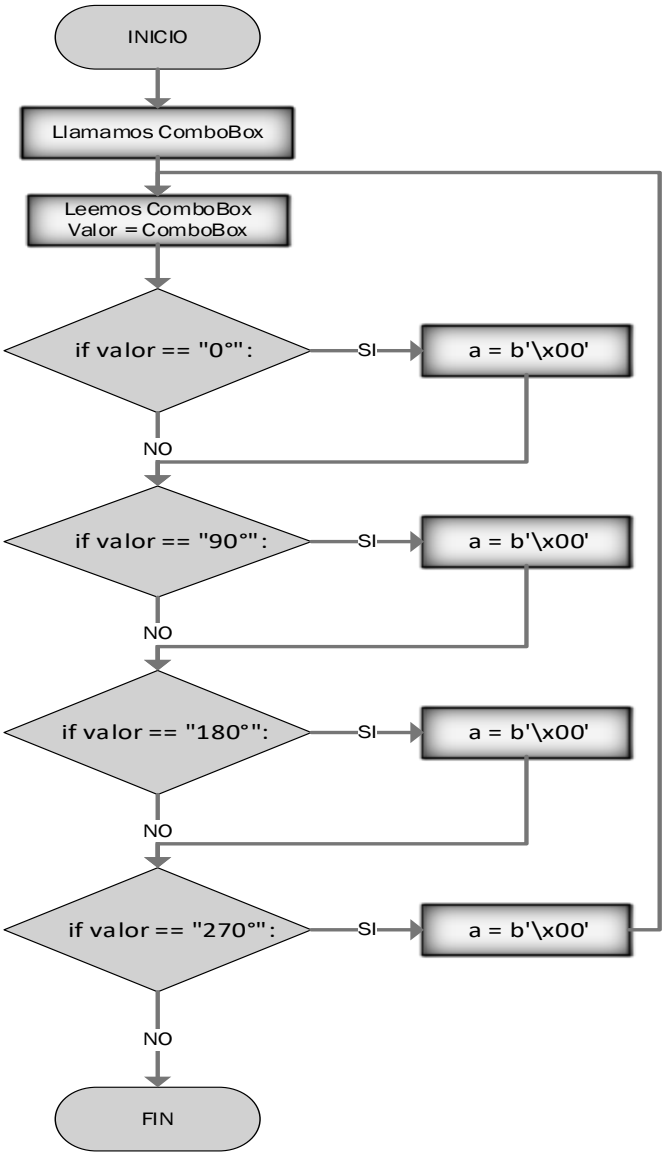


Figura 3.8 Diagrama de flujo de Adquisición de datos.

Las variables que se utiliza y los valores que puede tomar cada una se muestra en la siguiente tabla.

Tabla 3 Variables de la adquisición de datos del GUI

# de Variable	Nombre de variable	Segmentos que controla	Valores a tomar
1	ComboBox	A y D	b'\x00' b'\x01' b'\x02' b'\x03'
2	ComboBox_2	B y E	b'\x00' b'\x04' b'\x08' b'\x0c'
3	ComboBox_3	C y F	b'\x00' b'\x10' b'\x20' b'\x30'
4	ComboBox_4	D y G	b'\x00' b'\x40' b'\x80' b'\xc0'

3.3.1.3 Envio de datos

La comunicación a través del puerto serie se sigue utilizando actualmente para conectar el PC con otros dispositivos, en este caso con la FPGA que estará recibiendo datos de la interfaz. Para el envío de datos se utilizó el conversor USB a Serial TTL CP2102 utilizando un protocolo UART, mencionados en el capítulo dos. En la Figura 3.9 se muestra el diagrama de flujo para el envío de datos por el puerto serial en Python.

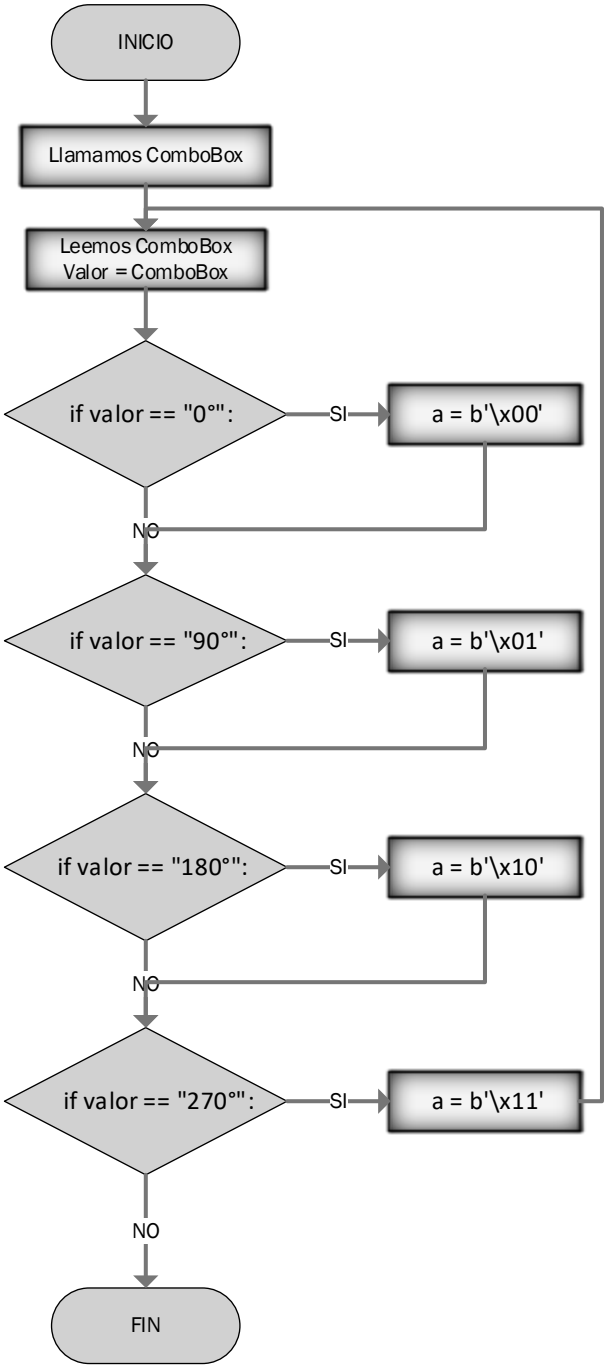


Figura 3.9 Diagrama de flujo de envío de datos.

3.3.2 Etapa 2: Procesamiento de datos FPGA

Esta segunda etapa es donde prácticamente se interpretan los datos recibidos de la GUI, posteriormente procesamos los mismo para poder tener el control de las señales generadas al igual que se genera las señales de 40 KHz con un desfase de $\frac{\pi}{2}$.

Se utilizo la FPGA ALTERA Cyclone IV CoreEP4CE6 la cual satisface todos los requerimientos específicos para implementar este proyecto, el software que se utilizo fue Quartus II, ver en el capítulo dos. Los lenguajes más utilizados para describir el hardware son VHDL y Verilog, los cuales tienen una sintaxis parecida a la de la programación en C, pero en lugar de ejecutar un programa, describen la organización de las diferentes partes del mismo a base de módulos interconectados. Se opto por utilizar Verilog para desarrollar todos los módulos o entidades que se utilizaron. En la Figura 3.10 se muestra un diagrama a bloque en cual representa nuestros módulos de la programación de la FPGA.

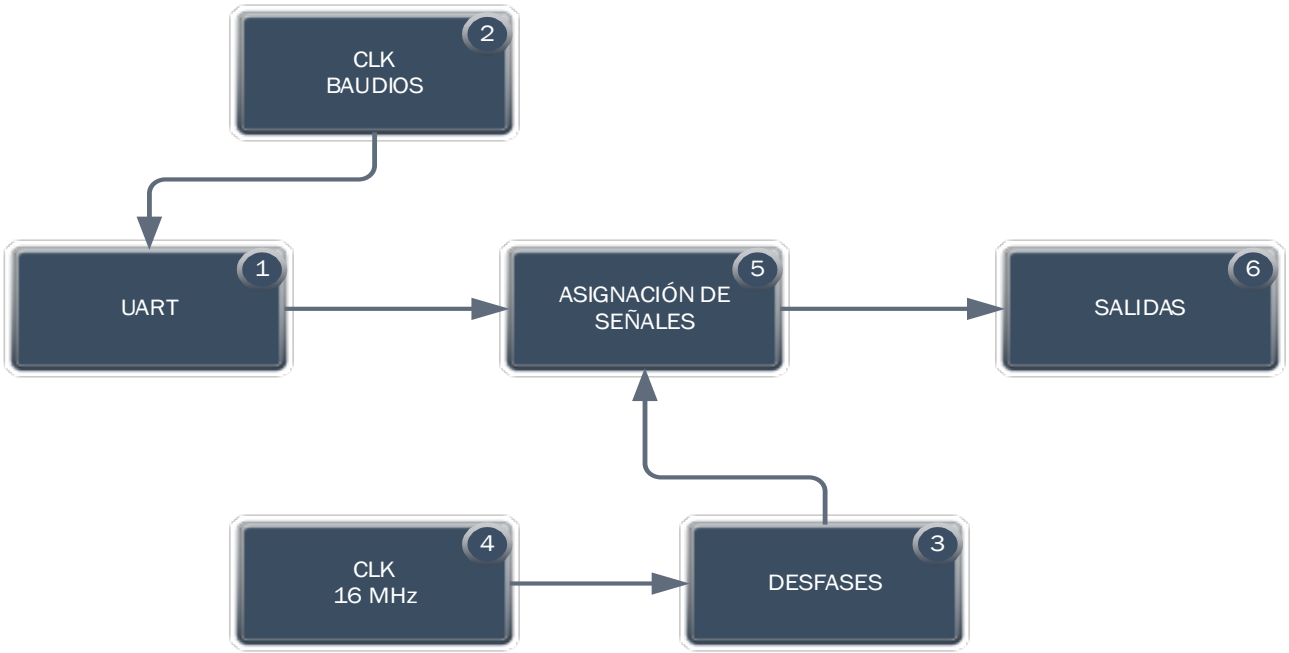


Figura 3.10 Diagrama a bloque de la FPGA.

Como se puede observar en la Figura 3.10, los módulos están interconectados dependiendo unos de otros. A continuación, se describen brevemente los módulos que se desarrollaron:

1. UART: Este módulo es el encargado de recibir los datos de la GUI.
2. CLK BAUDIOS: El módulo UART depende de una señal de reloj la cual define la velocidad en bits por segundos.
3. DESFASES: Este bloque o modulo se encarga de generar las señales de 40 KHz con su respectivo desfase.
4. CLK 16 MHz: La FPGA tiene un reloj de 50 MHz, para generar las señales se necesita una señal de reloj múltiplo de 40 KHz, para esto se crea un módulo el cual nos arroja una señal de 16 MHz.
5. ASIGNACIÓN DE SEÑALES: Este bloque se encargar de controlar todas las señales tanto las del UART como los desfases que se generan y dependiendo del valor recibido en el UART son las que señales desfasadas que deja pasar al siguiente bloque.
6. SALIDAS: Lo único que hace es determinar una salida física, es decir asignar pines de salida.

3.3.2.1 UART

La comunicación en serie asíncrona tiene las ventajas de menos línea de transmisión, alta confiabilidad y larga distancia de transmisión, por lo que se usa ampliamente en el intercambio de datos entre computadoras y periféricos. La comunicación en serie asíncrona generalmente se implementa mediante un transmisor receptor asíncrono universal (UART). Como se detalla en el Capítulo dos este tipo de comunicación lo que envía o recibe es una trama de bit Figura 3.11.

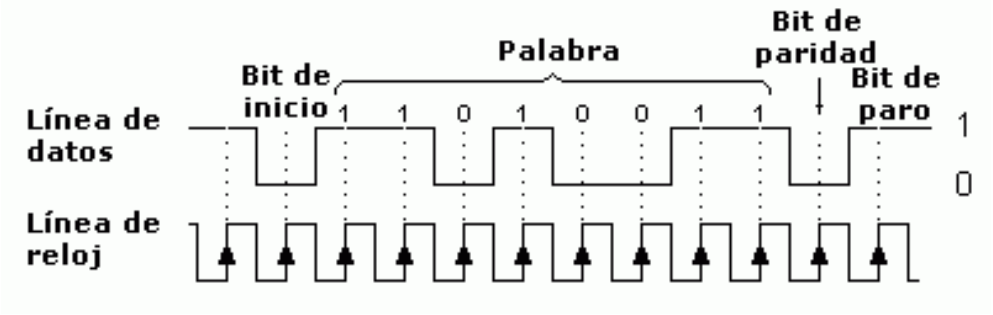


Figura 3.11 Trama de Bits UART.

La idea es crear un módulo que sea capaz de recibir datos en formato 8N1, es decir, 1 bit de start, 8 bits de datos, sin paridad y 1 bit de stop. Se asume el orden de envío

estándar LSB - MSB (primero el bit 0 y por último el bit 7) y una velocidad de 250000 bits por segundo (bps) siendo esta modificable.

El módulo de comunicación serie UART se divide en tres submódulos:

1. Generador de velocidad en baudios.
2. Módulo Receptor.
3. Módulo Transmisor.

Por lo tanto, la implementación del módulo de comunicación UART es en realidad la realización de los tres submódulos.

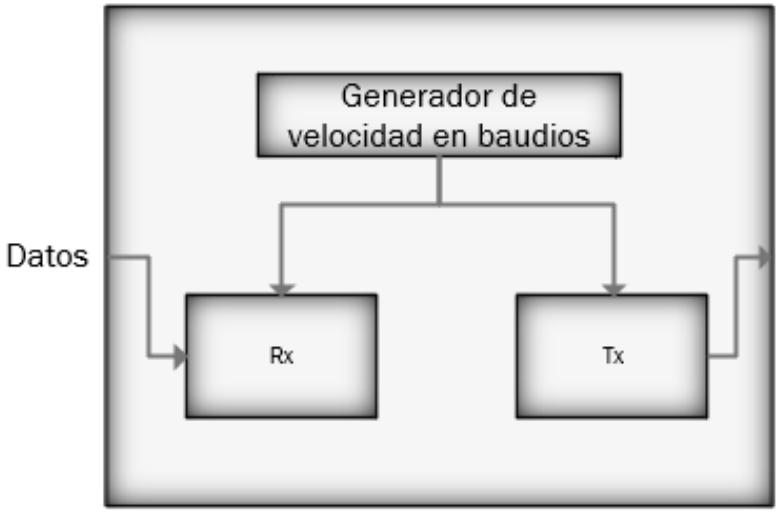


Figura 3.12 Modulo UART

Generador de velocidad en baudios

El generador de velocidad en baudios se utiliza para producir una señal de reloj local que es mucho más alta que la velocidad en baudios para controlar la recepción y transmisión de UART. La velocidad en baudios con la que se trabaja es de 250000 baudios. El reloj de la FPGA es 50MHz.

Para generar esta velocidad necesitamos una cadena de pulsos, en pocas palabras un contador, lo cual llamaremos "TICKS". Esta cadena de pulsos debe ser 16 veces más rápido que los datos UART para evitar errores; por lo tanto, si queremos que las "TICKS" sean 16 veces la frecuencia de la señal UART necesitamos una frecuencia 16 veces mayor que la de 250000 Hz.

El ancho de la señal UART es:

$$T = \frac{1}{f} = \frac{1}{250000} = 4\ \text{us}$$

El ancho del reloj principal es:

$$T = \frac{1}{f} = \frac{1}{50000000} = 20\ \text{ns}$$

En la siguiente operación se muestra cuántos pulsos de 20 ns necesitamos para obtener 4 us.

$$TICKS = \frac{\left(\frac{4\ us}{16}\right)}{20\ ns} = 12.5\ ticks$$

De esta manera generamos una señal de reloj en base al reloj de la FPGA, el generador de velocidad de baudios es simplemente una señal de reloj a la frecuencia que se desee y se le conoce como divisor de frecuencia.

Módulo receptor RX

El módulo receptor UART se utiliza para recibir las señales en serie en RXD y convertirlas en datos en paralelo.

Los diferentes bloques que componen el receptor asíncrono son los siguientes:

- Un registro de desplazamiento: donde se irán guardando bit por bit el byte que llegue.
- Un latch o registro de salida: donde se realizará una carga paralela desde el registro de desplazamiento del dato recibido una vez se compruebe que la recepción ha sido correcta.
- Un contador independiente para realizar el conteo de los bits que van llegando, respectivamente.

- Una máquina de estados (FSM, Finite-State Machine) encargada del control de los contadores, del registro de desplazamiento y del registro de salida.

En la Figura 3.13 se muestra la máquina de estados resultante para nuestro módulo de recepción de la UART.

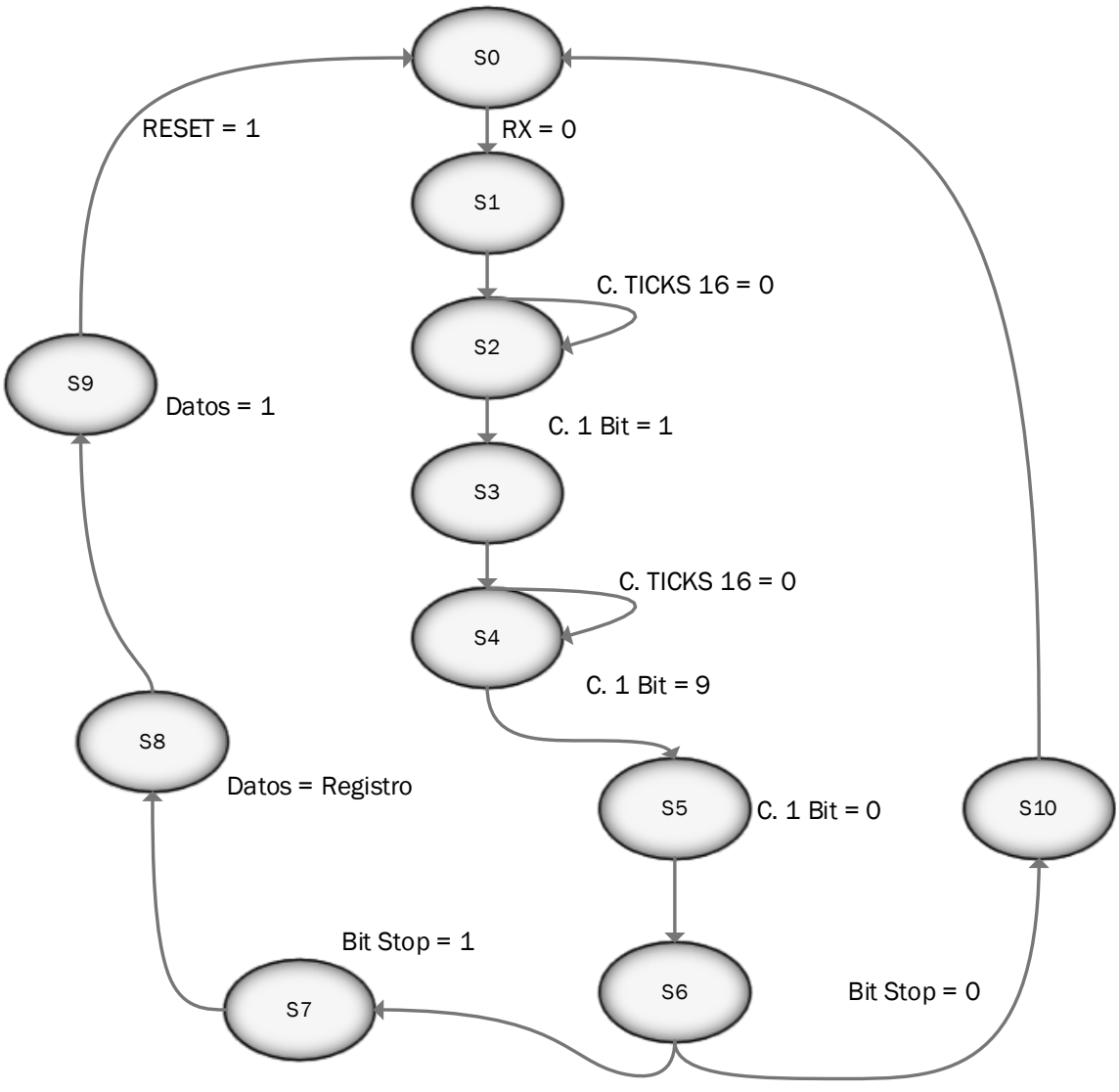


Figura 3.13 Diagrama de la máquina de estados del Receptor de la UART.

El funcionamiento del módulo receptor UART es el siguiente:

1. En el estado inicial, la FSM espera a que el pin RX valga 0.
2. En el instante en que RX pase a valor 0 la FSM inicializa un contador de 16 TICKS, en el momento que este contador alcanza su límite se pasa al siguiente estado.
3. Se inicializa un contador que va a contar la cantidad de bits 8 + 1 bit de stop = 9.
4. Se va recorriendo el valor de RX en el registro de desplazamiento, se reinicia el contador que tiene como límite el equivalente en tiempo a 1 bit a 250000 bps y se incrementa el contador del número de bits.
5. Si el contador de bits vale 9, saltamos al paso 8.
6. Esperamos a que el contador de tiempo para 1 bit llegue al límite
7. Saltamos al paso 4.
8. Si el bit de stop vale 1 cargamos el buffer de salida e indicamos que en el buffer de salida hay datos válidos, en caso contrario no se carga de buffer de salida.
9. Saltamos al paso 1.

El módulo de transmisión UART convierte los bytes en bits en serie de acuerdo con el formato de trama básico y transmite esos bits a través de TXD en este caso no se ocupa ya que solo estamos recibiendo datos, se crea el módulo completo para trabajos futuros.

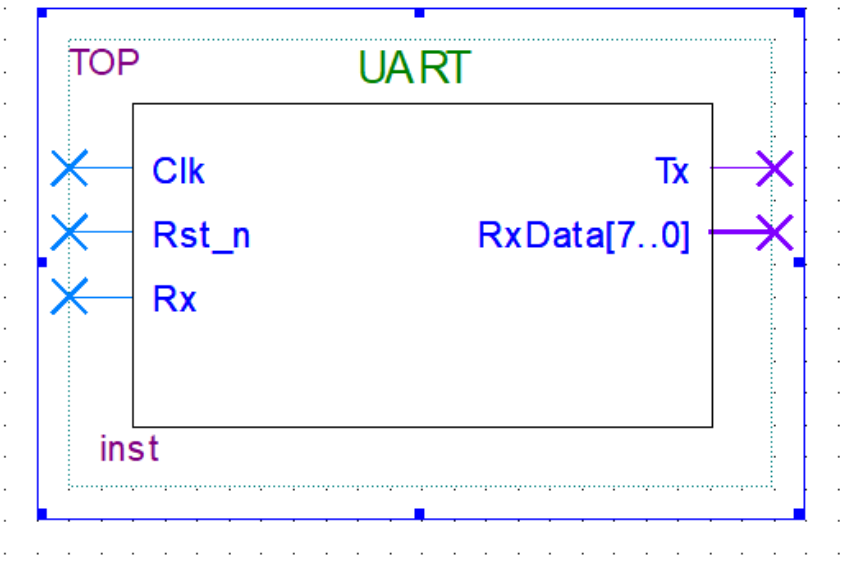


Figura 3.14 Bloque del módulo UART implementada.

En la Figura 3.14 se muestra el bloque del módulo UART implementado en el software Quartus II, dentro de este bloque se tiene la programación de todo lo descrito.

3.3.2.2 Desfases

Se tiene conocimiento que las ondas que levitan las partículas son sinusoidales, el problema es que esas ondas son bastante complejas de generar digitalmente. Sin embargo, es posible generar ondas cuadradas utilizando una FPGA y enviarlas como

señal a los transductores, los cuales, tienen una respuesta casi sinusoidal (A. Marzo, 2017).

En esta etapa, el módulo desfases genera 4 señales digitales de 40 KHz con un desfase de π/2 como lo muestra la Figura 3.15.

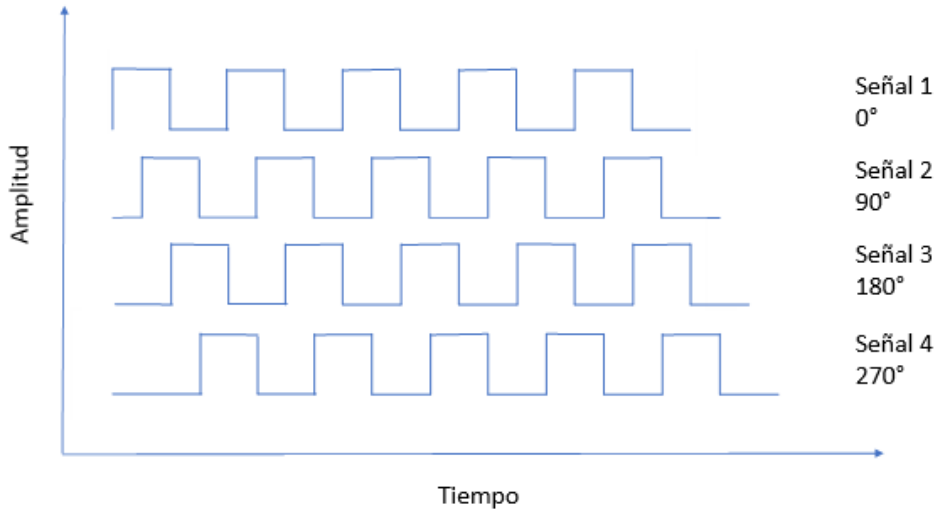


Figura 3.15 Desfases de señales.

El diseño del módulo se implementó lo siguiente:

1. Un divisor de frecuencia de 40 KHz.
2. Un contador para generar los desfases.

En este caso se trabaja con una señal de reloj de 16MHz ya que esta frecuencia es múltiplo entero de la frecuencia de operación de los transductores (40 KHz) y facilita la sincronización de las señales generadas y la operación de los transductores. La señal de 16 MHz la generamos con un Masterclock que es un bloque definido por Quartus el cual nos permite generar señales reloj con más definición ya que implementa un PLL de la FPGA.

En la figura 3.16 se muestra el bloque del Masterclock el cual está definido por Quartus II, cabe mencionar que en comparación a un divisor de frecuencia tradicional este módulo es más efectivo ya que tiene un lazo de control de fase generando así una señal digital exacta. En la figura 3.17 se muestra el algoritmo que se implementó en Verilog.

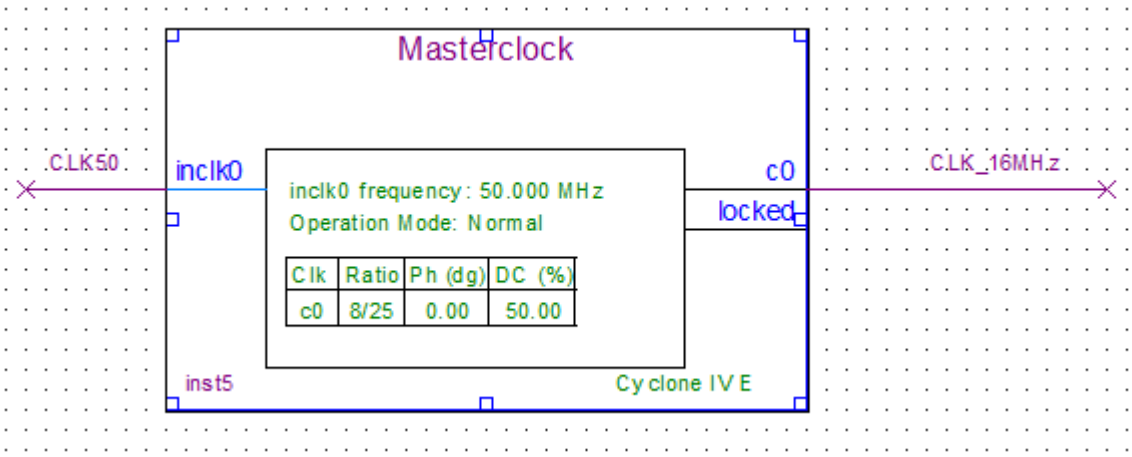


Figura 3.16 Masterclock Quartus II.

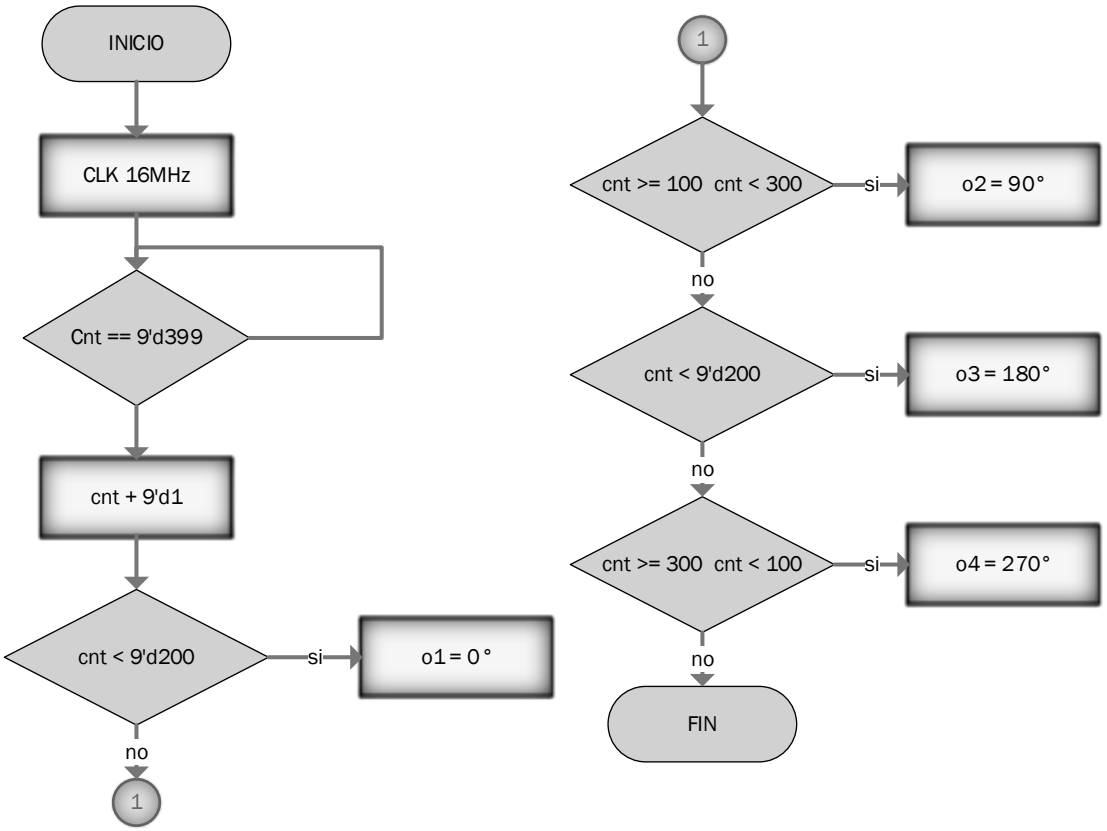


Figura 3.17 Diagrama de flujo Desfases.

En base a la señal de reloj de 16 MHz se procede a diseñar un divisor de frecuencia de 40 KHz. Se crea un contador de 40000 bit por segundo

$$Clk\ \frac{1}{f} = (16000000)\frac{1}{40000} = \quad 400 \text{ ticks}$$

Se implementa el contador en Verilog de la siguiente manera:

Primero definimos entradas y salidas de nuestro modulo como se muestra en la siguiente figura.

```
module desfases (
    input wire clk16m,
    output reg o1,
    output reg o2,
    output reg o3,
    output reg o4
    );
```

Figura 3.18 Definimos desfases.

Después se programa el divisor de frecuencia Figura 3.19.

```
reg [8:0] cnt = 9'd0;
always @(posedge clk16m) begin
  if (cnt == 9'd399)
    cnt <= 9'd0;
  else
    cnt <= cnt + 9'd1;
end
```

Figura 3.19 Divisor de frecuencia 40 KHz.

En estas líneas de programación se ingresa los ticks que necesita el contador para poder crear la señal de 40KHz.

El siguiente paso es crear un contador el cual nos genere los desfases de $\frac{\pi}{2}$ es decir 90° de desfase entre las 4 señales, por lo tanto, para tener ese retraso de la señal lo único que se hace es ir muestreando nuestra señal, pero a diferente tiempo es decir se divide los 400 ticks en 4 lo que resulta que tomamos la señal en alto a partir de 100 ticks se llega a los 300 y se pasa al estado bajo y como nuestro contador se reinicia cada vez que llega a 400 se cicla resultando una señal desfasada. Se muestra la programación en la Figura 3.20.

```
always @* begin
  if (cnt < 9'd200)
    o1 = 1'b1;
  else
    o1 = 1'b0;

  if (cnt >= 9'd100 && cnt < 9'd300)
    o2 = 1'b1;
  else
    o2 = 1'b0;

  if (cnt >= 9'd200)
    o3 = 1'b1;
  else
    o3 = 1'b0;

  if (cnt >= 9'd300 || cnt < 9'd100)
    o4 = 1'b1;
  else
    o4 = 1'b0;
    end
endmodule
```

Figura 3.20 Programación de Desfases.

Todos los registros por los cuales las señales desfasadas han pasado, tendrán un 1 en la salida mientras que los que no, tendrán un 0. De este modo la posición de los niveles altos y bajos indican que tan largo se propagó la señal de inicio antes de que llegara la señal de final. La cantidad de valores N obtenidos en alto es igual a la diferencia de tiempo entre la señal de inicio y la de final dividido entre el tiempo desfasado de las señales. Nuestro modulo implementado queda de la siguiente manera Figura 3.21.

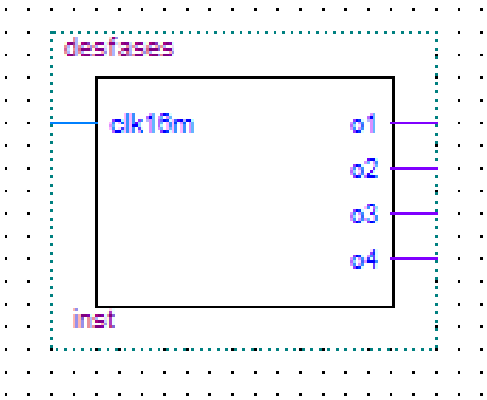


Figura 3.21 Bloque del módulo Desfases implementado.

3.3.2.3 Asiganacion de señales

El procesamiento de los datos recibidos del UART y de las señales generadas por el módulo Desfase se lleva a cabo con el módulo Control, el cual procesa y controla todas las señales del sistema. En la Figura 3.22 se muestra el diagrama de flujo de la recepción de los datos enviados de la GUI y dependiendo los datos recibidos es la que señal que se obtendrá en la salida, teniendo 4 estados posibles cada una de las señales de salida (0°, 90°, 180°, 270°). Cabe mencionar que este diagrama es el mismo para cada una de las variables que se tiene como salidas (A, B, C, D, E, F, G, H).

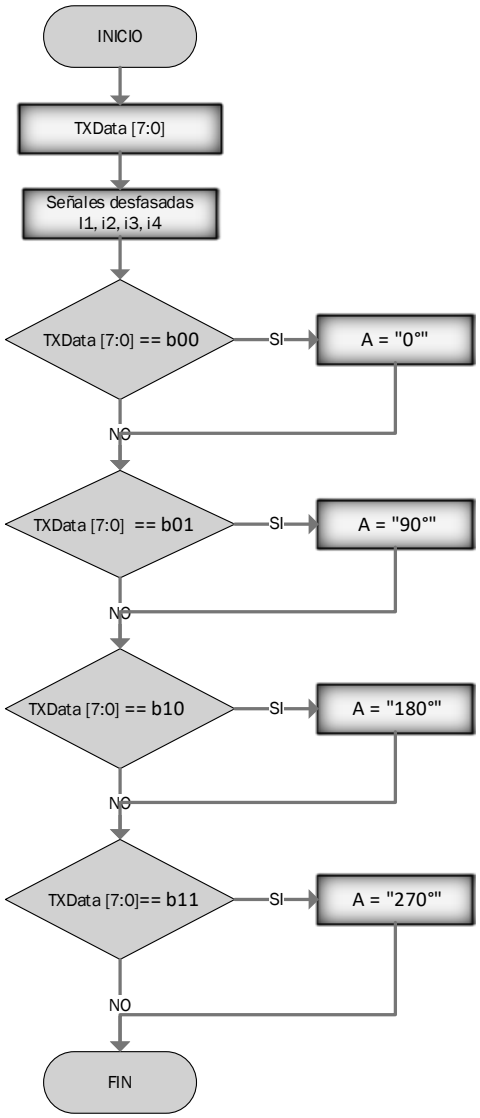


Figura 3.22 Diagrama de flujo del procesamiento de datos y señales desfasadas.

El módulo Control toma los datos recibidos de la UART y las señales de 40 KHz desfasadas posteriormente se compara los datos recibidos y dependiendo el dato recibido es la señal desfasa la que se le asigna en la salida, en este caso se les asigna a las salidas A-H. Los datos de la UART se comparan con los mismos datos que se envían desde la interfaz gráfica. En la Figura 3.23 se observa el módulo implementado en Quartus II.

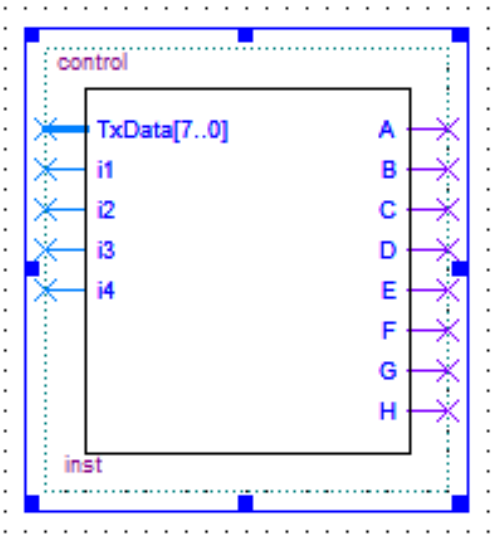
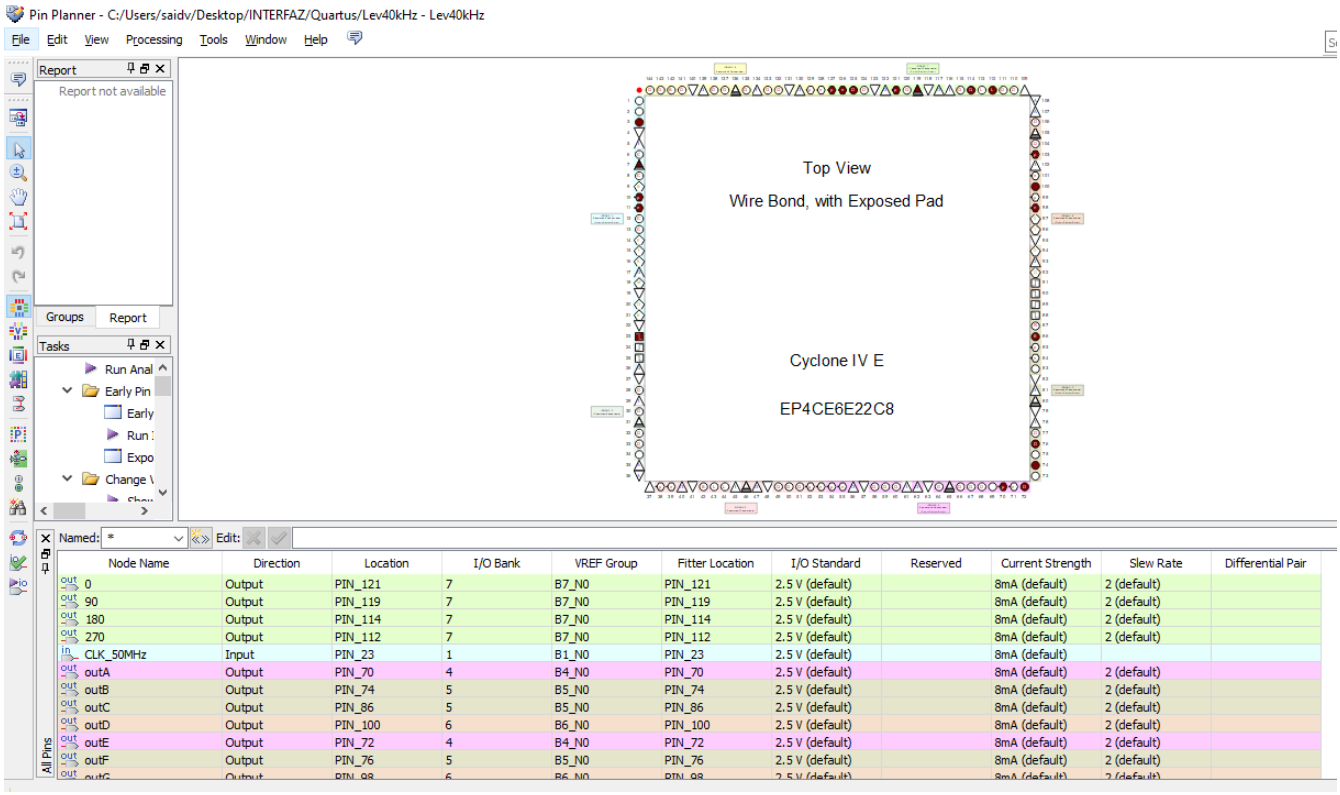


Figura 3.23 Bloque del módulo Control implementado.

Toda la programación que se realizó en la FPGA con software Quartus II se muestra en el Anexo C, la cual muestra el conjunto de todos los módulos diseñados ya interconectados entre sí.

3.3.2.3 Asignacion de pines

Para la asignación de los pines de nuestra FPGA, Quartus II tiene un apartado en el cual te muestra todo el pin de la placa FPGA y con la ayuda del esquemático de la placa Cyclone IV (Anexo D) se podrá asignar ya que cada pin tiene características definidas ya sea que sea el pin de nuestro reloj o que solo esos pines son de salida o entrada. En la Figura 3.24 se observa la asignación de pines que contiene Quartus II.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
0	Output	PIN_121	7	B7_N0	PIN_121	2.5 V (default)		8mA (default)	2 (default)	
90	Output	PIN_119	7	B7_N0	PIN_119	2.5 V (default)		8mA (default)	2 (default)	
180	Output	PIN_114	7	B7_N0	PIN_114	2.5 V (default)		8mA (default)	2 (default)	
270	Output	PIN_112	7	B7_N0	PIN_112	2.5 V (default)		8mA (default)	2 (default)	
CLK_50MHz	Input	PIN_23	1	B1_N0	PIN_23	2.5 V (default)		8mA (default)		
outA	Output	PIN_70	4	B4_N0	PIN_70	2.5 V (default)		8mA (default)	2 (default)	
outB	Output	PIN_74	5	B5_N0	PIN_74	2.5 V (default)		8mA (default)	2 (default)	
outC	Output	PIN_86	5	B5_N0	PIN_86	2.5 V (default)		8mA (default)	2 (default)	
outD	Output	PIN_100	6	B6_N0	PIN_100	2.5 V (default)		8mA (default)	2 (default)	
outE	Output	PIN_72	4	B4_N0	PIN_72	2.5 V (default)		8mA (default)	2 (default)	
outF	Output	PIN_76	5	B5_N0	PIN_76	2.5 V (default)		8mA (default)	2 (default)	

Figura 3.24 Asignación de pines Quartus II.

3.3.3 Etapa 3: Amplificadores

La señal que nos entrega la FPGA es muy débil para operar a los transductores directamente y se requiere una etapa de amplificación. Se utilizaron un par de drivers L298N Dual para amplificar las señales y un regulador de voltaje lineal de 5 V LM78M05 de montaje superficial para poder alimentar la FPGA y dar las señales lógicas que requiere el driver. La siguiente tabla se muestra las especificaciones del Driver L298N cumpliendo con los requisitos del sistema siendo una selección de amplificador efectiva.

Tabla 4 Especificaciones del driver L298N.

Especificaciones	
Frecuencia de funcionamiento	100 KHz
Voltaje de funcionamiento	4V ~ 35 V.
Voltaje Lógico	2.5V ~7V
Corriente máxima	3 A.
Potencia máxima	25W
Salidas	4

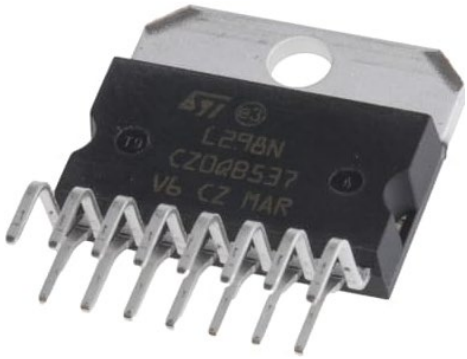


Figura 3.25 Driver L298N

.

En la tabla 5 se muestra las especificaciones que tiene el regular LM78M05.

Tabla 5 Especificaciones del regulador LM78M05.

Especificaciones	
Voltaje entrada máximo	7.2V. ~ 35V.
Voltaje de salida	5V.
Voltaje Lógico	2.5V ~7V
Corriente salida máxima	0.5 A.
Paquete	T0-252

Figura 3.26 Regulador LM78M05.

Se utiliza una configuración push-pull, esto quiere decir que se puede impulsar una corriente eléctrica positiva o negativa en una carga, para que el voltaje pico a pico que reciben los transductores sea el doble del voltaje de entrada. Cada amplificador puede llegar a manejar dos canales con hasta 70 Vpp y una resolución de fase de $\frac{\pi}{2}$, permitiendo duplicar los desfases en un trabajo a futuro. La figura 3.27, muestra el circuito empleado en la etapa de amplificación.

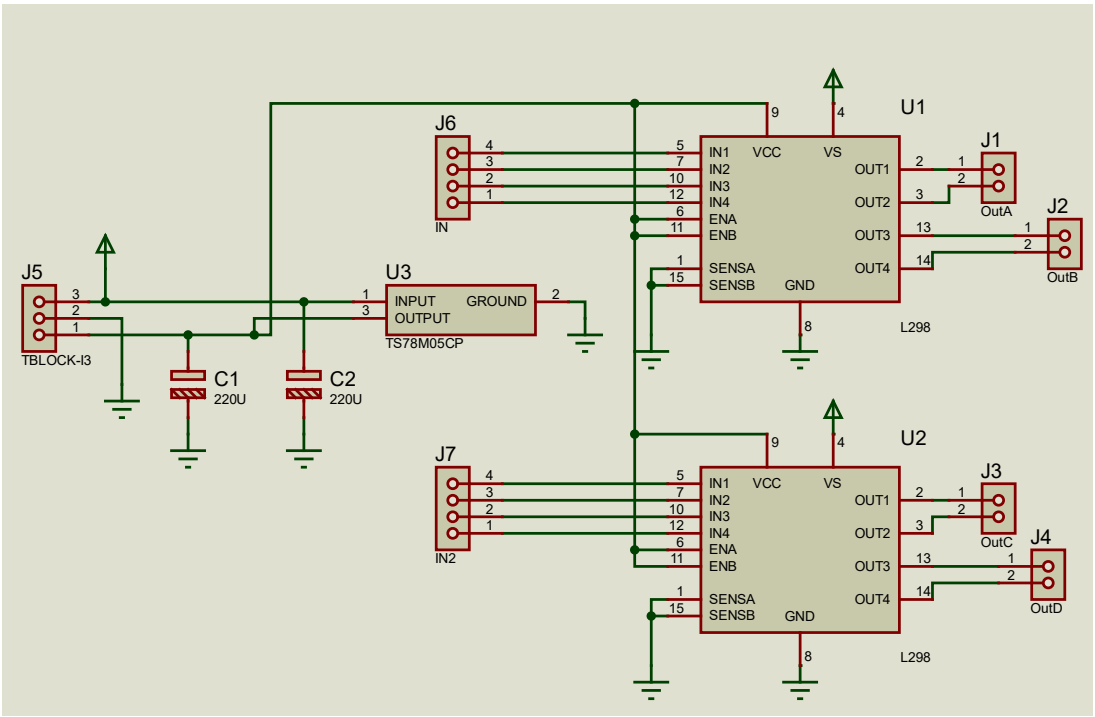


Figura 3.27 Circuito amplificador.

El PCB se realizó con el software PROTEUS, mismo que está enfocado en el diseño de circuitos electrónicos. La Figura 3.28 muestra el diseño del PCB añadiendo medidas de la placa electrónica e indicadores de las conexiones entradas, salidas y voltaje de alimentación.

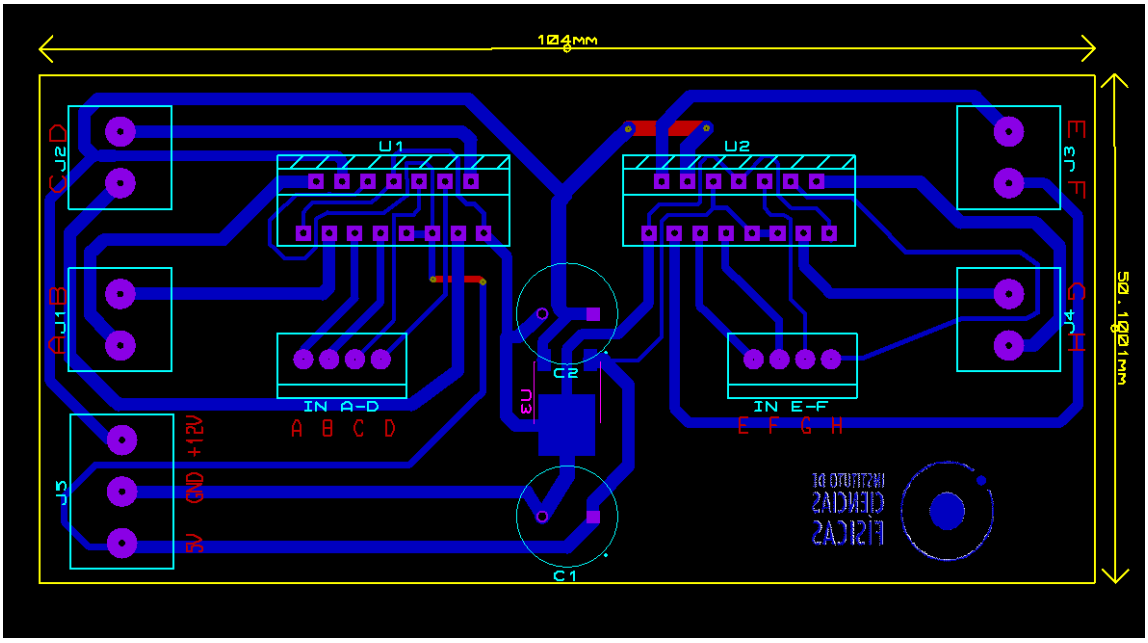


Figura 3.28 PCB del circuito amplificador.

El software tiene una visualizador 3D, permitiendo ver el diseño final del PCB como se muestra en la Figura 3.29.

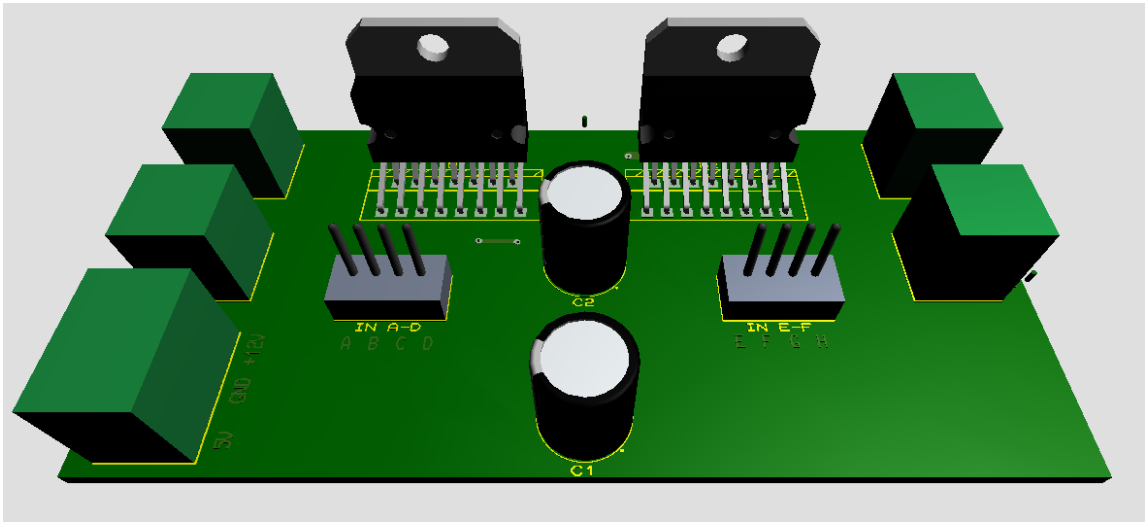


Figura 3.29 Placa Final en 3D.

3.3.4 Etapa 4: Arreglo de transductores

Los componentes principales del levitador son los transductores, elementos que transforman la señal de entrada eléctrica en ondas acústicas. Para operar en el aire, se encontró que los transductores para la medición de distancias brindan una buena potencia acústica y una frecuencia de resonancia constante. Existen diferentes tipos de transductores y estos dependen de la frecuencia a la que operan, del material por el cual están fabricados, de su tamaño entre otras características. Se trabaja con transductores de la marca Manorshi modelo MSO-A1040H07T (Figura3.30), debido a que las especificaciones son favorables para el sistema de levitación (Tabla 6).

Tabla 6 Transductores Manorshi MSO-A1040H07T.

Especificaciones	
Frecuencia de funcionamiento	40± 1.0 KHz
Voltaje de funcionamiento	0-40Vrms
Directividad	80°
Capacitancia	2400pF
Temperatura de funcionamiento	-20 ~ +80℃
Material de la carcasa	Aluminio
Tamaño	10 mm

Figura 3.30 Transductor Manorshi.

Dado que los transductores operan a una frecuencia de 40 KHz y como hemos visto en la parte teórica, podemos levitar en los nodos a objetos de tamaños menores a la mitad de la longitud de onda, en este caso es de aproximadamente 4 mm.

$$\lambda = \frac{Vs}{f} = \frac{346\ m/s}{40\ KHz} = 8{,}65\ mm$$

Donde:

λ = Longitud de onda.

Vs = Velocidad del sonido en el aire a 25 °C.

f = Frecuencia de los transductores.

Las geometrías con las que se trabajan en el instituto son cóncavas–cóncavas ya que, al enfocar a las ondas generadas, alcanzan presiones mayores. Para este trabajo se diseña un levitador con esta geometría, en la que ambos arreglos de transductores (matriz superior e inferior) se incrustan en una superficie con geometría cóncava. Para este trabajo de tesis en particular se diseñó cada arreglo de transductores con anillos concéntricos que contienen 4, 12 y 12 transductores, todos ellos con un punto focal común debido a la simetría esférica. Cada arreglo consta de 28 transductores y la cavidad con un total de 56 transductores. En la Figura 3.31 se muestra la impresión 3D diseñada y fabricada en el Laboratorio de Óptica del ICF que sirve de base para para ensamblar a los transductores.

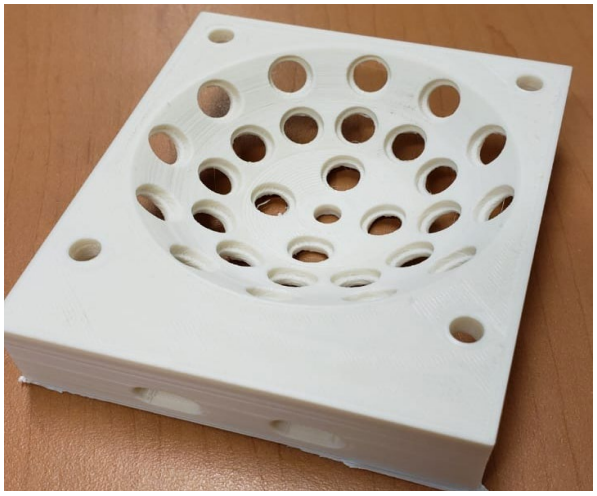


Figura 3.31 Impresión 3D de la base para los transductores.

Los anillos son diseñados estratégicamente para poder tener un arreglo de transductores con 4 segmentos como se observa en la Figura 3.32, a los que se les ha dado el nombre de A, B, C, D y E, F, G, H para el segundo arreglo, cada segmento se controla individualmente.

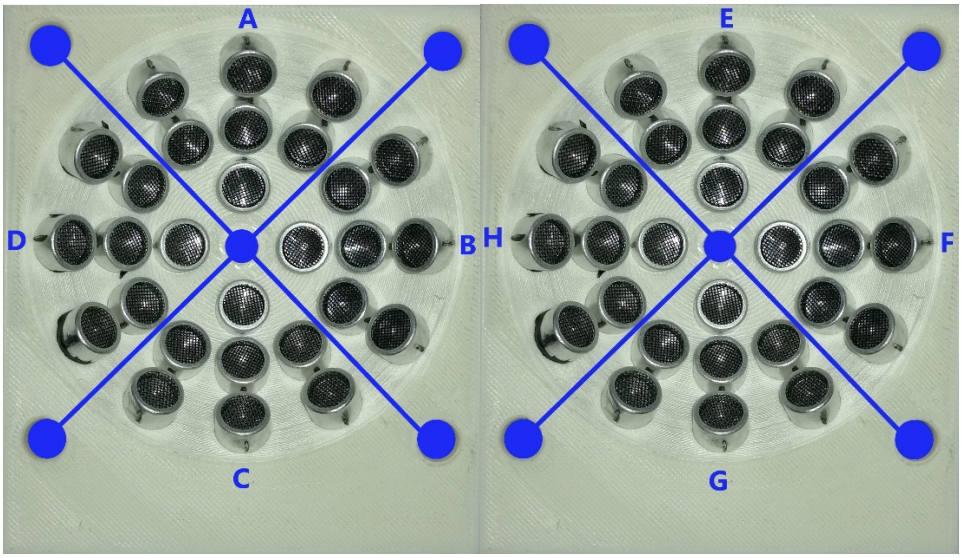


Figura 3.32 Arreglo de transductores con sus respectivos segmentos.

4. Capítulo IV: PRUEBAS Y RESULTADOS

En este capítulo se muestran las pruebas y los resultados obtenidos durante la implementación del diseño. Las pruebas se realizaron a cada una de las etapas que sistema de levitación acústica. Este capítulo muestra simulaciones y resultados experimentales.

4.1 Pruebas

En la Figura 4.1 se muestra un diagrama basándonos en este para realizar las pruebas correspondientes de cada una de las etapas del sistema.

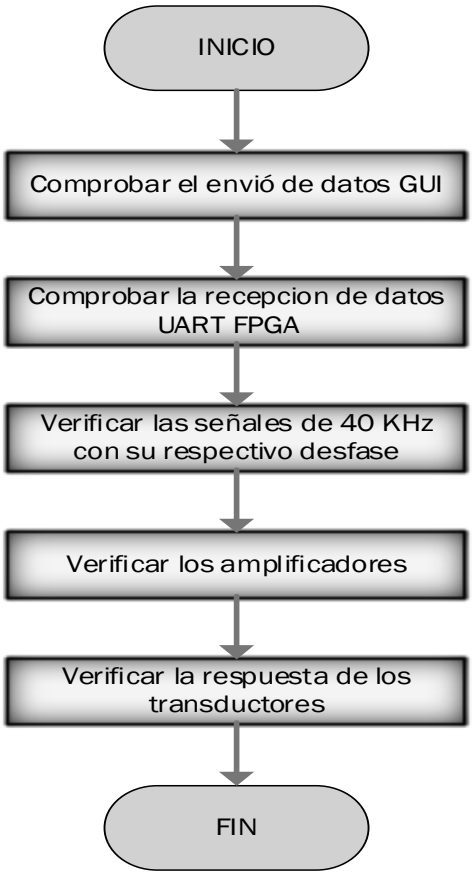


Figura 4.1 Proceso de pruebas.

4.1.1 Envió de datos GUI

Para poder verificar el envío de datos desde la interfaz gráfica de usuario se utilizó un osciloscopio mostrándonos la trama de datos enviados por el puerto serial y haciendo uso del Conversor USB a Serial TTL CP2102.

En la Figura 4.2 se muestra la trama de datos enviados, se enviaron dos valores decimales diferentes (1 y 3) a través del puerto serial, como se observar en la parte A se muestra la trama de datos del número 1 teniendo una equivalencia en binario el primer bit de datos en alto y en la parte B se muestra la trama de datos del número 3 teniendo una equivalencia en binario el primer y segundo bit de datos en alto, corroborando de esta manera el envío de datos, esta señal es medida en la salida del Conversor USB a Serial TTL siendo específicos en el pin de transmisión Tx.

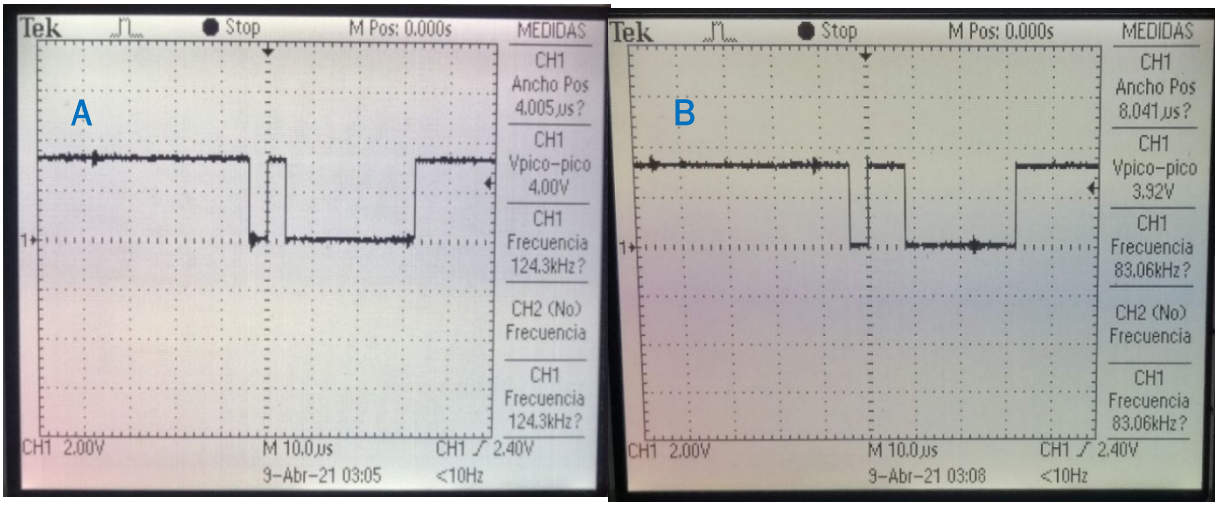


Figura 4.2 Trama de datos enviados de la GUI.

4.1.2 Recepción de datos FPGA

Para programar a la FPGA se diseñó e implemento un módulo UART que se encarga de recibir los datos provenientes de la GUI. La manera en que se comprobaron los datos recibidos fue asignando LED's de la placa FPGA como un bit de datos, tomando en cuenta los primeros 4 bits de datos y se utilizó el osciloscopio para monitorear los resultados.

En la Figura 4.3 se muestra los datos recibidos de la GUI, en la parte izquierda se muestra un LED encendido correspondiendo a un bit de dato en alto, en la parte derecha se muestra dos LEDs encendidos indicando que el valor recibido en decimal es el numero 3 ocupando dos bits de datos.

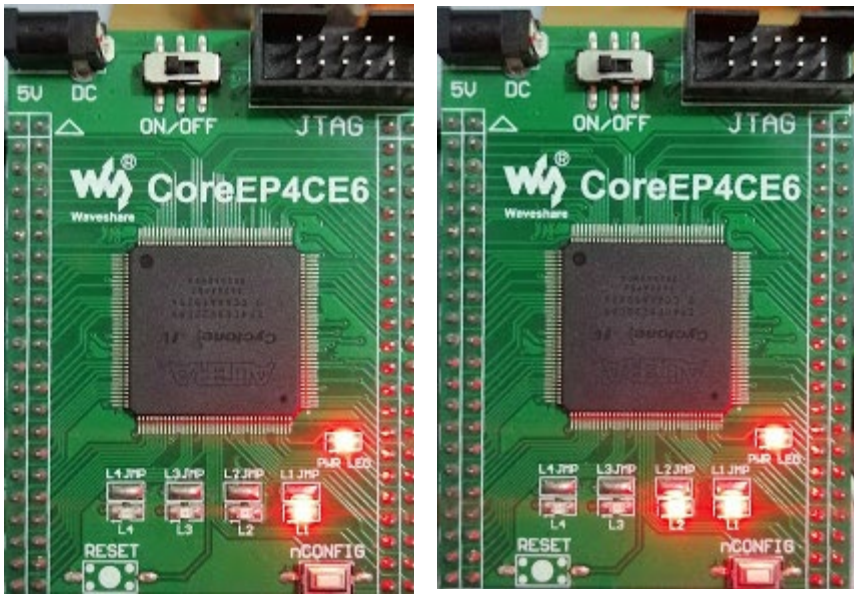


Figura 4.3 Datos recibidos en la FPGA.

4.1.3 Señales 40 KHz desfasadas

El módulo de Desfases visto en el capítulo 3 es el encargado de generar nuestras señales de 40 KHz con un desfase de $\frac{\pi}{2}$. Es modulo se hizo una simulación con EDA simulación, una herramienta de Quartus II que nos permite simular nuestros bloques implementados en el software, en la Figura 4.4 se muestra dicha simulación mostrándonos nuestras señales desfasas planteadas desde un momento.

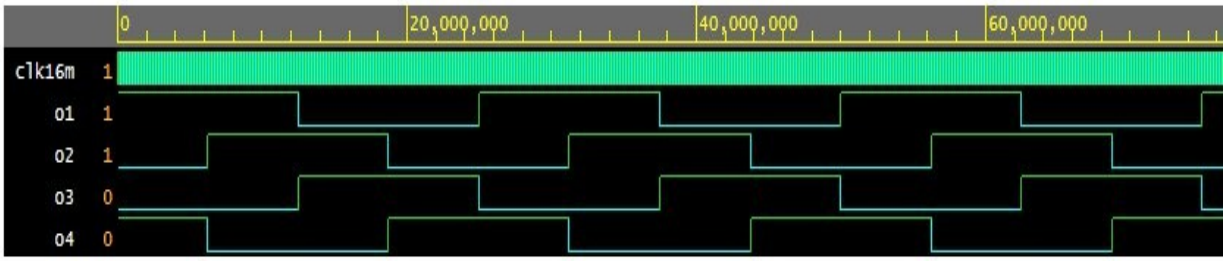


Figura 4.4 Simulación de nuestro modulo Desfases.

En la parte física se verificaron las señales haciendo uso de un osciloscopio, dadas las características de osciloscopio solo se fue posible verificar 2 señales al mismo tiempo, ya que solo tiene 2 canales, siendo una de referencia y la otra mostrando el desfase, en las siguientes figuras se muestra la señales medidas físicamente.

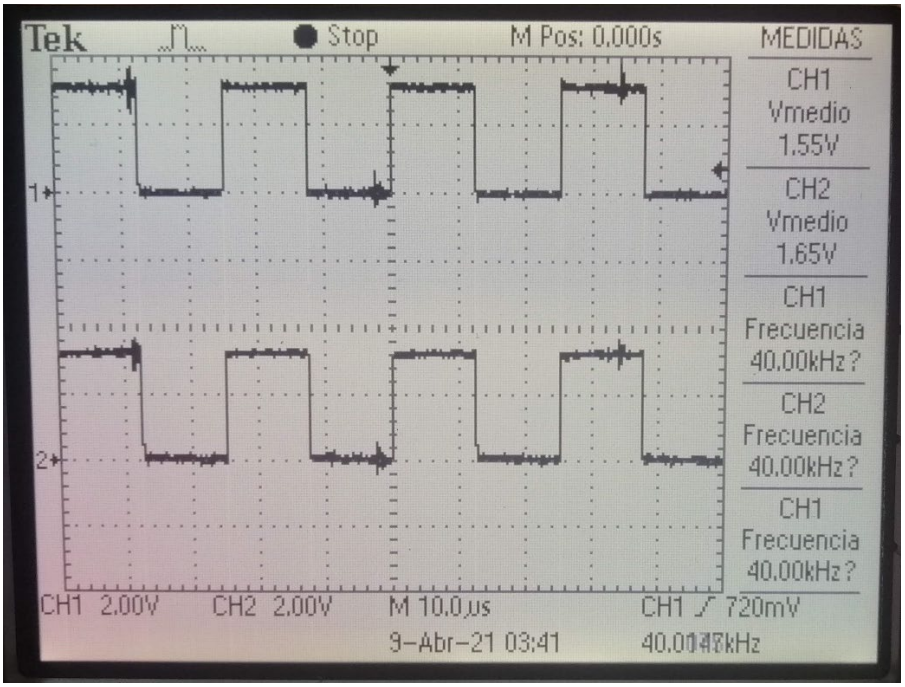


Figura 4.5 Señal de 40 KHz con una señal de desfase de 0°.

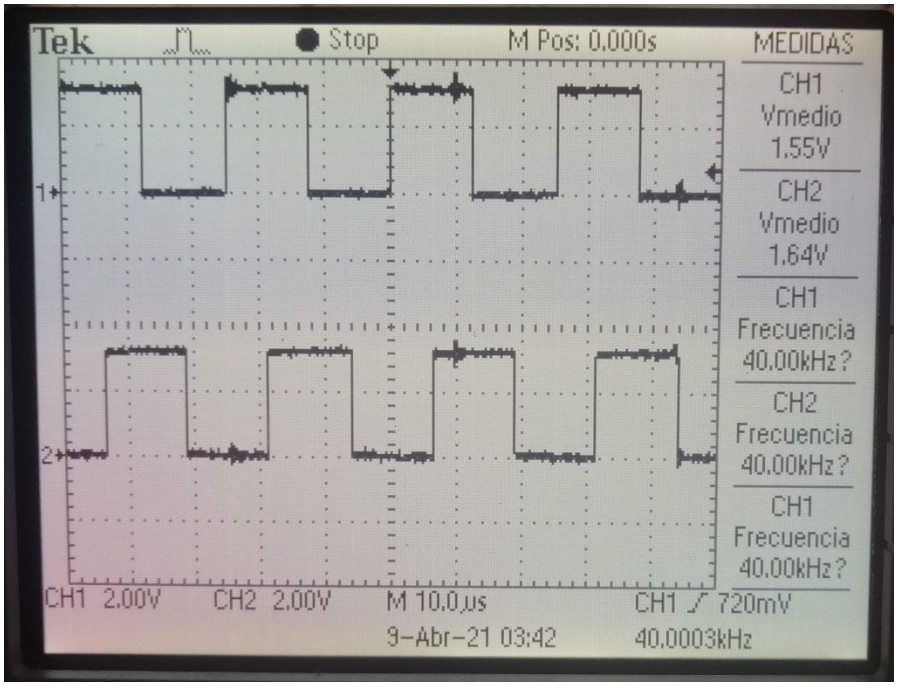


Figura 4.6 Señal de 40 KHz con una señal de desfase de 90°.

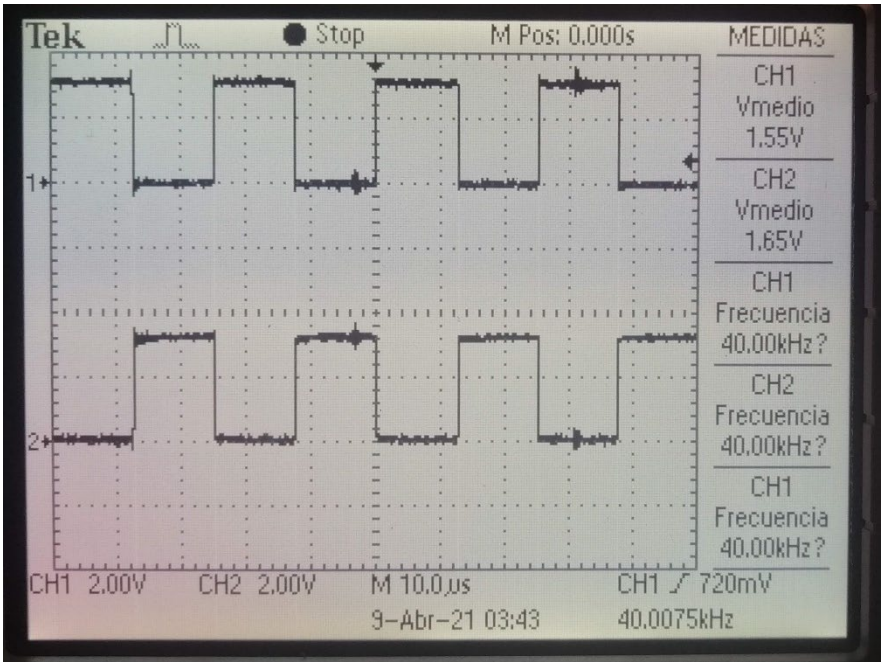


Figura 4.7 Señal de 40 KHz con una señal de desfase de 180°.

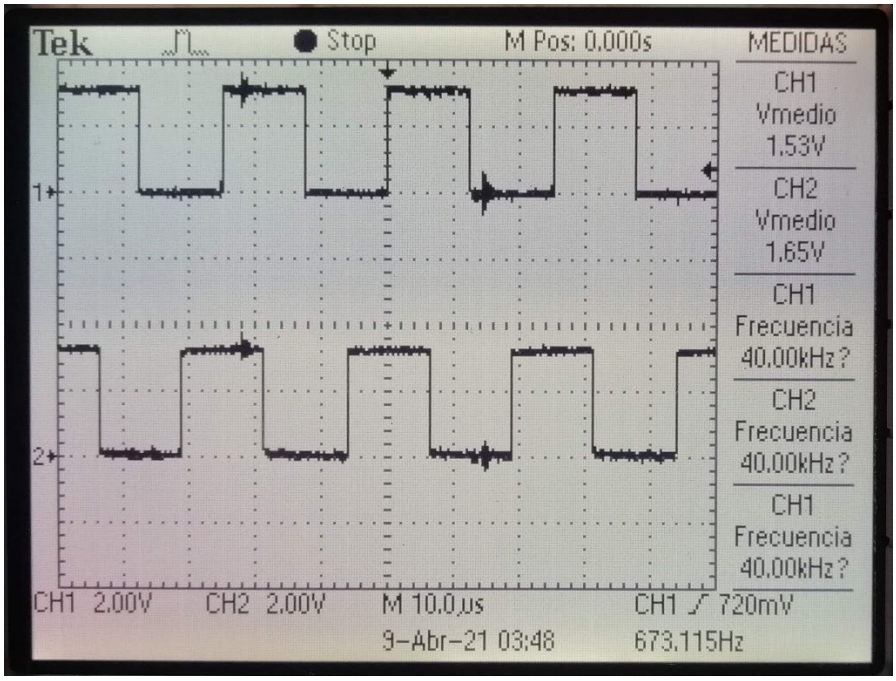


Figura 4.8 Señal de 40 KHz con una señal de desfase de 270°.

4.1.4 Señal amplificada

En la Figura 4.9 se observa la señal de 40 KHz amplificada a 12V. En el capítulo 3 se muestra las características que tiene los amplificadores que implementamos teniendo una comprobación efectiva, la figura se muestra el voltaje con la que se está amplificando, este voltaje puede variar teniendo una fuente de alimentación variable, también se muestra la frecuencia de salida siendo la misma y exacta.

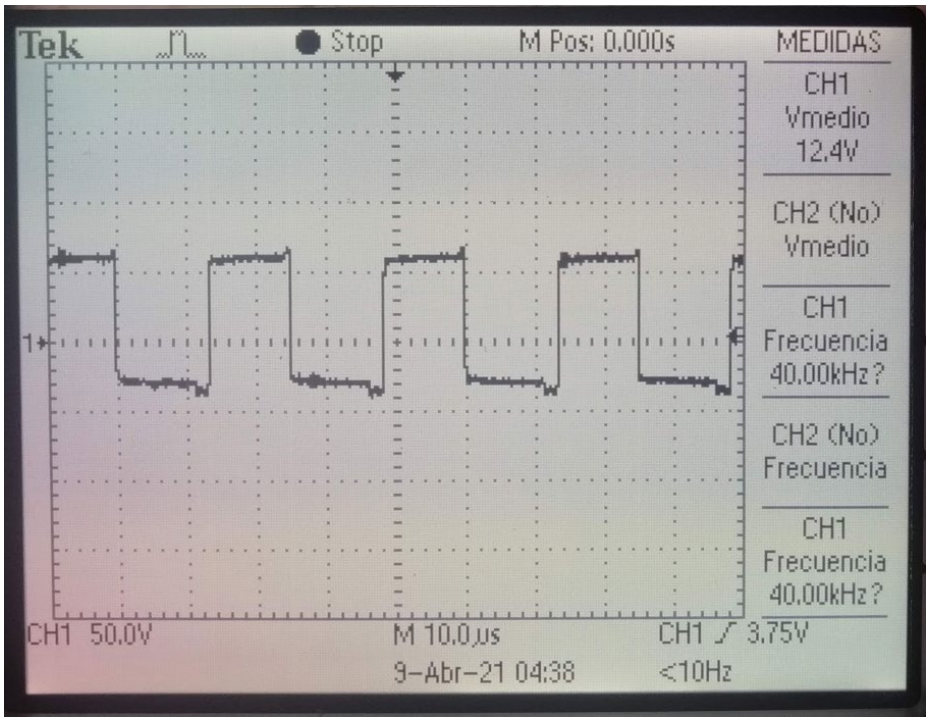


Figura 4.9 Señal amplificada.

4.1.5 Respuesta de transductores

En esta sección se muestra la respuesta que se tiene al momento de excitar a los transductores con la señal de 40 KHz, en importante mencionar que los transductores tienen polaridad y la forma en que los fabrican no suelen tener la polaridad que marca el fabricante por eso es necesario comprobar la polaridad con un multímetro.

En la Figura 4.10 se observa la respuesta del transductor (señal senoidal) en comparación con la señal amplificada de entrada (señal cuadrada). La medición de

esta señal de respuesta se midió con otro transductor como emisor conectado a una de las puntas de prueba del osciloscopio.

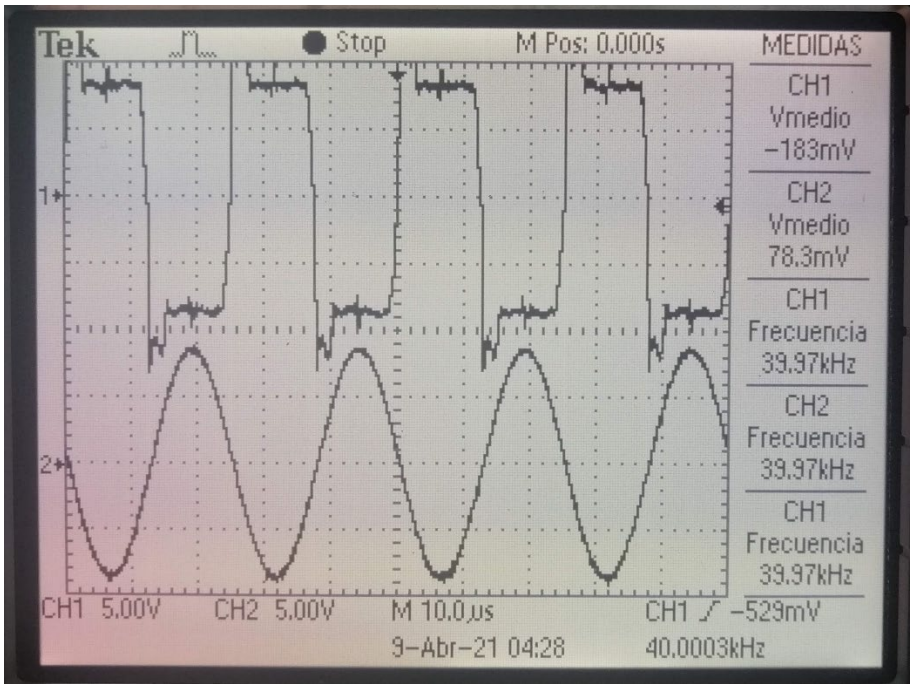


Figura 4.10 Señal de transductor vs señal amplificada.

4.2 Resultados

La implementación del sistema de levitación acústica es un resultado en sí mismo, dado que fue un proceso de pruebas, aprendizaje, cambios y mejoras. Sin embargo, el objetivo del proyecto es poder crear trampas acústicas, como se vio en la teoría con este sistema se podrá generar ondas estacionarias, trampas gemelas y sobre todo crear vórtices acústicos. Se simulo el campo acústico del sistema con el software Ultraino (Marzo, Ultraino: An Open Phased-Array System for Narrowband Airborne Ultrasound Transmission, 2015), es un software abierto que permite a los usuarios definir la geometría de la matriz; es decir, la posición y orientación de cada transductor, así como su amplitud y fase de salida. En este software el usuario también puede seleccionar el número de transductores y seleccionar diferentes frecuencias y aperturas para cada transductor. El software calcula el campo acústico complejo de frecuencia única (es decir, amplitud y fase) emitido por la matriz; también

permite visualizar las fuerzas de radiación acústica sobre las partículas levitando. El usuario puede cambiar manualmente la fase inicial y la amplitud de los transductores para explorar el efecto en el campo (Marzo, Ultraino: An Open Phased-Array System for Narrowband Airborne Ultrasound Transmission, 2015).

Para poder visualizar experimentalmente en tiempo real el campo acústico se utilizó la técnica de deflectometría schlieren en su configuración de arcoíris (Contreras, 2021). A continuación, se muestran imágenes comparativas de la distribución de la presión simulada en Ultraino y la distribución de presión obtenida experimentalmente.

En la Figura 4.11. A) se muestra la simulación de presión generada en un levitador que produce ondas estacionarias. Esto se logra al excitar todos los transductores en fase, es decir todos los segmentos (A, B, C, D, E, F, G y H) con la misma fase. La figura B) muestra la distribución de presión que se observa experimentalmente para dicho levitador. Esta similitud entre las distribuciones de presión simuladas numéricamente y medidas experimentalmente confirma la correcta implementación y el funcionamiento adecuado del levitador.

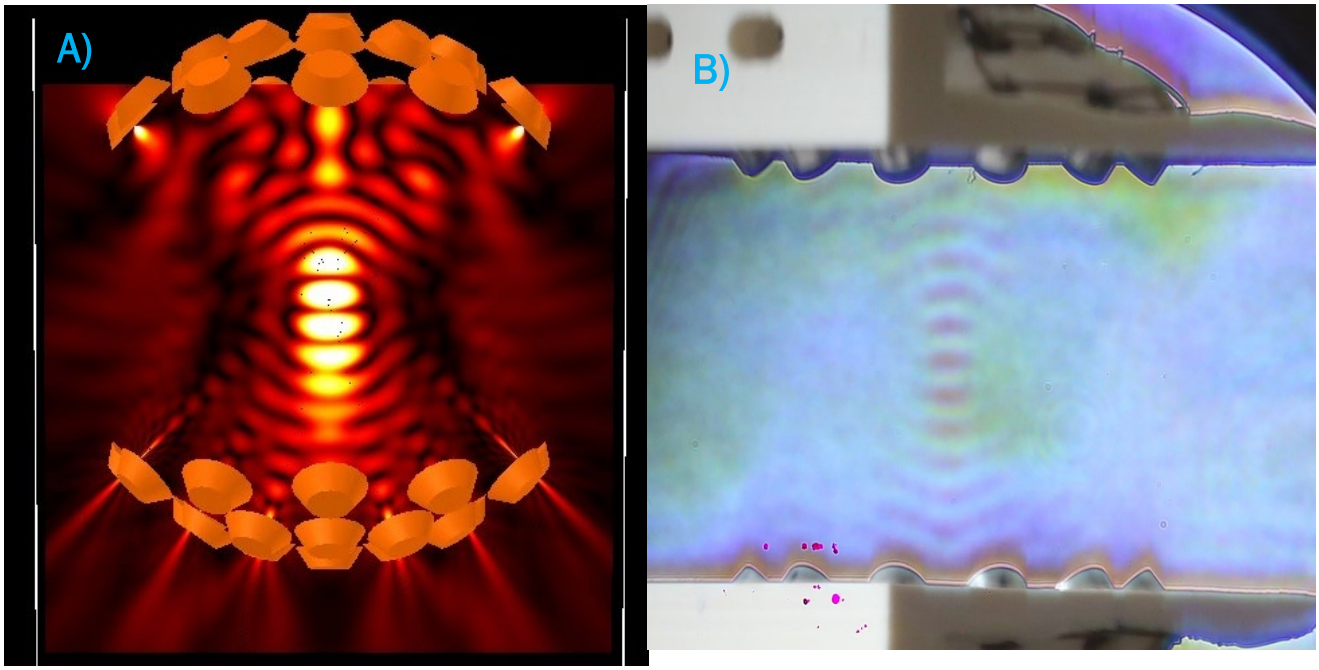


Figura 4.11 Campo acústico con ondas estacionarias. (el color naranja de los transductores representa fase cero).

Para poder generar el siguiente campo acústico lo que se hizo fue poner la mitad de cada matriz con un desfase de 180°, es decir en la primera matriz A y B = 0°, C y D = 180°, en la segunda matriz E y F =0°, G y H = 180°, teniendo como resultado una trampa gemela véase la siguiente referencia (Marzo, Holographic acoustic elements for manipulation of levitated objects, 2015).

Figuras 4.12 A) Se muestra la simulación en la cual se genera un levitador con trampas acústicas gemelas teniendo la configuración antes mencionada. B) Se puede visualizar experimentalmente el campo acústico teniendo una similitud con lo simulado mostrando claramente las trampas gemelas.

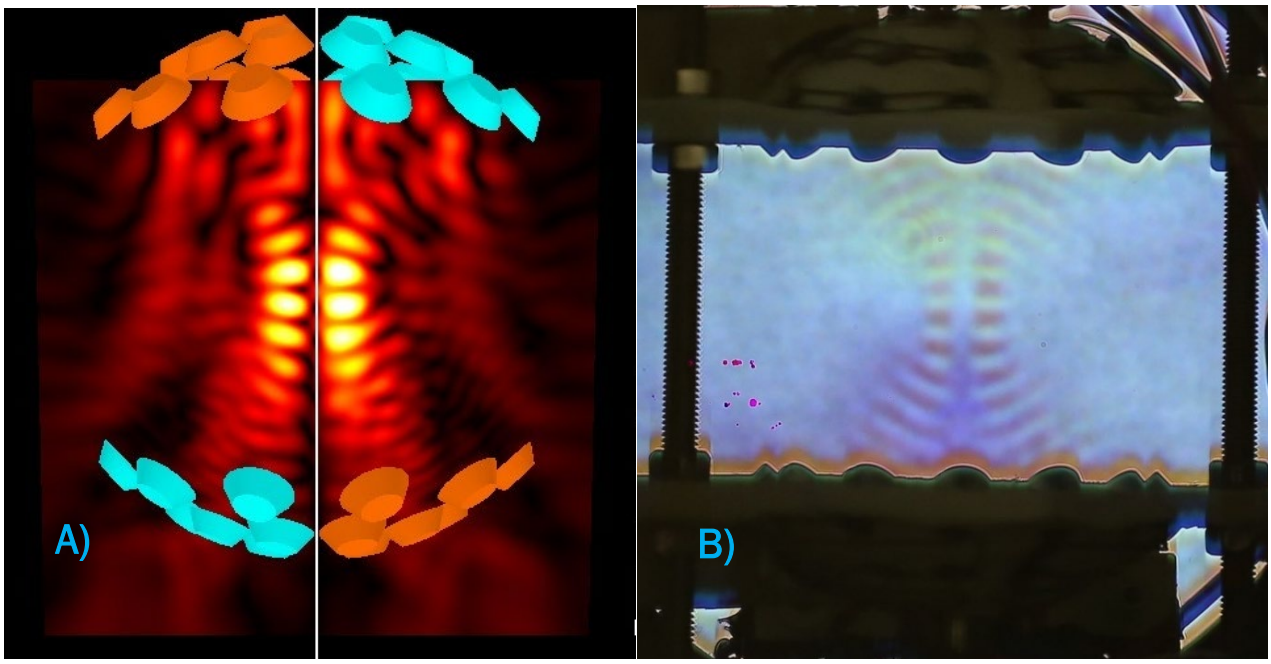


Figura 4.12 Campo acústico trampas gemelas (el color naranja representa fase 0 y el color azul fase de 180°).

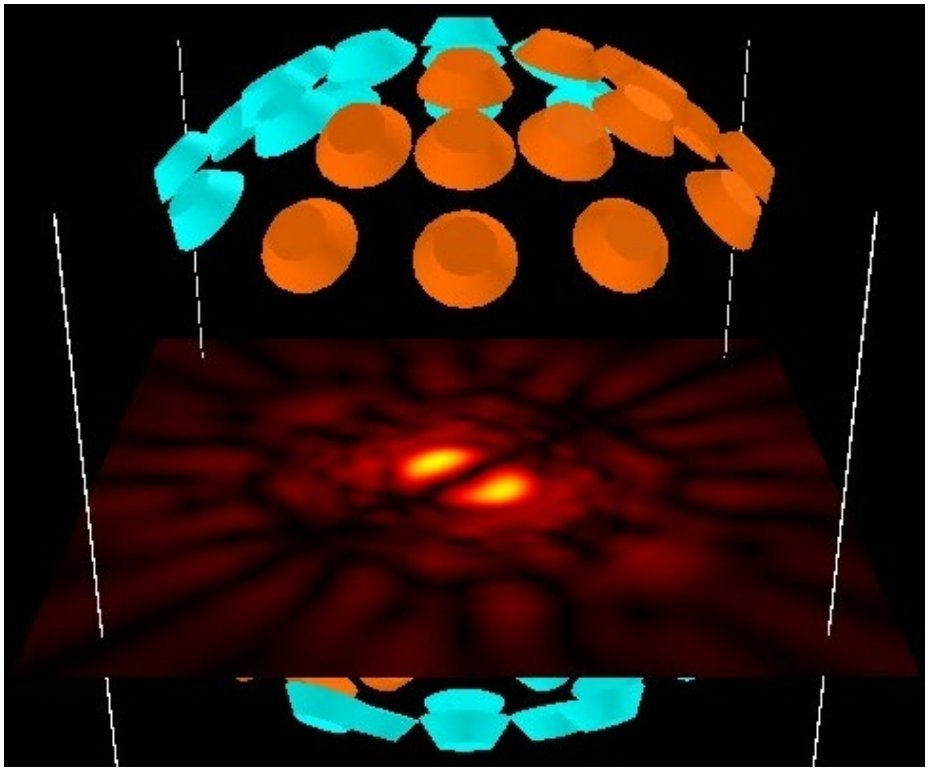


Figura 4.13 Campo acústico trampas gemelas vista superior.

En el caso de los vórtices acústicos, para poder crearlos se necesita superponer dos ondas giratorias como se ve en la parte teórica, una girando en el sentido de las manecillas del reloj y la otra al contrario de las manecillas del reloj, esto se logró poniendo los segmentos de la siguiente manera A y E = 0°, B y F = 90°, C y G = 180° y por último D y H = 270°.

La Figura 4.14. A) Se muestra la simulación en la cual se genera un levitador con un vórtice acústico, lográndose con la anterior configuración. B) Se puede visualizar físicamente el campo acústico teniendo una similitud con lo simulado.

En la Figura 4.15 se muestra una vista superior en la cual se puede observar el vórtice acústico.

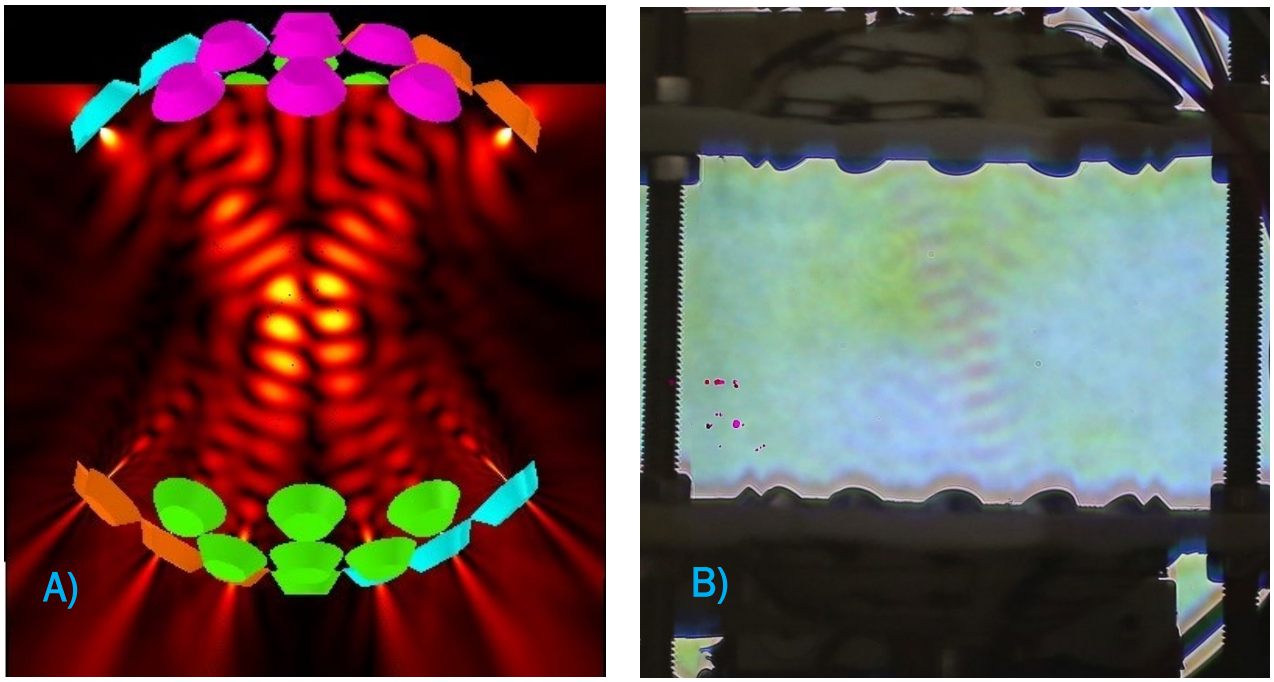


Figura 4.14 Campo acústico Vórtice (cada color indica una fase diferente de 90°).

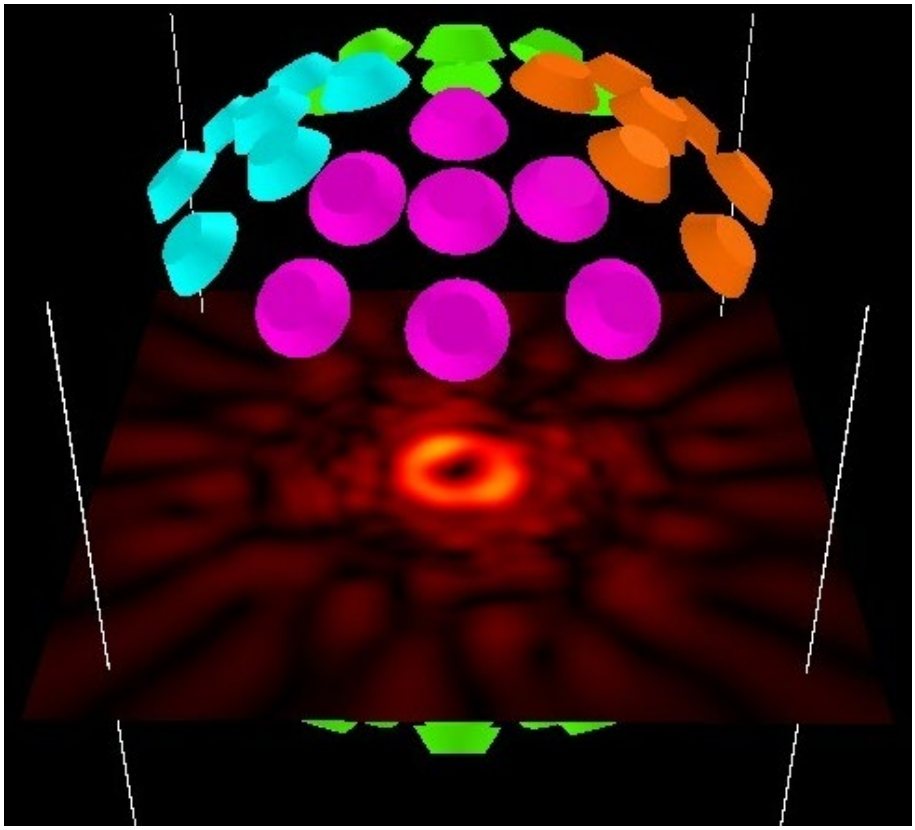


Figura 4.15 Campo acústico trampas gemelas vista superior.

Como resultado final del prototipo se tiene la Figura 4.16 en la se muestra la implementación teniendo un gabinete para su mejor manipulación al igual que se tiene conexiones para conectar los diferentes segmentos que se tiene y un interruptor indicando si este encendido o apagado el sistema.

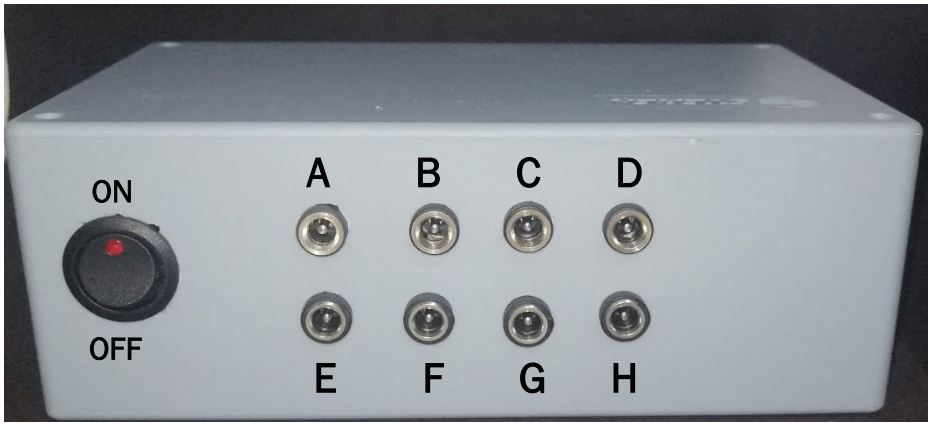


Figura 4.16 Gabinete del prototipo.

En la Figura 4.17 se muestra el arreglo se transductores implementado siendo el producto final del levitador acústico uniaxial con fase variable.

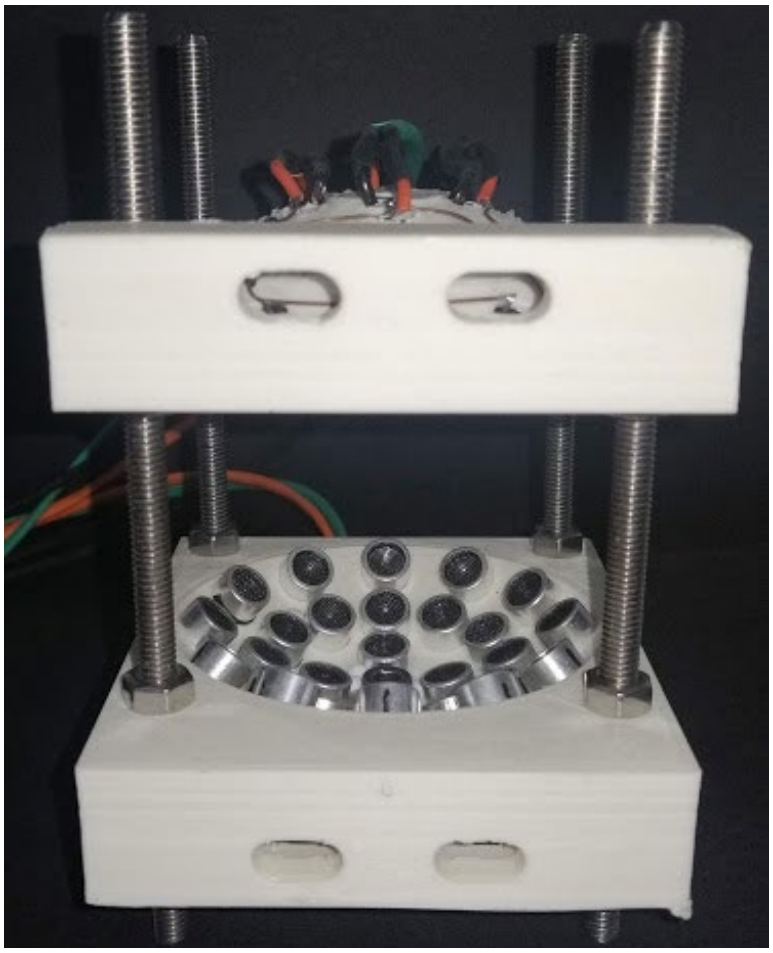


Figura 4.17 Levitador acústico implementado.

5. Capítulo V: CONCLUSIONES Y TRABAJOS A FUTURO

En el apartado presente se mencionan las conclusiones a las que se llegaron al haber culminado el proyecto, también se expresan trabajos futuros que se tienen acerca del proyecto.

5.1 Conclusiones

En este trabajo de tesis se documentó el desarrollo e implementación de un sistema de levitación acústica de fase variable cumpliendo con el objetivo principal del proyecto, el sistema es capaz de generar tres tipos de trampas acústicas en un mismo prototipo, las cuales son: onda estacionaria, trampa gemela y vórtice acústico.

Se ha descrito un dispositivo para generar y caracterizar ondas sonoras giratorias en el espacio libre. En este caso, el campo acústico giratorio se crea superponiendo dos modos ortogonales con un cambio de fase relativo de $\frac{\pi}{2}$ dando como resultado vórtices acústicos de carga topológica $m = 1$, demostrando experimentalmente lo mencionado por la teoría.

En base a los objetivos específicos el empleo de una FPGA tiene ciertas ventajas, entre ellas la flexibilidad que se obtiene al desarrollar un prototipo. Un microprocesador o microcontrolador comercial tiene ciertas funciones que no pueden ser modificadas. En cambio, un FPGA puede ser reprogramado con las funciones específicas que requiere un proyecto y un punto muy importante es que su procesamiento de datos es paralelo lo que permite que sea más rápido que un microprocesador. Las señales que generamos con la FPGA son exactamente 40 KHz teniendo un control de fase de $\frac{\pi}{2}$.

En cuanto a la interfaz de usuario, se diseñó de una manera intuitiva y amigable para el usuario, con dicha interfaz se puede controlar las fases a elección del usuario, la creación de un ejecutable de la interfaz permite al usuario una mejor portación e interacción ya que no requerirá conocimientos del lenguaje en el cual se desarrolló y podrá instalarse en cualquier computadora.

5.2 Trabajos futuros

En el proyecto realizado se consideraron posibles mejoras, las cuales plantean los siguientes trabajos a futuro.

- Adaptar una comunicación Bluetooth, eliminando así esa conexión física entra la PC y el sistema de levitación.
- Manipular la fase de cada uno de los transductores individualmente.
- Generar una resolución de fase más grande de la cual se tiene.
- Analizar los demás campos acústico que se llegaran a generar de acuerdo a los desfases de cada segmento.

Referencias

A. Marzo, A. B. (2017). “TinyLev: A multi-emitter single-axis acoustic levitator,” . *Sci. Instrum.,* , vol. 88, no. 8.

A. O. Santillán and K. Volke-Sepúlveda. (2009). “A demonstration of rotating sound waves in free space and the transfer of their angular momentum to matter,” . *Am. J. Phys.,* , vol. 77, no. 3, pp. 209–215,.

Andrade, M. A. (2017). Review of progress in acoustic levitation. *Institute of Physics, University of Sao Paulo*.

C. Beckhoff, D. K. (2012). A Partial Reconfiguration Fra- mework. En Field-Programmable Custom Computing Machines (FCCM). *IEEE 20th Annual International Symposium on*, 37–44.

Company, T. Q. (2021). *Qt Documentation*. Obtenido de Qt Designer Manual: https://doc.qt.io/qt-5/qtdesigner-manual.html

Contreras, V. (2021). Adjusting single-axis acoustic levitators in real time using rainbow schlieren deflectometry. *American Institute of Physics*.

Igoe, T. (2020). *galaxi0.wordpress.com/*. Obtenido de https://galaxi0.wordpress.com/el-puerto-serial/

Instituto de Ciencias Físicas, UNAM. (2020). *Intituto de Ciencias Físicas*. Obtenido de https://www.fis.unam.mx/laboratorios/42/laboratorio-de-optica-aplicada

Intel. (2013). *Edición web Quartus II*. Obtenido de https://fpgasoftware.intel.com/13.0sp1/

J. Lee, R. P. (2011). ZeroN: mid-air tangible interaction enabled by computer controlled magnetic levitation. *In Proceedings of the 24th annual ACM symposium on User interface software and technology’11(New York, NY)*, 327-336.

Karen Volke Sepúlveda, I. R. (2007). Pinzas ópticas:las delicadas manos de la luz. *Academia Mexicana de Ciencias.*, 18-25.

L. Cox, A. C. (2018). “Acoustic Lock: Position and orientation trapping of non-spherical sub-wavelength particles in mid-air using a single-axis acoustic levitator,” . *Appl. Phys. Lett.,* , vol. 113, no. 5.

Limited, R. C. (10 de 03 de 2021). *PyPi*. Obtenido de PyQt5 5.15.4: https://pypi.org/project/PyQt5/

Lin, S. (1995). Study on the multifrequency Langevin ultrasonic transducer,. *Ultrasonics 33*, 445–448.

Marzo, A. (2015). Holographic acoustic elements for manipulation of levitated objects. *Nature*.

Marzo, A. (2015). Ultraino: An Open Phased-Array System for Narrowband Airborne Ultrasound Transmission. *IEEE Xplore*.

Python. (2021). Obtenido de https://www.python.org/

R. H. Morris, E. R. (2019). “Beyond the Langevin horn: Transducer arrays for the acoustic levitation of liquid drops,”. *Phys. Fluids*, vol. 31, no. 10,.

textoscientificos.com. (s.f.). Obtenido de https://www.textoscientificos.com/redes/comunicaciones/modos

waveshare. (2015). Obtenido de https://www.waveshare.com/wiki/OpenEP4CE6-C_User_Manual

Anexos

Anexo A: Código GUI Gráficos Qt Designer

```
from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(704, 590)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(470, 140, 21, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.comboBox = QtWidgets.QComboBox(self.centralwidget)
        self.comboBox.setGeometry(QtCore.QRect(130, 140, 61, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.comboBox.setFont(font)
        self.comboBox.setObjectName("comboBox")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox.addItem("")
        self.comboBox_2 = QtWidgets.QComboBox(self.centralwidget)
        self.comboBox_2.setGeometry(QtCore.QRect(130, 180, 61, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
```

```
self.comboBox_2.setFont(font)
self.comboBox_2.setObjectName("comboBox_2")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.label_2 = QtWidgets.QLabel(self.centralwidget)
self.label_2.setGeometry(QtCore.QRect(470, 180, 41, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_2.setFont(font)
self.label_2.setObjectName("label_2")
self.comboBox_3 = QtWidgets.QComboBox(self.centralwidget)
self.comboBox_3.setGeometry(QtCore.QRect(130, 220, 61, 31))
font = QtGui.QFont()
font.setPointSize(12)
self.comboBox_3.setFont(font)
self.comboBox_3.setObjectName("comboBox_3")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.label_3 = QtWidgets.QLabel(self.centralwidget)
self.label_3.setGeometry(QtCore.QRect(470, 220, 21, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
```

```
self.label_3.setFont(font)
self.label_3.setObjectName("label_3")
self.comboBox_4 = QtWidgets.QComboBox(self.centralwidget)
self.comboBox_4.setGeometry(QtCore.QRect(130, 260, 61, 31))
font = QtGui.QFont()
font.setPointSize(12)
self.comboBox_4.setFont(font)
self.comboBox_4.setObjectName("comboBox_4")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.label_4 = QtWidgets.QLabel(self.centralwidget)
self.label_4.setGeometry(QtCore.QRect(470, 260, 21, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_4.setFont(font)
self.label_4.setObjectName("label_4")
self.boton1 = QtWidgets.QPushButton(self.centralwidget)
self.boton1.setGeometry(QtCore.QRect(250, 190, 61, 23))
self.boton1.setObjectName("boton1")
self.boton2 = QtWidgets.QPushButton(self.centralwidget)
self.boton2.setGeometry(QtCore.QRect(350, 190, 61, 21))
self.boton2.setObjectName("boton2")
self.etiqueta = QtWidgets.QLabel(self.centralwidget)
self.etiqueta.setGeometry(QtCore.QRect(320, 220, 41, 31))
self.etiqueta.setObjectName("etiqueta")
self.valA = QtWidgets.QLabel(self.centralwidget)
self.valA.setGeometry(QtCore.QRect(510, 150, 47, 13))
```

```
self.valA.setWordWrap(False)
self.valA.setObjectName("valA")
self.cambio = QtWidgets.QPushButton(self.centralwidget)
self.cambio.setGeometry(QtCore.QRect(60, 330, 111, 23))
self.cambio.setObjectName("cambio")
self.valB = QtWidgets.QLabel(self.centralwidget)
self.valB.setGeometry(QtCore.QRect(510, 190, 47, 13))
self.valB.setObjectName("valB")
self.valC = QtWidgets.QLabel(self.centralwidget)
self.valC.setGeometry(QtCore.QRect(510, 230, 47, 13))
self.valC.setObjectName("valC")
self.valD = QtWidgets.QLabel(self.centralwidget)
self.valD.setGeometry(QtCore.QRect(510, 270, 47, 13))
self.valD.setObjectName("valD")
self.unam_logo = QtWidgets.QLabel(self.centralwidget)
self.unam_logo.setGeometry(QtCore.QRect(10, 10, 171, 51))
self.unam_logo.setText("")
self.unam_logo.setPixmap(QtGui.QPixmap(":/logos/unam.png"))
self.unam_logo.setScaledContents(True)
self.unam_logo.setObjectName("unam_logo")
self.label_6 = QtWidgets.QLabel(self.centralwidget)
self.label_6.setGeometry(QtCore.QRect(580, 10, 101, 51))
self.label_6.setText("")
self.label_6.setPixmap(QtGui.QPixmap(":/logos/ICF.png"))
self.label_6.setScaledContents(True)
self.label_6.setObjectName("label_6")
self.label_7 = QtWidgets.QLabel(self.centralwidget)
self.label_7.setGeometry(QtCore.QRect(560, 140, 21, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
```

```
self.label_7.setFont(font)
self.label_7.setObjectName("label_7")
self.valE = QtWidgets.QLabel(self.centralwidget)
self.valE.setGeometry(QtCore.QRect(600, 150, 47, 13))
self.valE.setWordWrap(False)
self.valE.setObjectName("valE")
self.label_8 = QtWidgets.QLabel(self.centralwidget)
self.label_8.setGeometry(QtCore.QRect(560, 180, 41, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_8.setFont(font)
self.label_8.setObjectName("label_8")
self.label_9 = QtWidgets.QLabel(self.centralwidget)
self.label_9.setGeometry(QtCore.QRect(560, 220, 21, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_9.setFont(font)
self.label_9.setObjectName("label_9")
self.valF = QtWidgets.QLabel(self.centralwidget)
self.valF.setGeometry(QtCore.QRect(600, 190, 47, 13))
self.valF.setObjectName("valF")
self.label_10 = QtWidgets.QLabel(self.centralwidget)
self.label_10.setGeometry(QtCore.QRect(560, 260, 21, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_10.setFont(font)
```

```
self.label_10.setObjectName("label_10")
self.valH = QtWidgets.QLabel(self.centralwidget)
self.valH.setGeometry(QtCore.QRect(600, 270, 47, 13))
self.valH.setObjectName("valH")
self.valG = QtWidgets.QLabel(self.centralwidget)
self.valG.setGeometry(QtCore.QRect(600, 230, 47, 13))
self.valG.setObjectName("valG")
self.label_11 = QtWidgets.QLabel(self.centralwidget)
self.label_11.setGeometry(QtCore.QRect(430, 80, 211, 51))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_11.setFont(font)
self.label_11.setObjectName("label_11")
self.label_12 = QtWidgets.QLabel(self.centralwidget)
self.label_12.setGeometry(QtCore.QRect(30, 80, 141, 51))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_12.setFont(font)
self.label_12.setObjectName("label_12")
self.label_13 = QtWidgets.QLabel(self.centralwidget)
self.label_13.setGeometry(QtCore.QRect(50, 140, 51, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_13.setFont(font)
self.label_13.setObjectName("label_13")
self.label_14 = QtWidgets.QLabel(self.centralwidget)
```

```
self.label_14.setGeometry(QtCore.QRect(50, 180, 41, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_14.setFont(font)
self.label_14.setObjectName("label_14")
self.label_15 = QtWidgets.QLabel(self.centralwidget)
self.label_15.setGeometry(QtCore.QRect(50, 220, 41, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_15.setFont(font)
self.label_15.setObjectName("label_15")
self.label_16 = QtWidgets.QLabel(self.centralwidget)
self.label_16.setGeometry(QtCore.QRect(50, 260, 51, 31))
font = QtGui.QFont()
font.setPointSize(12)
font.setBold(True)
font.setWeight(75)
self.label_16.setFont(font)
self.label_16.setObjectName("label_16")
self.label_5 = QtWidgets.QLabel(self.centralwidget)
self.label_5.setGeometry(QtCore.QRect(200, 320, 211, 221))
self.label_5.setText("")
self.label_5.setPixmap(QtGui.QPixmap(":/logos/INFERIOR.jpg"))
self.label_5.setScaledContents(True)
self.label_5.setObjectName("label_5")
self.label_17 = QtWidgets.QLabel(self.centralwidget)
self.label_17.setGeometry(QtCore.QRect(420, 320, 211, 221))
self.label_17.setText("")
```

```
        self.label_17.setPixmap(QtGui.QPixmap(":/logos/SUPERIOR.jpg"))
        self.label_17.setScaledContents(True)
        self.label_17.setObjectName("label_17")
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label.setToolTip(_translate("MainWindow", "<html><head/><body><p align=\"center\"><span
style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label.setText(_translate("MainWindow", "A"))
        self.comboBox.setItemText(0, _translate("MainWindow", "--"))
        self.comboBox.setItemText(1, _translate("MainWindow", "0°"))
        self.comboBox.setItemText(2, _translate("MainWindow", "90°"))
        self.comboBox.setItemText(3, _translate("MainWindow", "180°"))
        self.comboBox.setItemText(4, _translate("MainWindow", "270°"))
        self.comboBox_2.setItemText(0, _translate("MainWindow", "--"))
        self.comboBox_2.setItemText(1, _translate("MainWindow", "0°"))
        self.comboBox_2.setItemText(2, _translate("MainWindow", "90°"))
        self.comboBox_2.setItemText(3, _translate("MainWindow", "180°"))
        self.comboBox_2.setItemText(4, _translate("MainWindow", "270°"))
        self.label_2.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_2.setText(_translate("MainWindow", "B"))
        self.comboBox_3.setItemText(0, _translate("MainWindow", "--"))
        self.comboBox_3.setItemText(1, _translate("MainWindow", "0°"))
        self.comboBox_3.setItemText(2, _translate("MainWindow", "90°"))
        self.comboBox_3.setItemText(3, _translate("MainWindow", "180°"))
        self.comboBox_3.setItemText(4, _translate("MainWindow", "270°"))
```

```
        self.label_3.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_3.setText(_translate("MainWindow", "C"))
        self.comboBox_4.setItemText(0, _translate("MainWindow", "--"))
        self.comboBox_4.setItemText(1, _translate("MainWindow", "0°"))
        self.comboBox_4.setItemText(2, _translate("MainWindow", "90°"))
        self.comboBox_4.setItemText(3, _translate("MainWindow", "180°"))
        self.comboBox_4.setItemText(4, _translate("MainWindow", "270°"))
        self.label_4.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_4.setText(_translate("MainWindow", "D"))
        self.boton1.setText(_translate("MainWindow", "ON"))
        self.boton2.setText(_translate("MainWindow", "OFF"))
        self.etiqueta.setText(_translate("MainWindow", "Estado"))
        self.valA.setText(_translate("MainWindow", "--"))
        self.cambio.setText(_translate("MainWindow", "Cambiar desfase"))
        self.valB.setText(_translate("MainWindow", "--"))
        self.valC.setText(_translate("MainWindow", "--"))
        self.valD.setText(_translate("MainWindow", "--"))
        self.label_7.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_7.setText(_translate("MainWindow", "E"))
        self.valE.setText(_translate("MainWindow", "--"))
        self.label_8.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_8.setText(_translate("MainWindow", "F"))
        self.label_9.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_9.setText(_translate("MainWindow", "G"))
        self.valF.setText(_translate("MainWindow", "--"))
        self.label_10.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_10.setText(_translate("MainWindow", "H"))
        self.valH.setText(_translate("MainWindow", "--"))
```

```
        self.valG.setText(_translate("MainWindow", "--"))
        self.label_11.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_11.setText(_translate("MainWindow", "Estado de los segmentos:"))
        self.label_12.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_12.setText(_translate("MainWindow", "Desfase:"))
        self.label_13.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_13.setText(_translate("MainWindow", "A y E"))
        self.label_14.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_14.setText(_translate("MainWindow", "B y F"))
        self.label_15.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_15.setText(_translate("MainWindow", "C y G"))
        self.label_16.setToolTip(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:12pt;\">A</span></p></body></html>"))
        self.label_16.setText(_translate("MainWindow", "D y H"))
import logos_rc


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Anexo B: Código funcional GUI

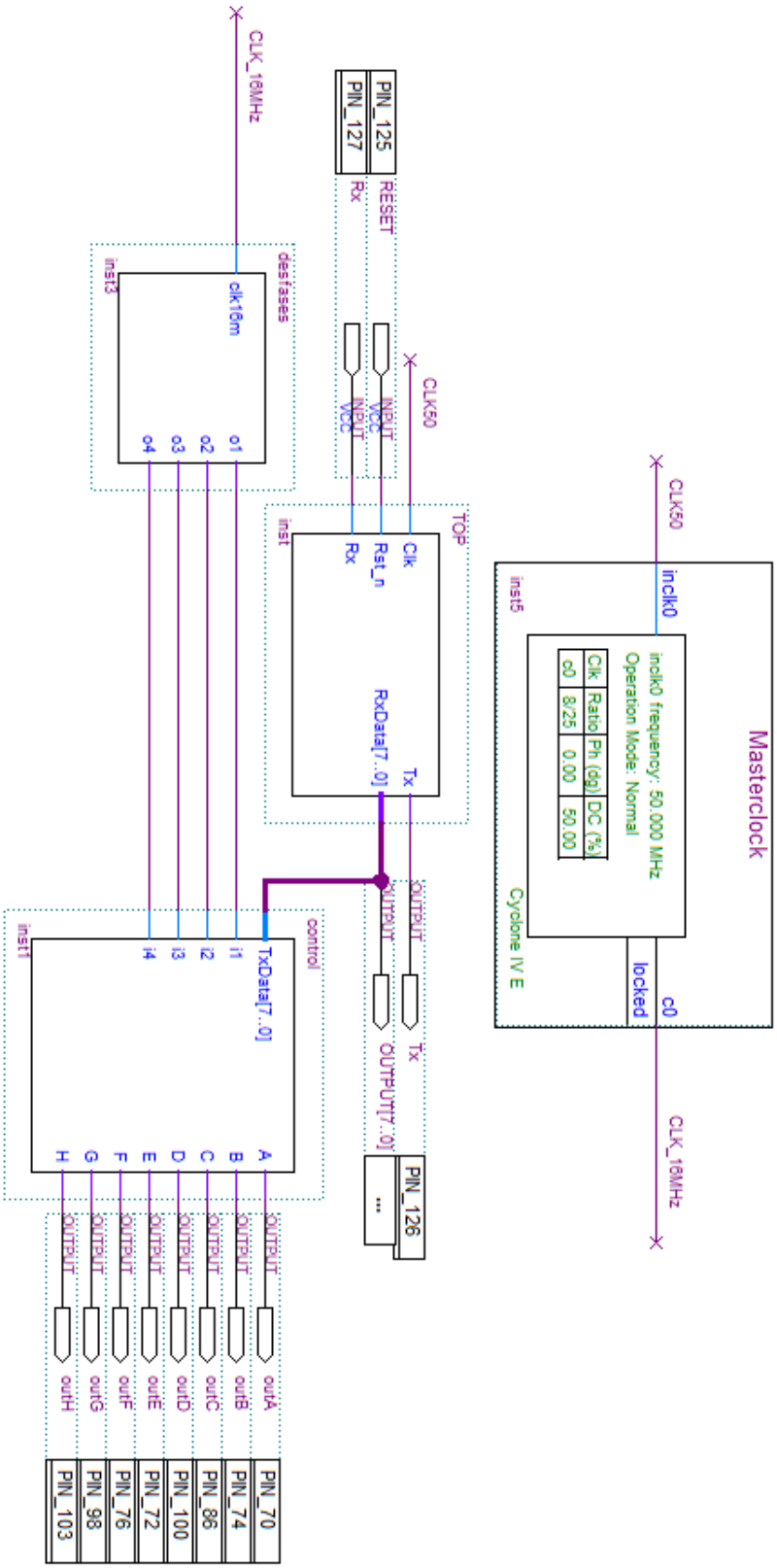
```
#Importamos todas las bibliotecas, clases y herramientas necesarias
import sys
import serial
```

```
import time
import logos
from PyQt5 import*
from PyQt5 import uic
from PyQt5 import QtGui
from PyQt5.QtWidgets import *
from PyQt5.uic import loadUi
from numpy import *

# Inicializamos nuestro puerto serial con 250000 baudios en el com5
fpga = serial.Serial('com5', 250000, timeout=1)

#Se crea una clase de la GUI
class ejemplo_GUI(QMainWindow):

    def __init__(self):
        super().__init__()
        loadUi("inter_app.ui",self) #Llamamos a la clase creada por Qt Designer
        self.setWindowTitle("LevInterface") #Le damos el nombre a nuestra aplicacion
        self.cambio.clicked.connect(self.valores) #Llammamos a nuestra funcion valores siempre que se presione el boton cambio



    #Esta funcion recolecta los datos de los menus despegables
    def valores(self):
        valor = self.comboBox.currentText()#obtenemos el valos de A y E
        if valor == "0°":
            a = b'\x00'
        elif valor == "90°":
            a = b'\x01'
        elif valor == "180°":
            a = b'\x02'
```

```
elif valor == "270°":
    a = b'\x03'


valor2 = self.comboBox_2.currentText() #obtenemos el valos de B y F
if valor2 == "0°":
    b = b'\x00'
elif valor2 == "90°":
    b = b'\x04'
elif valor2 == "180°":
    b = b'\x08'
elif valor2 == "270°":
    b = b'\x0c'

valor3 = self.comboBox_3.currentText() #obtenemos el valos de C y G
if valor3 == "0°":
    c = b'\x00'
elif valor3 == "90°":
    c = b'\x10'
elif valor3 == "180°":
    c = b'\x20'
elif valor3 == "270°":
    c = b'\x30'

valor4 = self.comboBox_4.currentText() #obtenemos el valos de D y H
if valor4 == "0°":
    d = b'\x00'
elif valor4 == "90°":
    d = b'\x40'
elif valor4 == "180°":
    d = b'\x80'
elif valor4 == "270°":
```

```
            d = b'\xc0'
        # Sumamos nuestros valores de cada seccion (A-H)
        arreglo = a[0] + b[0] + c[0] + d[0]
        Inferior = bytes([arreglo])#convertimos en bytes nuestros valores para poder enviarlos
        # Escribimos en nuestro puerto serial el valor que tengamos en la variable "Inferior"
        fpga.write(Inferior)
        #ponemos en las etiquetas los valores de cadaseccion (A-H)
        self.valA.setText(valor)
        self.valE.setText(valor)
        self.valB.setText(valor2)
        self.valF.setText(valor2)
        self.valC.setText(valor3)
        self.valG.setText(valor3)
        self.valD.setText(valor4)
        self.valH.setText(valor4)

if __name__ == '__main__':
        app = QApplication(sys.argv) #Creamos una instancia de QApplication
        GUI = ejemplo_GUI() #Crear una instancia de la GUI de su aplicación.
        GUI.show() #Muestre la GUI de su aplicación.
        sys.exit(app.exec_()) #salimos de nuestra aplicacion
        fpga.close() #Cerramos puerto serial
```

Anexo C: Esquemático implementado en Quartus II



Anexo D: Esquemático FPGA Cyclone IV CoreEP4CE6