UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO, IMPLEMENTACIÓN Y PRUEBAS DE VOLTÍMETRO VECTORIAL EN PLATAFORMA BASADA EN FPGA PARA MEDICIONES DE HOLOGRAFÍA DEL TELESCOPIO CCAT-PRIME

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

SEBASTIÁN ANTONIO JORQUERA TAPIA

PROFESOR GUÍA:
RICARDO FINGER CAMUS

MIEMBROS DE LA COMISIÓN:
PATRICIO MENA MENA
WALTER MAX-MOERBECK ASTUDILLO

Este trabajo ha sido financiado por ANID a través de sus fondos CATA-Basal PFB-06 y ALMA 31180005

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: SEBASTIÁN ANTONIO JORQUERA TAPIA
FECHA: 2020
PROF. GUÍA: RICARDO FINGER CAMUS

DISEÑO, IMPLEMENTACIÓN Y PRUEBAS DE VOLTÍMETRO VECTORIAL EN PLATAFORMA BASADA EN FPGA PARA MEDICIONES DE HOLOGRAFÍA DEL TELESCOPIO CCAT-PRIME

El proyecto CCAT-prime consiste en un proyecto colaborativo encabezado por las Universidad de Cornell, Universidad de Colgne, Universidad de Bonn, el Instituto Max-Planck y el CATC (Canadian Atacama Telescope Consortium). El proyecto tiene como fin la construcción y operación de un telescopio de arquitectura *cross-dragonian* en el cerro Chajnantor a 5600 metros sobre el nivel del mar, funcionando en las longitudes de onda $350 - 3100\mu m$ y cuya principal característica es un amplio campo de visión de hasta 8 grados.

Para una medición con una alta eficiencia de antena es necesario caracterizar los errores sobre la superficie del telescopio, para ello se emplea la técnica de holografía que utiliza la relación existente entre la transformada de Fourier de las corrientes superficiales y el patrón de radiación de la antena. Así la técnica consiste en medir el patrón de radiación e invertir la relación, obteniendo los errores en la superficie de la antena lo que permite realizar las correcciones necesarias. La medición del patrón de radiación se hace utilizando un voltímetro vectorial para obtener los valores de magnitud y fase del patrón de la antena.

En esta memoria se describe el proceso de diseño, implementación y pruebas de un voltímetro vectorial de alto rango dinámico utilizando técnicas de procesamiento de señales digitales en una plataforma basada en una FPGA de alto rendimiento. De manera simultanea se diseña, implementa y prueba un sistema de etiquetado temporal basado en el protocolo IRIG utilizando técnicas clásicas de diseño de sistemas digitales

Las especificaciones requeridas para el sistema de voltímetro vectorial son un rango dinámico sobre los 70 dB, un espaciado entre canales bajo los 10 kHz para señales de frecuencias en el rango 50 ± 3 MHz. Para el sistema de etiquetado temporal se requiere una resolución de etiquetado temporal mejor que 0,1 ms. Además, el sistema completo debe tener un tiempo entre dos salidas consecutivas (tasa de entrega de datos) de 1 ms.

Para caracterizar el funcionamiento de los sistemas se realizan diversas pruebas experimentales que entregan un rango dinámico de 75,8 dB en el peor desempeño, con un ancho de canal de 8,24 kHz para señales en la vecindad de 50 MHz considerando una tasa de entrega de datos de 1,21 ms. Para el etiquetado de tiempo se obtiene una resolución de etiquetado de $0{,}957\mu$s.

Para la biblioteca de Babel,

donde este documento estaba escondido

aún antes de que se escribiese.

Agradecimientos

Agradecimientos a toda la gente del laboratorio de Ondas Milimétricas, que han sido parte importante de esta memoria. Gracias por ser un rinconcito donde se puede encontrar conocimiento y buena onda por montones.

Gracias a mis padres y mi familia por su amor y cariño. Gracias por soportarme todo este tiempo de estudio y por ser incondicionales en todo sentido.

Gracias a mis amigos, por el tiempo que compartimos, por lo mucho que aprendí de ellos sin querer y lo mucho que compartieron sin pedir nada a cambio.

En fin, gracias a todos quienes han sido parte de mi vida. Porque al final todo lo que he aprendido o realizado ha sido en la medida de las opciones que se han presentado en mi camino, las cuales aprecio y agradezco profundamente.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

Debido a los movimientos de la Tierra hay zonas del universo que no pueden ser observadas en ciertas locaciones geográficas en determinados momentos del año. Es por ello que los distintos observatorios se han repartido alrededor del mundo, y en particular el norte de Chile ha mostrado ser uno de los sectores más favorables para la colocación de centros de observación en el hemisferio sur.

Dentro de la nueva generación de telescopios se encuentra Cerro Chajnantor Atacama Telescope prime (CCAT-prime) que espera estar operativo en el año 2021 en el cerro Chajnantor de la región de Atacama, y cuyo diseño está enfocado en la medición del efecto Sunyaev-Zel'dovich, que es un efecto producto del paso del fondo de radiación cósmica por una nube molecular altamente ionizada y la medición líneas espectrales de CII y CO, importantes trazadores de nubes moleculares y de formación estelar. Para ello el telescopio consiste en dos reflectores en configuración "Cross-dragonian off-axis" con un error de superficie efectiva (HWFE) bajo los 7μm RMS operando en las longitudes de onda en el rango $(350 - 3100)\mu$m, siendo capaz de iluminar sobre 10^5 detectores.

Al igual que otros telescopios, los reflectores de CCAT-prime están compuestos de paneles unidos a estructuras mecánicas que permiten el ajuste de posición de cada uno de ellos. Para dar con el posicionamiento óptimo se requiere de una metodología que permita cuantificar los errores de la disposición de dichos paneles. Una de las técnicas para realizar está labor es la holografía, que basa su operación en la relación existente entre las corrientes superficiales de la antena y el patrón de radiación en campo lejano. Sin esta calibración el telescopio no podría funcionar en su estado óptimo.

Es en este escenario que el Laboratorio de Ondas Milimétricas, ubicado en las dependencias del departamento de Astronomía de la Universidad de Chile, participa en el proyecto de CCAT-prime generando instrumentos necesarios para el proceso de holografía. En particular, esta memoria trata el desarrollo, implementación y pruebas de un voltímetro vectorial de alto rango dinámico que permita medir la amplitud y diferencia en fase de manera de caracterizar el patrón de radiación.

El desarrollo de este proyecto se basa en una cadena de bloques que realizan distintas técnicas de procesamiento digital implementados en una *Field Programmable Gate Array*(FPGA) de alto rendimiento, que permite generar hardware dedicado a tareas especificas.

Hipótesis

La hipótesis de este trabajo es que es posible implementar un voltímetro vectorial de alto rango dinámico con un sistema de etiquetado de tiempo incluido utilizando plataformas basadas en la tecnología Field Programmable Gate Array. El uso de este tipo de plataforma permitiría generar soluciones cuya relación costo-beneficio sea mejor que las opciones disponibles en el mercado para este problema en particular.

Objetivos Generales

El objetivo general de esta memoria es el diseño, implementación y pruebas de un voltímetro vectorial utilizando plataformas basadas en la tecnología Field Programmable Gate Array.

El sistema debe cumplir con los siguientes objetivos específicos enumerados a continuación.

1. Rango dinámico sobre 70 dB tomando como umbral un error de 1° en la fase.
2. Resolución de canal de 10 kHz.
3. Tiempo entre dos salidas consecutivas (tasa de entrega de datos) de 1 ms.
4. Resolución de etiquetado temporal mejor que 0,1 ms.
5. Las frecuencias de las señales entrantes se encuentran en el rango 50 ± 3 MHz

Alcances

Los alcances de este trabajo consisten en diseñar e implementar los subsistemas de voltímetro vectorial y de etiquetado de tiempo que cumplan los requerimientos enunciados anteriormente. Para la constatación del cumplimiento de dichas especificaciones se deben idear pruebas que permitan la caracterización del sistema.

Estructura del documento

Los capítulos de este documento están ordenados en el siguiente formato. E l capítulo 1 contiene el marco teórico con la información necesaria para el desarrollo del trabajo, colocando especial énfasis en las técnicas de procesamiento de señales utilizadas. En el capítulo 2 se describe el proceso de diseño de los subsistemas que componen al voltímetro vectorial que son implementados en la plataforma ROACH2. El capítulo 3 enumera una serie de pruebas realizadas sobre los sistemas implementados de manera de caracterizar y evaluar si se cumplen o no los requerimientos. Finalmente, este documento cierra con una sección de conclusiones en las cuales se resumen los principales resultados obtenidos, se revisa el cumplimiento de los objetivos y se dan directrices de posibles trabajos futuros.

Capítulo 1

Marco Teórico

1.1. Motivación

En la subsección Holografía se introduce el concepto de holografía para el campo de las antenas y se explica la necesidad del uso de un voltímetro vectorial para la realización de esta técnica de calibración.

En la subsección Field Programmable Gate Array se describe la tecnología *Filed Programmable Gate Array*, haciendo énfasis en su utilización como herramienta de procesamiento de señales.

En la subsección Metodología de diseño de un sistema digital se presentan metodologías clásicas de diseño de sistemas digitales.

1.1.1. Holografía

Para el campo de las antenas, la holografía es una técnica que permite caracterizar los errores existentes en la superficie del receptor. Estos errores están relacionados con una perdida en la eficiencia de la antena, lo que a su vez se traduce en una reducción en la sensibilidad comprometiendo por tanto su desempeño. Por tanto, para aplicaciones que requieren un funcionamiento óptimo es necesaria una etapa de calibración donde se identifiquen las zonas que contienen errores y se realicen las correcciones pertinentes.

Las relaciones existentes entre el patrón de radiación de campo lejano ($\vec{E}_{far\,\mathrm{field}}$) de una antena y sus corrientes superficiales ($\vec{J}$) pueden verse en las ecuaciones 1.1 y 1.2 donde j es la unidad imaginaria, $k = 2\pi/\lambda$ siendo λ la longitud de onda, η la impedancia del medio, J la corriente superficial inducida y las coordenadas r y r' son tomadas tal como se observan en la figura 1.1 [1]. Las expresiones T_θ y T_ϕ corresponden a las componentes en las coordenadas esféricas $\hat{\theta}$ y $\hat{\phi}$ respectivamente.

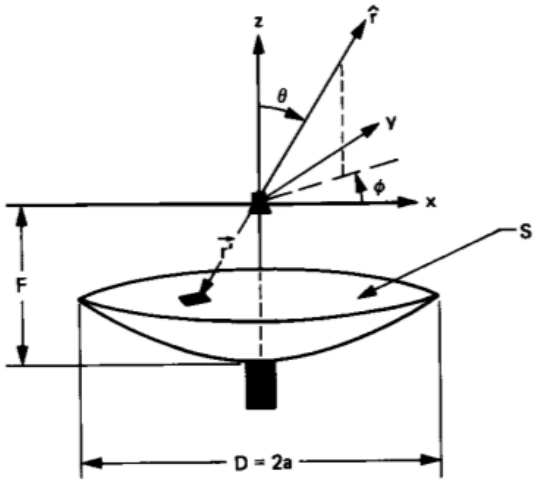


Figura 1.1: Sistema coordenado para ecuaciones 1.1, 1.2, 1.4 y 1.5 [1].

$$E_{farfield} = -jk\eta \frac{e^{-jkr}}{4\pi r}(T_\theta \hat{\theta} + T_\phi \hat{\phi}) \tag{1.1}$$

$$T(\theta, \phi) = \int_S \vec{J}(r') e^{jkr' \cdot \hat{r}} dS \tag{1.2}$$

$$\tag{1.3}$$

Haciendo el cambio de coordenadas $r \cdot r' = z' \cos(\theta) + ux' + vy'$ con $u = \sin(\theta)\cos(\phi)$ y $v = \sin(\theta)\sin(\phi)$ a la ecuación 1.2 y luego utilizando una expansión en series de Taylor se obtienen las expresiones en las ecuaciones 1.4 y 1.5. [1]

$$T(u, v) = \sum_{p=0}^{\infty} \frac{1}{p!}[-jk(1 - cos(\theta))]T_p \tag{1.4}$$

$$T_p = \int_S z'^p J(x', y') e^{jkz'} e^{jk(ux' + vy')} dx' dy' \tag{1.5}$$

Pese a que las expresiones que se obtienen no son muy amigables, se puede notar que a primer orden $E_{farfield} \propto T(u, v) \propto T_0$, y a su vez es posible identificar la expresión de la ecuación 1.5 como una transformada de Fourier en dos dimensiones donde (u, v) toman el rol de parámetros de frecuencia.

Resumiendo, a primer orden se tiene $E_{farfield} \propto \mathcal{F}\{J\}$ donde $\mathcal{F}\{\cdot\}$ es la trasformada de Fourier. Con este resultado es posible medir el patrón de radiación de la antena, invertir la relación explicita en las ecuaciones 1.1, 1.1, 1.4 y 1.5 para encontrar los errores de las corrientes superficiales. Este proceso se encuentra representado en la figura 1.2.

[1] si se desea observar el desarrollo completo se puede encontrar en [1]

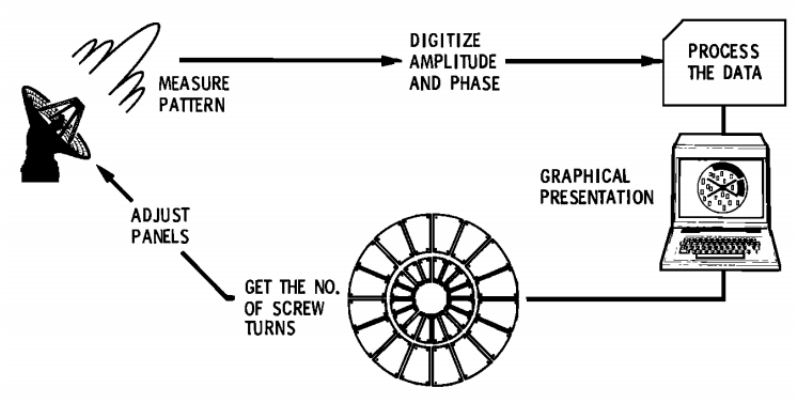


Figura 1.2: Ejemplo del proceso de Holografía. Se mide el patrón de radiación, se computan los errores superficiales de la antena y se procede a realizar los ajustes en los paneles [1]

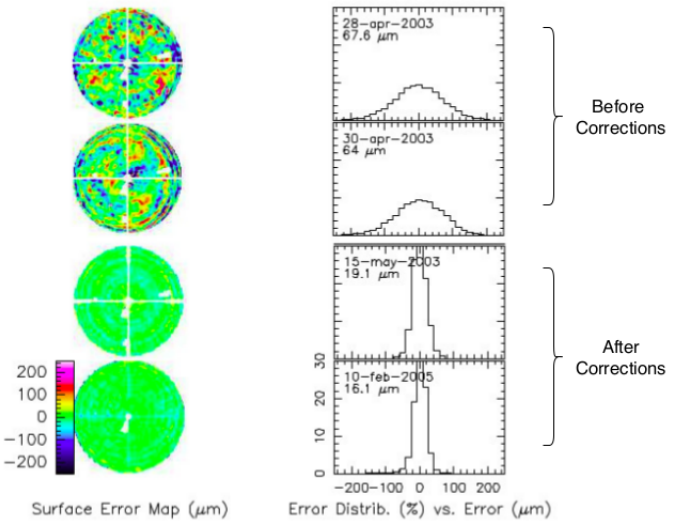


Figura 1.3: Ejemplo de cálculos de errores superficiales antes y después de aplicadas las correcciones utilizando método de holografía.[2]

Las mediciones de patrón de radiación se realizan utilizando una fuente emisora que irradia sobre la antena que se desea calibrar con una frecuencia determinada. Las mediciones de la antena se hacen utilizando voltímetros vectoriales, que entregan información de amplitud y fase de la señal recibida. Además de la antena que se desea calibrar también se utiliza una antena auxiliar de manera de tener una referencia para la fase y la magnitud medidas.

Una razón matemática para la utilización de voltímetros vectoriales en este tipo de mediciones es que para invertir correctamente una transformada de Fourier es necesario tener información tanto de la amplitud y de la fase [12].

En la figura 1.3 se muestran mediciones de los errores superficiales antes y después de correcciones sucesivas mediante el uso de holografía.

Para el caso de CCAT-prime la propuesta consiste en realizar el proceso de holografía

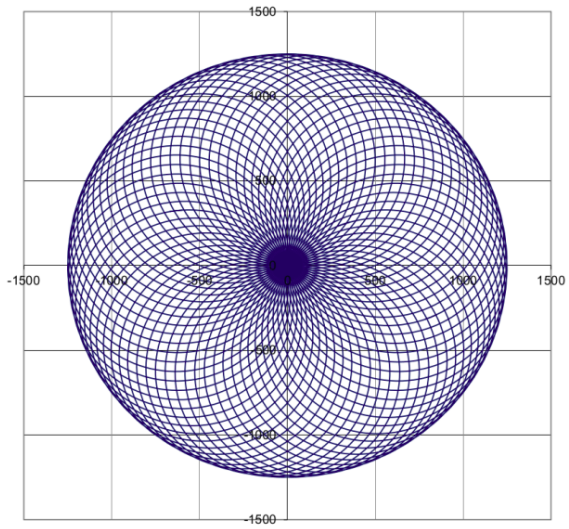


Figura 1.4: Orbita epicíclica para escaneo del patrón de radiación [3].

mientras el telescopio se encuentra en movimiento, de manera que los motores se mantengan operando a rapidez constante [3]. A esta técnica se le denomina *"On the fly holography"*. Una de las órbitas propuestas para el proceso de holografía se muestra en la figura 1.4 que corresponde a una orbita epicíclica. Una de las ventajas que entrega este tipo de orbitas es que se tienen múltiples pasos por el centro que tiene un valor conocido, con lo que se puede realizar una constatación de que las mediciones tomadas previamente han sido correctas y de no ser así realizar las acciones adecuadas.

La realización *"On the fly"* obliga a tener una mapeo entre las mediciones tomadas con el voltímetro vectorial y la posición en que se está midiendo el patrón. Una de las formas de realizar ese mapa es colocar una etiqueta temporal a cada valor entregado por el voltímetro vectorial y con ello asociar cada medición con la posición en la que se encontraba la antena en dicho momento. En particular, para CCAT-prime esto se traduce en un requerimiento de un tiempo de 1 ms entre cada par de valores entregados por el voltímetro vectorial (lo que denominaremos tasa de entrega de datos) y una resolución de marcado temporal mejor a 0,1 ms.

1.1.2. Field Programmable Gate Array

Las *Field Programmable Gate Array* (FPGA) son circuitos integrados programables que contiene bloques lógicos que pueden ser configurados e interconectados según las necesidades del programador. En la figura 1.5 se puede ver un diagrama básico de una FPGA, que consiste en una matriz de bloques lógicos interconectados.

A diferencia de otros circuitos programables como procesadores, las FPGAs son inherentemente Hardware programable debido a que no existe un set de instrucciones prefabricadas sino que es el programador mismo quien elige como interconectar los bloques lógicos que forman el chip y no existe un sistema central preestablecido que se encargue de verificar la ejecución de los procesos, lo que disminuye las latencias a la hora de ejecutar una tarea. Además, programar a nivel de compuerta lógica posibilita generar procesos que se ejecuten

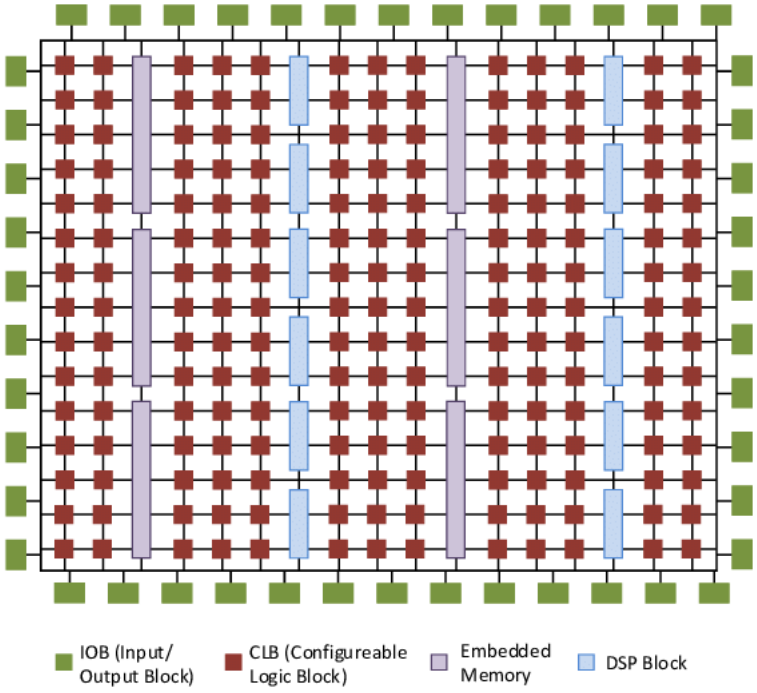


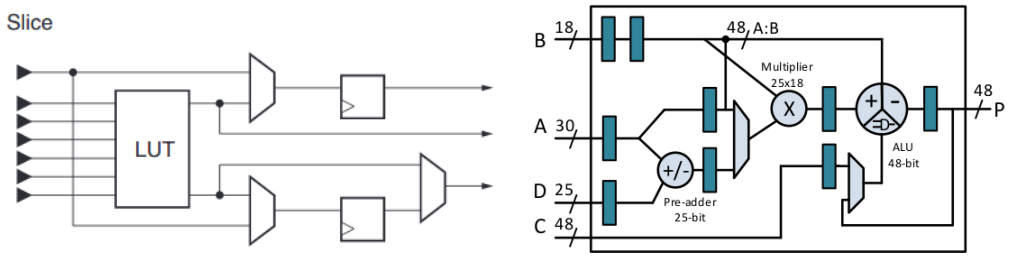
Figura 1.5: Ejemplo de arquitectura de una FPGA [4].

de manera paralela, lo que significa otra ventaja sobre los sistemas digitales que se ejecutan secuencialmente. Estas características hacen de las FPGAs herramientas atractivas para procesamiento de señales en tiempo real.

El poder de computo que han alcanzado las FPGAs ha permitido migrar algunos equipos analógicos a su contra parte digital, accediendo a las ventajas que entregan los sistemas digitales, como por ejemplo la depuración de modelos sin incurrir en etapas constructivas, fácil replicación de sistemas y re-utilización de Hardware entre otros.

El bloque elemental de las FPGAs son los bloques de lógica configurable (CLB) que de manera genérica consisten en tablas de consultas (*look up tables* o LUT). A partir de una LUT puede generarse cualquier tipo de lógica booleana solo viéndose limitada su funcionalidad a la cantidad de bits que maneja, es por ello que las FPGAs permiten la interconexión de grupos de CLB de manera de formar bloques lógicos de mayor complejidad. Además, es usual agregar un elemento de memoria en los CLB como un *flip-flop D*, lo que entrega aún más flexibilidad en su uso.

Aunque los CLBs permiten emular cualquier función lógica la implementación de estos sistemas no siempre es la más eficiente. Es por ello que las FPGAs vienen con bloques complementarios a los CLBs, donde destacan los bloques de memoria *Block RAM* (BRAM) y los bloques de procesamiento digital (DSP *slice*) que permiten generar multiplicaciones y acumulaciones que son operaciones recurrentes en el procesamiento de señales. En las figuras 1.6a y 1.6b se muestran los diagramas básicos de un CLB y un bloque DSP respectivamente.



(a) Diagrama básico de CLB [13]. (b) Diagrama básico de bloque DSP [4].

1.1.3. Metodología de diseño de un sistema digital

Pese a que las metodologías de diseño de cualquier proyecto son variadas y están en constante proceso de revisión en el desarrollo del proyecto, es posible agruparlas en dos clases dependiendo de la manera en la que se aborda la problemática a solucionar.

Bottom up es un método en el que se tiene una descripción sumamente detallada de como deben comportarse los elementos básicos que luego se enlazan para formar el sistema completo.

Top down, por otro lado, se enfoca en resumir el comportamiento del sistema sin especificar los detalles y a partir de dicha descripción se van dilucidando los bloques fundamentales.

Independiente de la metodología ha utilizar, el diagrama básico de un sistema complejo generalmente sigue la configuración *master-slave* donde existe un subsistema controlado (*slave*) y un subsistema controlador (*master*). En este tipo de arquitectura el sistema controlador se encarga de la toma de decisiones, el flujo del programa y debe entregar las señales adecuadas a los sistemas controlados de manera que el subsistema controlado realice una tarea determinada.

Como se menciona en la subsección Field Programmable Gate Array, una FPGA es un dispositivo que no posee una estructura de control pre-establecida con lo que al momento de implementar un algoritmo en este tipo de plataformas toma especial importancia el diseño del subsistema controlador que es quien en último término determina si el sistema en su totalidad se comporta tal como se desea. Es por ello que se han desarrollado una serie de técnicas que permiten esbozar los pasos que se deben seguir para crear un controlador.

De manera sumamente genérica para implementar un algoritmo complejo al interior de una FPGA es necesario descomponer el algoritmo en sus bloque de control más básicos, los que suelen denominarse estados. Los estados pueden realizar dos acciones:

1. Dependiendo de las señales de entrada y del estado actual puede realizar una transición hacia otro estado
2. Generar señales por los puertos del controlador, con la idea de que estas señales desencadenen una acción en el sistema controlado.

En la figura 1.7 se puede ver un ejemplo del diagrama de una máquina de estados finitos.

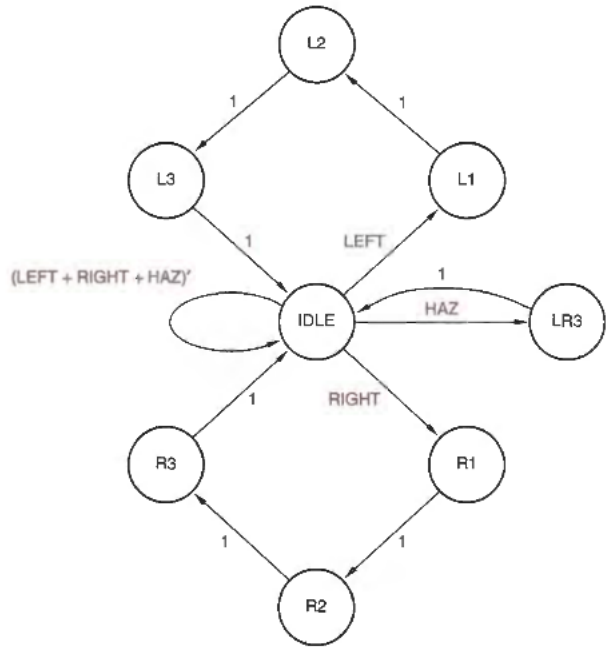


Figura 1.7: Ejemplo de un diagrama de una máquina de estado típica [5]

La principal motivación para llevar un algoritmo a su representación de máquina de estados es que cada estado puede ser mapeado con un valor binario, con lo que el problema de transición de un estado hacia otro se puede implementar como una tabla de verdad, que puede ser implementado con lógica booleana de manera sencilla.

Aunque el mapa de estados del problema es la meta final existen múltiples maneras de aproximarse a este objetivo. En particular en este documento se sigue la metodología propuesta en [6] con ciertas modificaciones. El método utilizado es de la clase *top-down* y se puede reducir a las siguientes fases:

1. **Diagrama de Flujo:** A partir del comportamiento que se desea que posea el sistema se genera un algoritmo que es representado mediante un diagrama de flujo.
2. **Diagrama de bloques del sistema:** Se reconocen los bloques que son necesarios para implementar el sistema. Además se definen las entradas y salidas de cada bloque.
3. **Diseño bloques que conforman el sistema controlado:** Se crean los bloques necesarios para el sistema controlado siguiendo las normas para las entradas y salidas que se tomaron en el punto 2.
4. **Diagrama de flujos de los bloques del sistema controlador:** A partir del diagrama de flujo se generan diagramas de flujo detallados para los bloques del subsistema controlador.
5. **Diagrama de estados de bloques en el sistema controlador:** Tomando como punto de partida los diagramas de flujos del controlador es posible reducir la complejidad del algoritmo en una serie de estados internos y respuestas del controlador ante determinadas entradas. Al diagrama que representa la transición y respuestas de la máquina se le denomina diagrama de estados.

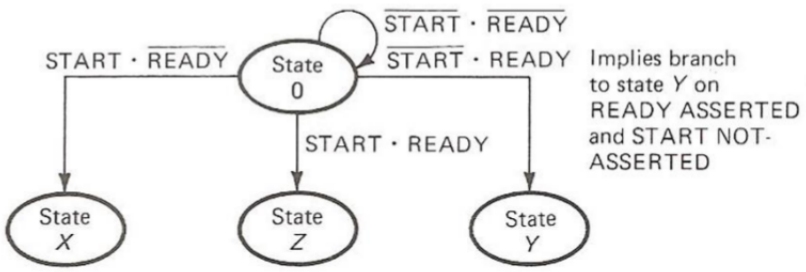


Figura 1.8: Ejemplo de simbología utilizada en diagrama de estado [6].

6. **Implementación del HDL de los controladores:** A partir de un diagrama de estados se puede construir de manera sencilla el HDL asociado que es lo que se sintetiza en el hardware de la FPGA.

Para un sistema complejo es usual que los controladores posean una gran cantidad de entradas y salidas, con ello es necesario adoptar una visualización diferente a la estándar utilizada por las máquinas de Moore o de Mealey que permita facilitar la lectura de transiciones y respuesta, y que además permita colocar condiciones de transiciones atípicas en los diagramas de máquinas de estado básica. La simbología que se utiliza está basada en la propuesta en [14] para los diagramas MDS (Mnemonic Documented State diagram).

En las figuras 1.8 y 1.9 se tiene un ejemplo de la simbología utilizada. Cada globo en las figuras representa un estado determinado, la evolución de los estados está dada por las flechas en el diagrama, las expresiones sobre las flechas representan las entradas del controlador y son las expresiones que se deben tener para que la transición de estados dada por la flecha tome lugar.

Por ejemplo, en la figura 1.8 dado el estado **0** existe una transición al estado **X** si la entrada **START** toma el valor 1, la entrada **READY** toma el valor 0 y el resto de las entradas (de existir) no importa su valor.

Las expresiones en 1.9 que se encuentran a los costados de los estados representan las salidas que se activan o desactivan dado que la máquina se encuentra en dicho estado. En el ejemplo 1.9 el símbolo **GATCMD**$\uparrow$ denota la activación de la salida **GATCMD** dado que la máquina se encuentra en el estado **X**. Otra nomenclatura es el símbolo **SOC**$\uparrow\downarrow=$ $(STATE\ S \cdot EOC \cdot \overline{CLK})$ que representa una activación y desactivación de la señal **SOC** si es que la expresión $(STATE\ S \cdot EOC \cdot \overline{CLK})$ es verdadera.

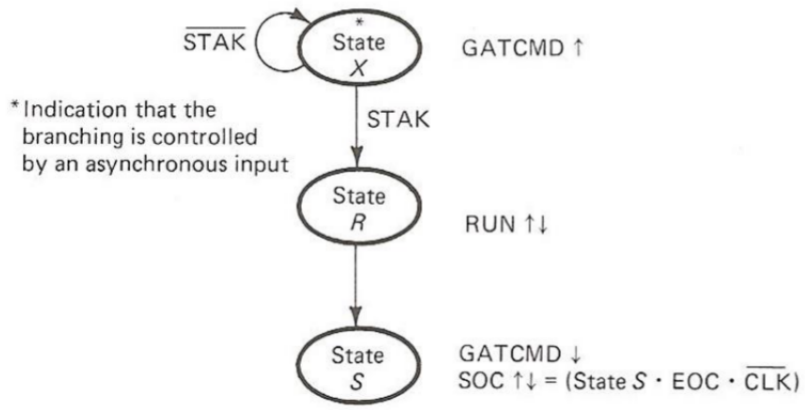


Figura 1.9: Ejemplo de simbología utilizada en diagrama de estado [6].

1.2. Razón señal a ruido y mejoras mediante procesamiento digital

En esta sección se define la métrica señal a ruido inherente a un convertidor análogo digital de N bits. La importancia de esta cantidad es que da una aproximación de los niveles de potencia que el sistema será capaz de analizar.

En la subsección Sobremuestreo y decimación se introduce el proceso de decimación y sobremuestreo que permite aumentar el valor de la señal a ruido mediante procesamiento digital.

En la subsección Filtros polifásicos para decimación se presentan las identidades nobles de sistemas digitales con múltiples relojes. Estas identidades permiten una implementación eficiente de un filtro decimador.

1.2.1. Relación señal a ruido (SNR) y rango dinámico

Para realizar procesamiento de señales al interior de una FPGA debe existir un nexo entre las señales analógicas y las señales digitalizadas. Los convertidores análogo-digital (ADC) desempeñan dicho rol.

La forma más genérica de caracterizar un ADC es por su tasa de muestreo, que es la velocidad con la que el ADC toma una muestra, y la cantidad de bits que posee cada muestra digitalizada. La cantidad de bits que posee un ADC representan el numero de niveles que tiene para representar una señal. Así un ADC de N bits posee 2^N niveles en los que puede representar una señal analógica. A modo de ejemplo, si un ADC tiene un rango de voltaje de 1 V y posee 10 bits se tienen 1024 niveles de representación con un espaciado de $1/1024 = 0{,}976$ mV entre cada nivel.

Debido a esta representación finita, al tomar una muestra que se encuentre entre dos niveles el ADC debe asignar esta muestra a un nivel u otro, esto induce un ruido en la medición que se denomina ruido de cuantificación.

Se define como relación señal a ruido como la razón existente entre la potencia de la señal de interés y la potencia del ruido de fondo.

$$SNR = \frac{P_{Señal}}{P_{Ruido}} \tag{1.6}$$

Para un convertidor análogo-digital (ADC) existe una relación entre el número de bits que se utiliza para representar una muestra y el valor de la razón señal a ruido que se puede obtener, considerando el ruido de cuantificación inherente a los niveles de representación del ADC.

Modelando el ruido de cuantificación como ruido blanco y dado un ADC de N bits se puede calcular la relación de señal a ruido que puede obtener el ADC utilizando la ecuación 1.7 [15].

$$SNR_{ADC} = 6{,}02N + 1{,}76[\mathrm{d}B] \tag{1.7}$$

Cabe decir la ecuación 1.7 es un cálculo considerando el mejor desempeño que se puede obtener con el ADC y por tanto lo normal es operar bajo ese valor. En las hojas de datos de los ADC se entrega información del número efectivo de bits (ENOB) que entrega una aproximación más certera del valor de SNR que se puede obtener al utilizar el ADC en la realidad.

Una definición que está fuertemente ligado con el concepto de SNR es el **rango dinámico**. El rango dinámico es la razón entre el mayor nivel medible con el menor nivel medible.

Por ejemplo, determinar el rango dinámico utilizando como umbral un error de 1° en la fase, equivale a encontrar la mayor y menor potencia que se puede inyectar al sistema sin que se supere el 1° de error y calcular la razón entre ellas. Esto es descrito por la ecuación 1.8 donde σ es la desviación estándar y $\measuredangle$(Min Power level) es la fase medida para la señal con potencia mínima detectable tal que cumpla el requerimiento de tener un error minimo a 1°.

$$\begin{aligned} &(\text{Max Power level} - \text{Min Power level}) \\ &subject\ to\ \sigma(\measuredangle(\text{Min Power level})) < 1^\circ \end{aligned} \tag{1.8}$$

1.2.2. Sobremuestreo y decimación

Pese a que la ecuación 1.7 entrega el límite del SNR que se puede lograr mediante el uso de un ADC, es posible mejorar dicho desempeño con técnicas de post-procesamiento digital. Una de las formas en que se logra aumentar el SNR es utilizando sobremuestreo seguido de un proceso de decimación.

La decimación es una técnica que consiste en una reducción de la frecuencia de muestreo al interior de un sistema digital, esto permite crear zonas al interior de un sistema digital que funcionan a diferentes velocidades y por tanto se rigen por relojes diferentes. La manera

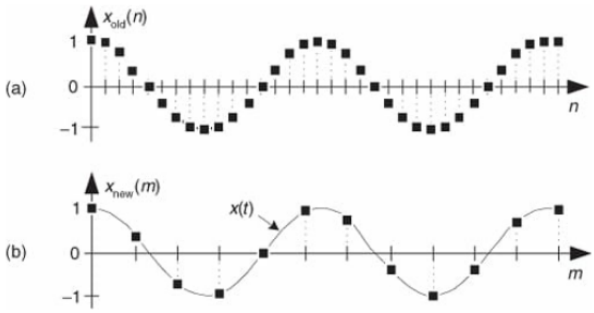


Figura 1.10: Conversión de frecuencia de muestreo: a) Señal original b) Señal con un *downsampling* de factor 3 [7].

más sencilla de disminuir la frecuencia de muestreo es simplemente descartar muestras. A esta acción se le denomina reducción de muestreo o *downsampling*. Se denomina factor de *downsampling* al factor por el cual se reduce la frecuencia del flujo de datos. Un ejemplo gráfico de *downsampling* se muestra en la figura 1.10.

La decimación busca dar uso a la información recolectada y considerarla de alguna manera y no simplemente descartarla. Para ello, antes de realizar un *downsampling* se coloca un filtro digital, el esquema de este proceso se puede ver en la figura 1.11.[16].

Para entender las razones detrás de la colocación de un filtro digital es bueno primero presentar la forma genérica que toma un filtro de respuesta finita (FIR) que puede ser observada en la ecuación 1.9. Explicando de manera muy genérica el filtro tiene N coeficientes α_n que ponderan cada una de las entradas $x(n)$ con lo que cada salida del filtro contiene información ponderada de las N muestras previas que entraron al filtro.

$$w(n) = \sum_{n=0}^{N} \alpha_n x(n) \tag{1.9}$$

Un ejemplo sencillo es tomar $\alpha_n = 1/N$ que convierte la expresión de la ecuación 1.9 en un valor promedio de las N entradas. Con lo que si la señal digitalizada se encuentra entre dos niveles del ADC es esperable que estadísticamente la asignación de los niveles tenga una tendencia y por tanto al calcular el promedio se pueda extraer un estimador del valor real.

La segunda motivación por la que se debe colocar un filtro es más sutil. Suponiendo que se muestrea a una frecuencia f y luego se realiza un *downsampling* de factor M se obtiene una nueva zona con una frecuencia de muestreo $f' = f/M$. Si no se colocase un filtro adecuado previo al *downsampling* las frecuencias que se encuentran en el rango $[f', f]$ caerían sobre el nuevo ancho de banda tras el *downsampling* debido al efecto del *aliasing*. Con ello, la tarea que desempeña el filtro es eliminar las frecuencias que se encuentran fuera del ancho de banda dado por f'. Esto puede observarse de manera gráfica en la figura 1.11 panel b).

Otro concepto que es necesario introducir es el de sobre muestreo u *oversampling*, esta técnica consiste en utilizar un ADC muestreando a una frecuencia mayor de la requerida por el teorema de Nyquist-Shannon.

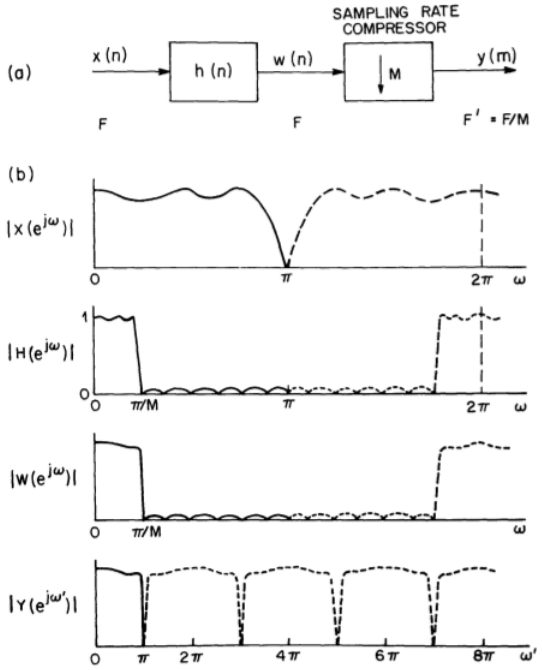


Figura 1.11: a) Diagrama de decimación. Reducción de la frecuencia de muestreo por un factor M. b) Efecto de cada parte en el proceso de decimación en espacio de la frecuencia. [8]

Como se mencionó en la sección previa, el ruido de cuantificación consiste en los errores que ocurren al utilizar un ADC que posee niveles de representación finitos, lo que se traduce como ruido asociado a la conversión digital. La manera usual de modelar este ruido es como un ruido blanco que se reparte de manera homogénea por todo el ancho de banda en el espacio de la frecuencia. Con ello la razón por la cuál se muestrea una frecuencia mayor es que el ruido de cuantificación se reparte en un ancho de banda mayor. En otras palabras, disminuimos la densidad espectral de ruido manteniendo la potencia integrada.

Las ideas de *oversampling* y decimación pueden usarse en conjunto, muestreando a una frecuencia mayor a la de Nyquist para reducir la densidad espectral de ruido y luego realizar una decimación de manera de quedarse solo con la banda de interés y ahora teniendo un ruido en la banda mucho menor en comparación al que se tendría al muestrear con la frecuencia de Nyquist [9].

El proceso de reducción del ruido de cuantificación mediante *oversampling* y decimación puede observarse en el diagrama de la figura 1.12.

Cabe recalcar que en esta aplicación, el filtro *anti-aliasing* toma un rol preponderante, ya que el rechazo del filtro determina cuanto ruido puede ingresar a la banda de interés mediante *aliasing*.

La ganancia en el SNR asociada a la disminución de la densidad espectral de ruido debido al *oversampling* esta dado por la ecuación 1.10.

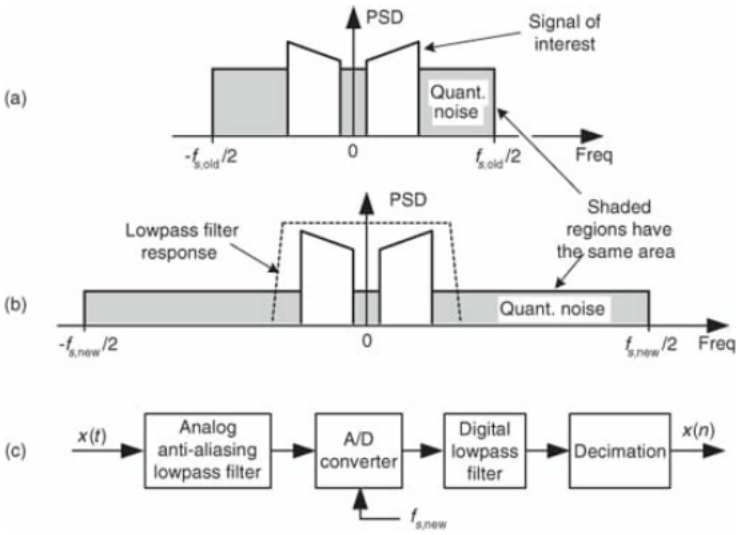


Figura 1.12: a) Ruido de cuantificación correspondiente a muestrear a $f_{s,old}$; b) ruido de cuantificación a muestrear a $f_{s,new} > f_{s,old}$; c) Diagrama de bloques del proceso de reducción del ruido de cuantificación mediante *oversampling* y decimación.[9]

$$SNR_{ADC-gain} = 10log_{10}\left(\frac{f_{Oversampling}}{f_{Nyquist}}\right)[\mathrm{d}B] \tag{1.10}$$

Si se escribe $f_{oversampled} = 4^M f_{Nyquist}$ la ganancia de SNR queda expresada como $SNR_{gain} = M \cdot 10log(4) = 6{,}02M[\mathrm{d}B]$, lo que podemos relacionar con la ecuación 1.7 e interpretar la ganancia de *oversampling* con un mayor cantidad de bits del ADC.

A modo de resumen, es posible mejorar el desempeño de un ADC mediante *oversampling* sumado con una etapa de decimación pagando el costo de observar un ancho de banda menor. Esto es crítico para sistemas que requieren grandes valores de SNR y/o rango dinámico.

1.2.3. Filtros polifásicos para decimación

En esta subsección se estudian distintas implementaciones de filtros de decimación y se introducen las identidades nobles para sistemas de múltiples relojes que permiten realizar una implementación de filtros paralelos que resulta ser más eficiente en el uso de recursos.

La estructura básica para realizar decimación es la mostrada en 1.11 a) donde primero se hace pasar las muestras por un filtro y luego se descartan muestras. Esta implementación no es óptima, debido a que se calculan valores que luego serán eliminados por el proceso de *downsampling*, con lo que esos valores carecen de importancia desde un principio.

La manera óptima de hacer una decimación se basa en utilizar las identidades nobles de sistemas de múltiples relojes que se pueden ver en la figura 1.13 [17], donde $H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots$ es la transformada Z de la respuesta del filtro, con lo que $H(z^D) = h(0) + h(1)z^{-D} + h(2)z^{-2D} + \cdots$

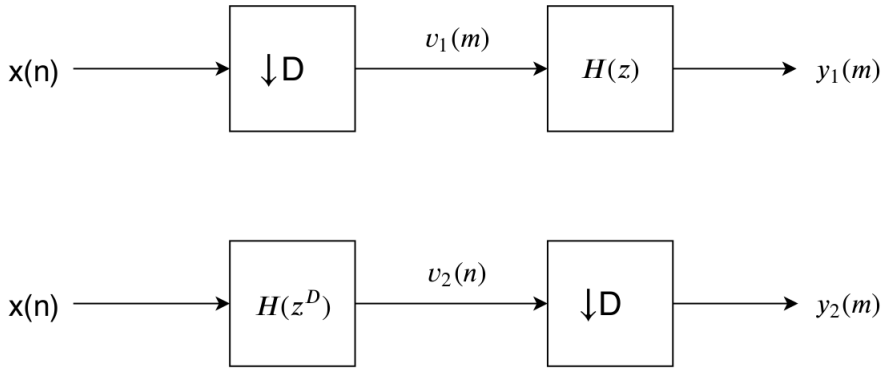


Figura 1.13: Identidades nobles de sistemas con múltiples velocidades. $H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots$ es la transformada Z de la respuesta del filtro y $H(z^D) = h(0) + h(1)z^{-D} + h(2)z^{-2D} + \cdots$

Para la prueba de la identidad se toma un filtro con coeficientes $h(n)$ de tamaño L, y se define el un nuevo filtro con coeficientes $h_D(m)$ de tamaño $D \cdot L$ tal como se muestra en la ecuación 1.11 notando que por construcción se tiene que $h_D(z) = h(z^D)$.

$$h_D(n) = \begin{cases} h(n/D) & \text{si} \quad n/D \in \mathbb{N} \\ 0 & \text{si no} \end{cases} \tag{1.11}$$

Para verificar la equivalencia entre las implementaciones mostradas en la figura 1.13 se busca tomar como punto de partida una de las implementaciones y luego mediante desarrollo algebraico llegar a la segunda implementación. Este desarrollo puede observarse en las ecuaciones 1.12 y 1.13 donde se obtiene $y_1(m) = y_2(m)$. Este resultado implica que mediante una serie de reglas es posible cambiar de posiciones el filtro *antialiasing* y el proceso de *downsampling*. Este resultado no es trivial, ya que típicamente se espera que luego del *downsampling* el *aliasing* ya tuvo lugar con lo que el ruido ya entro en la banda de interés.

$$y_2(m) = v_2(nD) = \sum_{s}^{DL} h_D(s)x(nD - s) = \sum_{k}^{L} h_D(Dk)x(D(n-k)) \tag{1.12}$$

$$= \sum_{k}^{L} h(k)x(D(n-k)) = \sum_{k}^{L} h(k)v_1(m-k) = y_1(m) \tag{1.13}$$

La importancia de cambiar de posiciones el filtro y el *downsampling* es que al colocar el filtro luego del descarte de muestras todas las salidas del filtro son validos y por tanto no se esta incurriendo en un malgasto de recursos. De todas formas la implementación puede ser mejorada aún más.

La estrategia a seguir consiste en elegir el tamaño del filtro adecuadamente de manera

que al utilizar las identidades nobles la expresión resultante sea un implementación de filtros en paralelo. Para ello si se quiere realizar una decimación de factor D se elige un filtro de tamaño $(P \cdot D)$.

Siguiendo la notación previa, tenemos que $y_2(m) = \sum_{k=0}^{PD-1} h(k)x(nD-k)$. Si se realiza el cambio de variable $k = k'D + \mathrm{d}$ con $k' \in (0, P-1)$ y $\mathrm{d} \in (0, D-1)$ se puede separar la suma original en dos sumas con índices dependientes.El desarrollo de la matemática puede verse en las ecuaciones 1.14 y 1.15.

$$y(m) = \sum_{k=0}^{DP-1} h(k)x(nD-k) = \sum_{k'=0}^{P-1}\sum_{\mathrm{d}=0}^{D-1} h(k'D+\mathrm{d})x(nD-k'D-\mathrm{d}) \tag{1.14}$$

$$= \sum_{\mathrm{d}=0}^{D-1} \underbrace{\sum_{k'=0}^{P-1} h(k'D+\mathrm{d})x((n-k')D-\mathrm{d})}_{\text{Subfiltro de largo P}} \tag{1.15}$$

Si en la ecuación 1.15 se toma un valor de d fijo se puede notar que la suma interna es la definición de un filtro FIR de largo P. Recordando que $\mathrm{d} \in (0, D-1)$ se tiene que la implementación descrita por las ecuación 1.15 es la suma de D subfiltros de largo P, donde los coeficientes de dichos subfiltros se obtienen a partir de los coeficientes del filtro original. Tomando ejemplo el subfiltro con $\mathrm{d} = 0$ y siguiendo la expresión de la ecuación 1.15 posee como coeficientes los valores $h(0), h(D), h(2D), \cdots, h((P-1)D)$, donde $h()$ son los coeficientes del filtro original.

Con las muestras que entran a cada subfiltro ocurre algo similar, las muestras que entran en el subfiltro $\mathrm{d} = 0$ serán de la forma $x(nD), x((n-1)D), \cdots x((n-P+1)D)$ que es la versión decimada de la secuencia $x(n)$ por un valor D. Para $\mathrm{d} = 1$ las muestras que entran son de la forma $x((nD-1)), x((n-1)D-1), \cdots x((n-P+1)D-1)$ que corresponde a la misma secuencia del subfiltro con $\mathrm{d} = 0$ pero retrasada una muestra. Para el resto de los subfiltros se tiene la misma relación, todos toman como entrada una versión decimada de $x(n)$ pero con una muestra de retraso. Recordando que se tienen D subfiltros se tiene que en esta implementación todas las muestras son utilizadas.

Este tipo de arquitectura se denomina filtro polifásico y para la decimación toma la forma canónica mostrada en 1.14 donde el *downsampling* se realiza mediante un conmutador con D posiciones, que direcciona el dato entrante a uno de los subfiltros y que al recorrer todas las posiciones permite la suma de los subfiltros en la salida, obteniendo una reducción de la frecuencia de muestreo.

El termino polifásico de esta implementación es debido a que cada rama contiene una versión decimada de la secuencia original desfasada en una muestra. Incurrir en esa muestra de diferencia entre las ramas introduce diferentes valores de fase que puede ser expresado como $exp(2\pi j\mathrm{d}/D)$.

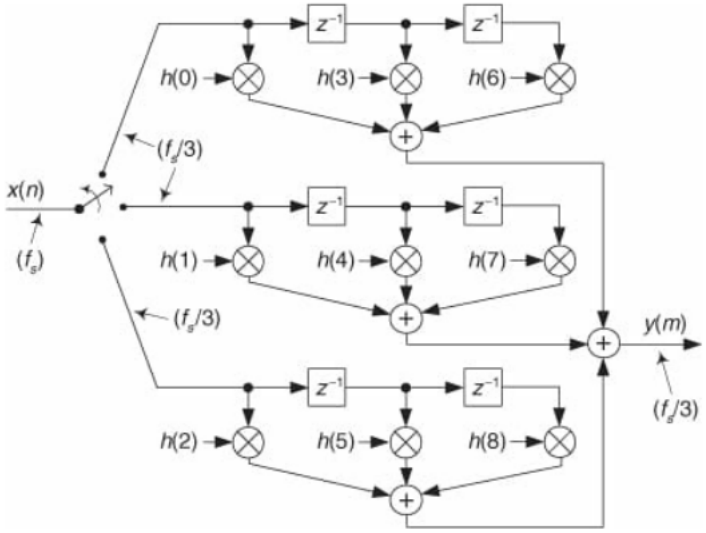


Figura 1.14: Filtro de decimación polifásico con factor de decimación 3.[7]

1.3. Generalidades sobre la transformada discreta de Fourier (DFT)

Una cantidad importante de procesos en el área de procesamiento digital ocurre en el espacio de las frecuencias. En particular para un voltímetro vectorial se busca obtener los valores de magnitud y fase en una frecuencia determinada con lo que utilizar una transformada de Fourier es una de las soluciones posibles a este problema.

En esta sección se definen conceptos relacionados con la transformada de Fourier, poniendo especial énfasis en su interpretación geométrica que lleva naturalmente a los efectos de esparcimiento de frecuencia en la versión discreta de la trasformada de Fourier.

En la subsección Esparcimiento de frecuencia se explica el concepto de esparcimiento en frecuencia de la transformada discreta de Fourier y como el tratamiento con ventanas permite manejar este efecto.

1.3.1. Notación matemática

Espacio Vectorial y espacio dual

Un conjunto de vectores $\mathrm{e_i}$ es una base ortonormal de un espacio V si cumple:

1. $\mathrm{e_i} \in V \; \forall \mathrm{i}$
2. El conjunto $\mathrm{e_i}$ es linealmente independiente.
3. Siendo $v \in V$ podemos escribir $v = \sum_{\mathrm{i}} v^{\mathrm{i}} \mathrm{e_i}$, donde los v^{i} son escalares y son
4. Pedimos además que $\mathrm{e_i e}_j = \delta_{\mathrm{i}j}$ con $\delta_{\mathrm{i}j}$ la delta de Kronecker.

En particular en esta sección se utiliza la notación bra-ket donde de manera sugerente los vectores se escriben según la expresión 1.16 donde se denota $|\mathrm{i}\rangle$ y al símbolo $|\cdot\rangle$ se le denomina "ket".

$$\langle v| = \sum_{\mathrm{i}} v^{\mathrm{i}} \, |\mathrm{i}\rangle \tag{1.16}$$

Dado un espacio vectorial V puede definirse el espacio dual V^*, por ejemplo para el caso donde V es un espacio Hilbert, V^* se corresponde con el conjunto de funciones lineales continuas sobre V. Nuevamente de manera sugerente se escribe $w \in V^*$ siguiendo la expresión en 1.17 donde el símbolo $\langle\cdot|$ se le denomina "bra". Debido a que el espacio dual también es un espacio vectorial los $\langle j|$ son las bases de V^* y los w_j son las componentes.[2]

$$\langle w| = \sum_{j} w_j \, \langle j| \tag{1.17}$$

Si se trabaja en un espacio de Hilbert por teorema de representación de Riesz existe un mapa $\Phi : V \to V^*$. Este mapa es biyectivo y preserva la norma de vectores y co-vectores.

Dado que en este documento se utiliza un espacio vectorial complejo, la acción de Φ sobre V y V^* queda determinada en las ecuaciones 1.18 y 1.19 donde la expresión $()^*$ es el complejo conjugado. Es decir, para pasar de un bra a un ket (o un ket en un bra) basta con conjugar las componentes y convertir el bra en un ket (o el ket en bra según corresponda).

$$\Phi(|v\rangle) = \Phi\left(\sum_{\mathrm{i}} v^{\mathrm{i}} \, |\mathrm{i}\rangle\right) = \sum_{\mathrm{i}} (v_{\mathrm{i}})^* \, \langle \mathrm{i}| = (\langle v|)^* \tag{1.18}$$

$$\Phi(\langle w|) = \Phi\left(\sum_{j} w_j \, \langle j|\right) = \sum_{j} (w^j)^* \, |j\rangle = (|w\rangle)^* \tag{1.19}$$

Producto Interno

Se define el producto interno $\langle\cdot|\cdot\rangle : V^* \times V \to \mathbb{C}$ que debe satisfacer las condiciones enumeradas a continuación.

1. $\langle w|v\rangle = (\langle v|w\rangle)^*$
2. $\langle ax + y|z\rangle = a \, \langle x|z\rangle + \langle y|z\rangle$ con a escalar.
3. $\langle v|v\rangle > 0$

Para el caso genérico de vectores expresados en bases $|\mathrm{i}\rangle$ se tiene el desarrollo mostrado en la ecuación 1.20.

$$\langle w|v\rangle = \sum_{j} w_j \, \langle j| \sum_{\mathrm{i}} v^{\mathrm{i}} \, |\mathrm{i}\rangle = \sum_{j} \sum_{\mathrm{i}} v^{\mathrm{i}} w_j \, \langle j|\mathrm{i}\rangle = \sum_{j} \sum_{\mathrm{i}} v^{\mathrm{i}} w_j \delta^j_{\mathrm{i}} = \sum_{\mathrm{i}} v^{\mathrm{i}} w_{\mathrm{i}} \tag{1.20}$$

[2]La posición de los índices de los v^{i} y w_j es solo notación.

Cambio de bases mediante proyección

Para comprender el efecto geométrico que tiene un cambio de base es ilustrativo observar lo que ocurre en $\mathbb{R}^2$. Supongamos que se tiene un vector $\vec{v} = A\,|x\rangle + B\,|y\rangle$ y se desea llevar este vector a la nueva base $(\mathrm{e}_1, \mathrm{e}_2)$ con $|\mathrm{e}_1\rangle = |x\rangle\cos(\theta) + |y\rangle\sin(\theta)$ y $|\mathrm{e}_2\rangle = |x\rangle\sin(\theta) - |y\rangle\cos(\theta)$.

Los valores de las componentes de $\vec{v}$ en este nuevo sistema coordenado pueden obtenerse utilizando el producto interno de $\mathbb{R}^2$. Por ejemplo la componente $|\mathrm{e}_1\rangle$ de $\vec{v}$ se consigue del calculo $\langle \mathrm{e}_1|v\rangle$, el álgebra puede verse en las ecuaciones 1.21, 1.22 y 1.22.

$$\langle \mathrm{e}_1|v\rangle = \left(\langle x|\cos(\theta) + \langle y|\sin(\theta)\right)\left(A\,|x\rangle + B\,|y\rangle\right) \tag{1.21}$$

$$= A\cos(\theta)\,\langle x|x\rangle + B\cos(\theta)\,\langle x|y\rangle + A\sin(\theta)\,\langle y|x\rangle + B\sin(\theta)\,\langle y|y\rangle \tag{1.22}$$

$$= A\cos(\theta) + B\sin(\theta) \tag{1.23}$$

Geométricamente al calcular $\langle \mathrm{e}_1|v\rangle$ se está proyectando las componentes (x,y) del vector $\vec{v}$ sobre la nueva coordenada e_1. Utilizando este concepto, la coordenada e_1 de $\vec{v}$ toma la forma $|v\rangle_{\mathrm{e}_1} = \left(\langle \mathrm{e}_1|v\rangle\right)|\mathrm{e}_1\rangle$. En la figura 1.15 se puede ver gráficamente la proyección del vector v sobre el eje e_1.

Un cambio de base de un vector $\vec{v}$ hacia un nuevo set de bases ortogonales $|\mathrm{e_i}\rangle$ puede escribirse tal como lo muestra la ecuación 1.24.

$$|v\rangle = \sum_{\mathrm{i}} \langle \mathrm{e_i}|v\rangle\,|\mathrm{e_i}\rangle \tag{1.24}$$

1.3.2. Interpretación geométrica de la transformada de Fourier

La notación braket permite generalizar los conceptos vistos en la subsección anterior a espacios de dimensiones infinita.

Con ello en mente, se define el espacio de funciones $\mathcal{L}^2$ con el producto interno mostrado en la ecuación 1.25. La importancia del espacio $\mathcal{L}^2$ es que contiene las funciones lineales que convergen para la norma-2, es decir que $|f| = \left(\langle f|f\rangle\right)^{1/2} < \infty$.

$$\langle f|g\rangle = \int f^*(x)g(x)\mathrm{d}x \tag{1.25}$$

La definición de la transformada de Fourier continua puede observarse en la ecuación 1.26, donde es inmediato notar que la transformación tiene la forma de un producto interno de la forma $\langle \mathrm{e}^{j2\pi\omega x}|f\rangle$ o equivalentemente la transformada de Fourier no es más que la proyección de la función f sobre la base $\mathrm{e}^{2\pi j\omega x}$ [18]. [3] Si se utiliza la relación de Euler $\mathrm{e}^{j\omega x} = cos(\omega x) +$

[3] Esta afirmación dista de ser matemáticamente formal, por ejemplo las funciones $\mathrm{e}^{\mathrm{i}kx} \notin \mathcal{L}^2$ puesto que la norma-2 de $\mathrm{e}^{\mathrm{i}kx}$ queda indefinida (aunque $\int_{k-\delta}^{k+\delta} \mathrm{e}^{\mathrm{i}kx}\mathrm{d}x \in \mathcal{L}^2$ y aproxima a una base en $\mathcal{L}^2$ cuando $\delta \to 0$).

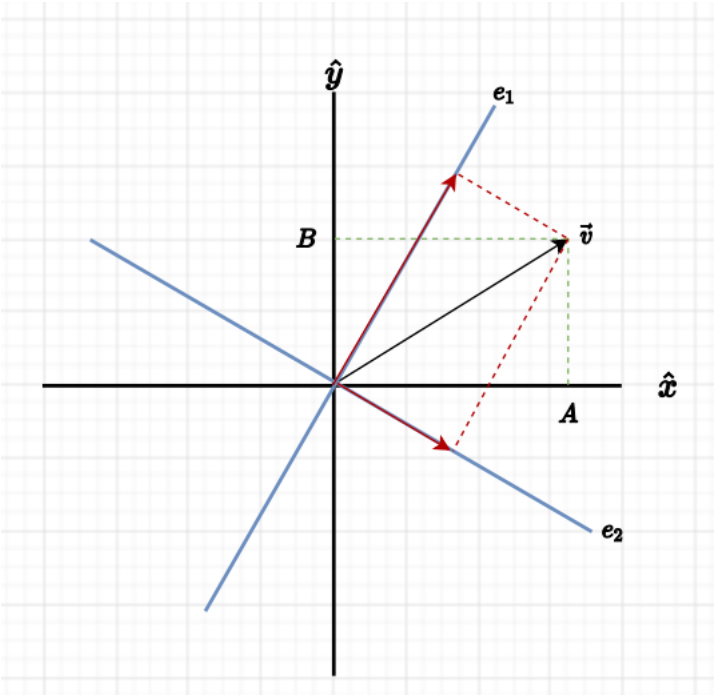


Figura 1.15: En rojo la proyección del vector $v = A\hat{x} + B\hat{y}$ sobre el nuevo sistema de coordenadas (e_1, e_2)

$jsin(\omega x)$ resulta natural relacionar la transformada de Fourier con la información espectral que se puede obtener de una función.

$$f(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-2\pi j\omega x}dx \tag{1.26}$$

Esta explicación geométrica facilita la compresión de la versión discreta de la transformada de Fourier (DFT) que puede observarse en la ecuación 1.27 donde N es el número de muestras con el cual se está calculando la transformada discreta y $W_N^k = e^{j2\pi k/N}$ son los denominados *twiddle factors* de la DFT, que equivalen a la nueva base en la que se proyecta la función a la cual se le calcula la transformada.

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi j}{N}nk} = \sum_{n=0}^{N-1} x_n W_N^{-kn} = \langle W_N^k | x\rangle \tag{1.27}$$

Las componentes en frecuencia de una señal $x[n]$ se pueden escribir tal como lo muestra las ecuaciones 1.28 y 1.29 donde $\hat{X}$ es el espectro de la función $x[n]$ y $\langle W_N^k|x\rangle = A_k e^{\phi_k}$, donde A_k se corresponde con la magnitud de la señal y ϕ_k es la fase de la señal en la frecuencia $2\pi k/N$.

Otro tópico que no se trata aquí son los espacios vectoriales involucrados con la transformación.

En este documento no es necesario lidiar con toda la matemática asociada al análisis funcional, para una descripción formal puede referirse a [19]

$$\hat{X} = \sum_{k=0}^{N-1} X_k \left|W_N^k\right\rangle = \sum_{k=0}^{N-1} \left\langle W_N^k | x \right\rangle \left|W_N^k\right\rangle \tag{1.28}$$

$$\rightarrow \hat{X} = \sum_{k} A_k \mathrm{e}^{j\phi_k} \left|W_N^k\right\rangle . \tag{1.29}$$

1.3.3. Esparcimiento de frecuencia

Una de las particularidades del paso de la forma continua de la transformada de Fourier a su versión discreta es que la DFT es una proyección sobre un número finito de *twiddle factors*. En concreto los *twiddle factors* que se utilizan en la DFT son un conjunto equi-espaciado en el intervalo de frecuencias normalizado $(0, 2\pi)$, lo que acarrea un error en la estimación de frecuencias.

El calculo de una DFT de una función exponencial $x(n) = \mathrm{e}^{j\frac{2\pi j\omega_0}{N}n}$ puede observarse en 1.30 donde se utiliza que la expresión de la DFT tiene forma de suma geométrica. Si $\omega_0 = W_N^{k'}$ se puede aproximar la formula en la ecuación 1.31 como $N\delta(\omega_0 - k')$ con δ la delta de Kronecker. De esta manera la potencia en cada canal espectral k queda dada por la expresión $||X_k|| = N^2\delta(\omega_0 - k')$

Este resultado aparece naturalmente si se utiliza que W_N^k son ortogonales entre si, es decir se cumple que $\langle W_N^{k'} | W_N^k \rangle = \delta(k' - k)$.

$$X(\omega_k) = \sum_{n=0}^{N-1} \mathrm{e}^{j2\pi(\omega_0-\omega_k)/N} \tag{1.30}$$

$$= \left(\frac{1 - \mathrm{e}^{j2\pi(\omega_0-\omega_k)N}}{1 - \mathrm{e}^{j2\pi(\omega_0-\omega_k)}}\right) \tag{1.31}$$

La solución genérica es que $\omega_0 \notin W_N^k$. Por simplicidad en el desarrollo mostrado en las ecuaciones 1.32 y 1.33 se calcula la potencia de la señal en los canales de a DFT.

$$||X_k||^2 = \frac{(1 - \mathrm{e}^{-2\pi j(\omega_0-k)N})(1 - \mathrm{e}^{2\pi j(\omega_0-k)N})}{(1 - \mathrm{e}^{-2\pi j(\omega_0-k)})(1 - \mathrm{e}^{2\pi j(\omega_0-k)})} \tag{1.32}$$

$$= \frac{2(1 - \cos(2\pi(\omega_0 - k)N))}{2(1 - \cos(2\pi(\omega_0 - k)))} = \frac{\sin^2(\pi(\omega_0 - k)N)}{\sin^2(\pi(\omega_0 - k))} \tag{1.33}$$

Si el resultado de 1.33 se normaliza por N^2 y se hace tender $N \rightarrow \infty$ se obtiene que el resultado tiende a $||X_k|| = sinc^2(\omega_0 - k)$. Que la respuesta en frecuencia tenga la forma de un sinc se traduce en que hay un lóbulo principal que toma su máximo en el *twiddle factor* más cercano a ω_0 y que existen lóbulos laterales en el resto de las frecuencias calculadas por la DFT [10]. Es decir, al calcular la DFT sobre una señal de frecuencia no contenida en los

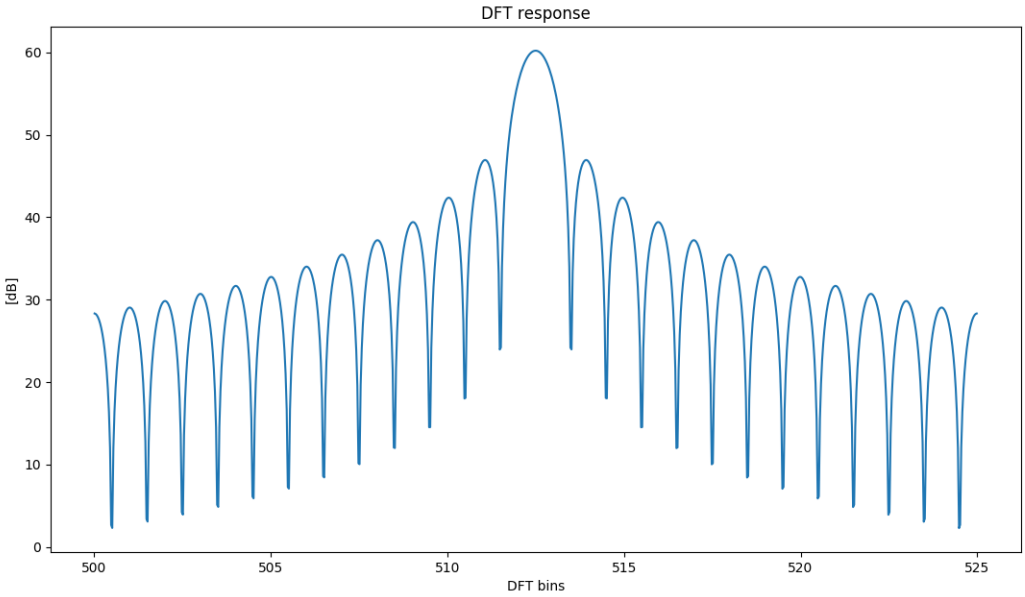


Figura 1.16: Efecto de esparcimiento de frecuencia en una DFT de largo 1024.

twiddle factors la potencia se reparte por todos los canales espectrales. A este efecto se le denomina esparcimiento en frecuencia de la DFT.

En 1.16 se muestra el esparcimiento en frecuencia en una DFT con largo 1024 de una señal con $\omega_0 = 512{,}5/1024 \cdot 2\pi$.

1.3.4. Ventanas

Como se menciona en la subsección anterior, el esparcimiento de frecuencias aparece por el uso de un número finito de *twiddle factors* y por tanto no puede ser eliminado. Sin embargo, mediante la técnica de adicionar una ventana es posible tratar su efecto.

Previo a cualquier desarrollo es necesario enunciar la siguiente propiedad de la transformada de Fourier.

Propiedad de convolución $\mathcal{F}\{f(x) \cdot g(x)\} = \mathcal{F}\{f(x)\} * \mathcal{F}\{g(x)\}$, donde $\mathcal{F}\{\}$ denota la transformación de Fourier y $*$ es el operador convolución.

Una manera ilustrativa de introducir el concepto de ventana es tomar una señal $x(n)$ con $n \in \mathbb{Z}$ y notar que la transformada discreta de Fourier de largo N puede definirse como muestra la ecuación 1.34 donde $w(n)$ es la función rectangular definida en 1.35. Dado que la función $w(n)$ delimita el lapso de tiempo en el cual se calcula la DFT se le suele denominar "ventana".

$$\hat{X}_k = \sum_{n=-\infty}^{\infty} w(n)x(n)\mathrm{e}^{j\frac{2\pi k}{N}n} \tag{1.34}$$

$$w_{N,rect}(n) = \begin{cases} 1 & \text{si} \quad 0 \leq n \leq N-1 \\ 0 & \text{si} \quad otherwise \end{cases} \tag{1.35}$$

Al escribir la DFT como lo muestra la ecuación 1.34 permite identificar que la secuencia a la que efectivamente se le está calculando la transformada de Fourier es una multiplicación de dos funciones $x(n)$ y $w(n)$. Esto permite aplicar la propiedad de la convolución de la transformada con lo que tenemos $\mathcal{F}(w(n)x(n)) = X(k) * W(k)$ con $X(k)$ y $W(k)$ las transformadas de Fourier de $x(x)$ y $w(n)$ respectivamente. Para el caso de la función ventana definida en 1.35 se tiene que $W(k) = sinc$ con lo que nuevamente se obtiene el esparcimiento en frecuencia.

La buena noticia que entrega 1.34 es que es posible limitar el efecto del esparcimiento en frecuencia utilizando otra función $w(n)$ que sea distinta a la función rectangular. Es decir, se puede elegir una ventana $w(n)$ que tenga otra respuesta en frecuencia de manera de tener una mejor respuesta de la DFT.

La mala noticia es que dado un largo N, las ventanas tienden a un compromiso entre el ancho del lóbulo principal y la altura de los lóbulos laterales, con lo que aunque el problema es tratable nunca puede ser solucionado de manera perfecta y es necesario inclinar la balanza en una u otra forma.

Una manera de mejorar el desempeño de una ventana es aumentar el tamaño de la misma, es decir dada una DFT de tamaño N intentar ingresar información contenida en un set de muestras de tamaño $N' = NP$ con P un entero. Aunque esto suena un poco extraño en un principio puede realizarse utilizando una arquitectura de filtros paralelos similares a la derivada para los filtros polifásicos para decimación en la sección1.2.3.

La derivación del sistema consiste en suponer que se calcula un DFT de tamaño $N' = NP$, realizar el cambio de variable $k = k'/P$ y utilizar el cambio de variable de un filtro polifásico $n' = n + Np$ con $n \in (0, N-1)$ y $p \in (0, P-1)$. El desarrollo puede verse en las ecuaciones 1.36 y 1.37 [10].

$$\hat{X}_{N'}(k') = \sum_{n'=0}^{N'} w(n')x(n')\mathrm{e}^{j\frac{2\pi n'}{N'}k'} = \sum_{n'=0}^{N'} w(n')x(n')\mathrm{e}^{j\frac{2\pi n'}{NP}kP} \tag{1.36}$$

$$= \sum_{n=0}^{N-1}\sum_{p=0}^{P-1} w(n+Np)x(n+Np)\mathrm{e}^{j\frac{2\pi k}{N}(n+Np)} = \sum_{n}^{N-1}\left(\underbrace{\sum_{p=0}^{P-1} h(n+Np)x(n+Np)}_{\text{Estructura polifásica}}\right)\mathrm{e}^{j\frac{2\pi k}{N}n} \tag{1.37}$$

En la ecuación 1.37 se puede identificar la estructura de una implementación de filtros polifásica seguido de una DFT de tamaño N. Con lo que pese a que se está ingresando una mayor cantidad de información se mantiene el tamaño de la DFT reduciendo de esta manera los lóbulos laterales sin sacrificar el ancho del lóbulo principal. La desventaja de esta solución es que dado que se requieren más muestras se pierde resolución temporal.

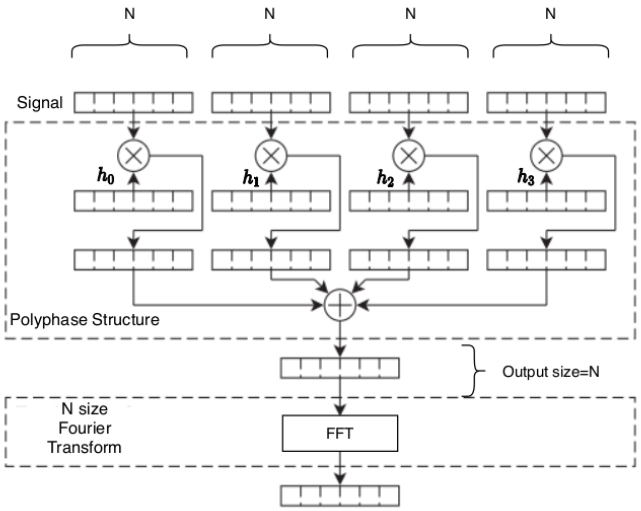


Figura 1.17: Implementación de banco de filtro polifásico seguido de una DFT[10]

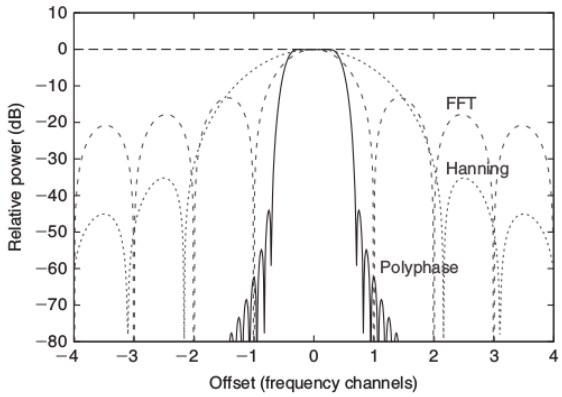


Figura 1.18: Ejemplo de respuesta de banco de filtro polifásico en comparación con una ventana rectangular y una ventana Hanning [10].

En la figura 1.17 se puede observar un diagrama de la implementación de este sistema. En 1.18 se muestra la comparación entre la respuesta ante ventanas rectangular, Hanning y una implementación polifásica de una ventana Hanning[10]. En comparación la implementación polifásica disminuye el ancho del lóbulo principal y disminuye tambien el alto de sus lóbulos laterales teniendo por tanto una mejor capacidad de localización espectral.

Cabe decir que está solución de filtros polifásicos es sumamente utilizada y promovida por el grupo Collaboration for Astronomy Signal Prossecing and Electronic Research (CASPER).

1.3.5. Ganancia intrínseca de la DFT

Una propiedad importante de la versión discreta de la transformada de Fourier es que permite detectar señales insertas en ruido, debido a que en la presencia de una señal existe una suma coherente cuando se calcula la DFT. De forma simple puede pensarse en este

fenómeno de la siguiente manera: Si se calcula la DFT en una frecuencia en la cual no existe señal está tendrá valores que siguen una distribución del tipo *Browniana* que en una suma de N muestras crece proporcional a $\sqrt{N}$. Por otro lado si existe una señal en la frecuencia en la que se calcula la DFT se incurre en una suma coherente y por tanto se espera que la señal crezca proporcional a N.

A continuación se bosqueja el efecto en el SNR que tiene el calculo de una DFT de tamaño N. Por simplicidad tomamos una señal de la forma $x(n) = Ae^{j\omega_{k'}n} + v(n)$ donde $\omega_{k'}$ coincide con un W_N^k y $v(n)$ representa el ruido del sistema que se modela como blanco. La resultante del calculo de la DFT sobre $x(n)$ se puede observar en la ecuación 1.38 donde $\delta(k'-k)$ es la delta de Kronecker [20]. Para derivar la ecuación en 1.39 se nota que la expresión de la DFT de la señal toma la forma de una suma geométrica. El siguiente paso se logra notando que cuando $\omega_{k'} \sim \omega_k$ se puede hacer una serie de Taylor obteniendo el resultado final.

$$\hat{X}(\omega_k) = A\sum_{n=0}^{N-1} \mathrm{e}^{j(\omega_{k'}-\omega_k)} + \hat{V}(\omega_k) \tag{1.38}$$

$$= A\left(\frac{1-\mathrm{e}^{j(\omega_{k'}-\omega_k)N}}{1-\mathrm{e}^{j(\omega_{k'}-\omega_k)}}\right) + \hat{V} = NA\delta(k'-k) + \hat{V}(\omega_k) \tag{1.39}$$

Sobre la expresión en la ecuación 1.39 se calcula la potencia y sobre esta calculamos la esperanza. Si el ruido $v(n)$ se comporta de manera gaussiana con una desviación estándar σ^2 se puede obtener directamente:

$$\mathcal{E}|\hat{X}(\omega_k)|^2 = N^2A^2\delta(k'-k) + N\sigma^2 \tag{1.40}$$

Esto implica que la mejora en el SNR debido al cálculo de la DFT queda dado por la ecuación 1.41, donde cabe recalcar que la derivación se hizo para una señal compleja y que para una señal real se tiene que hacer el reemplazo $N \to N/2$.

$$SNR_{DFT\,gain} = 10log_{10}\left(\frac{N^2A^2}{N\sigma^2}\right) = 10log_{10}(N) + 10log_{10}\left(\frac{A^2}{\sigma^2}\right)[\mathrm{d}B] \tag{1.41}$$

1.4. Resumen

En la sección Motivación se introduce el concepto de holografía que basa su funcionamiento en que el patrón de radiación y las corrientes superficiales de la antena son variables conjugadas canónicas con respecto a la transformada de Fourier. Con esta relación se puede medir el patrón de radiación y calcular las corrientes superficiales, lo que permite hacer correcciones sobre los paneles.

Para la medición de patrón de radiación se hace necesario el uso de un voltímetro vectorial que entrega valores de amplitud y fase de las señales medidas con la antena. En esta

memoria se busca diseñar, implementar y probar este tipo de máquinas utilizando para ello una plataforma basada en FPGA.

En la subsección Field Programmable Gate Array se describe la tecnología de las FPGAs y en Metodología de diseño de un sistema digital se hace una pequeña introducción a las metodología de diseño que deben considerarse a la hora de programar plataformas basadas en esta tecnología.

En la sección Razón señal a ruido y mejoras mediante procesamiento digital se define el concepto de razón de señal a ruido y rango dinámico, así como su relación con el número de bits que tiene un convertidor análogo digital. Esta cantidad es de suma importancia en sistemas que necesitan alto rango dinámico. En la subsección Sobremuestreo y decimación se presentan las técnicas de sobre muestreo y decimación que puede se utilizada para aumentar la relación señal a ruido mediante procesamiento digital. Finalmente en Filtros polifásicos para decimación se presenta arquitecturas para la implementación de filtros decimadores en plataformas digitales.

En la sección Generalidades sobre la transformada discreta de Fourier (DFT) se presenta una interpretación geométrica de la transformada de Fourier que explica de manera natural efectos presentes en la transformada discreta de Fourier. En Esparcimiento de frecuencia se presenta el efecto de esparcimiento en frecuencia de la DFT y en Ventanas se presenta el concepto de función ventana que permite tratar el efecto de esparcimiento. Por último en Ganancia intrínseca de la DFT se presenta la ganancia en razón señal a ruido que se obtiene por la aplicación de una DFT.

Capítulo 2

Diseño e implementación de subsistemas

Dada la naturaleza del problema es posible seccionar el problema en dos grandes hitos que pueden ser trabajados de manera separada. El primer hito consiste en el diseño e implementación de un sistema digital basado en técnicas de procesamiento de señales, que efectúa el cálculo de magnitud y fase relativa entre dos señales que cumpla con un rango dinámico de 70 dB tomando como umbral un error en la medición de la fase de 1°. Denominaremos esta máquina como voltímetro vectorial. El segundo hito trata el diseño e implementación de un sistema encargado de la colocación de una etiqueta de tiempo con una precisión mejor a 0,1 ms. En este subsistema se hacen uso de las técnicas clásicas de diseño de sistemas digitales mencionadas en el capítulo anterior.

Siguiendo esta línea, este capítulo está dividido en tres secciones. La primera enfocada al diseño del subsistema del voltímetro vectorial. La segunda sección describe el diseño del subsistema encargado del marcado de tiempo. Además, se agrega una sección que se refiere a la implementación de ambos modelos en la plataforma *Reconfigurable Open Architecture Computing Hardware 2* (ROACH2).

2.1. Voltímetro vectorial

El diagrama de bloques referente a la arquitectura propuesta para el voltímetro vectorial se puede observar en la figura 2.1.

Previo a una descripción detallada del diagrama de voltímetro vectorial es ilustrativo entregar las ideas generales de este modelo. Dado que el ADC de la plataforma ROACH2 es de 8 bits se tiene teóricamente un $SNR = 49{,}9$ dB. Como se desea que el voltímetro tenga un rango dinámico de por lo menos 70 dB una forma de aumentar la razón señal a ruido es tener una etapa donde se sobremuestrea la señal y se implementa un filtro de decimación.

En este sistema la medición de amplitud y fase se hace en el espacio de las frecuencias, con lo que una DFT se hace necesaria y con ella la colocación de una ventana para tratar el efecto de esparcimiento de potencia en los canales espectrales. Una vez en el espacio de las frecuencias se puede utilizar la representación $A_k e^{j\phi_k} |W_N^k\rangle$ de donde mediante identidades básicas se puede obtener A_k y ϕ_k.

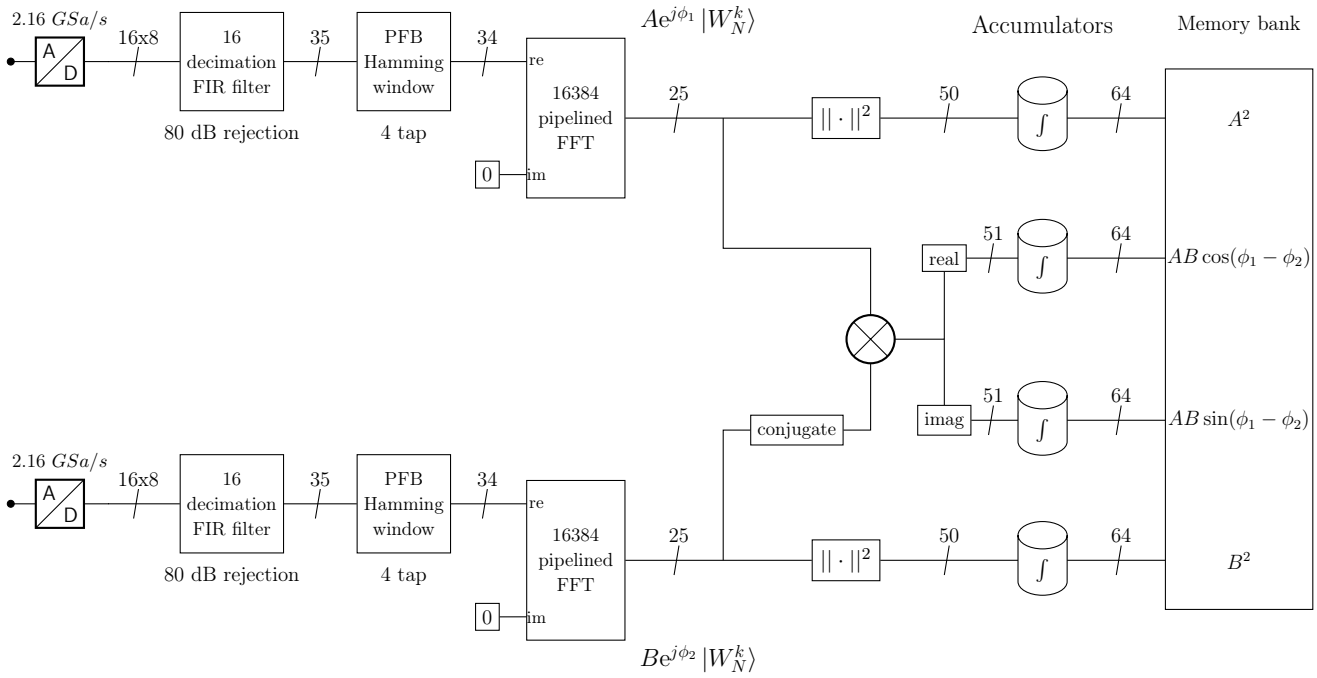


Figura 2.1: Diseño voltímetro vectorial. De izquierda a derecha se tiene un filtro polifásico de decimación de factor 16 para aumentar el SNR, luego hay un bloque que aplica una ventana para calcular una FFT de tamaño 16384. Despues de la FFT se calculan correlaciones y magnitudes de los canales espectrales y dichos valores ingresan a un acumulador programable. Finalmente los datos son almacenados en memoria para que el usuario pueda acceder a ellos.

A continuación se hace una descripción detallada del diagrama en la figura 2.1

2.1.1. Etapa de sobremuestreo

En este esquema se muestrean simultáneamente dos entradas una es la señal de la antena que se quiere caracterizar y la otra corresponde a una señal de una antena de referencia. Para digitalizar ambas señales se utilizan dos ADC de alta velocidad y posterior a la digitalización hay una etapa de decimación. La idea detrás de los primeros dos bloques de izquierda a derecha es utilizar la técnica de *oversampling* para aumentar el SNR. Con ello se establece una frecuencia de muestreo para los ADCs de 2,16 GSa/s que luego es decimado por un factor de 16. Esto se traduce en una reducción de la frecuencia de muestreo a un valor de 135 MHz lo que entrega un ancho de banda de 67,5 MHz después de calculada la DFT.

En el proceso de *oversamplig* se obtiene una mejora teórica del SNR de 12 dB debido a la reducción de la densidad de ruido de cuantificación. Esta mejora en el SNR está sujeta a la implementación del filtro *anti-aliasing* de decimación. En este caso se utiliza un filtro FIR de 560 coeficientes para utilizar una implementación polifásica. El filtro se diseña con una banda de paso en la región $(0, 64{,}8)$ MHz y un rechazo de ~ 80 dB en la banda de interés lo que se traduce en una limitante en el desempeño del sistema, ya que entrega el nivel del ruido que puede entrar mediante *aliasing* al reducir la frecuencia de muestreo. El diseño se hace utilizando el método de Remez disponible en la herramienta *FDA-tool* del software MATLAB. La respuesta del filtro puede verse en el panel superior de la figura 2.2.

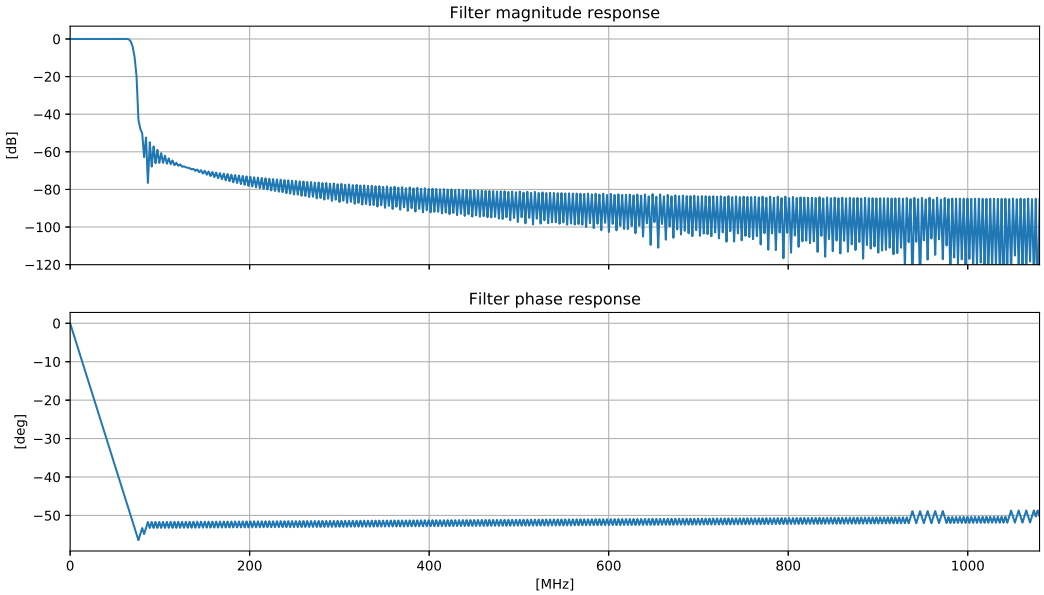


Figura 2.2: Respuesta de filtro *anti-aliasing* de etapa de decimación. En el panel superior se muestra la respuesta en magnitud del filtro. En el panel inferior se muestra la respuesta en fase del filtro.

Otra de las propiedades importantes que poseen los filtros FIR es que la respuesta en fase es lineal en la banda de paso, lo que implica que al utilizar dos filtros idénticos en las dos ramas del diagrama en la figura 2.1 no se introducen errores a la hora de calcular las fases relativas entre dos salidas simultaneas de los filtros. La linealidad de la respuesta en fase del filtro en la banda de paso puede observarse en el panel inferior de la figura 2.2.

2.1.2. Transformada de Fourier y ventana

Posterior a la reducción de la frecuencia de muestreo en el diagrama en la figura 2.1 sigue una secuencia en la cuál se coloca una ventana a los datos y se calcula una DFT. Esta sucesión de bloques tiene como finalidad el manejo del efecto de esparcimiento en frecuencia y es una práctica común en el mundo de procesamiento digital.

La ventana colocada es una ventana *Hamming* cuya ecuación se encuentra en 2.1 con N el tamaño de la ventana. Este tipo de ventanas se consideran un punto medio en la relación entre el ancho del lóbulo principal y la altura de los lóbulos laterales en comparación con otras ventanas que privilegian una de las dos características.

$$h(n) = 0{,}53836 + 0{,}46164 \cos\left(\frac{2\pi n}{N-1}\right) \tag{2.1}$$

La ventana se coloca mediante una arquitectura polifásica (PFB) de 4 sub-filtros, donde cada sub-filtro tiene tamaño igual al tamaño de la DFT que se calcula en el bloque siguiente.

Es decir el valor de $N = 4 \cdot DFT_{size}$ en 2.1, lo que se traduce en una disminución de la altura de los lóbulos laterales y una reducción del ancho del lóbulo principal.

Para este diseño la convergencia del bloque de PFB puede calcularse mediante la expresión en la ecuación 2.2 y para el caso del diagrama en la figura 2.1 se tiene $PFB_{time} = 0{,}485$ ms. Dado que el requerimiento considera un tiempo de computo de valores de amplitud y fase sucesivos de ~ 1 ms la fuente debe mantenerse constante por ese periodo de tiempo, con lo que la perdida en resolución temporal es aceptable.

$$PFB_{time} = \frac{Number\ PFB_{subfilters} \cdot DFT_{size}}{clock\ cycle} \tag{2.2}$$

El bloque que sigue al PFB en la figura 2.1 consiste en una transformada rápida de Fourier(FFT) que es una implementación eficiente de una DFT reduciendo la cantidad de operaciones utilizando simetrías con la condición de que el largo de la DFT debe ser una potencia de 2. El tamaño de la FFT es $2^{14} = 16384$, dado a que la entrada es puramente real la información se encuentra duplicada a la salida del bloque FFT y basta tomar 8192 canales. Recordando que el ancho de banda, tras la decimación, es 67,5 MHz se tiene un espaciado entre cada par de canales de 8,24 kHz, ateniéndose al requerimiento de un espaciado entre canales menor a 10 kHz. Además, la mejora en el SNR por el uso de una DFT es de 39,13 dB.

La FFT utilizada tiene un arquitectura *pipeline* lo que quiere decir que el bloque es capaz de recibir un paquete de datos, mientras entrega la DFT del paquete anterior. De esta manera, luego de la inicialización del bloque, la FFT proporciona valores en cada ciclo de reloj que se corresponden con los canales espectrales del paquete de datos anterior.

El formato del paquete de datos de salida de la FFT, consiste en entregar un canal espectral por ciclo ordenados de menor a mayor. De manera que si se sincronizan correctamente los bloques de FFTs de las dos ramas en 2.1 se puede tener que ambos bloques entreguen el mismo canal espectral en un ciclo. Es decir en cada ciclo tenemos un valor $\Psi = A_k \mathrm{e}^{j\phi_{1,k}} |W_N^k\rangle$ y $\Theta = B_k \mathrm{e}^{j\phi_{2,k}} |W_N^k\rangle$ con idéntico k en cada una de las ramas.

2.1.3. Espectrómetro y Correlador

Luego de la FFT en la figura 2.1 existen 4 caminos que se pueden tomar donde se opera para obtener la magnitud de las señales (caminos superior e inferior en la figura 2.1) y la fase relativa de cada canal (caminos centrales en figura 2.1).

Para el calculo de la magnitud al interior de un canal espectral se sigue la operatoria en la ecuación 2.3 con Ψ el valor entregado por la FFT en un ciclo. Para el cálculo de la magnitud asociada a cierta frecuencia basta calcular la norma de cada canal. A este calculo de la respuesta en magnitud de la señal entrante se le denomina espectrómetro.

$$A_k^2 = ||\Psi||^2 = \Im m\{\Psi\}^2 + \Re \mathrm{e}\{\Psi\}^2 \tag{2.3}$$

Para obtener el valor de la fase relativa entre las dos ramas se utiliza la matemática descrita en las ecuaciones 2.4 y 2.5. Para implementar las ecuaciones anteriores en un circuito digital es necesario conjugar una de las ramas y multiplicar ambas ramas, que es lo que se realiza en los caminos centrales en la figura 2.1. A esta arquitectura se le denomina correlador, debido a que es equivalente a una correlación en el dominio del tiempo.

$$\langle\Theta|\Psi\rangle = \left(A_k e^{j\phi_{1,k}} |W_N^k\rangle\right)\left(B_{k'} e^{j\phi_{2,k'}} |W_N^{k'}\rangle\right)^* = A_k B_{k'} e^{j(\phi_{1,k}-\phi_{2,k'})} \langle W_N^{k'}|W_N^k\rangle \tag{2.4}$$

$$= A_k B_{k'} e^{j(\phi_{1,k}-\phi_{2,k'})}\delta(k,k') = A_k B_k e^{j(\phi_{1,k}-\phi_{2,k})} \tag{2.5}$$

A partir de la expresión en la ecuación 2.5 es posible obtener el valor de la diferencia de fases utilizando la formula en la ecuación 2.6. Dado que el diseño considera el uso de una representación de punto fijo en los datos si se intenta calcular una arco tangente se incurre en un error numérico considerable, con lo que se decide realizar el cálculo de la arco tangente en un procesamiento posterior en un computador.

$$(\phi_{1,k} - \phi_{2,k}) = arctan\left(\frac{\Im m\{\langle\Theta|\Psi\rangle\}}{\Re e\{\langle\Theta|\Psi\rangle\}}\right) \tag{2.6}$$

2.2. Acumulación y almacenamiento

Después de calculadas las magnitudes y fase relativa entre las señales en el diagrama en la figura 2.1 se tiene un set de 4 acumuladores uno para cada camino. Un acumulador es un arreglo de memorias de tamaño igual al número de canales de la FFT, donde se suman los canales de múltiples salidas de la FFT. La salida con dirección k de un acumulador, queda determinado por la ecuación 2.7 donde N es el largo de la acumulación, y_k es la entrada al acumulador correspondiente al canal k-ésimo de la FFT.

$$x_{acc}(k) = \sum_{\mathrm{i}=0}^{N} y_k \tag{2.7}$$

De esta manera, un acumulador actúa como un operador promedio que no está normalizado por el factor N.[1] Por tanto se puede ver el proceso de acumulación como un filtro pasa-bajos, que entre otras cosas ayuda a reducir la variabilidad de las cantidades calculadas.

Finalmente en la figura 2.1 se tiene un banco de memorias BRAMs de dos puertos donde se guardan los valores de las magnitudes y la parte real e imaginaria del correlador. Este banco de memorias esta comunicado a una red LAN, con lo que se pueden extraer los datos tomados para post-procesarlos.

A continuación se resumen las principales características del subsistema de voltímetro vectorial en los siguientes puntos:

[1]Dado que se requiere trabajar con valores relativos entre las entradas, no es necesario ni deseable normalizar (la división es una operación costosa).

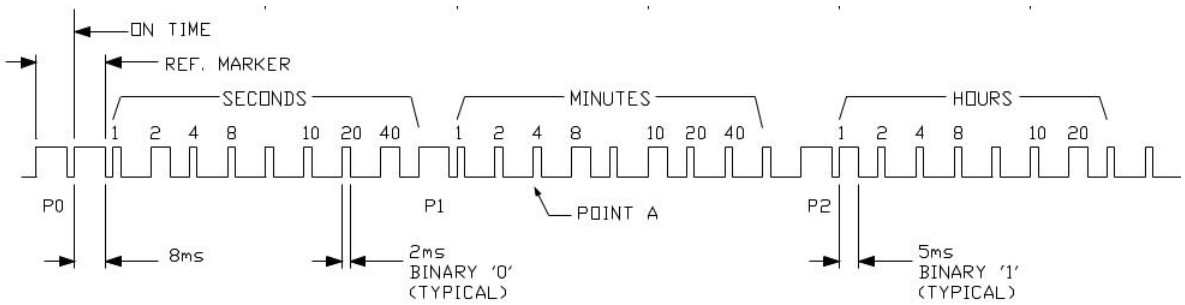


Figura 2.3: Ejemplo de trozo de protocolo IRIG con $bit\,time = 10ms$[11]

- Ancho de banda de 67,5 MHz.
- 8192 canales donde el espaciado entre cada par de canales es de 8,24$[kHz]$.
- Relación de señal a ruido teórica de 97,48 dB[2]
- Una tasa de entrega de datos de 121,36μs, cuando se tiene una acumulación 1.

2.3. Etiquetado de tiempo

2.3.1. IRIG time code

Para el etiquetado de tiempo se decidió utilizar el protocolo *Inter-range instrumentation group time codes*, por sus siglas IRIG time code. Previo a la descripción de diseño del sistema de etiquetado es necesario introducir el funcionamiento del protocolo IRIG e identificar los subsistemas necesarios.

De manera genérica, el protocolo IRIG consiste en 100 o 60 pulsos que se envían desde un reloj maestro a los dispositivos esclavos en un determinado periodo. La información temporal se encuentra codificada en la duración relativa de los pulsos, de manera que cada pulso representa un bit que puede tomar tres valores.

Primero es necesario definir el tiempo que le corresponde utilizar a cada bit al interior del paquete de datos, lo que se denomina *bit time*. A partir de esta duración se puede especificar los valores que puede tomar el bit como:

- Cero binario (0): $0{,}2 \cdot bit\,time$
- Uno binario (1): $0{,}5 \cdot bit\,time$
- Bit de referencia :$0{,}8 \cdot bit\,time$

El valor de cada bit también está determinado por la posición que ocupa en el paquete de datos. Por ejemplo los bits con índices contenidos en el intervalo (1-4) en el paquete representan las unidades de segundo, los bits 5, 6 y 7 corresponden a la información de las decenas de segundo [11]. Este tipo de codificación se le llama *binary coded decimal*(BCD) ya que pese a utilizar números binarios para representar los valores se mantiene el sistema de numeración decimal.

[2]Considerando un ENOB de 7.4

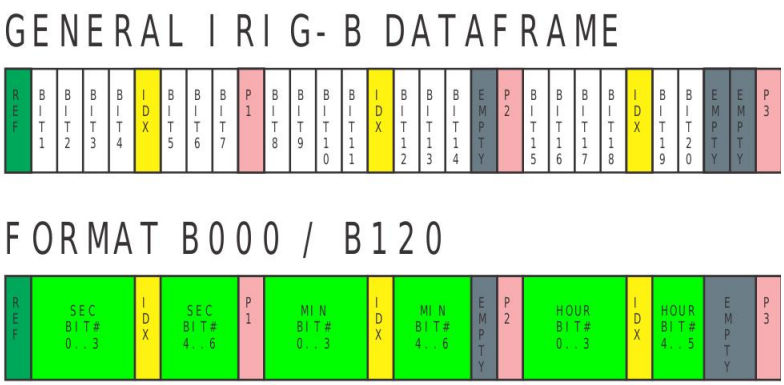


Figura 2.4: Ejemplo de valorización de los bits al interior de un paquete IRIG.

La estructura básica de IRIG consiste en sub-agrupaciones de 10 bits donde en el primer y último bit toman valores de bits de referencia, el resto de los bits entregan la información temporal. La presencia de estos bits de referencia permite encontrar el comienzo de un paquete de datos, ya que el bit final del anterior paquete antecede el bit de referencia del comienzo del nuevo paquete. Así para encontrar el comienzo basta con buscar dos bits de referencia consecutivos. En la figura 2.3 se muestra un fragmento del inicio de un paquete de datos de IRIG donde se puede observar las subagrupaciones de segundos, minutos y hora.

En la figura 2.4 se puede ver la valoración que se le hace a los bits dentro de una fracción de un paquete de IRIG. En Anexos se puede observar el paquete de datos completo para IRIG-B.

Existen múltiples implementaciones de IRIG, para este documento se utiliza la versión IRIG-B000 que es una implementación sin ninguna modulación, tiene un periodo de paquete de 1 s en el cual se envían 100 pulsos con un *bit time* de 10 ms.

2.3.2. Diseño de sistema de etiquetado temporal

Para el proceso de diseño se sigue la metodología mencionada en la subsección Metodología de diseño de un sistema digital donde se divide el problema en un sistema controlador y un sistema controlado.

El algoritmo que se desea implementar consiste en primero identificar el comienzo de un paquete de datos y decodificar la información contenida en el paquete. Y una vez fijado el tiempo actual se requiere de un mecanismo que permita la actualización del tiempo según va transcurriendo.

En la figura 2.5 se tiene el diagrama de flujo del algoritmo propuesto. Básicamente, al identificar la presencia de dos valores de referencia consecutivos se encuentra el comienzo de un paquete de datos lo que permite referenciar la posición de los pulsos siguientes en el formato del paquete de datos de IRIG.

Luego de identificado el comienzo del paquete comienza la etapa de decodificación de la información contenida en los pulsos. Para ello se utiliza un contador que cuenta la duración del

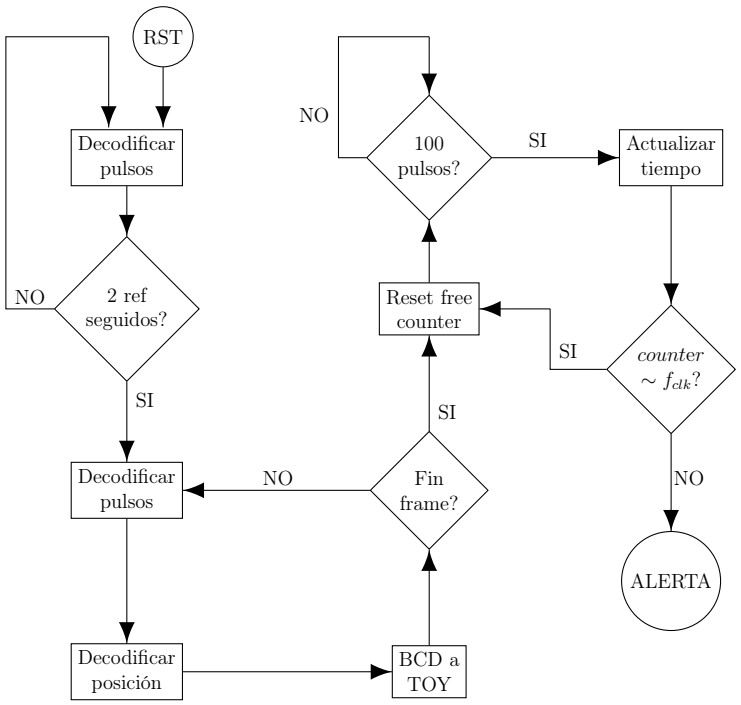


Figura 2.5: Diagrama de flujo de algoritmo de marcado de tiempo.

pulso en términos de ciclos de reloj lo que puede ser comparado con las duraciones asociadas a cada valor por el protocolo IRIG.

Debido a que la información llega en un formato BCD que no es cómodo de operar, se decidió cambiar al formato *Time Of the Year*(TOY) donde se escribe el tiempo como los segundos transcurridos en el año.

Cuando se decodifica el primer paquete de IRIG se puede establecer el tiempo actual pero este debe ser constantemente actualizado. Para manejar las actualizaciones se utiliza el conocimiento de que en IRIG-B000 un paquete contiene 100 pulsos y que su duración es de 1 s. De manera que basta contar los 100 pulsos y agregar un segundo al valor actual de los segundos en formato TOY para actualizar el paso de un segundo.

Para los valores de fracciones de segundos se opta por colocar un contador libre que se reinicia al comienzo de cada paquete de datos. De manera que la información de sub-segundos queda codificada en periodos del reloj de la FPGA. Este diseño permite comprobar el enclavamiento con el reloj maestro, ya que si el sistema funciona correctamente en cada actualización de los segundos se debiese tener que el contador de sub-segundos tiene un valor similar a la frecuencia del reloj utilizado por el sistema.

El diagrama de bloques del sistema se encuentra en la figura 2.6 donde las cajas de color azul son máquinas de estado finito que actúan como el subsistema de control. Las cajas de color rojo son bloques que pueden ser implementados con lógica combinacional y corresponden al subsistema controlado. Las cajas de color verde corresponden a memorias donde se almacena el valor del tiempo actual en segundos.

A continuación se coloca una breve descripción del sistema controlador y el sistema con-

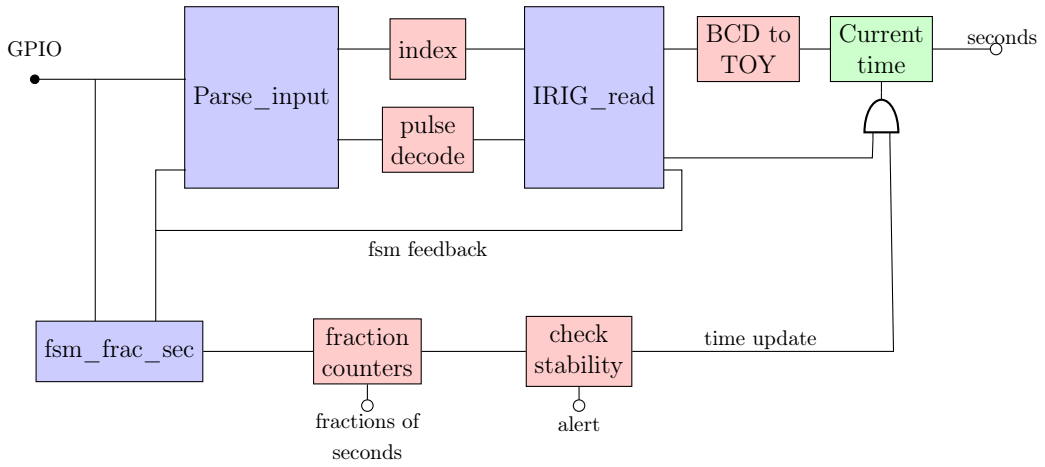


Figura 2.6: Diagrama de bloques de algoritmo de marcado de tiempo. En color azul se muestra el sistema de control, en color rojo los sistemas controlados y en color verde las memorias.

trolado. Para ver una descripción más detallada de los sub-módulos que componen a cada sistema referirse a Anexos

Sistema controlado

- **index**: Es un contador controlado por **Parse_input** que se encarga de llevar la cuenta del índice en el paquete de datos de IRIG del valor que se está recibiendo.
- **pulse decode**: Es un contador controlado por **Parse_input** que se encarga de contar la duración del pulso que se recibe, de esta manera puede hacerse una comparación con valores fijos y determinar el valor del bit.
- **BCD to TOY**: El controlador **IRIG_read** entrega una dirección en el rango (0-8) que codifica el tipo de dato que se está tratando y el dato en si (por ejemplo 1 representa decenas de segundo). Luego, debido a que el protocolo BCD utilizado por IRIG es *big endian* es necesario reordenar los datos de cada una de las direcciones en formato *little endian*. Finalmente se multiplican los valores de cada dirección por un factor adecuado para expresar cada una de las direcciones en su equivalente en segundos y sumando todos estos valores se obtiene el valor en formato TOY.
- **fraction counter**: Consiste en dos contadores. El primero cuenta la cantidad de pulsos recibidos y al llegar a 100 emite una uno lógico que aumenta en uno el valor de las memorias **current time**, este uno lógico es usado además por el módulo **clock stability**. El segundo contador cuenta las fracciones de segundo que han transcurrido y se reinicia cada vez que el primer contador emite el uno lógico.
- **clock stability**: Compara el valor del contador de fracciones de segundo con valores fijados por el usuario cuando el bloque **fraction counter** emite la señal de que ha transcurrido un segundo. Si se encuentra fuera de los rangos establecidos por el usuario emite una alerta.
- **current time**: Consiste en una memoria que almacena el valor calculado en **BCD to TOY** y lo carga en un contador que aumenta su valor ante la señal emitida por **fraction counter**.

Sistema controlador

- **Parse_input**: Debido a que es necesario decodificar la información de los pulsos en todo momento se crea esta maquina de estado finito que controla el bloque **pulse_decode**. También se hace cargo de llevar la cuenta del índice del dato en el paquete de IRIG.

  Para que la numeración de los índices sea correcto es necesario que exista un canal de comunicación con el módulo **IRIG_read** que es el controlador encargado de determinar el comienzo de un paquete de datos. Este canal está contenido en el bus *fsm feedback* en la figura 2.6.

  Las principales señales de **parse_input** son:

  - *hard_rst*(input): reinicia la maquina a un estado inicial conocido.
  - *gpio*(input): dato de IRIG que se obtiene utilizando un *General Purpose Input/Output*.
  - *cal*(input): señal que indica que se desea calibrar el reloj interno.
  - *in_frame*(input): señal emitida por **IRIG_read**, indica el comienzo de un paquete de datos de IRIG.
  - *terminate*(input): señal emitida por **IRIG_read**, indica que se termina la lectura de un paquete de datos, lo que permite que los bloques de actualización del tiempo actúen.
  - *continue*(input): señal emitida por **IRIG_read**. Se trata de una señal de que la información ha sido recibida correctamente.
  - *issue*(input): señal emitida por **IRIG_read**. Señala que el paquete recibido no sigue la estructura esperada.
  - *en_ind*(output): permite aumento del contador en **index**.
  - *en_cbh*(output): permite aumento del contador en **pulse decode**.
  - *data_ready*(output): indica a **IRIG_read** que el pulso de un dato de IRIG esta listo.

  El bloque completo de **parse_input** se encuentra en la figura 2.7.[3]

  En la figura 2.8 se encuentra el diagrama de flujo de **parse_input** que describe el comportamiento de la maquina de estado. A partir del diagrama de flujos es posible reducir cada acción descrita en la figura 2.8 en una transición de estados elementales.

  El diagrama de estados se encuentra en Anexos.

[3]Los puertos que no se mencionan en la descripción son señales de identificación de errores y no afectan al desempeño de la maquina.

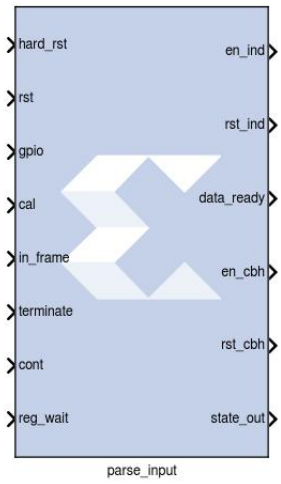


Figura 2.7: Bloque de maquina de estado finito **parse_input**

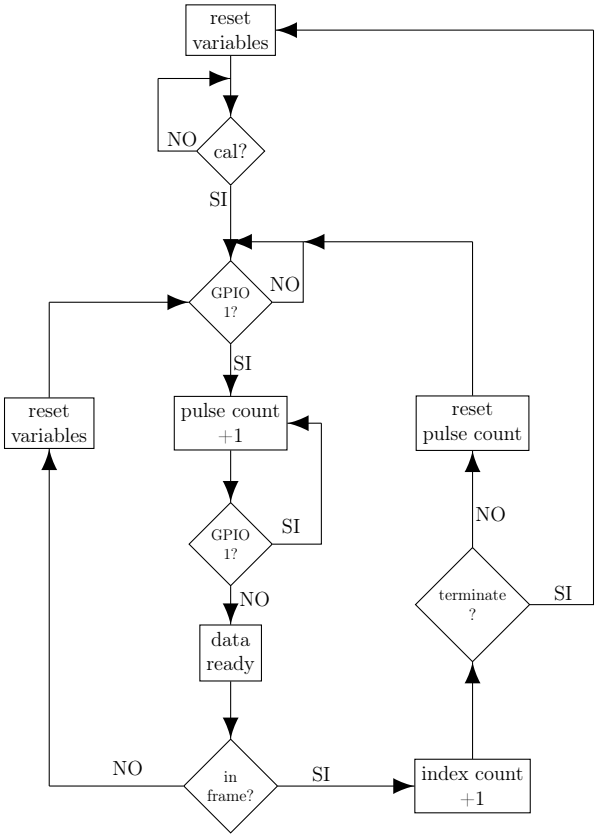


Figura 2.8: Diagrama de flujo para **parse_input**.

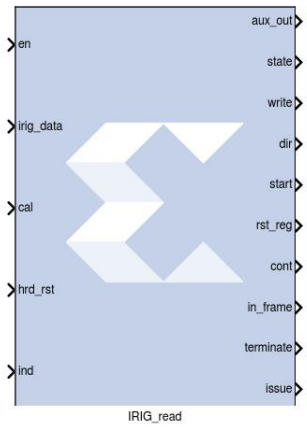


Figura 2.9: Bloque de maquina de estado finito **IRIG_read**.

- **IRIG_read**:

  Esta maquina se encarga de encontrar el comienzo de un paquete de datos de IRIG, para ello busca la presencia de 2 señales de referencia consecutivas utilizando el contador *count* y da aviso a **parse_input** colocando un uno lógico en la salida *in_frame*.

  Además, esta maquina posee un diccionario interno que traduce los índices entregados por el módulo **index** en valores que representan las unidades del dato que se está recibiendo. Por ejemplo el índice 16 del paquete de IRIG se traduce en un valor 3, que corresponde a la codificación de las decenas de minutos para el bloque **BCD to TOY** que utiliza esta información para ordenar los bits y multiplicar por el factor adecuado.

  En la figura 2.9 se encuentra el bloque completo de **IRIG_read**.

  Las principales señales de **IRIG_read** son:

  - *en*(input): Indica que hay un dato valido en *irig_data* y *ind*.
  - *irig_data*(input): Valor del pulso decodificado por **parse_input**.
  - *cal*(input): Señal que indica que se desea calibrar el reloj interno.
  - *hrd_rst*(input): Reinicia la maquina a un estado inicial conocido.
  - *ind*(input): Índice del dato entrante.
  - *write*(output): Valor decodificado del pulso que se entrega a **BCD to TOY**.
  - *dir*(output): Valor que entrega el diccionario interno de la maquina al evaluar en el índice dado por *ind*. O en pseudo-código dir=dic[ind].
  - *rst_reg*(output): Reinicia los bloques que están controlados por este controlador.
  - *cont, in_frame, terminate, issue*(outputs): Estas señales están descritas en el bloque anterior.[4]

  En la figura 2.10 se encuentra el diagrama de flujo que describe el comportamiento del bloque. A partir de este esquema se obtiene el diagrama de estados en Anexos.

[4]Los puertos de 2.9 que no se mencionan son señales de detección de errores.

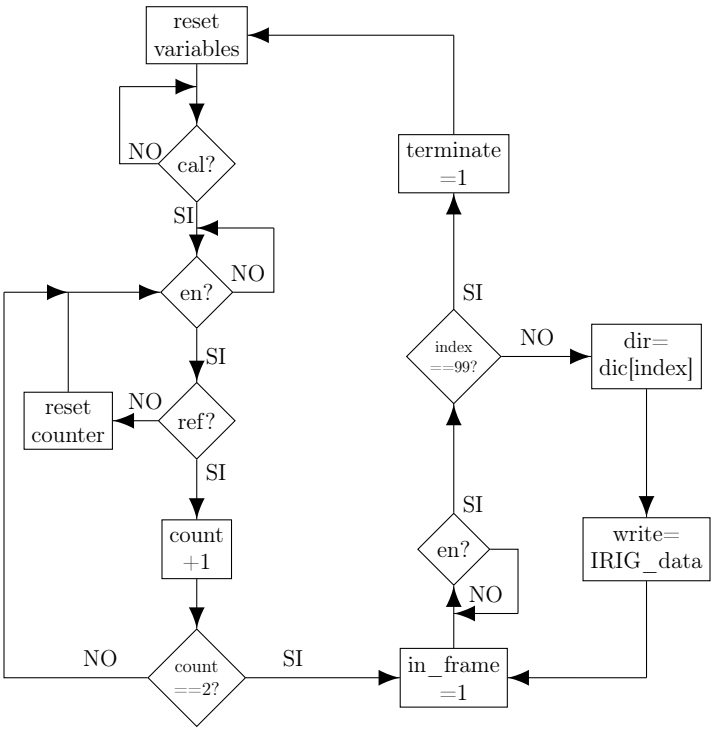


Figura 2.10: Diagrama de flujo para **IRIG_read**.

- **fsm_frac_sec**: Esta maquina funciona como un *debouncer*. Normalmente al existir una transición entre un cero y un uno lógico existe un periodo de tiempo acotado en el cual el sistema digital puede tomar aleatoriamente el valor uno o cero debido a que al encontrarse en un flanco, la señal aún no define su valor. Para la transición de uno a cero también ocurre el mismo efecto. Una forma de lidiar con este efecto es colocar una máquina que verifique la estabilidad de los valores antes de tomarlos como validos. A este tipo de maquinas se les denomina *debouncer*.

2.4. Implementación en hardware

Para la implementación de los sistemas se decide utilizar la plataforma basada en FPGA ROACH2 de la organización CASPER. El entorno de desarrollo seleccionado por CASPER utiliza la herramienta de diseño System Generator de Xilinx para la programación de FPGA. Esta herramienta usa la interfaz Simulink de MATLAB en conjunto con librerías provistas por Xilinx con ISE design Suite dando el soporte requerido para la simulación y compilación de los modelos generados.

El formato de programación es gráfica y consiste en la interconexión de bloques que realizan una determinada función, esto permite construir bloques complejos a partir de los bloques básicos provistos por Xilinx. De esta manera, CASPER posee un conjunto de librerías propias que son generadas y mantenidas por la comunidad que hace uso del hardware de CASPER.

El subsistema de voltímetro vectorial mostrado en la figura 2.1 se implementa utilizando los bloques provistos por CASPER y Xilinx, donde cabe destacar los bloques de ADC5G que funcionan a una frecuencia de muestreo de 2160 MHz, el bloque de decimación polifásica, la

transformación de Fourier 7.1 de Xilinx y los bloques de memoria BRAM que están conectados con un procesador PowerPC, que a su vez esta conectado a una interfaz LAN lo que permite extraer los datos procesados.

La implementación del voltímetro vectorial en el entorno Simulink puede observarse en Anexos.

Para el subsistema de etiquetado de tiempo, los módulos controlados en la figura 2.6 se implementan utilizando bloques de Xilinx y CASPER. Para los bloques controladores del subsistema crea el HDL necesario y luego se utiliza el bloque *Black Box* que permite introducir HDL en el entorno de System Generator.

La traducción desde diagrama de estado a HDL se hace utilizando el lenguaje *Verilog* y pueden encontrarse en Anexos.

La implementación del subsistema de etiquetado de tiempo se puede ver en Anexos.

Capítulo 3

Pruebas a los sistemas

Para verificar el cumplimiento de las métricas en los objetivos se realizan diferentes pruebas que permitan cuantificar el desempeño de los sistemas implementados sobre la plataforma ROACH2.

3.1. Pruebas voltímetro vectorial

3.1.1. Respuesta ante variación de potencia

El diagrama utilizado para la primera prueba al subsistema de voltímetro vectorial se puede observar en la figura 3.1, donde se utilizan dos generadores funcionando a 50 MHz conectados a una ROACH2 y al Vector Network Analizer(VNA) E8364c.

En este montaje uno de los generadores se mantiene a una potencia constante mientras el otro generador realiza un barrido en potencia que recorre el rango (-3, -88) dBm con un espaciado de 0,1 dBm, de manera que exista una diferencia de hasta 85 dB entre las entradas que llegan a la ROACH2. Los códigos utilizados para la medición pueden encontrarse en Anexos.

Luego de realizadas las mediciones se pueden calcular los promedios y desviación estándar sobre las magnitudes y fases medidas por la ROACH2 y calcular la potencia para la cual el sistema alcanza los limites establecidos por los requerimientos (1° de error en la fase). Adicionalmente se mide de manera simultanea las magnitudes y fases de las señales utilizando el VNA para corroborar las mediciones tomadas con el voltímetro implementado.

Para que exista sincronía entre los datos tomados con la ROACH2 y el VNA se utiliza una señal del generador que da aviso cuando se ha concluido una transición de un valor de potencia a otro y se conecta con el disparador externo (*trigger*) del VNA, lo que desencadena la medición.

Para la ROACH2, se utiliza un largo de acumulación de 10, lo que equivale a una tasa de entrega de datos de 1,2 ms. Para el VNA se ocupa un *IF bandwidth* de 10 kHz.

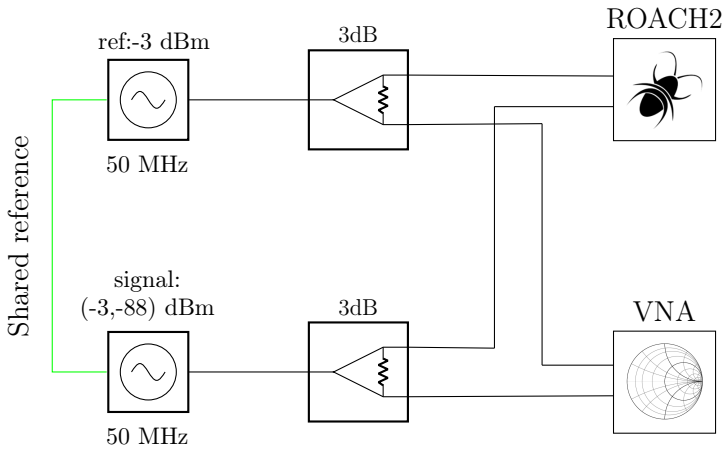


Figura 3.1: Primera prueba a voltímetro vectorial.

Para cada nivel de potencia se toman 1024 mediciones, se adquieren estos datos en un computador y sobre ellas se calculan métricas estadísticas. En la figura 3.2 se pueden observar dichas estadísticas.

En el panel **a** de la figura 3.2 se muestra la curva asociada al promedio de la fase medida en cada una de las potencias. En este gráfico se pude observar que existe un comportamiento escalonado en la evolución de la fase medida. Debido a lo inusual de este comportamiento, se puede suponer que dicho efecto se debe a mecanismos internos del generador al variar la potencia, con lo que de ser cierta esta hipótesis se debiese observar un comportamiento similar en la curva tomada con el VNA.

En el panel **b** se tiene la curva correspondiente al cálculo de la desviación estándar de la fase medida. Para los valores de potencia medidos se tiene una desviación máxima de 3,24° para 85 dB de diferencia entre las entradas. Se alcanza la cota de 1° de error a los 74,2 dB, lo que marca el rango dinámico del sistema.[1]

En el panel **c** se puede observar el cálculo del promedio en la magnitud relativa entre las entradas de la ROACH2. Se puede observar una tendencia lineal al variar la potencia, lo que es esperable, pero aproximadamente a los 45 dB se puede ver una pequeña variación de la pendiente que toma la recta, con lo que nuevamente es necesario contrastar está medición con la de otro instrumento.

En el panel **d** se presenta la curva de la desviación de las magnitudes relativas medidas. Pese a que existen espurios para determinados niveles de potencia, para todos los valores se tienen desviaciones bajo los 0,49 dB.

Es importante notar que las curvas de desviaciones de la figura 3.2 indican que los valores medidos con el modelo son congruentes entre ellos mismos, con lo que previo a la constatación utilizando el VNA estos valores ya entregan información del desempeño del sistema.

[1]Notar que este rango dinámico está tomado utilizando un divisor de potencia que se traduce en una perdida de 3 dB, con lo que sin este dispositivo se espera tener un valor de rango dinámico cercano a los 77,4 dB que son los valores que se obtienen en las secciones subsiguientes.

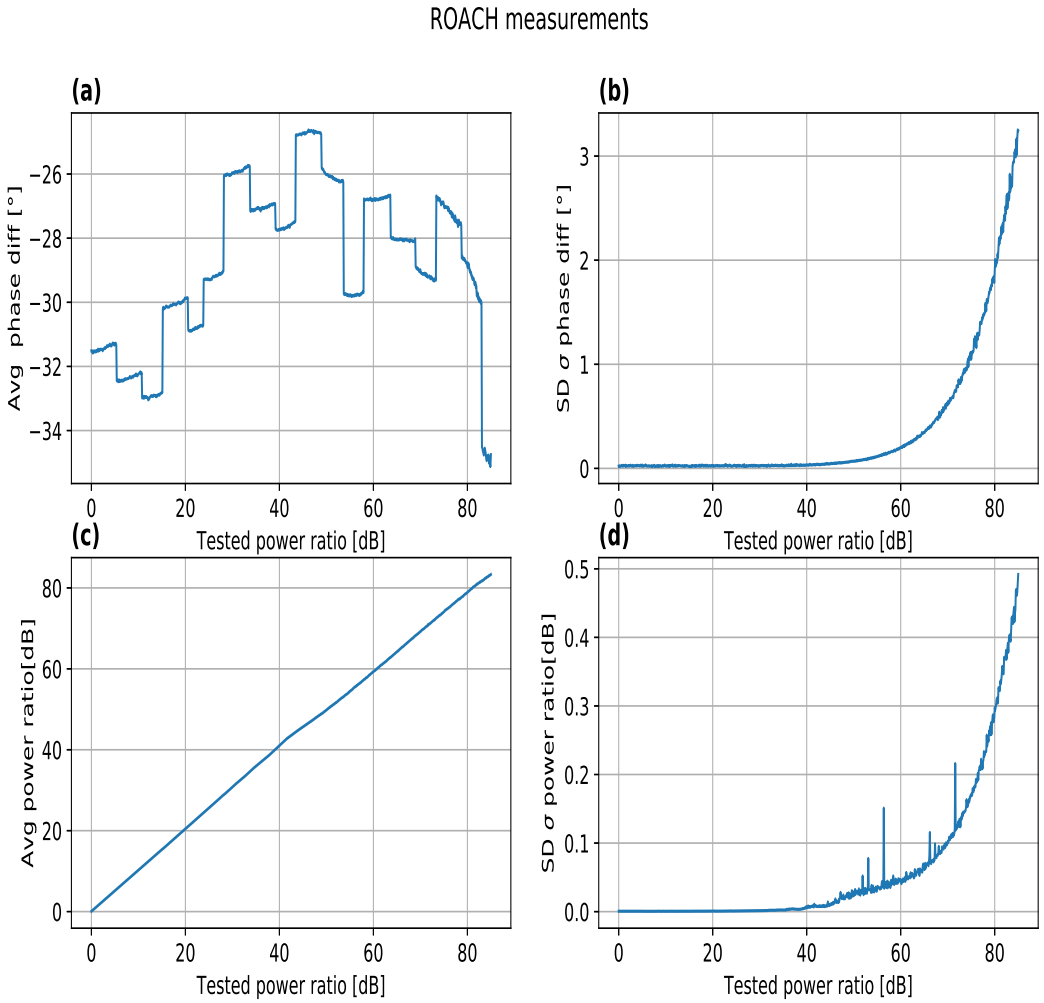


Figura 3.2: Medidas estadísticas para resultados del modelo de voltímetro vectorial en ROACH2, dada la configuración en 3.1. **a**: Promedio fase relativa; **b**: Desviación estándar fase relativa; **c**: Promedio magnitud relativa; **d**: Desviación estándar magnitud relativa.

Los datos recolectados con el VNA se encuentran en la figura 3.3, donde en el panel izquierdo se muestra el valor de la fase relativa entre las entradas y en el panel derecho se puede observar la diferencia entre las magnitudes. En este caso se puede notar nuevamente el comportamiento escalonado en el valor de la fase, pero el instrumento al rededor de los 50 dB comienza a tener variaciones considerables en la fase medida y aproximadamente a los 65 dB la medición se convierte en prácticamente ruido.

En la figura 3.4 se muestran superpuestas las curvas de fase obtenidas utilizando la ROACH2 y el VNA. Se puede notar gráficamente que las curvas están fuertemente correlacionadas, lo que indica que efectivamente las variaciones de fase de forma escalonada se deben a mecanismos internos de los generadores.

Al tener dos mediciones se pueden calibrar las variaciones debidas a configuraciones internas de los generadores restando las curvas medidas con el VNA con las medidas por la

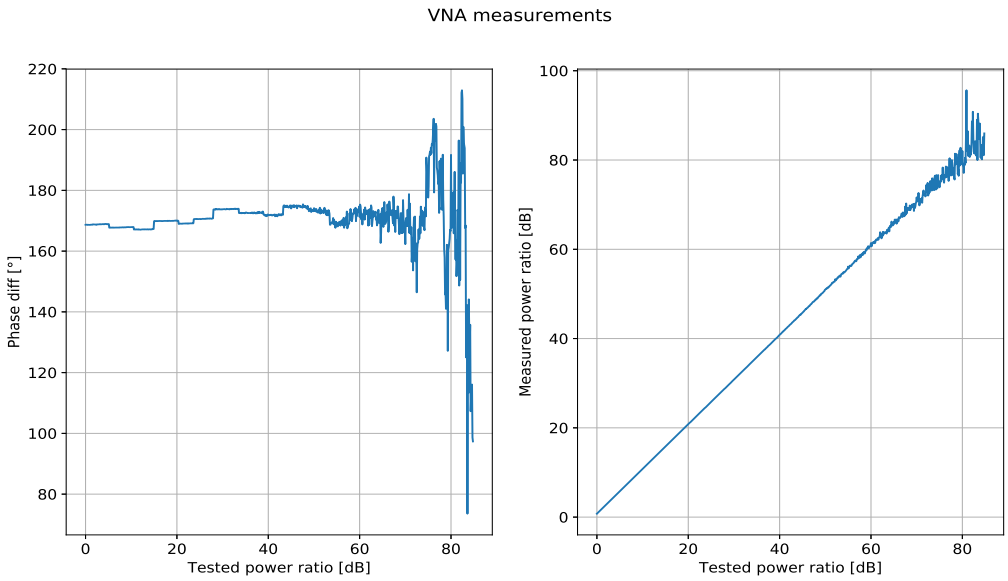


Figura 3.3: Valores tomados con VNA, dada la configuración en la figura 3.1.

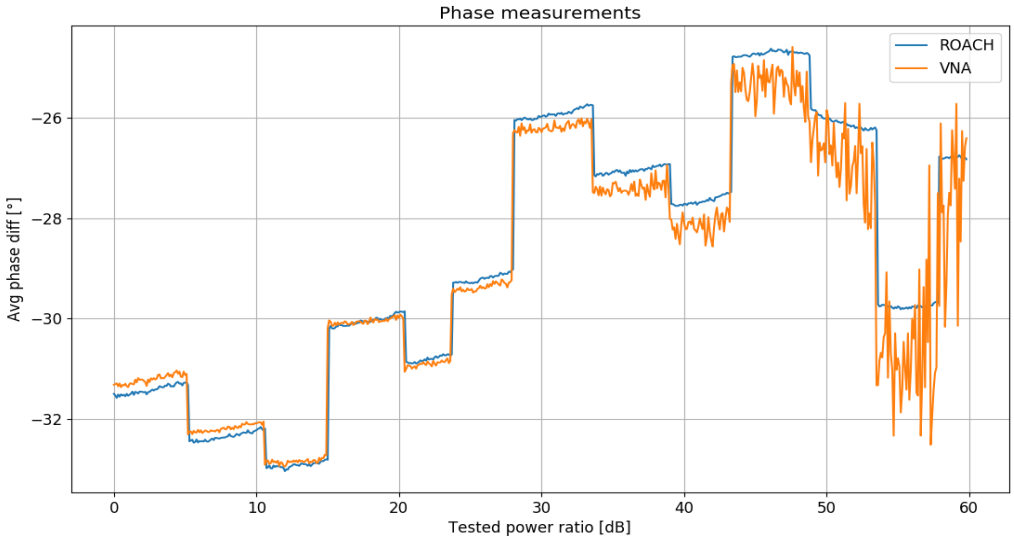


Figura 3.4: Superposición curvas de fase medidas con ROACH2 (azul) y VNA (naranjo).

ROACH2.

La resta de las curvas $ROACH_{curve} - VNA_{curve}$ entregan los gráficos en las figuras 3.5 y 3.6 donde para hacer una buena interpretación de los gráficos hay que considerar que a partir de los 50 dB de diferencia entre las entradas, la respuesta del VNA está dominada por el ruido, con lo que estas curvas están calibradas en la medida que la resolución del VNA lo permite.

De la calibración de la fase en la figura 3.5 se puede decir que se tiene una variación menor a 1° bajo los 50[dB] y traspasado ese umbral aumenta debido a que la curva del VNA no se

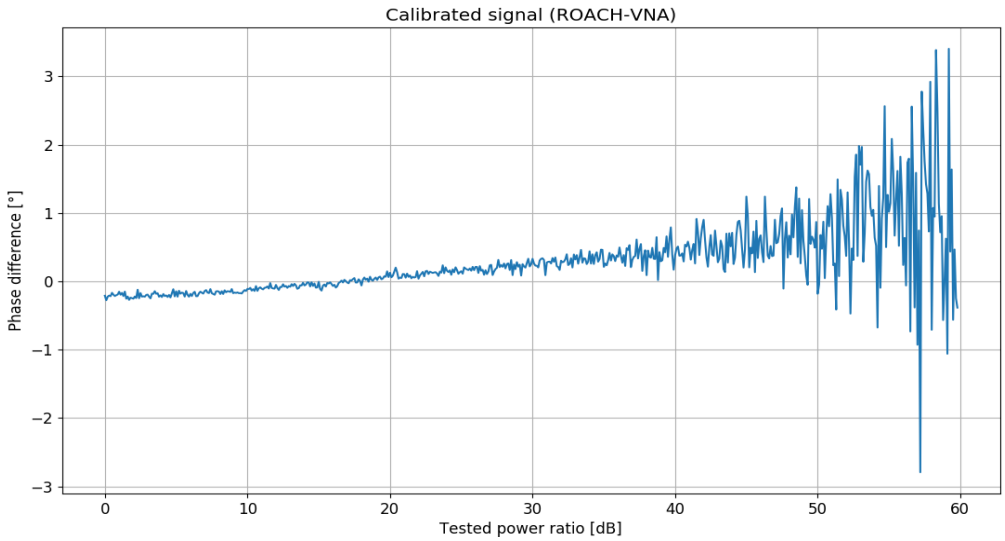


Figura 3.5: Substracción de las curvas de fase relativa entre las entradas medidas con ROACH2 y con VNA.

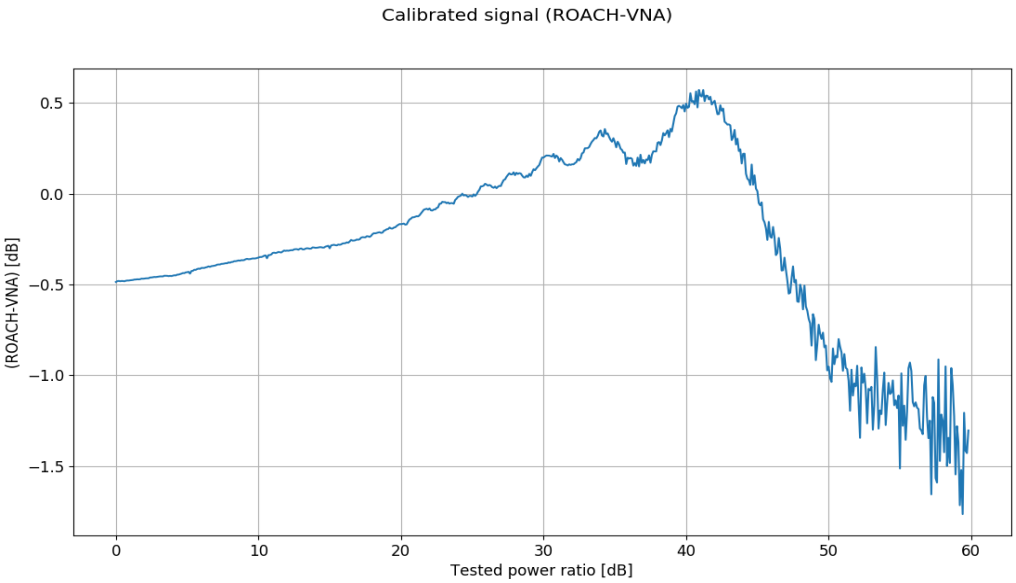


Figura 3.6: Substracción de las curvas de magnitud relativa entre las entradas medidas con ROACH2 y con VNA.

comporta adecuadamente para diferencias de potencia mayores.

Igualmente en la curva de magnitud calibrada en la figura 3.6 se tiene una variación menor a los 0,5 dB para potencias bajo los 50 dB, la cual puede ser explicada por diferentes respuestas en los divisores de potencia utilizados en el montaje.

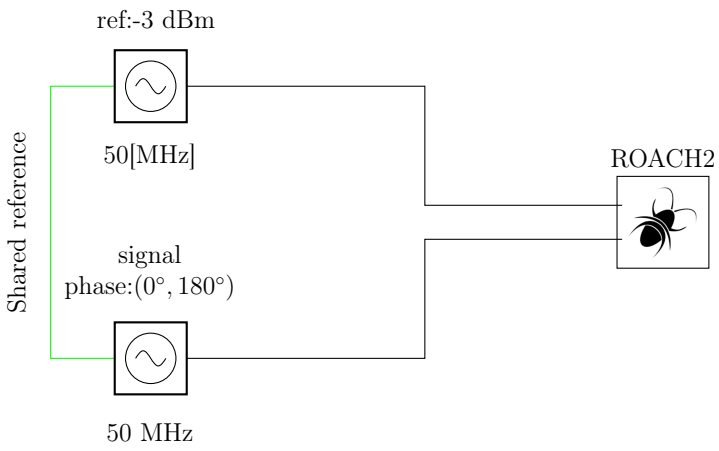


Figura 3.7: Segunda prueba a voltímetro vectorial.

3.1.2. Respuesta ante variación en fase

El diagrama de la prueba realizada se encuentra en la figura 3.7 donde se conectan dos generadores funcionando a 50 MHz a la ROACH2. Posteriormente se cambia la fase de uno de los generadores tomando valores en el rango $(0°, 180°)$ con un espaciado de $10°$. Para cada valor de fase se toman 1024 muestras y se calcula el valor promedio y la desviación estándar.

Se decide realizar esta prueba utilizando dos potencias relativas entre los generadores para poder observar el deterioro de las mediciones al aumentar la diferencia de potencias. Las diferencias de potencia entre las entradas utilizadas son 0 dB y 70 dB.

Los resultados pueden observarse en las figuras 3.8 y 3.9.

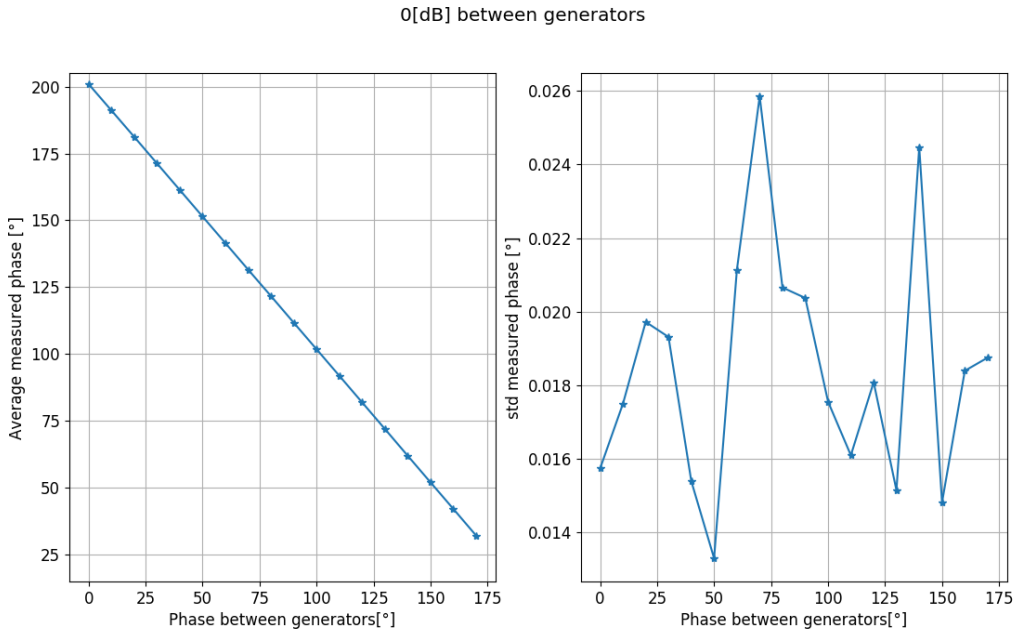


Figura 3.8: Medición de variación de fase con 0 dB entre las entradas.

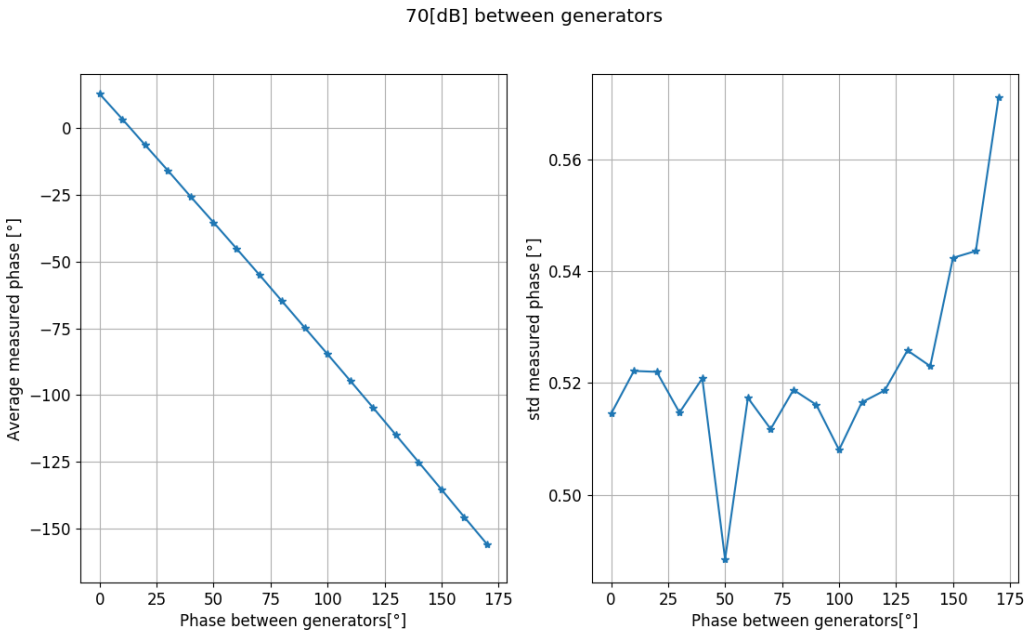


Figura 3.9: Medición de variación de fase con 70 dB entre las entradas.

En la curva de la figura 3.8 se obtiene que la desviación estándar máxima es de 0,026° cuando existen 0 dB de diferencia entre las entradas y en la curva mostrada en la figura 3.9 se tiene una desviación máxima de 0,571° al tener 70 dB. Con ello se cumple nuevamente el requerimiento de tener un rango dinámico sobre los 70[dB] tomando como umbral un error de 1°.

Tabla 3.1: Diferencia de potencia para la cual el sistema alcanza una desviación de 1°.

Frequency[MHz]	Relative power[dB]	distance nearest W_N^k[kHz]
47	78.3	0.488
48	76.9	3.479
49	79.2	1.770
50	78.6	1.220
51	75.8	4.028
52	78.0	1.037

3.1.3. Respuesta ante distintas frecuencias

Como los requerimientos no especifican fuertemente la frecuencia de operación sino que se limita a enunciar cualitativamente un rango de utilización cercano a los 50 MHz, se hace necesario probar la respuesta del sistema ante diferentes frecuencias.

Para ello se utiliza el montaje en la figura 3.7 manteniendo la fase constante en ambos generadores y haciendo una variación en potencia en el rango $(-3, -88)$ dBm con un espaciado de 0,1 dBm.. Para cada uno de los valores se toman 1024 muestras y se calculan los valores promedios y de desviación estándar. Luego se cambia la frecuencia de operación de los generadores y se repite el procedimiento. Los resultados de los cálculos de las variables estadísticas de esta prueba se encuentran en las figuras 3.10 y 3.11.

En el panel izquierdo de la figura 3.10 que para diferencias mayores de ~ 50 dB se tienen cambios de pendientes en las curvas de cada frecuencia. Del panel derecho de la figura 3.10 se puede ver que hay diferencias en el comportamiento de las curvas de desviación estándar para cada frecuencia, donde la mayor desviación se tiene para las frecuencias 48 MHz y 51 MHz y las menores desviaciones se tienen para las frecuencias 49 MHz y 50 MHz.

En el panel izquierdo de la figura 3.11 se puede observar que los valores promedios de la fase tienen formas similares pero existe un desplazamiento vertical de las curvas. Esto puede atribuirse a que al tener el largo de los cables fijos al cambiar las frecuencias hace que las señales lleguen con distinto desfase, pero aún presentan las variaciones que aparecen a partir del funcionamiento interno de los generadores.

Es importante notar en el panel izquierdo de la figura 3.11 existen diferencias en los comportamientos de las fases promedio de cada curva. Esto se condice con la información que entregan las curvas de desviación estándar que se encuentran en el panel derecho de la figura 3.11 donde las frecuencias 48 MHz y 51 MHz tienen las mayores curvas de desviación estándar, y las curvas de las frecuencias 50 MHz y 49 MHz poseen los menores valores de desviación.

Para cuantificar el desempeño del sistema ante diferentes frecuencias se busca cuando la desviación de la fase alcanza un error correspondiente a 1° que es el valor umbral para determinar el rango dinámico establecido por los requerimientos. La información está contenida en la tabla 3.1.

La variaciones en el rango dinámico mostradas en la tabla 3.1 tienen su explicación en el

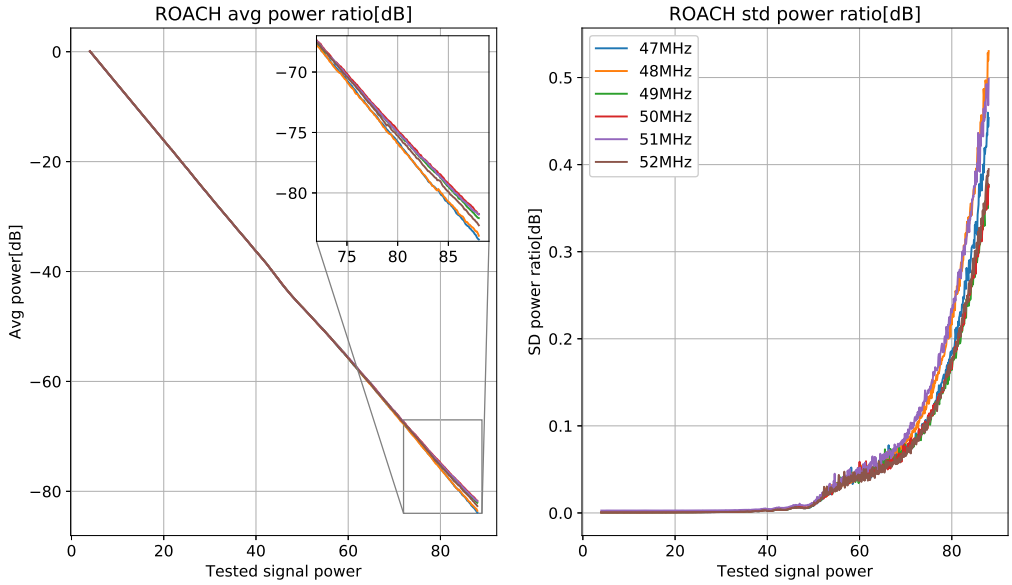


Figura 3.10: Métricas estadísticas para la medición de la magnitud relativa entre las entradas para diferentes frecuencias.

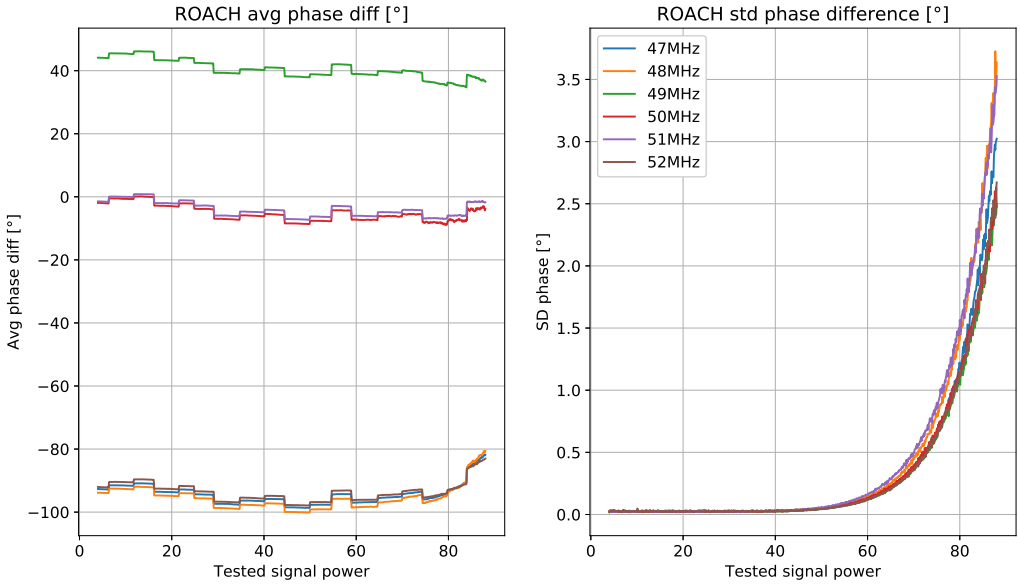


Figura 3.11: Métricas estadísticas para la medición de la fase relativa entre las entradas para diferentes frecuencias.

efecto de esparcimiento espectral de la DFT. Básicamente las peores respuestas corresponden a las frecuencias que se encuentran más alejados de los *twiddle factors*, por ejemplo para $51[MHz]$ y $48[MHz]$ se encuentran en medio de los canales espectrales que es el peor caso posible ya que la potencia se distribuye mayoritariamente en dos canales espectrales, esto explica que el umbral de $1°$ se alcance antes. En tanto frecuencias que se encuentran cercanos a los *twiddle factors* poseen mejores respuestas. De cualquier forma, en todas las frecuencias se cumple el requerimiento de tener un rango dinámico sobre los $70[\mathrm{d}B]$ ya que en todas las frecuencias el error de $1°$ se alcanza pasado dicho umbral.

3.1.4. Respuesta ante ruido

Debido a que en el uso real del voltímetro vectorial al hacer holografía se trabaja en un entorno con ruido en esta prueba se intenta observar el desempeño en un ambiente con ruido agregado.

Otro tema a considerar, es que en la aplicación en terreno la señal de referencia es captada por una antena de referencia. Dado que al momento de hacer la holografía la antena estará en movimiento no se puede asegurar que la señal de referencia sea captada con la máxima ganancia.

Por tanto en esta prueba se busca ver la respuesta ante distintos valores de potencia en la señal de referencia.

El diagrama del montaje utilizado puede verse en la figura 3.12 donde se utilizan dos generadores de ruido blanco cuyo *Excess Noise Ratio*(ENR) es de aproximadamente 15 seguido de un filtro pasabajos con banda de paso de (0-70) MHz que luego entra a una cadena de amplificación para finalmente ingresar a un combinador de potencia que suma el ruido con la señal proveniente de un generador para entrar a la ROACH2.

La potencia integrada de ruido añadida a la referencia es de $-27{,}5$ dBm y para la señal se adicionan $-44{,}5$ dBm. En la figura 3.13 se muestra el efecto de la adición de ruido en el piso de ruido del voltímetro vectorial.

Para la prueba se selecciona un valor de potencia para la referencia y se realiza una variación de potencia en la señal primaria, se toman los datos y se calcula el valor promedio y desviación estándar para la magnitud relativa y la fase relativa entre las entradas.

La variación en potencia en la señal toma valores en el rango $(-3, -88)$ dBm con un espaciado de 1 dBm, mientras que la referencia toma valores en el rango $(-3, -88)$ dBm con un espaciado de 5 dBm. [2]

Los resultados de estas mediciones pueden observarse en las figuras 3.14, 3.15, 3.16 y 3.17.

[2]Es importante mencionar que en la aplicación real la señal de referencia no debiese variar más de 10 dB al ser medida. De esta manera realizar el barrido de potencia entre $(-3, -88)$ dBm en la señal de referencia es solo para la caracterización del sistema.

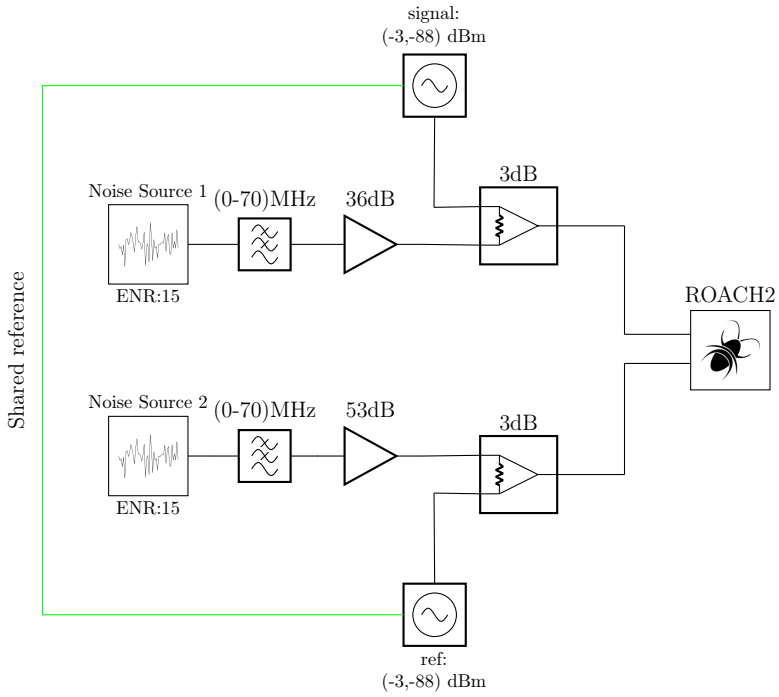


Figura 3.12: Prueba de adición de ruido a señales ingresadas a voltímetro vectorial.

En la figura 3.14 se puede observar que la magnitud relativa promedio posee un comportamiento concordante con lo esperado, ya que al disminuir la potencia de la señal de referencia existe un corrimiento vertical en las curvas igual a la disminución en la referencia.

En la figura 3.15 se observa que la desviación estándar en la magnitud relativa tiene un decaimiento con respecto a la curva de desviación de la primera prueba al voltímetro vectorial mostrada en la figura 3.2 medida sin ruido. Lo notable de la relación de estas curvas es que el deterioro al utilizar una referencia en el rango $(-3, -23)$ dBm es bastante similar. Además cabe recalcar que el deterioro para estas curvas es equivalente a desplazar hacia la izquierda la curva de desviación de potencia medida sin ruido en la figura 3.2 en ~ 20 dB, que observando la figura 3.13 corresponde al nivel de ruido adicionado por la fuente de ruido en la señal de calibración. Así por ejemplo, para la medición sin ruido se tiene una desviación de 0,2 dB en 80 dB de potencia relativa entre las entradas, mientras que en las mediciones con ruido el error de 0,2 dB se alcanzan con 60 dB de diferencia relativas.

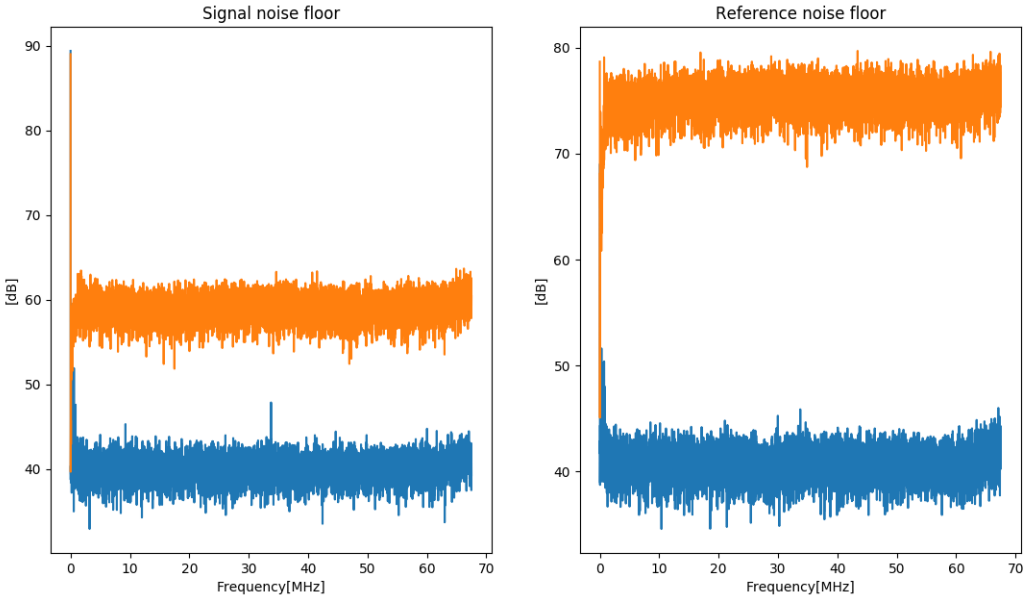


Figura 3.13: En azul piso de ruido sin fuente de ruido encendida, en anaranjado piso de ruido con fuente de ruido encendida.

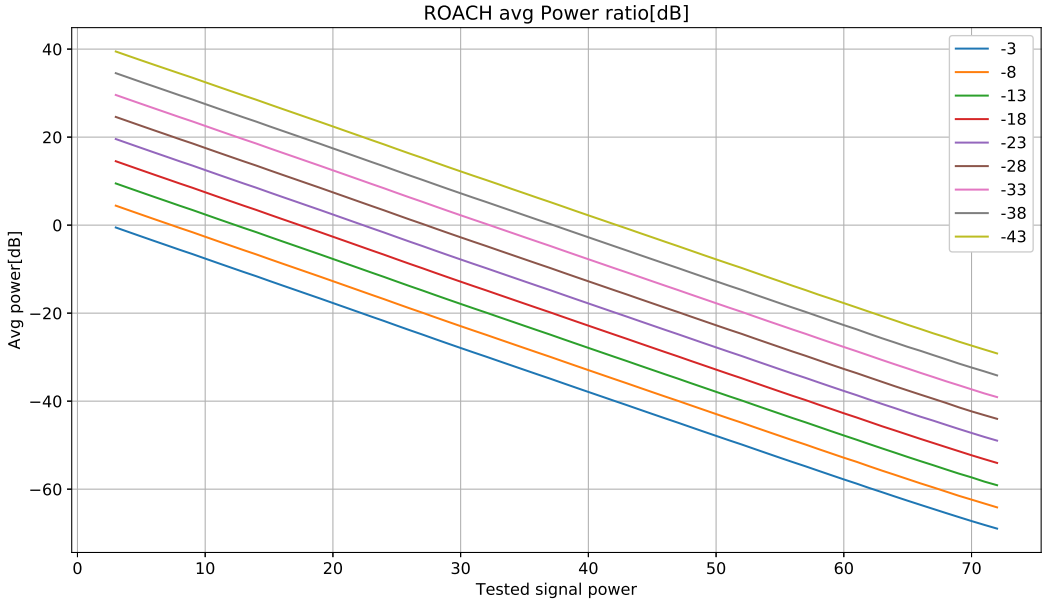


Figura 3.14: Curvas de promedio de magnitud relativa entre las entradas para diferentes valores de potencia en la referencia.

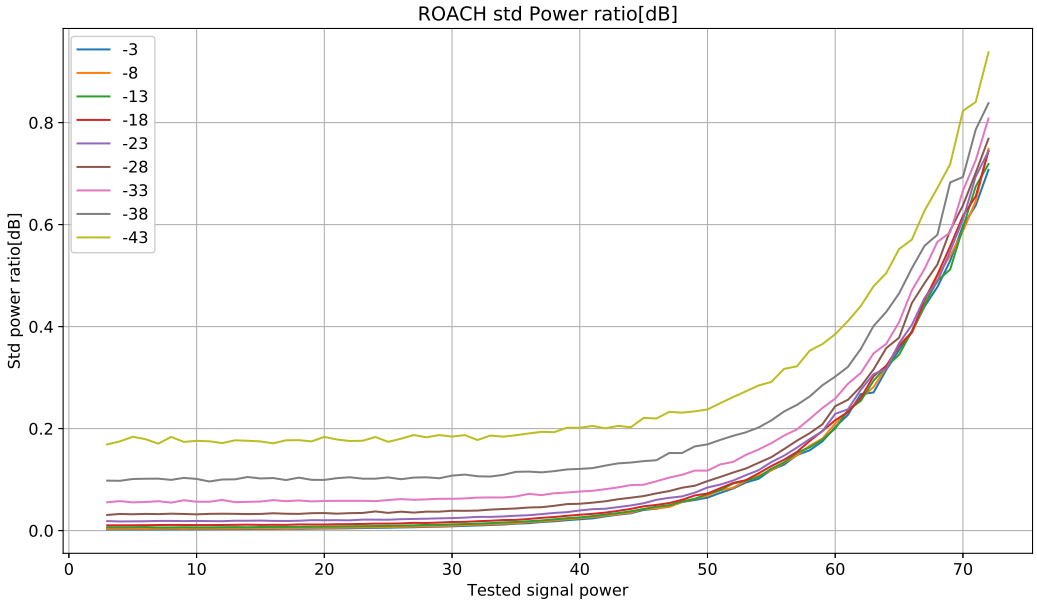


Figura 3.15: Curvas de desviación estándar de magnitud relativa entre las entradas para diferentes valores de potencia en la referencia.

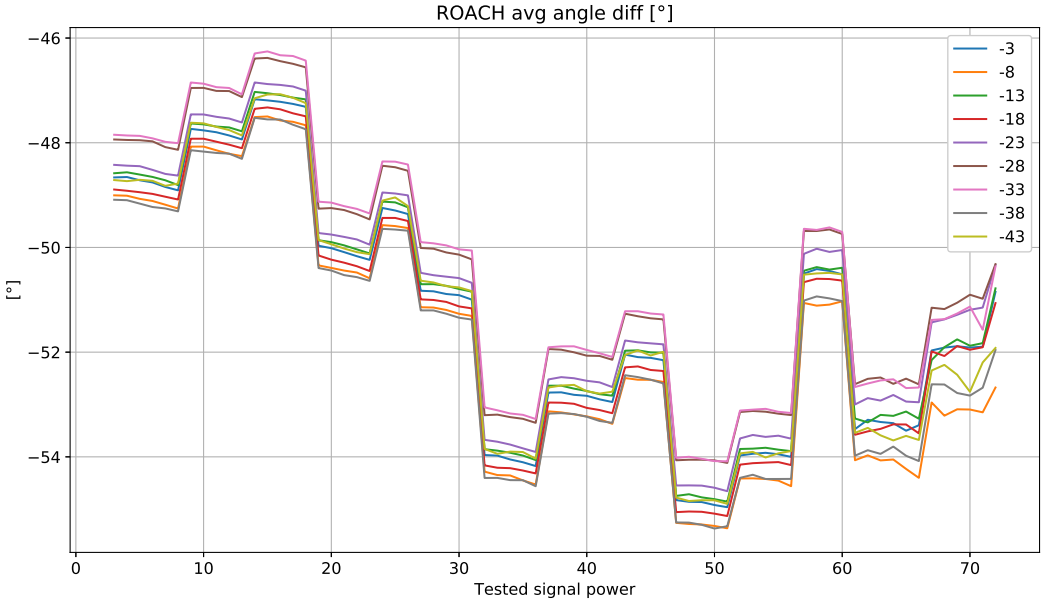


Figura 3.16: Curvas de promedio de fase relativa entre las entradas para diferentes valores de potencia en la referencia.

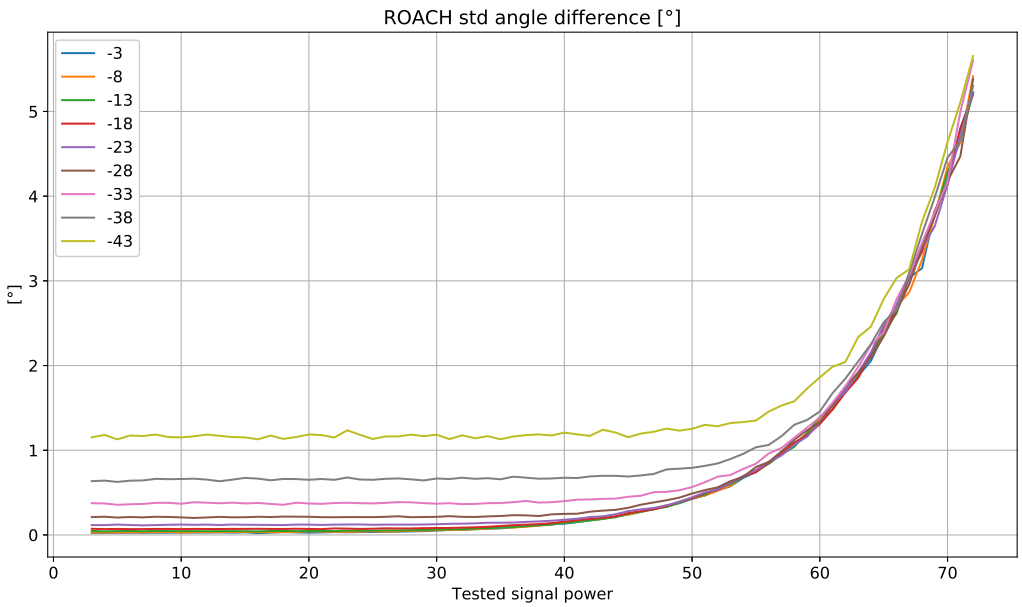


Figura 3.17: Curvas de desviación estándar de fase relativa entre las entradas para diferentes valores de potencia en la referencia.

En la figura 3.16 se puede ver que todas las curvas de promedio de la fase relativa poseen el mismo comportamiento escalonado que se tiene en la figura 3.2 pero nuevamente existe un desplazamiento vertical entre las curvas. Que se mantenga la forma de las curvas es esperable, ya que corresponde a la respuesta ante variación de potencia del generador que estudiado en la subsección 3.1.1. Mientras que el desplazamiento vertical de las curvas se puede explicar ya que el generador que produce la señal de referencia también posee su propia curva de respuesta ante la variación de potencia.

En la figura 3.17 se observa que la desviación estándar de la fase sufre de un deterioro considerable en comparación con la figura 3.2. Al igual que en la desviación de la magnitud se tiene que el efecto de variar la potencia en el rango $(-3, -23)$ dBm no introduce un mayor cambio en la estadística, y nuevamente las curvas pertenecientes a dicho rango de potencia en la referencia corresponde a un desplazamiento en ~ 20 dB a la izquierda en la curva de desviación estándar en la figura 3.2 medida sin ruido.

3.2. Pruebas marcado de tiempo

Para probar el subsistema de marcado de tiempo se utiliza el reloj GPS **Xli-Time frequency**, que entrega el código IRIG-B000 por uno de sus puertos y además posee un puerto de **Pulse Per Second**(PPS) que corresponde a una señal que emite un pulso de duración $20 \pm 1 \mu$s en cada segundo. El código IRIG es emitido con $5V_{pp}$ en fase con la señal $PPS \pm 200$ ns.

La prueba preliminar consiste en calibrar el subsistema de marcado de tiempo utilizando los pines de propósito general (GPIOs) de la ROACH2 y que el tiempo que se lee en los

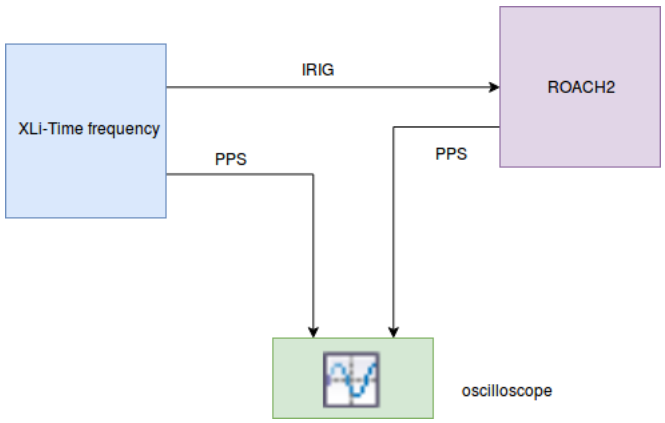


Figura 3.18: Prueba para el subsistema de marcado de tiempo.

registros de la ROACH2 sea consistente con el tiempo en el reloj Xli. Esto se cumple empíricamente e indica que los paquetes de IRIG son correctamente interpretados por el sistema, pero debido a que el monitor en el reloj Xli tiene como unidad mínima los segundos con esta prueba preliminar solo se puede asegurar la concordancia en dicho orden de magnitud.

La siguiente prueba consiste en evaluar el desempeño para magnitudes bajo el segundo. Para ello se utiliza el montaje en la figura 3.18. Es este diagrama se utiliza un GPIO de la ROACH para recibir la señal de IRIG desde el reloj Xli. Luego de calibrado el subsistema se utiliza otro GPIO para generar un PPS en la ROACH2.

De esta manera se puede conectar el PPS generado por el reloj Xli y el PPS producido por la ROACH2 en un osciloscopio y comparando ambas señales se puede obtener el error relativo de actualización de los segundos en la ROACH2 con respecto al reloj Xli.

El resultado de la prueba propuesta para el sistema de marcado de tiempo puede observarse en la figura 3.19 donde se tiene una diferencia entre ambas señales de 0,957μs. Las resonancias que se observan en la figura 3.19 se debe a que los GPIOs no estan diseñados para funcionar a altas frecuencias con lo que la transición de $1-0$ no es perfecta. [3]

Para ver la estabilidad del sistema se coloca como límite inferior al sistema un valor de 0,9999 y un límite superior 1,0001 en el módulo *clock stability* en 2.3. Estos límites se comparan con el número de cuentas que se toman entre un PPS de la ROACH2 y el siguiente PPS, es decir mide las cuentas que se toman en un segundo para determinar si el reloj se mantiene estable. Para estos valores umbrales no se presentan alertas de desenclave en 40 minutos. Además, en esos 40 minutos se revisa periódicamente el resultado que entrega el PPS de la ROACH2 y el reloj Xli en el osciloscopio y se tienen resultados del mismo orden de 3.19.

[3]Este es uno de los motivos por los que en el diseño del sistema de marcado de tiempo se coloca un **debouncer**.

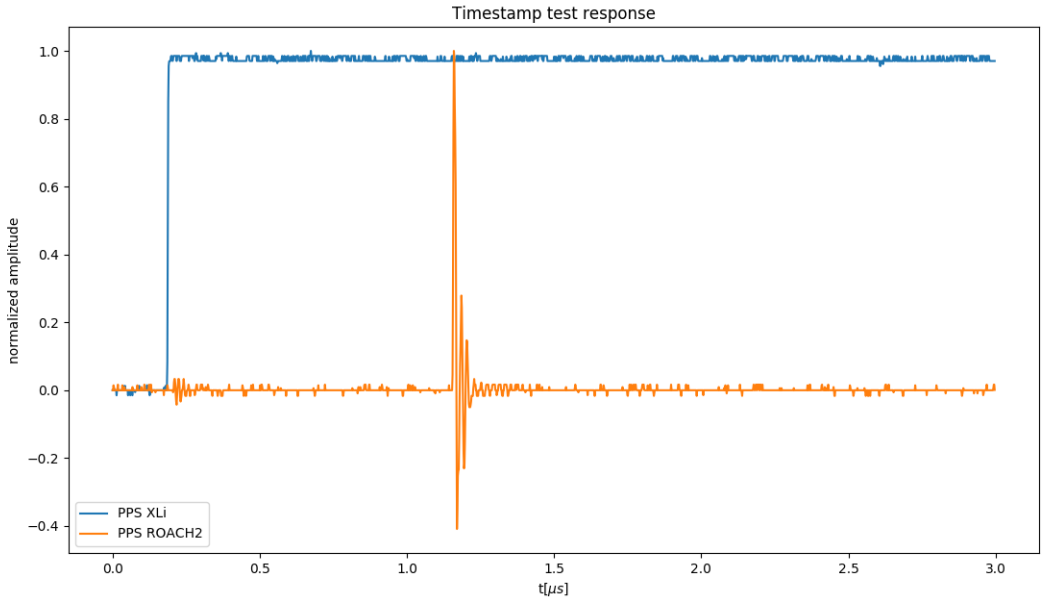


Figura 3.19: Resultados para prueba en figura 3.18 medido en el osciloscopio.

Conclusión

En este trabajo se describe el proceso de diseño, implementación y pruebas de un voltímetro vectorial y de un sistema de marcado de tiempo en la plataforma basada en FPGA ROACH2 de CASPER diseñado para ser utilizado en el proceso de holografía para el telescopio CCAT-prime.

En la subsección Respuesta ante variación de potencia se comparan las mediciones del voltímetro vectorial implementado con un Vector Network Analizer obteniendo coherencia entre las mediciones tomadas, lo que indica un correcto funcionamiento del voltímetro vectorial. De manera adicional, para las configuraciones de esta prueba el voltímetro vectorial desarrollado presenta un mejor desempeño que el VNA.

Las pruebas en la subsección Respuesta ante distintas frecuencias muestran que el rango dinámico del voltímetro vectorial es de 75,8 dB en su peor desempeño y de 79,2 dB al operar en condiciones óptimas, cumpliendo el requerimiento de un sistema con rango dinámico sobre los 70 dB. La variación en el rango dinámico puede explicarse mediante el efecto de esparcimiento de frecuencia que distribuye potencia en todos los canales espectrales de la DFT. De esta manera se obtienen mejores desempeños para frecuencias cercanas a los *twiddle factors* de la DFT y por lo tanto se recomienda operar el voltímetro vectorial en dichas frecuencias.

En Respuesta ante ruido se estudió la respuesta del sistema al trabajar en entornos ruidosos y teniendo distintos valores de potencia en la señal de referencia. De estas mediciones se obtiene un deterioro en el rango dinámico concordante al nivel de ruido adicionado y que para señales de referencia en el rango $(-3, -23)$ dBm no existe un degrado significativo en la precisión del sistema.

Para el subsistema de marcado de tiempo se utilizó el protocolo IRIG-B000 y se diseñaron e implementaron los módulos necesarios, utilizando para ello técnicas clásicas de diseño de sistemas digitales.

Para examinar la resolución del marcado de tiempo, se genera una señal de actualización de segundo por la ROACH2 y se compara con el PPS producido por el reloj maestro con el que se calibró previamente el sistema. Con esta metodología se mide un desfase de 0,957μs con lo que se cumple el objetivo de una resolución mejor que 0,1 ms.

El trabajo futuro para el proyecto de voltímetro vectorial consiste en el desarrollo de los mecanismos de almacenamiento y extracción de los datos tomados en una campaña de calibración sin perdida de información. Debido a que una campaña tiene una duración de

aproximadamente 30 minutos, es necesario idear una manera de almacenar y/o traspasar estos datos para que sean post-procesados.

Dado que la señal de referencia puede verse sujeta a variaciones de frecuencia es deseable almacenar un número variable de canales espectrales y/o hacer una selección en tiempo real para identificar que canal espectral debe guardarse en memoria.

Otra mejora posible al modelo consiste en adaptar los filtros *anti-aliasing* de la etapa de decimación para alcanzar el SNR teórico de 97,8 dB. Debido a que el rechazo de los filtros actuales es de aproximadamente 80 dB se permite que entre ruido de ese nivel a la banda de interés por medio de aliasing

A modo de conclusión general en este trabajo se logra el cumplimiento de los objetivos fijados y se verifica la hipótesis que da inicio a esta memoria. Se obtuvo como resultado un voltímetro vectorial de alto rango dinámico implementado sobre una plataforma basada en FPGA específicamente diseñado y probado para operar la banda de frecuencias requeridas por el proceso de holografía de CCAT-prime y que posee un mejor desempeño que un VNA en esta aplicación, lo que transforma al voltímetro vectorial implementado en una solución atractiva por su relación de costo-beneficio.

Bibliografía

[1] Y. Rahmat-Samii. Surface diagnosis of large reflector antennas using microwave holographic metrology: An iterative approach. *Radio Science*, 19(5):1205–1217, September 1984.

[2] Carlo Neri, Fabio Pollastrone, and Onofrio Tudisco. Fully Digital implementation of a high dynamic fast Vector Voltmeter. In *2009 23rd IEEE/NPSS Symposium on Fusion Engineering*, pages 1–4, San Diego, CA, USA, June 2009. IEEE.

[3] Urs U. Graf Richard E. Hills, Nicolas A. Reyes and S. Parshley. Plan for holography measurements of ccat-prime using an artificial source. Internal document for the CCAT-P holography design.

[4] Bajaj Ronak and Suhaib A Fahmy. Mapping for maximum performance on fpga dsp blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):573–585, 2015.

[5] John F Wakerly. *Digital design*. Pearson Australia Pty Limited, 2016.

[6] Francisco Javier Rivera Serrano. Diseño e implementación de la correlación y de la correntropía cruzada, utilizando fpga. Master's thesis, Universidad de Chile, 2017.

[7] Richard G. Lyons. *Understanding Digital Signal Processing*, chapter 10.- Sample Rate Conversion. 3rd edition.

[8] Norbert J Fliege. *Multirate digital signal processing*, volume 994. John Wiley New York, 1994.

[9] Richard G. Lyons. *Understanding Digital Signal Processing*, chapter 13.-Digital Signal Processing Tricks. 3rd edition.

[10] Christopher Harris and Karen Haines. A Mathematical Review of Polyphase Filterbank Implementations for Radio Astronomy. *Publications of the Astronomical Society of Australia*, 28(4):317–322, 2011.

[11] U.S. Army White Sands Missile Range. *IRIG SERIAL TIME CODE FORMATS*, September 2004.

[12] D.J. Rochblatt and B.L. Seidel. Microwave antenna holography. *IEEE Transactions on*

Microwave Theory and Techniques, 40(6):1294–1300, June 1992.

[13] Nick Mehta. Xilinx 7 series fpgas: the logical advantage. *Xilinx WP405*, 2012.

[14] William I Fletcher. An engineering approach to digital design. 1980.

[15] Alan V Oppenheim, John R Buck, and Ronald W Schafer. *Discrete-time signal processing. Vol. 2.* Upper Saddle River, NJ: Prentice Hall, 2001.

[16] Ronald E Crochiere and Lawrence R Rabiner. Interpolation and Decimation of Digital Signals- A Tutorial Review. *PROCEEDINGS OF THE IEEE*, 6(3):32, 1981.

[17] Fred Harris. *Multirate signal processing for communication systems*. Prentice Hall PTR, Upper Saddle River, N.J, 2004.

[18] Prof. Brad Osgood. Lecture notes for ee261 the fourier transform and its applications.

[19] Michael Lamoureux Nicolas Lerner Joachim Toft Hans G. Feichtinger, Bernard Helffer. *Pseudo-Differential Operators: Quantization and Signals*.

[20] Julius O Smith. Spectral audio signal processing, october 2008 draft. *Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University, Stanford, California*, 94305, 2008.

Anexos

Paquete de datos IRIG

Figura A 1: Formato de paquete de datos de IRIG.

Diseño subsistemas de marcado de tiempo

Módulo index y pulse count

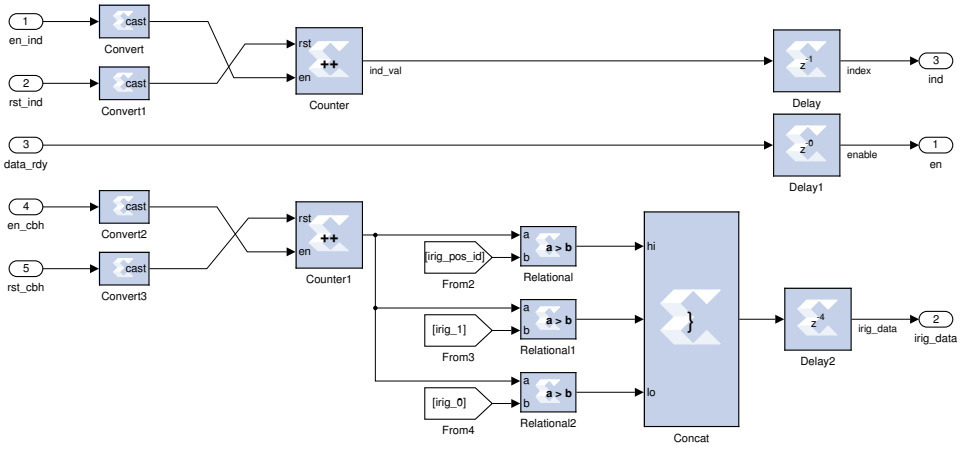


Figura A 2: Implementación de módulo **index** y **pulse count**.

En la figura A 2 se muestra el módulo **index** y **pulse count** que son controlados por la máquina de estado **Parse_input**. El contador ubicado en la zona inferior en la figura cuenta la duración del pulso de IRIG y luego en el comparador a la derecha se decodifica la información. El contador en la zona superior de la figura lleva la cuenta del índice actual del pulso que se está tratando. La señal **data_rdy** entrega un señal de validación que llega a la máquina de estado **IRIG_read**.

Módulo BCD to TOY

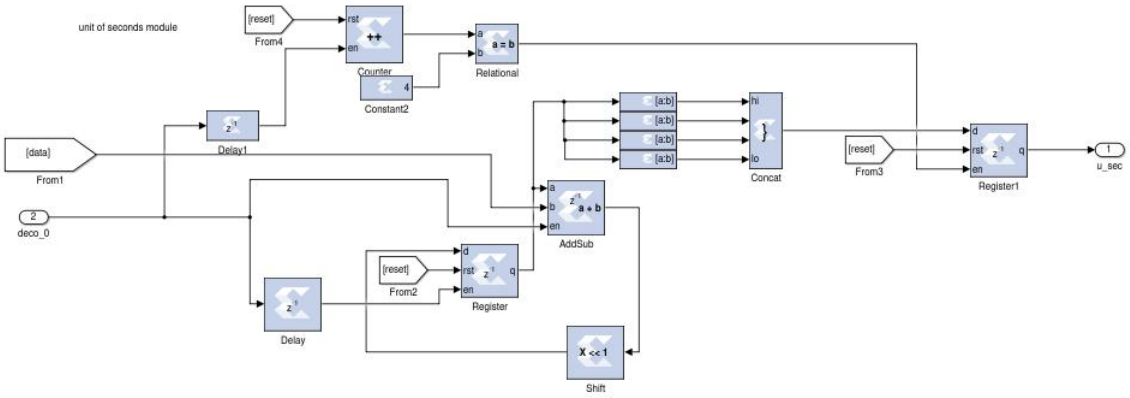


Figura A 3: Parte del subsistema **BCD to TOY**.

La lógica en A 3 es el bloque elemental que tiene como tarea pasar la información de formato *big endian* a *little endian*. Básicamente, al llegar un valor nuevo se escribe en un registro y se realiza un ordenamiento de los bits hasta que todos los bits son recibidos.

Debido a que en BCD los valores se dividen en magnitudes como unidades, decenas de segundos, unidades de minuto, etcétera, se replica la unidad básica en la figura A 3 para cada una de estas magnitudes.

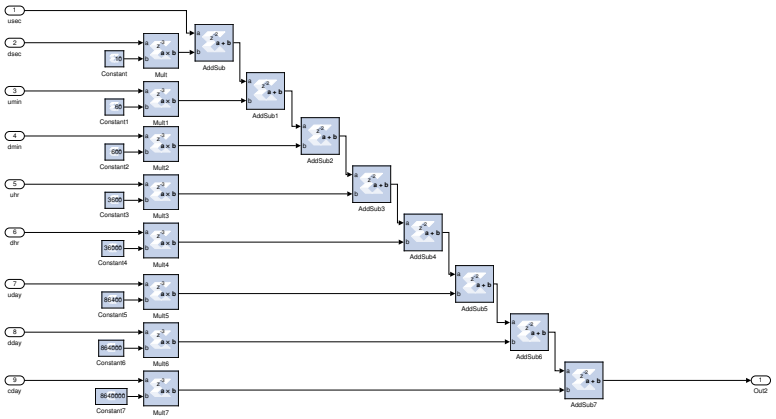


Figura A 4: Parte de subsistema **BCD to TOY**.

En la figura A 4 se tiene la parte final del módulo **BCD to TOY** donde se pondera cada magnitud utilizada en IRIG (por ejemplo unidades de segundo, unidades de minutos, etcétera) por un factor adecuado para llevar la información a su equivalente en segundos.

Módulos fraction_counter y check stability

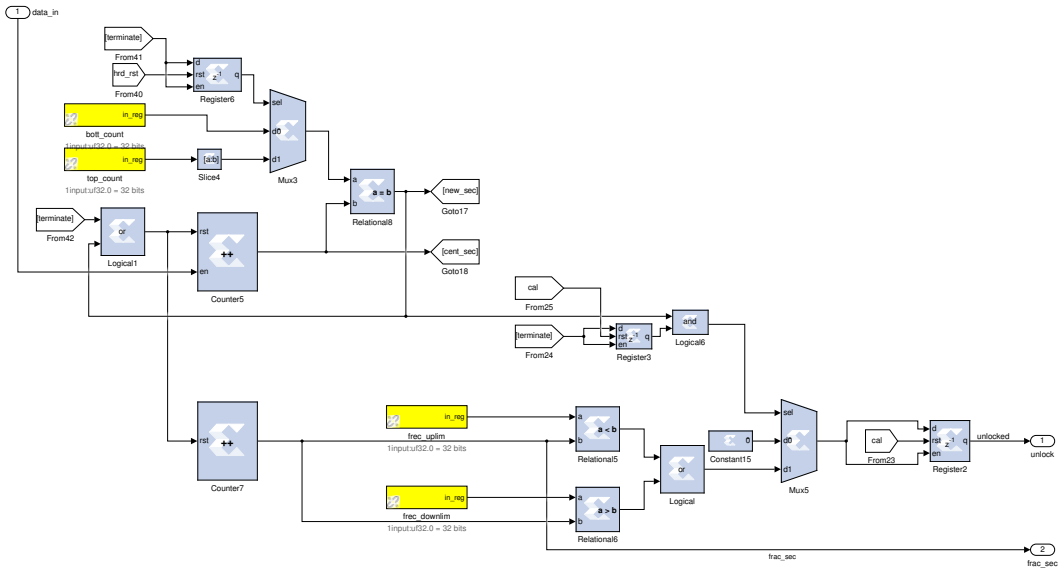


Figura A 5: Módulos **fraction_counter** y **check stability**

En la figura A 5 están las implementaciones de **fraction_counter** y **check stability**.

En este modelo existen dos contadores, el superior cuenta los pulsos dentro del paquete de datos de IRIG y compara su valor con los registros configurados por el usuario. Para un correcto funcionamiento el registro **bott_count** y **top_count** deben programarse con un valor 100, de manera que cuando el contador alcance dicho valor genere una señal que indica el comienzo de otro segundo.

El contador que está en la parte inferior de la figura A 5 funciona de manera libre y se reinicia cada vez que el contador superior desencadena la señal de nuevo segundo. Con ello su conteo equivale a fracciones de cada segundo y se utiliza de dicha forma.

Además al desencadenarse la señal de nuevo segundo se compara el valor del contador inferior con dos registros que demarcan el rango de valores aceptables que debe tener el contador de sub-segundos. Si el contador no se encuentra dentro del rango se entrega una señal de alarma.

Diagrama de estados para Parse_input

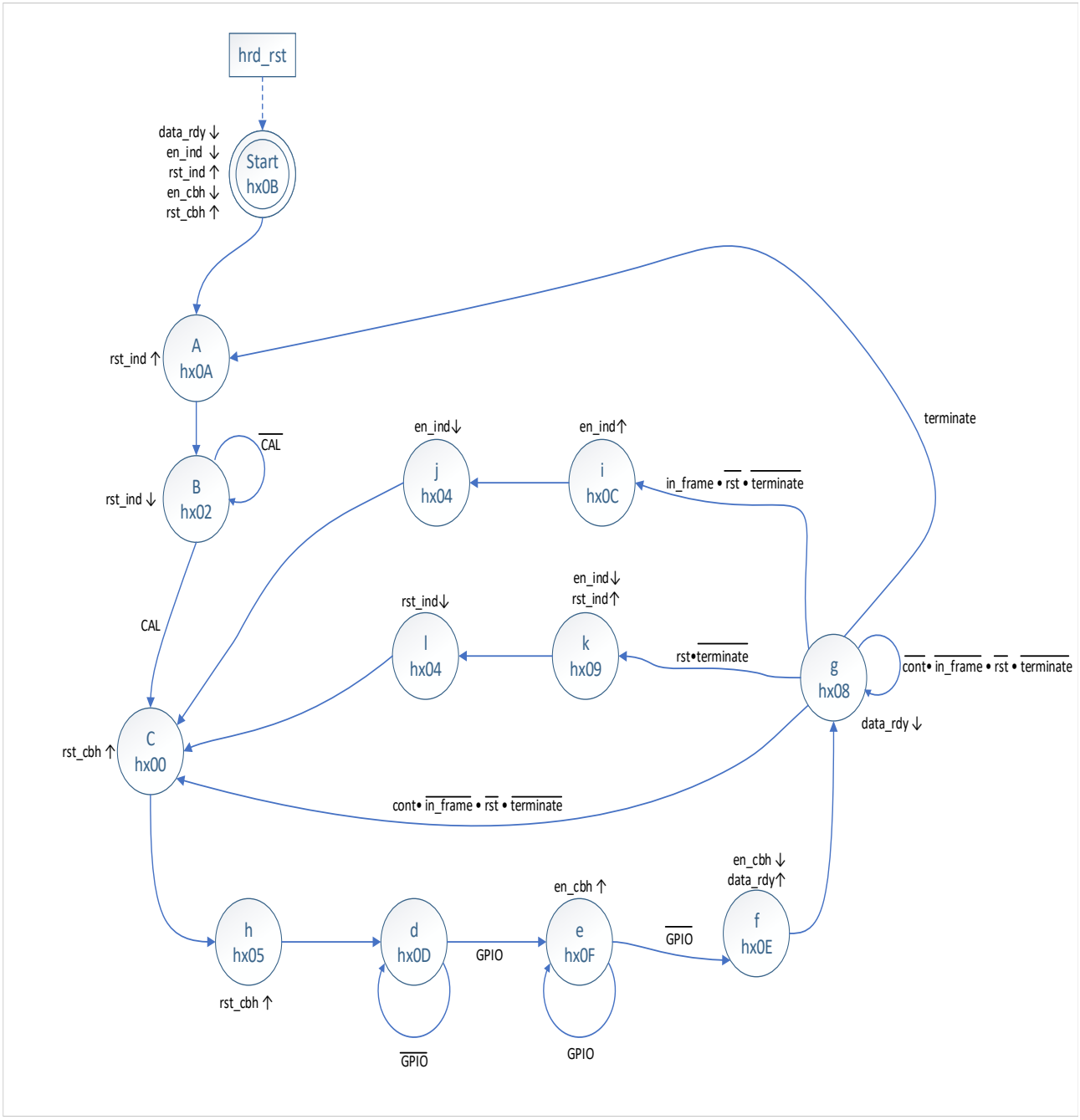


Figura A 6: Diagrama de estados de **Parse_input**.

Diagrama de estados para IRIG_read

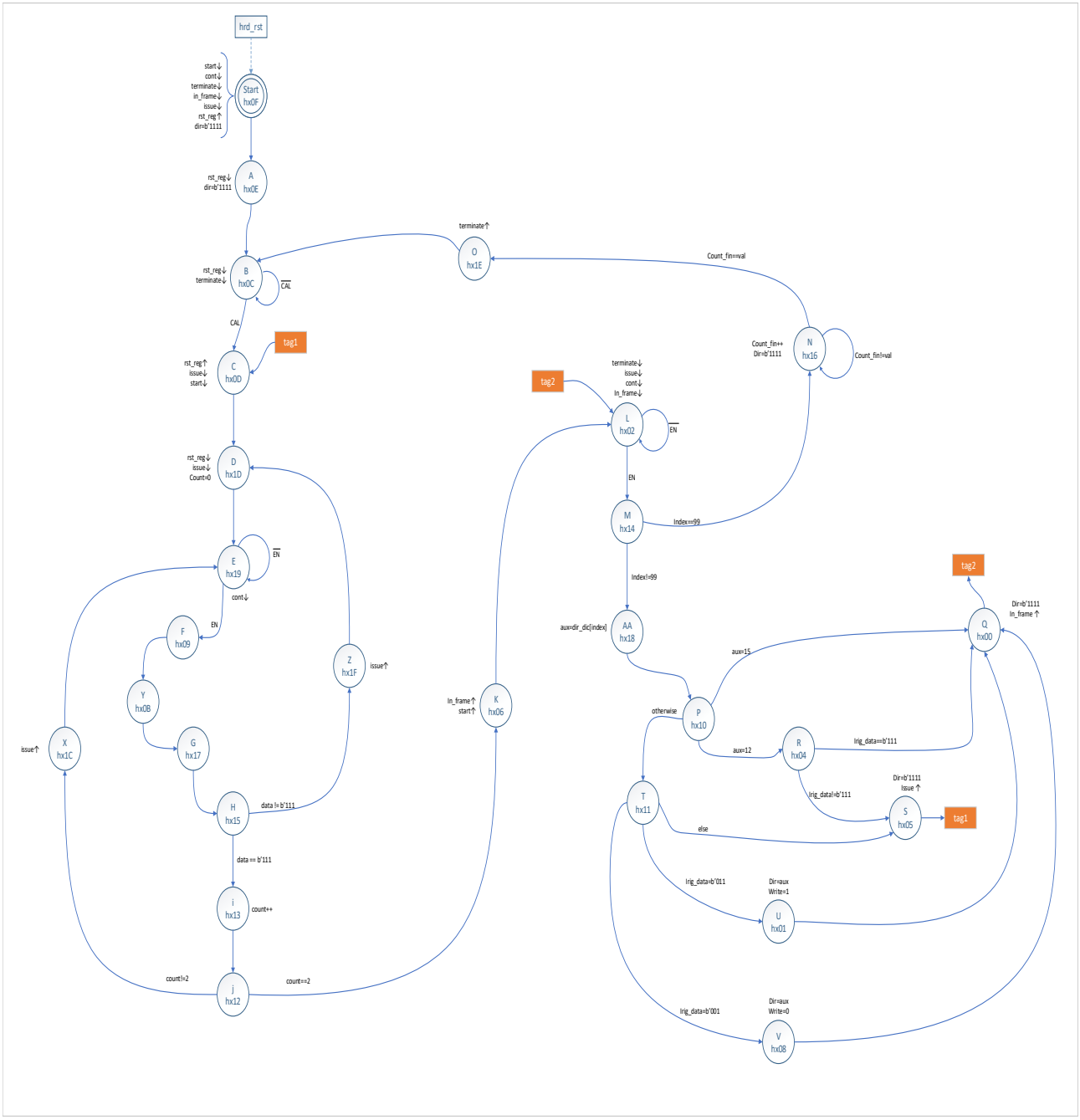


Figura A 7: Diagrama de estados de **IRIG_read**

HDL para máquina de estado finito parse_input

```
1  //
       ------------------------------------------------------------------------
2  //
3  // Title        : parse_inputs
4  // Design       : irig_read_input
5  // Author       : Seba
6  // Company      : Universidad de Chile
7  //
8  //
       ------------------------------------------------------------------------
9  //
10 // File         : c:\Users\seba\Documents\active-hdl\irig_read_input\
       irig_read_input\src\parse_inputs.v
11 // Generated    : Thu Feb  7 01:13:07 2019
12 // From         : interface description file
13 // By           : Itf2Vhdl ver. 1.22
14 //
15 //
       ------------------------------------------------------------------------
16 //
17 // Description :
18 //
19 //
       ------------------------------------------------------------------------
20 `timescale 1 ns / 1 ps
21
22 //{{ Section below this comment is automatically maintained
23 //   and may be overwritten
24 //{module {parse_inputs}}
25 module parse_inputs ( clk ,ce ,hard_rst ,rst ,gpio ,cal ,in_frame ,
       terminate ,cont ,en_ind ,rst_ind ,data_ready ,en_cbh ,rst_cbh,
       state_out, reg_wait);
26
27 output en_ind ;
28 wire en_ind ;
29 output rst_ind ;
30 wire rst_ind ;
31 output data_ready ;
32 wire data_ready ;
33 output en_cbh ;
34 wire en_cbh ;
35 output rst_cbh ;
36 wire rst_cbh ;
37 output [3:0] state_out;
38 wire  [3:0] state_out;
39
40 input clk ;
41 wire clk ;
42 input ce ;
43 wire ce ;
44 input hard_rst ;
```

```
wire hard_rst ;
input rst ;
wire rst ;
input gpio ;
wire gpio ;
input cal ;
wire cal ;
input in_frame ;
wire in_frame ;
input terminate ;
wire terminate ;
input cont ;
wire cont ;
input[31:0] reg_wait;
wire[31:0] reg_wait;
//}} End of automatically maintained section

// -- Enter your statements here -- //
reg[31:0] aux;
reg[31:0] counter = 0;
reg[3:0] next_state;
parameter start = 4'b1011;
parameter a = 4'b1010;
parameter b = 4'b0010;
parameter c = 4'b0000;
parameter d = 4'b1101;
parameter e = 4'b1111;
parameter f = 4'b1110;
parameter g = 4'b1000;
parameter h = 4'b0101;
parameter i = 4'b1100;
parameter j = 4'b0001;
parameter k = 4'b1001;
parameter l = 4'b0100;
parameter m = 4'b0110;
reg[3:0] actual_state = start;
reg en_ind_r=0, rst_ind_r=0, en_cbh_r=0, rst_cbh_r=0, data_ready_r=0;

always@(posedge clk, posedge hard_rst)begin
  if(hard_rst)
    actual_state <= start;
  else
    actual_state <= next_state;
end

always@(*)begin
  case(actual_state)
    start:
      next_state = a;
    a:
      next_state = b;
    b:
      if(~cal)    next_state = b;
      else if(cal)  next_state = c;
    c:
      next_state = h;
```

```
101     d:
102       if(~gpio)   next_state = d;
103       else if(gpio) next_state = e;
104     e:
105       if(gpio)    next_state = e;
106       else if(~gpio)  next_state = m;
107     f:
108       next_state = g;
109     g:
110       if(terminate) next_state = a;
111       else if(rst& ~terminate)  next_state = k;
112       else if(in_frame & ~rst & ~terminate )  next_state = i;
113       else if(cont & ~in_frame & ~rst & ~terminate) next_state = c;
114       else  next_state = g;
115     h:
116       next_state = d;
117     i:
118       next_state = j;
119     j:
120       next_state = c;
121     k:
122       next_state = l;
123     l:
124       next_state = c;
125       m:
126       begin
127         if(aux < counter) next_state = f;
128         else              next_state = c;
129       end
130   endcase
131 end
132
133 always@(posedge clk) begin
134   case(actual_state)
135     start:
136     begin
137       data_ready_r = 0;
138       en_ind_r = 0;
139       rst_ind_r = 1;
140       en_cbh_r = 0;
141       rst_cbh_r = 1;
142          aux = reg_wait;
143     end
144     a:
145       rst_ind_r = 1;
146     b:
147       rst_ind_r = 0;
148     c:
149       begin
150         rst_cbh_r = 1;
151         counter = 0;
152       end
153     e:
154       begin
155       en_cbh_r = 1;
156          counter = counter +1;
```

```
157       end
158     f:
159     begin
160       en_cbh_r = 0;
161       data_ready_r = 1;
162     end
163     g:
164       data_ready_r = 0;
165     h:
166       rst_cbh_r = 0;
167     i:
168       en_ind_r = 1;
169     j:
170       en_ind_r = 0;
171     k:
172       begin
173       en_ind_r = 0;
174       rst_ind_r = 1;
175       end
176     l:
177       rst_ind_r = 0;
178       m:
179       en_cbh_r = 0;
180   endcase
181 end
182 
183 assign data_ready = data_ready_r;
184 assign en_ind = en_ind_r;
185 assign rst_ind = rst_ind_r;
186 assign en_cbh = en_cbh_r;
187 assign rst_cbh = rst_cbh_r;
188 assign state_out = actual_state;
189 endmodule
```

Listing 1: Código Verilog para maquina de estado finito **parse_input**

HDL para máquina de estado finito IRIG_read

```
1  //
       ------------------------------------------------------------------------
2  //
3  // Title        : irig_read_2
4  // Design       : irig_read
5  // Author       : Seba
6  // Company      : Universidad de Chile
7  //
8  //
       ------------------------------------------------------------------------
9  //
10 // File         : c:\Users\seba\Documents\active-hdl\irig_read\irig_read\
       src\irig_read_2.v
11 // Generated    : Mon Feb 18 19:36:10 2019
12 // From         : interface description file
13 // By           : Itf2Vhdl ver. 1.22
14 //
15 //
       ------------------------------------------------------------------------
16 //
17 // Description :
18 //
19 //
       ------------------------------------------------------------------------
20 `timescale 1 ns / 1 ps
21
22 //{{ Section below this comment is automatically maintained
23 //   and may be overwritten
24 //{module {irig_read_2}}
25 module irig_read_2 ( clk ,ce ,state ,en ,irig_data ,cal ,hrd_rst ,ind ,
       write ,dir ,start ,rst_reg ,cont ,in_frame ,terminate ,issue, aux_out )
       ;
26
27 output [3:0] aux_out;
28 wire [3:0] aux_out;
29 output [4:0] state ;
30 wire [4:0] state ;
31 output write ;
32 wire write ;
33 output [3:0] dir ;
34 wire [3:0] dir ;
35 output start ;
36 wire start ;
37 output rst_reg ;
38 wire rst_reg ;
39 output cont ;
40 wire cont ;
41 output in_frame ;
42 wire in_frame ;
43 output terminate ;
44 wire terminate ;
```

```
output issue ;
wire issue ;

input clk ;
wire clk ;
input ce ;
wire ce ;
input en ;
wire en ;
input [2:0] irig_data ;
wire [2:0] irig_data ;
input cal ;
wire cal ;
input hrd_rst ;
wire hrd_rst ;
input [7:0] ind ;
wire [7:0] ind ;


reg[1:0] count;
reg[7:0] count_fin;
reg[3:0] aux;

reg[4:0] next_state;



parameter start_state = 5'b01111;
parameter a = 5'b01110;
parameter b = 5'b01100;
parameter c = 5'b01101;
parameter d = 5'b11101;
parameter e = 5'b11001;
parameter f = 5'b01001;
parameter g = 5'b10111;
parameter h = 5'b10101;
parameter i = 5'b10011;
parameter j = 5'b10010;
parameter k = 5'b00110;
parameter l = 5'b00010;
parameter m = 5'b10100;
parameter n = 5'b10110;
parameter o = 5'b11110;
parameter p = 5'b10000;
parameter q = 5'b00000;
parameter r = 5'b00100;
parameter s = 5'b00101;
parameter t = 5'b10001;
parameter u = 5'b00001;
parameter v = 5'b01000;
parameter x = 5'b11100;
parameter y = 5'b01011;
parameter z = 5'b11111;
parameter aa = 5'b11000;
parameter [7:0]val = 8'b00000011;
reg[3:0]dir_dic[0:100];
```

```
101
102 /* =  ' {4'b1111, 4'b0000, 4'b0000, 4'b0000, 4'b0000, 4'b1111, 4'b0001, 4'
    b0001, 4'b0001, 4'b1100,
103                                 4'b0010, 4'b0010, 4'b0010, 4'b0010, 4'
    b1111, 4'b0011, 4'b0011, 4'b0011, 4'b1111, 4'b1100,
104                                 4'b0100, 4'b0100, 4'b0100, 4'b0100, 4'
    b0111, 4'b0101, 4'b0101, 4'b1111, 4'b1111, 4'b1100,
105                                 4'b0110, 4'b0110, 4'b0110, 4'b0110, 4'
    b1111, 4'b0111, 4'b0111, 4'b0111, 4'b0111, 4'b1100,
106                                 4'b1000, 4'b1000, 4'b1111, 4'b1111, 4'
    b1111, 4'b1111, 4'b1111, 4'b1111, 4'b1111, 4'b1111};  //es el
    directorio con las direcciones , hay que definirlo aun...
107
108 */
109
110 reg terminate_, cont_, in_frame_, issue_, rst_reg_, start_, write_;
111 reg[3:0] dir_;
112
113 reg[4:0] actual_state=start_state;
114
115
116 initial begin
117     dir_dic[0] = 4'b1111; dir_dic[1] = 4'b0000; dir_dic[2]=4'b0000;
    dir_dic[3]=4'b0000; dir_dic[4]=4'b0000; dir_dic[5]=4'b1111; dir_dic
    [6]=4'b0001; dir_dic[7]=4'b0001; dir_dic[8] = 4'b0001; dir_dic[9] = 4'
    b1100;
118     dir_dic[10] = 4'b0010; dir_dic[11] = 4'b0010; dir_dic[12] = 4'b0010;
    dir_dic[13] = 4'b0010; dir_dic[14]=4'b1111;dir_dic[15] = 4'b0011;
    dir_dic[16] = 4'b0011; dir_dic[17] = 4'b0011; dir_dic[18] = 4'b1111;
    dir_dic[19] = 4'b1100;
119     dir_dic[20] = 4'b0100; dir_dic[21] = 4'b0100; dir_dic[22] = 4'b0100;
    dir_dic[23] = 4'b0100; dir_dic[24] = 4'b1111; dir_dic[25] = 4'b0101;
    dir_dic[26] = 4'b0101; dir_dic[27] = 4'b1111; dir_dic[28] = 4'b1111;
    dir_dic[29] = 4'b1100;
120     dir_dic[30] = 4'b0110; dir_dic[31] = 4'b0110; dir_dic[32] = 4'b0110;
    dir_dic[33]=4'b0110; dir_dic[34] = 4'b1111; dir_dic[35] = 4'b0111;
    dir_dic[36] = 4'b0111; dir_dic[37] = 4'b0111; dir_dic[38] = 4'b0111;
    dir_dic[39] = 4'b1100;
121     dir_dic[40] = 4'b1000; dir_dic[41] = 4'b1000; dir_dic[42] = 4'b1111;
    dir_dic[43]=4'b1111; dir_dic[44] = 4'b1111; dir_dic[45] = 4'b1111;
    dir_dic[46] = 4'b1111; dir_dic[47] = 4'b1111; dir_dic[48] = 4'b1111;
    dir_dic[49] = 4'b1100;
122 dir_dic[50] = 4'b1111; dir_dic[51] = 4'b1111; dir_dic[52] = 4'b1111;
    dir_dic[52] = 4'b1111; dir_dic[53] = 4'b1111; dir_dic[54] = 4'b1111;
    dir_dic[55] = 4'b1111; dir_dic[56] = 4'b1111; dir_dic[57] = 4'b1111;
    dir_dic[58] = 4'b1111; dir_dic[59] = 4'b1111;
123 dir_dic[60] = 4'b1111; dir_dic[61] = 4'b1111; dir_dic[62] = 4'b1111;
    dir_dic[63] = 4'b1111; dir_dic[64] = 4'b1111; dir_dic[65] = 4'b1111;
    dir_dic[66] = 4'b1111; dir_dic[67] = 4'b1111; dir_dic[68] = 4'b1111;
    dir_dic[69] = 4'b1111;
124 dir_dic[70] = 4'b1111; dir_dic[71] = 4'b1111; dir_dic[72] = 4'b1111;
    dir_dic[73] = 4'b1111; dir_dic[74] = 4'b1111; dir_dic[75] = 4'b1111;
    dir_dic[76] = 4'b1111; dir_dic[77] = 4'b1111; dir_dic[78] = 4'b1111;
    dir_dic[79] = 4'b1111;
125 dir_dic[80] = 4'b1111; dir_dic[81] = 4'b1111; dir_dic[82] = 4'b1111;
    dir_dic[83] = 4'b1111; dir_dic[84] = 4'b1111; dir_dic[85] = 4'b1111;
```

```
   dir_dic[86] = 4'b1111; dir_dic[87] = 4'b1111; dir_dic[88] = 4'b1111;
   dir_dic[89] = 4'b1111;
dir_dic[90] = 4'b1111; dir_dic[91] = 4'b1111; dir_dic[92] = 4'b1111;
   dir_dic[93] = 4'b1111; dir_dic[94] = 4'b1111; dir_dic[95] = 4'b1111;
   dir_dic[96] = 4'b1111; dir_dic[97] = 4'b1111; dir_dic[98] = 4'b1111;
   dir_dic[99] = 4'b1111;
dir_dic[100] = 4'b1111;

rst_reg_ = 0;
cont_ = 0;
dir_ = 4'b1111;
start_ = 0;
write_ = 0;
in_frame_ = 0;
terminate_ =0;
issue_ = 0;
aux = 4'b1111;
end

always@(posedge clk, posedge hrd_rst)begin
  if(hrd_rst)
    actual_state <= start_state;
  else
    actual_state <= next_state;
end

always@(*)begin
  case(actual_state)
      start_state:
        next_state = a;
      a:
        next_state = b;
      b:
        if(~cal)    next_state = b;
        else if(cal)  next_state = c;
      c:
        next_state = d;
      d:
        next_state = e;
      e:
        if(~en)     next_state = e;
        else if(en)   next_state = f;
      f:
        next_state = y;
      g:
      next_state = h;
    h:
      if(irig_data == 3'b111) next_state = i;
      else      next_state = z;
    i:
      next_state = j;
    j:
      if(count == 2)    next_state = k;
      else        next_state = x;
    k:
      next_state = l;
```

```
177     l:
178       if(~en)      next_state = l;
179       else if(en)   next_state = m;
180     m:
181       if(ind==100)       next_state = n;
182       else         next_state = aa;
183     n:
184       if(count_fin==val)  next_state = o;
185       else         next_state = n;
186     o:
187       next_state = b;
188     p:
189       if(aux==4'b1100)  next_state = r;
190       else if(aux==4'b1111) next_state = q;
191       else if(aux<9)    next_state = t;
192     q:
193       next_state = l;
194     r:
195       if(irig_data == 3'b111)   next_state = q;
196       else            next_state = s;
197     s:
198       next_state = c;
199     t:
200       if(irig_data == 3'b011)     next_state = u;
201       else if(irig_data == 3'b001)  next_state = v;
202       else              next_state = s;
203     u:
204       next_state = q;
205     v:
206       next_state = q;
207     x:
208       next_state = e;
209     y:
210       next_state = g;
211                 z:
212                         next_state = d;
213                 aa:
214                         next_state = p;
215
216     default:
217       next_state = c;
218   endcase
219 end
220
221 always@(posedge clk) begin
222   case(actual_state)
223     start_state:
224       begin
225         start_ = 0;
226         terminate_ = 0;
227         cont_ = 0;
228         in_frame_ = 0;
229         issue_ = 0;
230         rst_reg_ = 1;
231         dir_ = 4'b1111;
232         write_ = 0;
```

```
233       end
234     a:
235         begin
236         dir_ = 4'b1111;
237         rst_reg_ = 0;
238       end
239     b:
240       begin
241         terminate_ = 0;
242         rst_reg_ = 0;
243       end
244     c:
245       begin
246         start_ = 0;
247         rst_reg_ = 1;
248         issue_ = 0;
249       end
250     d:
251       begin
252         count = 0;
253         rst_reg_ = 0;
254                                   issue_ = 0;
255       end
256     e:
257                           begin
258                               rst_reg_ = 0;
259           cont_ = 0;
260                           end
261                 f:
262                           begin
263                               rst_reg_ = 0;
264                           end
265     i:
266       count = count + 1;
267     k:
268       begin
269         in_frame_ = 1;
270         start_ = 1;
271       end
272     l:
273       begin
274         in_frame_ = 0;
275         terminate_ = 0;
276         issue_ = 0;
277         cont_ = 0;
278         count_fin = 0;
279       end
280     n:
281       begin
282       dir_ = 4'b1111;
283       count_fin =count_fin + 1;
284       end
285     o:
286       terminate_ = 1;
287     aa:
288       aux = dir_dic[ind];//aux = dir_dic[4'd1];
```

```
289     q:
290       begin
291         dir_ = 4'b1111;
292         in_frame_ = 1;
293       end
294     s:
295       begin
296         dir_ = 4'b1111;
297         issue_ = 1;
298       end
299     u:
300       begin
301         dir_ = aux;
302         write_ = 1;
303       end
304     v:
305       begin
306         dir_ = aux;
307         write_ = 0;
308       end
309     x:
310       cont_ = 1;
311                 z:
312                         issue_ = 1;
313   endcase
314 end
315 
316 assign start = start_;
317 assign terminate = terminate_;
318 assign dir = dir_;
319 assign write = write_;
320 assign rst_reg = rst_reg_;
321 assign cont = cont_;
322 assign in_frame = in_frame_;
323 assign state = actual_state;
324 assign issue = issue_;
325 assign aux_out = aux;
326 
327 endmodule
```

Listing 2: Código Verilog para maquina de estado finito **IRIG_read**

HDL para máquina de estado finito fsm_fac_sec

```
1  //
       ------------------------------------------------------------------------------
2  //
3  // Title        : parse_inputs
4  // Design       : irig_read_input
5  // Author       : Seba
6  // Company      : Universidad de Chile
7  //
8  //
       ------------------------------------------------------------------------------
9  //
10 // File         : c:\Users\seba\Documents\active-hdl\irig_read_input\
       irig_read_input\src\parse_inputs.v
11 // Generated    : Thu Feb  7 01:13:07 2019
12 // From         : interface description file
13 // By           : Itf2Vhdl ver. 1.22
14 //
15 //
       ------------------------------------------------------------------------------
16 //
17 // Description :
18 //
19 //
       ------------------------------------------------------------------------------
20 `timescale 1 ns / 1 ps
21
22 //{{ Section below this comment is automatically maintained
23 //   and may be overwritten
24 //{module {parse_inputs}}
25 module fsm_frac_sec (clk ,ce ,hard_rst, gpio,new_count, waiting, threshold
       , state );
26
27
28 output new_count;
29 wire new_count;
30 output[2:0] state;
31 wire [2:0]  state;
32
33 input clk;
34 wire clk;
35 input ce;
36 wire ce;
37 input hard_rst;
38 wire hard_rst;
39 input gpio;
40 wire gpio;
41 input [31:0] threshold;
42 wire [31:0] threshold;
43 input [31:0] waiting;
44 wire [31:0]  waiting;
45
```

```
reg[31:0] aux;
reg[31:0] aux_wait;
reg[31:0] counter=0;
reg[2:0] actual_state;
reg[2:0] next_state;

reg new_count_r = 0;

parameter start = 3'b000;
parameter a = 3'b100;
parameter b = 3'b101;
parameter c = 3'b001;
parameter d = 3'b011;
parameter e = 3'b111;
parameter f = 3'b110;


// -- Enter your statements here -- //

always@(posedge clk, posedge hard_rst)begin
  if(hard_rst)
    actual_state <= start;
  else
    actual_state <= next_state;
end


always@(*)begin
  case(actual_state)
    start:
      next_state = a;
    a:
      if(gpio)        next_state = b;
      else            next_state = a;
    b:
            begin
        if(counter >aux)                next_state = c;
        else if (~gpio)      next_state = start;
        else                           next_state = b;
            end
          c:
      next_state = d;
    d:
      if(counter>aux_wait)      next_state = e;
      else          next_state = d;
    e:
      if(~gpio)       next_state = f;
      else          next_state = e;
    f:
      next_state = a;
  endcase
end


always@(posedge clk) begin
  case(actual_state)
```

```
102     start:
103     begin
104         aux = threshold;
105         new_count_r = 0;
106         aux_wait = waiting;
107     end
108     b:
109       counter = counter+1;
110     c:
111       begin
112         new_count_r = 1;
113         counter = 0;
114       end
115         d:
116       begin
117         counter = counter+1;
118         new_count_r = 0;
119       end
120     f:
121       counter = 0;
122   endcase
123 end
124 assign new_count = new_count_r;
125 assign state = actual_state;
126 endmodule
```

Listing 3: Código Verilog para maquina de estado finito **fsm_fac_sec**

Códigos de mediciones

```
from generator import *
import numpy as np
import matplotlib.pyplot as plt
import time
import struct
import ipdb


def test_meas(signal_ip, ref_ip, ind_ref=0 ):
    cal_signal =    {'type'         :       'visa',
                         'connection'   :       'TCPIP::192.168.1.34::
   INSTR',
                         'def_freq'     :       50,
                         'def_power'    :       -3
        }

    cal_ref =    {'type'         :       'visa',
                    'connection'   :       'TCPIP::192.168.1.34::INSTR',
                     'def_freq'     :       50,
                     'def_power'    :       -3
    }

    signal_pow = np.linspace(-3,-83,801)
    ref_pow = -3
    aux = 'TCPIP::'+signal_ip+'::INSTR'
    cal_signal['connection'] = aux
    source_signal = create_generator(cal_signal)
    aux = 'TCPIP::'+ref_ip+'::INSTR'
    cal_ref['connection'] = aux
    source_ref = create_generator(cal_ref)
    i = 0
    source_ref.set_power_dbm(ref_pow)
    source_ref.turn_output_on()
    for i in range(len(signal_pow)):
        source_signal.set_power_dbm(signal_pow[i])
        source_signal.turn_output_on()
        time.sleep(1)

    print('fin')



def charact_meas(fpga, signal_ip, ref_ip, filename, freq, ind_ref=0):
    """funcion para caracterizar rango dinamico sin ruido!
    """
    cal_signal =    {'type'         :       'visa',
                         'connection'   :       'TCPIP::192.168.1.34::
   INSTR',
                         'def_freq'     :       50,
                         'def_power'    :       -3
        }

    cal_ref =    {'type'         :       'visa',
                    'connection'   :       'TCPIP::192.168.1.34::INSTR',
                     'def_freq'     :       50,
```

```
                    'def_power'     :       -3
    }
    signal_pow = np.linspace(-5,-90,851)
    ref_pow = -6
    aux = 'TCPIP::'+signal_ip+'::INSTR'
    cal_signal['connection'] = aux
    cal_signal['def_freq'] = freq
    source_signal = create_generator(cal_signal)
    aux = 'TCPIP::'+ref_ip+'::INSTR'
    cal_ref['connection'] = aux
    cal_ref['def_freq'] = freq
    source_ref = create_generator(cal_ref)
    source_ref.set_power_dbm(ref_pow)
    source_ref.turn_output_on()
    aux = np.zeros([len(signal_pow),4,1024])
    i = ind_ref
    f_a = file(filename+'_A', 'a')
    f_b = file(filename+'_B', 'a')
    f_phase = file(filename+'_ang', 'a')
    flag = 0
    start = time.time()
    while(i<len(signal_pow)):
        if(not flag):
            print(signal_pow[i])
            source_signal.set_power_dbm(signal_pow[i])
            source_signal.turn_output_on()
            fpga.write_int('reading_data',1)
            fpga.write_int('reading_data',0)
            while(not fpga.read_int('full_mem')):
                pass
        try:
            raw_A = fpga.read('powA', 1024*8,0)
            raw_B = fpga.read('powB', 1024*8,0)
            raw_phase = fpga.read('phase', 2048*8,0)
            f_a.write(raw_A)
            f_b.write(raw_B)
            f_phase.write(raw_phase)
            powB = struct.unpack('>1024Q', raw_B)
            powA = struct.unpack('>1024Q', raw_A)
            phase = struct.unpack('>2048q', raw_phase)
            re = phase[::2]
            im = phase[1::2]
            aux[i,0,:] = powA
            aux[i,1,:] = powB
            aux[i,2,:] = re
            aux[i,3,:] = im
            i = i+1
            flag = 0
        except:
            flag = 1
    f_a.close()
    f_b.close()
    f_phase.close()
    end = time.time()-start
    print('it took + %f secs'%end)
    return [signal_pow, aux]
```

```
110
111
112
113 def noise_meas(fpga, signal_ip, ref_ip, ref_val, filename, ind_ref=0):
114     cal_signal =    {'type'         :       'visa',
115                      'connection'   :       'TCPIP::192.168.1.34::
    INSTR',
116                      'def_freq'     :       50,
117                      'def_power'    :       -3
118         }
119
120     cal_ref =    {'type'         :       'visa',
121                   'connection'   :       'TCPIP::192.168.1.34::INSTR',
122                    'def_freq'     :       50,
123                    'def_power'    :       -3
124     }
125     #ipdb.set_trace()
126     signal_pow = np.linspace(-3,-88,86)
127     ref_pow = ref_val
128     aux = 'TCPIP::'+signal_ip+'::INSTR'
129     cal_signal['connection'] = aux
130     source_signal = create_generator(cal_signal)
131     aux = 'TCPIP::'+ref_ip+'::INSTR'
132     cal_ref['connection'] = aux
133     source_ref = create_generator(cal_ref)
134     source_ref.set_power_dbm(ref_pow)
135     source_ref.turn_output_on()
136     aux = np.zeros([len(signal_pow),6,1024])
137     i = ind_ref
138     f_a = file(filename+'_A', 'a')
139     f_b = file(filename+'_B', 'a')
140     f_phase = file(filename+'_ang', 'a')
141     f_time = file(filename+'_time', 'a')
142     flag = 0
143     start = time.time()
144     while(i<len(signal_pow)):
145         if(not flag):
146             print(signal_pow[i])
147             source_signal.set_power_dbm(signal_pow[i])
148             source_signal.turn_output_on()
149             fpga.write_int('reading_data',1)
150             fpga.write_int('reading_data',0)
151             while(not fpga.read_int('full_mem')):
152                 pass
153         try:
154             print('trying')
155             raw_A = fpga.read('powA', 1024*8,0)
156             raw_B = fpga.read('powB', 1024*8,0)
157             raw_phase = fpga.read('phase', 2048*8,0)
158             raw_time = fpga.read('time', 1024*8)
159             f_a.write(raw_A)
160             f_b.write(raw_B)
161             f_phase.write(raw_phase)
162             f_time.write(raw_time)
163             powB = struct.unpack('>1024Q', raw_B)
164             powA = struct.unpack('>1024Q', raw_A)
```

```
            phase = struct.unpack('>2048q', raw_phase)
            re = phase[::2]
            im = phase[1::2]
            time_ = struct.unpack('>2048I', raw_time)
            sec = time_[::2]
            fracs = time_[1::2]
            aux[i,0,:] = powA
            aux[i,1,:] = powB
            aux[i,2,:] = re
            aux[i,3,:] = im
            aux[i,4,:] = sec
            aux[i,5,:] = fracs
            i = i+1
            flag = 0
        except:
            flag = 1
    f_a.close()
    f_b.close()
    f_phase.close()
    f_time.close()
    end = time.time()-start
    print('it took + %f secs'%end)
    return [signal_pow, aux]



def test_meas(fpga, signal_ip, ref_ip, ref_val, filename, ind_ref=0):
    cal_signal =    {'type'          :       'visa',
                         'connection'    :       'TCPIP::192.168.1.34::
    INSTR',
                         'def_freq'      :       50,
                         'def_power'     :       -3
        }

    cal_ref =    {'type'          :       'visa',
                     'connection'    :       'TCPIP::192.168.1.34::INSTR',
                      'def_freq'      :       50,
                      'def_power'     :       -3
    }
    #ipdb.set_trace()
    signal_pow = np.linspace(-5,-90,86)
    ref_pow = ref_val
    aux = 'TCPIP::'+signal_ip+'::INSTR'
    cal_signal['connection'] = aux
    source_signal = create_generator(cal_signal)
    aux = 'TCPIP::'+ref_ip+'::INSTR'
    cal_ref['connection'] = aux
    source_ref = create_generator(cal_ref)
    source_ref.set_power_dbm(ref_pow)
    source_ref.turn_output_on()
    aux = np.zeros([len(signal_pow),6,1024])
    i = ind_ref
    f_a = file(filename+'_A', 'a')
    f_b = file(filename+'_B', 'a')
    f_phase = file(filename+'_ang', 'a')
```

```
220     f_time = file(filename+'_time', 'a')
221     flag = 0
222     start = time.time()
223     while(i<len(signal_pow)):
224         if(not flag):
225             print(signal_pow[i])
226             source_signal.set_power_dbm(signal_pow[i])
227             source_signal.turn_output_on()
228             fpga.write_int('reading_data',1)
229             fpga.write_int('reading_data',0)
230             while(not fpga.read_int('full_mem')):
231                 pass
232         try:
233             print('trying')
234             raw_A = fpga.read('powA', 1024*8,0)
235             raw_B = fpga.read('powB', 1024*8,0)
236             raw_phase = fpga.read('phase', 2048*8,0)
237             raw_time = fpga.read('time', 1024*8)
238             f_a.write(raw_A)
239             f_b.write(raw_B)
240             f_phase.write(raw_phase)
241             f_time.write(raw_time)
242             powB = struct.unpack('>1024Q', raw_B)
243             powA = struct.unpack('>1024Q', raw_A)
244             phase = struct.unpack('>2048q', raw_phase)
245             re = phase[::2]
246             im = phase[1::2]
247             time_ = struct.unpack('>2048I', raw_time)
248             sec = time_[::2]
249             fracs = time_[1::2]
250             aux[i,0,:] = powA
251             aux[i,1,:] = powB
252             aux[i,2,:] = re
253             aux[i,3,:] = im
254             aux[i,4,:] = sec
255             aux[i,5,:] = fracs
256             i = i+1
257             flag = 0
258         except:
259             flag = 1
260     f_a.close()
261     f_b.close()
262     f_phase.close()
263     f_time.close()
264     end = time.time()-start
265     print('it took + %f secs'%end)
266     return [signal_pow, aux]
```

Listing 4: Código en Python para control de generadores y adquisición de datos de ROACH2

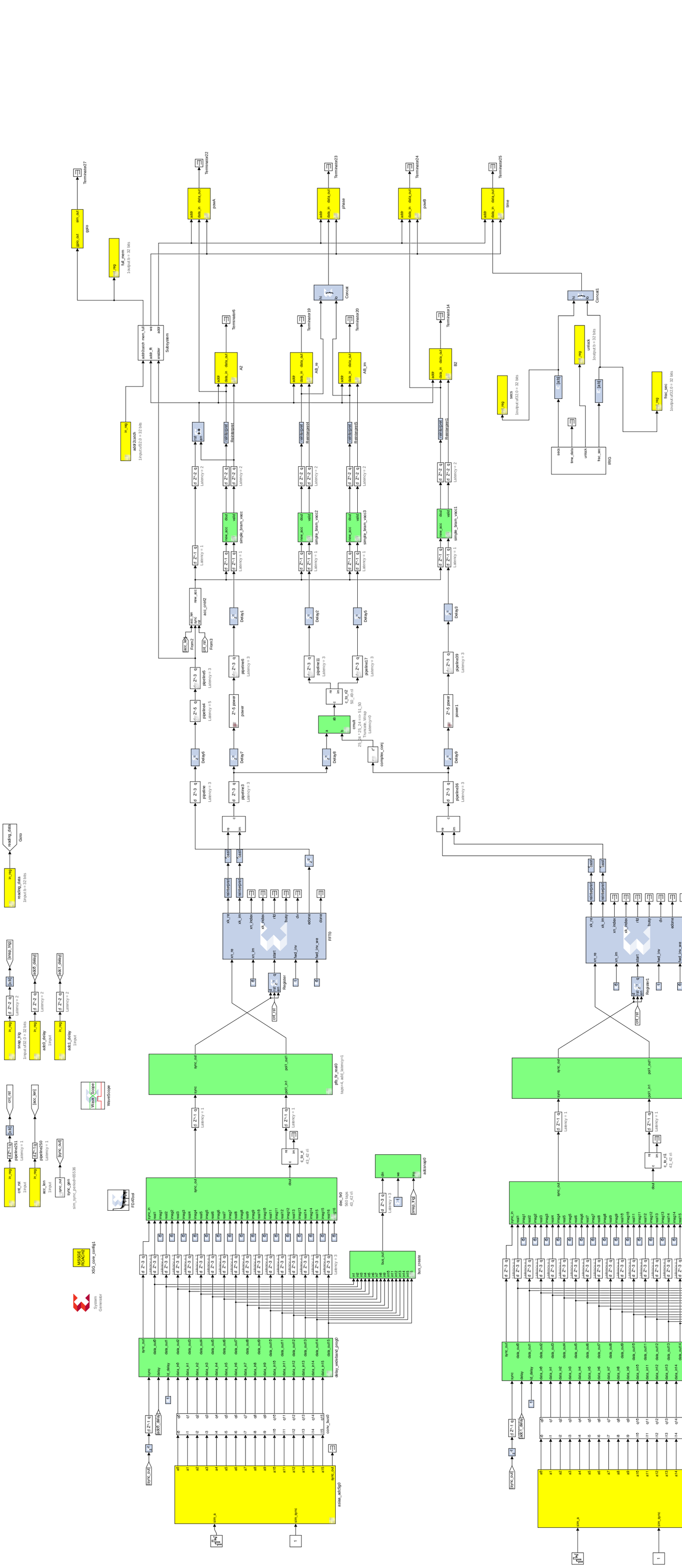


Figura A 8: Implementación de voltímetro vectorial en Simulink.

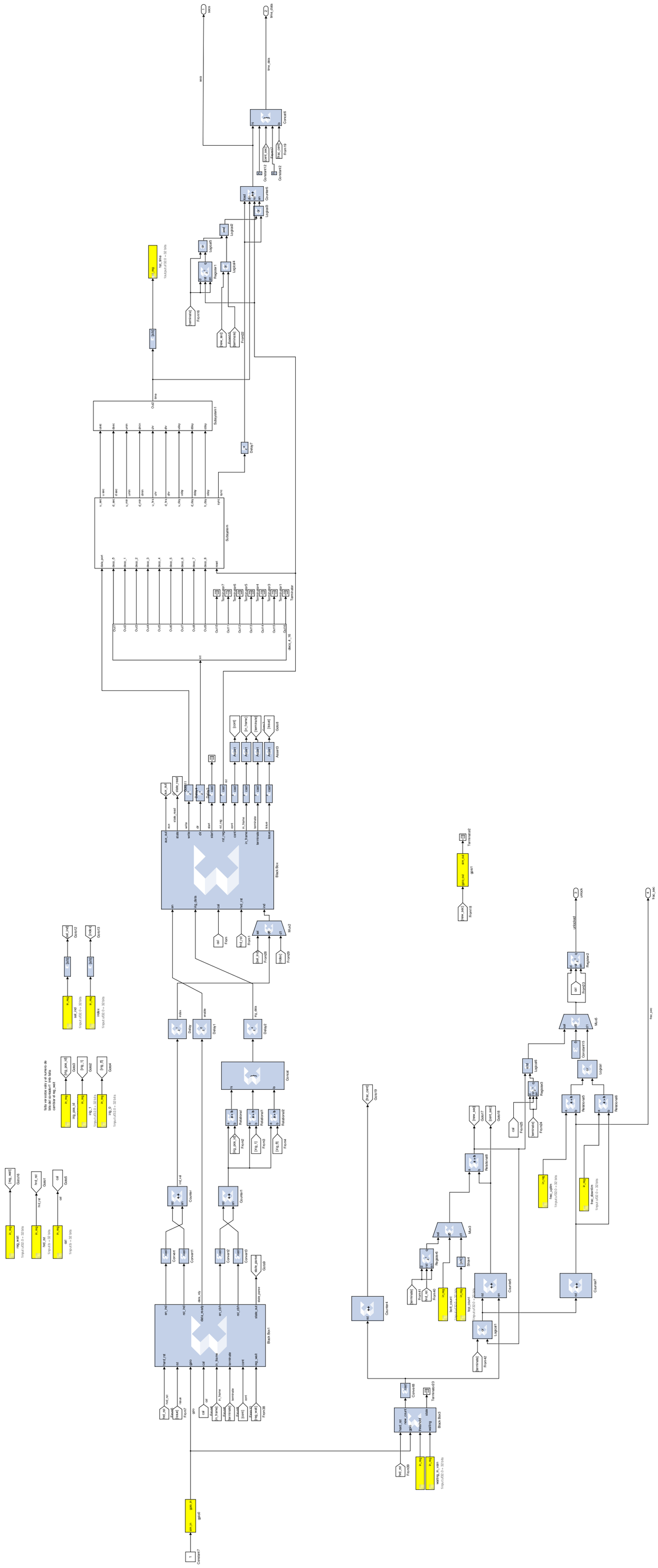


Figura A 9: Implementación de marcado de tiempo en Simulink.