# UNIVERSIDAD NACIONAL DEL CALLAO

## FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

## ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

# "DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE SEÑALES DE RADIOFRECUENCIA BASADO EN FPGA SoC PARA LA OPERACIÓN DE UN TRANSMISOR DE RADAR IONOSONDA EN EL RADIO OBSERVATORIO DE JICAMARCA"

**TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO**


**Autor :**
**B.Sc. Brayan Lui Estalla Quinteros**

**Asesor :**
**M.Sc. Jaime Alberto Vallejos Laos**

**Co-asesor :**
**Dr. Marco Antonio Milla Bravo**


**Línea de Investigación : Ingeniería Y Tecnología**

**Callao – 2024**

**PERÚ**

# Tesis_Brayan Lui Estalla Quinteros

**14%** Textos sospechosos

4% Similitudes
< 1% similitudes entre comillas
0% entre las fuentes mencionadas
11% Idiomas no reconocidos

**Nombre del documento:** Tesis_Brayan Lui Estalla Quinteros.pdf
**ID del documento:** bca97d96cb2b0713012bddeb1abe7e451edba11a
**Tamaño del documento original:** 13,58 MB

**Depositante:** FIEE PREGRADO UNIDAD DE INVESTIGACION
**Fecha de depósito:** 21/3/2024
**Tipo de carga:** interface
**fecha de fin de análisis:** 21/3/2024

**Número de palabras:** 22.907
**Número de caracteres:** 169.509

Ubicación de las similitudes en el documento:

## Fuentes de similitudes

### Fuentes principales detectadas

| N° | Descripciones | Similitudes | Ubicaciones | Datos adicionales |
|---|---|---|---|---|
| 1 | unac.edu.pe<br>https://unac.edu.pe/wp-content/uploads/documentos/transparencia/articulo-11/11-2/transparencia-... | 2% | | Palabras idénticas: 2% (431 palabras) |
| 2 | repositorio.ues.edu.sv<br>https://repositorio.ues.edu.sv/bitstreams/3a9a94da-8178-4b34-9fc4-1bfe7ae6f3a6/download | < 1% | | Palabras idénticas: < 1% (149 palabras) |
| 3 | oa.upm.es<br>https://oa.upm.es/47722/1/JHON_JAIRO_BARONA_MENDOZA.pdf<br>1 fuente similar | < 1% | | Palabras idénticas: < 1% (97 palabras) |
| 4 | ru.dgb.unam.mx<br>https://ru.dgb.unam.mx/bitstream/20.500.14330/TES01000617587/3/0617587.pdf | < 1% | | Palabras idénticas: < 1% (90 palabras) |
| 5 | revistatelematica.cujae.edu.cu<br>https://revistatelematica.cujae.edu.cu/index.php/tele/article/download/103/102/307 | < 1% | | Palabras idénticas: < 1% (80 palabras) |

### Fuentes con similitudes fortuitas

| N° | Descripciones | Similitudes | Ubicaciones | Datos adicionales |
|---|---|---|---|---|
| 1 | INFORME FINAL DE TESIS (1).pdf \| INFORME FINAL DE TESIS (1) #2b4a0f<br>El documento proviene de mi biblioteca de referencias | < 1% | | Palabras idénticas: < 1% (26 palabras) |
| 2 | abacoenred.com<br>https://abacoenred.com/wp-content/uploads/2019/02/El-proyecto-de-investigación-F.G.-Arias-2012-... | < 1% | | Palabras idénticas: < 1% (25 palabras) |
| 3 | www.itq.edu.mx<br>http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes de materias/Apuntes_V... | < 1% | | Palabras idénticas: < 1% (18 palabras) |
| 4 | ru.dgb.unam.mx<br>https://ru.dgb.unam.mx/bitstream/20.500.14330/TES01000678100/3/0678100_A1.pdf | < 1% | | Palabras idénticas: < 1% (15 palabras) |
| 5 | inaoe.repositorioinstitucional.mx<br>https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/2028/1/RicardezTMR.pdf | < 1% | | Palabras idénticas: < 1% (18 palabras) |

### Fuentes mencionadas (sin similitudes detectadas)

Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1 http://jro.igp.gob.pe/digisonde/dps4text.htm
2 http://www.digisonde.com/pdf/Digisonde4DManual_LDI-web.pdf
3 https://redpitaya.com/

| INFORMACIÓN BÁSICA | |
|---|---|
| **FACULTAD** | FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA |
| **UNIDAD DE INVESTIGACIÓN** | DE LA FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA |
| **TÍTULO** | "DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE SEÑALES DE RADIOFRECUENCIA BASADO EN FPGA SoC PARA LA OPERACIÓN DE UN TRANSMISOR DE RADAR IONOSONDA EN EL RADIO OBSERVATORIO DE JICAMARCA" |
| **AUTOR** | ❖ B.Sc. ESTALLA QUINTEROS BRAYAN LUI<br>❖ CÓDIGO ORCID: 0000-0001-8897-0367<br>❖ DNI: 76658936 |
| **ASESOR**<br><br>**CO-ASESOR** | ❖ M.Sc.. JAIME ALBERTO VALLEJOS LAOS<br>❖ CÓDIGO ORCID: 0000-0003-4519-4657<br>❖ DNI: 08786103<br>❖ Dr. MARCO ANTONIO MILLA BRAVO<br>❖ CÓDIGO ORCID: 0000-0001-9067-863X<br>❖ DNI: 09860554 |

| | |
|---|---|
| **LUGAR DE EJECUCIÓN** | RADIO OBSERVATORIO DE JICAMARCA |
| **UNIDADES DE ANÁLISIS** | PARÁMETROS DE OPERACIÓN DE UN TRANSMISOR DE RADAR IONOSONDA EN EL RADIO OBSERVATORIO DE JICAMARCA |
| **TIPO DE INVESTIGACIÓN** | TIPO APLICADA, DISEÑO EXPERIMENTAL, NIVEL EXPLICATIVO, ENFOQUE CUANTITATIVO |
| **TEMA OCDE** | INGENIERÍA Y TECNOLOGÍA |

# UNIVERSIDAD NACIONAL DEL CALLAO
# FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
# ACTA PARA LA OBTENCIÓN DEL TÍTULO PROFESIONAL POR LA MODALIDAD DE TESIS SIN CICLO DE TESIS

Al día 04 del mes de junio de 2024 siendo las 11:00 horas se reunió el Jurado Examinador de la Facultad de Ingeniería Eléctrica y Electrónica de la Universidad Nacional del Callao, aprobada mediante Resolución Decanal N°126-2024-DFIEE, conformado por los siguientes docentes ordinarios:

| | |
|---|---|
| **Dr. Ing. JACOB ASTOCONDOR VILLAR** | **Presidente** |
| **M. Sc. Ing. JULIO CESAR BORJAS CASTAÑEDA** | **Secretario** |
| **Dr. Ing. FERNANDO MENDOZA APAZA** | **Vocal** |
| **Dr. Ing. ABILIO BERNARDINO CUZCANO RIVAS** | **Suplente** |

Asimismo se dio inicio a la exposición de TESIS del señor Bachiller **ESTALLA QUINTEROS, Brayan Lui;** quienes habiendo cumplido con los requisitos para obtener el Título Profesional en Ingeniería Electrónica como lo señalan los Arts. N°s 08 al 10 del Reglamento de Grados y Títulos, sustentará la Tesis Titulada: **"DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE SEÑALES DE RADIOFRECUENCIA BASADO EN FPGA SoC PARA LA OPERACIÓN DE UN TRANSMISOR DE RADAR IONOSONDA EN EL RADIO OBSERVATORIO DE JICAMARCA"** con el quórum Reglamentario de Ley, se dio inicio a la exposición, considerando lo establecido en el Art. N° 80 del Reglamento de Grados y Títulos dado por Resolución N° 150-23-CU, en el Sub Capítulo II, corresponde al otorgamiento del Título Profesional con Tesis sin Ciclo de Tesis, efectuadas las deliberaciones pertinentes se acordó:

Dar por APROBADO Calificativo MUY BUENO nota: 18 al expositor **ESTALLA QUINTEROS, Brayan Lui;** con lo cual se dio por concluida la sesión, siendo las 12:00 horas del día del mes y año en curso.

Es copia fiel del folio N° 261 del Libro de Actas de Sustentación de Tesis de la Facultad de Ingeniería Eléctrica y Electrónica – UNAC.

**Dr. Ing. JACOB ASTOCONDOR VILLAR**
**PRESIDENTE**

**M. Sc. Ing. JULIO CESAR BORJAS CASTAÑEDA**
**SECRETARIO**

**Dr. Ing. FERNANDO MENDOZA APAZA**
**VOCAL**

**Dr. Ing. ABILIO BERNARDINO CUZCANO RIVAS**
**SUPLENTE**

**UNIVERSIDAD NACIONAL DEL CALLAO**
**FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA**
**ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA**

# INFORME FAVORABLE

EL PRESIDENTE DEL JURADO EVALUADOR DE SUSTENTACIÓN DE TESIS DE LA ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA DE LA FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA DE LA UNIVERSIDAD NACIONAL DEL CALLAO; que suscribe:

Deja constancia de que el día martes 04 de junio de 2024, el señor Bachiller en Ingeniería Electrónica **ESTALLA QUINTEROS, Brayan Lui;** quien habiendo cumplido con los requisitos establecidos en la normativa, sustento la Tesis titulada: **"DISEÑO E IMPLEMENTACIÓN DE UN GENERADOR DE SEÑALES DE RADIOFRECUENCIA BASADO EN FPGA SoC PARA LA OPERACIÓN DE UN TRANSMISOR DE RADAR IONOSONDA EN EL RADIO OBSERVATORIO DE JICAMARCA",** habiendo obtenido la nota de **18 (dieciocho)**, según el folio 261 del Libro de Actas de Sustentación.

Se extiende el presente en cumplimiento de lo establecido en el Reglamento de Grados y Títulos de Pregrado de la Universidad Nacional del Callao.

Callao, 04 de junio de 2024

.....
**DR. ING. JACOB ASTOCONDOR VILLAR**
**Presidente del Jurado Evaluador**

# DEDICATORIA

A Dios por guiarme siempre, a mis padres Guisela Quinteros y Luis Estalla, a mis hermanos Luis, Cristhian, Vicente y Jazmin quienes me apoyaron incondicionalmente, a Jhassmin por siempre alentarme en todo momento.

# AGRADECIMIENTO

A mi madre Guisela Quinteros que siempre estuvo en todo momento alentándome, al Dr. Edgar del Aguila por su amistad, consejos brindados y siempre motivarme a investigar e inventar. Al M.Sc. Jaime Vallejos por sus enseñanzas en mi etapa universitaria y el apoyo brindado como asesor.

Al Dr. Marco Milla de quien estoy agradecido por la confianza puesta en mí al desarrollar esta investigación, paciencia y conocimientos brindados. Al BSc. Joaquín Verástegui por los grandes momentos compartidos en el laboratorio de IDI y compartir conmigo esa pasión por el hardware y FPGA`s

A Jhassmin por su amor, confiar en mí y bellos momentos compartidos.

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# RESUMEN

El presente trabajo de tesis tiene como objetivo principal el diseño e implementación de un generador de señales de radiofrecuencia para transmisor de radar ionosonda que está basado en el SDR Red Pitaya Signal Lab 250-12 que comprende de un FPGA SoC Zynq 7020 y puede transmitir señales moduladas con un barrido de frecuencias desde 1 MHz hasta 25 MHz. Para el diseño se utiliza el entorno de desarrollo Vivado de Xilinx-AMD. La síntesis del hardware se basa en el lenguaje VHDL y el estilo de descripción comportamental para los módulos controlador SPI, mapa de registros, oscilador controlado numéricamente (NCO), modulador BPSK y OOK, multiplexor y un módulo de sincronismo con entrada de clock de referencia de GPS de 10Mhz y trigger para el inicio de envío de las señales. Luego se agrega el Ip Core Clocking Wizard que nos permite elevar la frecuencia interna de clock del FPGA a 250 MHz.

Los módulos mencionados e Ip Cores se integran utilizando el estilo de descripción estructural con la herramienta Create Design Block en el software Vivado. Finalmente se realizaron simulaciones con banco de pruebas (test bench), pruebas de funcionamiento en el laboratorio de IDI y en la estación del radar ionosonda “Vertical Incidence Pulsed Ionospheric Radar” (VIPIR).

**Palabras claves:** ionosonda, Red Pitaya, SDR, sondeo ionosférico.

# ABSTRACT

The main objective of this thesis work is the design and implementation of a radiofrequency signal generator for an ionosonde radar transmitter that is based on the Red Pitaya Signal Lab 250-12 SDR that comprises a Zynq 7020 SoC FPGA and can transmit signals. modulated with a frequency sweep from 1 MHz to 25 MHz.

The Xilinx-AMD Vivado development environment is used for the design. The hardware synthesis is based on the VHDL language and behavioral description style for the SPI controller modules, register map, numerically controlled oscillator (NCO), BPSK and OOK modulator, multiplexer, and a synchronization module with reference clock input 10Mhz GPS and trigger to start sending signals. Then the IP Core Clocking Wizard is added that allows us to raise the internal clock frequency of the FPGA to 250 MHz.

The mentioned modules and IP Cores are integrated using the structural description style with the Create Design Block tool in the Vivado software. Finally, simulations were carried out with a test bench, and functional tests in the IDI laboratory and at the “Vertical Incidence Pulsed Ionospheric Radar” (VIPIR) ionosonde radar station.



**Keywords:** ionosonde, Pitaya Network, SDR, ionospheric survey.

# INTRODUCCIÓN

Con los avances tecnológicos actuales el estudio de la aeronomía es de suma importancia para los sistemas de posicionamiento global basados en satélites (GNSS) y las aplicaciones derivadas de ellos, por ejemplo, la radionavegación terrestre, marítima y aérea.

En diversas partes del mundo, se han realizado numerosos estudios con el fin de analizar el comportamiento de la propagación de señales de radio en la ionosférica, dichos estudios arrojaron como resultado la presencia de errores en la recepción de datos, derivados de los retardos en la propagación [1].

Una ionosonda es un radar de HF que envía un barrido de pulsos electromagnéticos entre 1 a 25mhz hacia la ionosfera las cuales son reflejadas y recibidas posteriormente para procesar el eco. Los ionogramas son la relación frecuencia de sondeo versus altura virtual de reflexión y nos dan una idea aproximada de la densidad electrónica de la ionósfera [2]. Por lo que, controlar la operación de un transmisor de radar ionosférico permitiría estudiar las capas altas de la atmósfera además de cumplir un rol importante en la obtención de densidades de electrones y corregir errores en los sistemas GNSS.

En el presente proyecto de investigación se busca diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para operar un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

# I. PLANTEAMIENTO DEL PROBLEMA

## 1.1. Descripción de la realidad problemática

En el territorio peruano la propagación de radiofrecuencias se ve afectada por su interacción con la ionósfera sin poderse corregir, dando como resultado diferentes fenómenos que alteran su comportamiento de manera impredecible.
El Radio Observatorio de Jicamarca (ROJ) que es sede científica del Instituto Geofísico del Perú (IGP) realiza estudios de aeronomía con la finalidad de comprender las capas altas de la atmósfera específicamente la ionósfera, en las cuales los procesos de ionización y recombinación son importantes.
Las ionosondas son instrumentos científicos que nos permiten obtener los parámetros ionosféricos y sus efectos en las señales del sistema global de navegación por satélite (GNSS).
Existen ionosondas como Digisonde que son instrumentos reconocidos mundialmente. Sin embargo, son muy costosos el importarlos y darle el mantenimiento respectivo. Y en orden de exitosamente realizar estudios de aeronomía, un generador de señales de radiofrecuencia para operar un transmisor de radar ionosonda es requerido.
En el presente proyecto de investigación se propondrá un diseño de un generador de señales de radiofrecuencia basado en Red Pitaya que comprende de un FPGA SoC para operar un transmisor de radar ionosonda, dicho generador transmitirá una señal modulada con un sweep de frecuencias desde 1 MHz hasta 25 MHz el cual permite el estudio de la ionósfera en el Radio Observatorio de Jicamarca.

## 1.2. Formulación del problema

### 1.2.1. Problema general

- **Problema general:** ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para operar un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?

### 1.2.2. Problemas específicos

- **Problema específico 1:** ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?
- **Problema específico 2:** ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para emitir un barrido de frecuencias en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?
- **Problema específico 3:** ¿Es posible diseñar e implementar un circuito modulador de señales de radiofrecuencia basado en FPGA SoC?

## 1.3. Objetivos

### 1.3.1. Objetivos generales

- **Objetivo general**: Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para operar un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

### 1.3.2. Objetivos específicos

- **Objetivo específico 1:** Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

- **Objetivo específico 2:** Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para emitir un barrido de frecuencias en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

- **Objetivo específico 3:** Diseñar e implementar un circuito modulador de señales de radiofrecuencia basado en FPGA SoC.

## 1.4. Justificación

### 1.4.1. Justificación Teórica

La justificación teórica tiene como propósito generar reflexión y debate académico sobre el conocimiento existente, contrastar resultados o hacer epistemología del conocimiento existente [5].

De lo expuesto del autor, el presente trabajo de investigación tiene una justificación teórica ya que se estudiará la teoría de radares ionosféricos para el nuevo diseño de generador de señales para radar ionosonda que aporta con el conocimiento y la utilización de la misma en otros trabajos de investigación.

### 1.4.2. Justificación Práctica

Se tiene justificación práctica cuando el desarrollo de un proyecto ayuda a resolver un problema o, por lo menos, propone estrategias que al aplicarse contribuirían a resolverlo [5].

De lo expuesto por el autor, el presente trabajo de investigación tiene una justificación práctica puesto que el generador de señales de radiofrecuencia permitirá operar el transmisor de radar ionosonda para el estudio de la aeronomía.

### 1.4.3. Justificación Metodológica

La justificación metodológica se refiere cuando un proyecto propone un nuevo método o una nueva estrategia para generar conocimiento válido y confiable [5].

De lo expuesto por el autor, el presente trabajo de investigación tiene una justificación metodológica ya que en él se propone un esquema metodológico innovador que tiene como finalidad un nuevo diseño de generador de señales de radiofrecuencia para el transmisor de radar ionosonda que será puesto en evaluación en todo el periodo de implementación del mismo.

### 1.4.4. Justificación Social

Se refiere a cuál es su trascendencia para la sociedad, quiénes o en qué modo se beneficiarán con los resultados de la investigación, en conclusión, ¿qué alcance o proyección social tiene? [6].

De lo expuesto por el autor, el presente trabajo de investigación tiene una justificación social puesto que al operar el transmisor de radar ionosonda se obtendrá información de los fenómenos ionosféricos que beneficiará a la sociedad.

## 1.5. Limitaciones de la investigación

### 1.5.1. Limitación teórica

El presente trabajo de investigación presenta una limitación teórica dado que existen pocas referencias bibliográficas o escasa información sobre el diseño de generadores de señales de radiofrecuencia para radares ionosféricos.

# II. MARCO TEÓRICO

## 2.1. Antecedentes: Internacional y nacionales

### 2.1.1. Antecedentes internacionales

- En el artículo científico titulado: ”Implementación de una Ionosonda electrónica para la supervisión de la ionosfera desde la Tierra vía una columna proyectada a través de USRP” [2] publicado por MULTIDISCIPLINARY DIGITAL PUBLISHING INSTITUTE (MDPI), se tuvo como objetivo plantear una propuesta de monitoreo del contenido total de electrones de la ionosfera desde el punto de vista instrumental. Se utilizó la metodología de diseño de instrumentación científica, se diseña un radar para sondeo ionosférico basado en tecnología USRP con la finalidad de conocer el comportamiento o respuesta de cada modelo en una zona de alta actividad ionosférica, dentro de la franja de anomalía ecuatorial (diseño experimental, enfoque cuantitativo y nivel descriptivo). Se llegó a la conclusión de que la implementación de una ionosonda usando tecnología USRP cumple con las características deseadas y el post-procesamiento es capaz de imprimir las alturas de la ionosfera para el rango diseñado.

  De este trabajo de investigación se puede destacar cómo implementaron un transmisor de radar ionosonda para generar la forma de los pulsos de radar y el procesamiento de la información de los ecos provenientes de la ionosfera.

- En el artículo científico titulado: ”Diseño en FPGA de una ionosonda generación” [3] indexado en la revista digital de las tecnologías de la información y las comunicaciones, se tuvo como objetivo el diseño de un radar de sondeo vertical de la ionosfera, para ello se utilizó la metodología de la técnica de modulación por código binario de fase y compresión de pulsos (diseño experimental, enfoque cuantitativo y nivel

descriptivo). Llegando a la conclusión de que las técnicas actuales de compresión de pulsos y modulación por código binario de fase aumenta la relación señal a ruido además de proporcionar una ganancia de procesamiento lo cual se comprueba mediante simulaciones realizadas en los softwares Simulink y Quartus. Finalmente, el autor recomienda culminar la simulación de la integración coherente debido a que la computadora no cumplía con los requerimientos para tal simulación y los resultados preliminares obtenidos de las simulaciones constituyen un punto de partida para la futura implementación física del diseño de una ionosonda en el Instituto de Geofísica y Astronomía en Cuba.

Del artículo desarrollado por el autor, se aprecia cómo utilizan el software Quartus para el diseño, simulación y síntesis de hardware del transmisor de radar ionosonda basado en FPGA.

### 2.1.2. Antecedentes nacionales

- Proyecto de investigación titulado: ”Implementación de sistema de radar ionosonda en RED PITAYA” [3] publicado en el repositorio de prácticas preprofesionales del Radio Observatorio de Jicamarca (ROJ), tuvo como objetivo el diseño de un sistema de radar ionosonda basado en red pitaya que transmite una onda modulada con un sweep de frecuencias desde 1 MHz hasta 25 MHz, teniendo la posibilidad de modular la señal portadora durante cada step del sweep, para ello se utilizó la metodología de diseño electrónico por bloques del radar ionosonda enfocado a pruebas de laboratorio, que tiene como primer paso la recolección de los requisitos funcionales, luego levantar los diagramas esquemáticos y posteriormente proceder con simulaciones por bloque (diseño experimental, enfoque cuantitativo y nivel descriptivo).
  El autor concluye que se ha logrado simular el diseño del sistema de transmisión, recepción y de interfaz entre el FPGA y el procesador en el software VIVADO. Finalmente, el autor recomienda realizar el test bench

lógico global del diseño de radar ionosonda y testear el sistema transmitiendo mediante una antena e incorporar una señal de GPS para el sincronismo del mismo.

Del trabajo de investigación desarrollado por el autor, se concluye la importancia de la recolección de los requisitos funcionales del radar ionosonda y posteriormente analizar y diseñar el sistema radar mediante descripción por bloques, además de utilizar los Ip Core brindados en el software VIVADO. Se resalta que dentro de los antecedentes nacionales solo existe dicha investigación.

## 2.2. Bases teóricas

### 2.2.1. Ionósfera

La ionosfera es la parte superior de atmósfera terrestre que se extiende aproximadamente desde los 50 a 1000 km, está conformada por moléculas disociadas y partículas cargadas eléctricamente, por efectos de la radiación ultravioleta y los rayos X del sol, las moléculas suspendidas en esta parte superior de la atmósfera, son calentadas y agitadas a tal punto que sufren diferentes procesos de disociación que involucran desprendimientos de electrones [2].

Las variaciones del comportamiento en la ionósfera, están directamente relacionadas con la radiación emitida desde el sol, el movimiento de la Tierra y los cambios en la actividad solar.

Esta parte de la atmósfera comprende capas denominadas D, E, F1, F2. Las capas D, E y F1 dependen de los ángulos de incidencia del sol y tienen un mayor tamaño de la densidad de ionización en verano que en invierno,

en cambio la capa F2 tiene un mayor tamaño de ionización en invierno que en verano.

Las múltiples aplicaciones de radiocomunicaciones que utilizamos actualmente dependen del comportamiento del plasma ionosférico. El término plasma se refiere a gases parcial o totalmente ionizados [18]. Debido a los diversos factores que afectan la ionosfera es de suma importancia desarrollar y poner en operación instrumentos científicos que nos ayuden a estudiar el clima espacial y modelar el comportamiento de la ionosfera.

### 2.2.2. Radar ionosonda

Una ionosonda es un radar HF que transmite pulsos electromagnéticos en un barrido de frecuencias desde un determinado punto de la Tierra hacia la ionosfera y las señales reflejadas son recepcionadas en el mismo u otro punto de la Tierra. Los pulsos enviados a la ionosfera logran tener la máxima intensidad de reflexión cuando la frecuencia de propagación de la onda de radio es igual a la frecuencia del plasma ionosférico.

Al procesar el eco las ionosondas realizan un registro de la reflexión llamado la altura virtual vs la frecuencia que fue enviada la señal, a esta construcción de registros se les conoce como perfil ionosférico o ionogramas y en ellos se puede visualizar el comportamiento de la ionosfera.

El Centro de Investigaciones Atmosféricas de la Universidad de Massachusetts Lowell (UMLCAR) ha producido una versión de baja potencia de sus sondas Digisonde TM, la sonda portátil Digisonde TM (DPS) capaz de realizar mediciones de la ionosfera superior y proporcionar procesamiento en tiempo real y análisis para caracterizar la propagación de la señal de radio para apoyar las comunicaciones o las operaciones de vigilancia [7].


Fig. 1. Sistema digisonda DPS-4 [8].

Fig. 2. Antena transmisora delta cruzada simple del sistema DPS-4 [16].

Fig. 3. Una de las 4 antenas receptoras del sistema DPS-4 [16].

### 2.2.3. Rango, resolución y ecuación del radar

#### a. Altura virtual (hv)

Las ionosondas tienen como principal objetivo medir la altura a la cual se refleja una frecuencia específica en alguna capa de la ionosfera. Para calcular la distancia a la cual se refleja la señal transmitida o también llamada altura virtual (hv), se emplea la siguiente ecuación 2.1 [9].

$$h_v = \frac{C.\Delta t}{2} \qquad (2.1)$$

Donde:

$h_v$ : representa una altura que no es la real sino la altura a la cual se reflejaría la señal si viajara a la velocidad de la luz.

$C$ : Velocidad de la luz.

$\Delta t$ : Tiempo de demora del eco reflejado.

**b. Altura máxima**

Los parámetros de los cuales depende directamente la altura máxima son la capacidad de detección del sistema, frecuencia de repeticiones del pulso (frp) del radar ionosonda o su inversa qué es el tiempo máximo de retardo del pulso (trp).

Cabe mencionar que se requiere esperar un tiempo suficiente después del pulso emitido para que sea detectado antes de que se transmita el siguiente pulso por el radar ionosonda. Por consiguiente, la frecuencia a la cual los pulsos pueden ser transmitidos está determinada por la mayor altura que esta puede transmitir.

La altura máxima se representa en la siguiente ecuación 2.2:

$$H = \frac{C}{2.f.r.p} \tag{2.2}$$

Donde:

$H$ : Altura máxima, se sustituye por la altura virtual $h_v$ en la ecuación 2.1.

frp: frecuencia de repetición de pulsos, se sustituye por el inverso del tiempo de demora del eco reflejado $\Delta t$ en la ecuación 2.1.

**c. Resolución**

La resolución es la distancia mínima en la que la ionosonda puede detectar entre dos blancos diferentes, en este caso específico de estudio de aeronomía puede detectar diferentes capas ionosféricas.
La resolución se representa en la siguiente ecuación 2.3:

$$\Delta H = \frac{C.T}{2} = \frac{C}{2.B} \tag{2.3}$$

$\Delta H$ : Resolución, se sustituye por la altura virtual *hv* en la ecuación 2.1.
T: Representa la longitud del pulso, se sustituye por el inverso del tiempo de demora del eco reflejado $\Delta t$ en la ecuación 2.1.
B: El ancho de banda del radar y es igual al inverso de la longitud del pulso B=1/T.

**d. Ecuación del radar**

Según los principios de diseño de sistemas de radar, la ecuación del radar es la estimación del rango en función de características deseadas y se representa en la ecuación 2.4 [12].

$$P_r = \frac{P_t \cdot G_t}{4\pi \cdot R^2} \cdot \frac{\sigma}{4\pi \cdot R^2} \cdot A_e \tag{2.4}$$

Donde:
Pr: Potencia recibida.
Pt: Potencia irradiada.
Gt: Ganancia de la antena.
σ: Superficie efectiva de dispersión.

Ae: Área efectiva de la antena receptora.
R: Distancia

Teniendo en cuenta las consideraciones de la ecuación del radar y si se define el rango máximo del radar R máx., cuando la señal recibida es igual a la mínima señal detectable se obtiene la ecuación 2.5 [12].

$$R_{máx}^4 = \frac{P_t \cdot G_t \cdot A_e \cdot \sigma}{(4\pi)^2 \cdot S_{mín}} \tag{2.5}$$

Se concluye que para obtener una alta resolución se requiere de pulsos de corta duración y para detectar blancos distantes es necesario una gran potencia pico de pulso Pt además de una larga duración de pulso t, o una pequeña frp.

### 2.2.4. Ionograma

Los ionogramas son mediciones de trazos correspondientes al barrido de señales de alta frecuencia reflejados, los cuales son generados por una ionosonda. Debido a que los pulsos de radio viajan más lento dentro de la ionósfera que en espacio libre, la altura aparente o “virtual” es medida en vez de la altura real.

Para frecuencias cercanas al máximo nivel de frecuencia en la capa correspondiente, la altura virtual tiende al infinito debido a que el pulso debe viajar una distancia finita a una velocidad cero. Las frecuencias en las que se produce dicho fenómeno se denominan frecuencias críticas [1].

Con los ionogramas se pueden obtener los siguientes parámetros de las señales reflejadas: frecuencia, alcance (o altura para mediciones de incidencia vertical), amplitud, fase, desplazamiento y propagación Doppler, ángulo de llegada y polarización de onda.

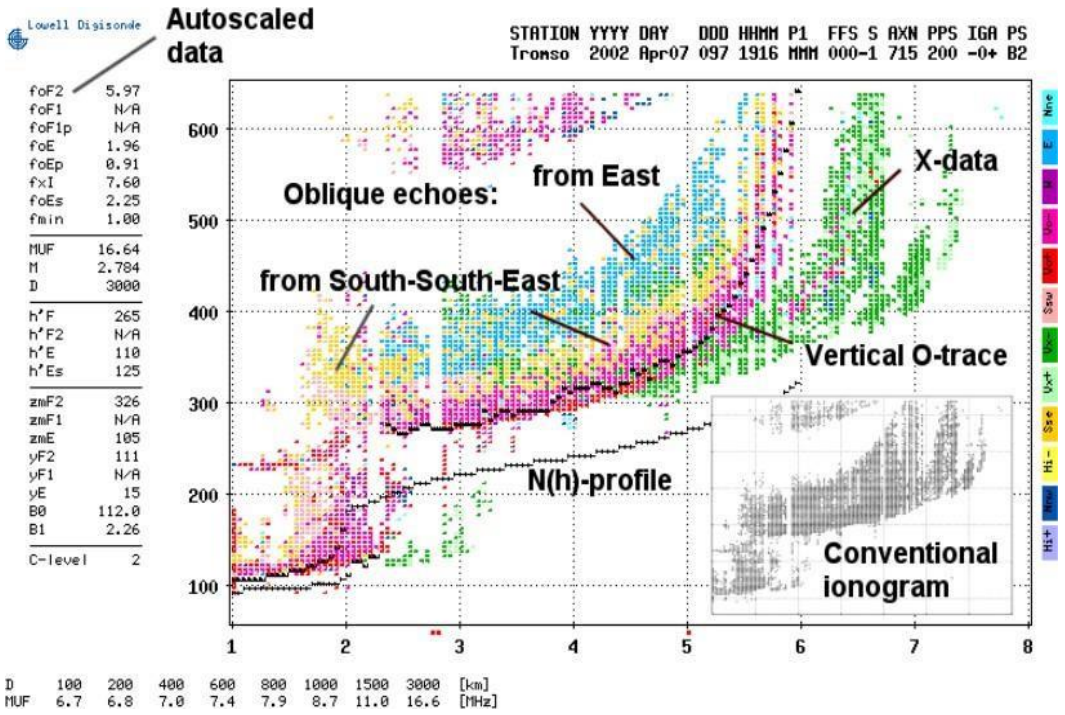


Fig. 4. Ionograma escalado automáticamente con software ARTIST4 [8].

### 2.2.5. Matrices de puertas programables por campo FPGA

Una FPGA es un tipo de dispositivo electrónico formado por bloques funcionales unidos a través de un array de conexiones programables, permitiéndonos describir un circuito digital a partir de un lenguaje descriptivo que puede ser el VHDL o Verilog. Internamente se componen principalmente de puertas lógicas, biestables, memorias, multiplexores, puertos de entrada y salida, entre otros.

En comparación con un procesador, este tiene una estructura fija y se modifica su comportamiento a través del programa escrito, ejecutado de forma secuencial. En cambio, en una FPGA se varía la estructura interna, constituyendo un circuito electrónico, lo que permite que tanto la eficiencia como la velocidad debido a que la FPGA puede trabajar en paralelismo.

### 2.2.6. Red Pitaya SIGNAL lab 250-12

SIGNAL lab 250-12 es el producto de Red Pitaya más sofisticado, creado para aplicaciones industriales e investigación más exigentes, donde la robustez y el rendimiento al sintetizar hardware dentro de estos dispositivos. Esta tarjeta de evaluación está equipada con PoE (alimentación a través de Ethernet), lo que brinda la posibilidad de alimentar y comunicarse con SIGNAL lab utilizando un solo cable Ethernet, internamente comprende de una FPGA Xilinx Zynq 7020, un ARM Cortex-A9 C de doble núcleo, Frecuencia de muestreo 250MSPS, resolución de DAC de 14 bits y resolución de ADC de 12 bits.

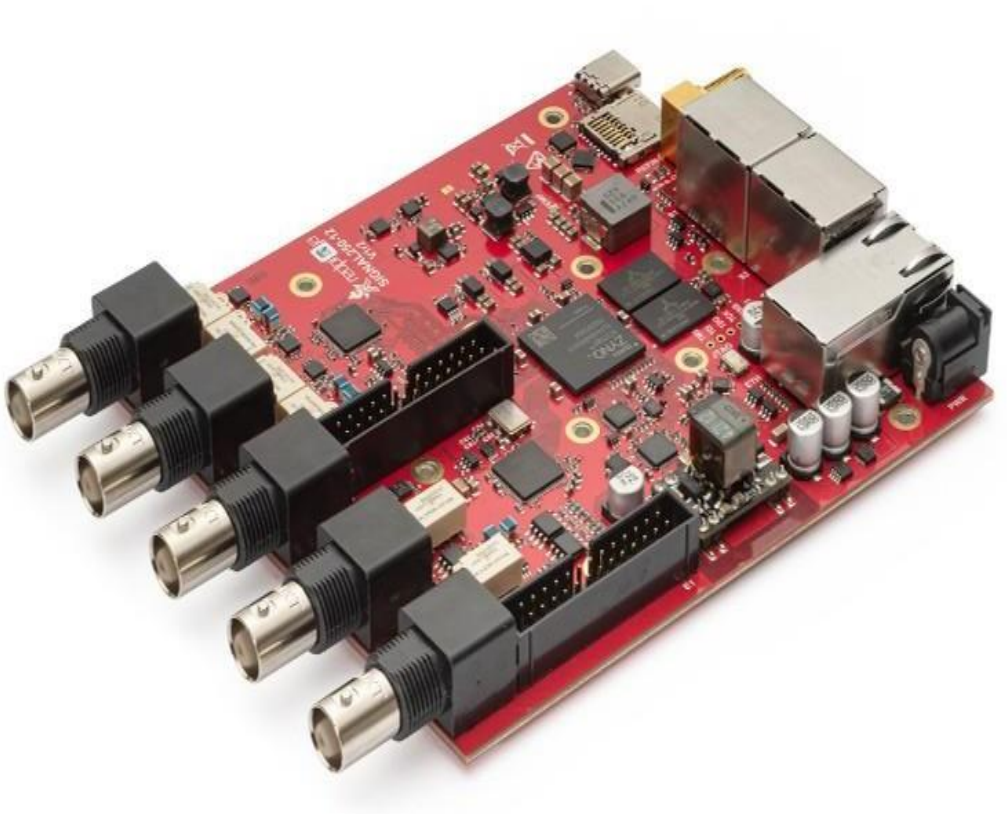

Fig. 5. Tarjeta de evaluación Red Pitaya SIGNAL lab 250-12 [10].

### 2.2.7. Lenguaje de descripción de hardware (VHDL)

Es un lenguaje de descripción y modelado diseñado para describir la funcionalidad y la organización de sistemas hardware digitales. El VHDL es un lenguaje con una sintaxis amplia y a la vez flexible, lo que permite realizar el modelado de una forma estructural y a la vez el flujo de datos en el hardware. Las ventajas que ofrece el uso del VHDL son:

- Permite el diseño, modelado y comprobación del sistema, sintetizando hardware en bajo nivel.
- Permite dividir el diseño del hardware en unidades más pequeñas para su descripción.
- Los circuitos que se describen siguen una estructura de síntesis.
- Basado en un estándar IEEE STD 1076-1987, para que así pueda ser usado por toda la comunidad de ingenieros.

### 2.2.8. Niveles de abstracción

La abstracción es un modelo simplificado del diseño de hardware, una abstracción de alto nivel contiene sólo los datos más vitales. Por otro lado, una abstracción de bajo nivel es más detallada.

Los niveles de abstracción se emplean para clasificar modelos HDL según el grado de detalle y precisión de sus descripciones [11]. Los niveles de abstracción descritos desde el punto de vista de simulación y síntesis del circuito pueden definirse como sigue:

- **Funcional o algorítmico:** Se refiere a la relación funcional entre las entradas y salidas del circuito o sistema, sin hacer referencia a la realización final (implementación).
- **Nivel de Transferencia de registros (RTL):** Consiste en la partición del sistema en bloques funcionales sin considerar a detalle la realización final de cada bloque

- ❖ **Lógico:** El circuito se expresa en términos de ecuaciones lógicas o de compuertas.

### 2.2.9. Estilos de descripción de hardware

El lenguaje VHDL presenta tres estilos de descripción que dependen del nivel de abstracción.

a. **Descripción comportamental:** Describe el comportamiento de las salidas del circuito con respecto a las entradas, en este estilo de descripción de hardware no se proporciona información de cómo estará conformado el circuito.

```
1  ----------------------------------------------------------------------
2  -- Module Name: mux 4to1 sequential statement (if-elsif-else)
3  ----------------------------------------------------------------------
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity mux_4to1_iee is
8    Port (
9              a,b,c,d  :  in  std_logic;
10             s        :  in  std_logic_vector(1 downto 0);
11             f        :  out std_logic
12   );
13 end mux_4to1_iee;
14
15 architecture Behavioral of mux_4to1_iee is
16 begin
17             mux: process(a,b,c,d,s)
18             begin
19                     if (s="00") then
20                         f <= a;
21                     elsif (s="01") then
22                         f <= b;
23                     elsif (s="10") then
24                         f <= c;
25                     else
26                         f <= d;
27                     end if;
28             end process mux;
29 end Behavioral;
```

Fig. 6. Descripción comportamental de un multiplexor.

Fuente: Elaboración propia.

b. **Descripción flujo de datos:** Especifica el circuito mediante un conjunto de expresiones booleanas concurrentes, en este estilo se describe el flujo de la información.

```
1  ----------------------------------------------------------------------
2  -- Project Name: mux 2to1 - simple signal assignment statement
3  ----------------------------------------------------------------------
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity mux_2to1_ssas is
8    Port (
9             a : in  std_logic;
10            b : in  std_logic;
11            s : in  std_logic;
12            f : out std_logic
13   );
14 end mux_2to1_ssas;
15
16 architecture Behavioral of mux_2to1_ssas is
17 begin
18            f <= ((not s) and a) or (s and b);
19 end Behavioral;
```

Fig. 7. Descripción del flujo de datos de un multiplexor.

Fuente: Elaboración propia.

c. **Descripción Estructural:** Especifica el circuito describiendo distintos módulos y las conexiones entre estos.

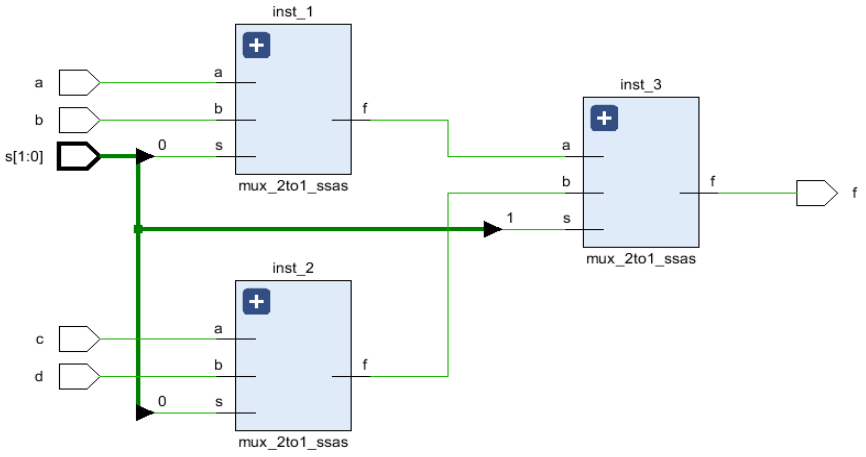


Fig. 8. Descripción estructural de un multiplexor.

Fuente: Elaboración propia.

## 2.3. Marco conceptual

### 2.3.1. Controlador de radar

Para el diseño del controlador de radar ionosonda en primer lugar se comprenderá el protocolo de comunicación SPI dado que será el medio de interfaz de comunicación entre el embebido y el FPGA.

Mediante la descripción de hardware se define la interfaz SPI en modo A, el cual hace la lectura del data frame de n bits, para modo práctico de este ejemplo usamos 16 bits cuando la polaridad del CLK inicia en bajo (CPOL=0) y luego se detecta la fase del CLK en flanco de subida (CPHA=0).

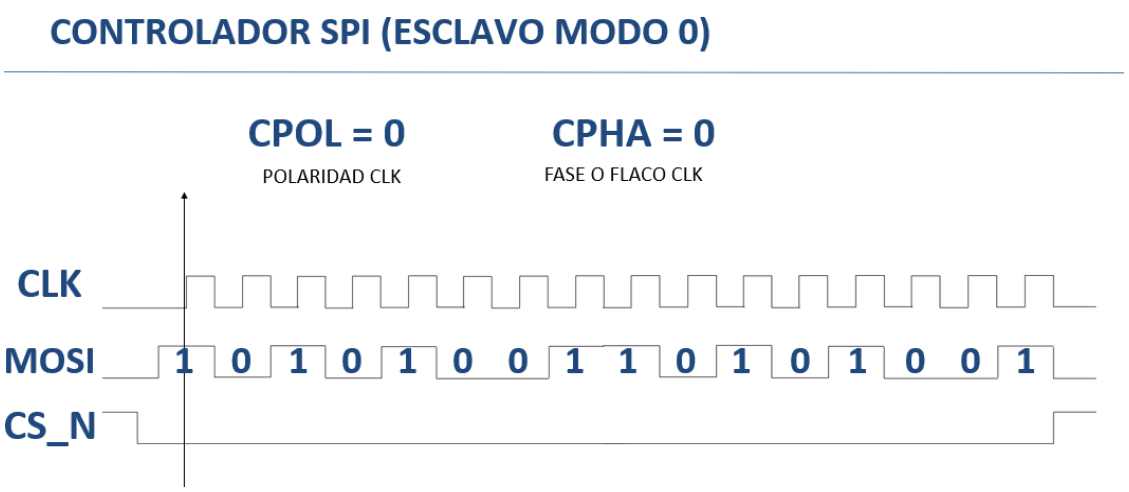


Fig. 9. Protocolo de comunicación SPI.

Fuente: Elaboración propia.

Posteriormente el data frame de 16 bits se divide en tres salidas llamadas W/R(1 bit), ADDRESS(6 bits), DATA(7 bits) y por medio de la señal W/R se almacenan los datos en el bloque mapa de registros.

Estos registros contendrán la configuración del barrido de frecuencias de 1 a 25 MHz, ancho de pulso, Ipp, código, repeticiones. A continuación, se muestra la tabla 1 donde se aprecia cada uno de los registros y sus respectivas longitudes de palabras, esta tabla la tenemos como referencia para el diseño del controlador de radar.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | CONTROL | - | - | - | - | - | - | R/W START 0 | R/W RST 1 | $00 |
| 1 | FREQ_1 | R/W FREQ7 0 | R/W FREQ6 0 | R/W FREQ5 0 | R/W FREQ4 0 | R/W FREQ3 0 | R/W FREQ2 0 | R/W FREQ1 0 | R/W FREQ0 0 | $01 |
| | FREQ_2 | - | - | R/W FREQ13 0 | R/W FREQ12 0 | R/W FREQ11 0 | R/W FREQ10 0 | R/W FREQ9 0 | R/W FREQ8 0 | $02 |
| | PULSE WIDTH_1 | R/W PULSE-W7 0 | R/W PULSE-W6 0 | R/W PULSE-W5 0 | R/W PULSE-W4 0 | R/W PULSE-W3 0 | R/W PULSE-W2 0 | R/W PULSE-W1 0 | R/W PULSE-W0 0 | $03 |
| | PULSE WIDTH_2 | R/W PULSE-W15 0 | R/W PULSE-W14 0 | R/W PULSE-W13 0 | R/W PULSE-W12 0 | R/W PULSE-W11 0 | R/W PULSE-W10 0 | R/W PULSE-W9 0 | R/W PULSE-W8 0 | $04 |
| | IPP_1 | R/W IPP7 0 | R/W IPP6 0 | R/W IPP5 0 | R/W IPP4 0 | R/W IPP3 0 | R/W IPP2 0 | R/W IPP1 0 | R/W IPP0 0 | $05 |
| | IPP_2 | R/W IPP15 0 | R/W IPP14 0 | R/W IPP13 0 | R/W IPP12 0 | R/W IPP11 0 | R/W IPP10 0 | R/W IPP9 0 | R/W IPP8 0 | $06 |
| | IPP_3 | R/W IPP23 0 | R/W IPP22 0 | R/W IPP21 0 | R/W IPP20 0 | R/W IPP19 0 | R/W IPP18 0 | R/W IPP17 0 | R/W IPP16 0 | $07 |
| | CODE _1 | R/W CODE7 0 | R/W CODE6 0 | R/W CODE5 0 | R/W CODE4 0 | R/W CODE3 0 | R/W CODE2 0 | R/W CODE1 0 | R/W CODE0 0 | $08 |
| | ○ | | | | | | | | | |
| | ○ | | | | | | | | | |
| | ○ | | | | | | | | | |
| | CODE _16 | - | - | - | R/W CODE124 0 | R/W CODE123 0 | R/W CODE122 0 | R/W CODE121 0 | R/W CODE120 0 | $23 |
| | REPEAT | R/W REPEAT7 0 | R/W REPEAT6 0 | R/W REPEAT5 0 | R/W REPEAT4 0 | R/W REPEAT3 0 | R/W REPEAT2 0 | R/W REPEAT1 0 | R/W REPEAT0 0 | $24 |

Tabla 1. Mapa de registros del generador de señales de radiofrecuencia.

Fuente: Elaboración propia.

### 2.3.2. Oscilador controlado numéricamente (NCO)

El NCO para generar una onda sinusoidal requiere de la palabra de sintonización de n bits o también llamado incremento de fase de n bits, dicha señal de entrada nos permite determinar la frecuencia de la salida de onda sinusoidal. El incremento de fase se expresa en un formato llamado medición de ángulo binario (BAM), en el que el bit más significativo (MSB) de la palabra representa 180º, el siguiente bit representa 90º, y así sucesivamente.

El NCO comprende un acumulador de fase, un convertidor de fase a amplitud y registros. En el acumulador de fase, se ingresa la señal de incremento de fase a la salida de la suma acumulada, implementada como un sumador seguido de un registro (REG), esto produce una fase de aumento uniforme, incrementada a la velocidad del reloj del sistema. Luego la suma acumulada (MSBs) se envía a un convertidor de fase a amplitud, que es una tabla de búsqueda que produce un valor de k bit también seguido de un registro que representa la amplitud de la onda sinusoidal en la fase de entrada [12].

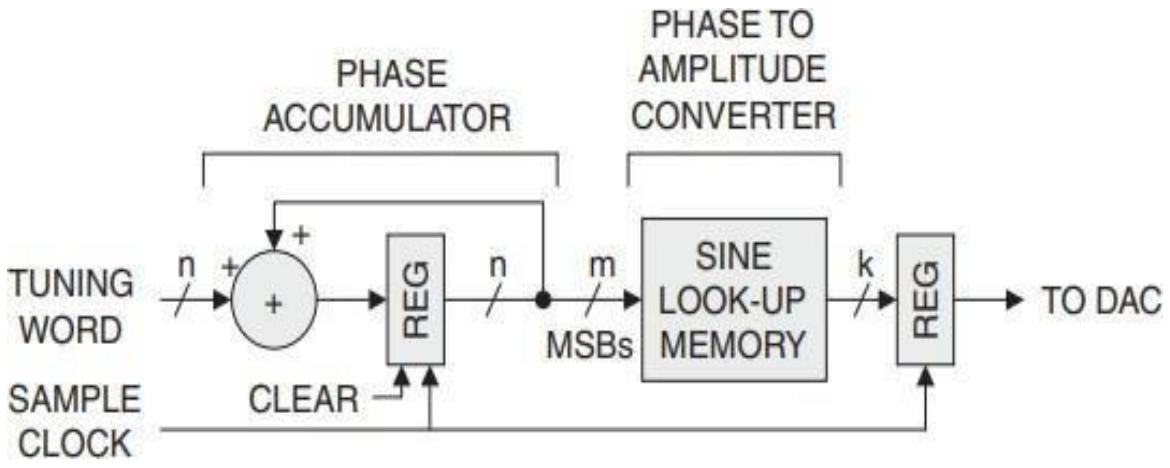


Fig. 10. Diagrama de bloque del NCO [12].

Teniendo los valores de los parámetros de frecuencia interna del FPGA, frecuencia de la onda senoidal, incremento de fase y ancho de fase, entonces la frecuencia de la onda senoidal se puede representar como:

$f_{clk}$ = Frecuencia interna del FPGA
$f_{out}$ = Frecuencia de la onda senoidal
$\Delta\theta$ = Incremento de fase
$2^{B_{\theta(n)}}$ = Ancho de fase

$$f_{out} = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}} \quad (2.6)$$

Para generar una onda senoidal con una frecuencia de 1 MHz, el incremento de fase requerido para generar dicha onda es el siguiente:

$$\Delta\theta = \frac{f_{out} x 2^{B_{\theta(n)}}}{f_{clk}}$$

$$\Delta\theta = \frac{1_{Mhz} x 2^{14}}{250_{Mhz}}$$

$$\Delta\theta = 65.036$$

Al incremento de fase se debe aumentar un valor de 0.5 y luego truncar a un número entero e introducirlo en la ecuación (4) dando la siguiente frecuencia de salida, que puede ser validada por los instrumentos que se cuenta en el laboratorio de Investigación y Desarrollo e Innovación (IDI) ubicado en el Radio Observatorio de Jicamarca.

$$f_{out} = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}}$$

$$f_{out} = \frac{250_{Mhz} x 66}{2^{14}}$$

$$f_{out} = 1.00708_{Mhz}$$

### 2.3.3. Modulación BPSK

La modulación BPSK es una modulación por desplazamiento de fase de dos estados, donde la fase de la portadora se establece en 0 o π según el valor de la señal moduladora. La señal moduladora m(t) se define como una secuencia binaria que se multiplica con una portadora sinusoidal c(t), obteniendo así la

señal modulada BPSK. Cuando la señal de salida del modulador m(t) se encuentra en un estado '1' la señal modulada se ve exactamente como la portadora con 0º de fase inicial, mientras que, si se encuentra en un estado "0", la señal modulada tiene una portadora con fase inicial de π, la modulación BPSK se representa en la ecuación 2.7.

$$s_i(t) = \begin{cases} s_1(t) = -A\cos(2\pi f_c t), & if\ 0_T \\ s_2(t) = +A\cos(2\pi f_c t), & if\ 1_T \end{cases} \tag{2.7}$$

En la modulación BPSK, la señal transmitida es s(t) = m(t)c(t), donde 0 < t < Tb

Cabe mencionar que T es el tiempo de duración del pulso codificado, t es la duración de los sub pulsos y la fase de cada sub pulso es modificable para que varíe entre 0 y π. En esta modulación una de las características fundamentales es que se puede obtener su relación de compresión con la ecuación 2.8.

$$N = T / t \tag{2.8}$$

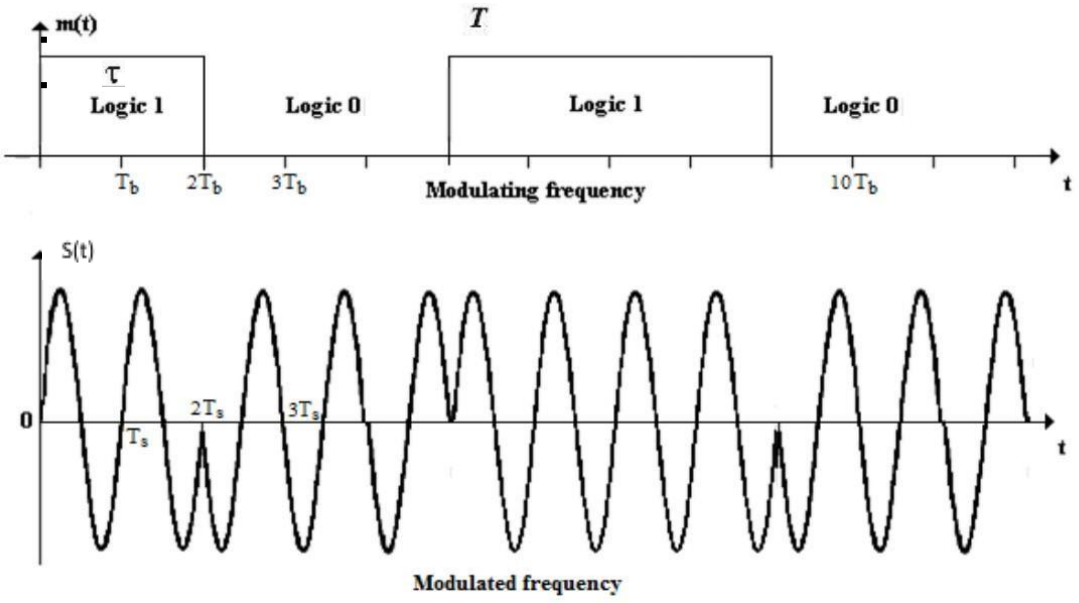


Fig. 11. Modulación por código de desplazamiento de fase (BPSK) [14].

Para el diseño del generador de señales de radiofrecuencia que operará al transmisor de radar ionosonda se requiere de un modulador de BPSK digital el cual se implementará en descripción de hardware VHDL en el software Vivado.

### 2.3.4. Modulación OOK

La modulación de desplazamiento de amplitud binario (On Off Keying) es un caso particular de la modulación ASK. Este tipo de modulación tiene un comportamiento de conmutación ya que cuando se ingresa una señal binaria unipolar f(t), la cual, si la señal del mensaje es "1", entonces la salida será una señal sinusoidal, si el mensaje la señal es "0", entonces la salida será 0. Su función se representa en la siguiente ecuación 2.9.

$$f_{ASK}(t) = f(t)cos(\omega_c t) \tag{2.9}$$

- si $f(t) = 0V \Rightarrow f_{ASK}(t) = 0$
- si $f(t) = AV \Rightarrow f_{ASK}(t) = Acos(\omega_c t)$

A continuación se muestran los gráficos de la modulación OOK.

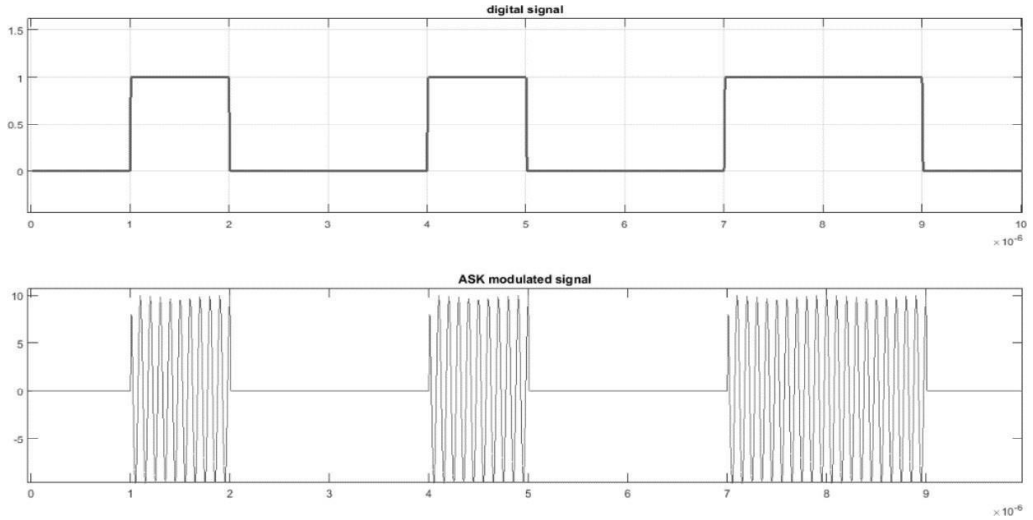


Fig. 12. Modulación de desplazamiento de amplitud binario (OOK) [13].

La modulación OOK será utilizada para la generación del barrido de frecuencias de 1 a 25 MHz que estarán sincronizados con la señal de GPS de 10Mhz e iniciará el envío de las señales de radiofrecuencia cuando la entrada trigger detecta un flanco de subida (*rising edge*).

### 2.4. Definición de términos básicos

- ROJ: Radio Observatorio de Jicamarca.
- IGP: Instituto Geofísico del Perú.
- UMLCAR: Centro de Investigaciones Atmosféricas de la Universidad de Massachusetts Lowell.
- GNSS: Sistemas de posicionamiento global basados en satélites.
- USRP: Universal Software Radio Peripheral.
- FPGA SoC: Arreglos de puertas programables en campo de sistema en chip.
- Zynq SoC: Son dispositivos de la familia Xilinx que integran un FPGA y un procesador.
- VHDL: Es un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción.
- Ip Cores: Son diseños de hardware de uso libre.
- RTL: Nivel de Transferencia de registro.
- NCO: Oscilador controlado numéricamente.
- BPSK: Modulación de desplazamiento de fase binario.
- OOK: Modulación de desplazamiento de amplitud binario.

- Ionogramas: Son diagramas generados por ionosondas, que nos permiten medir las frecuencias críticas de cada capa y las alturas virtuales.
- Altura virtual (HV): Estas alturas son mayores que la altura real y son generadas por los ionogramas.
- FoF2: Frecuencia crítica de la capa F2.
- IPP: Período entre pulsos emitidos por el generador de señales.
- HF: Alta frecuencia.
- SNR: Relación señal / ruido.

# III. HIPOTESIS Y VARIABLES

## 3.1. Hipótesis general e hipótesis específicas

### 3.1.1. Hipótesis general

- Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se operará un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

### 3.1.2. Hipótesis específicas

- **Hipótesis específica 1:** Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se podrá sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

- **Hipótesis específica 2:** Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se podrá emitir un barrido de frecuencia en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.

- **Hipótesis específica 3:** El diseño e implementación del circuito modulador de señales de radiofrecuencias basado en FPGA SoC permitirá realizar modulación BPSK y OOK.

## 3.2. Definición conceptual de variables

### 3.2.1. Variable independiente

**Diseño e implementación de un generador de señales de radiofrecuencia basado en FPGA SoC:** Un generador de señales es aquel dispositivo electrónico que genera patrones de señales periódicas o no periódicas y en este trabajo de investigación se generan de manera digital.

**a. Dimensión 1:**

**Sintetización de hardware:** La sintetización de hardware es el proceso de generación de una representación en puertas lógicas de una descripción hardware (VHDL o Verilog).

**Indicador 1:** Simulaciones con el Test bench.
**Indicador 2:** Pruebas en el laboratorio.

### 3.2.2. Variable dependiente

**Operación de transmisor de radar ionosonda en el Radio Observatorio de Jicamarca:** Los transmisores de ionosonda son radares HF que transmiten pulsos electromagnéticos desde un determinado punto de la Tierra hacia la ionosfera.

**a. Dimensión 1:**

**Sincronismo:** En telecomunicaciones el sincronismo es muy utilizado para que diversos equipos de un mismo sistema puedan compartir la misma señal de reloj y mandar información de forma que todos los equipos estén sincronizados.

**Indicador 1:** Desfase entre la señal generada y la señal de GPS.

**a. Dimensión 2**

**Análisis espectral de las frecuencias:** El análisis espectral de frecuencias mide y traza la potencia de la señal en un rango de frecuencias determinado para ello se utiliza el instrumento analizador de espectro.

**Indicador 1:** Verificación del barrido de frecuencias con el analizador de espectro.

## 3.3. Operacionalización de variables

### 3.3.1. Variable independiente

**Diseño e implementación de un generador de señales de radiofrecuencia basado en FPGA SoC:** El diseño del generador de señales se implementa en la plataforma de desarrollo Red Pitaya que comprende de un FPGA SoC, con el cual se pudo modificar la frecuencia del pulso, ancho del pulso, período entre pulsos, repeticiones de pulsos.

**a. Dimensión 1**

**Sintetización de hardware:** Se sintetiza cada uno de los bloques que comprende el generador de señales desarrollado en el software Vivado 2019.1.

**Indicador 1:** Simulaciones con el Test bench.
**Indicador 1:** Pruebas en el laboratorio.

### 3.3.2. Variable dependiente

**Operación de transmisor de radar ionosonda en el Radio Observatorio de Jicamarca:** Los transmisores de radar ionosonda se sincronizan con la señal

de GPS y esperan una señal de Trigger para empezar a generar un barrido de frecuencias que son enviadas a las capas altas de la atmósfera.

**a. Dimensión 1**

**Sincronismo:** Se sincronizó las señales generadas con la señal de GPS.

**Indicador 1:** Desfase entre la señal generada y la señal de GPS.

**a. Dimensión 2**

**Análisis espectral de las frecuencias:** El análisis espectral de frecuencias nos permite verificar la correcta generación de frecuencias de las señales.

**Indicador 1:** Verificación del barrido de frecuencias con el analizador de espectro.

## Matriz de operacionalización

| Variable | Definición conceptual | Definición operacional | Dimensión | Definición conceptual | Definición operacional | Indicadores | Escalas y valores |
|---|---|---|---|---|---|---|---|
| **Variable Independiente**<br><br>Diseño e implementación de un generador de señales de radiofrecuencia basado en FPGA SoC. | Un generador de señales es aquel dispositivo electrónico que genera patrones de señales periódicas o no periódicas y en este trabajo de investigación se generan de manera digital. | El diseño del generador de señales se implementará en la plataforma de desarrollo Red Pitaya que comprende de un FPGA SoC, en el cual se podrá modificar la frecuencia del pulso, ancho del pulso, período entre pulsos, repeticiones de pulsos. | Sintetización de hardware | La sintetización de hardware es el proceso de generación de una representación en puertas lógicas de una descripción hardware (VHDL o Verilog). | Se sintetiza cada uno de los bloques que comprende el generador de señales desarrollado en el software Vivado 2019.1. | I1:Simulaciones con el Test bench.<br><br>I2: Pruebas en el laboratorio. | (0-1) |
| **Variable Dependiente**<br><br>Operación de transmisor de radar ionosonda en el Radio Observatorio de Jicamarca. | Los transmisores de ionosonda son radares HF que transmiten pulsos electromagnéticos desde un determinado punto de la Tierra hacia la ionosfera. | Los transmisores de ionosonda se sincronizan con la señal de GPS y generan un barrido de frecuencias que son enviadas a las capas altas de la atmósfera. | Sincronismo | El sincronismo es muy utilizado para que diversos equipos de un mismo sistema puedan compartir la misma señal de reloj y mandar información de forma que todos los equipos estén sincronizados. | Se sincronizan las señales generadas con la señal de GPS. | I1: Desfase entre la señal generada y la señal de GPS. | (0-500ps) |
| | | | Análisis espectral de las frecuencias | El análisis espectral de frecuencias mide y traza la potencia de la señal en un rango de frecuencias determinado para ello se utiliza el instrumento analizador de espectro. | El análisis espectral de frecuencias nos permite verificar la correcta generación de frecuencias de las señales. | I2: Verificación del barrido de frecuencias con el analizador de espectro. | (1-25Mhz) |

Tabla 2. Matriz de operacionalización.

Fuente: Elaboración propia.

# IV. DISEÑO METODOLÓGICO

## 4.1. Tipo y diseño de investigación

### 4.1.1. Tipo de investigación

El presente trabajo de investigación es del tipo aplicado, ya que se aplica el diseño e implementación del generador de señales para operar un transmisor de radar ionosonda en periodos de tiempo establecidos según el cronograma de ejecución.

### 4.1.2. Diseño de investigación

El presente trabajo de investigación tiene un diseño experimental, pues los parámetros de operación del transmisor de radar ionosonda son estimulados por el diseño del generador de señales de radiofrecuencia basado en FPGA SoC.

### 4.1.3. Nivel de investigación

El presente trabajo de investigación tiene un nivel de investigación del tipo explicativo pues se explica cómo la sintetización de hardware del generador de señales logró operar el transmisor de radar ionosonda.

### 4.1.4. Enfoque de la investigación: cuantitativo

El presente trabajo de investigación tiene un enfoque de investigación del tipo cuantitativo pues los parámetros de operación del radar ionosonda son medidos y analizados mediante simulaciones con test bench e instrumentos de laboratorio en el Radio Observatorio de Jicamarca.

## 4.2. Método de investigación

De acuerdo al problema de investigación planteado, en primer lugar, se definen los requisitos funcionales que requiere el generador de señales de radiofrecuencia basado en FPGA SoC, en segundo lugar se define el hardware para el generador de señales de radiofrecuencias en este caso la plataforma de de desarrollo que integre una FPGA Soc (Red Piñata SIGNAL Lab 250-12 )y un embebido (Seeeduino Xiao SAMD21) para la configuración de registros. El lenguaje de descripción de hardware que se utiliza es el VHDL, en el entorno de desarrollo Vivado versión 2019.1, en tercer lugar se define el estilo de descripción de hardware, para este diseño se utiliza la descripción comportamental para los módulos controlador del generador de señales de radiofrecuencia, oscilador controlado numéricamente (NCO), modulador BPSK y OOK. Luego en cuarto lugar se agrega el Ip Core Clocking Wizard que nos permite elevar la frecuencia interna hasta 250 MHz, cabe mencionar que el sincronismo es muy importante para los radares ionosondas es por ello que el presente diseño de generador de señales de radiofrecuencia incluye un módulo de sincronismo el cual es proporcionado por el repositorio de la plataforma de desarrollo Red Pitaya. Además el generador de señales de radiofrecuencia posee una entrada para la señal de trigger que es un pulso enviado cada 5 minutos y con dicho pulso se inicia la generación del barrido de frecuencias. Como siguiente paso en quinto lugar se integran todos los módulos mencionados e Ip Cores utilizando el estilo de descripción estructural con la herramienta Create Design Block en el software Vivado. En el sexto paso se realizan las configuraciones en la Red Pitaya para poder habilitar la entrada de trigger y elevar el voltaje pico pico de la señal pulsada a transmitir. El séptimo paso es realizar el diseño de placa electrónica en modo shield que contenga al embebido, entradas y salidas (BNC-SMA) y el circuito convertidor de señal de clock senoidal a cuadrada esto debido a que Red Pitaya solo acepta señal cuadra de clock. Finalmente en el octavo paso se realizan las pruebas de simulación con el test bench (banco de pruebas) y también pruebas físicas con los equipos de laboratorio en el Radio Observatorio de Jicamarca.

### 4.2.1 Diagrama de la metodología

En la figura 13 se muestra el diagrama de la metodología que se desarrolla en cada uno de los pasos para el diseño e implementación de generador de señales de radiofrecuencia basado en FPGA SoC.

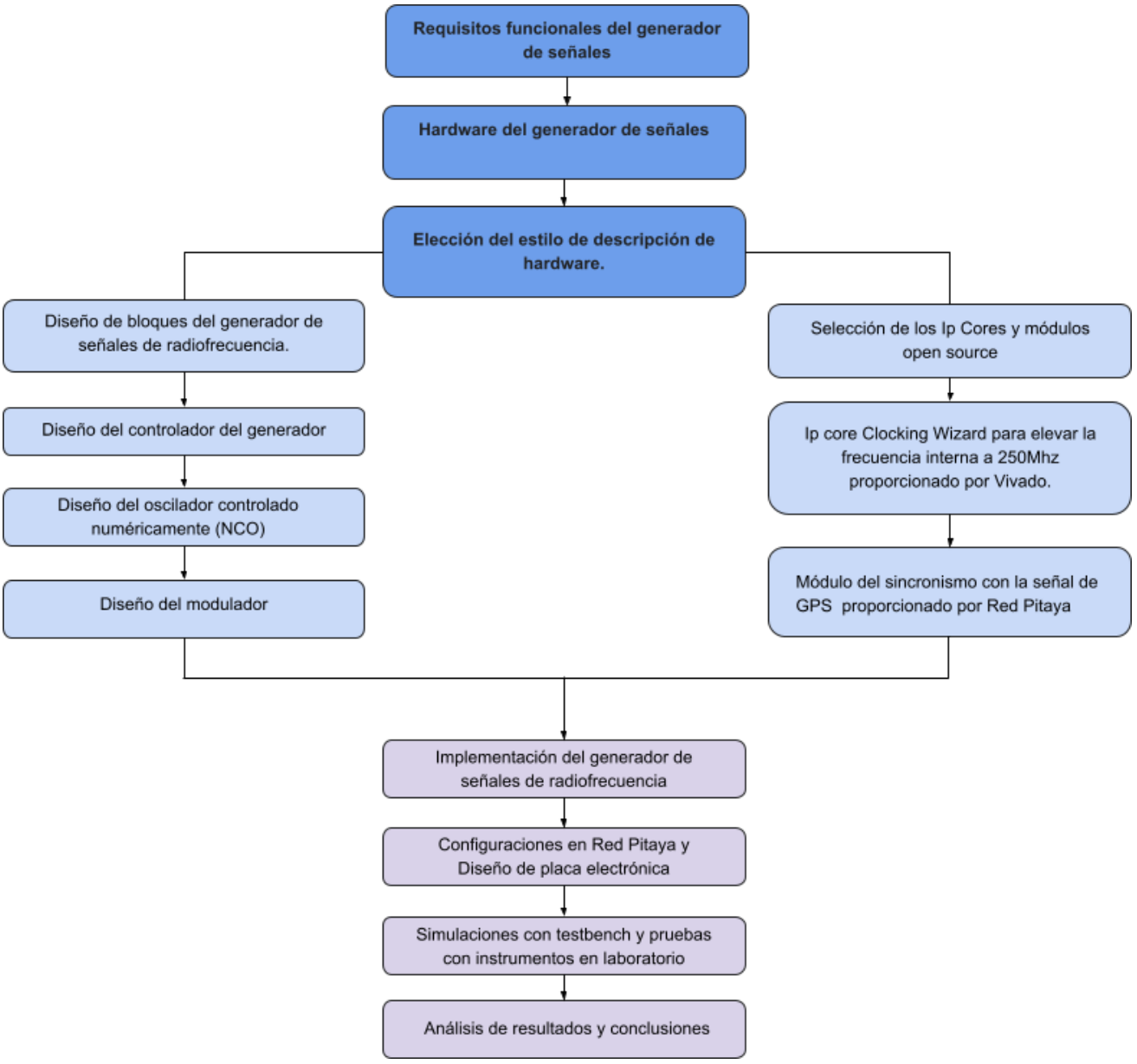


Fig. 13. Diagrama de la metodología.

Fuente: Elaboración propia

### 4.2.2. Requisitos funcionales del generador de señales

De acuerdo al problema de investigación planteado. En primer lugar, se definen los requisitos funcionales que requiere el generador de señales de radiofrecuencia basado en FPGA SoC.

Requisitos funcionales del generador de radiofrecuencia:

- Cantidad de pulsos = 1808 pulso.
- Ancho del pulso = 10 us - 60 us.
- Período entre pulsos emitidos (IPP) = 1 ms - 10 ms.
- Pasos entre frecuencias= 8 Khz.
- % de pasos entre frecuencia = 0.5 %.
- barrido de frecuencias = 1 - 25 Mhz.
- Repeticiones = Cada 4 envíos de frecuencias distintas y luego se repiten 4 veces dichas frecuencias.

Ejemplo de repeticiones de las frecuencias, donde:

F= frecuencias.

R= repeticiones.

Los valores de la frecuencia se encuentran en Mhz.

| | F1 | F2 | F3 | F4 |
|---|---|---|---|---|
| R1 | 1.601 | 1.609 | 1.617 | 1.625 |
| R2 | 1.601 | 1.609 | 1.617 | 1.625 |
| R3 | 1.601 | 1.609 | 1.617 | 1.625 |
| R4 | 1.601 | 1.609 | 1.617 | 1.625 |

Tabla 3. Ejemplo de repeticiones de frecuencias.

Fuente: Elaboración propia.

### 4.2.3. Diseño de bloques del generador de señales de radiofrecuencia.

El generador de señales de radiofrecuencia comprende en su diseño tres bloques principales; controlador spi, mapa de registros y generador de pulsos, el tercer bloque a su vez engloba a tres sub-bloques los cuales son el oscilador controlado numéricamente (NCO), modulador de señales y multiplexor. El diseño permiten configurar el ancho del pulso, ipp, acumulador de fase y repeticiones, para generar dos señales, la primera es la que se encarga de realizar el sweep de frecuencias de 1 mhz a 25 mhz y la otra un señal de gate que está activa sólo durante la ventana de cada pulso emitido.

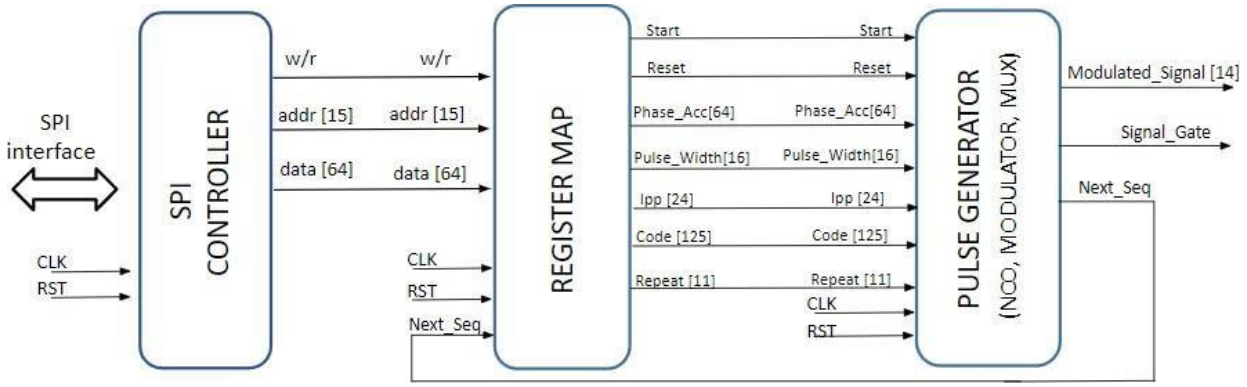


Fig. 14. Diseño de bloques del generador de radiofrecuencias.

Fuente: Elaboración propia.

### 4.2.4. Diseño del controlador SPI

El diseño del controlador de radar ionosonda comprende del protocolo de comunicación SPI para la comunicación entre el sistema embebido y el FPGA. Para ello se usa la descripción de hardware y se define la interfaz SPI en modo A, el cual hace la lectura del data frame de 80 bits cuando la polaridad del CLK inicia en bajo (CPOL=0) y luego se detecta la fase del CLK en flanco de subida

(CPHA=0). La cantidad de data frame que se transmite por la interfaz spi es de 80 bits, el bit más significativo (MSB) es de escritura y lectura (W/R), los siguientes 15 bits son de direcciones (ADDRESS) y los 64 bits menos significativo (LSB) son los datos de configuración (DATA) y por medio de la señal W/R se almacenan los datos en el bloque mapa de registros en este caso usamos el Ip Core Bloque de Memoria RAM nativa (BRAM).

En la siguiente tabla n°1 se muestra la distribución del mapa de registros, la primera dirección es de control, la segunda es la configuración del ancho del pulso e IPP, la tercera y cuarta son los valores del código para modulación,cabe mencionar que a partir del quinto registro hasta la posición 1812 son los valores de las frecuencias a generar.

| ADDRESS | NAME ADDRESS | WORD LENGTH | | | |
|---|---|---|---|---|---|
| $0000 | CONTROL | - | R/W | R/W | R/W |
| | | RESERVED | BAUD | REPEAT | RST-START |
| | | - | 0x0000 | 0x0000 | 0x0000 |
| $0001 | PULSE WIDTH - IPP | - | R/W | R/W | R/W |
| | | RESERVED | PULSE_W | IPP2 | IPP1 |
| | | - | 0x0000 | 0x0000 | 0x0000 |
| $0002 | CODE_1 | R/W | R/W | R/W | R/W |
| | | CODE4 | CODE3 | CODE2 | CODE1 |
| | | 0x0000 | 0x0000 | 0x0000 | 0x0000 |
| $0003 | CODE_2 | R/W | R/W | R/W | R/W |
| | | CODE8 | CODE7 | CODE6 | CODE5 |
| | | 0x0000 | 0x0000 | 0x0000 | 0x0000 |
| $0004 | FREQUENCY_1 | R/W | R/W | R/W | R/W |
| | | FREQ4 | FREQ3 | FREQ2 | FREQ1 |
| | | 0x0000 | 0x0000 | 0x0000 | 0x0000 |
| ○ ○ ○ | | | | | |
| $1812 | FREQUENCY_1808 | R/W | R/W | R/W | R/W |
| | | FREQ4 | FREQ3 | FREQ2 | FREQ1 |
| | | 0x0000 | 0x0000 | 0x0000 | 0x0000 |

Tabla 4. Tabla de mapa de registros para el controlador SPI.

Fuente: Elaboración propia.

### 4.2.5. Diseño de bloques de Numerically Controlled Oscillator *(NCO)*

El diseño del NCO comprende de dos bloques; acumulador de fase y convertidor de fase a amplitud, para generar una onda sinusoidal se requiere del incremento de fase de 64 bits, dicha señal de entrada nos permite determinar la frecuencia de la onda sinusoidal de salida.

En el acumulador de fase, se ingresa la señal de incremento de fase de 64 bits a la salida de la suma acumulada, implementada como un sumador seguido de un registro, esto produce una fase de aumento uniforme que incrementa con cada flanco de subida del clock del sistema.

La salida de la suma acumulada pasa por un truncamiento de 64 bits a 16 bits antes de ser enviada al bloque convertidor de fase a amplitud, que es la tabla de búsqueda que contiene un ¼ de la onda senoidal seguido de un registro que almacena la amplitud de la onda sinusoidal.

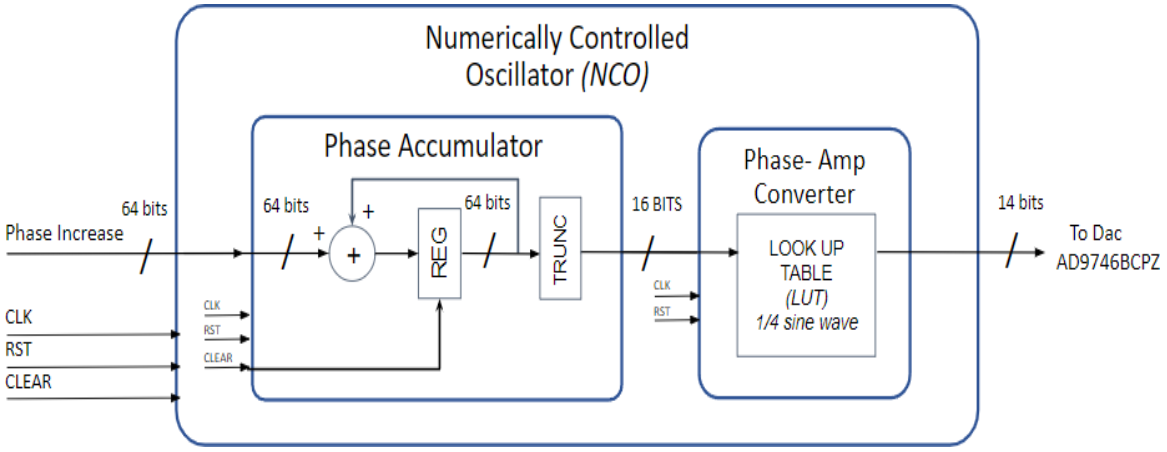


Fig. 15. Diseño de bloques de Numerically Controlled Oscillator (NCO).

Fuente: Elaboración propia.

### 4.2.6. Cálculos para el acumulador de fase

El cálculo de acumulador de fase de cada frecuencia requiere conocer los valores de los siguientes parámetros; la frecuencia de clock del FPGA, frecuencia de la onda senoidal que se quiere generar, incremento de fase y ancho de fase, entonces la frecuencia de la onda senoidal se puede representar como:

$$f_{out} = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}} \quad (1)$$

$f_{clk}$ = Frecuencia de clock del FPGA *(250Mhz)*
$f_{out}$ = Frecuencia de la onda senoidal
$\Delta\theta$ = Incremento de fase
$2^{B_{\theta(n)}}$ = Ancho de fase

Para generar una onda senoidal con una frecuencia de 1 MHz, el incremento de fase requerido para generar dicha onda es el siguiente:

$$\Delta\theta = \frac{f_{out} x 2^{B_{\theta(n)}}}{f_{clk}}$$

$$\Delta\theta = \frac{1_{Mhz} x 2^{64}}{250_{Mhz}}$$

$$\Delta\theta = \frac{1_{Mhz} x 2^{64}}{250_{Mhz}}$$

$$\Delta\theta = 73786976294838000$$

Al incremento de fase se debe realizar un truncamiento para obtener un valor entero, pero debido a que contamos con un ancho de fase de 2^64 el

incremento de fase es entero. Posteriormente introducimos el incremento de fase en la ecuación (1) para obtener la frecuencia real de salida.

$$f_{out} = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}}$$

$$f_{out} = \frac{250_{Mhz} x 73786976294838000}{2^{64}}$$

$$f_{out} = 1_{Mhz}$$

La resolución del generador de señales de radiofrecuencia se obtiene reemplazando el incremento de fase con el mínimo valor en la ecuación (1).

$$res = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}}$$

$$res = \frac{250_{Mhz} x 1}{2^{64}}$$

$$res = 0.00000000135525272$$

### 4.2.7. Hardware para el generador de radiofrecuencias

Se seleccionó el SDR Red pitaya SIGNAL lab 250-12 ya que comprende de un FPGA ZYNQ-XC7Z020-3CLG400E más el Dual Core ARM Cortex A9, ideal para desarrollar toda la descripción de hardware dentro de esta plataforma SDR, además de contar con DAC-AD9746BCPZ de 14 bits 2 ch, ADC 250 Msps de 12 bits x 2 canales, entrada de Trigger para poder ejecutar el diseño sintetizado en una hora y fecha específica, entrada de clock de referencia de 10Mhz, puerto ethernet de 1Gbit, Extensiones de GPIOs, soporta SD card con sistema operativo Linux.

El Seeed Studio XIAO SAMD21 basado en un ARM Cortex M0 + de 32 bits y 48 MHz (SAMD21G18) con Flash de 256 KB y SRAM de 32 KB, es el

encargado de enviar mediante protocolo de comunicación SPI los parámetros de configuración hacia los GPIOs del Red Pitaya SIGNAL lab.

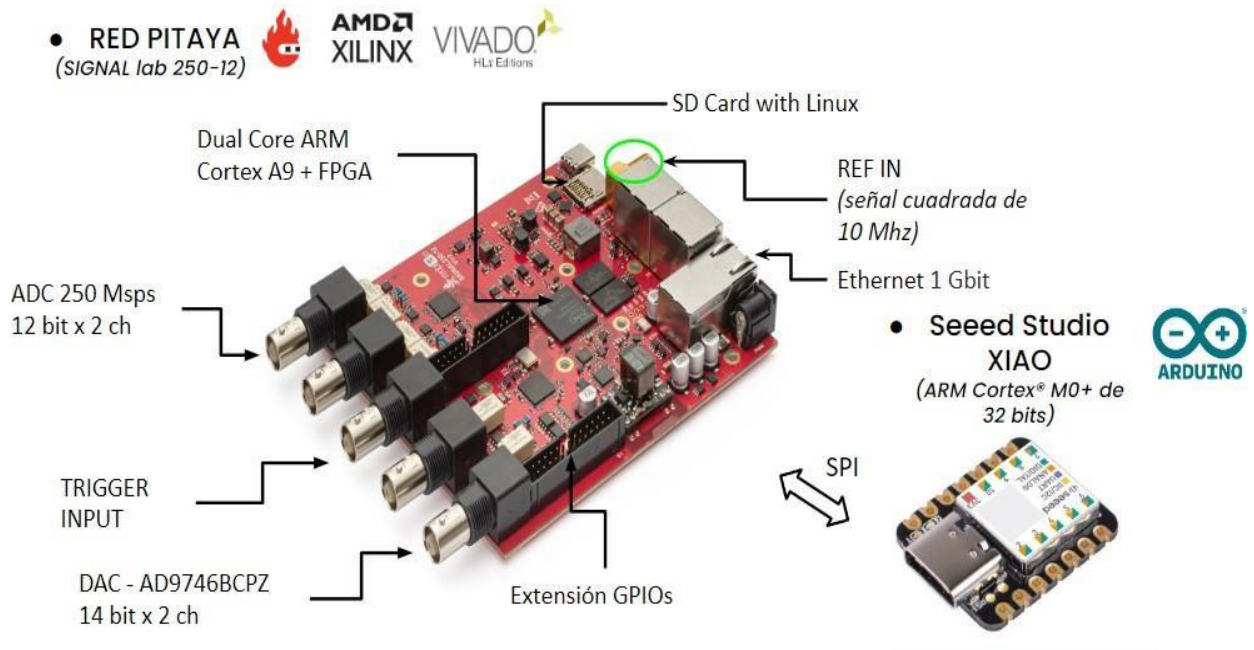


Fig. 16. Hardware del generador de señales de radiofrecuencia.

Fuente: Elaboración propia.

### 4.2.8. Elección del estilo de descripción de hardware.

En esta etapa define el estilo de descripción de hardware, el diseño de generador de señales de radiofrecuencia utilizó el estilo de descripción comportamental para cada uno de los módulos, controlador SPI, mapa de registros, oscilador controlado numéricamente (NCO), modulador BPSK y OOK, multiplexor. Para la integración de todos los módulos e Ip Cores se utiliza el estilo de descripción estructural en el software Vivado 2019.1.

### 4.2.9. Implementación del módulo de sincronismo

Cabe mencionar que el sincronismo es muy importante para los radares ionosondas es por ello que el presente diseño de generador de señales de

radiofrecuencia incluye un módulo de sincronismo con clock 10 Mhz del GPS llamado red_pitaya_kh_0 el cual es proporcionado por el repositorio de la plataforma de desarrollo Red Pitaya. Se instancia el módulo red_pitaya_kh_0 y se procede a realizar las conecciones con la entrada de clock de referencia REFin, VCXO Si 571 y Clocking Wizard (Ip Core que permite multiplicar o dividir frecuencias) se instancia en su modo diferencial, mediante descripción de hardware.

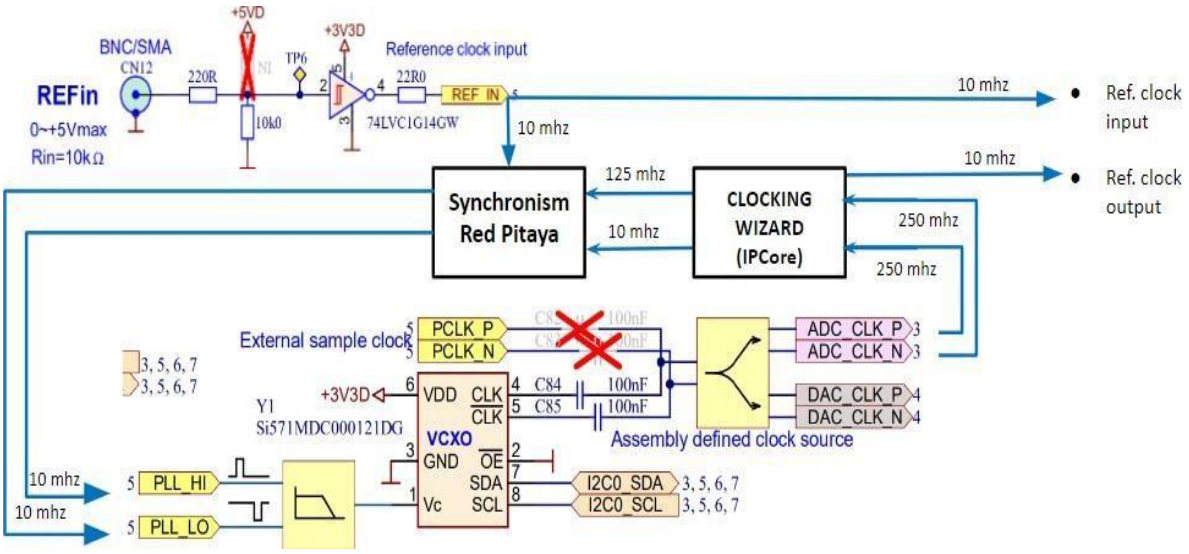


Fig. 17. Conexiones del circuito de sincronismo.

Fuente: Elaboración propia.

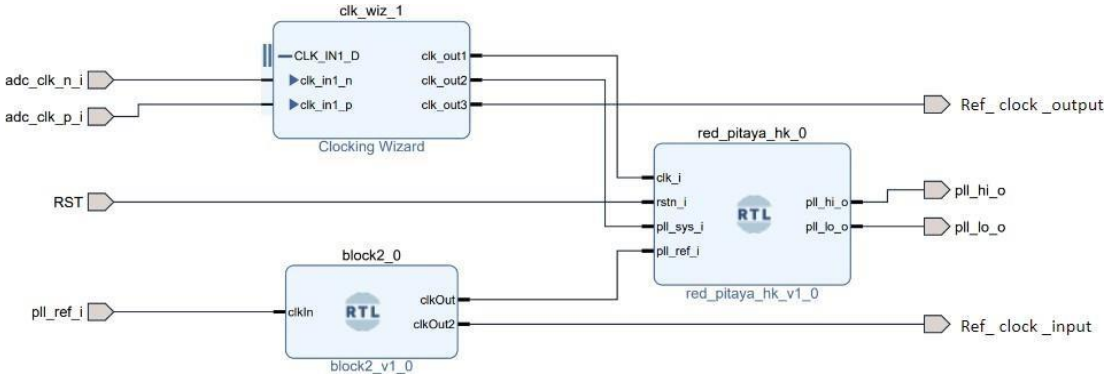


Fig. 18. Diseño de bloques del circuito de sincronismo en Vivado.

Fuente: Elaboración propia.

### 4.2.10. implementación del generador de señales de radiofrecuencia

Finalmente se integran los módulos de controlador SPI, mapa de registros, oscilador controlado numéricamente (NCO), modulador BPSK y OOK, multiplexor e Ip Cores utilizando el estilo de descripción estructural con la herramienta Create Design Block en el software Vivado.

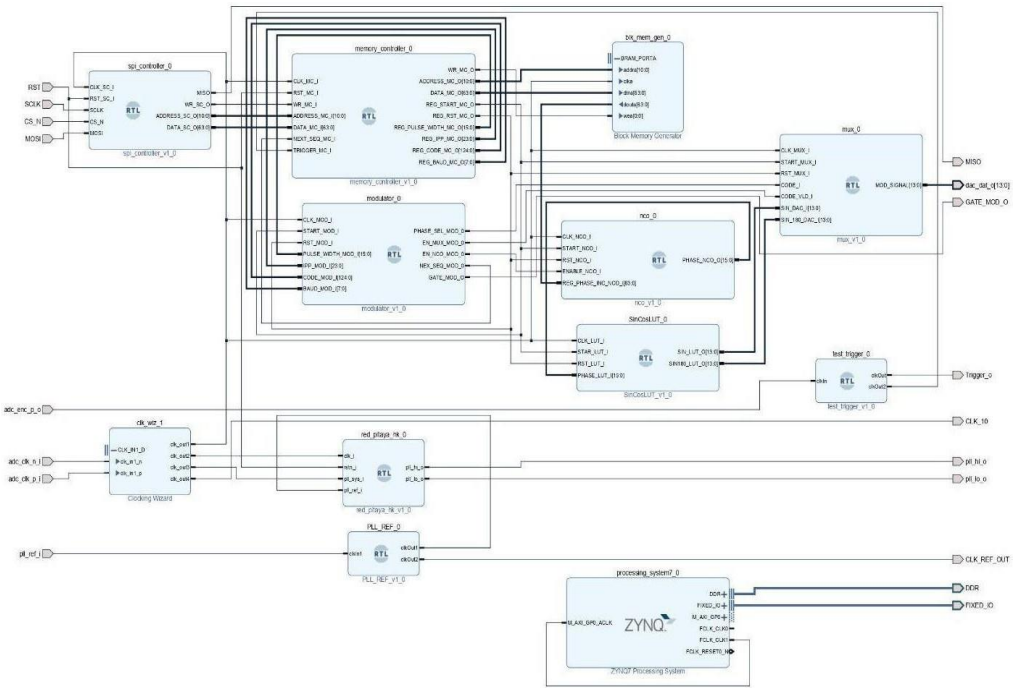


Fig. 19. Diseño de bloques del circuito de sincronismo en Vivado.

Fuente: Elaboración propia.

### 4.2.11. Configuraciones en Red Pitaya

#### a. Habilitación de la entrada de trigger

La entrada de la señal de trigger proveniente del receptor GPS trimble tiene un voltaje de 5v que pasa por un divisor de voltaje y se obtiene 1.63v que ingresa hacia In(+) del comparador de alta velocidad y en la entrada In(-) del comparador está conectada la señal de TrigRef proveniente MCP 4716 y este es programado mediante el protocolo de comunicación i2c.

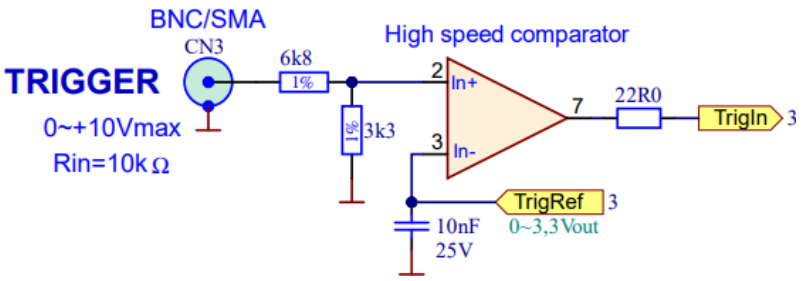


Fig. 20. Señal de trigger conectada al High speed comparator [10].


Fig. 21. Trigger level de 10 bit Dac - mcp 4716 [10].

A continuación se muestra el código de programación en el IDE arduino del mcp 4716 el cual tiene la dirección 0x60.

```
#include <Wire.h>
#include "MCP4716.hpp"

MCP4716 MCP4716(0x60);

void setup() {
  // put your setup code here, to run once:
  Wire.begin(); // Join I2C as Master
  Wire.setClock(100000);
  MCP4716.setGain(1);
  MCP4716.setVref(1);
  MCP4716.setVout(512);//512
}
```

Fig. 22. Programación del mcp 4716.

Fuente: Elaboración propia.

### b. Elevar el voltaje pico pico de la señal pulsada

La plataforma Red Pitaya cuenta con relés que al activarlos podemos cambiar el voltaje pico pico de la señal pulsadas a transmitir provenientes del DAC de 14 bits de alta velocidad (AD9746-BCPZ), dichos relés se controlan con el expansor MAX 7311 que proporciona 16 bits entrada/salida paralela.

Por defecto en modo desactivado los relés proporcionan un voltaje de 4.44v al medir en alta impedancia (1M) y 2.22v al medir con carga de 50 ohm.

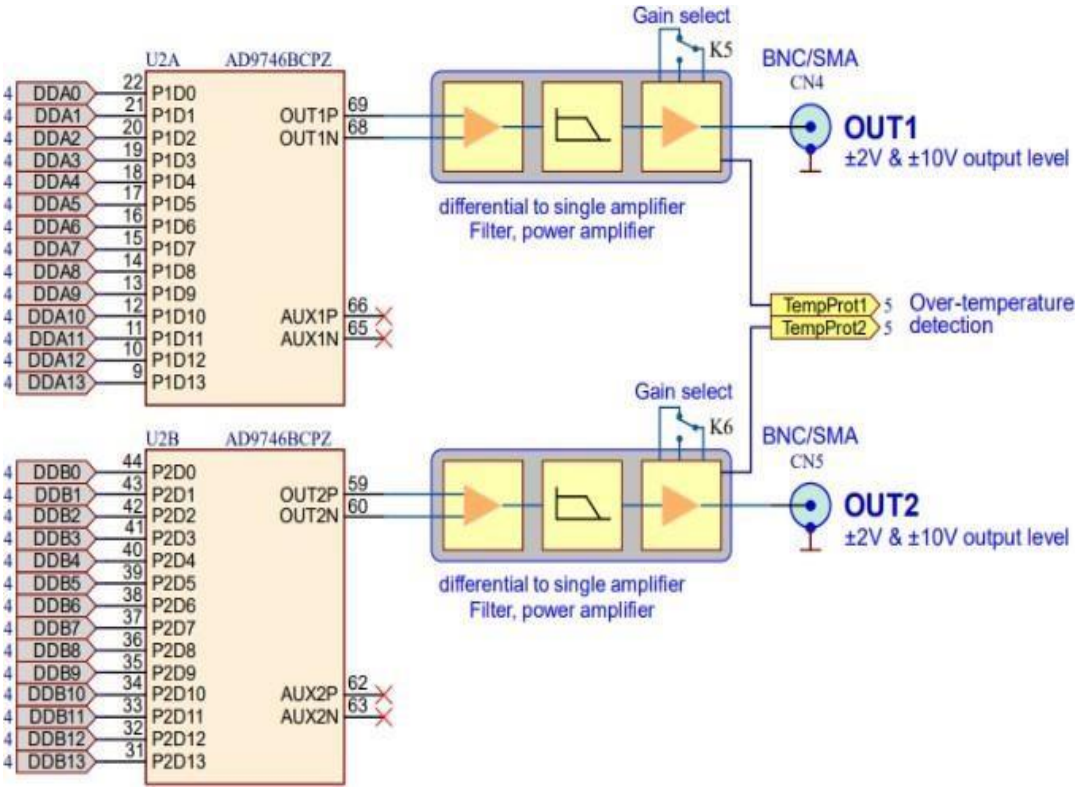


Fig. 23. DAC-AD9746BCPZ de 14 bits conectados a los relés [10].

Al activar los relés proporcionan un voltaje de 21.6v al medir en alta impedancia (1M ohm) y 10.8v al medir con carga de 50 ohm. A continuación se muestra el código de programación en el IDE de arduino para el expansor MAX 7311, el cual tiene la dirección 0x20.

```
#include <Wire.h>
#include <max7311.h>
max7311 mcp(0x20);

void setup(){
  Serial.begin(115200);
  delay(100);
  mcp.begin();
  mcp.gpioPinMode(OUTPUT);
}

void loop(){
  for (int i = 0; i < 2; i++) {
    mcp.gpioDigitalWrite(i, HIGH);
    Serial.println("GPIO º"+String(i)+" is ON");
    delay(1000);
  }
  while(1);
}
```

Fig. 24. Programación del expansor  MAX 7311.

Fuente: Elaboración propia.

### 4.2.12. Diseño de placa electrónica

#### a. Consideraciones generales para el diseño de la placa electrónica

- La Red Pitaya Signal Lab 250-12 al estar basada en la FPGA SoC Xilinx Zynq 7020 - XC7Z020 podemos acceder al Programmable Logic (PL) y al Processing System (PS).
- Para este trabajo de tesis se requiere acceder al PL por medio del protocolo de comunicación SPI para poder configurar el generador de radiofrecuencias tales configuraciones como; listado de frecuencias, IPP, ancho de pulso y repeticiones, también acceder al PS por medio del protocolo de comunicación I2C para configurar el hardware de Red Pitaya Signal Lab 250-12, estas configuraciones son habilitación de la entrada de trigger y elevar el voltaje pico pico de la señal pulsada.
- La Red Pitaya Signal Lab 250-12 cuenta con el hardware necesario para recepcionar la señal de CLK cuadrada de 10 Mhz que se utiliza como CLK de referencia para generador de radiofrecuencia. Los receptores de

GPS de la marca TRIMBLE proporciona dos señales una señal de PPS que es un pulso que se emite cada segundo y la una señal senoidal de 10 Mhz, para poder conectar la señal de CLK proveniente del receptor de GPS TRIMBLE al Red Pitaya se realiza un circuito que transforma la señal senoidal a cuadrada.

- Debido a que los GPIOs del PL y PS de la Red Pitaya Signal Lab 250-12 están accesibles en parte superior por medio de conectores jumper hembra se procede a realizar un shield que se va poder montar encima brindando su practicidad en la conexión.

### b. Lista de componentes electrónicos

| Qty | Value | Package |
|---|---|---|
| 2 | Resistencia 10k ohm | axial |
| 10 | Condensador 100nF | C1206 |
| 2 | Resistencia 10K ohm | R1206 |
| 2 | Condensador 10nF | C0805 |
| 1 | Condensador 10uF | C0805 |
| 1 | Resistencia 150 ohm | R1206 |
| 2 | Resistencia 1K ohm | R1206 |
| 1 | Resistencia 1.5K ohm | R1206 |
| 3 | Condensador 1uF | C0805 |
| 1 | Inductor 1uH | SM-1206 |
| 3 | Resistencia 30 ohm | R1206 |
| 1 | Resistencia 75 ohm | R1206 |
| 1 | Resistencia 750 ohm | R1206 |
| 1 | Single-Supply Comparators AD8611ARZ - Ultrafast 4 ns | SOIC127P600X175-8N |
| 1 | Clock Fanout Buffer NB3L553 | SOIC08 |
| 7 | Tiny Logic UHS 3-Input Or Gate NC7SZ32 | SOT23-5 |
| 1 | Low-Dropout Regulator TLV1117LV33 | SOT223 |
| 4 | Conector SMA hembra con borde de montaje | BU-SMA-V |
| 5 | Conector BNC hembra con base acodado | BU-BNC-V |
| 1 | SEEEDUINO XIAO-SAMD21 | MOD-2.54-21X17.8MM |

Tabla 5. Lista de componentes electrónicos para el shield.

Fuente: Elaboración propia.

### c. Esquemático de la placa electrónica

El desarrollo de la placa electrónica se realizó en el software EAGLE 9.6.2, en la figura 25 se muestran las conexiones de los GPIOs del PL y PS.

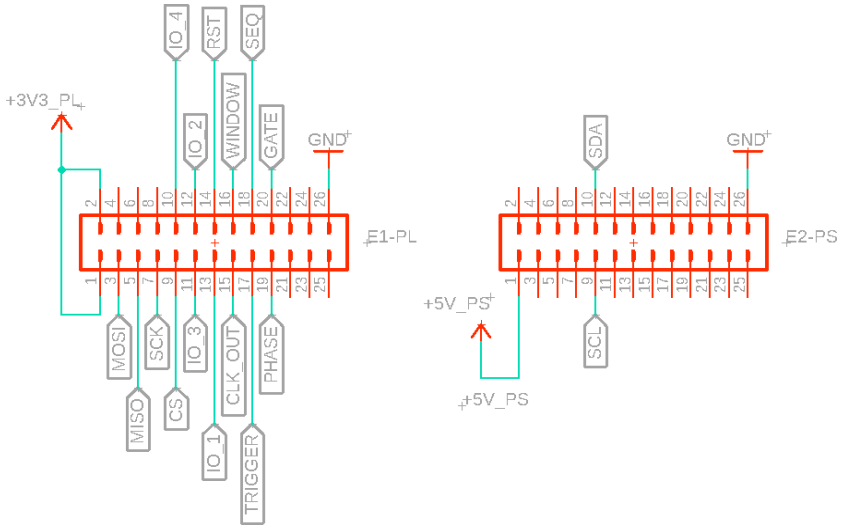


Fig. 25. Conexiones de los GPIOs del PL y PS.

Fuente: Elaboración propia.

En la figura 26 se muestran las conexiones del microcontrolador Seeeduino Xiao SAMD21 hacia los GPIOs del PL y PS. y la fuente de alimentación de 3.3v la obtiene del PL .

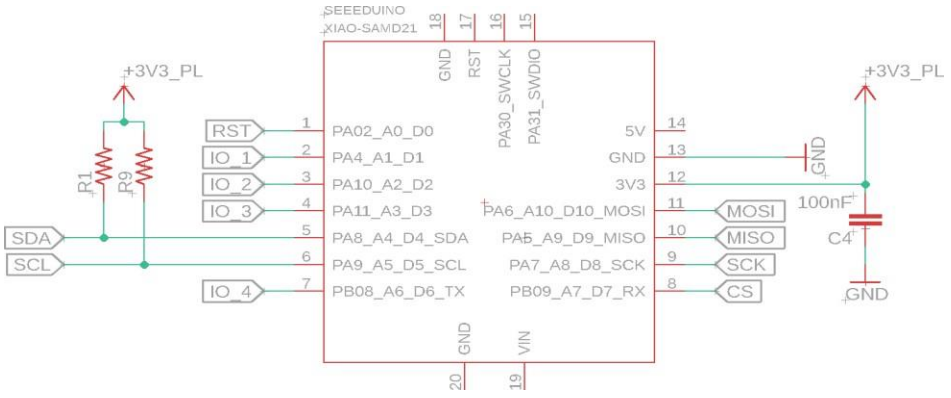


Fig. 26. Conexiones del Seeeduino Xiao.

Fuente: Elaboración propia.

En la figura 27 se muestran la etapa de regulación de voltaje y se hace uso del regulador lineal de baja caída TLV1117LV, la fuente de entrada de 5v se obtiene del PS.

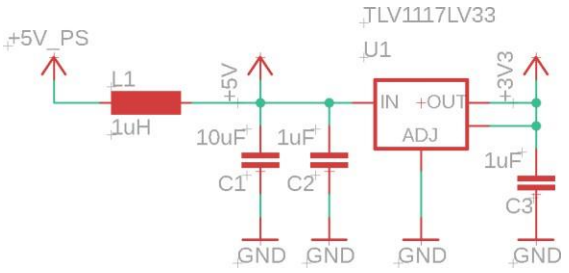


Fig. 27. Conexiones del regulador lineal de baja caída TLV1117LV.

Fuente: Elaboración propia.

En la figura 28 se muestran las conexiones del circuito que transforma la onda senoidal a cuadrada, y tiene como principal circuito integrado (CI) al comparador de alta velocidad AD8611, la señal cuadrada de 10 mhz se obtiene de la salida GPS_CLK.

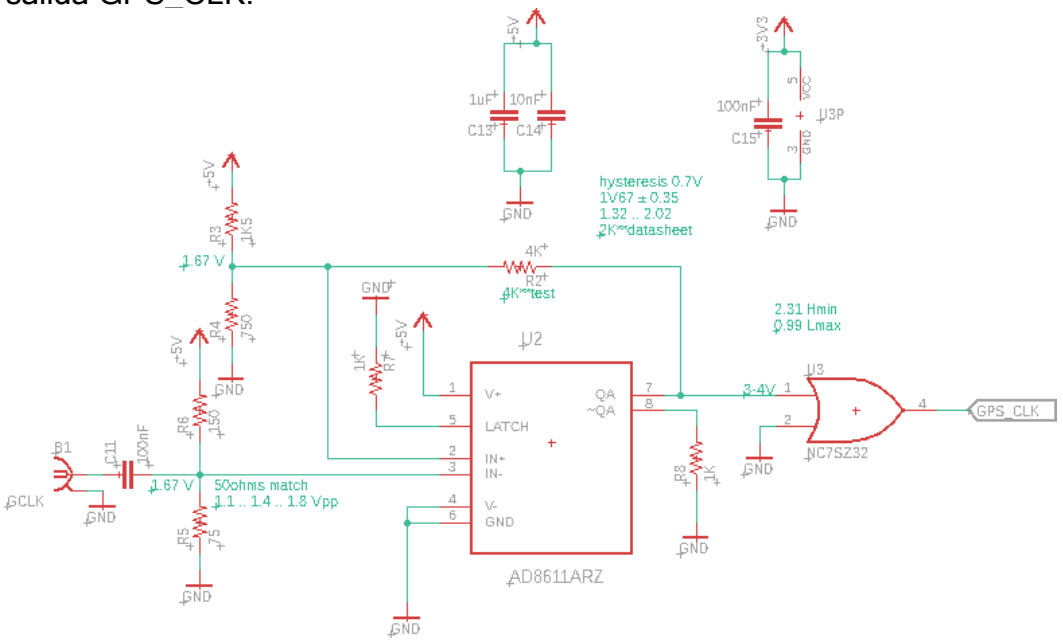


Fig. 28. Conexiones del comparador de alta velocidad AD8611.

Fuente: Elaboración propia.

Las conexiones del circuito de distribución de clock, está basado en el circuito integrado (CI) NB3L553 que es un buffer de distribución diseñado principalmente para señales de CLK de 1 entrada a 4 salidas. Se optó por el NB3L553 debido a que garantiza específicamente minimizar la distorsión de la señal de clock entre un dispositivo a otro.

Para este trabajo de tesis se obtienen dos señales de clock a partir de la señal GPS_CLK mediante este circuito de distribución de clock, ambas señales de clock B2 y B3 tienen salida SMA. La salida B2 se conecta a la entrada SMA del clock de referencia de 10Mhz del Red Pitaya llamado REF_IN y la otra salida B3 se utiliza para poder conectar al osciloscopio y verificar si se está generando correctamente la señal cuadrada de 10 Mhz. En la figura 29 se muestran las conexiones del circuito de distribución de clock.

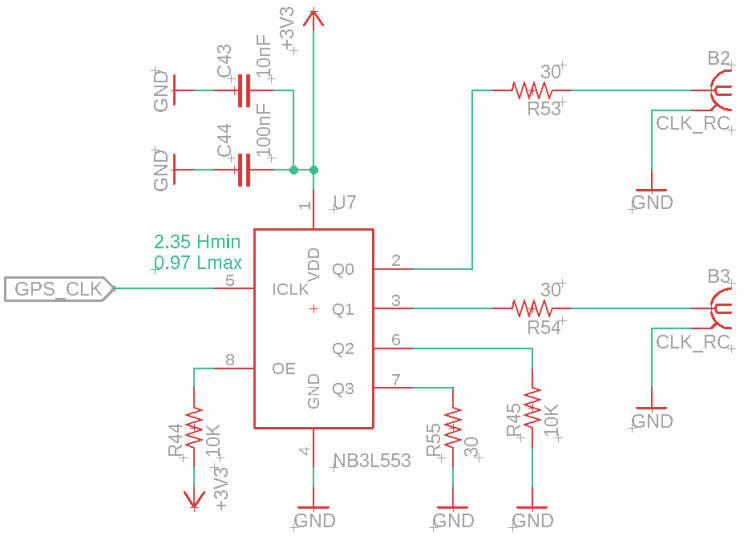


Fig. 29. Conexiones del circuito de distribución de clock.

Fuente: Elaboración propia.

Las señales de salida del PL están conectadas a conectores BNC y SMA. Se tiene 4 salidas del PL conectadas a los conector BNC hembra con base acodado y por otro lado se tiene 2 salidas del PL conectadas a los conectores SMA hembra con borde de montaje.

Las salidas que van hacia los conectores BNC hembra con base acodado son las señales digitales de “WINDOW” que muestra la ventana de inicio a final de las 1808 frecuencias, el “GATE” nos permite visualizar la ventana del ancho del pulso transmitido, la “PHASE” nos indica la fase del pulso y la señal de “SEQ” muestra una bandera de cada cambio de frecuencia realizado.

Las salidas que van hacia los conectores “SMA” hembra con borde de montaje son la señal de TRIGGER que nos indica la bandera de inicio de la generación de señales de radiofrecuencias y la señal CLK_OUT nos permite confirmar si hay sincronismo comparando esta señal cuadrada de 10 Mhz generada con la señal cuadrada de 10 Mhz de referencia.

En la figura 30 se muestran las conexiones de las señales de salida del PL hacia los conectores BNC y SMA.

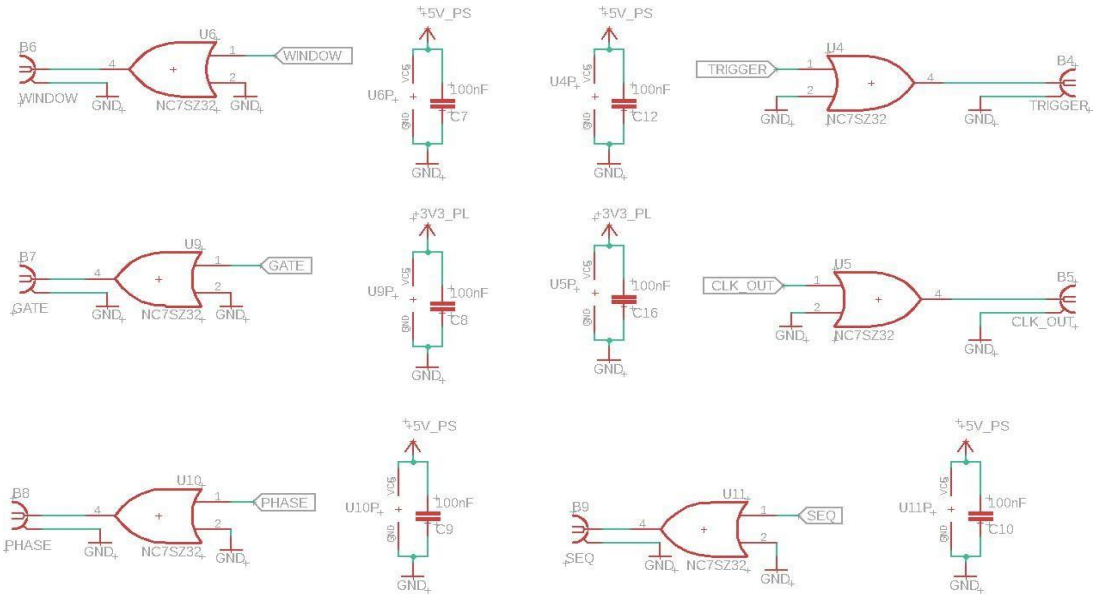


Fig. 30. Conexiones de las salidas del PL hacia los conectores BNC y SMA.

Fuente: Elaboración propia.

Finalmente se genera el archivo PulseGen.brd para realizar el ruteo de las pistas en el software Eagle.

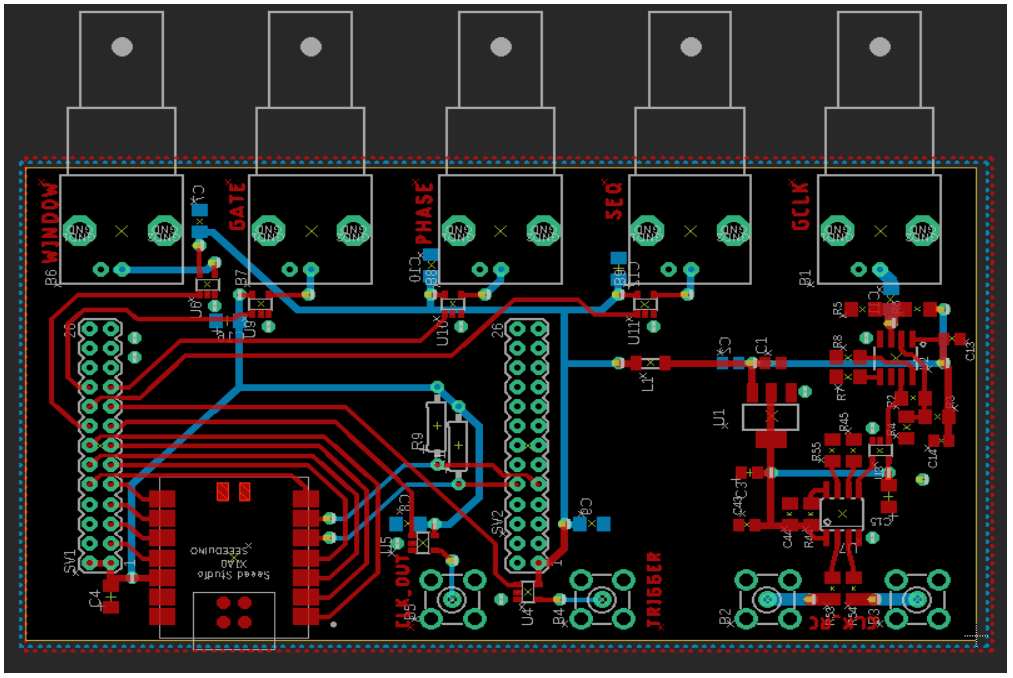


Fig. 31. Ruteo de pista de placa electrónica en software Eagle.

Fuente: Elaboración propia.

El software Eagle tiene la opción de visualizar el diseño final de la placa electrónica al enlazarlo con el software fusion 360.

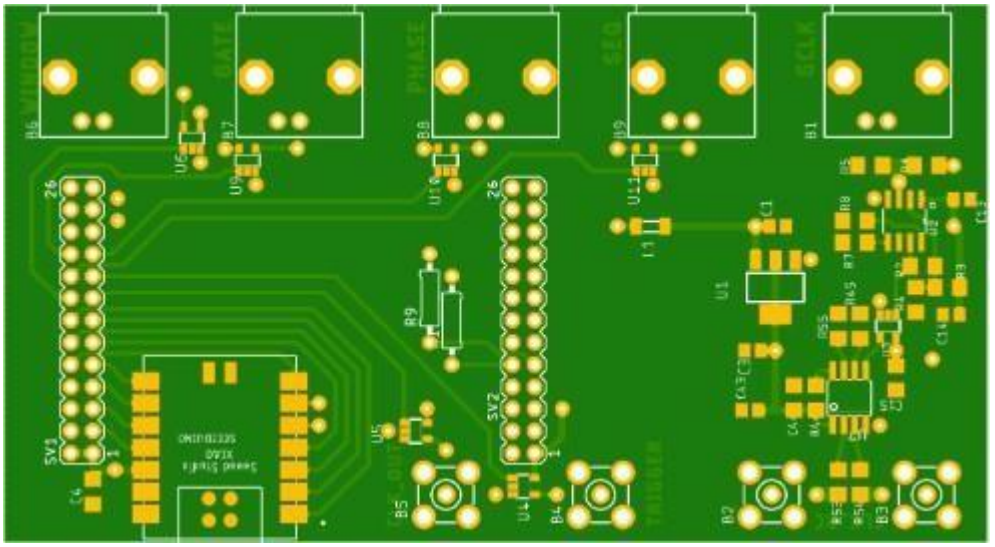

Fig. 32. Diseño final de la placa electrónica visualizado desde el software fusion 360.

Fuente: Elaboración propia.

### 4.2.13. Simulación con el test bench

Luego de integrar todos los módulos que comprende el generador de radiofrecuencias, se genera el test bench el cual describe las configuraciones iniciales y el banco de pruebas para realizar las simulaciones y resultados preliminares.

### 4.2.14. Pruebas con los instrumentos de laboratorio

El generador de señales de radiofrecuencia pasará por pruebas de laboratorio, para el cual se necesitará el generador de señales que simulará la señal de GPS de 10Mhz y la señal de Trigger que indica en qué momento se transmite el pulso. para la visualización de las señales se requerirá del Osciloscopio y Analizador de espectro.

## 4.3. Población y muestra

### 4.3.1. Población

La población para el presente trabajo de investigación son un conjunto de datos conformado por frecuencias que será tomados por el osciloscopio y analizador de frecuencias al realizar las pruebas de laboratorio en el Radio Observatorio de Jicamarca.

### 4.3.2. Muestra

La muestra para el presente trabajo de investigación son del tipo no probabilística ya que, son seleccionadas en base a distintos periodos de tiempo representativos. Se tomarán distintas muestras dentro del periodo de pruebas de laboratorio.

### 4.4. Lugar de estudio

La realización del diseño e implementación de un generador de señales de radiofrecuencia basado en FPGA SoC para la operación de un transmisor de radar ionosonda se realizó con las tarjetas de desarrollo e instrumentos de laboratorio proporcionados por el Radio Observatorio de Jicamarca que forma parte del IGP.

### 4.5. Técnicas e instrumentos para la recolección de datos

En el presente proyecto de tesis se estudió y se puso en práctica la teoría de controlador de radar ionosonda, generación de NCO, modulación BPSK y OOK. Los instrumentos a utilizados son el osciloscopio TEKTRONIK MDO-3054 con un ancho de banda de 500Mhz y analizador de espectro Agilent Technologies N9912A-4Ghz. Adicionalmente se contrastó el envío de señales de radiofrecuencia pulsadas con los datos obtenidos por el receptor de radar ionosonda IER basado en USRP y el receptor del radar ionosonda VIPIR. Cabe mencionar que tanto la instrumentación científica y locaciones fueron proporcionados por el Radio Observatorio de Jicamarca que sirve para el análisis de los resultados y conclusiones.

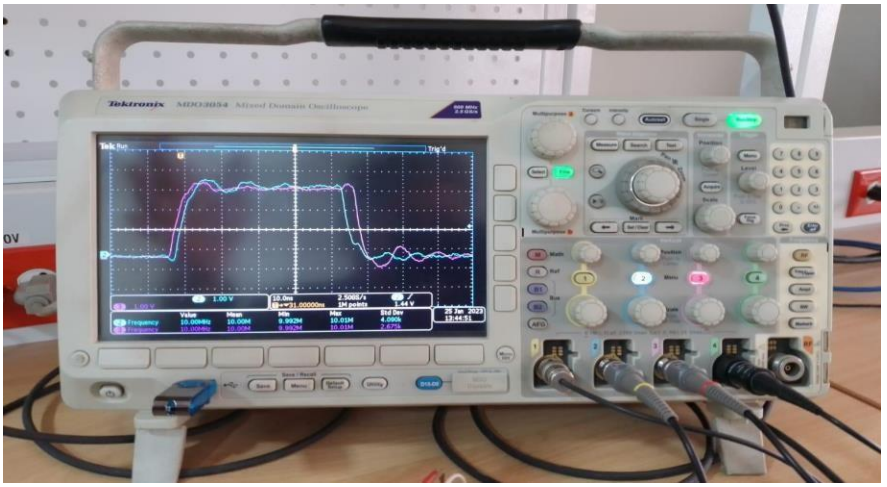


Fig. 33. Osciloscopio TEKTRONIK MDO3054 - 500Mhz.

Fuente: Elaboración propia.


Fig. 34. Analizador de espectro Agilent Technologies N9912A - 4Ghz.

Fuente: Elaboración propia.

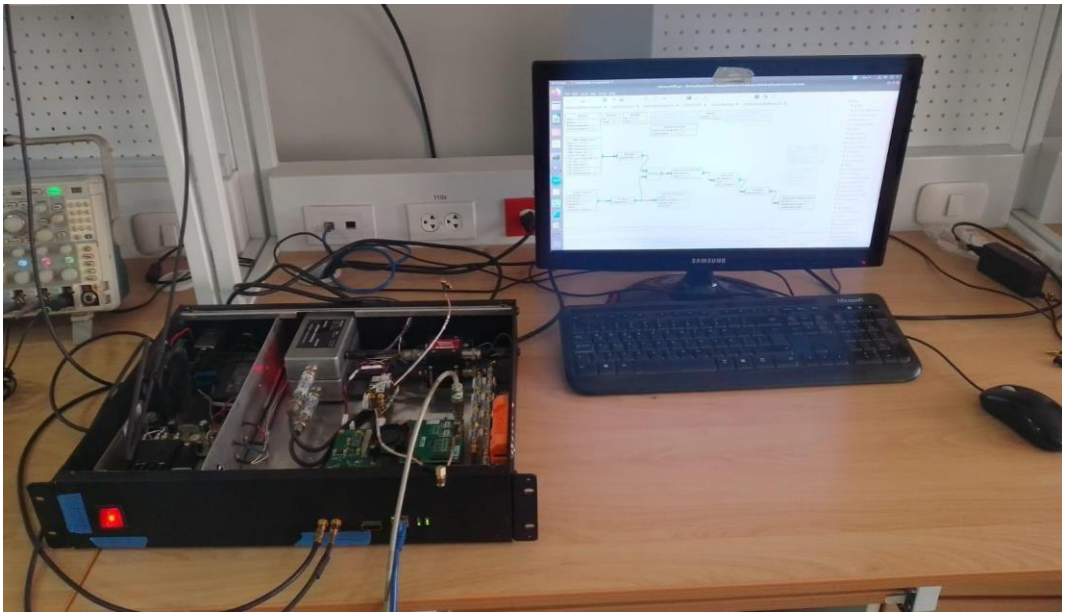

Fig. 35. Receptor de radar ionosonda IER basado en USRP.

Fuente: Elaboración propia.

(a) (b)

Fig. 36. Receptor del radar ionosonda VIPIR. (a) Gabinete, (b) Computadora.

Fuente: Elaboración propia.

### 4.6. Análisis y procesamiento de datos.

Para el análisis y procesamiento de datos se utiliza el software Vivado 2019.1. el cual nos permite obtener los gráficos de las simulaciones con banco de pruebas (test bench) y los instrumentos de laboratorio tales como el osciloscopio TEKTRONIK MOD-3054 con un ancho de banda de 500Mhz y el analizador de espectro Agilent Technologies N9912A-4Ghz para la obtención gráficos de la generación del barrido de frecuencias, gráficos del Análisis espectral en frecuencia y gráficos de la comparación de señales con persistencia. Finalmente se someterá a prueba con instrumentos científicos validados y que actualmente están operando en el Radio Observatorio de Jicamarca. A continuación, se describe los análisis que se realizarán:

**a. Gráfico de simulaciones con test bench**

Para ver la correcta simulación de generación de los pulsos generados con las configuraciones iniciales y un banco de pruebas.

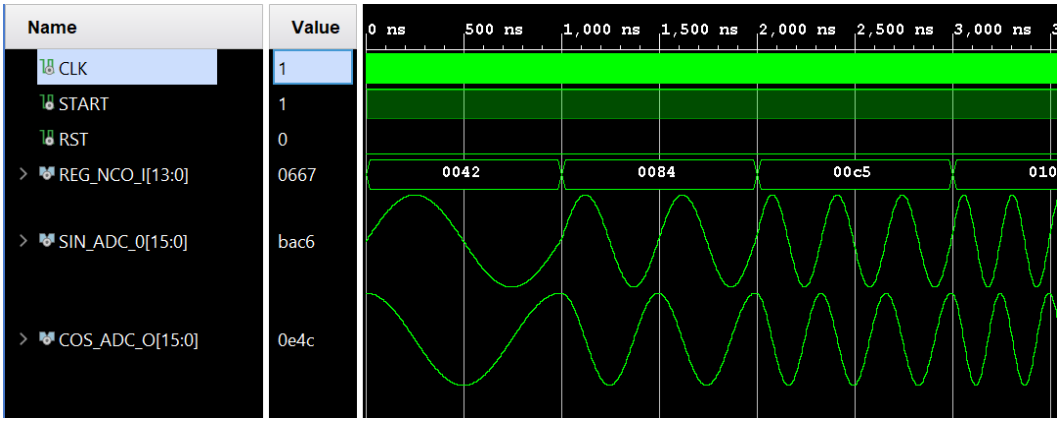


Fig. 37. Gráfico de simulaciones con banco de pruebas (test bench).

Fuente: Elaboración propia

**b. Gráficos de la generación de señales**

Para ver la correcta generación del barrido de frecuencias al detectar la señal de trigger que es recibida cada 5 minutos, también verificar el ancho del pulso transmitido y período entre pulsos emitidos por el generador de señales.

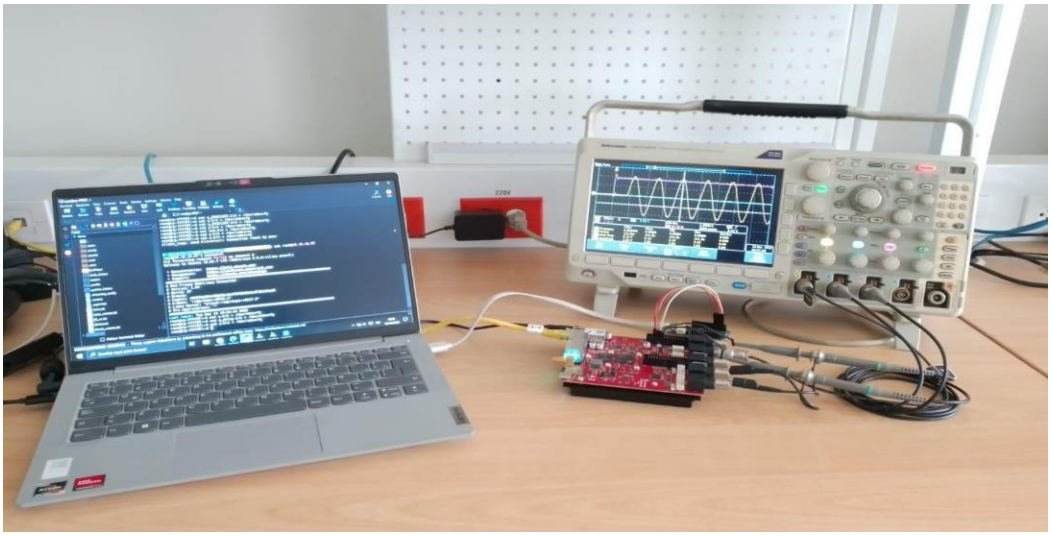

Fig. 38. Gráficos de la generación de señales con osciloscopio TEKTRONIK.

Fuente: Elaboración propia.

**c. Análisis espectral en frecuencia:**

Para ver la correcta generación del barrido de frecuencias a través del analizador de espectro.

Fig. 39. Gráficos del análisis espectral en frecuencia.

Fuente: Elaboración propia.

**d. Comparación de señales con persistencia:**

Para calcular cuánto es el desplazamiento de la señal generado con respecto a la señal de sincronismo de GPS se utilizará el osciloscopio y se configuró la lectura de las dos señales con persistencia infinita de esta manera se pudo visualizar los valores de desplazamiento máximo y mínimo entre ambas señales. Para este trabajo de tesis se utilizó el receptor de GPS de la marca TRIMBLE.


Fig. 40. Receptor y antena de GPS de la marca TRIMBLE.

Fuente: Elaboración propia.

**e. Prueba de bola de cobre**

Esta prueba se realiza con el receptor ionosonda IER (Ionospheric Echoes Receiver) basado en USRP, para verificar el correcto envío del barrido de las 1808 frecuencias. La señal RF del generador se conecta a atenuadores para poder disminuir el voltaje pico pico a alrededor de 500 a 750 mv que es el rango que soporta la entradas del receptor IER.

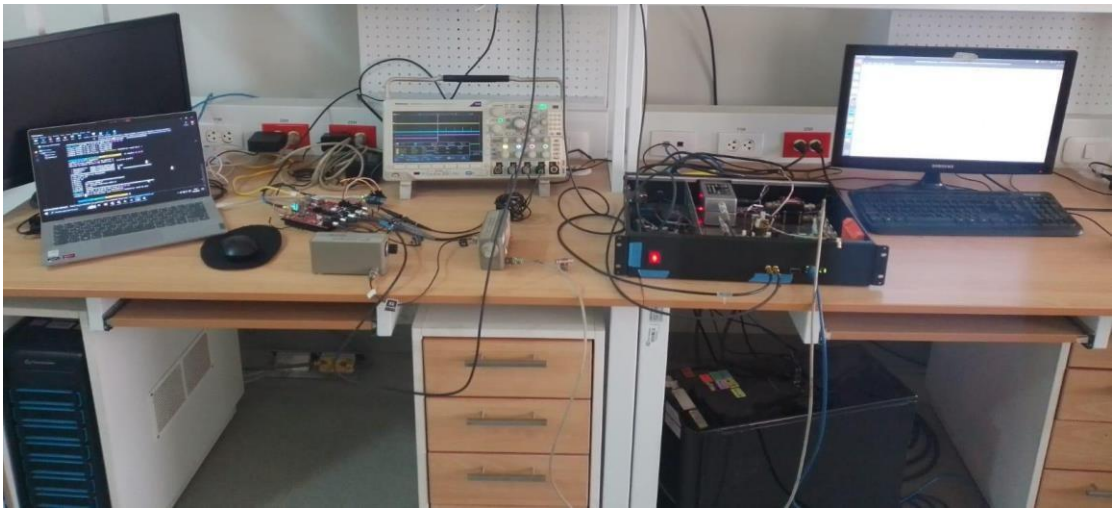

Fig. 41. Prueba de bola de cobre.

Fuente: Elaboración propia.

**f. Prueba de transmisión**

Esta prueba se realiza en las instalaciones del Radar Ionosonda “Vertical Incidence Pulsed Ionospheric Radar” (VIPIR) que se encuentra ubicado en el Radio Observatorio de Jicamarca, este radar comprende de la etapa de generación de señales de radiofrecuencia (RF), etapa de transmisión de potencia y recepción de señales. Se procede a conectar las señales de RF y GATE del nuevo generador de señales de radiofrecuencia (RF) basado en FPGA Soc hacia la etapa de transmisión de potencia, se ejecuta la síntesis de hardware y se espera a la señal de trigger que se recepciona cada 5 minutos para el inicio del barrido de frecuencias.

Fig. 42. Estructura interna del Radar ionosonda VIPIR (Generador de señales de radiofrecuencia, etapa de potencia, receptor de ecos ionosféricos).

Fuente: Elaboración propia.

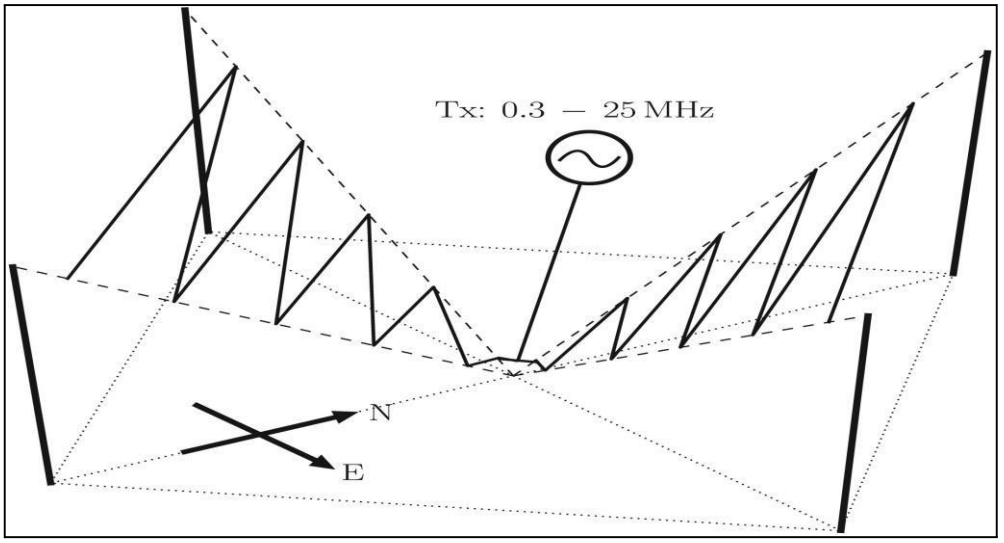


Fig. 43. Diagrama de Antena logarítmica periódica de banda ancha del transmisor de Radar ionosonda VIPIR [15].

Fig. 44. Antena logarítmica periódica de banda ancha del transmisor de Radar ionosonda VIPIR.

Fuente: Elaboración propia.

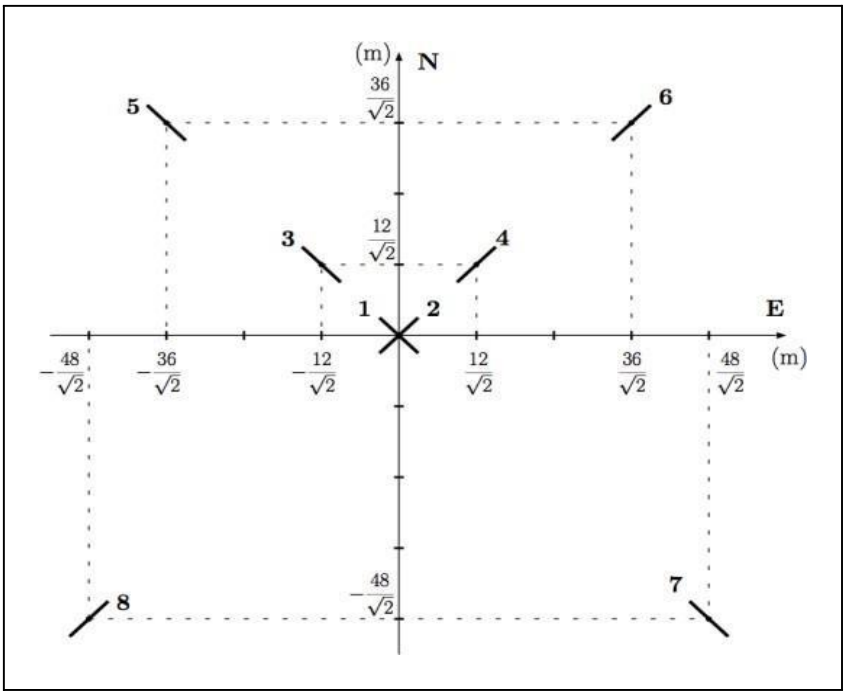


Fig. 45. Arreglo de 8 antenas dipolo del receptor de radar VIPIR [17].

Fig. 46. Una de las ocho antenas dipolo del receptor de radar VIPIR.

Fuente: Elaboración propia.

# V. RESULTADOS

## 5.1. Resultados descriptivos.

### 5.1.1. Resultados de la simulaciones

Realizado el diseño del generador de señales de radiofrecuencia en software Vivado 2019.1 se procede a realizar las simulaciones para la verificación del correcto funcionamiento. Vivado comprende de la herramienta de simulación y para ejecutarlo se requiere un código en VHDL que integra el banco de pruebas llamado testbench.

Se ejecuta el testbench del numerically controlled controller (NCO) para simular el barrido de frecuencias.

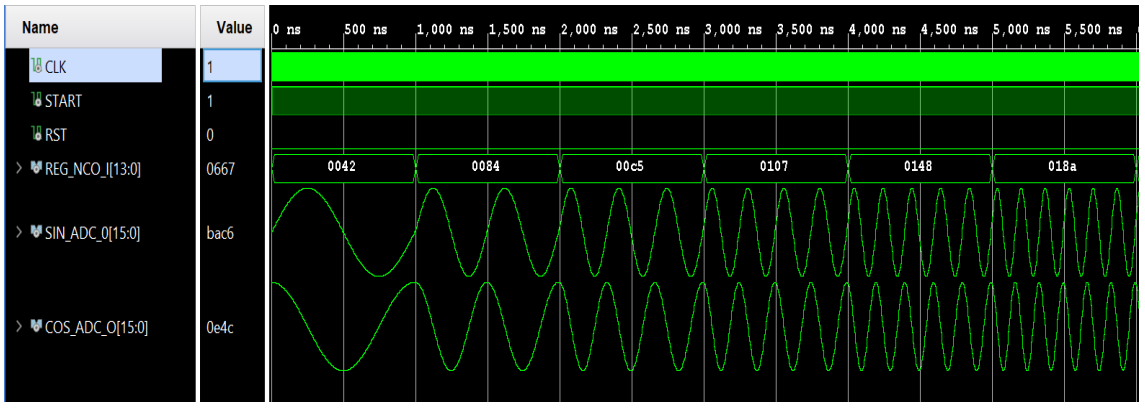


Fig. 47. Testbench del numerically controlled controller (NCO).

Fuente: Elaboración propia.

Se ejecuta el testbench del generador de señales de radiofrecuencia para un período entre pulsos emitidos (IPP) de 1 ms, un ancho de pulso de 10 us y modulación BPSK y OOK.

En la figura 48 se puede corroborar que la simulación con el banco de pruebas de un IPP de 1ms responde de manera correcta dado que el primer pulso se emite en el tiempo de 8 000 032 000 ps y el segundo a 9 000 032 000 ps.

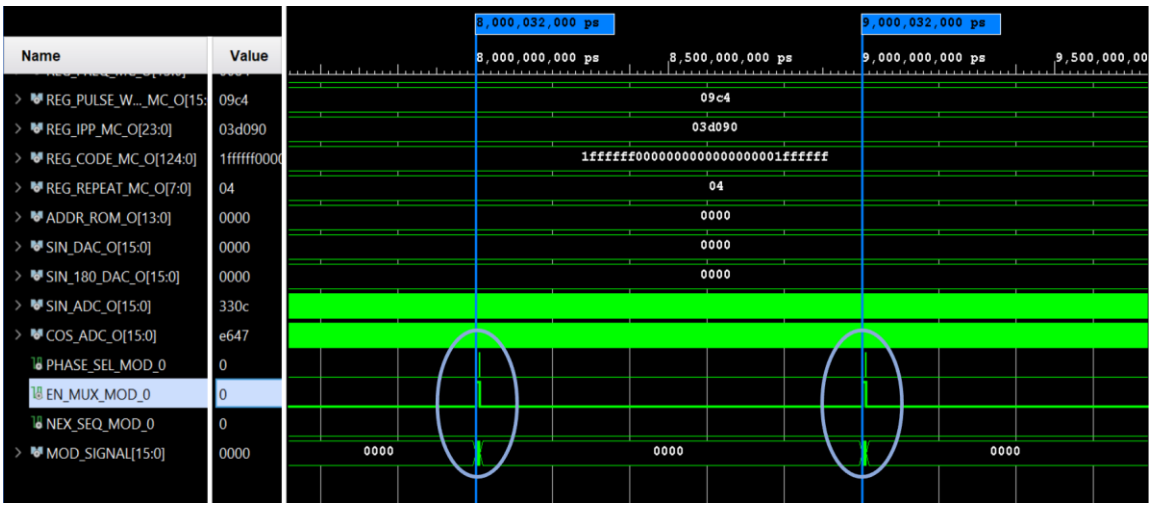


Fig. 48. Testbench para verificación del período entre pulsos emitidos (IPP).

Fuente: Elaboración propia.

En la figura 49 se puede corroborar que la simulación con el banco de pruebas de un ancho de pulso de 1us responde de manera correcta dado que la señal de radiofrecuencia de pulsada inicia la transmisión en el tiempo de 9 000 032 000 ps y culmina la transmisión a 9 010 032 000 ps.

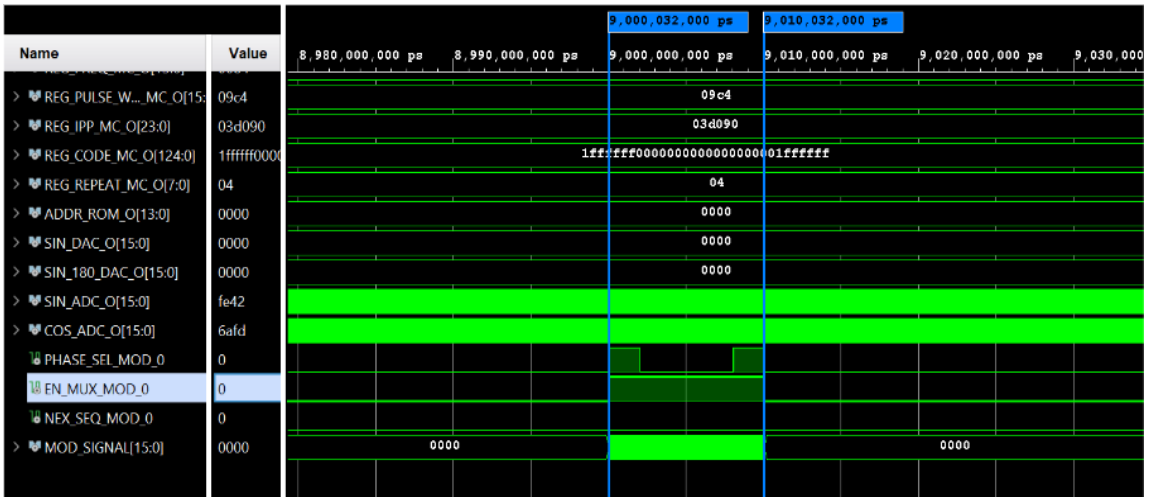


Fig. 49. Testbench para verificación del ancho de pulso emitido.

Fuente: Elaboración propia.

En la figura 50, se confirma de manera concluyente que la simulación utilizando el banco de pruebas con un ancho de pulso modulado BPSK de 1 microsegundo exhibe un comportamiento preciso. Esto se evidencia al observar que la señal de radiofrecuencia pulsada inicia la transmisión a los 9 000 032 000 picosegundos y finaliza dicha transmisión a los 9 010 032 000 picosegundos.

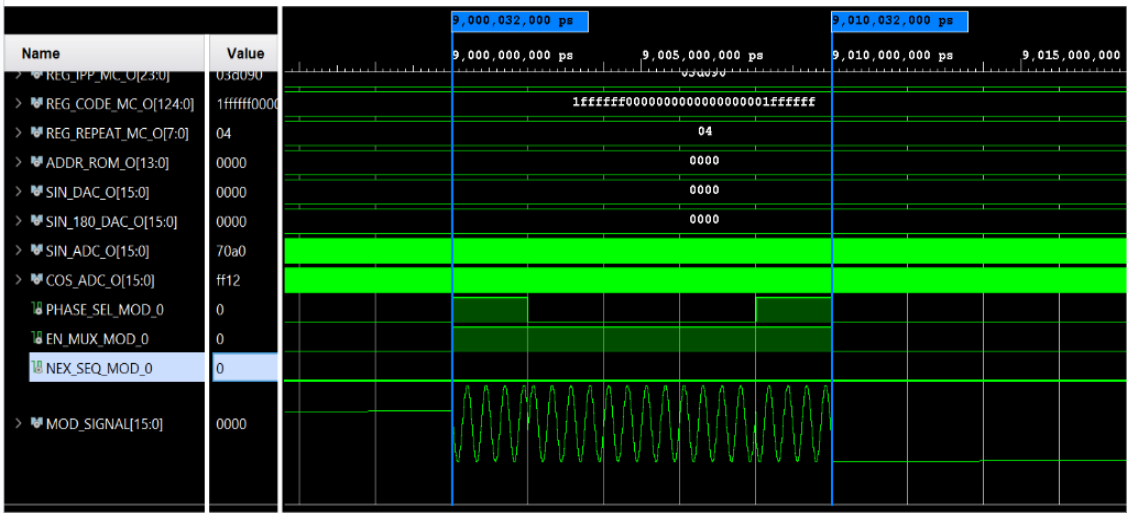


Fig. 50. Testbench para verificación del ancho de pulso BPSK emitido.

Fuente: Elaboración propia.

Las señales de RF requieren de una señal llamada Gate que es la ventana de tiempo que se activará el transmisor de alta potencia y debido a que los transmisores son analógicos requieren un tiempo mínimo para poder estar listos para la trasmisión de señales pulsadas. Es así que la señal Gate inicia 18us antes y termina 18us después.

En el Testbench de la duración de las señales RF y GATE se puede verificar que la señal de GATE inicia la transmisión en el tiempo 5 000 014 000 us y la señal de RF inicia posteriormente en el tiempo 5 018 014 000 us dando así la diferencia de inicio de 18 us.

La señal de RF culmina la transmisión en el tiempo 5 078 014 000 us y la señal de GATE culmina posteriormente en el tiempo 5 096 014 000 us dando así la diferencia de culminación de 18 us.

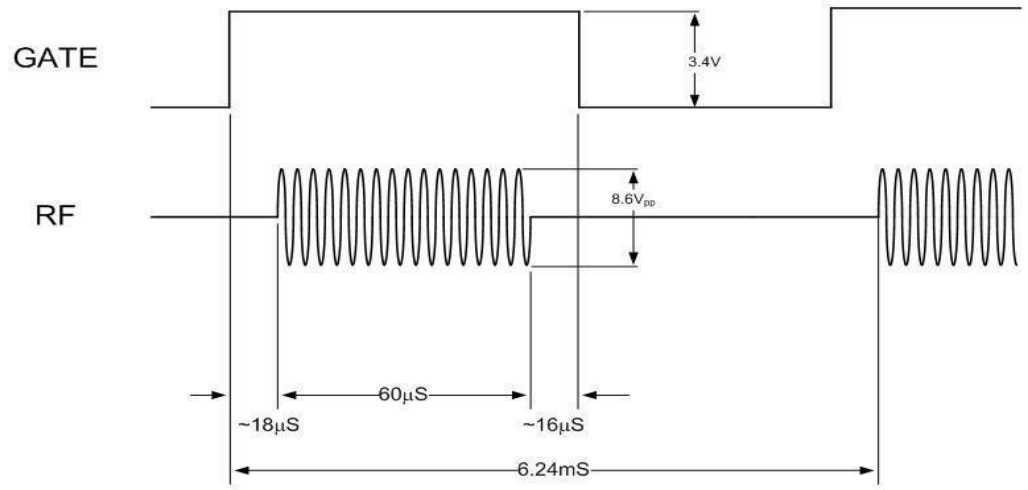


Fig. 51. Descripción gráfica de las duraciones de tiempo y amplitudes de las señales RF y GATE.

Fuente: Elaboración propia.

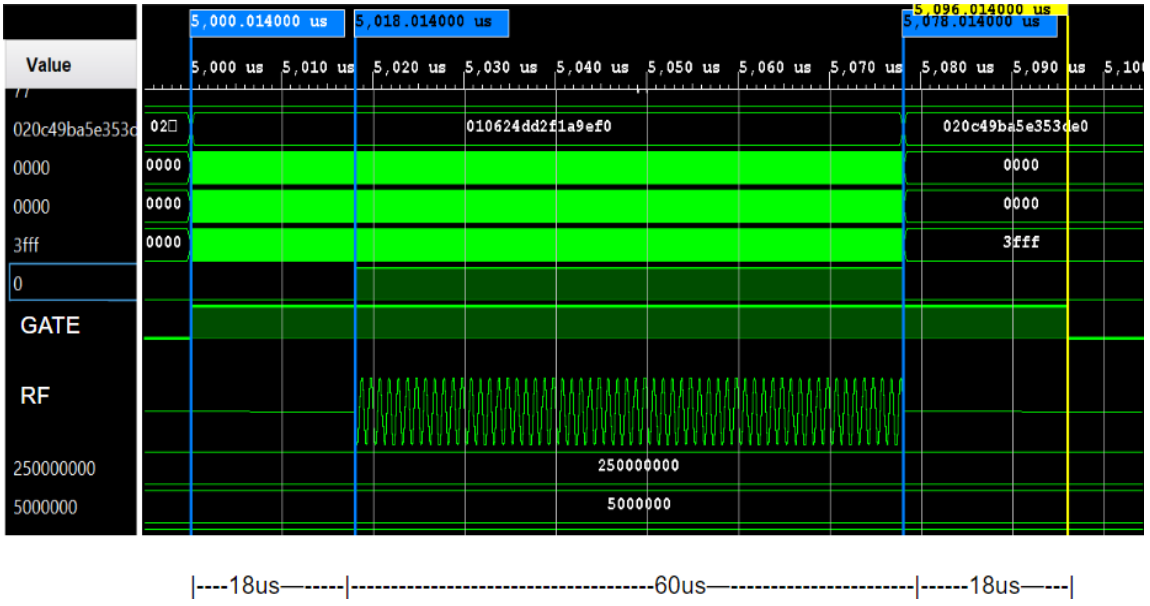


Fig. 52. Testbench de la duración de las señales RF y GATE.

Fuente: Elaboración propia.

### 5.1.2. Resultados de la implementación del Generador de señales de radiofrecuencia.

#### 5.1.2.1. Resultados de la implementación de la tarjeta electrónica

Se diseñó la placa electrónica para ser montada en la parte superior de la placa Red Pitaya Signal Lab 250-12. La placa electrónica proporciona 5 salidas extras con conector del tipo BNC, tales son; Windows, Gate, Phase, Next-Sequence y Clk-External(sinusoidal). Adicionalmente proporciona 4 salidas del tipo SMA, tales son; 2 salidas de Clk-External(square), Trigger(PCB) y Clk-internal. La tarjeta Red Pitaya Signal Lab 250-12 posee 2 ADC de alta velocidad, 2 DAC de alta velocidad, una señal de Trigger(Red Pitaya), CLK-input(Red Pitaya). La placa electrónica acondiciona las características necesarias para que la tarjeta Red Pitaya Signal Lab 250-12 opere como un generador de señales de radiofrecuencia.

Características de las señales de la tarjeta electrónica:

a. **Entradas BNC:**

- Clk-External(sinusoidal) = Ingresa un señal senoidal de 10 Mhz proveniente del receptor GPS de la marca TRIMBLE para luego ser convertida a una señal cuadrada de 10 Mhz.

b. **Salida BNC:**

- Windows = Señal de la ventana de operación del barrido de frecuencias (1808 pulsos de radiofrecuencias).
- Gate = Señal de la ventana de operación de un pulso (96 us).
- Phase = Señal que nos indica la fase de señal de radiofrecuencia, estas pueden ser seno(0º) o seno(180º).
- Next-Sequence = Señal que envía un flag cada cambio de frecuencia.

c. **Salida SMA:**

- Clk-External(square) = Se generan 2 señales cuadras de 10 Mhz a partir de la señal senoidal de 10 Mhz que ingresó al Clk-External(sinusoidal). Uno de los canales se envía al CLK-input(Red Pitaya) y el otro lo utilizamos de monitoreo.
- Trigger(PCB) = La señal de inicio de operación del generador de señales de radiofrecuencia ingresa por Trigger(Red Pitaya) y la podemos monitorear por este canal.
- Clk-internal = Una de las señales cuadras de 10 Mhz ingresa a CLK-input(Red Pitaya) y luego la síntesis de hardware desarrollada genera un Clk-internal con el que opera todo el sistema y lo podemos monitorear por este canal.

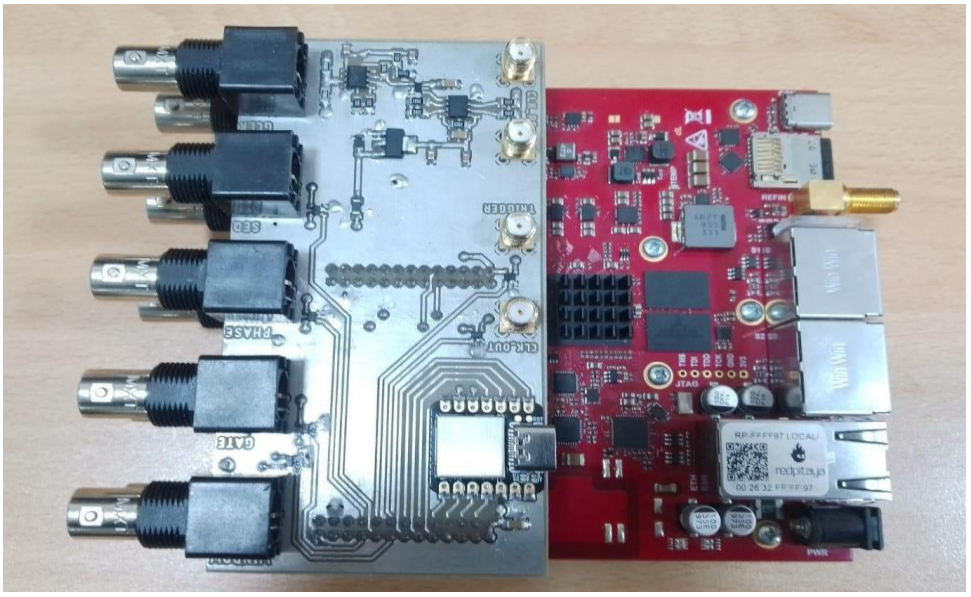

Fig. 53. Tarjeta electrónica Implementada y montada en la placa Red Pitaya Signal Lab 250-12.

Fuente: Elaboración propia.

#### 5.1.2.2. Resultados de la implementación del Rack

Para la puesta en operación en las instalaciones del radar ionosonda “Vertical Incidence Pulsed Ionospheric Radar” (VIPIR) se implementó dentro de un rack 13x17x5.25 pulgadas. con los sistemas de alimentación, ventilación e

interfaces de comunicación empotrados en el panel del rack necesarios para su correcto funcionamiento.

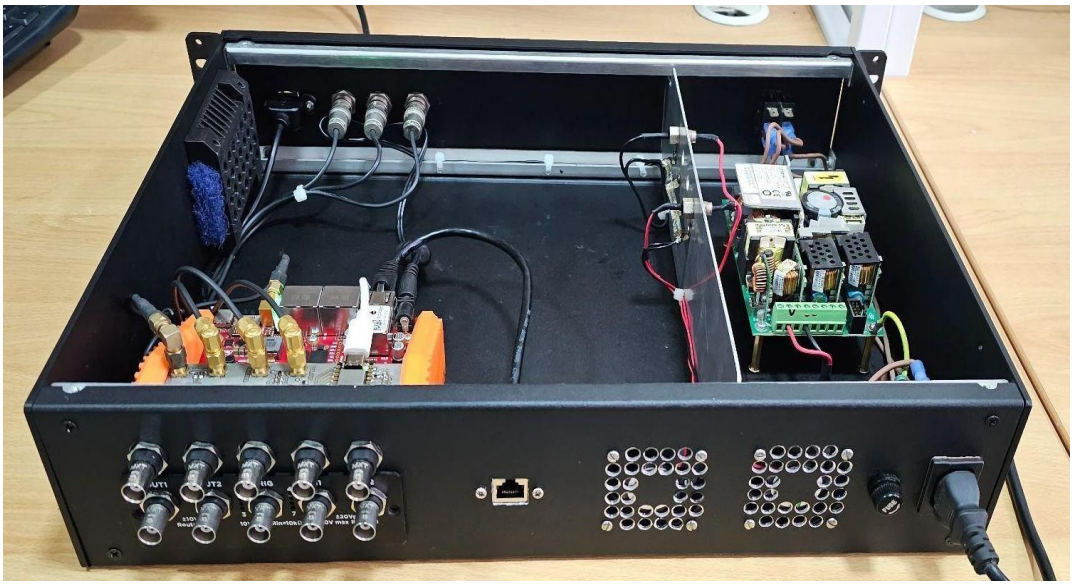

Fig. 54. Parte posterior del rack del generador de señales de radiofrecuencia para operar transmisor de radar ionosonda.

Fuente: Elaboración propia.

Fig. 55. Parte frontal del rack del generador de señales de radiofrecuencia para operar transmisor de radar ionosonda.

Fuente: Elaboración propia.

### 5.1.3. Cargar el archivo Bitstream en la Red Pitaya Signal Lab 250-12

En el software Vivado 2019.1 se desarrolló la síntesis, implementación y Bitstream (.bit) del trabajo de investigación, basado en el lenguaje de descripción VHDL. Se utilizó una laptop AMD Ryzen 7 de la serie 5700 U con Radeon Graphics, frecuencia de reloj 1.80 GHz y memoria RAM de 16 GB, lo que disminuyó los tiempos de generación de archivo Bitstream (.bit), el tiempo de generación de .bit es aproximadamente 3 minutos.

Para el cargado del archivo Bitstream se emplea el software MobaXterm, debido a que la Red Pitaya Signal Lab 250-12 comprende de un sistema operativo (OS) basado en Ubuntu 16.04.7 LTS accedemos al OS mediante el protocolo de comunicación Secure Shell (SSH), se configura de IP dinámica a estática para poder acceder siempre con la misma IP, copiamos el archivo .bit y lo cargamos al sistema mediante el siguiente comando:
>>cat rf_signal_generator.bit > /dev/xdevcfg

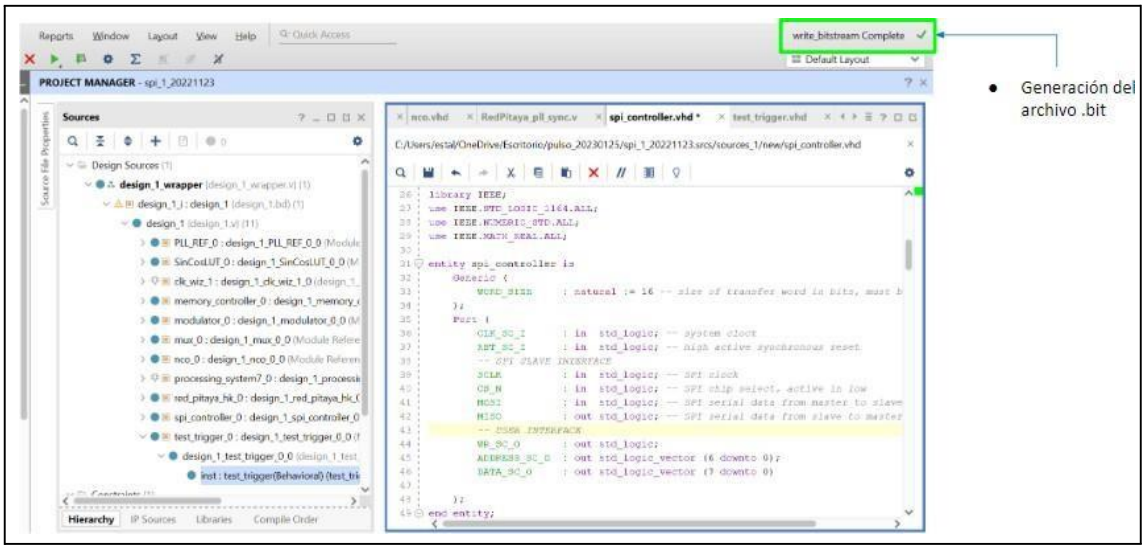


Fig. 56. Generación del archivo Bitstream .bit en el software Vivado 2019.1.

Fuente: Elaboración propia.

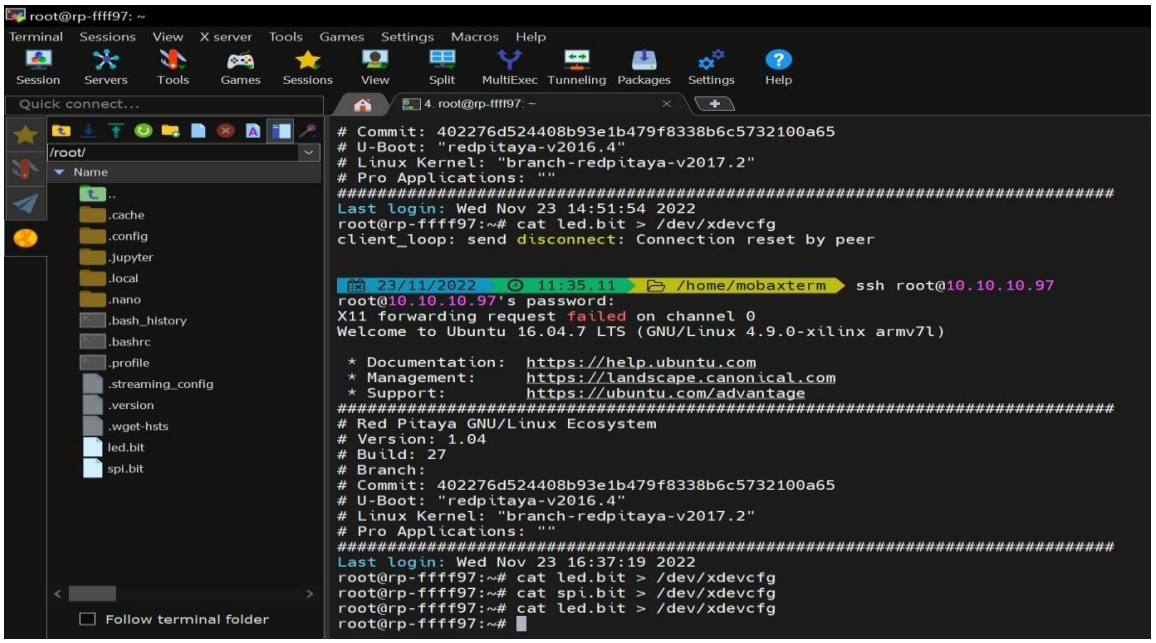


Fig. 57. Cargar el archivo Bitstream en la Red Pitaya Signal Lab 250-12.

Fuente: Elaboración propia.

### 5.1.4. Sincronismo de radar

#### 5.1.4.1. Sincronismo de la señal de clock de GPS y clock generado por generador de señales de radiofrecuencia

Los radares requieren sincronizarse con la señal de GPS, para el correcto envío y adquisición de señales de radiofrecuencia, En este caso utilizaremos el receptor de GPS TRIMBLE que proporciona la señal senoidal de 10 Mhz a la tarjeta electrónica y luego ésta envía la señal cuadrada de 10 Mhz llamada Clk-External(square) a la entrada de CLK-input(Red Pitaya) correspondiente al generador de señales de radiofrecuencia.

En la figura 58. se visualiza dos señales, la señal de color Lila corresponde al Clk-External(square) de 10 Mhz y la señal de color Jade corresponde al clock de 10 Mhz que es generada por el sistema desarrollado llamado Clk-internal.

Cabe mencionar que este trabajo de investigación opera con DAC de alta velocidad es por ello que la señal de 10 Mhz generada por el sistema luego se eleva la frecuencia a 250 Mhz usando PLL (ipCore) para sincronizar a todos los módulos interconectados mediante el diseño estructural..


Fig. 58. Sincronismo de la señal senoidal de clock de GPS convertida a señal cuadrada (Lila) y clock generado por generador de señales de radiofrecuencia (Jade).

Fuente: Elaboración propia.

### 5.1.4.2. Comparación de la señal de clock de GPS y clock generada por sistema con persistencia

Para esta comparación se realiza un zoom con el osciloscopio y se configura en persistencia infinita para poder visualizar el desfase entre la señal generada y la señal de GPS. En la figura 59 se puede observar con ayuda de los marcadores del osciloscopio que la señal de clock generada por el sistema (Jade) tiene un jitter máximo de 500 ps.

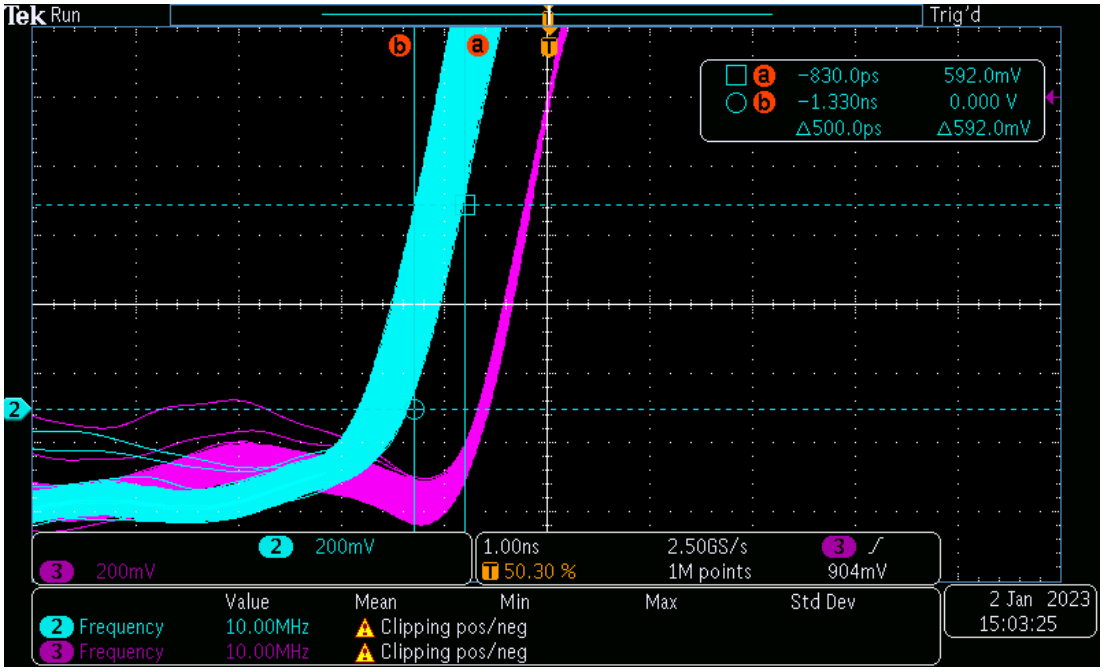


Fig. 59. Señal de clock generada por el sistema con un jitter de 500 ps.

Fuente: Elaboración propia.

### 5.1.5 Análisis de la generación de señales de radiofrecuencia

#### 5.1.5.1. Análisis en el tiempo de la generación de señales de radiofrecuencia

En este análisis se emplea el osciloscopio para poder capturar todas las señales de Rf emitidas por el generador, considerando que son 1808 frecuencias se seleccionaron las frecuencias 1 Mhz, 2 Mhz, 3 Mhz y 25 Mhz moduladas en OOK y BPSK (señal RF de color amarillo) a un ancho de pulso de 10us (señal de color lila), con la finalidad de analizar la mínima y máxima generación de frecuencias requerida para resolver la problemática abordada en este trabajo de investigación.

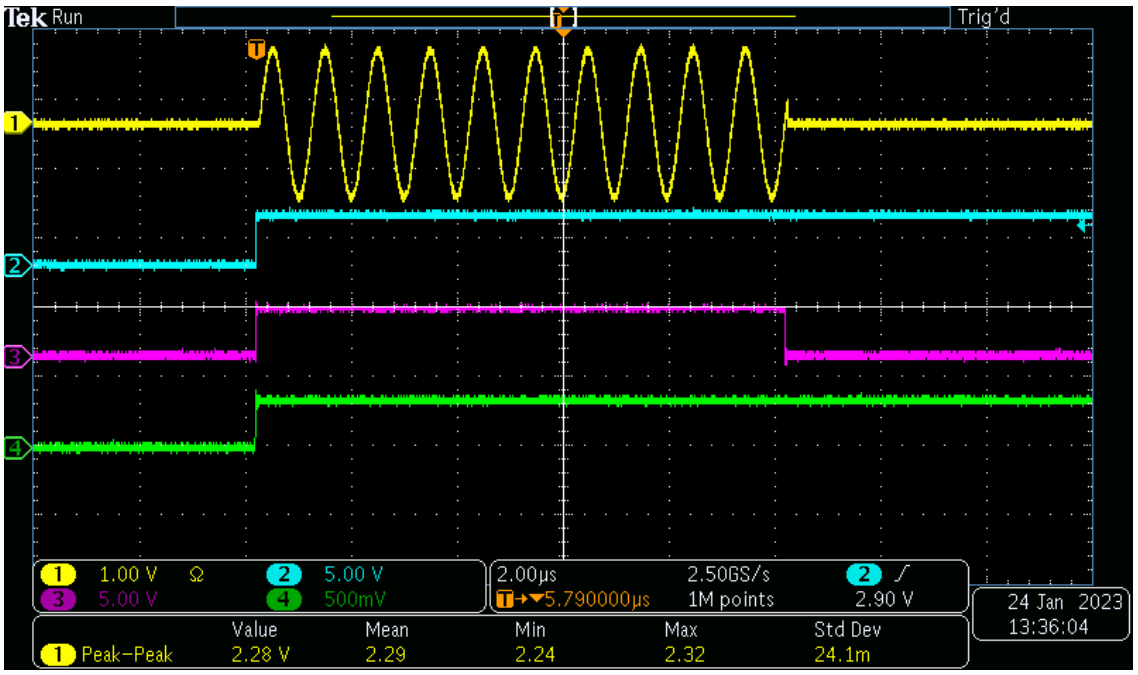


Fig. 60. Señal de radiofrecuencia generada a 1 Mhz con modulación OOK.

Fuente: Elaboración propia.

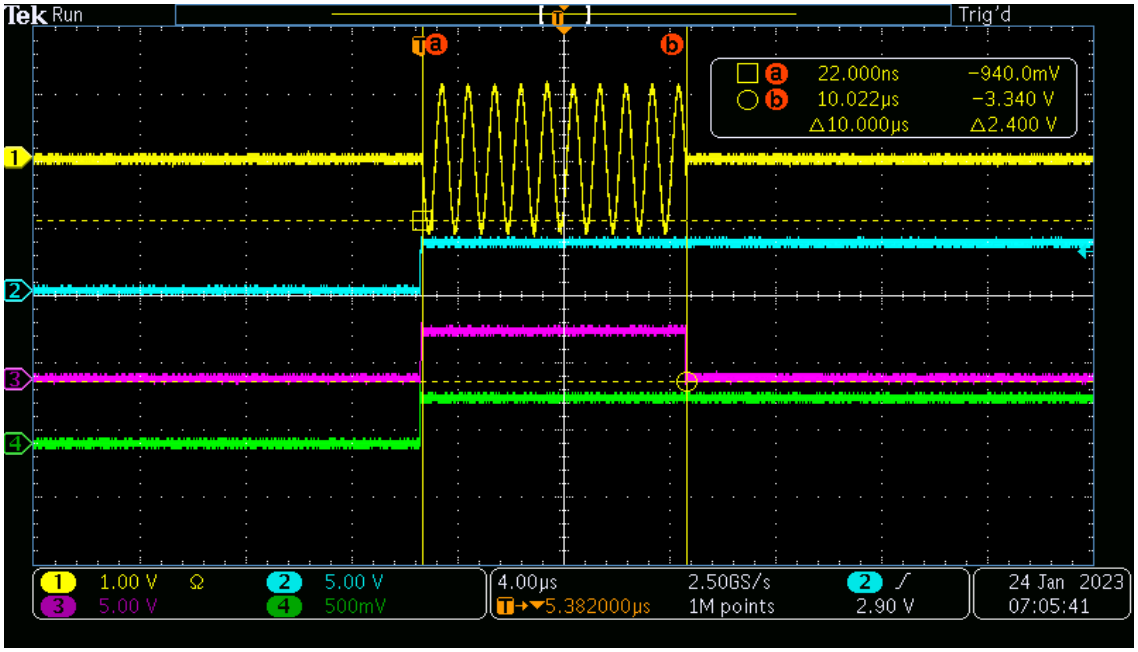


Fig. 61. Señal de radiofrecuencia generada a 1 Mhz con modulación OOK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

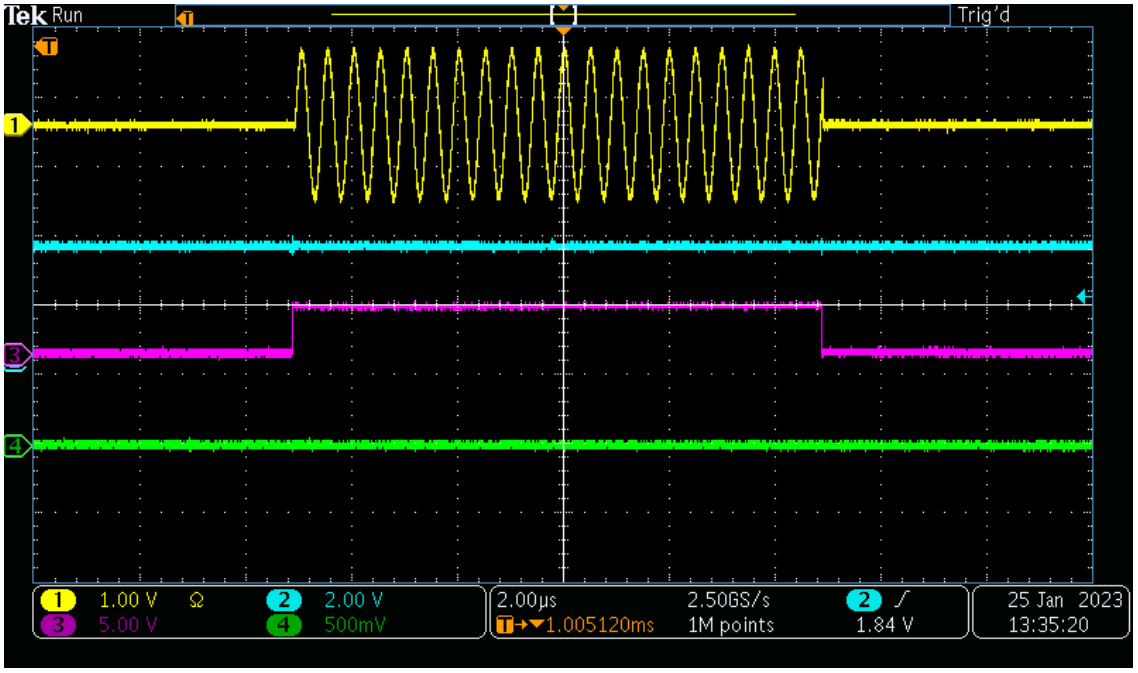


Fig. 62. Señal de radiofrecuencia generada a 2 Mhz con modulación OOK.

Fuente: Elaboración propia.

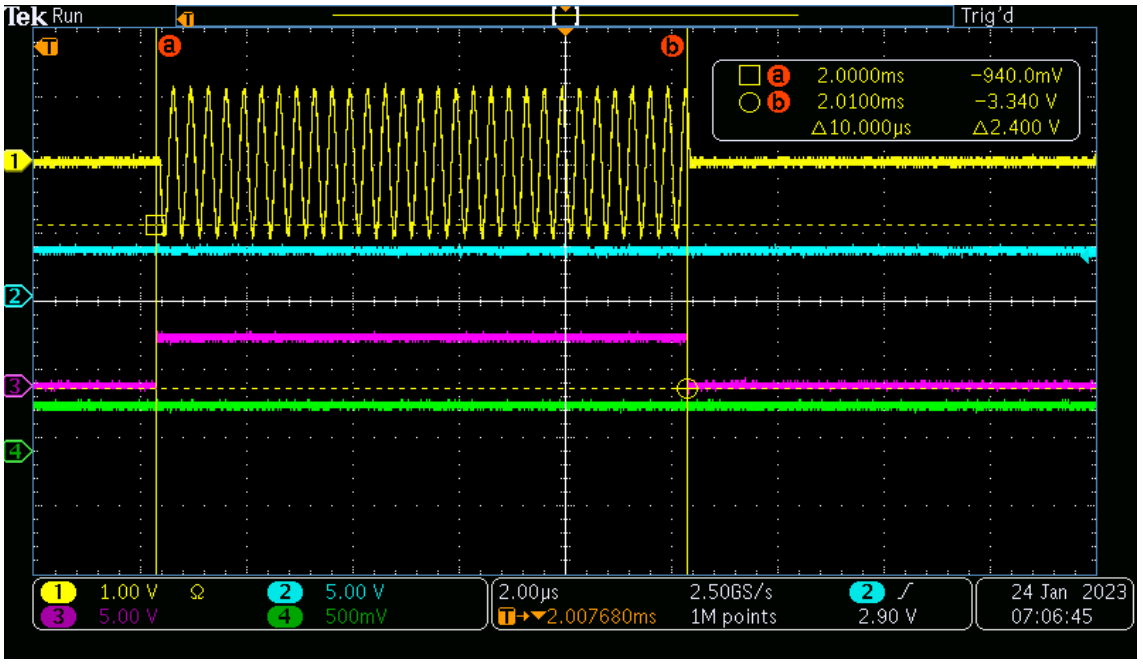


Fig. 63. Señal de radiofrecuencia generada a 2 Mhz con modulación OOK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

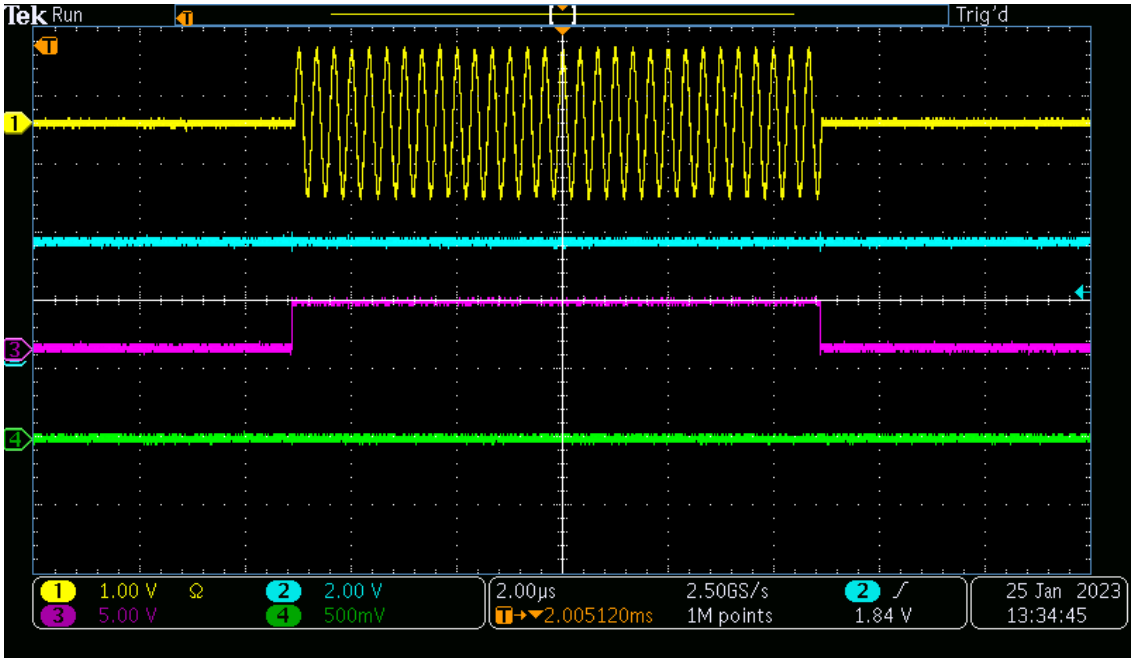


Fig. 64. Señal de radiofrecuencia generada a 3 Mhz con modulación OOK.

Fuente: Elaboración propia.

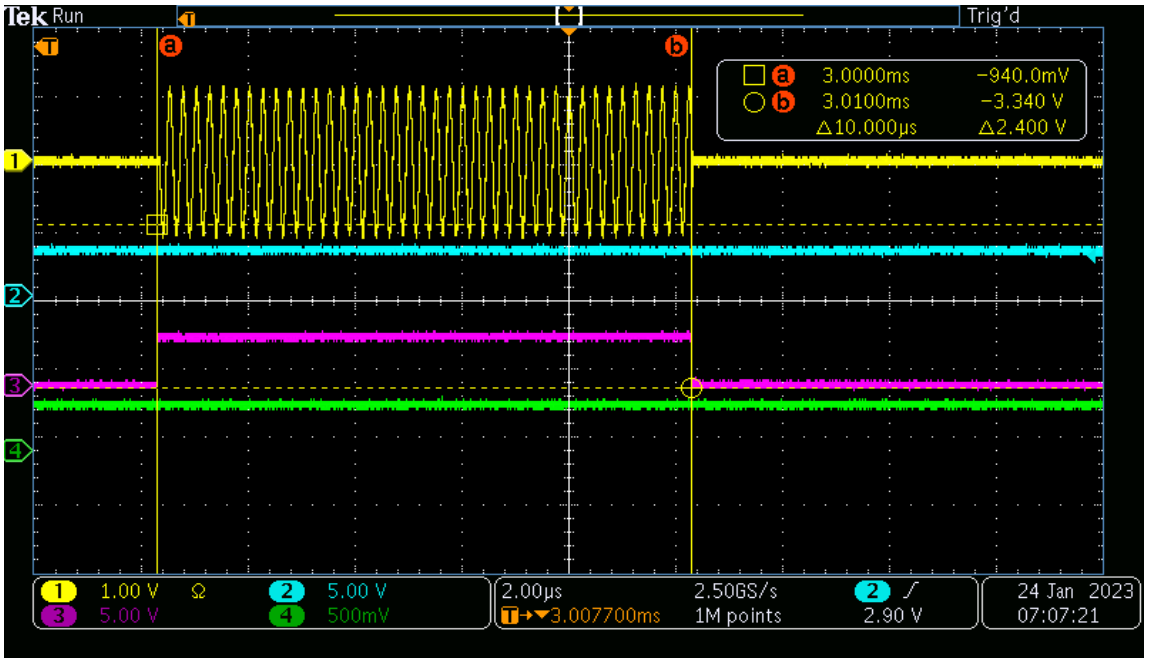


Fig. 65. Señal de radiofrecuencia generada a 3 Mhz con modulación OOK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

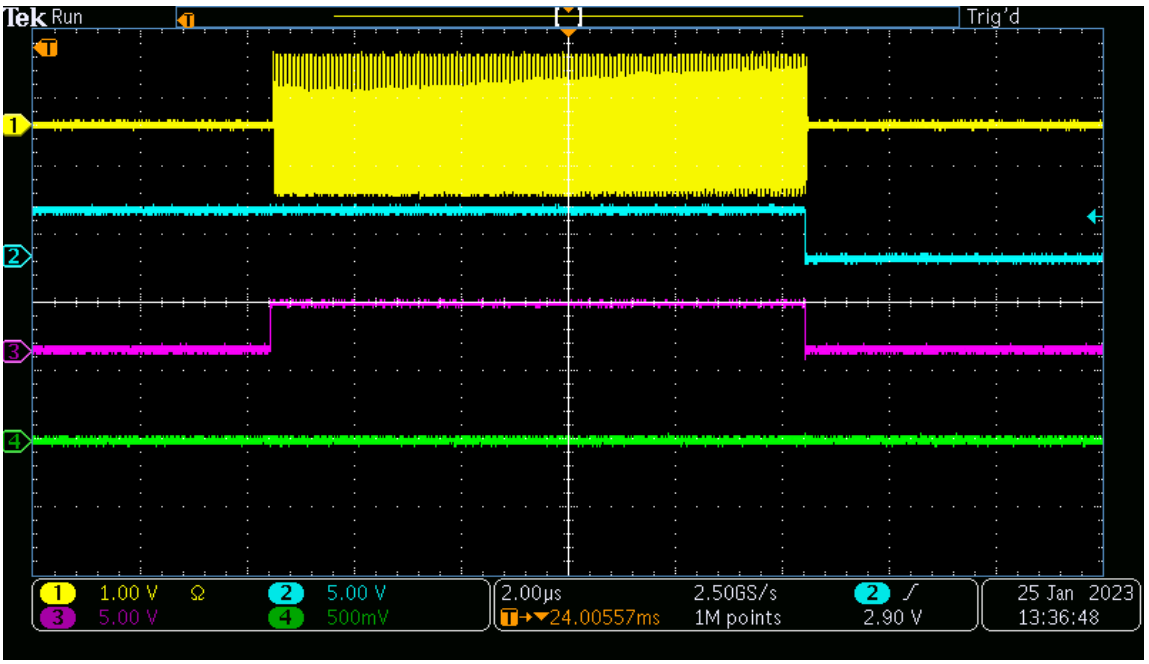


Fig. 66. Señal de radiofrecuencia generada a 25 Mhz con modulación OOK.

Fuente: Elaboración propia.

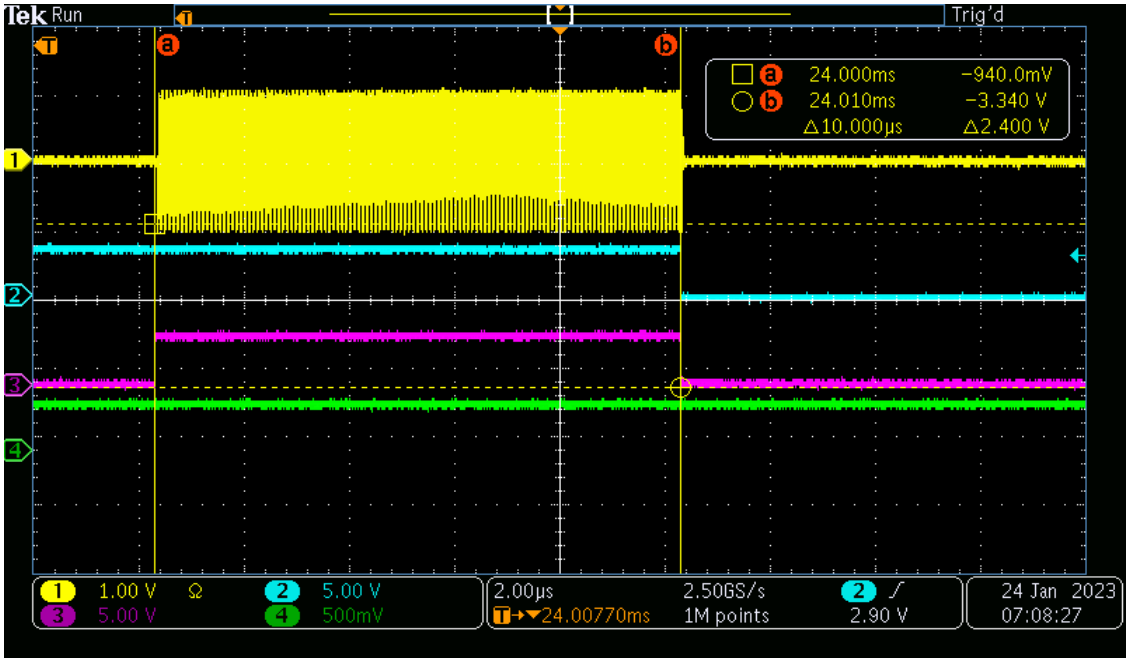


Fig. 67. Señal de radiofrecuencia generada a 25 Mhz con modulación OOK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia.

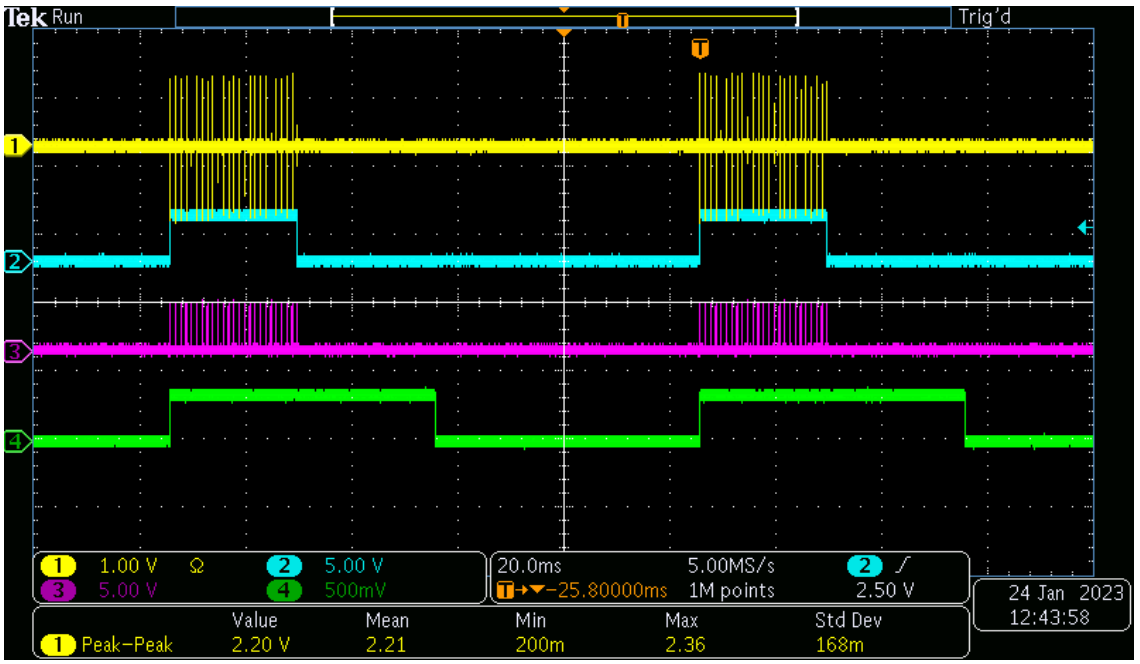


Fig. 68. Barrido de frecuencias de 1 a 25 Mhz modulación OOK.

Fuente: Elaboración propia.

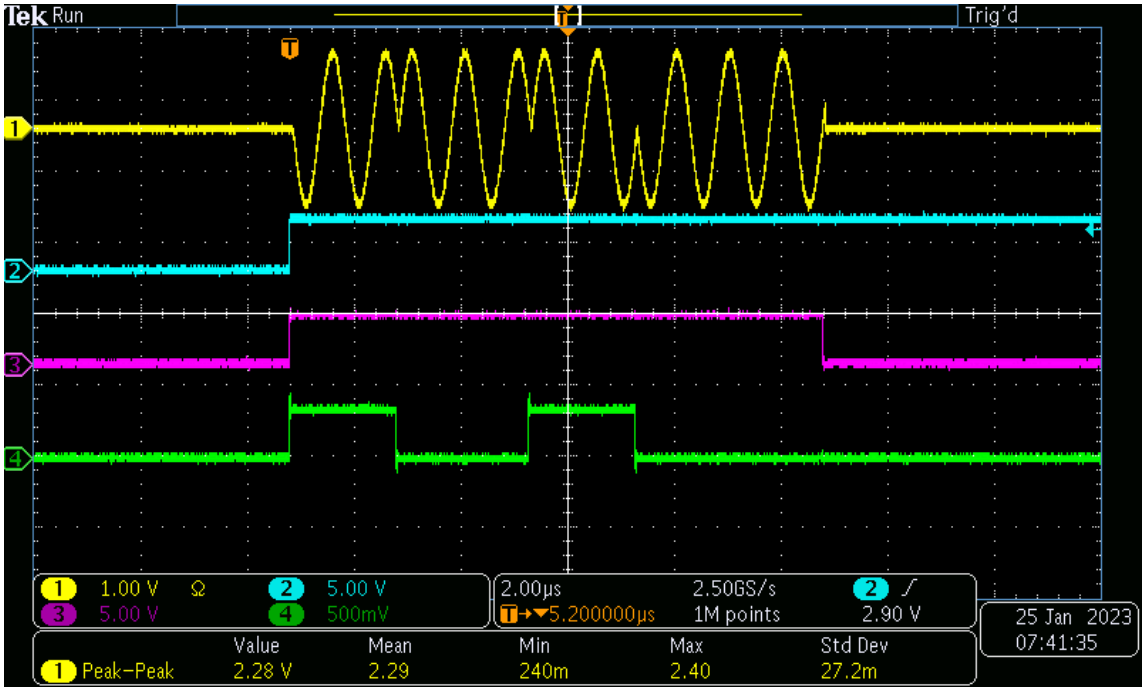


Fig. 69. Señal de radiofrecuencia generada a 1 Mhz con modulación BPSK.

Fuente: Elaboración propia.

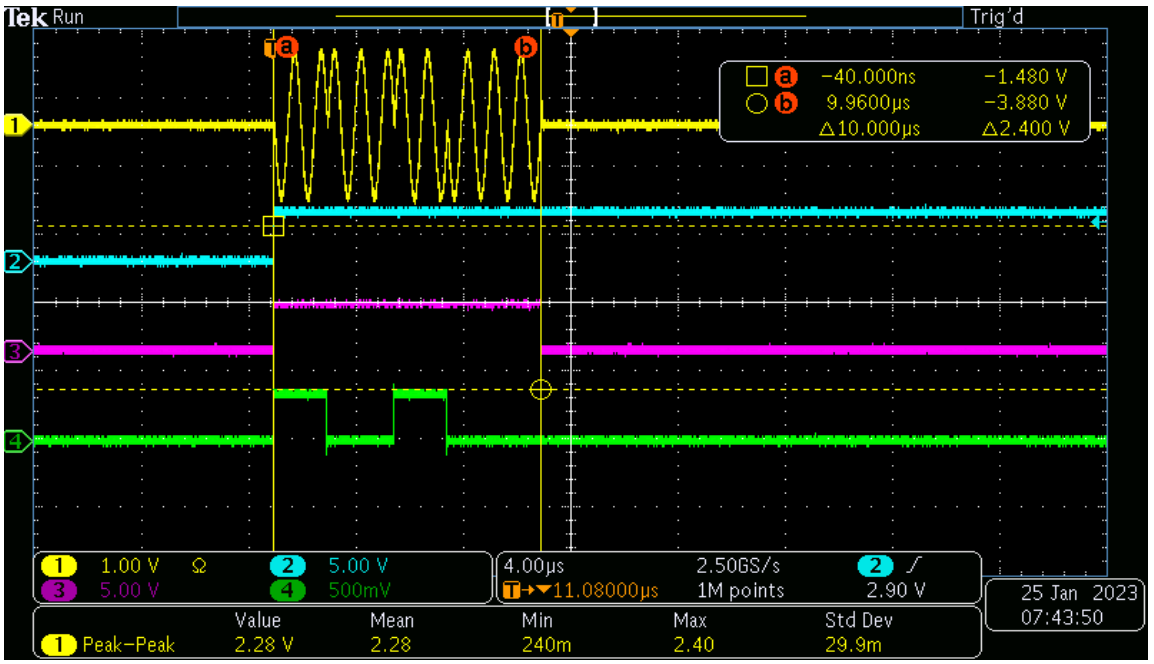


Fig. 70. Señal de radiofrecuencia generada a 1 Mhz con modulación BPSK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

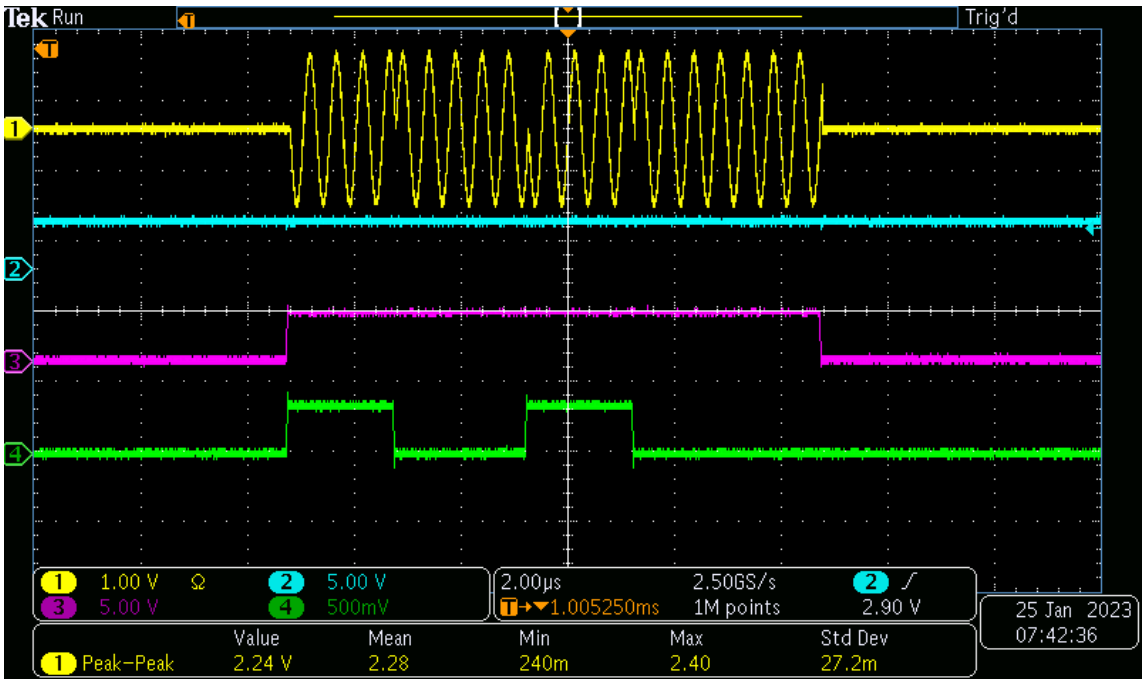


Fig. 71. Señal de radiofrecuencia generada a 2 Mhz con modulación BPSK.

Fuente: Elaboración propia.

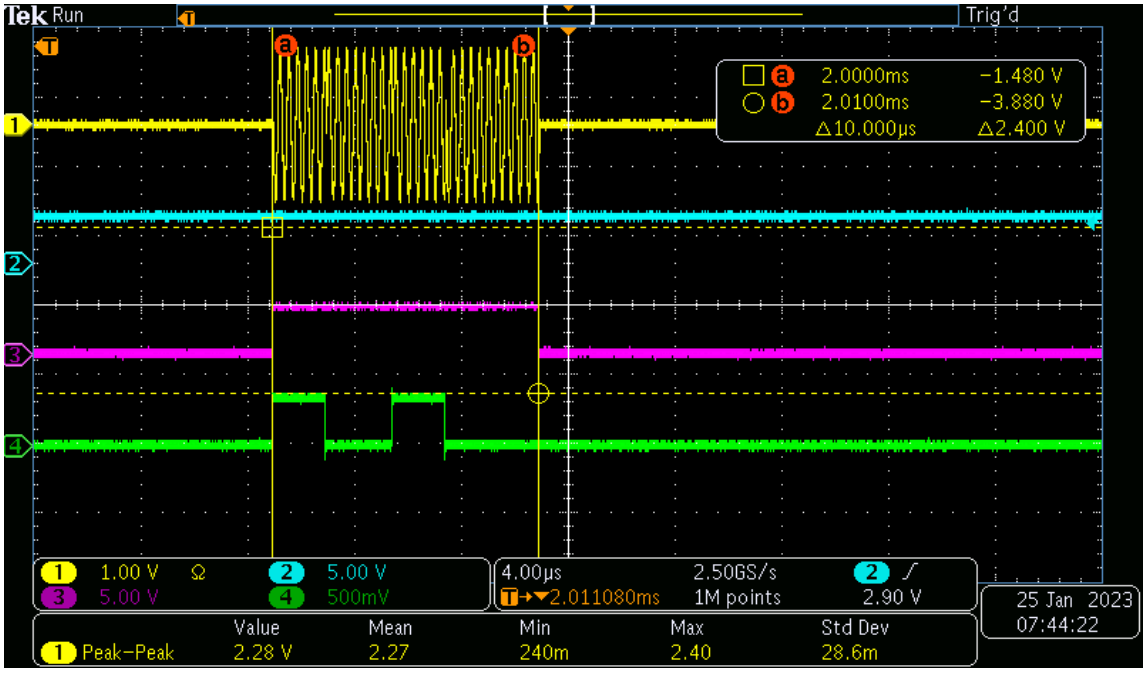


Fig. 72. Señal de radiofrecuencia generada a 2 Mhz con modulación BPSK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

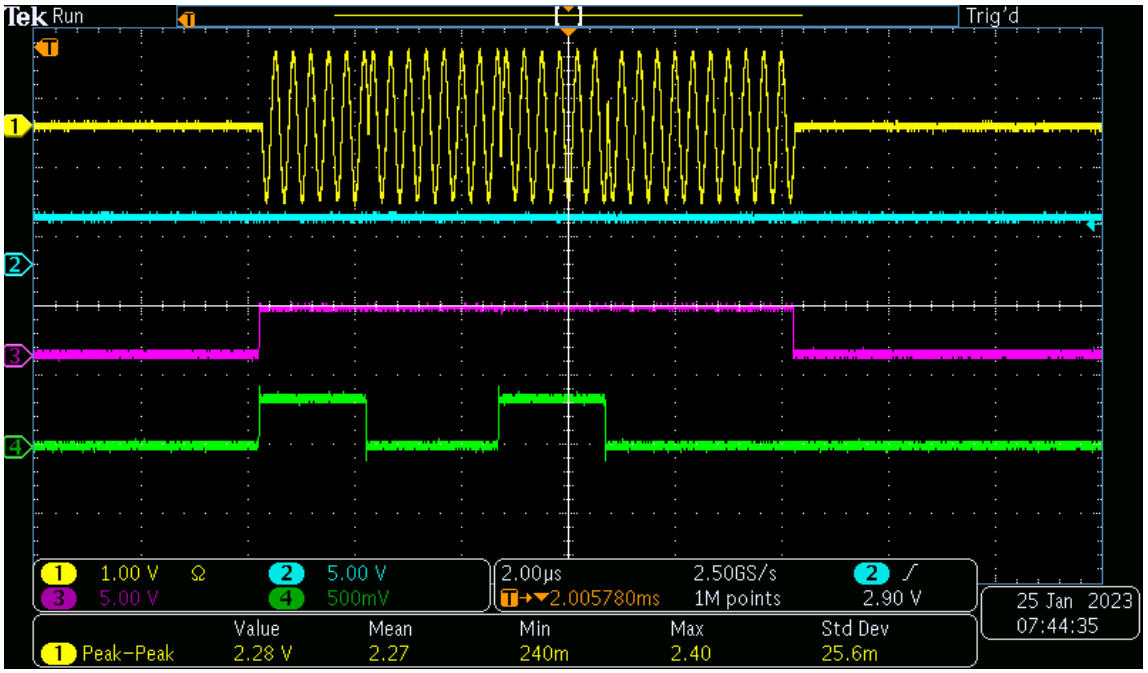


Fig. 73. Señal de radiofrecuencia generada a 3 Mhz con modulación BPSK.

Fuente: Elaboración propia.

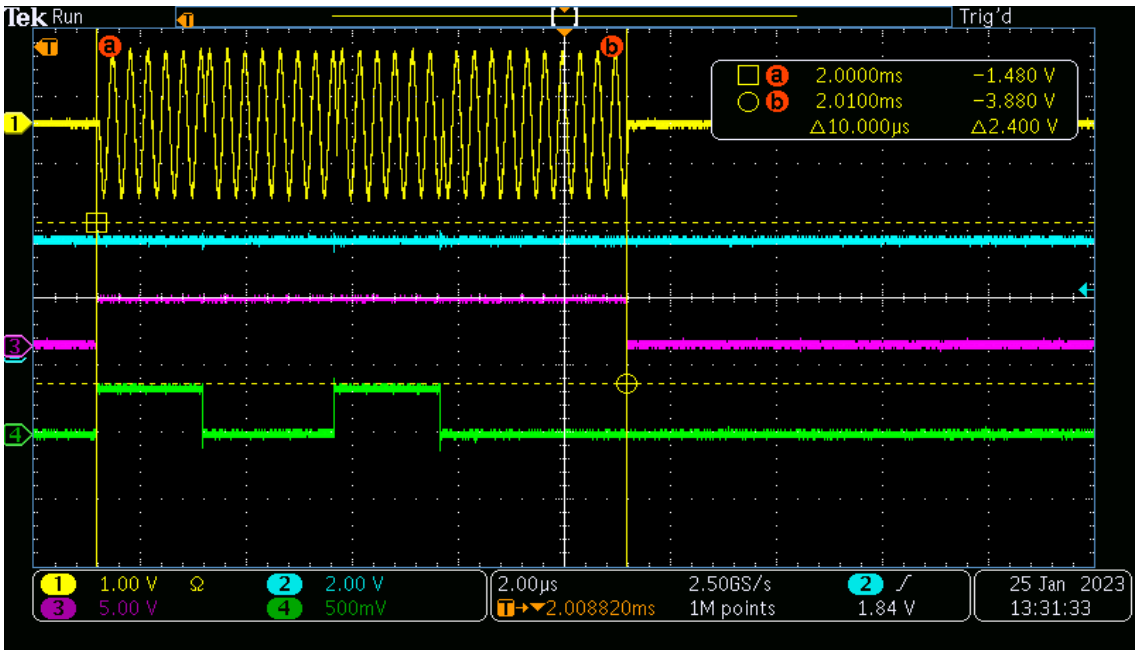


Fig. 74. Señal de radiofrecuencia generada a 3 Mhz con modulación BPSK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia.

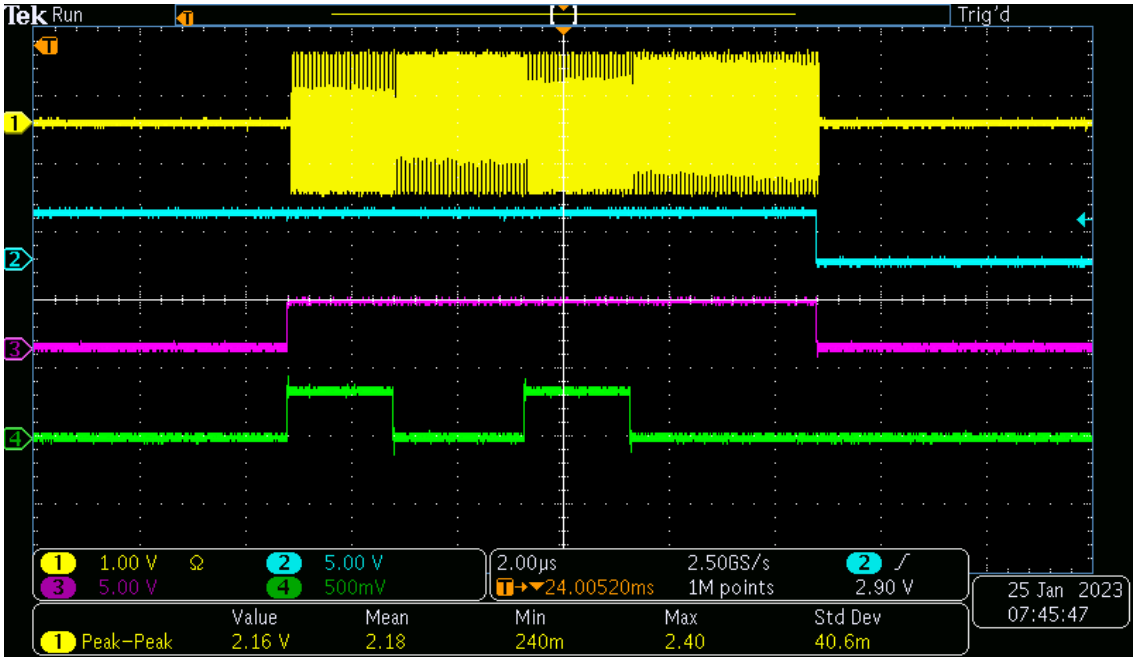


Fig. 75. Señal de radiofrecuencia generada a 25 Mhz con modulación BPSK.

Fuente: Elaboración propia.

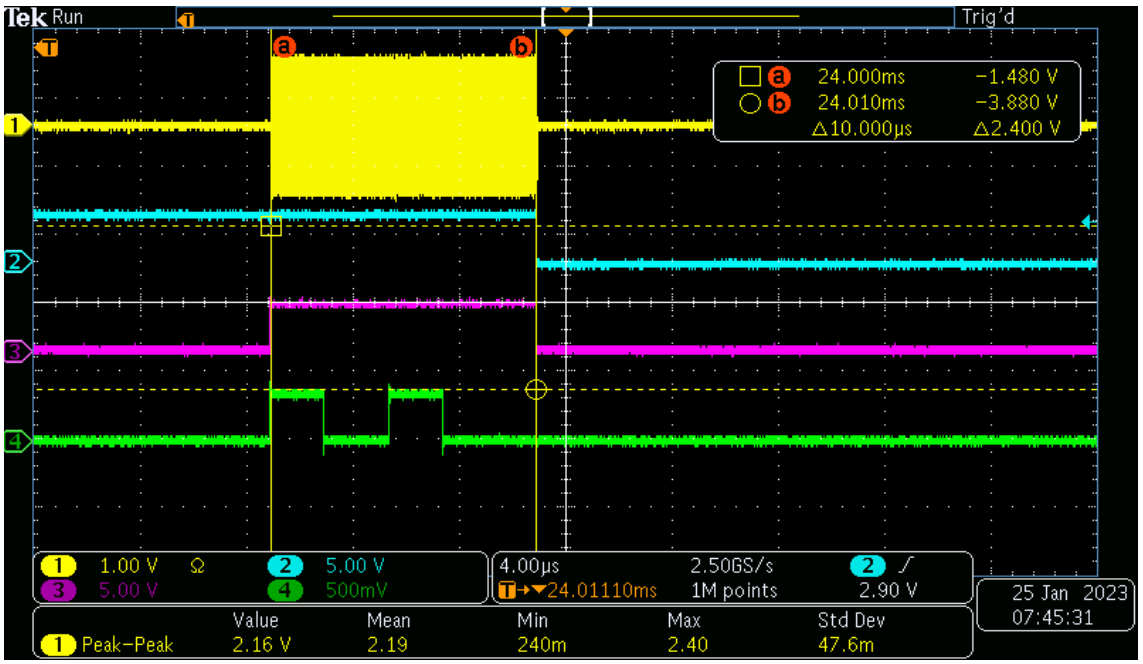


Fig. 76. Señal de radiofrecuencia generada a 25 Mhz con modulación BPSK y ancho de pulso a 10 us medido con curso del osciloscopio.

Fuente: Elaboración propia

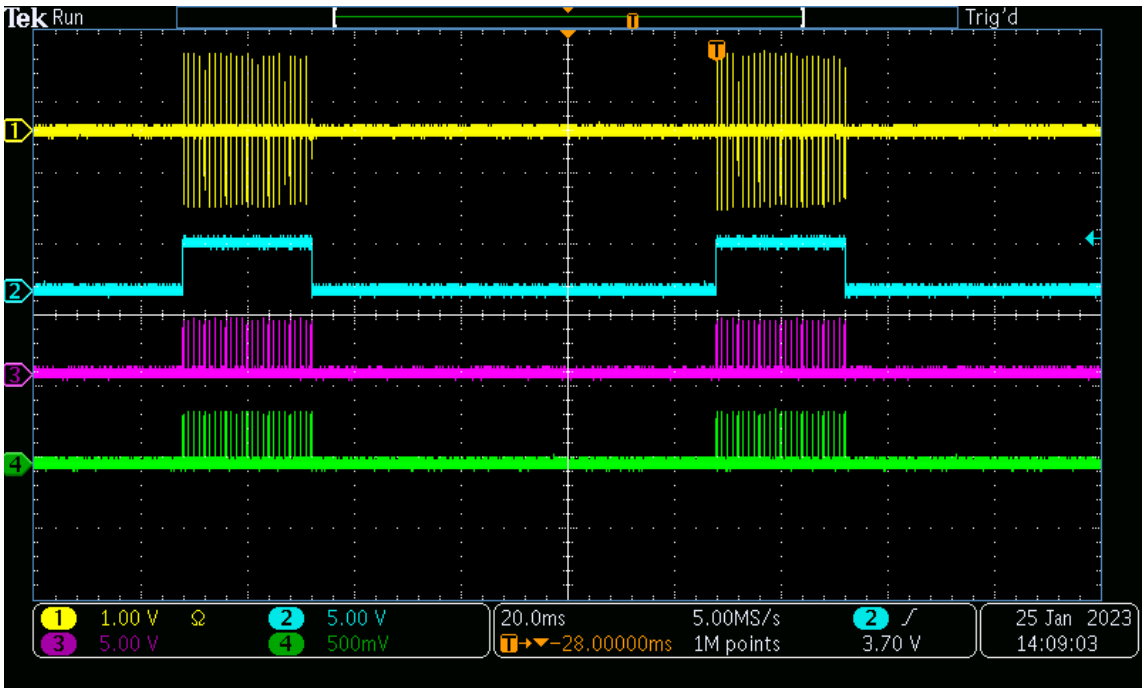


Fig. 77. Barrido de Frecuencias de 1 a 25 Mhz modulación BPSK.

Fuente: Elaboración propia.

### 5.1.5.2. Análisis espectral de la generación de señales de radiofrecuencia

Se configura el generador de señales de radiofrecuencia para que emita la frecuencia de 1.0000010 Mhz, 1.0000015 Mhz, 1.0000016 Mhz, 24.993897 Mhz. La salida de señal RF del generador de radiofrecuencia y la señal de 10 Mhz proveniente del receptor de GPS se conectan al analizador de espectro con la siguiente configuración; frecuencia central igual a la que es emitida la señal RF, span de 10 Khz, ancho de banda (BW) de 10 HZ.

Cabe mencionar que la resolución del generador de señales de RF desarrollado en este trabajo de investigación posee la siguiente resolución en unidades de hertz.

$f_{clk}$ = Frecuencia de clock del FPGA *(250Mhz)*
$f_{out}$ = Frecuencia de la onda senoidal
$\Delta\theta$ = Incremento de fase
$2^{B_{\theta(n)}}$ = Ancho de fase

$$res = \frac{f_{clk} x \Delta\theta}{2^{B_{\theta(n)}}}$$

$$res = \frac{250_{Mhz} x 1}{2^{64}}$$

$$res = 0.00000000000135525272$$

Dando la posibilidad de generar frecuencias con un mayor rango de decimales y por ende tener pasos más cortos entre frecuencias para el barrido de frecuencias.
En las siguientes figuras se verifica la correcta generación de señales captadas por el analizador de espectro.

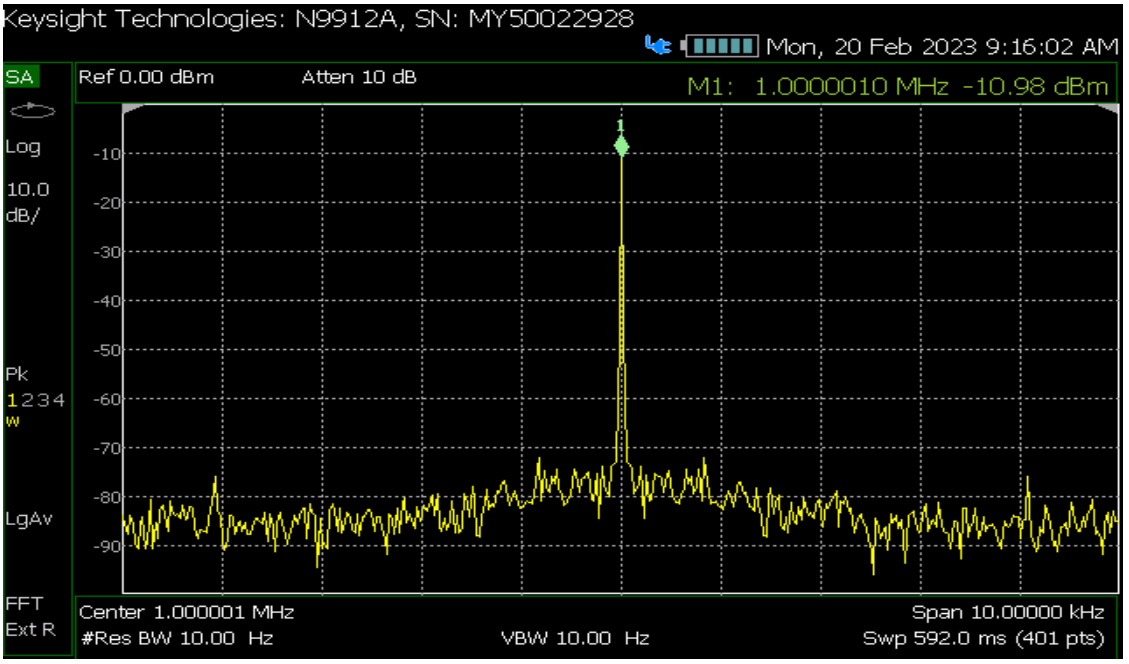


Fig. 78. Verificación de la generación de señal de radiofrecuencia a 1.0000010 Mhz.

Fuente: Elaboración propia.

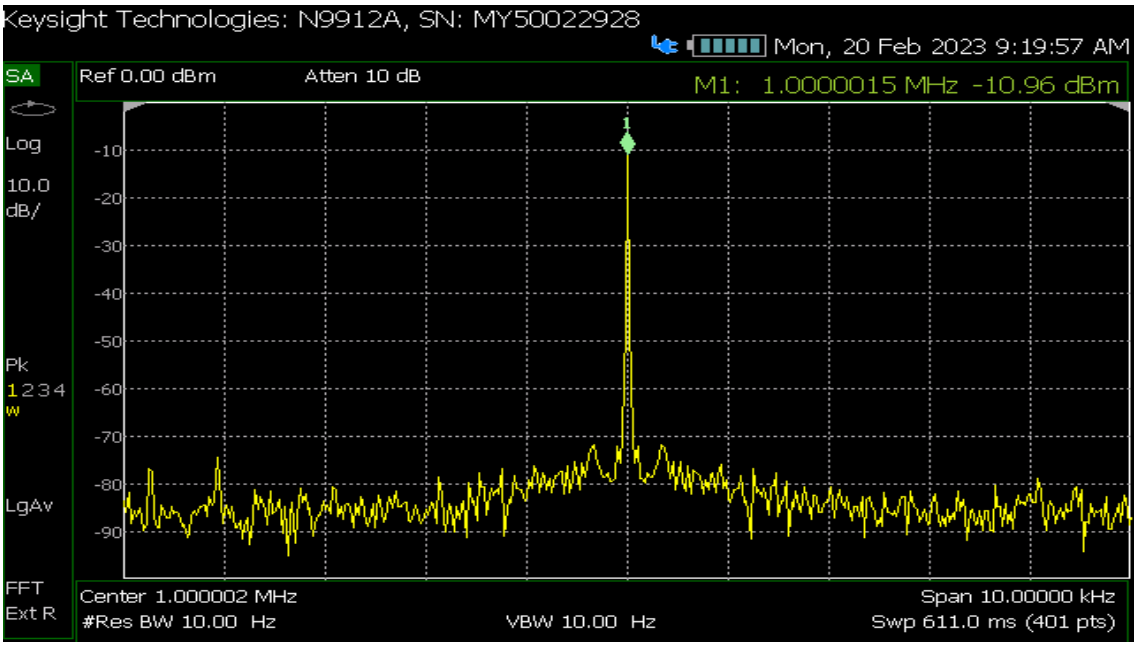


Fig. 79. Verificación de la generación de señal de radiofrecuencia a 1.0000015 Mhz.

Fuente: Elaboración propia.

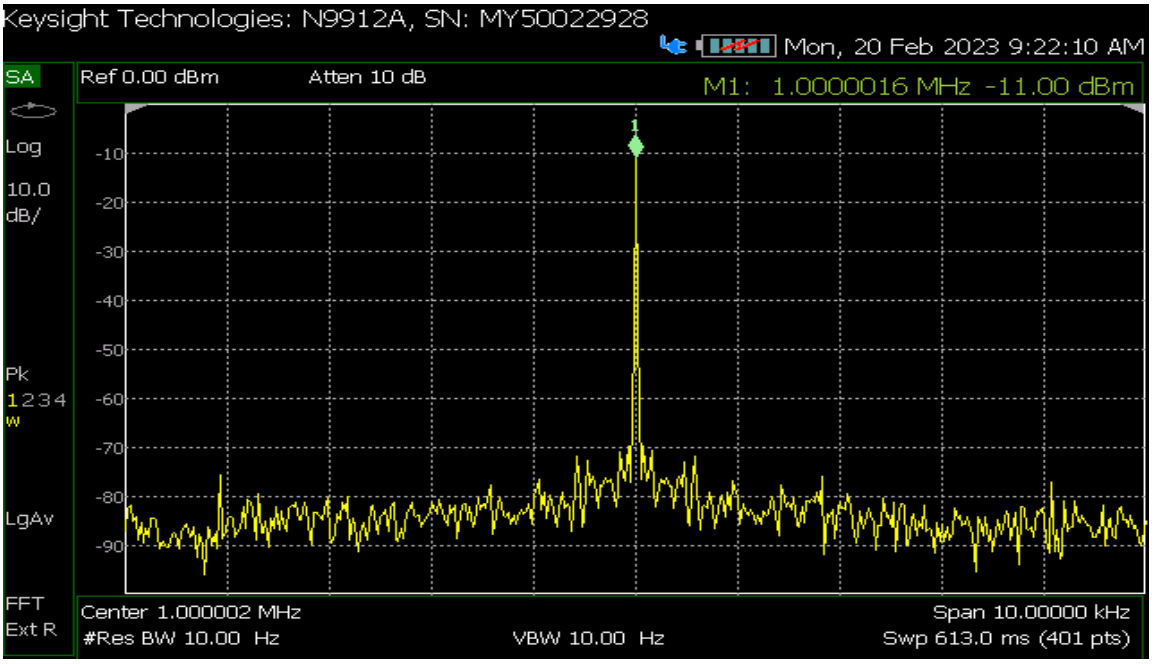


Fig. 80. Verificación de la generación de señal de radiofrecuencia a 1.0000016 Mhz.

Fuente: Elaboración propia

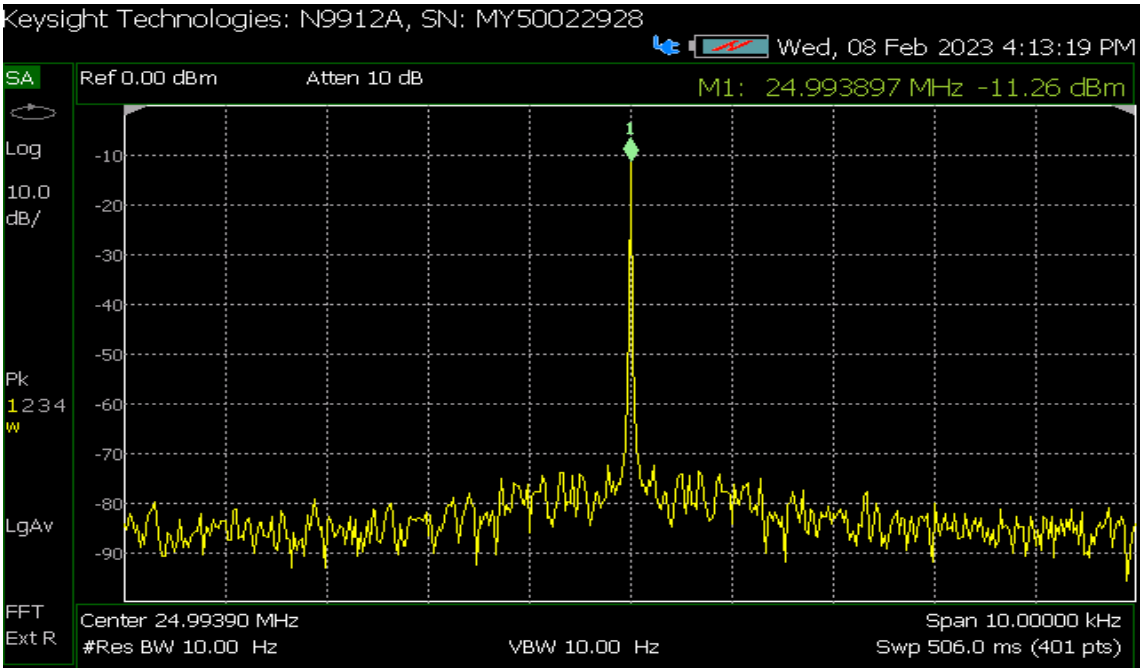


Fig. 81. Verificación de la generación de señal de radiofrecuencia a 24.993897 Mhz.

Fuente: Elaboración propia.

### 5.1.6. Resultados de Prueba Bola de Cobre

En esta etapa comprende del envió de las señales de RF atenuadas a un voltaje pico pico de 760 mv por medio de cables coaxiales hacia un receptor ionosférico para este caso se utilizó el receptor “Ionospheric Echoes Receiver” *(IER)* basado en USPR.

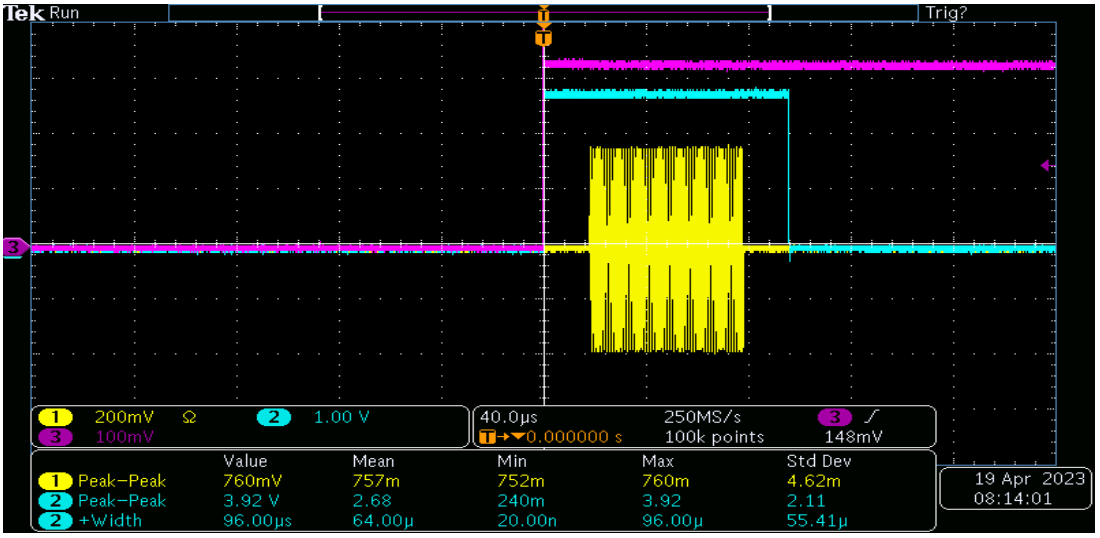


Fig. 82. Señal RF atenuada a 760 mv.

Fuente: Elaboración propia.

Se realizó la prueba bola de cobre y a continuación se muestran las capturas de recepción de las 1808 frecuencias.

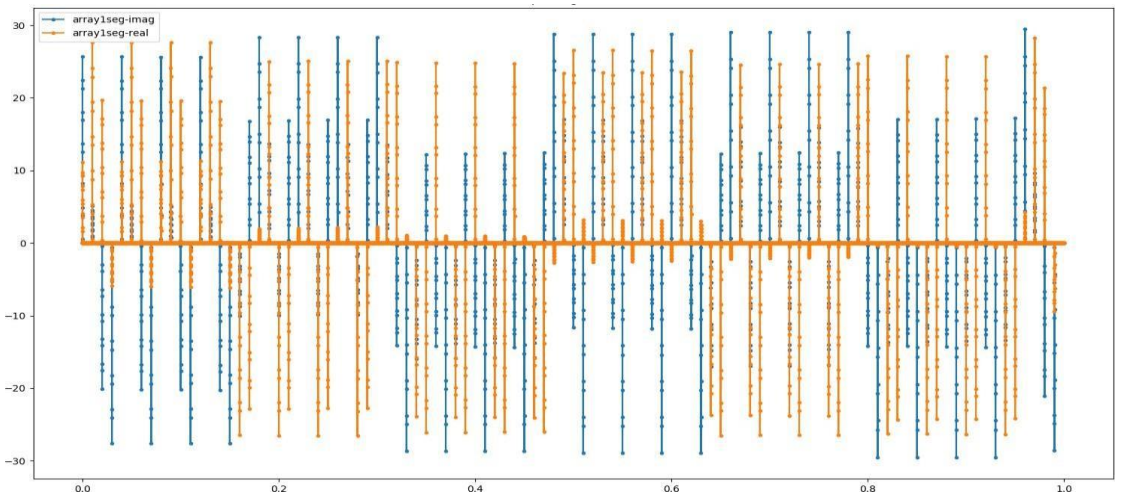


Fig. 83. Adquisición de señales RF en el segundo 1 con el receptor IER.

Fuente: Elaboración propia.

El generador de señales se configuró para enviar cada 4 frecuencias que se repiten 4 veces, en la figura 84 se observa los trazos de color azul es la parte imaginaria de las señales y los de color naranja es la parte real de la señales. además se verifica que cada frecuencia siempre tiene la misma fase.

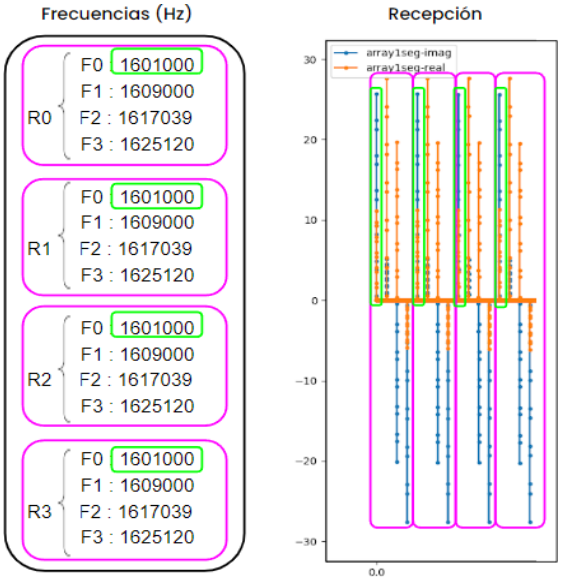


Fig. 84. Análisis del barrido de frecuencias del segundo 1.

Fuente: Elaboración propia.

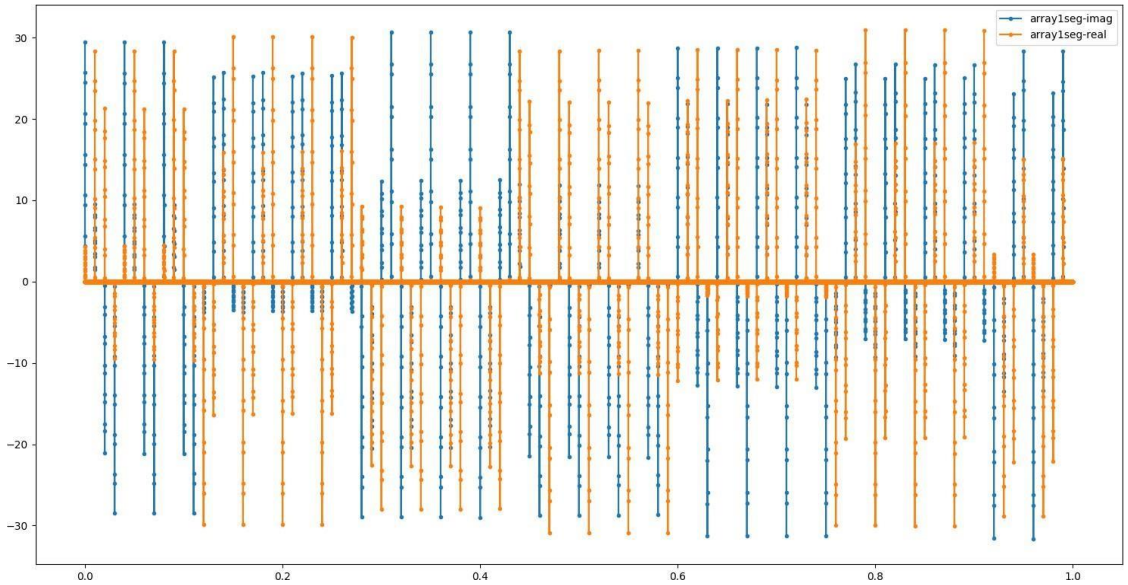


Fig. 85. Adquisición de señales RF en el segundo 2 con el receptor IER.

Fuente: Elaboración propia.

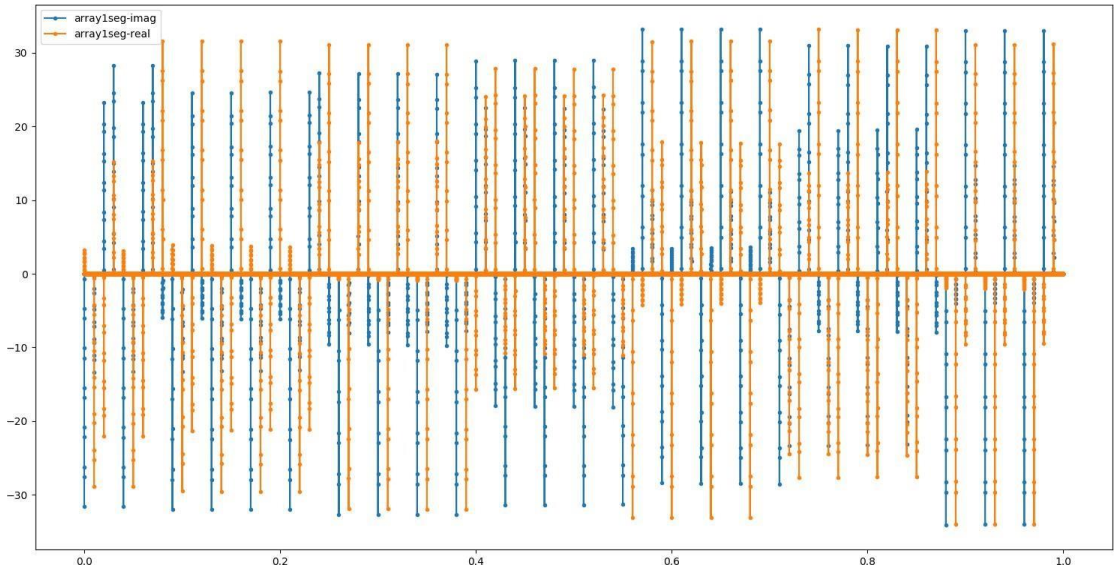


Fig. 86. Adquisición de señales RF en el segundo 3 con el receptor IER.

Fuente: Elaboración propia.

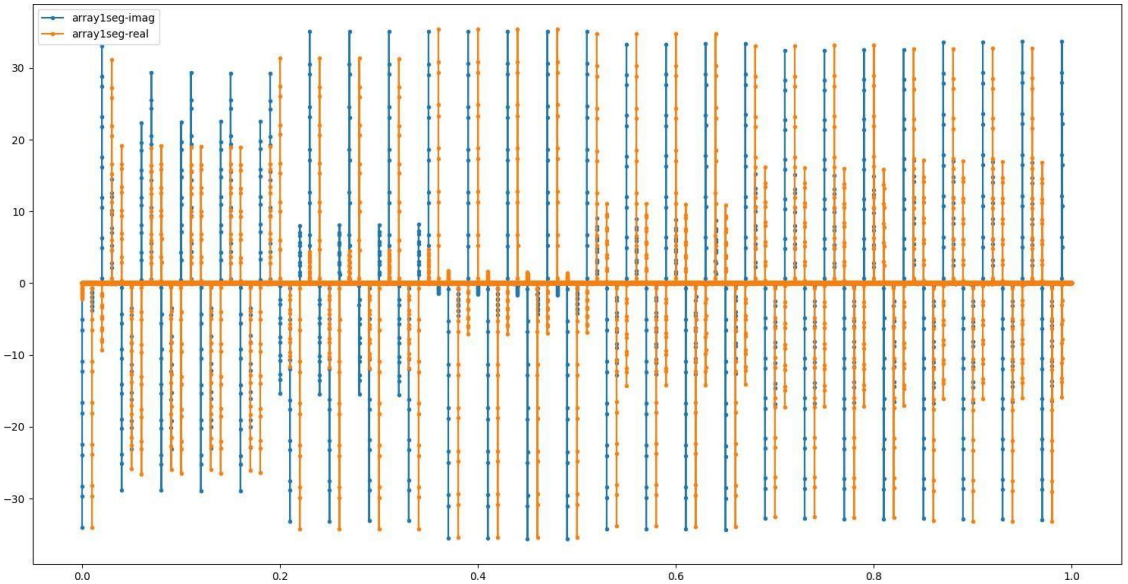


Fig. 87. Adquisición de señales RF en el segundo 4 con el receptor IER.

Fuente: Elaboración propia.

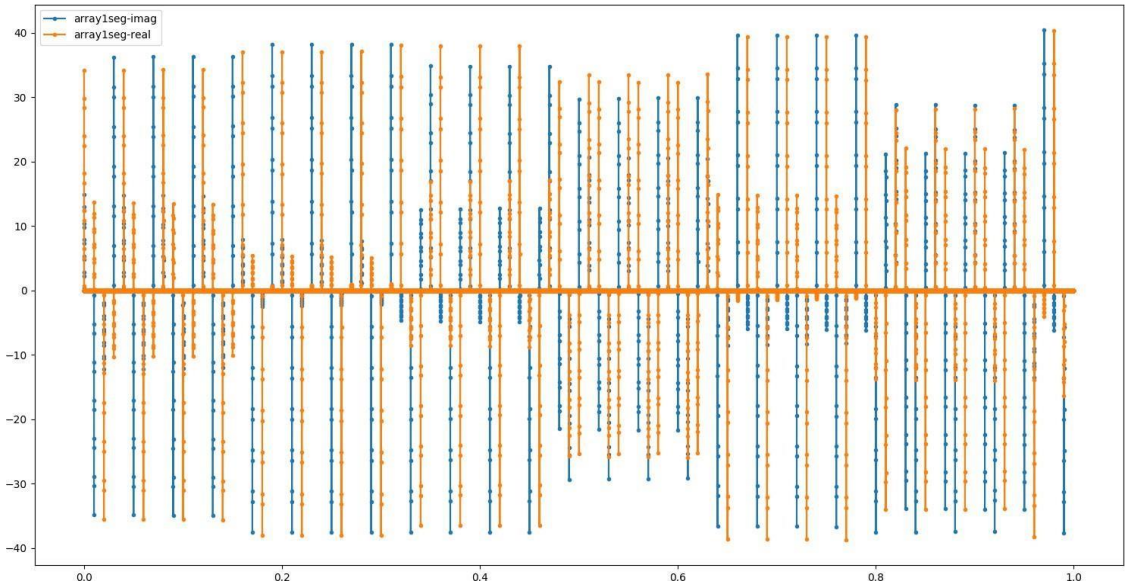


Fig. 88. Adquisición de señales RF en el segundo 5 con el receptor IER.

Fuente: Elaboración propia.

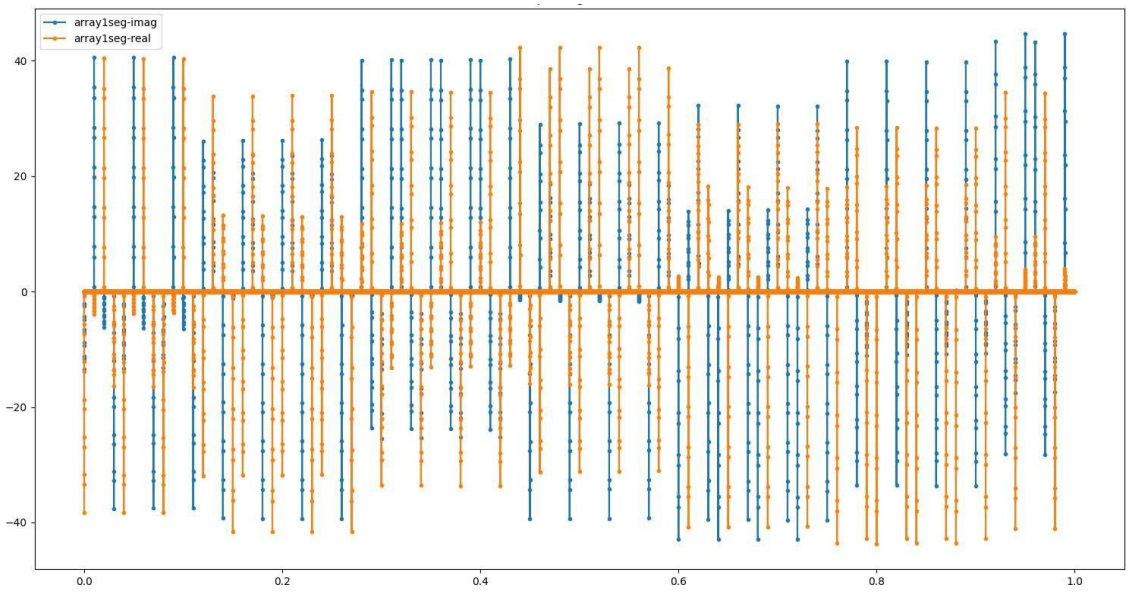


Fig. 89 Adquisición de señales RF en el segundo 6 con el receptor IER.

Fuente: Elaboración propia.

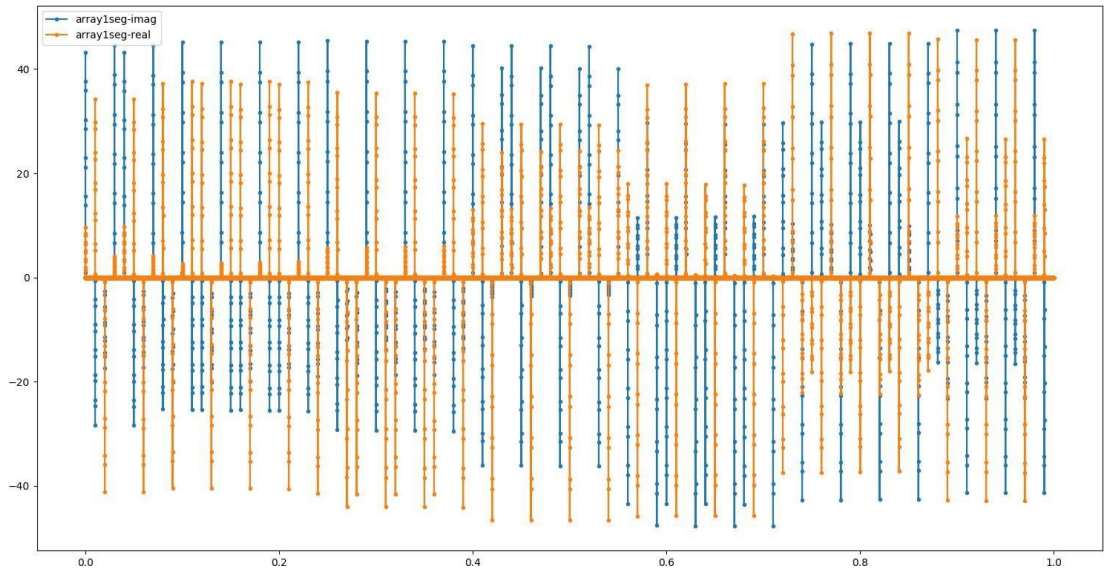


Fig. 90. Adquisición de señales RF en el segundo 7 con el receptor IER.

Fuente: Elaboración propia.

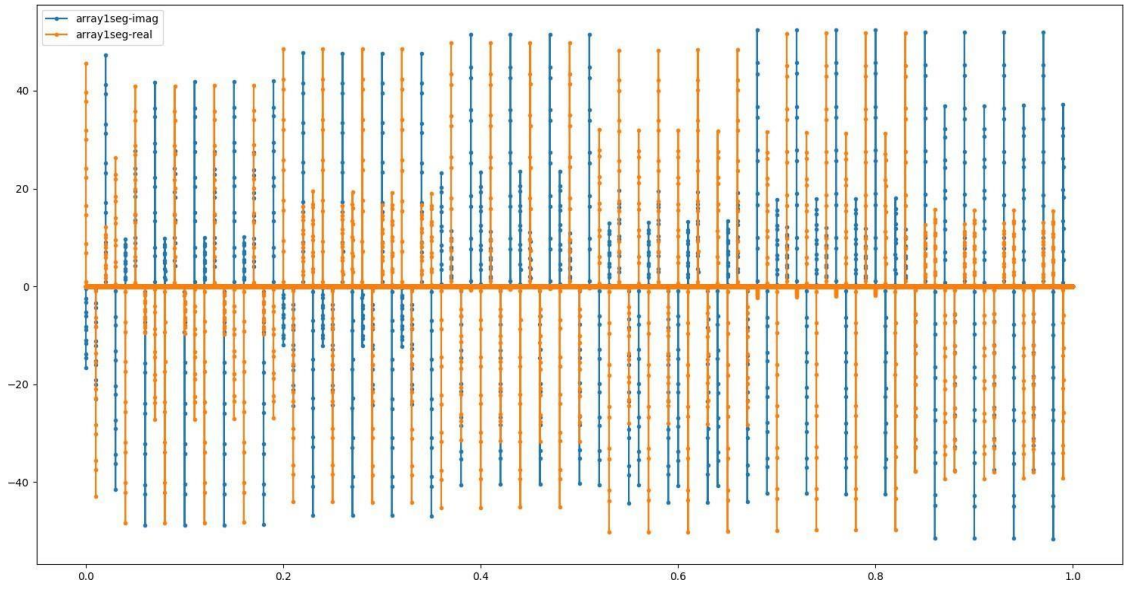


Fig. 91. Adquisición de señales RF en el segundo 8 con el receptor IER.

Fuente: Elaboración propia.

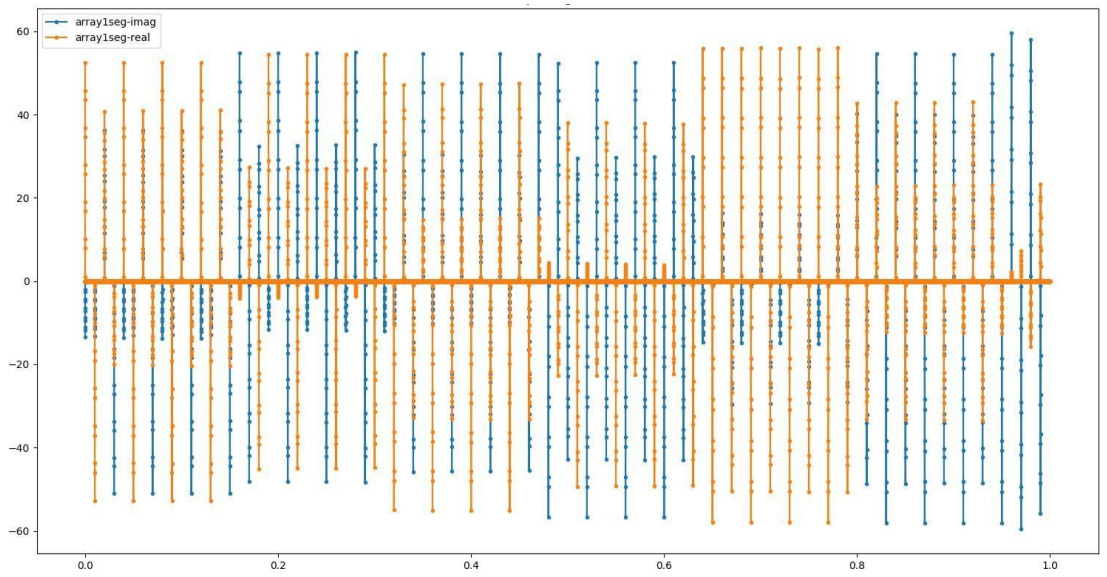


Fig. 92. Adquisición de señales RF en el segundo 9 con el receptor IER.

Fuente: Elaboración propia.

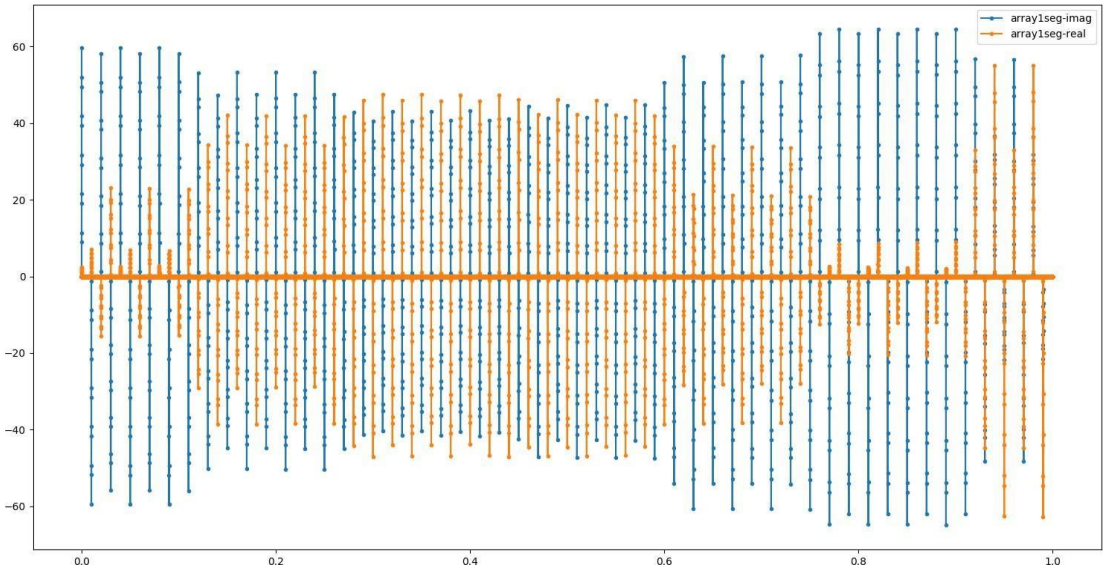


Fig. 93. Adquisición de señales RF en el segundo 10 con el receptor IER.

Fuente: Elaboración propia.

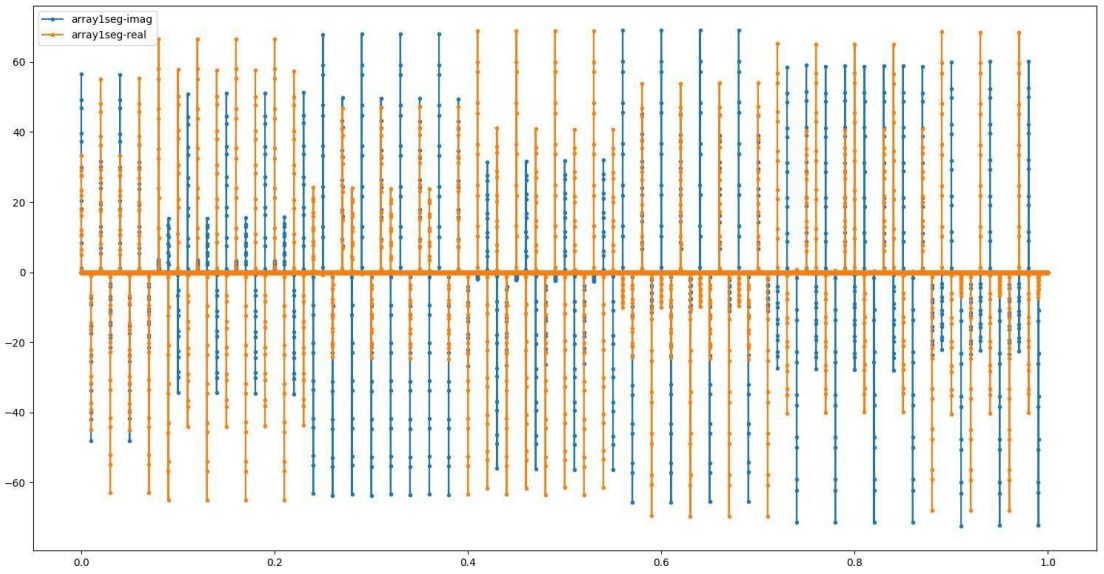


Fig. 94 Adquisición de señales RF en el segundo 11 con el receptor IER.

Fuente: Elaboración propia.

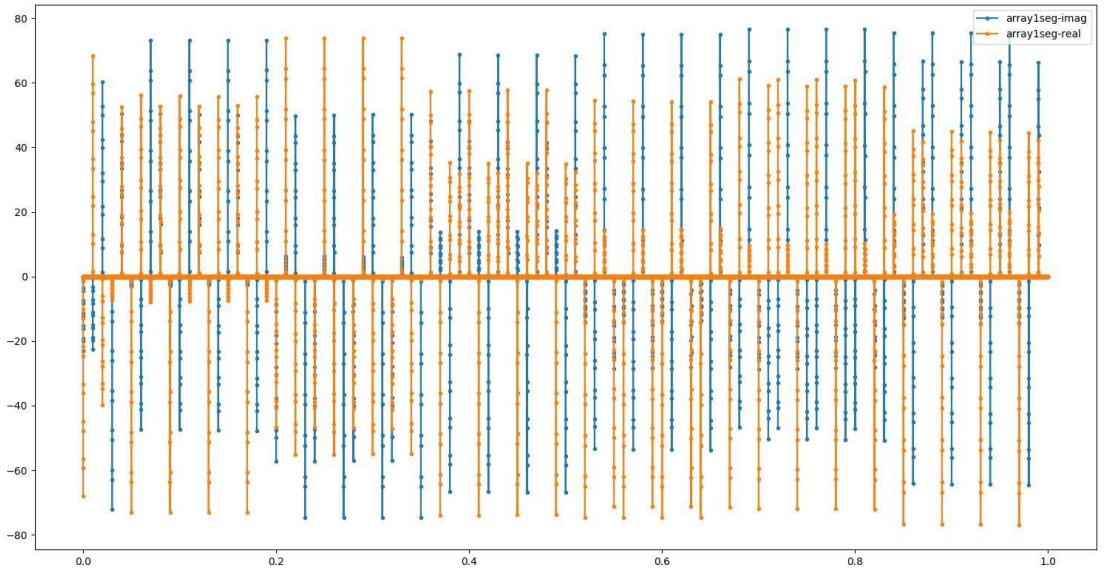


Fig. 95. Adquisición de señales RF en el segundo 12 con el receptor IER.

Fuente: Elaboración propia.

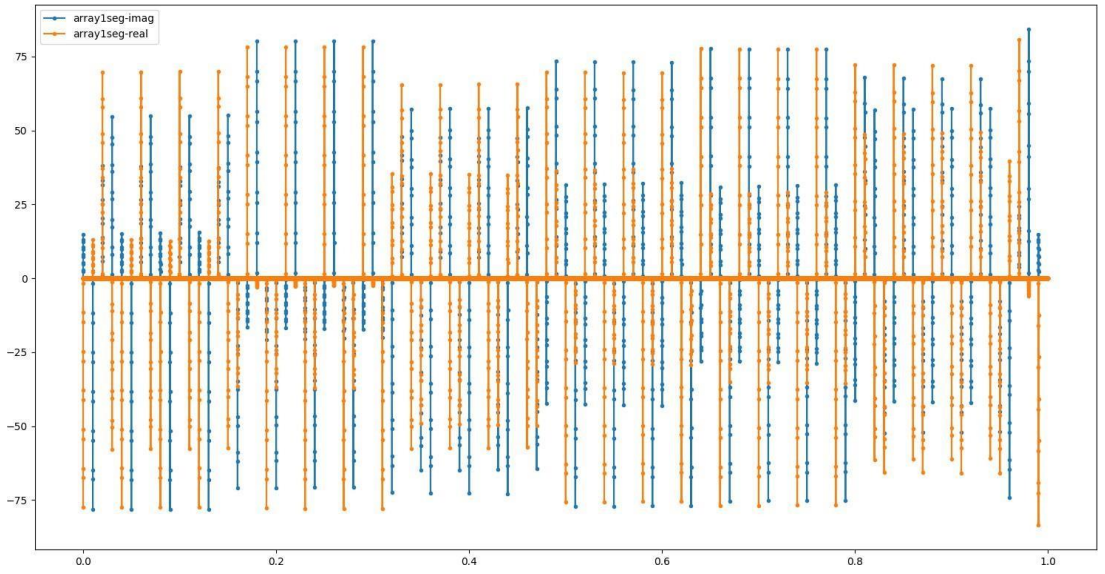


Fig. 96. Adquisición de señales RF en el segundo 13 con el receptor IER.

Fuente: Elaboración propia.

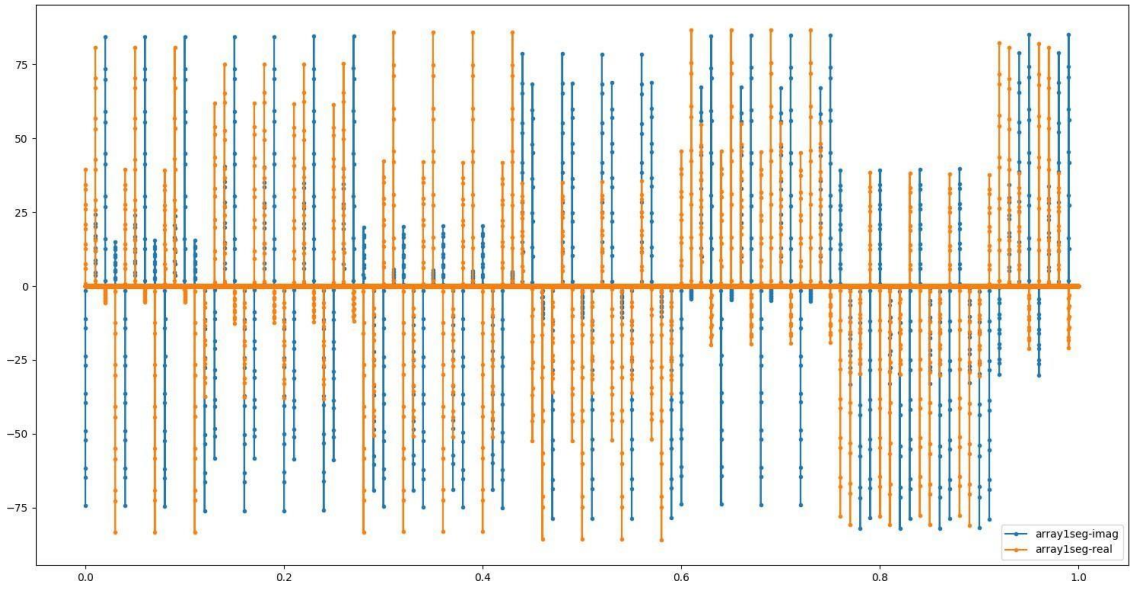


Fig. 97. Adquisición de señales RF en el segundo 14 con el receptor IER.

Fuente: Elaboración propia.

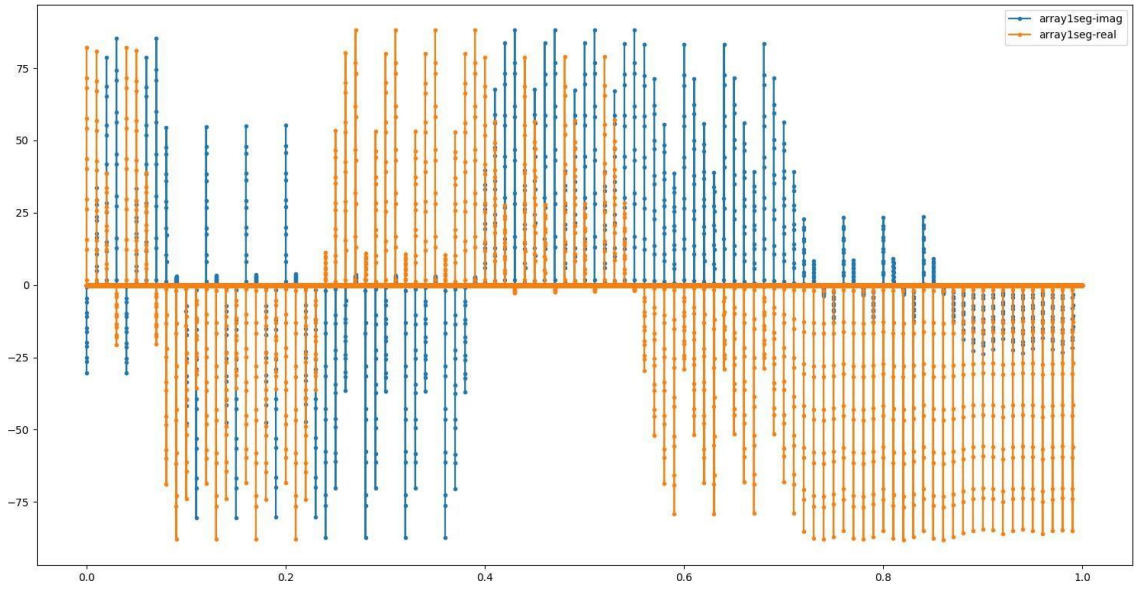


Fig. 98. Adquisición de señales RF en el segundo 15 con el receptor IER.

Fuente: Elaboración propia.

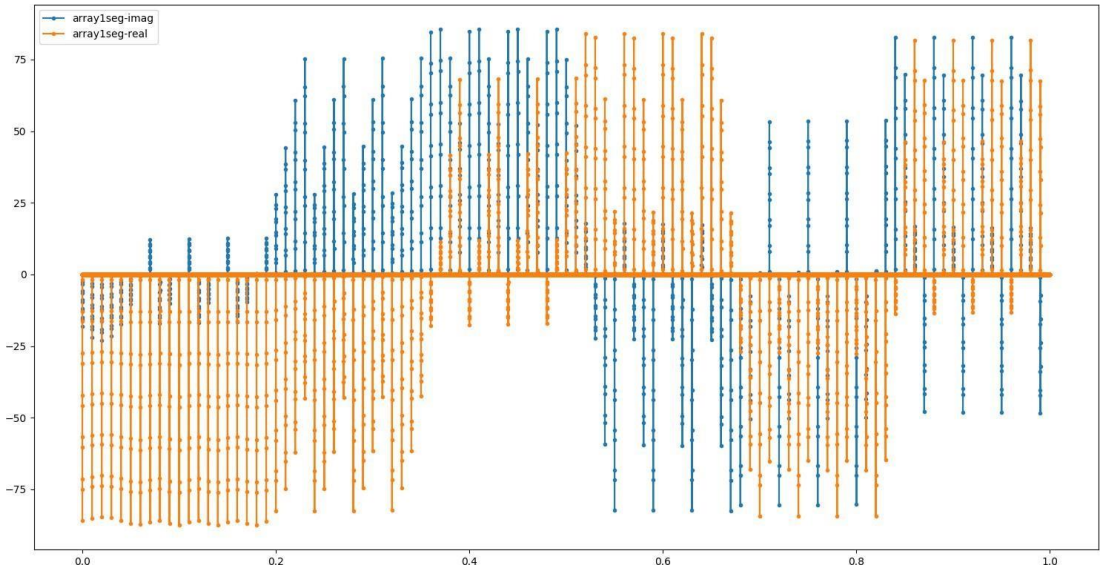


Fig. 99. Adquisición de señales RF en el segundo 16 con el receptor IER.

Fuente: Elaboración propia.

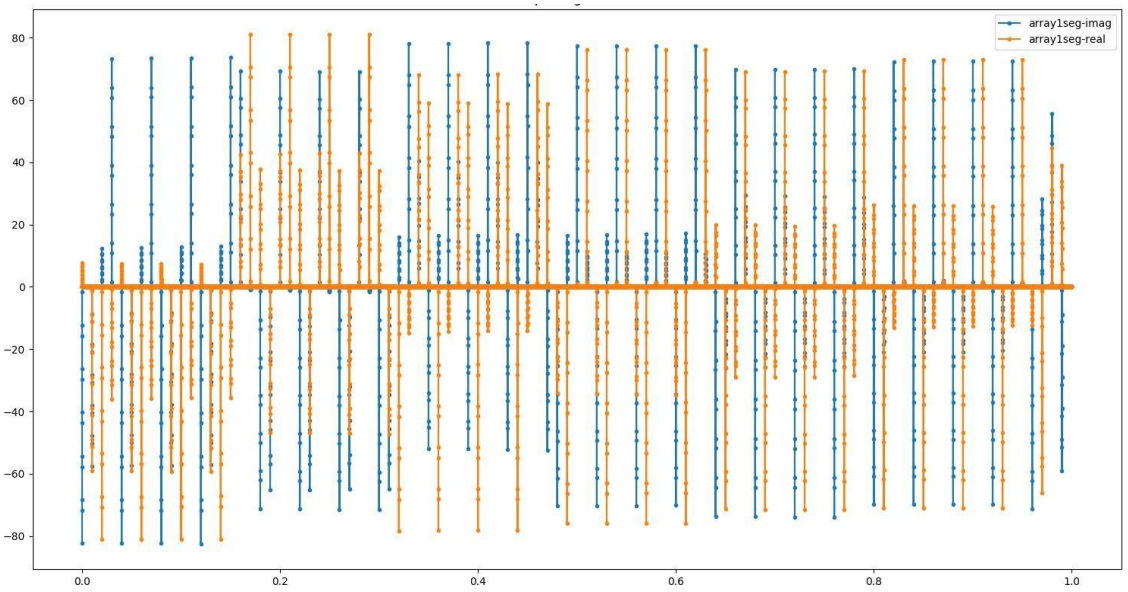


Fig. 100. Adquisición de señales RF en el segundo 17 con el receptor IER.

Fuente: Elaboración propia.

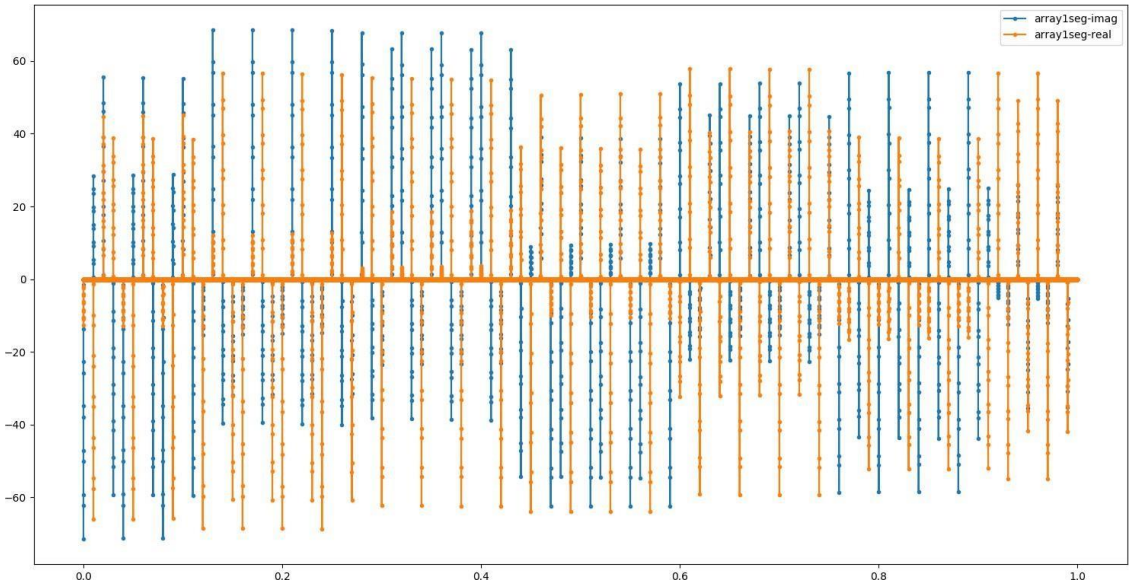


Fig. 101. Adquisición de señales RF en el segundo 18 con el receptor IER.

Fuente: Elaboración propia.

Las imágenes que muestran la adquisición del segundo 1 al 18 cada una de ellas adquieren 100 frecuencias respectivamente.

En el segundo 19 se adquieren las últimas 8 frecuencias restantes del barrido de 1808 frecuencias.

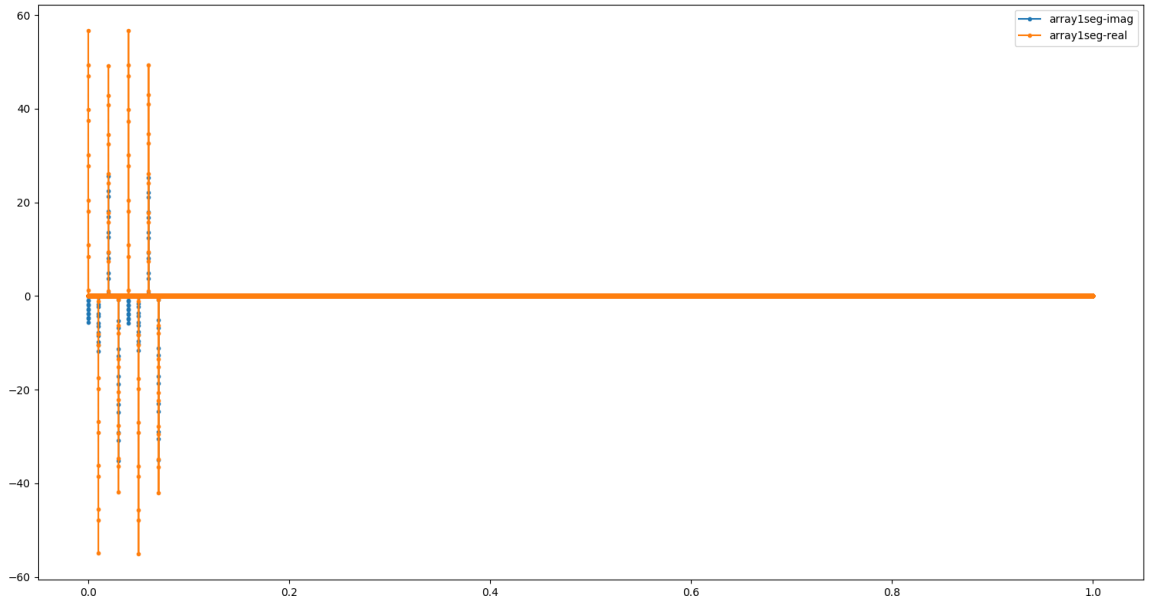


Fig. 102. Adquisición de señales RF en el segundo 19 con el receptor IER.

Fuente: Elaboración propia.

#### 5.1.6.1. Ionograma de la recepción señales en la prueba bola de cobre

Los datos obtenidos de la prueba de bola de cobre se utilizaron para la generación del ionograma en la figura 103, donde el eje vertical hace referencia a las alturas virtuales que van desde 0 Km a 140 Km y el eje horizontal hace referencia al barrido de frecuencias de 1Mhz a 15 MHZ.

Cabe mencionar que el barrido de frecuencias enviadas se grafican en la altura 0 Km (amarillo) debido a que las señales de radiofrecuencia no salieron a las capas altas de la atmósfera.

.

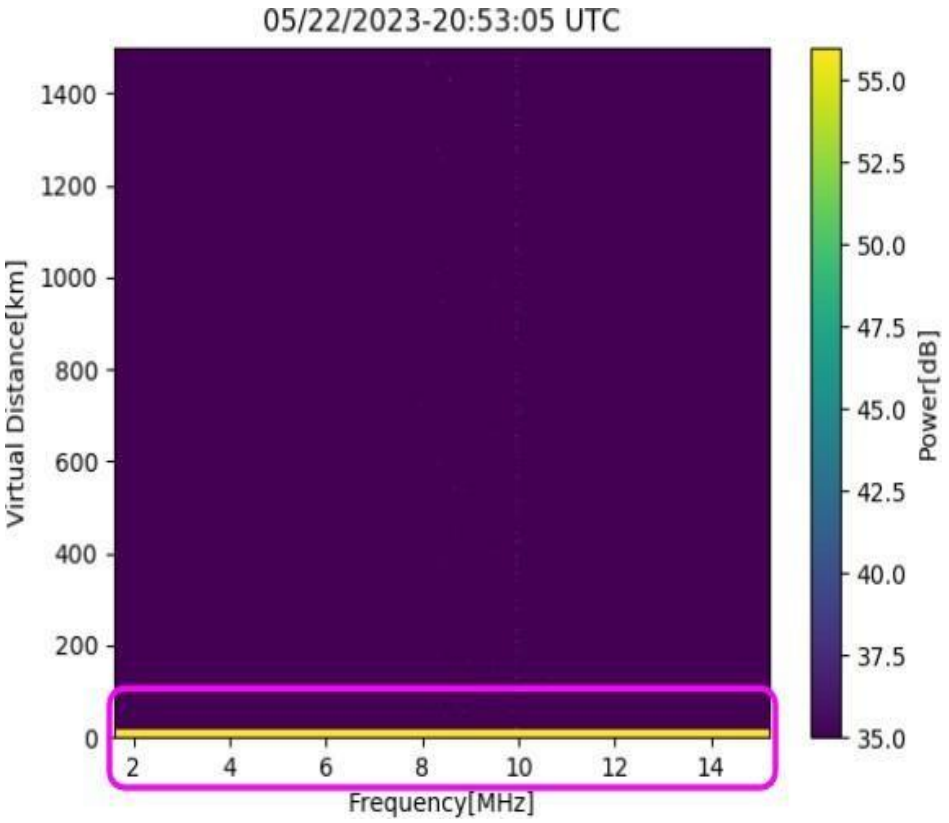


Fig. 103. Ionograma con datos de bola de cobre.

Fuente: Elaboración propia.

### 5.1.7. Operación del Generador de RF con transmisor de 4KW

El generador de señales de radiofrecuencia envía el barrido de frecuencias de 1 a 15 Mhz hacía el transmisor de alta potencia 4 KW y posteriormente mediante la antena logarítmica periódica de banda ancha transmite pulsos de ondas de radio que se propagan.

Cabe mencionar que se configura el generador de RF para que emita una señal de 10v pico-pico y luego se atenúa a una señal de 8.48v pico-pico antes de conectar al transmisor de alta potencia de 4 KW. En la figura 104 se muestran las señales para operar el transmisor y son las siguientes; la señal RF (Amarilla), señal de GATE (Jade), señal de WINDOW (Lila).

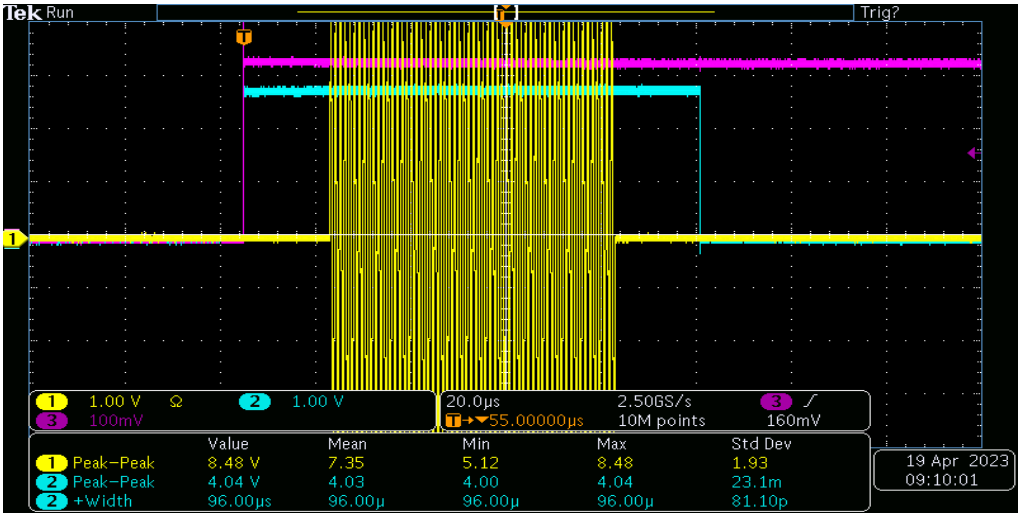


Fig. 104. Señal RF atenuada a 8.48V pico pico.

Fuente: Elaboración propia.

El generador de señales de radiofrecuencia al operar el transmisor del radar ionosonda “Vertical Incidence Pulsed Ionospheric Radar” (VIPIR) permite realizar estudios de Clima espacial en el Perú tales como estudios de sondeo vertical ionosféricos con el receptor VIPIR ya que está ubicado en el Radio Observatorio de Jicamarca y estudios de sondeo oblicuo ionosféricos con el receptor “Ionospheric Echoes Receiver” *(IER)* basado en USPR que está ubicado en la ciudad PIURA.

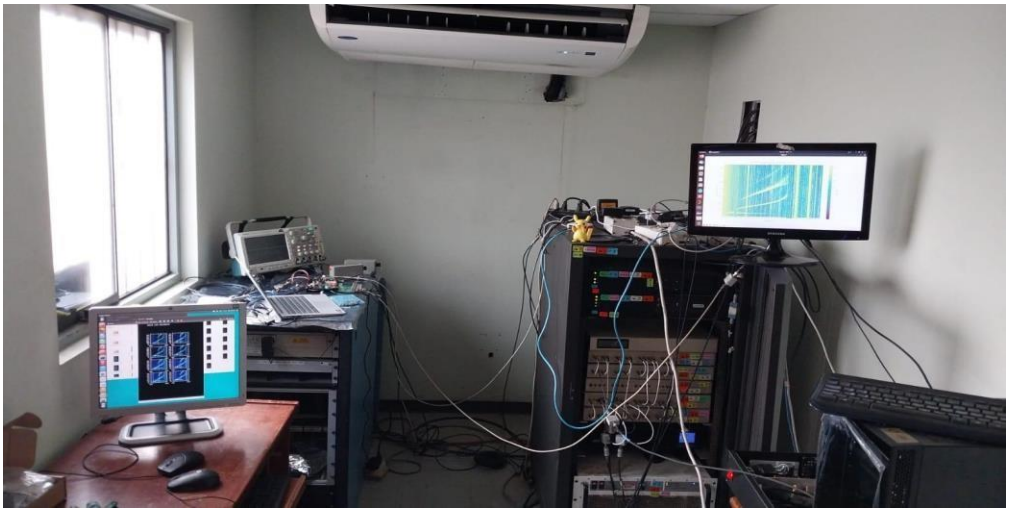

Fig. 105. Pruebas del generador de RF en las instalaciones de VIPIR.

Fuente: Elaboración propia

### 5.1.7.1. Sondeo vertical ionosféricos

Las señales de RF enviadas a la ionósfera fueron adquiridas con los 8 canales de recepción del radar VIPIR ubicado en el Radio Observatorio de Jicamarca (ROJ) y se generaron ionogramas de sondeo vertical que muestran el comportamiento de la ionósfera en ese momento, adquisición de datos se hicieron bajo el estándar horario UTC, se muestran a continuación los siguientes resultados:

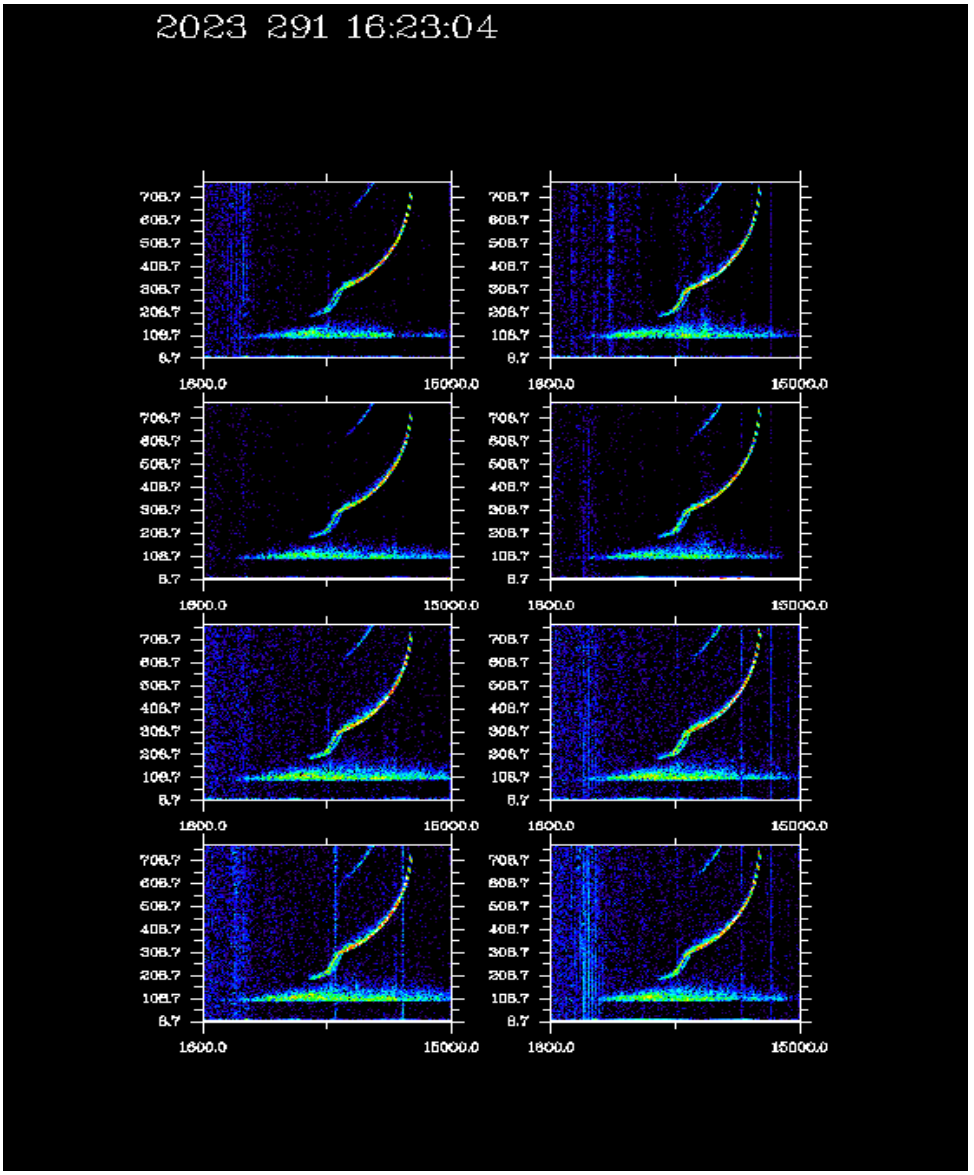


Fig. 106. Ionogramas generados con el receptor VIPIR en la Fecha:18-10-2023 hora:11:23 am.

Fuente: Elaboración propia.

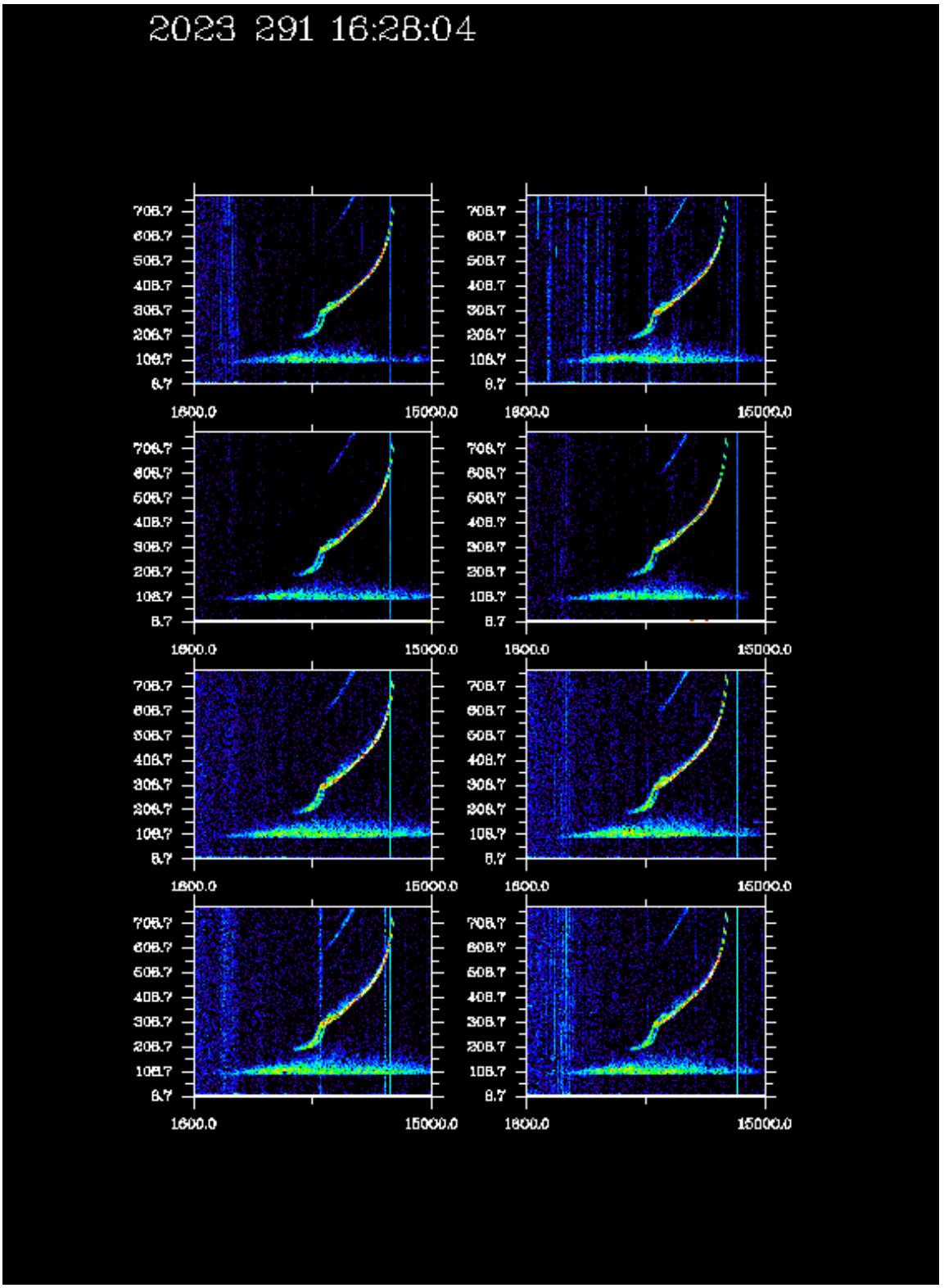


Fig. 107. Ionogramas generados con el receptor VIPIR en la Fecha:18-10-2023 hora:11:28 am.

Fuente: Elaboración propia.

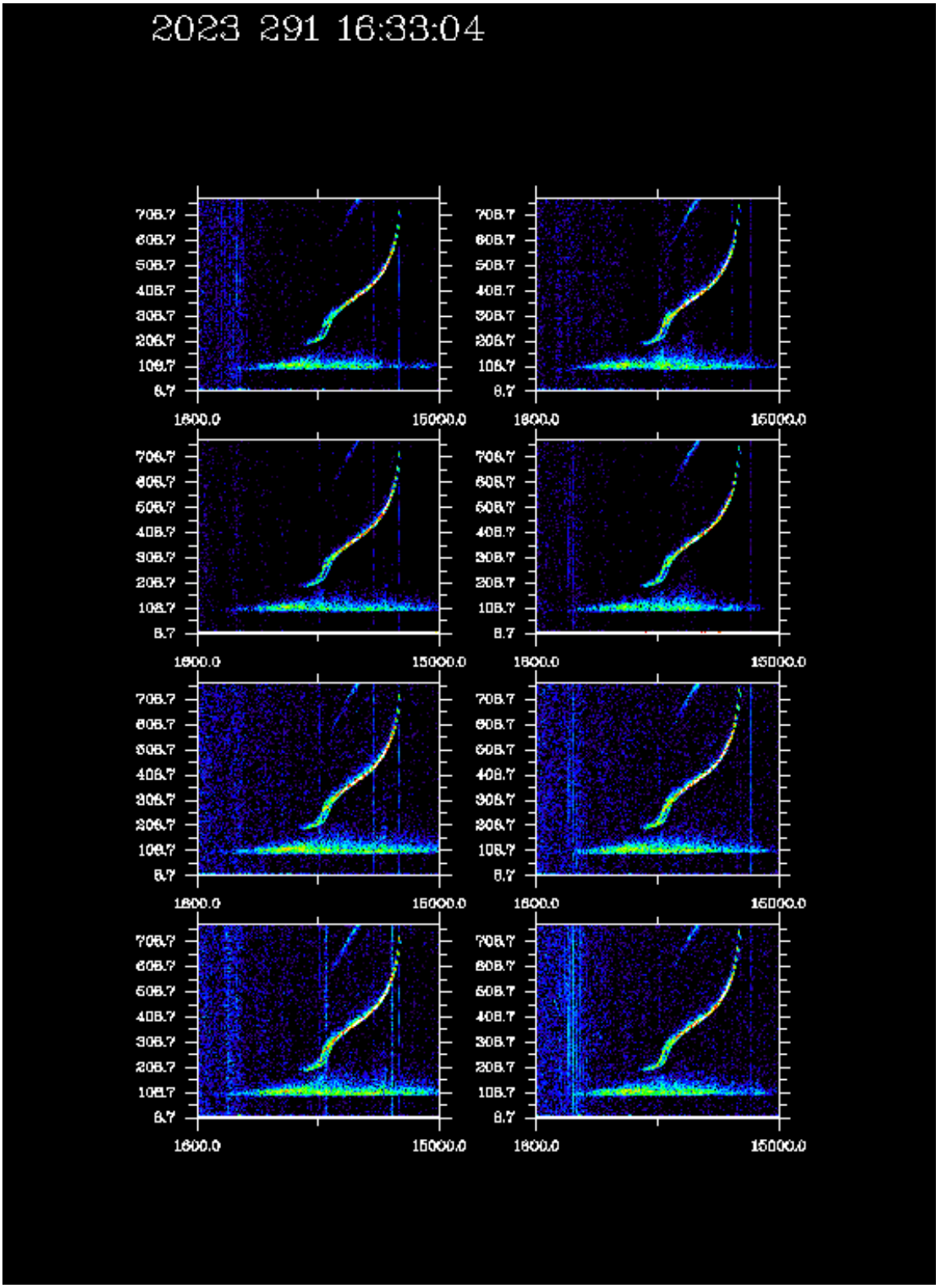


Fig. 108. Ionogramas generados con el receptor VIPIR en la Fecha:18-10-2023 hora:11:33 am.

Fuente: Elaboración propia.

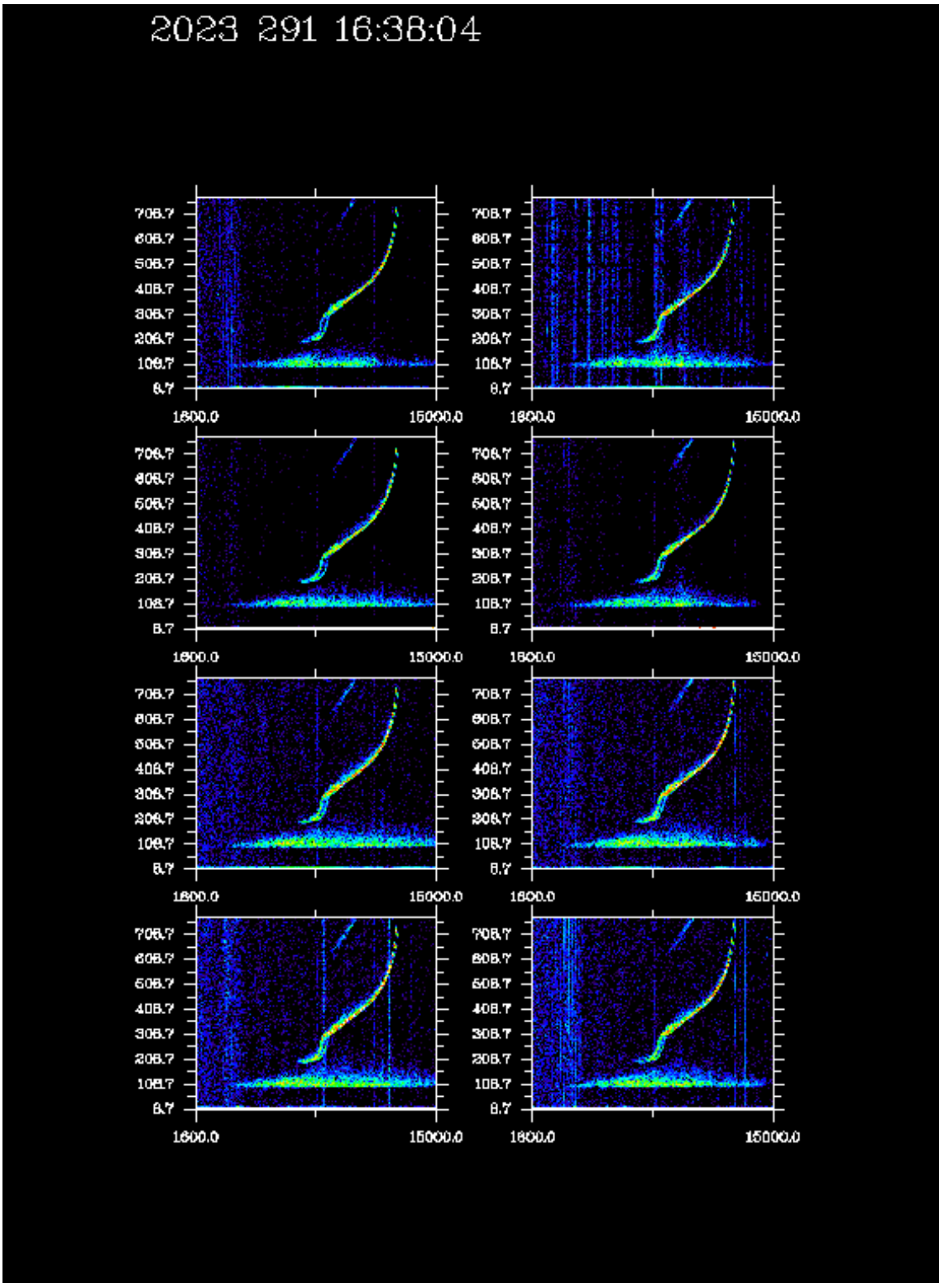


Fig. 109. Ionogramas generados con el receptor VIPIR en la Fecha:18-10-2023 hora:11:38 am.

Fuente: Elaboración propia.

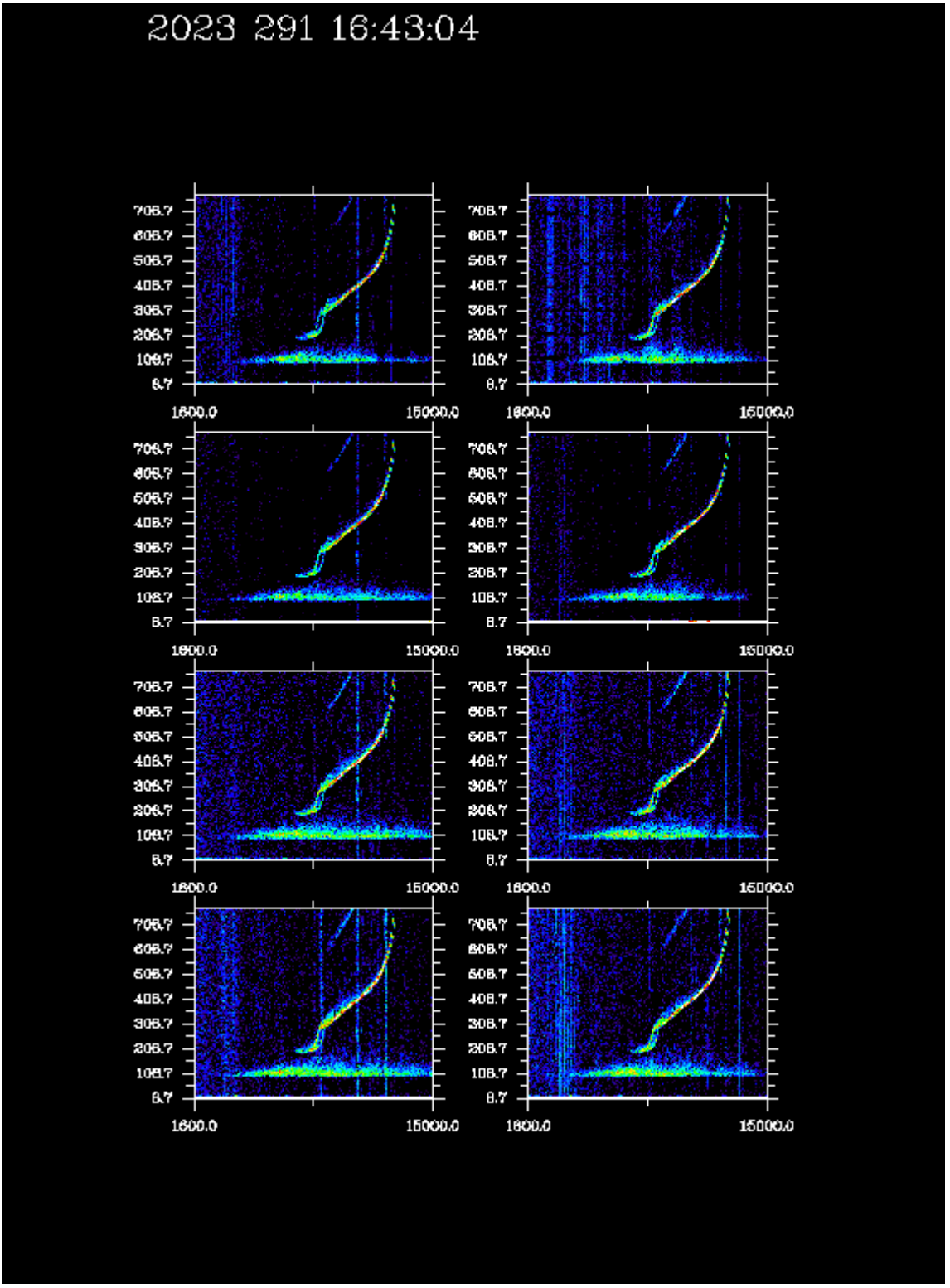


Fig. 110. Ionogramas generados con el receptor VIPIR en la Fecha:18-10-2023 hora:11:43 am.

Fuente: Elaboración propia.

### 5.1.7.2. Sondeo oblicuo ionosféricos

Las señales de RF enviadas a la ionósfera fueron adquiridas con un canal del receptor de ecos ionosféricos IER ubicado en la ciudad PIURA y se generaron ionogramas de sondeo oblicuo que muestran el comportamiento de la ionósfera en ese momento, adquisición de datos se hicieron bajo el estándar horario UTC, se muestran a continuación los siguientes resultados:

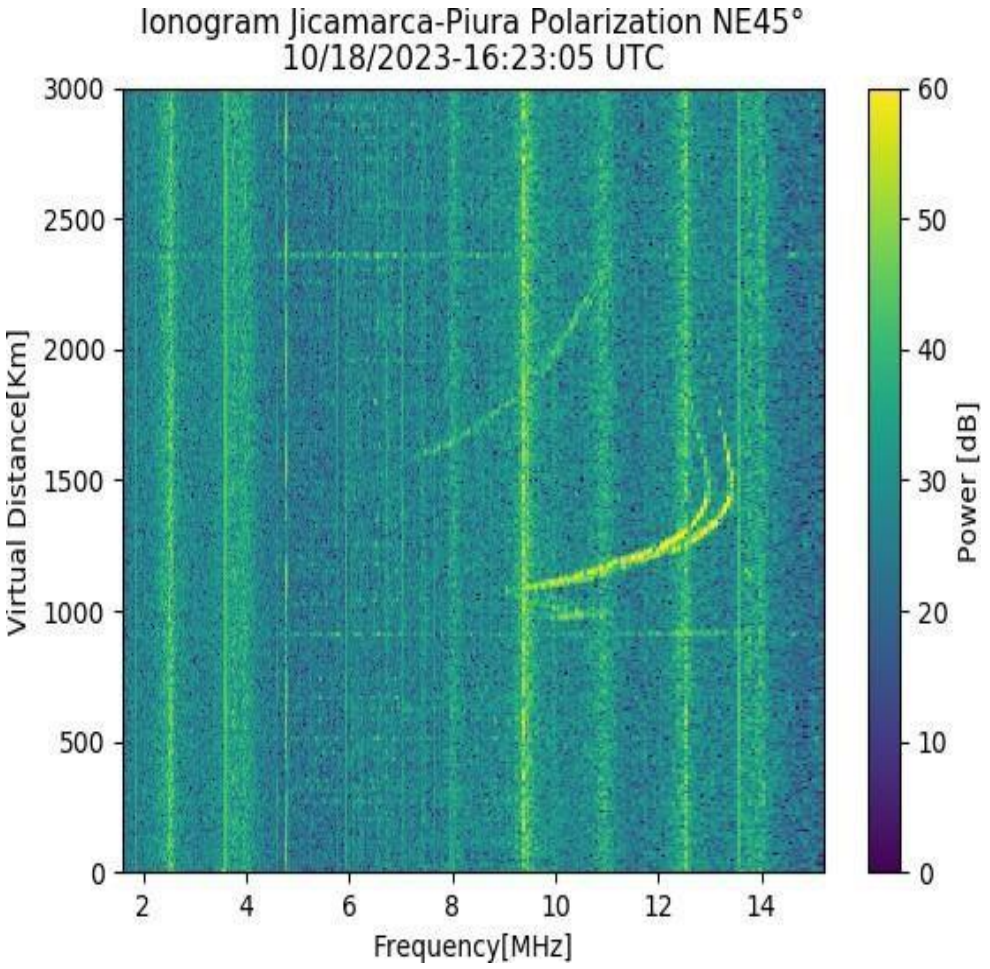


Fig. 111. Ionogramas generados con el receptor IER en la Fecha:18-10-2023 hora:11:23 am.

Fuente: Elaboración propia.

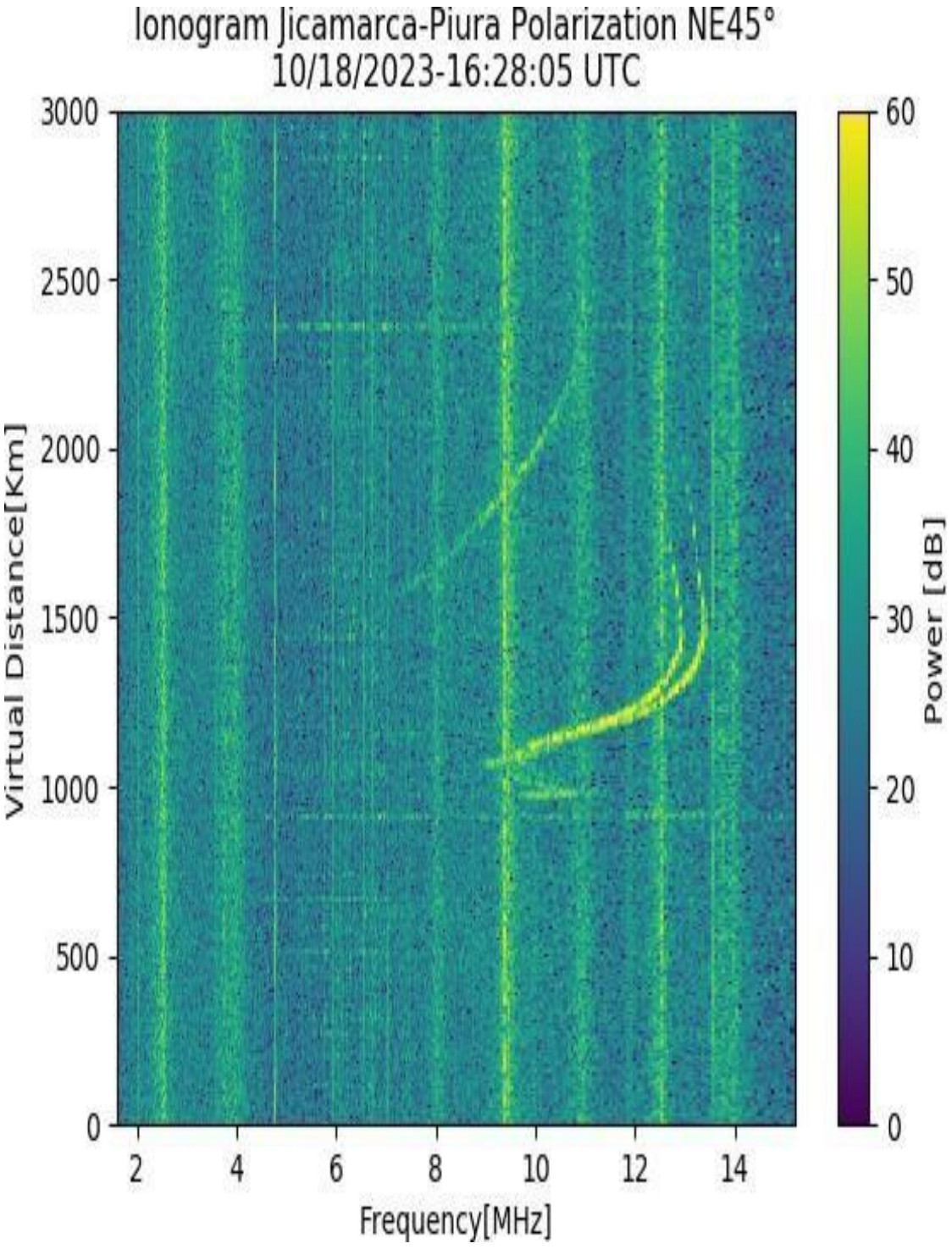


Fig. 112. Ionogramas generados con el receptor IER en la Fecha:18-10-2023 hora:11:28 am.

Fuente: Elaboración propia.

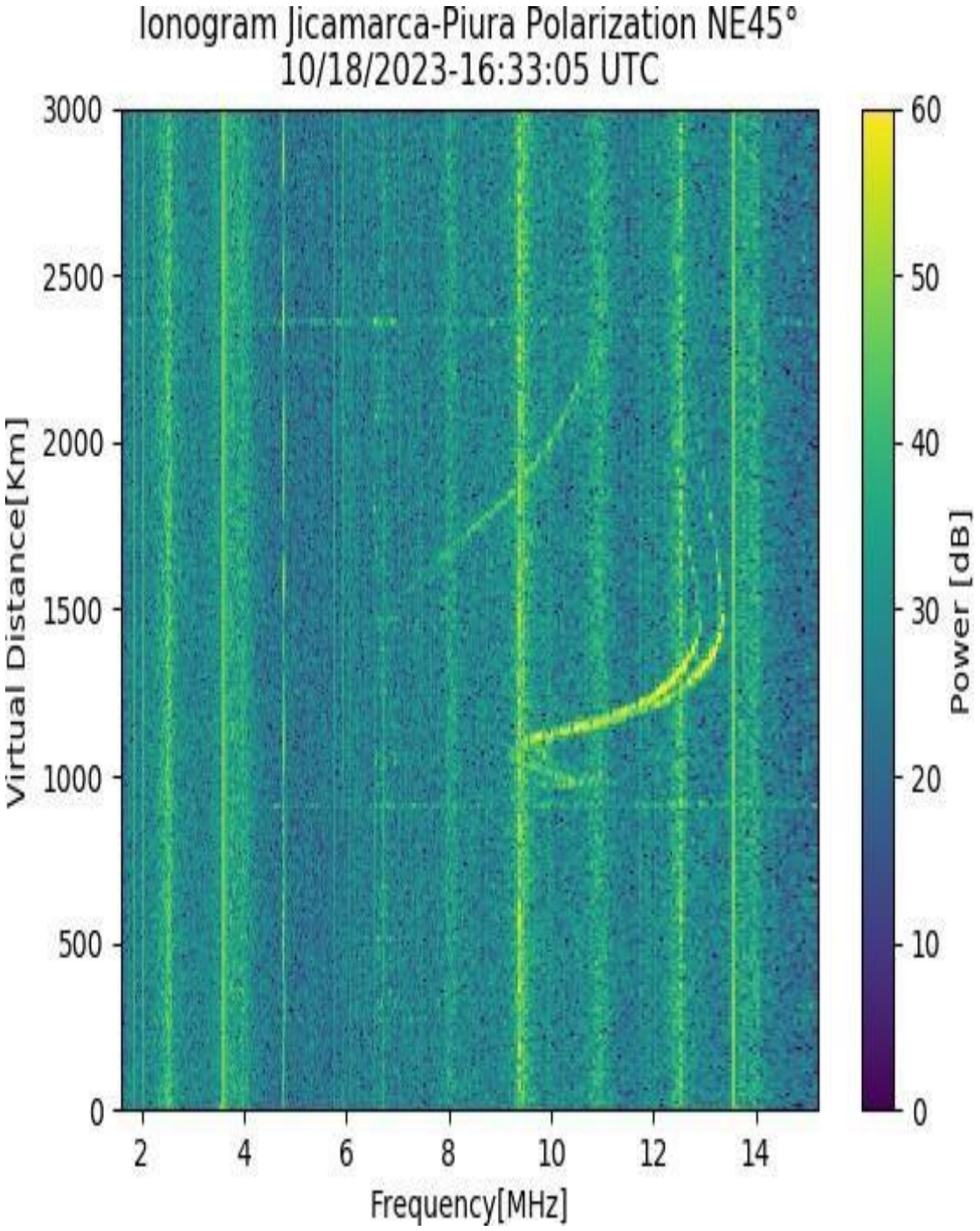


Fig. 113. Ionogramas generados con el receptor IER en la Fecha:18-10-2023 hora:11:33 am.

Fuente: Elaboración propia.

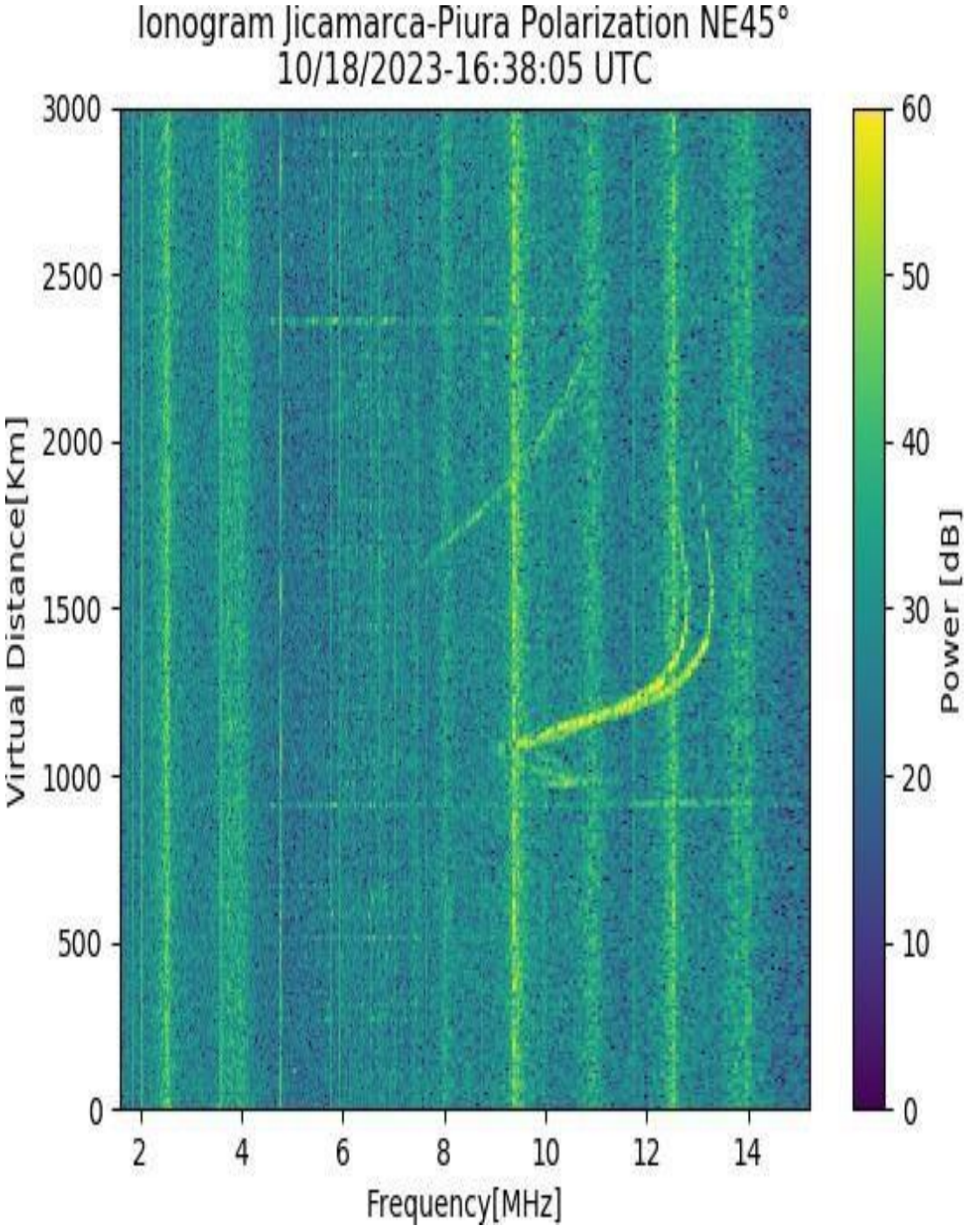


Fig. 114. Ionogramas generados con el receptor IER en la Fecha:18-10-2023 hora:11:38 am.

Fuente: Elaboración propia.

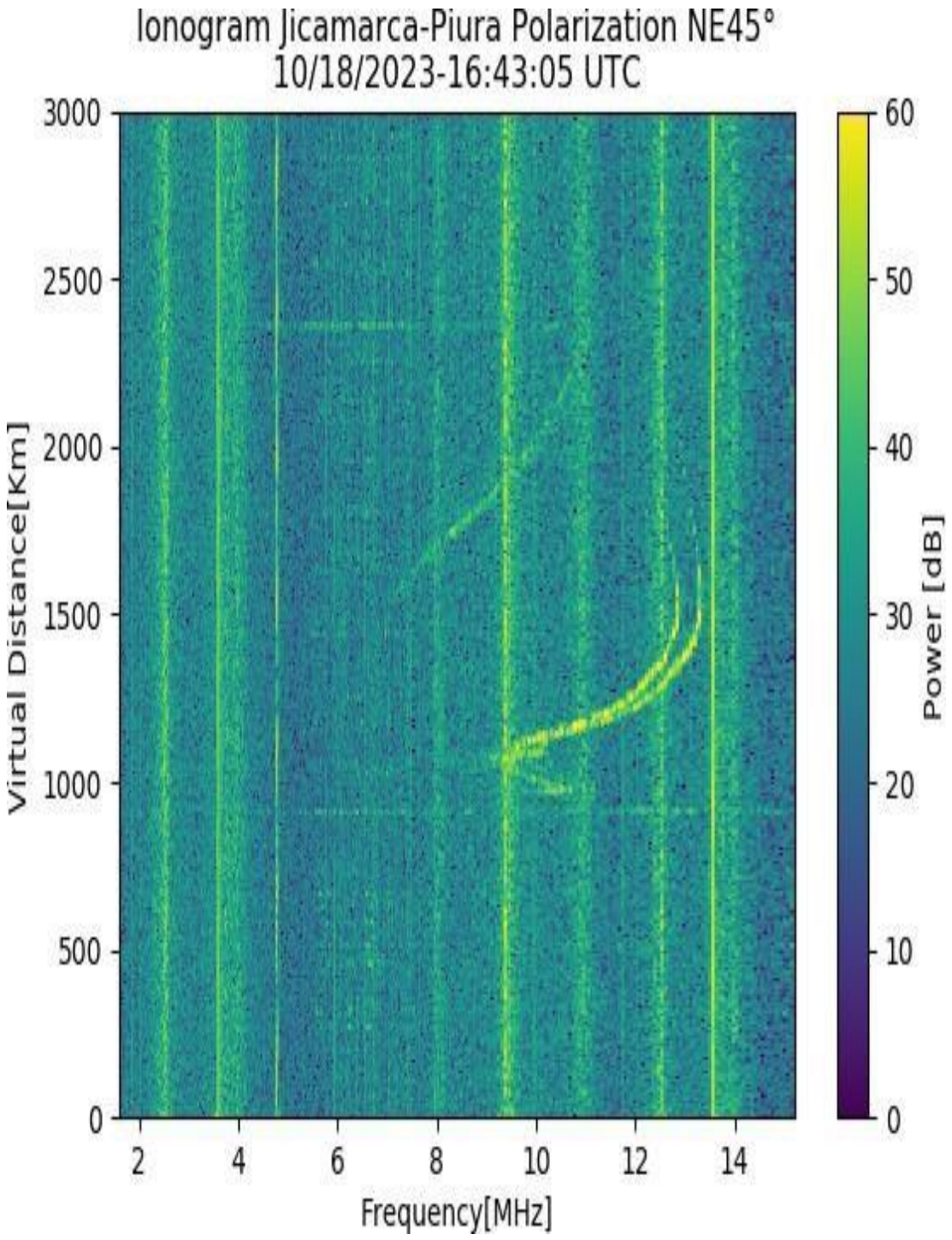


Fig. 115. Ionogramas generados con el receptor IER en la Fecha:18-10-2023 hora:11:43 am.

Fuente: Elaboración propia.

### 5.1.8. Recursos lógicos utilizados

La generación del archivo .bit que comprende el generador de señales de radiofrecuencia se realizó en el software Vivado 2019.1 y utilizó los siguientes recursos lógicos; LUT, LUT-RAM, FLIP-FLOP's, BRAM, GPIO'S (IO), BUFG, MMCM(multiplicadores de frecuencia). En la figura 116 se muestra el porcentaje de la utilización de cada uno de los recursos.

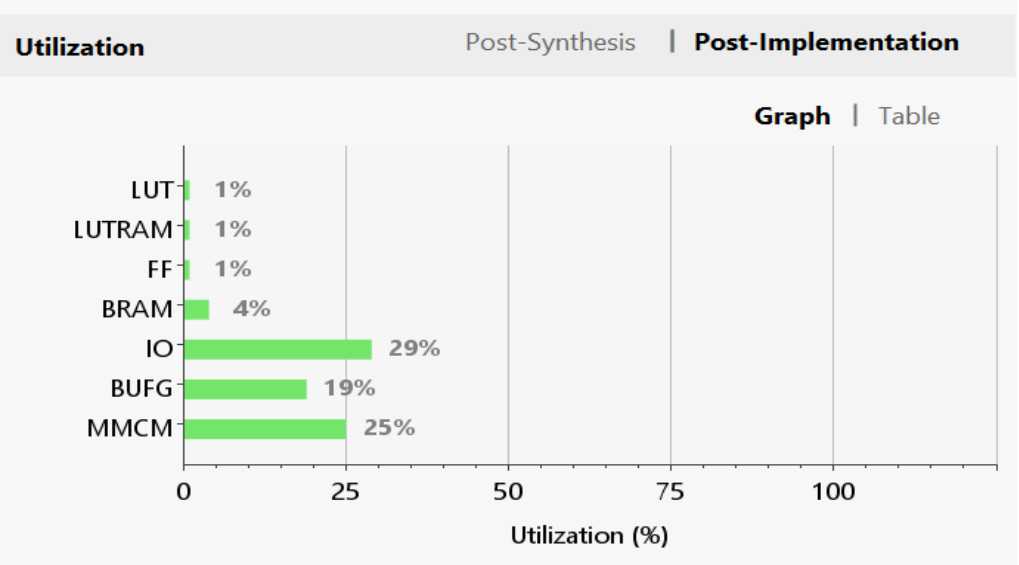


Fig. 116. Recursos lógicos utilizados en el generador de RF.

Fuente: Elaboración propia.

### 5.1.9. Temperatura de operación

En esta etapa analizamos la temperatura que alcanza en el modo de operación del generador de señales de RF sintetizado en la tarjeta electrónica Red Pitaya Signal Lab 250-12, dando como resultado una temperatura promedio de 45º Centígrados y el componente con mayor temperatura hallado fue el ADC de alta velocidad. llegando a un pico de temperatura de 50º Centígrados.

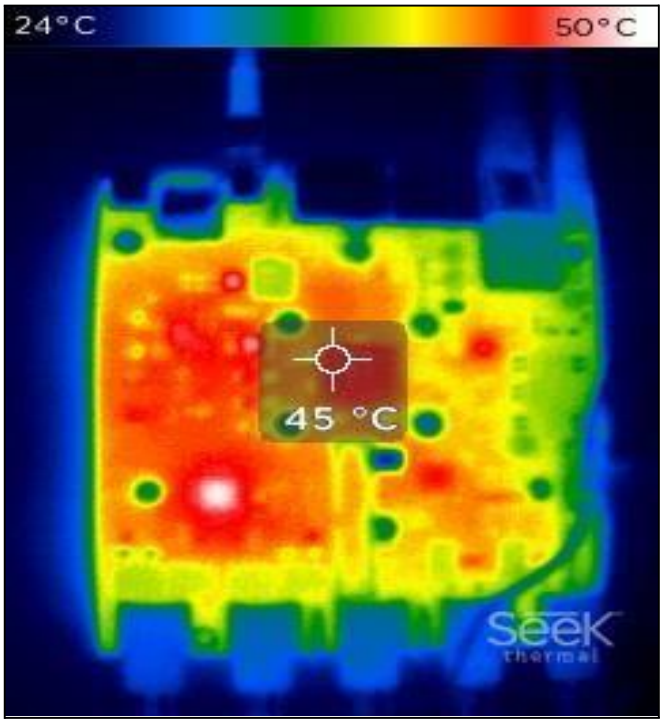


Fig. 117. Temperatura promedio del generador de RF.

Fuente: Elaboración propia.

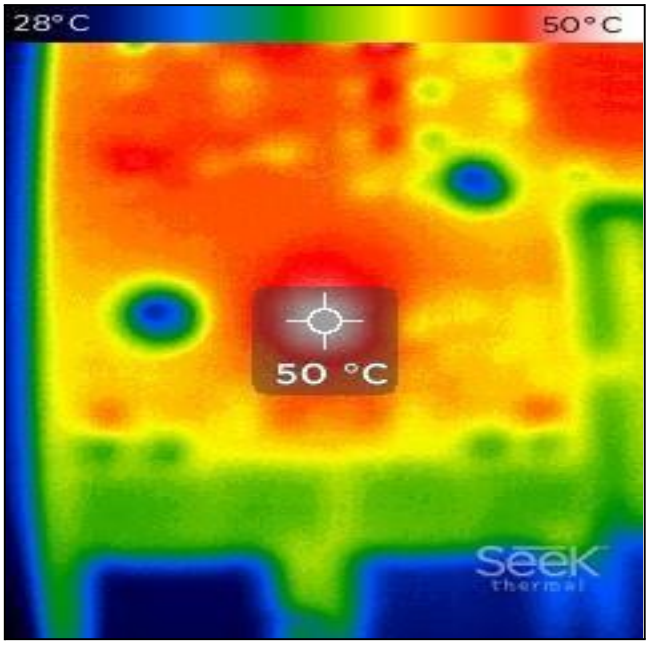


Fig. 118. Temperatura del ADC incorporado en la tarjeta Red Pitaya Signal Lab 250-12.

Fuente: Elaboración propia.

# VI. DISCUSIÓN DE RESULTADOS

## 6.1. Contrastación y demostración de la hipótesis con los resultados.

### 6.1.1. Contrastación y demostración de la hipótesis general con los resultados.

- **Contrastación de hipótesis general:**
  El diseño e implementación del generador de señales de radiofrecuencias basado en FPGA SoC al operar el transmisor de alta potencia de 4 KW para radar ionosonda en el Radio Observatorio de Jicamarca comprueba la hipótesis general.

### 6.1.2. Contrastación y demostración de las hipótesis específicas con los resultados.

- **Contrastación de hipótesis específica 1:**
  La implementación del módulo de sincronismo mediante la descripción hardware estructural, sincroniza el generador de señales de radiofrecuencia con la señal del receptor de GPS de 10 Mhz, además de generar los clock diferenciales de 250 Mhz (CLK-LOW, CLK-HIGH) que opera el DAC y ADC de alta velocidad de la placa Red Pitaya Signal Lab 250-12 y un clock de 250 Mhz que controla el sistema desarrollado en este trabajo de investigación, con lo mencionado se comprueba la hipótesis específica 1.

- **Contrastación hipótesis específica 2:**
  El diseño e implementación de los módulos de oscilador controlado numéricamente (NCO) y registros de memoria (RAM) que almacena los incrementos de fase o sintonizadores de palabra, generan el barrido de 1808 frecuencias para el transmisor de radar ionosonda, con lo mencionado se comprueba la hipótesis específica 2.

- **Contrastación hipótesis específica 3:**
  El diseño e implementación del módulo de modulación BPSK y OOK permite envíar señales de radiofrecuencias moduladas mediante el transmisor de alta potencia de 4 KW, con lo mencionado se comprueba la hipótesis específica 3.

## 6.2 Responsabilidad ética de acuerdo a los reglamentos vigentes

Como autor de la presente investigación soy responsable de la información brindada en el documento de Tesis titulado “Diseño e implementación de un generador de señales de radiofrecuencia basado en FPGA SoC para la operación de un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca”, de acuerdo con las normativas actuales de la Universidad Nacional del Callao

# VII. CONCLUSIONES

- Al realizar éste trabajo de investigación se concluye la correcta generación de las 1808 frecuencias, con la posibilidad de ampliar este rango de frecuencias debido a que se puede ampliar la longitud de palabra de cada registro y también se puede ampliar la cantidad de direcciones ya que se tiene todo el control del diseño.

- Se logró el sincronismo de la señal de 10 Mhz del receptor de GPS con el clock de 10 Mhz del sistema desarrollado en la investigación. A partir del clock del sistema se utiliza PLL (multiplicadores de frecuencia), para poder sincronizar los módulos que comprende el sistema y también los DAC y ADC de alta velocidad.

- Se realizaron pruebas exhaustivas para analizar las señales en el tiempo y en análisis espectral para corroborar que la frecuencia que se configuró en sistema desarrollado en esta investigación sea la misma que envíe al transmisor de alta potencia de 4 KW.

- Se realizaron pruebas de bola de cobre con el receptor "Ionospheric Echoes Receiver" *(IER)* basado en USPR, además de generar ionogramas con las señales adquiridas en ésta prueba.

- Se puso en operación el transmisor de 4 KW de radar ionosonda con el rack del generador de señales de radiofrecuencia en las instalaciones del radar "Vertical Incidence Pulsed Ionospheric Radar" (VIPIR)

- Los ecos ionosféricos generados por las señales de radiofrecuencia emitidas fueron captados por los receptores de VIPIR (ubicado en Lima) y IER (ubicado en Piura). A partir de estos datos, se procedió a la creación de ionogramas con el fin de analizar el estado de la ionosfera en momentos específicos.

# VIII. RECOMENDACIONES

- A partir del diseño implementado del generador de señales de radio frecuencia se recomienda realizar el receptor de radar ionosonda en la tarjeta Red Pitaya Signal Lab 250-12, para tener un radar ionosonda completamente funcional.

- Se recomienda utilizar el ARM cortex incorporado dentro del FPGA SoC para aprovechar todas las capacidades de la tarjeta Red Pitaya Signal Lab 250-12.

# IX. REFERENCIAS BIBLIOGRÁFICAS


[1] C. C. E. Castro, “Estudio de factibilidad de implementación de un sistema para medir el retardo en la propagación ionosférica”, San Salvador, 2019.

[2] C. F. Q. R. C. R. P. J. Jhon Jairo Barona Mendoza, “Implementation of an Electronic Ionosonde to Monitor the Earth’s Ionosphere via a Projected Column through USRP”, *Sensors - MDPI,* vol. 17, nº 5, pp. 946 - 969, 2017.

[3] G. Fariñas, ”Diseño en FPGA de una ionosonda generación”, *Telemática,* vol. 12, nº 2, pp. 42 - 55, 2013.

[4] A. J. C. Ortecho, “Implementación de sistema de radar ionosonda en Red Pitaya”, Repositorio del Instituto Geofísico del Perú, Lima, 2018.

[5] C. Bernal, “Metodología de la investigación”, Pearson education, 2010.

[6] C. F. y. P. B. R. Hernández, “Metodología de la Investigación”, México D.F.: McGraw-Hill, 2014.

[7] J. R. Observatory, «Jicamarca Digisonde,» 2006. [En línea]. Available: http://jro.igp.gob.pe/digisonde/dps4text.htm. [Último acceso: 7 10 2022].

[8] L. D. International, “Lowell Digisonde International”, [En línea]. Available: http://www.digisonde.com/pdf/Digisonde4DManual_LDI-web.pdf. . [Último acceso: 11-10-2022].

[9] B. R. Mahafza, Radar systems analysis and design using matlab, Huntsville, Alabama, USA: Taylor & Francis Group, 2013.

[10] R. Pitaya, “Red Pitaya”, [En línea]. Available: https://redpitaya.com/. [Último acceso: 13 10 2022].

[11] P. P. Chu, “RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability”, New Jersey: John Wiley & Sons, 2006.

[12] M. I. Skolnik, “Radar Handbook”, New York: Mc Graw Hill, 2008.

[13] A. A. s. a. B. Bazuin, “Toward Digital Transmitters with Amplitude Shift Toward Digital Transmitters with Amplitude Shift Toward Digital Transmitters with Amplitude Shift”, *IEEE Xplore,* pp. 1 - 7, 2017.

[14] A. G. S.O. Popescu, “Performance comparison of the BPSK and QPSK”, *Ieee xplore,* vol. 17, 2011.

[15] P. Reyes, E. Kudeki, G. Lehmacher, J. Chau, M. Milla, “VIPIR and 50 MHz Radar Studies of Gravity Wave Signatures in 150-km Echoes Observed at Jicamarca”, *AGU,* vol. 125, 2020.

[16] J. M. Suclupe, "Estudio de las capas E esporádicas tipo Blanketing sobre el Radio Observatorio de Jicamarca", Lima, 2019.

[17] R. Martin, "Study of waves observed in the equatorial ionospheric valley region using jicamarca ISR and Vipir ionosonde", Urbana, Illinois, 2017.

[18] E. Kudeki, “Applications of Radiowave Propagation”, Illinois, 2006.

# X. ANEXOS

## 8.1. Matriz de consistencia

<table>
<tr><th>Problema</th><th>Objetivos</th><th>Hipótesis</th><th colspan="3">Variables e indicadores</th><th>Método</th></tr>
<tr><td>Problema general: ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para operar un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?</td><td>Objetivo general: Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para operar un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.</td><td>Hipótesis general: Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se operará un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.</td><td colspan="3" rowspan="2"><table>
<tr><th>Variable</th><th>Dimensión</th><th>Indicadores</th></tr>
<tr><td>Variable independiente<br><br>Diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC:Es un dispositivo electrónico que genera patrones de señales periódicas o no periódicas y en este trabajo de investigación se generan de manera digital.</td><td>Sintetización de hardware</td><td>I1: Simulaciones con el Test bench.<br><br>I2: Pruebas en el laboratorio.</td></tr>
<tr><td rowspan="2">Variable dependiente<br><br>Operación de transmisor de radar ionosonda en el Radio Observatorio de Jicamarca: Los transmisores de ionosonda son radares HF que transmiten pulsos electromagnéticos desde un determinado punto de la Tierra hacia la ionosfera.</td><td>Sincronismo</td><td>I1: Desfase entre la señal generada y la señal de GPS.</td></tr>
<tr><td>Análisis espectral de las frecuencias</td><td>I2: Visualización de barrido de frecuencias con el analizador de espectro.</td></tr>
</table></td><td>Experimental</td></tr>
<tr><td>Problema específico 1: ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?<br><br>Problema específico 2: ¿Es posible diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA Soc para emitir un barrido de frecuencias en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca?<br><br>Problema específico 3: ¿Es posible diseñar e implementar un circuito modulador de señales de radiofrecuencia basado en FPGA SoC.</td><td>Objetivo específico 1: Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.<br><br>Objetivo específico 2: Diseñar e implementar un generador de señales de radiofrecuencia basado en FPGA SoC para emitir un barrido de frecuencias en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.<br><br>Objetivo específico 3: Diseñar e implementar un circuito modulador de señales de radiofrecuencia basado en FPGA SoC.</td><td>Hipótesis específica 1: Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se podrá sincronizar la señal de GPS en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.<br><br>Hipótesis específica 2: Con el diseño e implementación de un generador de señales de radiofrecuencias basado en FPGA SoC se podrá emitir un barrido de frecuencia en un transmisor de radar ionosonda en el Radio Observatorio de Jicamarca.<br><br>Hipótesis específica 3: El diseño e implementación del circuito modulador de señales de radiofrecuencias basado en FPGA SoC permitirá realizar modulación BPSK y OOK.</td><td colspan="3"></td><td>TIPO: APLICADA<br>NIVEL: EXPLICATIVO<br>ENFOQUE: CUANTITATIVO</td></tr>
</table>

Fuente: Elaboración propia.

## 8.2. Registros de frecuencias

| FREQUENCY REGISTER | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Nº | OUT FREQ | PHASE WIDTH (2^64) | CLK FREQ | SAMPLES_WAVE | PHASE INCREMENT | ROUNDING P. I. (DEC) | ROUNDING P. I. (HEX) | FREQ_GENERATE | ERROR |
| 1 | 1000000 | 18446744073709551616 | 250000000 | 250 | 73786976294838000 | 73786976294838000 | 0106 24DD 2F1A 9EF0 | 1000000.000000 | 0.000000 |
| 2 | 2000000 | 18446744073709551616 | 250000000 | 125 | 147573952589676000 | 147573952589676000 | 020C 49BA 5E35 3DE0 | 2000000.000000 | 0.000000 |
| 3 | 3000000 | 18446744073709551616 | 250000000 | 83.33333333 | 221360928884514000 | 221360928884514000 | 0312 6E97 8D4F DCD0 | 3000000.000000 | 0.000000 |
| 4 | 4000000 | 18446744073709551616 | 250000000 | 62.5 | 295147905179352000 | 295147905179352000 | 0418 9374 BC6A 7BC0 | 4000000.000000 | 0.000000 |
| 5 | 5000000 | 18446744073709551616 | 250000000 | 50 | 368934881474190000 | 368934881474190000 | 051E B851 EB85 1AB0 | 5000000.000000 | 0.000000 |
| 6 | 6000000 | 18446744073709551616 | 250000000 | 41.66666667 | 442721857769028000 | 442721857769028000 | 0624 DD2F 1A9F B9A0 | 6000000.000000 | 0.000000 |
| 7 | 7000000 | 18446744073709551616 | 250000000 | 35.71428571 | 516508834063866000 | 516508834063866000 | 072B 020C 49BA 5890 | 7000000.000000 | 0.000000 |
| 8 | 8000000 | 18446744073709551616 | 250000000 | 31.25 | 590295810358704000 | 590295810358704000 | 0831 26E9 78D4 F780 | 8000000.000000 | 0.000000 |
| 9 | 9000000 | 18446744073709551616 | 250000000 | 27.77777778 | 664082786653542000 | 664082786653542000 | 0937 4BC6 A7EF 9670 | 9000000.000000 | 0.000000 |
| 10 | 10000000 | 18446744073709551616 | 250000000 | 25 | 737869762948380000 | 737869762948380000 | 0A3D 70A3 D70A 3560 | 10000000.000000 | 0.000000 |
| 11 | 11000000 | 18446744073709551616 | 250000000 | 22.72727273 | 811656739243218000 | 811656739243218000 | 0B43 9581 0624 D450 | 11000000.000000 | 0.000000 |
| 12 | 12000000 | 18446744073709551616 | 250000000 | 20.83333333 | 885443715538056000 | 885443715538056000 | 0C49 BA5E 353F 7340 | 12000000.000000 | 0.000000 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 13 | 13000000 | 18446744073709551616 | 250000000 | 19.23076923 | 959230691832894000 | 959230691832894000 | 0D4F DF3B 645A 1230 | 13000000.000000 | 0.000000 |
| 14 | 14000000 | 18446744073709551616 | 250000000 | 17.85714286 | 1033017668127730000 | 1033017668127730000 | 0E56 0418 9374 A950 | 14000000.000000 | 0.000000 |
| 15 | 15000000 | 18446744073709551616 | 250000000 | 16.66666667 | 1106804644422570000 | 1106804644422570000 | 0F5C 28F5 C28F 5010 | 15000000.000000 | 0.000000 |
| 16 | 16000000 | 18446744073709551616 | 250000000 | 15.625 | 1180591620717410000 | 1180591620717410000 | 1062 4DD2 F1A9 F6D0 | 16000000.000000 | 0.000000 |
| 17 | 17000000 | 18446744073709551616 | 250000000 | 14.70588235 | 1254378597012250000 | 1254378597012250000 | 1168 72B0 20C4 9D90 | 17000000.000000 | 0.000000 |
| 18 | 18000000 | 18446744073709551616 | 250000000 | 13.88888889 | 1328165573307080000 | 1328165573307080000 | 126E 978D 4FDF 1D40 | 18000000.000000 | 0.000000 |
| 19 | 19000000 | 18446744073709551616 | 250000000 | 13.15789474 | 1401952549601920000 | 1401952549601920000 | 1374 BC6A 7EF9 C400 | 19000000.000000 | 0.000000 |
| 20 | 20000000 | 18446744073709551616 | 250000000 | 12.5 | 1475739525896760000 | 1475739525896760000 | 147A E147 AE14 6AC0 | 20000000.000000 | 0.000000 |
| 21 | 21000000 | 18446744073709551616 | 250000000 | 11.9047619 | 1549526502191600000 | 1549526502191600000 | 1581 0624 DD2F 1180 | 21000000.000000 | 0.000000 |
| 22 | 22000000 | 18446744073709551616 | 250000000 | 11.36363636 | 1623313478486440000 | 1623313478486440000 | 1687 2B02 0C49 B840 | 22000000.000000 | 0.000000 |
| 23 | 23000000 | 18446744073709551616 | 250000000 | 10.86956522 | 1697100454781270000 | 1697100454781270000 | 178D 4FDF 3B64 37F0 | 23000000.000000 | 0.000000 |
| 24 | 24000000 | 18446744073709551616 | 250000000 | 10.41666667 | 1770887431076110000 | 1770887431076110000 | 1893 74BC 6A7E DEB0 | 24000000.000000 | 0.000000 |
| 25 | 25000000 | 18446744073709551616 | 250000000 | 10 | 1844674407370950000 | 1844674407370950000 | 1999 9999 9999 8570 | 25000000.000000 | 0.000000 |

Fuente: Elaboración propia.

## 8.3. Registros de ancho de pulso y baudios

| PULSE WIDTH REGISTER | | | | | | BAUD REGISTER | |
|---|---|---|---|---|---|---|---|
| nº | PULSE_WIDTH(us) | PULSE_FREQ(hz) | FREQ_CLK | PULSE_WIDTH(DEC) | PULSE_WIDTH(HEX) | BAUD(DEC) | BAUD(HEX) |
| 1 | 0.000001 | 1000000 | 250000000 | 250 | FA | 1 | 1 |
| 2 | 0.000002 | 500000 | 250000000 | 500 | 1F4 | 3 | 3 |
| 3 | 0.000003 | 333333.3333 | 250000000 | 750 | 2EE | 5 | 5 |
| 4 | 0.000004 | 250000 | 250000000 | 1000 | 3E8 | 7 | 7 |
| 5 | 0.000005 | 200000 | 250000000 | 1250 | 4E2 | 9 | 9 |
| 6 | 0.000006 | 166666.6667 | 250000000 | 1500 | 5DC | 11 | B |
| 7 | 0.000007 | 142857.1429 | 250000000 | 1750 | 6D6 | 13 | D |
| 8 | 0.000008 | 125000 | 250000000 | 2000 | 7D0 | 15 | F |
| 9 | 0.000009 | 111111.1111 | 250000000 | 2250 | 8CA | 17 | 11 |
| 10 | 0.00001 | 100000 | 250000000 | 2500 | 9C4 | 19 | 13 |
| 60 | 0.00006 | 16666.66667 | 250000000 | 15000 | 3A98 | 119 | 77 |
| 262 | 0.000262 | 3816.793893 | 250000000 | 65500 | FFDC | 523 | 20B |

Fuente: Elaboración propia.

## 8.4. Registros de ancho del IPP

| IPP REGISTER | | | | | |
|---|---|---|---|---|---|
| nº | IPP_WIDTH(ms) | IPP_FREQ(hz) | FREQ_CLK | IPP(DEC) | IPP(HEX) |
| 1 | 0.001 | 1000 | 250000000 | 250000 | 3D090 |
| 2 | 0.002 | 500 | 250000000 | 500000 | 7A120 |
| 3 | 0.003 | 333.3333333 | 250000000 | 750000 | B71B0 |
| 4 | 0.004 | 250 | 250000000 | 1000000 | F4240 |
| 5 | 0.005 | 200 | 250000000 | 1250000 | 1312D0 |
| 6 | 0.006 | 166.6666667 | 250000000 | 1500000 | 16E360 |
| 7 | 0.007 | 142.8571429 | 250000000 | 1750000 | 1AB3F0 |
| 8 | 0.008 | 125 | 250000000 | 2000000 | 1E8480 |
| 10 | 0.01 | 100 | 250000000 | 2500000 | 2625A0 |
| 67 | 0.067 | 14.92537313 | 250000000 | 16750000 | FF95B0 |

Fuente: Elaboración propia.

## 8.5. Código .ino del microcontrolador Seeeduino XIAO

```
#include "all_freq_list_1808.h"
#include <SPI.h>
#include <Wire.h>
#include <max7311.h>
max7311 mcp(0x20);
//#include <PriUint64.h>

const int CS = 7;              //chip select
const int fpga_pin = 1;        //FPGA Confirmation
int previous_state = HIGH; //state
int actual_state;
int timeRelay = 100;

void setup(void)
{
  Serial.begin(115200);
  //relé - mcp
  mcp.begin();//x.begin(true) will override automatic SPI initialization
  mcp.gpioPinMode(OUTPUT);
  //spi Master
  pinMode(CS, OUTPUT);
  digitalWrite(CS, HIGH);
  pinMode(fpga_pin, INPUT);
  SPI.begin();
  SPI.beginTransaction (SPISettings (1000000, MSBFIRST, SPI_MODE0));
  Serial.println("--- Radiofrequency signal generator ---");
  ADC_DAC();
}


void loop(void)
{
  actual_state = digitalRead(fpga_pin);

  if (actual_state == LOW && previous_state == HIGH) {

    Serial.println("--- Falling edge detected ---");
```

```
ADC_DAC();

//star-stop
digitalWrite(CS, LOW);
SPI.transfer16(0x8000);//address
SPI.transfer16(0x0077);//baud ： 0x0013-10us | 0x0077-60us
SPI.transfer16(0x07F0);//repeat : 0x0018-25rp | 0x070F-1807rp  |
//0x07F0-2032rp
SPI.transfer16(0x0000);//reserved
SPI.transfer16(0x0000);//stop
digitalWrite(CS, HIGH);
delay(10);

//ipp pulse-width
digitalWrite(CS, LOW);
SPI.transfer16(0x8001);//address
SPI.transfer16(0x0000);
SPI.transfer16(0x3A98); //pul_Wd: 0x09C4-10us | 0x3A98-60us
SPI.transfer16(0x0026); //ipp ： 0x0003-1ms | 0x0026-10ms
SPI.transfer16(0x25A0); //ipp ： 0xD090-1ms | 0x25A0-10ms
digitalWrite(CS, HIGH);
delay(1);

//code
digitalWrite(CS, LOW);
SPI.transfer16(0x8002);//address
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
digitalWrite(CS, HIGH);
delay(1);

digitalWrite(CS, LOW);
SPI.transfer16(0x8003);//address
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
SPI.transfer16(0x0000);
digitalWrite(CS, HIGH);
delay(1);
```

```
//frequency
//for (uint16_t i=0; i<sizeof(freqs)/4; i++) {
for (uint16_t i=0; i<1808; i++) {
  int fpga_nco = 64;
  int usrp_nco = 32;

  //Serial.print("FPGA resolution 64 bits : ");
  double fpga_res = ((double)250000000.0)/((double)pow(2,fpga_nco));
  //Serial.println( fpga_res,40);

  //Serial.print("USRP resolution 32 bits : ");
  double usrp_res = ((double)100000000.0)/((double)pow(2,usrp_nco));
  //Serial.println(usrp_res,40);

  volatile double temp2 = ((double)freqs[i])*((double)pow(2,fpga_nco));
  volatile uint64_t temp = (temp2)/250000000;
      volatile double fpga_freq =
      (((double)temp)*250000000.0)/((double)pow(2,fpga_nco));
  //Serial.print("FPGA Freq. : ");
  //Serial.println(fpga_freq,64);

  volatile uint64_t usrp = freqs[i] << 32;
  volatile uint64_t usrp2 = usrp/100000000;
      volatile double usrp_freq =
      (((double)usrp2)*100000000.0)/((double)pow(2,usrp_nco));
  //Serial.print("USRP Freq. : ");
  //Serial.println(usrp_freq,64);

  //Serial.print("Error Frecuency : " );
  double freq_err = (abs(fpga_freq - usrp_freq));
  //Serial.println(freq_err,64);

  /*
  Serial.print("FPGA Accumulator : " );
  Serial.println(temp);
  Serial.print("FPGA Accumulator hexa : " );
  Serial.println(PriUint64<HEX>(((uint64_t)((temp)))));
  Serial.println(PriUint64<HEX>(((uint16_t)(0xFFFF&(temp>>(48))))));
  Serial.println(PriUint64<HEX>(((uint16_t)(0xFFFF&(temp>>(32))))));
  Serial.println(PriUint64<HEX>(((uint16_t)(0xFFFF&(temp>>(16))))));
```

```
    Serial.println(PriUint64<HEX>(((uint16_t)(0xFFFF&(temp)))));
    */

    digitalWrite(CS, LOW);
    SPI.transfer16((uint16_t)(32768+(4+i)));
    SPI.transfer16((uint16_t)(0xFFFF&(temp>>(48))));
    SPI.transfer16((uint16_t)(0xFFFF&(temp>>(32))));
    SPI.transfer16((uint16_t)(0xFFFF&(temp>>(16))));
    SPI.transfer16((uint16_t)(0xFFFF&(temp)));
    digitalWrite(CS, HIGH);
    delay(1);
   }

   //star
   digitalWrite(CS, LOW);
   SPI.transfer16(0x8000);//address
   SPI.transfer16(0x0077);//baud  : 0x0013-10us | 0x0077-60us
   SPI.transfer16(0x07F0);//repeat : 0x0018-25rp | 0x070F-1807rp  |
        //0x07F0-2032rp
   SPI.transfer16(0x0000);//reserved
   SPI.transfer16(0x0001);//star
   digitalWrite(CS, HIGH);
   delay(10);
  }

    previous_state = actual_state; //Update the previous state with the
current state
}


void ADC_DAC(){
  // -------------ALL OFF------------
  for (int i = 0; i < 15; i++) {
    mcp.gpioDigitalWrite(i, LOW);
    delay(timeRelay);
  }
  Serial.println("---LOW DAC/DAC---");


  //------------HIGH ALL DAC---------
  // DAC 2 = 0,1 (K6)
  // DAC 1 = 2,3 (K5)
```

```
int a= 0;
int b= 4;

for (int i = a; i < b; i++) {
  mcp.gpioDigitalWrite(i, HIGH);
  delay(timeRelay);
}
Serial.println("---HIGH DAC---");


//----------HIGH ALL ADC----------
// ADC 2 Atenuator = 8,9 (K4)
// ADC 1 Atenuator = 10,11 (K3)
// ADC 2 AC/DC = 12,13 (K2)
// ADC 1 AC/DC = 14,15 (K1)
int c= 8;
int d= 16;

for (int i = c; i < d; i++) {
  mcp.gpioDigitalWrite(i, HIGH);
  delay(timeRelay);
}
Serial.println("---HIGH ADC---");
```

## 8.6. Código .h del microcontrolador Seeeduino XIAO

```
const uint64_t freqs[1808] = {      1601000,
                                    1609000,
                                    1617039,
                                    1625120,
                                    1601000,
                                    1609000,
                                    1617039,
                                    1625120,
                                    1601000,
                                    1609000,
                                    1617039,
                                    1625120,
                                    1601000,
                                    1609000,
                                    1617039,
                                    1625120,
                                       .
                                       .
                                       .

                                    14872322,
                                    14946678,
                                    15021407,
                                    15096509,
                                    15171986,
                                    14946678,
                                    15021407,
                                    15096509,
                                    15171986,
                                    14946678,
                                    15021407,
                                    15096509,
                                    15171986,
                                    14946678,
                                    15021407,
                                    15096509,
                                    15171986};
```

## 8.7. Código VHDL del módulo spi controller

```
----------------------------------------------------------------------------------
-- Company          :    Jicamarca Radio Observatory
-- Engineer         :    Brayan Lui Estalla Quinteros
--
-- Create Date      :    14:00:03 18/04/2023
-- Design Name      :
-- Module Name      :    spi controller
-- Project Name     :    Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision         :
-- Additional Comments:
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

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

entity spi_controller is
    Generic (
        REG_ADDRESS_WIDTH : natural := 11;  -- address output width
        REG_DATA_WIDTH    : natural := 64;  -- data output width
        WORD_SIZE         : natural := 80   -- size of transfer word in
                                               bits, must be power of two
    );
```

```
    Port (
       CLK_SC_I       : in  std_logic; -- system clock
       RST_SC_I       : in  std_logic; -- high active synchronous reset
       SCLK           : in  std_logic; -- SPI clock
       CS_N           : in  std_logic; -- SPI chip select, active in low
       MOSI           : in  std_logic; -- SPI serial data from master to slave
       MISO           : out std_logic; -- SPI serial data from slave to master
       WR_SC_O        : out std_logic;
      ADDRESS_SC_O : out std_logic_vector (REG_ADDRESS_WIDTH-1
                          downto 0);
      DATA_SC_O    : out std_logic_vector (REG_DATA_WIDTH-1 downto 0)

    );
end entity;

architecture RTL of spi_controller is

    constant BIT_CNT_WIDTH : natural :=
    natural(ceil(log2(real(WORD_SIZE))));
    signal sclk_meta          : std_logic := '0';
    signal cs_n_meta          : std_logic := '1';
    signal mosi_meta          : std_logic := '0';
    signal sclk_reg           : std_logic := '0';
    signal cs_n_reg           : std_logic := '1';
    signal mosi_reg           : std_logic := '0';

    signal spi_clk_reg            : std_logic := '0';
    signal spi_clk_redge_en : std_logic := '0';
    signal spi_clk_fedge_en  : std_logic := '0';

    signal spi_cs_n_reg          : std_logic := '1';
    signal spi_cs_n_redge_en : std_logic := '0';
    signal spi_cs_n_fedge_en : std_logic := '0';

    signal bit_cnt          : unsigned(BIT_CNT_WIDTH-1 downto
    0):=(others=>'0');
    signal data_shreg     : std_logic_vector(WORD_SIZE-1 downto
    0):=(others=>'0');

    signal s_w_r            : std_logic := '0';
    signal s_address       : std_logic_vector(REG_ADDRESS_WIDTH-1
    downto 0):=(others=>'0');
```

```
signal  s_data                  : std_logic_vector(REG_DATA_WIDTH-1 downto
0):=(others=>'0');

begin

   ------------------------------------------------------------------------
   -- INPUT SYNC REGISTER
   ------------------------------------------------------------------------

   sync_ffs_p : process (CLK_SC_I)
   begin
      if (RST_SC_I = '1') then
         sclk_meta <= '0';
         cs_n_meta <= '1';
         mosi_meta <= '0';
         sclk_reg  <= '0';
         cs_n_reg  <= '1';
         mosi_reg  <= '0';
      elsif (rising_edge(CLK_SC_I)) then
         sclk_meta <= SCLK;
         cs_n_meta <= CS_N;
         mosi_meta <= MOSI;
         sclk_reg  <= sclk_meta;
         cs_n_reg  <= cs_n_meta;
         mosi_reg  <= mosi_meta;
      end if;
   end process;

   ------------------------------------------------------------------------
   -- SPI CLOCK REGISTER
   ------------------------------------------------------------------------

   spi_clk_reg_p : process (CLK_SC_I)
   begin
      if (rising_edge(CLK_SC_I)) then
         if (RST_SC_I = '1') then
            spi_clk_reg <= '0';
         else
            spi_clk_reg <= sclk_reg;
         end if;
      end if;
   end process;
```

```
-------------------------------------------------------------------------------
-- SPI CHIP SELECT REGISTER
-------------------------------------------------------------------------------

spi_cs_n_reg_p : process (CLK_SC_I)
begin
   if (rising_edge(CLK_SC_I)) then
      if (RST_SC_I = '1') then
         spi_cs_n_reg <= '1';
      else
         spi_cs_n_reg <= cs_n_reg;
      end if;
   end if;
end process;

-------------------------------------------------------------------------------
-- SPI CLOCK FLAGS
-------------------------------------------------------------------------------

spi_clk_fedge_en <= not sclk_reg and spi_clk_reg;
spi_clk_redge_en <= sclk_reg and not spi_clk_reg;

-------------------------------------------------------------------------------
-- SPI CHIP SELECT FLAGS
-------------------------------------------------------------------------------

spi_cs_n_fedge_en <= not cs_n_reg and spi_cs_n_reg;
spi_cs_n_redge_en <= cs_n_reg and not spi_cs_n_reg;

-------------------------------------------------------------------------------
-- RECEIVED BITS COUNTER
-------------------------------------------------------------------------------

      bit_cnt_p : process (CLK_SC_I)
      begin
            if (rising_edge(CLK_SC_I)) then
                  if (RST_SC_I = '1') then
                        bit_cnt <= (others => '0');
                        data_shreg <= (others => '0');
                  else
```

```
                if (spi_cs_n_reg = '1') then
                    bit_cnt <= (others => '0');

                elsif (spi_clk_redge_en = '1') then
                    bit_cnt <= bit_cnt + 1;
                   data_shreg <= data_shreg(WORD_SIZE-2 downto 0) &
                   mosi_reg;
                    if (bit_cnt =  WORD_SIZE-1) then
                       bit_cnt <= (others => '0');
                    end if;
                end if;
              end if;
          end if;
      end process;

-------------------------------------------------------------------------------
-- MISO
-------------------------------------------------------------------------------

miso_p : process (CLK_SC_I)
begin
   if (rising_edge(CLK_SC_I)) then
      if (RST_SC_I = '1') then
                          MISO <= '0';
      elsif (spi_clk_fedge_en = '1' and cs_n_reg = '0') then
         MISO <= data_shreg(WORD_SIZE-1);
      end if;
   end if;
end process;

-------------------------------------------------------------------------------
-- REGISTER OF OUTPUT SIGNALS
-------------------------------------------------------------------------------

output : process (CLK_SC_I)
begin
   if (rising_edge(CLK_SC_I)) then
      if (RST_SC_I = '1') then
         s_w_r         <= '0';
         s_address  <= (others=>'0');
         s_data         <= (others=>'0');
```

```
        elsif(spi_cs_n_redge_en='1') then
            s_w_r       <= data_shreg(WORD_SIZE-1);
            s_address   <= data_shreg(REG_ADDRESS_WIDTH+
                             REG_DATA_WIDTH-1 downto REG_DATA_WIDTH);
            s_data      <= data_shreg(REG_DATA_WIDTH-1 downto 0);
        else
            s_w_r       <= '0';
            s_address   <= (others=>'0');
            s_data      <= (others=>'0');
          end if;
        end if;
    end process;

    -------------------------------------------------------------------------
    -- REGISTER OF OUTPUT SIGNALS
    -------------------------------------------------------------------------

    WR_SC_O         <=  s_w_r;
   ADDRESS_SC_O <= s_address;
   DATA_SC_O        <= s_data;

end architecture;
```

## 8.8. Código VHDL del módulo memory controller

```
----------------------------------------------------------------------------------
-- Company           :     Jicamarca Radio Observatory
-- Engineer          :     Brayan Lui Estalla Quinteros
--
-- Create Date       :     14:00:03 18/04/2023
-- Design Name       :
-- Module Name       :     memory controller
-- Project Name      :     Ionosonde
-- Target Devices    :
-- Description:
--
-- Revision          :
-- Additional Comments:
----------------------------------------------------------------------------------

--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

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

entity memory_controller is
    Generic (
        REG_ADDRESS_WIDTH  : natural := 11;  -- register address width
        REG_DATA_WIDTH     : natural := 64;  -- register word width
        REG_REPEAT_WIDTH   : natural := 11;  -- register repeat width
        REG_BAUD_WIDTH     : natural := 8;   -- register baud width
        REG_PULSE_WIDTH    : natural := 16;  -- register pulse width
        REG_IPP_WIDTH      : natural := 24;  -- register ipp width
        REG_CODE_WIDTH     : natural := 125; -- register code width
```

```
    REG_PHASE_INC_WIDTH : natural := 64   -- register frequency width
  );

  Port (
    CLK_MC_I              : in  std_logic;
    RST_MC_I              : in  std_logic;
    WR_MC_I               : in  std_logic;
    ADDRESS_MC_I          : in std_logic_vector (REG_ADDRESS_WIDTH-1
                           downto 0);
    DATA_MC_I             : in  std_logic_vector (REG_DATA_WIDTH-1
                           downto 0);
    WR_MC_O               : out  std_logic;
    ADDRESS_MC_O          : out  std_logic_vector
                           (REG_ADDRESS_WIDTH-1 downto 0);
    DATA_MC_O             : out std_logic_vector (REG_DATA_WIDTH-1
                           downto 0);
    NEXT_SEQ_MC_I         : in  std_logic;
    TRIGGER_MC_I          :  in  std_logic;
    TRIGGER1_MC_O         : out std_logic :='0';
    REG_START_MC_O  : out std_logic :='0';
    REG_RST_MC_O          : out std_logic :='0';
    REG_PULSE_WIDTH_MC_O : out std_logic_vector
                           (REG_PULSE_WIDTH-1 downto 0):=(others=>'0');
    REG_IPP_MC_O          : out std_logic_vector(REG_IPP_WIDTH-1
                            downto 0):=(others=>'0');
    REG_CODE_MC_O         : out std_logic_vector(REG_CODE_WIDTH-1
                            downto 0):=(others=>'0');
    REG_BAUD_MC_O         : out std_logic_vector(REG_BAUD_WIDTH-1
                            downto 0):=(others=>'0')
   );

end memory_controller;

architecture Behavioral of memory_controller is

  signal  counter_nex_seq :  std_logic_vector(REG_REPEAT_WIDTH-1
   downto 0):=(others=>'0');
  signal  s_reg_start_prev  :  std_logic:='0';
  signal s_reg_start            :  std_logic:='0';
  signal s_reg_rst              : std_logic:='0';
  signal  s_reg_phase_inc  :  std_logic_vector(REG_PHASE_INC_WIDTH-1
  downto 0):=(others=>'0');
```

```
  signal  s_reg_repeat  :  std_logic_vector(REG_REPEAT_WIDTH-1
  downto 0):=(others=>'0');
  signal   s_tg_meta_1   :   std_logic:='0';
  signal   s_tg_meta_2   :   std_logic:='0';
  signal   s_tg_meta_3   :   std_logic:='0';
  signal    s_tg_fedge    :   std_logic:='0';
  signal s_tg_redge         : std_logic:='0';
  signal    s_pulse_on    :   std_logic:='0';
  signal    s_pulse_off   :   std_logic:='0';
  signal  s_reg_pulse_off : std_logic:='0';
  signal S_WR_MC_O    : std_logic;
 signal S_ADDRESS_MC_O    : std_logic_vector
 (REG_ADDRESS_WIDTH-1 downto 0);
 signal S_DATA_MC_O     : std_logic_vector (REG_DATA_WIDTH-1
 downto 0);


 -- counter gate
 signal s_reg_pulse_off_a_2  : std_logic:='0';
  signal s_counter             : integer range 0 to 24999999 :=0;
  signal s_cnt_on_a          : integer range 0 to 24999999 :=0;

--type type_array is array (0 to 2**REG_ADDRESS_WIDTH - 1) of
std_logic_vector(REG_DATA_WIDTH-1 downto 0);
type type_array is array (0 to 3) of
std_logic_vector(REG_DATA_WIDTH-1 downto 0);
signal register_array : type_array := (others => (others => '0'));

begin

  ---------------------------------------------------------------------------
  -- REGISTER WRITING
  ---------------------------------------------------------------------------

  process(CLK_MC_I)
  begin
     if rising_edge(CLK_MC_I) then
        if WR_MC_I = '1' then
              if (to_integer(unsigned(ADDRESS_MC_I))) < 4 then
                 register_array(to_integer(unsigned(ADDRESS_MC_I))) <=
                 DATA_MC_I;
              else
```

```
                S_WR_MC_O <= WR_MC_I ;
                S_ADDRESS_MC_O <= ADDRESS_MC_I;
                S_DATA_MC_O <= DATA_MC_I;
            end if;
        else
          S_WR_MC_O <= '0';
            if s_pulse_on='1' then
              S_ADDRESS_MC_O <=
              std_logic_vector(unsigned(counter_nex_seq) + 4);
            end if;
      end if;
   end if;
end process;




--------------------------------------------------------------------------------
-- TRIGGER REGISTER
--------------------------------------------------------------------------------


process (CLK_MC_I)
begin
   if (rising_edge(CLK_MC_I)) then
      if RST_MC_I = '1' then
         s_tg_meta_1 <= '0';
      else
         s_tg_meta_1 <= TRIGGER_MC_I;
      end if;
   end if;
end process;


process (CLK_MC_I)
begin
   if (rising_edge(CLK_MC_I)) then
      if RST_MC_I = '1' then
         s_tg_meta_2 <= '0';
      else
         s_tg_meta_2 <= s_tg_meta_1;
      end if;
```

```
    end if;
end process;


process (CLK_MC_I)
begin
    if (rising_edge(CLK_MC_I)) then
        if RST_MC_I = '1' then
            s_tg_meta_3 <= '0';
        else
            s_tg_meta_3 <= s_tg_meta_2;
        end if;
    end if;
end process;



-------------------------------------------------------------------------------
-- TRIGGER FLAGS
-------------------------------------------------------------------------------

s_tg_fedge <= not s_tg_meta_2 and s_tg_meta_3;
s_tg_redge <=  s_tg_meta_2 and not s_tg_meta_3;

-------------------------------------------------------------------------------
-- PULSE ON/OFF LOGIC
-------------------------------------------------------------------------------

process (CLK_MC_I)
begin
    if (rising_edge(CLK_MC_I)) then
        if RST_MC_I = '1' then
            s_reg_start <= '0';
            s_pulse_on <= '0';
        else
            if s_reg_start_prev = '1' then
              if s_tg_redge = '1' then -- pulse on
                s_pulse_on  <= '1';
                s_reg_start <= '1';
                s_cnt_on_a <= 0;
              elsif s_reg_pulse_off = '1' then -- pulse off
```

```
--                    s_pulse_on  <= '0';
--                    s_reg_start <= '0';
                    s_reg_pulse_off_a_2 <= '1';
                    s_cnt_on_a <= 0;
                elsif s_reg_pulse_off_a_2 = '1' then --bb2
                      if(s_cnt_on_a=4499)then
                         s_pulse_on  <= '0';
                         s_reg_start <= '0';
                         s_cnt_on_a <= 0;
                         s_reg_pulse_off_a_2 <= '0';
                         --s_cnt_on_a <= s_cnt_on_a +1;
                      elsif (s_cnt_on_a < 4499)then
                         s_cnt_on_a <= s_cnt_on_a +1;
                      end if;
                else
                  s_pulse_on <= s_pulse_on;
                  s_reg_start <= s_reg_start;
                end if;

              else
                s_reg_start <= '0';
                s_pulse_on  <= '0';
                s_cnt_on_a <= 0;
              end if;
           end if;
      end if;
   end process;

   -------------------------------------------------------------------------
   -- PULSE OFF LOGIC AND COUNTER NEXT SEQUENCE
   -------------------------------------------------------------------------

   process(CLK_MC_I)
   begin
      if rising_edge(CLK_MC_I) then
         if RST_MC_I = '1' then
            counter_nex_seq <= (others=>'0');
            s_pulse_off     <=  '0';
         else
            if NEXT_SEQ_MC_I='1' then
               if(counter_nex_seq = s_reg_repeat) then--pulse off
                  counter_nex_seq <= (others=>'0');
```

```
                s_pulse_off     <= '1';
              else
                counter_nex_seq <= counter_nex_seq + 1;
                s_pulse_off     <= '0';
              end if;
            else
              s_pulse_off <= '0';

          end if;
        end if;
    end if;
end process;

-------------------------------------------------------------------------
-- REGISTER PULSE OFF
-------------------------------------------------------------------------

process(CLK_MC_I)
begin
    if rising_edge(CLK_MC_I) then
       s_reg_pulse_off <= s_pulse_off;
    end if;
end process;

-------------------------------------------------------------------------
-- REGISTER OF OUTPUT SIGNALS
-------------------------------------------------------------------------

process(CLK_MC_I)
begin
if rising_edge(CLK_MC_I) then
     if RST_MC_I = '1' then
         REG_START_MC_O  <= '0';          --1bit    used of 16bits
         s_reg_start_prev       <= '0';          --1bit    used of 16bits
         REG_RST_MC_O       <= '0';          --1bit    used of 16bits
        REG_PULSE_WIDTH_MC_O <= (others=>'0');  --  used of 16bits
         REG_IPP_MC_O         <= (others=>'0');  --24bits  used of 32bits
         REG_CODE_MC_O    <= (others=>'0');  --125bits used of 128bits
         s_reg_repeat          <= (others=>'0');  --16bits  used of 16bits
         REG_BAUD_MC_O    <= (others=>'0');  --8bits  used of 16bits
         TRIGGER1_MC_O      <= '0';                -- flag of each pulse
```

```
        else
           REG_START_MC_O<= s_reg_start;
           s_reg_start_prev    <= register_array(0)(0);
           REG_RST_MC_O    <= register_array(0)(1);
           s_reg_repeat        <= register_array(0)(32+REG_REPEAT_WIDTH
                                  -1 downto 32);
           REG_BAUD_MC_O <= register_array(0)
                                    (48+REG_BAUD_WIDTH-1 downto 48);
           REG_IPP_MC_O    <= register_array(1)(REG_IPP_WIDTH-1
                                  DOWNTO 0);
          REG_PULSE_WIDTH_MC_O<= register_array(1)
                                    (32+REG_PULSE_WIDTH-1 DOWNTO 32);

          REG_CODE_MC_O      <= register_array(3)(60 DOWNTO
                                    0)&register_array(2);
          TRIGGER1_MC_O      <= NEXT_SEQ_MC_I;
        end if;
      end if;
    end process;

    WR_MC_O          <= S_WR_MC_O ;
    ADDRESS_MC_O <= S_ADDRESS_MC_O;
    DATA_MC_O        <= S_DATA_MC_O;

end Behavioral;
```

## 8.9. Código VHDL del módulo oscilador controlado numéricamente (NCO)

```
----------------------------------------------------------------------------------
-- Company          :    Jicamarca Radio Observatory
-- Engineer         :    Brayan Lui Estalla Quinteros
--
-- Create Date      :    14:00:03 18/04/2023
-- Design Name      :
-- Module Name      :    nco
-- Project Name     :    Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision         :
-- Additional Comments:
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

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

entity nco is

   generic (
      REG_PHASE_INC_WIDTH : natural := 64; -- phase increment width
      PHASE_WIDTH         : natural := 16  -- phase width
   );
```

```
    port (
       CLK_NCO_I              :  in  std_logic;
       START_NCO_I          :  in  std_logic;
       RST_NCO_I             :   in  std_logic;
       ENABLE_NCO_I : in std_logic;REG_PHASE_INC_NCO_I : in
       std_logic_vector(REG_PHASE_INC_WIDTH-1 DOWNTO
       0):=(others=>'0');
       PHASE_NCO_O        : out std_logic_vector(PHASE_WIDTH-1
       DOWNTO 0):=(others=>'0')
    );
end entity;

architecture behavioral of nco is

signal  phase_acc : std_logic_vector(REG_PHASE_INC_WIDTH-1
downto 0):=(others=>'0');

begin

    --------------------------------------------------------------------------------
    -- PHASE ACCUMULATOR
    --------------------------------------------------------------------------------

    phase_acc_reg: process(CLK_NCO_I)
    begin
     if rising_edge(CLK_NCO_I) then
       if RST_NCO_I = '1' then
          phase_acc <= (others => '0');
       else
          --if START_NCO_I = '1' then
             if ENABLE_NCO_I = '1' then
                phase_acc <= std_logic_vector(unsigned(phase_acc) +
                unsigned(REG_PHASE_INC_NCO_I));
             else
                phase_acc <= (others => '0');
             end if;
          --else
             --phase_acc <= (others => '0');
          --end if;
        end if;
     end if;
    end process phase_acc_reg;
```

```
    ------------------------------------------------------------------------
    -- ASSIGNMENT OF OUTPUT SIGNALS
    ------------------------------------------------------------------------
    PHASE_NCO_O <= phase_acc(REG_PHASE_INC_WIDTH-1 downto
    (REG_PHASE_INC_WIDTH - PHASE_WIDTH));

end behavioral;
```

**8.10. Código VHDL del módulo Tabla de búsqueda del seno(0º) y seno(180º)**

```
----------------------------------------------------------------------------------
-- Company          :     Jicamarca Radio Observatory
-- Engineer         :     Brayan Lui Estalla Quinteros
--
-- Create Date      :     01:57:46 11/02/2023
-- Design Name      :
-- Module Name      :     SinCosLUT
-- Project Name     :     Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision         :
-- Additional Comments:
-- Simple sin/cos LUT
-- Register input/output to allow usual quarter-wave symmetry
--
--
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------


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




entity SinCosLUT is
        generic (
```

```
        PHASE_WIDTH  : natural := 16;
        OUTPUT_WIDTH : natural := 14
        --LUT_SIZE: natural := 14
    );
    port (
        CLK_LUT_I   :   in   std_logic;
        STAR_LUT_I  :  in  std_logic;
        RST_LUT_I     : in std_logic;
        PHASE_LUT_I : in std_logic_vector(PHASE_WIDTH-1
        downto 0);
        SIN_LUT_O       : out std_logic_vector(OUTPUT_WIDTH-1
        downto 0):=(others=>'0');
        SIN180_LUT_O : out std_logic_vector(OUTPUT_WIDTH-1
        downto 0):=(others=>'0')
    );

end entity;

architecture arch of SinCosLUT is

  signal sin_tdata_temp : std_logic_vector(OUTPUT_WIDTH-1 downto
  0):= (others => '0'); --
  signal sin180_tdata_temp : std_logic_vector(OUTPUT_WIDTH-1 downto
  0):= (others => '0');--
  signal cos_tdata_temp : std_logic_vector(OUTPUT_WIDTH-1 downto
  0):= (others => '0');--

    --0 to pi/2 sin look up table
    constant LUT_SIZE : natural := 2**(PHASE_WIDTH-2);
    type lut_array is array(LUT_SIZE-1 downto 0) of
    signed(OUTPUT_WIDTH-1 downto 0);
    function fill_lut return lut_array is
        variable lut : lut_array;
        variable tmp : integer;
        constant SCALE : real := real(2**(OUTPUT_WIDTH-1)) - 1.0;

    begin

        for ct in 0 to LUT_SIZE-1 loop
            tmp := integer( SCALE *
            sin((MATH_PI/2.0)*real(ct)/real(LUT_SIZE)) );
            lut(ct) := to_signed(tmp, OUTPUT_WIDTH);
```

```
            end loop;
            return lut;
        end function;

        --seems this should be constant but then rom_style requires signal
        signal lut : lut_array := fill_lut;
        attribute rom_style : string;
        attribute rom_style of lut : signal is "block";
        signal sin_addr, cos_addr : natural range 0 to
        2**(PHASE_WIDTH-2)-1;
        subtype ADDR_SLICE is natural range PHASE_WIDTH-3 downto 0;
       signal sin_tdata_reg, cos_tdata_reg : signed(OUTPUT_WIDTH-1
       downto 0):=(others=>'0');--

        signal sign_bit : std_logic:= '0';
        signal ones_complement_addr_bit : std_logic:= '0';
        signal sin_sign_bit_d : std_logic := '0';
        signal cos_sign_bit, cos_sign_bit_d : std_logic := '0';

begin

        sign_bit <= PHASE_LUT_I(PHASE_LUT_I'high);
        ones_complement_addr_bit <= PHASE_LUT_I(PHASE_LUT_I'high -
        1);

        sin_port : process(CLK_LUT_I)
            variable lut_data : signed(OUTPUT_WIDTH-1 downto
            0):=(others=>'0');
        begin
            if rising_edge(CLK_LUT_I) then
      if STAR_LUT_I ='1' then
                    --register addr with possible ones complement
                    if ones_complement_addr_bit = '0' then
                        sin_addr <= to_integer(unsigned(PHASE_LUT_I
                                (ADDR_SLICE)));
                    else
                        sin_addr <= to_integer(unsigned(not
                                PHASE_LUT_I(ADDR_SLICE)));
                    end if;

                    --Register output data from BRAM
                    sin_tdata_reg <= lut_data;
```

```
                    lut_data := lut(sin_addr);
                else
            sin_tdata_reg <= (others=>'0');
            lut_data          := (others=>'0');
        end if;
                end if;
end process;


        sin_sign_bit_delay : entity work.DelayLine
        generic map ( DELAY_TAPS => 3)
        port map( clk => CLK_LUT_I, rst => RST_LUT_I, data_in(0) =>
        sign_bit, data_out(0) => sin_sign_bit_d);
        sin_tdata_temp    <= std_logic_vector(sin_tdata_reg) when
        sin_sign_bit_d = '0' else not std_logic_vector(sin_tdata_reg);
        sin180_tdata_temp <= not std_logic_vector(sin_tdata_temp);

   ---------------------------------------------------------------------------
   -- OUTPUT SIGNALS  REGISTERED
   ---------------------------------------------------------------------------

   process(CLK_LUT_I)
   begin
      if (rising_edge(CLK_LUT_I))then
        if STAR_LUT_I ='1' then
                SIN_LUT_O <=  sin_tdata_temp;
                SIN180_LUT_O <= sin180_tdata_temp;
        else
                SIN_LUT_O <= (others=>'0');
                SIN180_LUT_O <= (others=>'0');
        end if;
      end if;
   end process;

end architecture;
```

**8.11. Código VHDL del módulo delay para la Tabla de búsqueda del seno(0º) y seno(180º)**

```
----------------------------------------------------------------------------------
-- Company          :     Jicamarca Radio Observatory
-- Engineer         :     Brayan Lui Estalla Quinteros
--
-- Create Date      :     14:00:03 18/04/2023
-- Design Name      :
-- Module Name      :     SinCosLUT
-- Project Name     :     Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision         :
-- Additional Comments:
-- Shift register delay line with synchronous reset
-- Original authors Colm Ryan and Blake Johnson
--
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity DelayLine is
  generic(
    REG_WIDTH      : natural := 1;
    RESET_VALUE : std_logic := '0';
    DELAY_TAPS     : natural := 1
  );
```

```
  port (
    clk      : in  std_logic;
    rst      : in  std_logic;
    data_in  : in  std_logic_vector(REG_WIDTH-1 downto 0);
    data_out : out std_logic_vector(REG_WIDTH-1 downto 0)
  );
end entity;

architecture arch of DelayLine is

type DELAY_LINE_t is array(DELAY_TAPS-1 downto 0) of
std_logic_vector(REG_WIDTH-1 downto 0);
shared variable delay_line : DELAY_LINE_t := (others => (others =>
RESET_VALUE));
attribute shreg_extract : string;
attribute shreg_extract of delay_line : variable is "TRUE";

begin

   -------------------------------------------------------------------------------
   -- OUTPUT SIGNALS  REGISTERED
   -------------------------------------------------------------------------------

   main : process(clk)
   begin
     if rising_edge(clk) then
       if rst = '1' then
         delay_line := (others => (others => RESET_VALUE));
         data_out <= (others => RESET_VALUE);
       else
         delay_line := delay_line(delay_line'high-1 downto 0) & data_in;
         data_out <= delay_line(delay_line'high);
       end if;
     end if;
   end process;

end architecture;
```

## 8.12. Código VHDL del módulo Modulador OOK - BPSK

```
----------------------------------------------------------------------------------
-- Company          :     Jicamarca Radio Observatory
-- Engineer         :     Brayan Lui Estalla Quinteros
--
-- Create Date      :     14:00:03 18/04/2023
-- Design Name      :
-- Module Name      :     modulator
-- Project Name     :     Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision         :
-- Additional Comments:
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity modulator is
   Generic (
      REG_PULSE_WIDTH : natural := 16;  -- register pulse width
      REG_IPP_WIDTH       : natural := 24;  -- register ipp width
      REG_CODE_WIDTH  : natural := 125;  -- register code width
      REG_BAUD_WIDTH  : natural := 8     -- register baud width
   );
```

```
    port (
       CLK_MOD_I          : in std_logic; --250Mhz
       START_MOD_I        : in std_logic;
       RST_MOD_I          : in std_logic;
       PULSE_WIDTH_MOD_I : in std_logic_vector(REG_PULSE_WIDTH-1
       downto 0);
       IPP_MOD_I        : in std_logic_vector(REG_IPP_WIDTH-1 downto 0);
       CODE_MOD_I           : in std_logic_vector(REG_CODE_WIDTH-1
       downto 0);
       BAUD_MOD_I           : in std_logic_vector(REG_BAUD_WIDTH-1
       downto 0);
       PHASE_SEL_MOD_0 :  out std_logic:='0';
       EN_MUX_MOD_0       :  out std_logic:='0';
       EN_NCO_MOD_0       : out std_logic:='0';--IPP_PULSE
       NEX_SEQ_MOD_0      :  out std_logic:='0';
       FLAG_FASE          :  out std_logic:='0';
       FLAG_EN_MUX        :  out std_logic:='0';
       FLAG_NEX_SEQ       :  out std_logic:='0';
       GATE_MOD_O         :  out std_logic:='0'
    );
end entity;

architecture Behavioral of modulator is

   ---------------------------------------------------------------------------
    -- IPP AND PULSE (ip)
    ---------------------------------------------------------------------------

    --counter ipp (2**24=16777216)
    signal s_cnt_ipp  : integer range 0 to (2**REG_IPP_WIDTH) := 0;
    signal s_cnt_on    : integer range 0 to 24999999:=0;
    signal s_cnt_off  : integer range 0 to 24999999:=0;
    signal s_ip_prev  : std_logic:='0';
    signal s_rf           : std_logic:='0';
    signal s_gate       : std_logic:='0';



    ---------------------------------------------------------------------------
    -- CODE_WIDTH AND PHASE
    ---------------------------------------------------------------------------
```

```
 --counter to change codes (125)
signal s_cnt_ChangeCode : integer range 0 to
(REG_CODE_WIDTH-1) := 0;
--counter sub pulse for each pulse width  ((2**16)/250)-1= 261.144 ==
262
signal s_cnt_SubPulse : integer range 0 to 262 := 0;
signal s_flag_en_mux  :  std_logic:='0';

 ---------------------------------------------------------------------------
 -- NEXT SEQUENCE
 ---------------------------------------------------------------------------

 --counter of next sequence flag (2**16)=65536
signal s_cnt_NextSeq  : integer range 0 to (2**REG_PULSE_WIDTH) := 0;


 ---------------------------------------------------------------------------
 -- LATENCY PULSE WIDTH
 ---------------------------------------------------------------------------

 signal  s_lat_1_en_mux  :  std_logic:='0';
 signal  s_lat_2_en_mux  :  std_logic:='0';
 signal  s_lat_3_en_mux  :  std_logic:='0';
 signal s_lat_4_en_mux  :  std_logic:='0';


begin

 ---------------------------------------------------------------------------
 -- IPP AND PULSE (ip)
 ---------------------------------------------------------------------------

      process (CLK_MOD_I)
      begin
      if (rising_edge(CLK_MOD_I)) then
            if RST_MOD_I = '1' then
       s_cnt_ipp <= 0;
       s_cnt_on  <= 0;
       s_cnt_off <= 0;
       s_ip_prev <= '0';
       s_rf      <= '0';
       s_gate    <= '0';
```

```
else
   if START_MOD_I ='1' then
      if (s_cnt_ipp = to_integer(unsigned(IPP_MOD_I))-1) then
         s_cnt_ipp <= 0;
         s_cnt_on  <= 0;
         s_cnt_off <= 0;
      else
          s_cnt_ipp <= s_cnt_ipp + 1;
          if (s_cnt_ipp < to_integer
           (unsigned(PULSE_WIDTH_MOD_I))+4499) then
            s_ip_prev <= '1';
            s_gate    <= '1';
                    if(s_cnt_on = 4499)then
                       s_cnt_off <= 0;
                       s_rf      <= '1';
                    else
                       s_cnt_on <= s_cnt_on +1;
                    end if;
          else
            s_ip_prev <= '0';
            s_rf      <= '0';
                    if s_ip_prev ='0' then
                                if (s_cnt_off = 4499) then
                                   s_gate <= '0';
                                else
                                   s_cnt_off <= s_cnt_off +1;
                                end if;
                    end if;
         end if;
      end if;
    else
      s_cnt_ipp <= 0;
      s_cnt_on  <= 0;
      s_cnt_off <= 0;
      s_ip_prev <= '0';
      s_rf      <= '0';
      s_gate    <= '0';
    end if;
          end if;
   end if;
   end process;
```

```
EN_NCO_MOD_0 <= s_rf;
GATE_MOD_O     <= s_gate;

--------------------------------------------------------------------------------
-- CODE_WIDTH AND PHASE
--------------------------------------------------------------------------------

      process(CLK_MOD_I)
      begin
      if (rising_edge(CLK_MOD_I)) then
        if RST_MOD_I = '1' then
              s_cnt_ChangeCode <= 0;
              s_cnt_SubPulse <= 0;
              s_flag_en_mux <='0';
      PHASE_SEL_MOD_0 <= '0';
      FLAG_FASE <= '0';
    else
       if START_MOD_I = '1' then
          if s_rf = '1' then
             s_flag_en_mux <='1';
             PHASE_SEL_MOD_0 <= CODE_MOD_I(s_cnt_ChangeCode);
             FLAG_FASE <= CODE_MOD_I(s_cnt_ChangeCode);

             if (s_cnt_SubPulse = to_integer(unsigned(BAUD_MOD_I)))
             then
                s_cnt_SubPulse <= 0; -- counter sub pulse for each pulse
                width finished
                if s_cnt_ChangeCode = 124 then
                   s_cnt_ChangeCode <= 0; -- counter to change codes
                else
                   s_cnt_ChangeCode <= s_cnt_ChangeCode + 1; -- counter
                   to change codes
                end if;
             else
                -- counter sub pulse for each pulse width  and  counter to
                   change codes
                s_cnt_SubPulse <= s_cnt_SubPulse +1;
             end if;
```

```
            else
               s_cnt_ChangeCode      <= 0;
               s_cnt_SubPulse        <= 0;
               s_flag_en_mux         <= '0';
               PHASE_SEL_MOD_0   <= '0';
               FLAG_FASE             <= '0';
            end if;
                  end if;
               end if;
      end if;
      end process;

-------------------------------------------------------------------------------
-- NEXT SEQUENCE
-------------------------------------------------------------------------------

repeat:process(CLK_MOD_I)
begin
   if rising_edge(CLK_MOD_I) then
      if  RST_MOD_I='1' then
          NEX_SEQ_MOD_0 <= '0';
          FLAG_NEX_SEQ  <= '0';
          s_cnt_NextSeq <= 0 ;
      else
          if s_rf = '1' then
             if (s_cnt_NextSeq =
             (to_integer(unsigned(PULSE_WIDTH_MOD_I)))-1) then
                s_cnt_NextSeq <= 0 ;
                NEX_SEQ_MOD_0 <= '1';
                FLAG_NEX_SEQ  <= '1';
             else
                s_cnt_NextSeq <= s_cnt_NextSeq + 1;
                NEX_SEQ_MOD_0 <= '0';
                FLAG_NEX_SEQ  <= '0';
             end if;
          else
             NEX_SEQ_MOD_0 <= '0';
             FLAG_NEX_SEQ  <= '0';
          end if;
      end if;
   end if;
```

```
    end process;

    ------------------------------------------------------------------------------
    -- LATENCY PULSE WIDTH
    ------------------------------------------------------------------------------

    lat_1_pulse_width:process(CLK_MOD_I)
    begin
       if rising_edge(CLK_MOD_I) then
          s_lat_1_en_mux <= s_flag_en_mux;
       end if;
    end process;


    lat_2_pulse_width:process(CLK_MOD_I)
    begin
       if rising_edge(CLK_MOD_I) then
          s_lat_2_en_mux <= s_lat_1_en_mux;
       end if;
    end process;

    lat_3_pulse_width:process(CLK_MOD_I)
    begin
       if rising_edge(CLK_MOD_I) then
          s_lat_3_en_mux <= s_lat_2_en_mux;
       end if;
    end process;

    lat_4_pulse_width:process(CLK_MOD_I)
    begin
       if rising_edge(CLK_MOD_I) then
          FLAG_EN_MUX <= s_lat_3_en_mux;  --EN_MUX_MOD_0
          EN_MUX_MOD_0 <= s_lat_3_en_mux; --EN_MUX_MOD_0
       end if;
    end process;

end Behavioral;
```

## 8.13. Código VHDL del módulo Multiplexor

```
----------------------------------------------------------------------------------
-- Company           :     Jicamarca Radio Observatory
-- Engineer          :     Brayan Lui Estalla Quinteros
--
-- Create Date       :     14:00:03 18/04/2023
-- Design Name       :
-- Module Name       :     multiplexor
-- Project Name      :     Ionosonde
-- Target Devices    :
-- Description:
--
-- Revision          :
-- Additional Comments:
----------------------------------------------------------------------------------
--  _____________
--  /   /\/   /
-- /___/  \  /    FPGA       : Xilinx
-- \   \   \/     Version    : 19.1
--  \   \
--  /   /
-- /___/   /\
-- \   \  /  \
--  \___\/\___\
----------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity mux is
        generic (
                OUTPUT_WIDTH : natural := 14
        );

    port (
      CLK_MUX_I   :   in   std_logic;
      START_MUX_I :  in  std_logic;
      RST_MUX_I   :   in   std_logic;
      CODE_I          :  in  std_logic;
      CODE_VLD_I   : in  std_logic;
```

```
    SIN_DAC_I        : in  std_logic_vector(OUTPUT_WIDTH-1 downto 0);
    SIN_180_DAC_I : in  std_logic_vector(OUTPUT_WIDTH-1 downto 0);
    MOD_SIGNAL     : out std_logic_vector(OUTPUT_WIDTH-1 downto
    0):=(others=>'0')
       );

end entity;

architecture behavioral of mux is

  signal s_mod_signal : std_logic_vector(OUTPUT_WIDTH-1 downto
  0):=(others=>'0');

begin

  ---------------------------------------------------------------------------
  -- MUX
  ---------------------------------------------------------------------------

  process (CLK_MUX_I)
  begin
    if rising_edge(CLK_MUX_I) then
      if RST_MUX_I ='1' then
         s_mod_signal <= (others => '0');
      else
          --if START_MUX_I ='1' then
             if CODE_VLD_I='1' then
                 if CODE_I ='1' then
                 s_mod_signal <= SIN_DAC_I;
                 else
                    s_mod_signal <= SIN_180_DAC_I;

                 end if;
              else
                s_mod_signal <= (others=>'0');
              end if;
           --else
             --s_mod_signal <= (others=>'0');
           --end if;
       end if;
    end if;
   end process;
```

```
    ------------------------------------------------------------------------
    -- ASSIGNMENT OF OUTPUT SIGNALS
    ------------------------------------------------------------------------

    MOD_SIGNAL <= s_mod_signal(OUTPUT_WIDTH-1 downto 0);

end behavioral;
```

## 8.14. Código Verilog del módulo de sincronismo

```
----------------------------------------------------------------------------------
-- Company          :    Jicamarca Radio Observatory
-- Engineer         :    Brayan Lui Estalla Quinteros
--
-- Create Date      :    14:00:03 18/04/2023
-- Design Name      :
-- Module Name      :   synchronism
-- Project Name     :    Ionosonde
-- Target Devices   :
-- Description:
--
-- Revision       :
-- Additional Comments:
--
-- Original authors Matej Oblak - Red Pitaya
--
----------------------------------------------------------------------------------


module red_pitaya_hk(
  // system signals
  input              clk_i      ,     // clock
  input              rstn_i     ,     // reset - active low
  // global configuration
  input              pll_sys_i,      // system clock
  input              pll_ref_i,      // reference clock
  output              pll_hi_o,      // PLL high
  output              pll_lo_o       // PLL low
);

//---------------------------------------------------------------------------------
//
//  Simple FF PLL

// detect RF clock
reg  [16-1:0] pll_ref_cnt = 16'h0;
reg  [ 3-1:0] pll_sys_syc  ;
reg  [21-1:0] pll_sys_cnt  ;
reg           pll_sys_val  ;
reg           pll_cfg_en   ;
```

```
//wire [32-1:0] pll_cfg_rd   ;

//assign pll_cfg_en = 1'b1;

always @(posedge pll_ref_i) begin
pll_ref_cnt <= pll_ref_cnt + 16'h1;
end

always @(posedge clk_i)
if (rstn_i == 1'b1) begin//0
pll_sys_syc <= 3'h0 ;
  pll_sys_cnt <= 21'h100000;
  pll_sys_val <= 1'b0 ;
end else begin
  pll_sys_syc <= {pll_sys_syc[3-2:0], pll_ref_cnt[14-1]} ;

  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
    pll_sys_cnt <= 21'h1;
  else if (!pll_sys_cnt[21-1])
    pll_sys_cnt <= pll_sys_cnt + 21'h1;

  // pll_sys_clk must be around 102400 (125000000/(10000000/2^13))
  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
    pll_sys_val <= (pll_sys_cnt > 102385) && (pll_sys_cnt < 102415) ;
  else if (pll_sys_cnt[21-1])
    pll_sys_val <= 1'b0 ;
end


reg  pll_ff_sys ;
reg  pll_ff_ref ;
wire pll_ff_rst ;
wire pll_ff_lck ;

// FF PLL functionality
always @(posedge pll_sys_i or negedge pll_ff_rst) begin
  if (!pll_ff_rst)  pll_ff_sys <= 1'b0 ;
  else              pll_ff_sys <= 1'b1 ;
end
always @(posedge pll_ref_i or negedge pll_ff_rst) begin
  if (!pll_ff_rst)  pll_ff_ref <= 1'b0 ;
  else              pll_ff_ref <= 1'b1 ;
```

```
end
assign pll_ff_rst = !(pll_ff_sys && pll_ff_ref) ;
assign pll_ff_lck = (!pll_ff_sys && !pll_ff_ref);

assign pll_lo_o = !pll_ff_sys && ( pll_sys_val);
assign pll_hi_o =  pll_ff_ref || (!pll_sys_val);

//assign pll_lo_o = !pll_ff_sys && ( pll_sys_val &&  pll_cfg_en);
//assign pll_hi_o =  pll_ff_ref || (!pll_sys_val || !pll_cfg_en);


// filter out PLL lock status
reg  [21-1:0]  pll_lck_lcnt  ;
reg  [21-1:0]  pll_lck_hcnt  ;
reg  [ 4-1:0] pll_lck_sts  ;

always @(posedge clk_i)
if (rstn_i == 1'b1) begin//0
pll_lck_lcnt <= 21'h0 ;
pll_lck_hcnt <= 21'h0 ;
pll_lck_sts <=  2'b0 ;
end else begin

  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
    pll_lck_lcnt <= 21'h1;
  else if (pll_lo_o)
    pll_lck_lcnt <= pll_lck_lcnt + 21'h1;

  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
    pll_lck_hcnt <= 21'h1;
  else if (!pll_hi_o)
    pll_lck_hcnt <= pll_lck_hcnt + 21'h1;


  // pll_lck_cnt threshold 70% of whole period
  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
    pll_lck_sts[0] <= (pll_lck_lcnt > 21'd80000) && (pll_lck_hcnt >
21'd80000);

  pll_lck_sts[1] <= pll_lck_sts[0] && pll_sys_val;

  if (pll_sys_syc[3-1] ^ pll_sys_syc[3-2])
```

```
    pll_lck_sts[3:2] <= {(pll_lck_lcnt > 21'd80000), (pll_lck_hcnt >
21'd80000)};
end

assign pll_cfg_rd = {{32-14{1'h0}}, pll_lck_sts[3:2], 3'h0,pll_lck_sts[1],
3'h0,pll_sys_val, 3'h0,pll_cfg_en};

endmodule
```