Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Unidad de linealización y normalización para un estimador de parámetros de uso en un sistema de optimización de energía en paneles fotovoltaicos

Informe de Proyecto de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Adrián Ignacio Cervantes Segura

Cartago, 17 de junio de 2016

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación Unidad de linealización y normalización para un estimador de parámetros de uso en un sistema de optimización de energía en paneles fotovoltaicos, realizado por el señor Adrián Ignacio Cervantes Segura y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Ing. Leonardo Rivas Arce

Profesor lector

Ing. Leonardo Sandoval Cascante

Profesor lector

Ing. Alfonso Chacón Rodríguez

Profesor asesor

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Adrián Ignacio Cervantes Segura

Cartago, 17 de junio de 2016

Céd: 1-1508-0317

Resumen

En esta tesis se presenta el diseño de un linealizador de corriente descrito en lenguaje Verilog, basado en el estándar para aritmética de coma flotante de IEEE 754 . Para este proyecto se usa el formato de precisión simple de dicho estandar, de 32 bits para implantar una función logaritmo natural por medio del algoritmo CORDIC. Para propósitos de interfaz del linealizador a desarrollar, se añaden además una unidad de normalización de datos y otra de conversión de coma fija a coma flotante. Estos sistemas serán usadnos en un sistema para optimizar la eficiencia de un panel fotovoltaico con una relación I-V no lineal. Este linealizador ha sido diseñado considerando restricciones de área, velocidad de procesamiento y consumo de potencia, con miras a integrarse dentro de un sistema fotovoltaico energéticamente eficiente. Las pruebas de los circuitos diseñados se realizaron sobre una placa de desarrollo Digilent.

Palabras clave: Aritmética Binaria, Formato IEEE 754, coma fija, FPGA, Verilog.

Abstract

Here, the design and verification of an electrical current linealizing module written in Verilog, based on the IEEE 754 floating point standard, and using a CORDIC algorithm to perform a natural logarithm operation is presented. The system is to be used to optimize the efficiency of photovoltaic panels with a non-linear I-V relation. The design is tested on a Digilint Nexys-4 FPGA board.

Keywords: Binary arithmetic, IEEE 754 Format, Fixed point, FPGA, Verilog.

a mis queridos padres

Agradecimientos

En primer lugar a Dios, quién me ha dado sabiduría y ha sido un pilar importante a lo largo de esta carrera.

Al Ing. Alfonso Chacón Rodríguez, quién ha sido un gran apoyo en el desarrollo del proyecto y en cursos a lo largo de la carrera, por los consejos, paciencia, conocimiento compartido y su nivel de profesionalismo.

A mi familia, por apoyarme a lo largo de toda mi educación, por la ayuda y por los grandes sacrificios que ha hecho para poder llegar donde estoy hoy, por los valores y por toda la ayuda que me han dado en todos estos años.

Finalmente, agradezco a las personas que de una u otra forma me han apoyado a lo largo del proceso.

Adrián Ignacio Cervantes Segura

Cartago, 17 de junio de 2016

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Entorno del proyecto

Hoy en día es cada vez más común el tema de las energías limpias, tales como la eólica y la solar, dados fenómenos como la creciente necesidad energética y el aumento en el la contaminación producto de las fuentes tradicionales. La instalación de fuentes de suministro con paneles fotovoltaicos ha llegado a ser una tendencia en Costa Rica, estas fuentes de suministro son sistemas de autoconsumo de energía, y a su vez son utilizados para comercializar la energía a otras empresas.

Un sistema de abastecimiento de energía solar requiere de paneles solares, acumuladores de energía, inversores (conversión de corriente continua en corriente alterna) y reguladores; sin embargo actualmente las redes de suministros no cuentan con un sistema regulador de tensión, que se encargue de ubicar el punto de operación de potencia máxima. Esto es un proceso complejo, debido a la variación no lineal de la corriente y la tensión del panel con respecto a la temperatura e irradiancia del medio en el que se encuentra. El objetivo principal de este proyecto se basa en ofrecer los datos necesarios de un panel fotovoltaico para que un algoritmo de optimización pueda buscar el punto de tensión donde se obtenga la máxima potencia.

El desarrollo de este proyecto se basa en aumentar la eficiencia del sistema completo para un panel previamente escogido: se utilizará un panel modelo KS10T de la empresa KYOCERA SOLAR.

El proyecto linealizador-normalizador se realizó en el Instituto Tecnológico de Costa Rica, Escuela de Ingeniería Electrónica, bajo la asesoría del coordinador del SESLab Dr. Carlos Meza, y el coordinador del DCILab Dr. Alfonso Chacón. Ambos laboratorios se encargan de presentar propuestas de sistemas electrónicos que sean útiles para el desarrollo tecnológico y sostenibilidad, de manera que los recursos sean aprovechados de la mejor forma, brindado soluciones en el área de energías limpias.

1.2 Descripción del problema y justificación

La curva característica $i_{pv} - V_{pv}$ de una celda solar no tiene un comportamiento lineal, de manera que si se requiere estimar parámetros a partir de la corriente y tensión de estos utilizando algún estimador lineal, se deben linealizar-normalizar las entradas al algorimo estimador. Estos resultados, después deberán de nuevo desnormalizarce y deslinealizarse para aplicarlos a algún algoritmo de optimización. Para el modelo del panel se tienen cuatro tipos de configuraciones que consideran las pérdidas resistivas de las celdas, paralelas y serie.

Estas operaciones implican ya la necesidad de una unidad de procesamiento de coma flotante. En el DCILab ya se posee alguna experiencia en desarrollo de estas unidades (ver [15][16]). Es por ello que se pudo ofrecer ya una posibilidad de solución a las necesidades planteadas por el SESLab para este proyecto. Estudiando el problema en particular, se definieron los siguientes requisitos para el módulo numérico a realizar:

- Se debe basar en el formato IEEE 754.
- Utilizar una arquitectura de 32 bits.
- Utilizar Verilog como lenguaje de descripción de hardware.
- Optimizar las unidades para requerir la menor cantidad de recursos.

1.3 Síntesis del problema

¿Cómo realizar una linealización y normalización de las relaciones I-V de un panel fotovoltaico en formato IEEE 754, con miras a la optimización de su punto de operación?

1.4 Enfoque de la solución

En este proyecto se siguió un enfoque basado en el diseño en ingeniería y el desarrollo modular (top-down) de sistemas digitales. Se partió de una especificación abstracta hasta llegar a la síntesis física de la solución sobre un dispositivo programable. Para ello fue necesario el uso de distintas herramientas de software EDA que apoyaron el diseño y verificación de las distintas etapas. Este software incluyó el paquete integrado de desarrollo sobre FPGAs Vivado de Xilinx, y el software de cálculo matemático Python y el desarrollo de algunos modelos teóricos para comprender el problema técnico a resolver.

1.5 Meta

La meta de este proyecto es la de aprovechar de forma óptima la radiación solar que incide sobre un panel fotovoltaico, aumentando la eficiencia e impulsar la utilización de energías

limpias.

1.6 Objetivos y estructura

1.6.1 Objetivo general

Desarrollar una unidad de linealización y normalización para un estimador de parámetros Corriente-Tensión de un panel fotovoltaico.

1.6.2 Objetivos específicos

- Construir un módulo de procesamiento aritmético en formato IEEE 754, que linealice la corriente del modelo de un panel fotovoltaico, por medio de una operación logarítmica, con una corriente exponencial como parámetro de entrada y una corriente lineal 'y' en la salida.
  Indicador: La precisión del algoritmo implementado en hardware tiene un error menor al 5% contra un modelo de referencia de alto nivel.
- Construir un circuito que convierta de coma flotante a coma fija, la corriente lineal 'y' en la salida del linealizador.
  Indicador:La precisión del algoritmo implementado en hardware tiene un error menor al 5% contra un modelo de referencia de alto nivel.
- Construir un circuito que normalice los parámetros de corriente lineal 'y' y tensión lineal 'z', ambos en coma fija, para las entradas del estimador.
  Indicador: La precisión del algoritmo implementado en hardware tiene un error menor al 5% contra un modelo de referencia de alto nivel.

1.6.3 Estructura

El capítulo 2 se describen los conceptos teóricos que serán utilizados a través del desarrollo del proyecto. En el capitulo 3 se detalla el diseño del algoritmo de cálculo y control, implementación y los resultados obtenidos para el circuito de linealización. En el capítulo 4 se presentan el diseño del algoritmo y control, implementación y los resultados obtenidos para el circuito de normalización. El capitulo 5 muestra el sistema completo, junto con sus pruebas y resultados. El capítulo 6 se ofrecen conclusiones del proyecto, y recomendaciones. Finalmente, en el capitulo 7 se puede observar la bibliografía utilizada.

Capítulo 2

Procedimiento metodológico

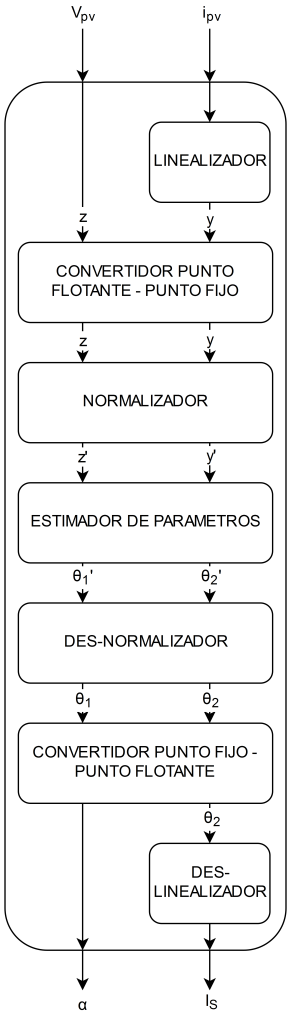

Figura 2.1: Diagrama de solución para el sistema completo para aumentar la eficiencia de los paneles fotovoltaicos por medio de un linealizador, estimador de parámetros, deslinealizador.

Primeramente se realizó un proceso de recopilación de información dentro de temas como: curvas y modelo de un PV, ecuaciones características de los PV, algoritmo de CORDIC, estándar IEEE 754 y operaciones en coma fija. Posteriormente se utilizó este estándar para iniciar el diseño del linealizador y el normalizador del sistema general del panel fotovoltaico, este se puede observar en la figura 2.1.

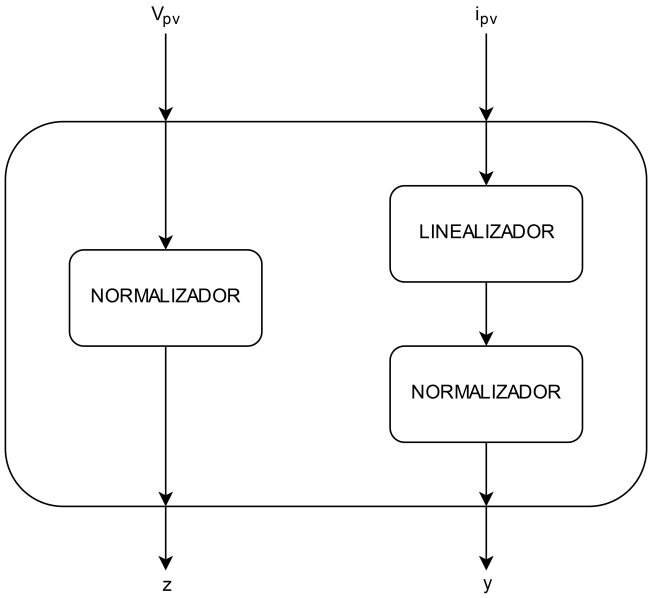

Figura 2.2: Diagrama de solución para el sistema de linealización y normalización, con entradas de corriente y tensión del panel fotovoltaico y salidas de corriente y tensión linealizadas y normalizadas.

Para el diseño del los circuitos linealización-normalización de la figura 2.2 se utilizó el diseño modular, ya que se brinda una mejor perspectiva de las etapas del diseño que se debían realizar, primeramente se asignaron las entradas y salidas que requería cada unidad. Una vez ejecutada la primera etapa de diseño se dividió en pequeños bloques funcionales.

Este diseño modular, se implementó utilizando el lenguaje Verilog, y se verificaron los resultados comparando el modelo en alto nivel realizado en Python, contra los resultados obtenidos en las simulaciones Post Place & Route.

Por último, se realizaron las pruebas en una FPGA artix-7 para comprobar el funcionamiento del hardware y sus respectivas comparaciones con las simulaciones.

Capítulo 3

Marco teórico

3.1 Descripción

Actualmente en los sistemas de paneles fotovoltaicos, es de suma importancia la eficiencia energética. Para comprender todas las variables que inciden sobre dicha eficiencia, se debe estudiar el funcionamiento del sistema, curvas características y sus parámetros, y poseer algún modelo matemático aproximado que contemple fenómenos no ideales tales como las pérdidas y fenómenos no deseados en el panel.

Ahora, existen muchos tipos de celdas solares [11]. El más común se compone de una junta *p-n*, la energía se genera por medio del efecto fotovoltaico y de la radiación electromagnética proveniente del sol, que incide sobre la capa del semiconductor. Los fotones al tener mayor energía que la banda prohibida del semiconductor crean un par electrón-hueco, y el campo eléctrico ejercido en la junta *p-n* mueve los electrones (*portadores*), produciendo una fotocorriente que es directamente proporcional a la radiación del sol [1]. Los semiconductores que componen el panel, poseen un comportamiento exponencial y no lineal muy similar al de un diodo [9].

3.1.1 Curvas Corriente-Tensión(I-V) para un PV

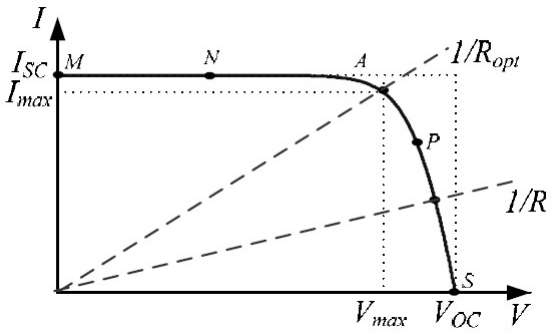

Figura 3.1: Curva característica de corriente(A)-tensión(V) para un un panel fotovoltaico, (Tomado de [2])

La curva característica de un PV se puede obtener manteniendo fijos los parámetros de irradiancia(S) y temperatura(T), en condiciones controladas. Si se tiene una carga en las terminales de salida, la potencia entregada solo dependerá del valor de la carga, de manera que si la carga es pequeña (puntos M-N de la figura 3.1) el panel se comportará como una fuente de corriente, pero si la carga es grande (puntos PS de la figura 3.1) se comportará como una fuente de tensión [2].

Para realizar la caracterización de una celda se deben realizar las siguientes pruebas:

- *Corriente de corto circuito* I_{sc} : Se define como el valor máximo de la corriente generada por el panel, en condiciones de cortocircuito $V = 0$.
- *Tensión de circuito abierto* V_{oc}: Se define como el valor de tensión en la junta *p-n*, cuando se tiene una corriente generada $I = 0$.
- *Punto de máxima potencia*: El punto A de la figura 3.1 representa la potencia máxima de la carga resistiva, $P_{max} = V_{max} I_{max}$.

3.1.2 Modelos del panel fotovoltaico

Un panel fotovoltaico se puede modelar de manera simple (ideal), utilizando una fuente de corriente en paralelo con un diodo: la corriente de salida será proporcional a la radiación solar sobre la celda (foto-corriente).

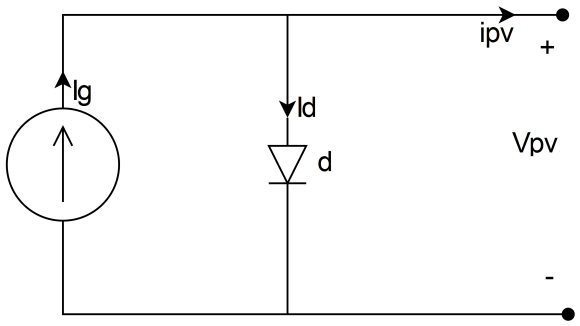

Figura 3.2: Modelo simple ideal para un panel fotovoltaico, compuesto por un diodo y una fuente de corriente

La figura 3.2 muestra el modelo básico. No obstante, este modelo es ideal. Por ello, se añade más complejidad al mismo, para reflejar más las no idealidades del panel. Tales variables son:

- Dependencia de la temperatura, corriente de saturación del diodo (Is) y foto corriente (Ig).
- Pérdidas debidas al flujo de corriente (Rs) y pérdidas con referencia a tierra (Rp).
- Número de celdas analisis análisis n.

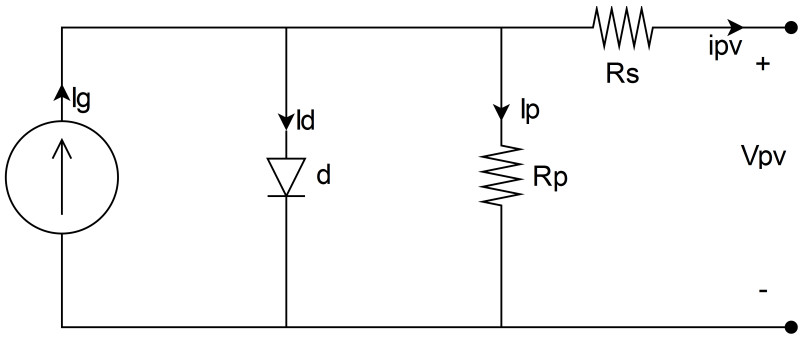

Figura 3.3: Modelo con características no ideales incluidas para un panel fotovoltaico, compuesto por un diodo, una fuente de corriente, perdidas resistivas por cada celda

A partir del modelo con pérdidas se puede deducir la ecuación que describe las corrientes i_{pv} e I_g , como sigue a continuación [4]:

$$i_{pv} = I_s - i_d + i_p \tag{3.1}$$

$$I_g = 2i_{pv} + \frac{V_{pv} + i_{pv}R_s}{R_p} - I_s + I_s e^{\frac{V_{pv}+i_{pv}V_{pv}}{nv_t}} \tag{3.2}$$

Despejando

$$i_{pv} = \frac{1}{2}\left[I_s + I_g - \frac{V_{pv} + i_{pv}R_s}{R_p} - I_s e^{\frac{V_{pv}+i_{pv}V_{pv}}{nv_t}}\right] \tag{3.3}$$

En general, la corriente que fluye por las terminales de un generador fotovoltaico está determinada por tres funciones de corriente:

- *Ig*: corriente generada debido al efecto fotoeléctrico.
- *id*: corriente de pérdida debido a la juntura p-n.
- *ip*: corriente de pérdida de naturaleza resistiva.

Para obtener un modelo del comportamiento estático del generador fotovoltaico se supone lo siguiente:

- *Ig*: depende de la Irradiancia (S), pero no depende de la tensión en las terminales del generador fotovoltaico (Vpv).
- *ip* e *id*: dependen de la tensión Vpv.
- *ip*: depende de la temperatura (T).

De esta forma, la expresión que define i_{pv} es:

$$i_{pv}\left(V_{pv}, T, S\right) = i_g\left(V_{pv}\right) - i_d\left(V_{pv}, T\right) \tag{3.4}$$

Según se definan las funciones i_{pv} e i_d, se obtendrán modelos con complejidad y precisiones distintas, a partir de los siguientes casos:

Tabla 3.1: Modelos para un PV: ideal, con pérdidas en serie R_s y con pérdidas en paralelo R_p

Modelos	i_g	i_p	i_d
1	KS	-	$I_s\left(T\right)\left[e^{\frac{V_{pv}}{v_t}} - 1\right]$
2	KS	$G_p V_{pv}$	$I_s\left(T\right)\left[e^{\frac{V_{pv}}{v_t}} - 1\right]$
3	KS	-	$I_s\left(T\right)\left[e^{\frac{V_{pv}+i_{pv}R_s}{v_t}} - 1\right]$
4	KS	$G_p V_{pv} + G_p i_{pv} R_s$	$I_s\left(T\right)\left[e^{\frac{V_{pv}+i_{pv}R_s}{v_t}} - 1\right]$

De manera general se tiene:

$$i_{pv}\left(V_{pv}\right) = G_p V_{pv} + G_p i_{pv} R_s \tag{3.5}$$

$$i_d\left(V_{pv}\right) = I_s\left(T\right) e^{\frac{V_{pv}}{v_t}} e^{\frac{i_{pv}R_s}{v_t}} - I_s\left(T\right) \tag{3.6}$$

El modelo general del comportamiento estático de un generador PV, también se puede representar de la siguiente manera:

$$i_{pv} = KS - G_p V_{pv} + I_s\left(T\right) - G_p i_{pv} R_s - I_s\left(T\right) e^{\frac{V_{pv}}{v_t}} e^{\frac{i_{pv} R_s}{v_t}} \tag{3.7}$$

$$I_s\left(T\right) e^{\frac{V_{pv}}{v_t}} e^{\frac{i_{pv} R_s}{v_t}} = KS - G_p V_{pv} + I_s\left(T\right) - G_p i_{pv} R_s - i_{pv} \tag{3.8}$$

La ecuación 3.8 es no lineal. Aplicando una linealización, se tiene [3]:

$$y = ln\left(KS - G_p V_{pv} + I_s\left(T\right) - G_p i_{pv} R_s - i_{pv}\right) \tag{3.9}$$

si $I_g = KS >> I_s$

$$y = ln\left(KS - G_p V_{pv} - G_p i_{pv} R_s - i_{pv}\right) \tag{3.10}$$

$$z = V_{pv} + i_{pv} R_s \tag{3.11}$$

La ecuación lineal que describe el estimador de parámetros se define como:

$$y = \theta_1 z + \theta_2 \tag{3.12}$$

Posteriormente al proceso de cálculo de parámetros se tiene:

$$\theta_1 = ln\left(I_s\left(T\right)\right) \tag{3.13}$$

$$\theta_2 = \alpha \tag{3.14}$$

3.2 Algoritmo de CORDIC

El algoritmo *Coordinate Rotational Digital Computer* (CORDIC) es un método numérico, que realiza cierto número de iteraciones para encontrar el valor deseado, según sea la función que se desea calcular. Este algoritmo es utilizado para implementar funciones trigonométricas, logarítmicas y exponenciales. La facilidad de implementación, hace que sea uno de los algoritmos más utilizados en el ámbito de la electrónica digital. CORDIC utiliza desplazamientos, sumas, restas y tablas de busqueda(LUTs) con valores previamente precargados que dependerán de la operación en cálculo. Para este algoritmo existen tres metodos: el método circular, lineal y el hiperbólico.

Las ecuaciones generales para el algoritmo de CORDIC se definen como:

$$X_{i+1} = X_i - m d_i 2^{-i} Y_i \tag{3.15}$$

$$Y_{i+1} = Y_i - d_i 2^{-i} X_i \tag{3.16}$$

$$Z_{i+1} = Z_i - d_i e\,(i) \tag{3.17}$$

donde $e\,(i)$ se muestra en la tabla 3.2 según corresponde para cada caso:

Tabla 3.2: Sistema de coordenadas unificado (CORDIC): circular, linear e hiperbólico. Tomado de [5]

m	Sistema de coordenadas	Valor de $e\,(i)$
1	Circular	$tan^{-1}\,(2^{-i})$
0	Lineal	2^{-1}
-1	Hiperbólico	$tanh^{-1}\,(2^{-i})$

3.2.1 Sistema de coordenadas hiperbólico

Para el cálculo de algunas funciones con el algoritmo aumenta mucho la complejidad, de manera que se deben utilizar las siguientes identidades [6]:

$$\tan z = \frac{\sin z}{\cos z} \tag{3.18}$$

$$\tanh z = \frac{\sinh z}{\cosh z} \tag{3.19}$$

$$\exp z = \sinh z + \cosh z \tag{3.20}$$

$$\ln \omega = 2 \cdot \tanh^{-1}\left(\frac{y}{x}\right) \tag{3.21}$$

donde:

$$x = \omega + 1 \tag{3.22}$$

$$y = \omega - 1 \tag{3.23}$$

3.2.2 Logaritmo natural utilizando el algoritmo hiperbólico de CORDIC

Para la función $Ln(\omega)$, se puede utilizar algoritmo de CORDIC en modo hiperbólico [12]. Primera se debe calcular la función $\tanh^{-1}\left(\frac{y}{x}\right)$ a partir de las siguientes ecuaciones:

$$X_{i+1} = X_i + d_i \cdot 2^{-i} \cdot Y_i \tag{3.24}$$

$$Y_{i+1} = Y_i + d_i \cdot 2^{-i} \cdot X_i \tag{3.25}$$

$$Z_{i+1} = Z_i - d_i \cdot tanh^{-1}\left(2^{-i}\right) \tag{3.26}$$

Donde i es el indice de cada iteración. Las iteraciones 4, 13, 40,... k, 3k+1 se deberán repetir para garantizar la convergencia. d_i es el signo de Y_i invertido, es decir que cuando el signo de Y_i es negativo, d_i será positivo y viceversa.

Utilizando la ecuación 3.21, se definen los valores iniciales $X_0 = \omega + 1$, $Y_0 = \omega - 1$ y $Z_0 = 0$, cuando $i = 0$.

Cabe destacar que el rango de convergencia para este algoritmo [7] esta definido como:

$$0,106843 \leq \omega \leq 9,35947 \tag{3.27}$$

donde ω es el valor del argumento del logaritmo natural.

El resultado final de Z_i, contiene el valor de $\tanh^{-1}\left(\frac{y}{x}\right)$, sin embargo se debe multiplicar por un factor de 2 para completar el cálculo del logaritmo natural, según la identidad de la ecuación 3.21

3.2.3 Exponencial utilizando el algoritmo hiperbólico de CORDIC

Para una función $e^{(\omega)}$, con el algoritmo de CORDIC, se debe utilizar las ecuaciones 3.24, 3.25 y 3.26 de manera iterativa, donde el valor final de X y Y, son el resultado de $cosh(\omega)$ y $\sinh(\omega)$ respectivamente. Se debe tomar en cuenta la repetición de las iteraciones (i) 4, 13, 40,... k, 3k+1 para garantizar la convergencia dando una mejor precisión en el cálculo.

Los valores iniciales se definen como constantes $X_0 = 1,20753406$, $Y_0 = 0$ y $Z_0 = \omega$, donde ω, es el valor del argumento que se desea calcular, y d_i es el signo de Z_i.

Cabe destacar que el rango de convergencia para este algoritmo se puede definir como:

$$0 \leq \omega \leq 1 \tag{3.28}$$

El valor final del cálculo, se obtiene por medio de una suma de dos funciones, dadas en la identidad de la ecuación 3.20.

3.3 Coma flotante

La codificación para el formato coma flotante se realiza mediante el estándar *IEEE 754*, este requiere de tres campos en la palabra [10]:

- Signo
- Exponente
- Mantisa

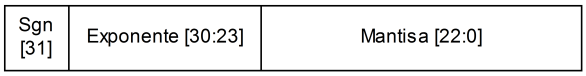

Figura 3.4: Formato IEEE 754 coma flotante para 32 bits

Para el formato *IEEE 754* de la figura 3.4 en 32-bits [8] se asigna:

- 1 Bit para signo
- 8 Bits para exponente
- 23 Bits para mantisa

Donde el bit de signo sigue la representación típica del formato signo-magnitud: 0=negativos, 1=positivos.

El exponente (8-bits) puede representar un rango desde 0 hasta 255, sin embargo se tiene:

- $[0 - 127]$ exponentes negativos
- $[128 - 255]$ exponentes positivos

De esta manera para convertir un exponente positivo, en el valor correspondiente del formato coma flotante se debe realizar la siguiente suma: $exp_{float} = 127 + exp_{numero}$. Por otro lado si se desea pasar de un valor decimal a coma flotante se deben realizar los siguientes pasos:

1. Representar el signo con su debido bit
2. Conversión decimal a binario coma fija
3. Conversión binario a notación científica
4. Agrupar en signo, exponente y mantisa

3.4 Coma fija

La representación de un número en coma fija cuenta con tres partes fundamentales:

- Signo
- Parte entera
- Parte fraccionaria

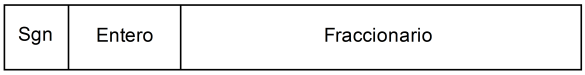

Figura 3.5: Formato para un número en coma fija

En la aritmética simple se utilizan los signos +/-, para saber si un número es positivo o negativo, para representar el signo en coma fija se utiliza el bit más significativo. Si el número es positivo, el bit de signo será un 0 y si el número es negativo, será un 1 [13]. Esta representación puede ser signo-magnitud o de complemento a dos [14]. La coma se asigna de manera arbitraria según se requiera.

Capítulo 4

Sistema de linealización

La corriente i_{pv} generada por un panel, posee un comportamiento exponencial, de manera que se requiere de una linealización. Esta se realizará por medio de una función logarítmica que se estimará a través de un algoritmo de cálculo denominado CORDIC, el cual permite realizar una aproximación de la función de manera recursiva con una cantidad de iteraciones finita. El algoritmo de CORDIC para un logaritmo natural utiliza el sistema de coordenadas hiperbólico.

El rango de convergencia para este algoritmo es de $0,106843 < T < 9,35947$ donde T es el argumento del logaritmo natural. El valor máximo $T = 0,58A$ es seleccionado a partir de la corriente en condiciones máximas para el panel. Para aprovechar de una mejor forma el intervalo de convergencia del algoritmo, se puede escalar por medio de una división entre una constante $C = 16$ seleccionada de manera que se pueda realizar el escalado a través de desplazamientos. El nuevo intervalo de convergencia está acotado dentro de los rangos: $0,00667769 < T < 0,58496687$. Este desplazamiento en el rango de convergencia se realizó debido a que en los sistemas reales en que se realizará la prueba final, se pueden presentar valores de corriente más bajos que 0,106843A.

La constante C utilizada en el escalado se debe compensar en el logaritmo con la siguiente igualdad:

$$Ln\,(T) = Ln\,(16T) - Ln\,(16) \tag{4.1}$$

4.1 Algoritmo de CORDIC hiperbólico implementado en software

Para comprobar el debido funcionamiento del este algoritmo se crea un programa de alto nivel en Python.

```
def CordicLn(T): #Algoritmo de cordic para resolver un Ln

    #Condiciones iniciales
    Z_ant=0
    X_ant=T+1
    Y_ant=T-1

    #Look-up table
    Bm=[0.54930614, 0.25541281, 0.12565721, 0.06258157, 0.06258157, 0.03126
    i=0

    #Cantidad de desplazamientos por iteracion
    Em=[ 2**-1, 2**-2, 2**-3,2**-4,2**-4,2**-5,2**-6,2**-7,2**-7,2**-8,2**-


    while i<23:

        Dm= -Y_ant/(abs(Y_ant))          # Signo de Y
        Z_act= Z_ant + Dm*-2*Bm[i]       # Calcula el valor de Z actual
        X_act= X_ant + Em[i]*Dm*Y_ant    # Calcula el valor X actual
        Y_act= Y_ant + Em[i]*Dm*X_ant    # Calcula el valor Y actual
        Z_ant = Z_act    # Agrega el valor actual al valor anterior de Z
        Y_ant = Y_act    # Agrega el valor actual al valor anterior de Y
        X_ant = X_act    # Agrega el valor actual al valor anterior de X
        i = i + 1        # Contador para cada iteracion
        ln=Z_act         # Resultado para cada iteracion

    return ln #resulado final del logaritmo natural


>>>
>>> T=0.2
>>> CordicLn(16*T) - math.log(16)
-1.6094370822397814
>>> math.log(0.2)
-1.6094379124341003
>>>
```

Figura 4.1: Algoritmo de CORDIC en Python

En la figura 4.1 se observa el programa realizado para la verificación del algoritmo en alto nivel. Este programa cuenta con las modificaciones realizadas en el rango de convergencia, con respecto al escalado y compensación.

Para simplificar el diseño e implementación del hardware, se realiza una serie de modificaciones en las ecuaciones del algoritmo:

- Se sustituye la resta del valor actual de Z_{i+1} por una suma.
- Se incluye el signo en la tabla LUT de Z.
- El resultado final del algoritmo se debe multiplicar por 2. Sin embargo este escalado se puede realizar a los valores almacenados en la LUT de Z, esto para evitar una multiplicación.

4.2 Sistema de linealización implementado en hardware (CORDIC)

Figura 4.2: Bloque principal de linealización: Entradas y salidas para la unidad logaritmo natural basada en el algoritmo de CORDIC e implementado en hardware.

La figura 4.2 contiene el bloque general del algoritmo de CORDIC, que poseen 4 entradas: CLK , T , Begin_LN , RST_LN y 4 salidas: ACK_LN , RESULT , U_F , O_F

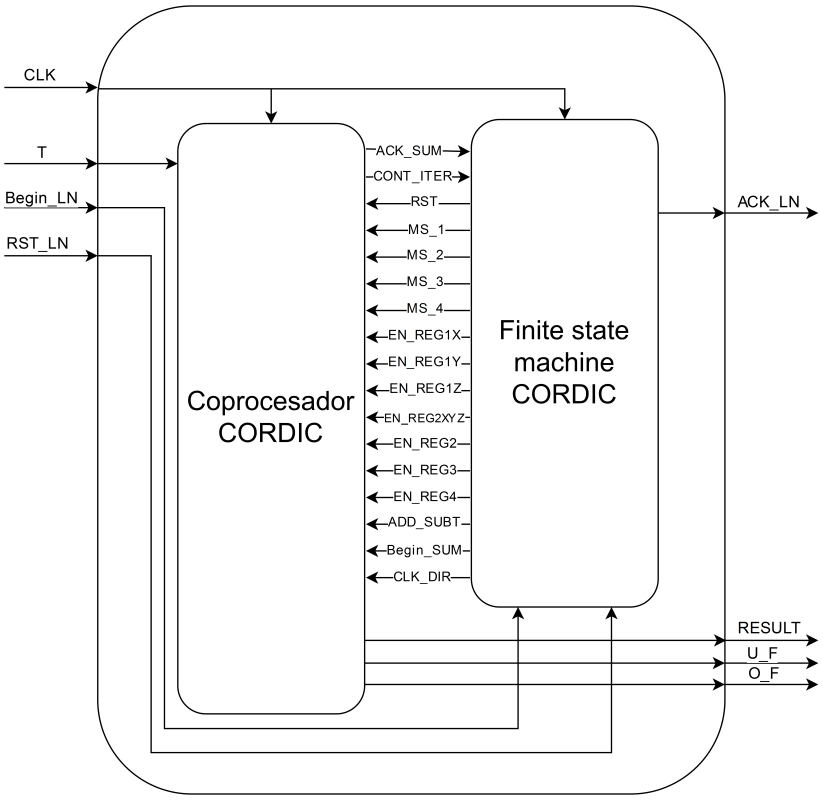

Figura 4.3: Sistema de linealización: unidad de coma flotante de 32bits basada en el algoritmo de CORDIC, con su respectiva máquina de estados finita como control.

El sistema de linealización de la figura 4.3 cuenta con dos módulos principales:

- *Coprocesador Cordic*: realiza todas las operaciones requeridas por el algoritmo y se encarga del manejo de los datos en el cálculo.
- *Control*: se encarga de proveer las señales de control requeridas por el coprocesador, según las condiciones que se tenga en cada estado.

Señales de datos:

- *T*: dato de entrada, argumento del logaritmo natural.
- *RESULT*: resultado de la operación logaritmo natural.

Señales de control:

- *CLK*: reloj del sistema.
- *Begin_LN*: encargada de dar inicio a la operación logaritmo natural.
- *RST_LN*: realiza un reset a la unidad CORDIC tanto para el coprocesador como para la máquina de estados.
- *ACK_LN*: indica que el cálculo ya fue realizado.
- *Begin_SUM*: se encarga de dar inicio a la unidad de suma-resta coma flotante.
- *ACK_SUM*: indica que ha realizado el cálculo en la unidad de suma-resta coma flotante.
- *O_F*: indica si la suma-resta en coma flotante tiene un over-flow.
- *U_F*: indica si la suma-resta en coma flotante tiene un under-flow.
- *CLK_DIR*: activa el enable del contador de iteraciones.
- *CONT_ITER*: indica el número de iteración y sirve de comparación para que la máquina de estados pueda detenerse en el número de iteraciones que se requieran.
- *RST*: realiza el reset de todos los registros de la unidad.
- *MS_1 , MS_2 , MS_3 , MS_4*: Realizan la selección de cada multiplexor, Mux1, Mux2, Mux3, Mux's4 respectivamente.
- *EN_REG1X , EN_REG1Y , EN_REG1Z*: activa los enable de los registros de la primera etapa REG1X, REG1Y, REG1Z respectivamente, para almacenar datos.
- *EN_REG2XYZ*: activa el enable del registro REG2XYZ de la segunda etapa.
- *EN_REG2*: activa el enable del registro REG2 de la segunda etapa.
- *EN_REG3*: activa el enable del registro REG3 del dato inicial.
- *EN_REG4*: activa el enable del registro REG4 del dato final.

4.3 Diseño e implementación de la unidad de linealización por medio del algoritmo de CORDIC

El diseño de este algoritmo se basa en una arquitectura segmentada, que almacena y procesa varios datos en las distintas etapas. Para esto es de suma importancia una buena sincronización y así evitar datos erróneos a través del proceso de cálculo. Este coprocesador utiliza el formato IEEE 754 de 32Bits, para una adecuada exactitud en la aproximación del logaritmo natural y poder utilizar un número menor de iteraciones.

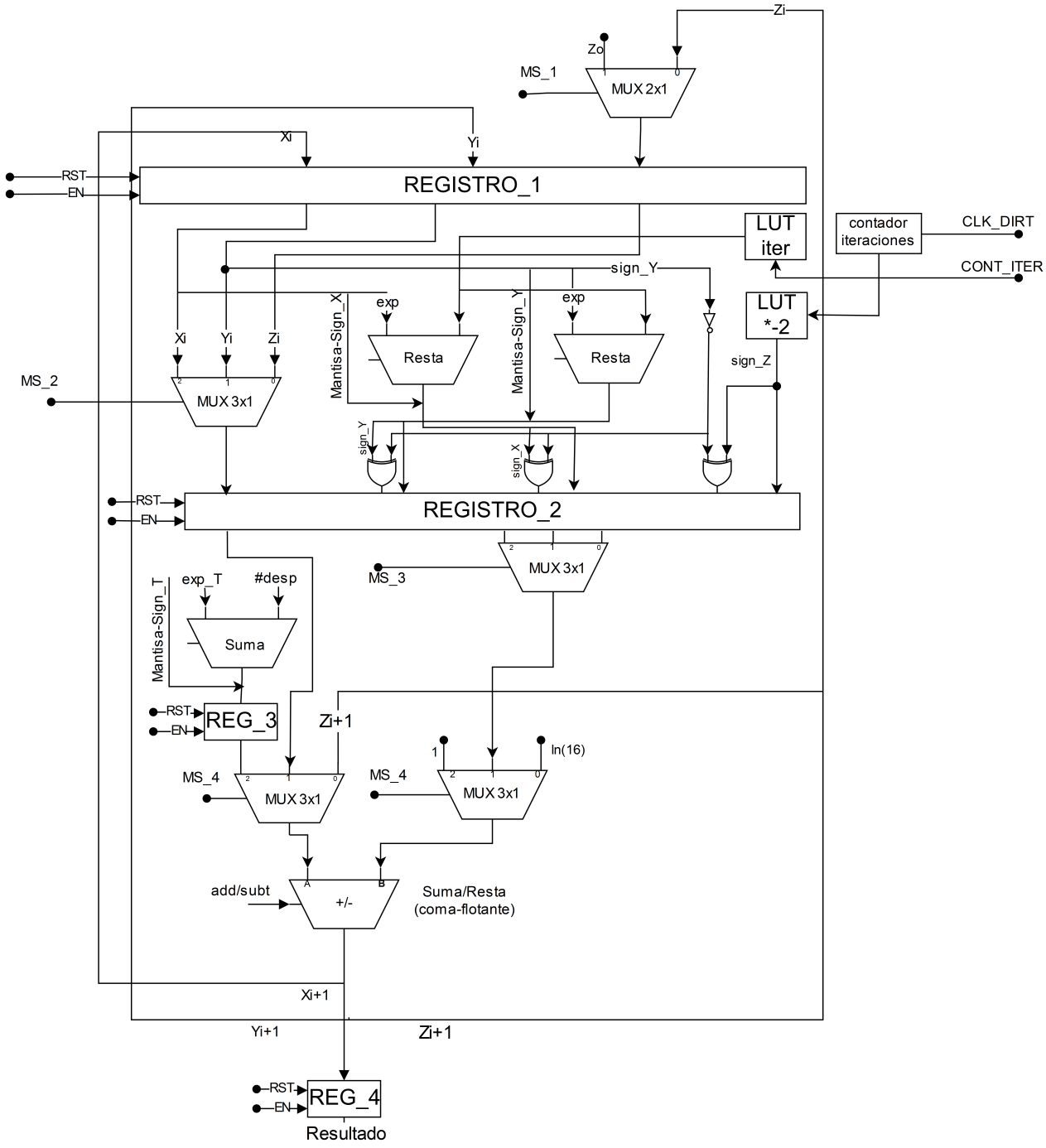

Figura 4.4: Diagrama a nivel de bloques del módulo segmentado diseñado para el cálculo de una función logarítmica con el algoritmo de CORDIC en hardware

En la figura 4.4 se puede observar la arquitectura de procesamiento en coma flotante propuesta para el algoritmo de CORDIC.

Esta etapa de linealización inicia con la carga de las condiciones iniciales, donde Z_0 contiene un valor inicial cero. Para el valor inicial de X_0 y Y_0 primeramente se aplica el escalado 16*T, hecho por medio de cuatro desplazamientos 2^{-4}. Por lo tanto este movimiento en formato coma flotante, se traduce como una suma de 4 en el exponente del argumento T. Posteriormente se realizan las siguientes operaciones en coma flotante, $X_0 = T + 1$ y $Y_0 = T - 1$. Estas tres constantes dan inicio al proceso de cálculo de manera iterativa, por lo que se requiere almacenarlas en un registro $(Registro1)$ en la primera etapa de segmentación.

Este método (CORDIC) es un cálculo cruzado es decir, para el próximo valor X_i se requiere el valor de Y_0 con un desplazamiento (resta en el exponente de la variable Y_0) y un cambio de signo δ_i. Para Y_i se aplica un concepto similar con valores de X_0 según las ecuaciones que describen el algoritmo en modo hiperbólico. La variable de mayor interés es Z_i esta contiene el valor final del cálculo, para esto se requiere una ROM con valores previamente cargados (LUT_Z) y el signo δ.

Para el signo δ de las operaciones en el algoritmo de CORDIC hiperbólico, se utiliza un circuito de comparación entre los signos de X_i, Y_i y Z_i contra el signo de Y_i invertido, la siguiente tabla describe el funcionamiento del circuito diseñado:

Tabla 4.1: Signo δ de la iteración siguiente para las variables X_{i+1} , Y_{i+1} y Z_{i+1}, comparando el signo de las variables Xi , Yi y Zi contra el signo de Yi invertido

Sign X_0	Sign Z_0	Sign Y_0	Sign $\sim Y_0$	Sign X_i	Sign Z_i	Sign Y_i
0	0	0	1	1	1	1
0	0	1	0	0	0	1
0	1	0	1	1	0	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1

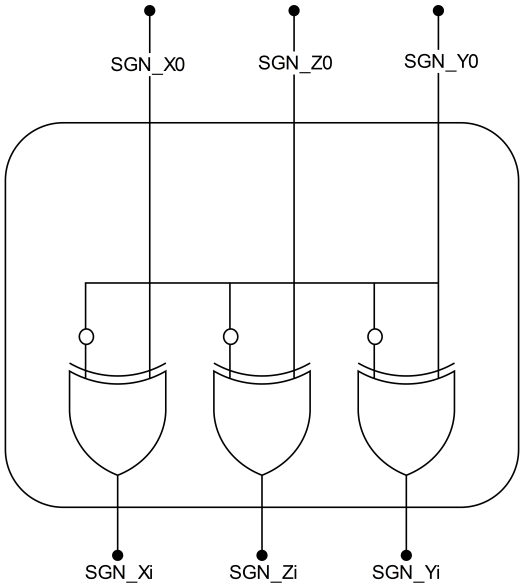

Figura 4.5: Circuito de comparación de signo actual δ, para las variables X_{i+1} , Y_{i+1} y Z_{i+1} de la iteración siguiente, utilizando la tabla 4.1

A partir de la tabla 4.1 se extrae el circuito de comparación de signo de la figura 4.5.

Dentro del diseño del linealizador se requiere de tablas de rápido acceso, precargadas con los valores constantes para cada iteración.Se necesitan al menos dos tipos de LUTs:

- *LUT_Z*: Almacena los valores de $-2 \cdot arctanh\,(2^{-i})$, para cada iteración.
- *LUT_ITER*: Almacena los desplazamientos que se deben realizar para cada iteración, ya que las iteraciones 4 y 13 repiten desplazamientos como se menciona en el marco teórico.

El acceso a cada valor de la tablas se realiza mediante un único contador de iteraciones, que a cada tabla la dirección del dato que se quiere extraer. Ambas LUT se encuentran sincronizadas en cuanto al numero de iteración.

Anteriormente se describió el proceso para el cálculo de las constantes iniciales X_0 , Y_0 y Z_0. El proceso para el cálculo de las variables X_{i+1} , Y_{i+1} y Z_{i+1}, no se puede realizar de manera simultanea, ya que solo se cuenta con un sumador coma flotante, decisión tomada para disminuir el área de la unidad total.

Las variables modificadas (desplazamiento y signo) son almacenadas en el *Registro 2*. La secuencia de cálculo toma primeramente los valores que se necesitan para X_{i+1}; luego se realiza la suma $X_i + \delta \cdot Y_{desplazado\_i}$ y se almacena el resultado en el *Registro 1*. Seguidamente se procede con el cálculo de Y_{i+1} y se realiza la suma $Y_i + \delta \cdot X_{desplazado\_i}$ almacena en el *Registro 1*, por ultimo se calcula el valor de Z_{i+1}, con la suma $Z_i + LUT_i$. Esta se almacena en el *Registro 1*; este proceso se realiza N iteraciones (N=número entero). La resta $Z_i - Ln\,(16)$ contrarresta el efecto del escalado aplicado al argumento (T), al inicio del cálculo del logaritmo natural. El resultado final Z_i contiene el valor $Ln\,(T)$ y

es almacenado en el *Registro 4*.

4.4 Sistema de control para la unidad de coprocesamiento CORDIC por medio de un máquina de estados finita (FSM)

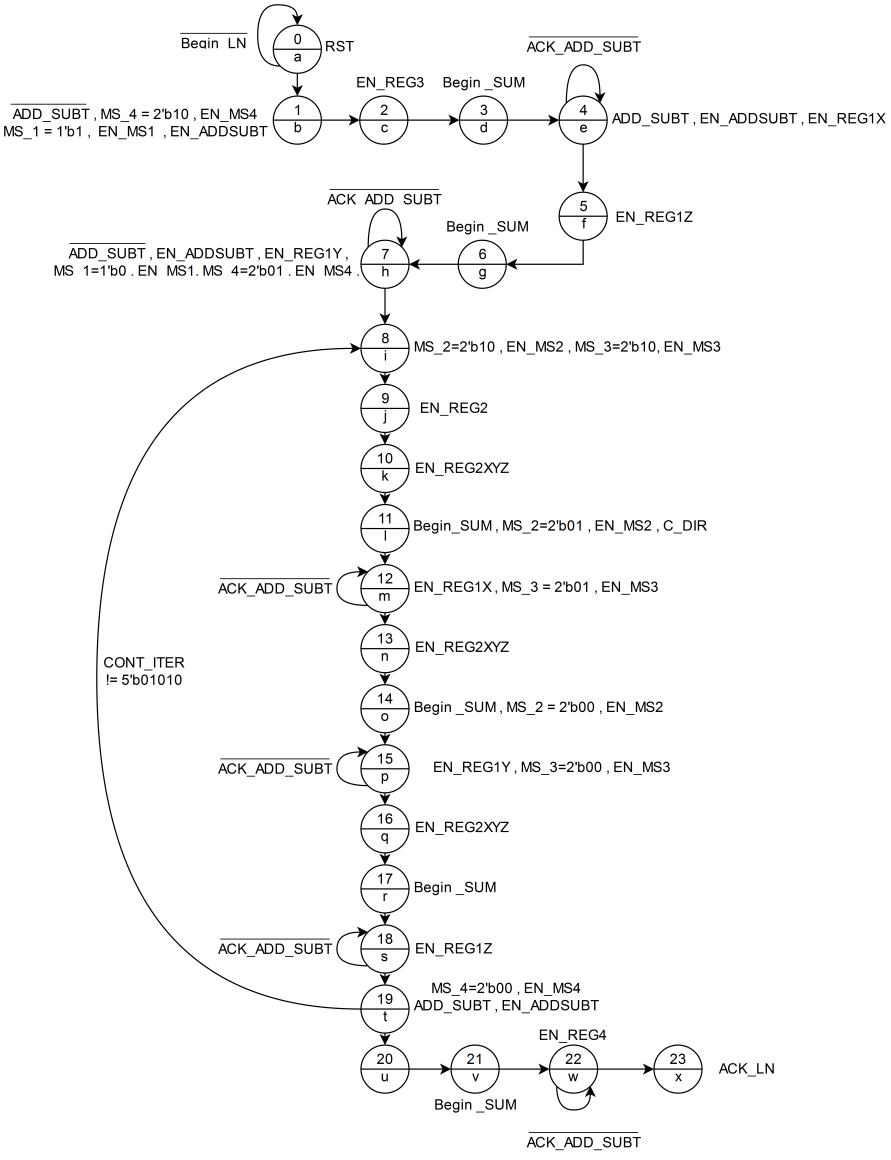

Figura 4.6: Máquina de estados finitos para para la unidad de coprocesamiento CORDIC

El sistema de control necesita sincronizarse, ya que los datos deben ser almacenados de manera correcta y estar listos con cierto tiempo ("setup-time") antes de que requeridos por otro segmento del coprocesador CORDIC. Se diseñó una máquina de estados finitos,

donde inicialmente se calculan los valores iniciales de X_0, Y_0 y Z_0. La máquina brinda las señales de control requeridas, dentro de una secuencia de cálculo de X_{i+1},Y_{i+1} y Z_{i+1} respectivamente, y cada vez que se recorre esta secuencia, se realizará una cuenta de iteración. Esa máquina posee con una variable de entrada que indica el número de iteración en el que se encuentra el algoritmo, de manera que cuando se llega a la iteración N, finalice el cálculo.

4.5 Verificación del módulo CORDIC sobre una placa de desarrollo Nexys-4

La unidad se describió en Verilog. Inicialmente se realizaron pequeños bloques pertenecientes a cada elemento requerido por la arquitectura diseñada. Se desarrolló el coprocesador CORDIC y la unidad de control en bloques separados, de manera que se pudieran realizar pruebas sin dependencia de los bloques entre sí, para una mejor depuración de errores, optimización y rediseño. Finalmente se procede a la etapa de pruebas, con simulaciones al bloque completo, como se muestra en la figura 4.7.

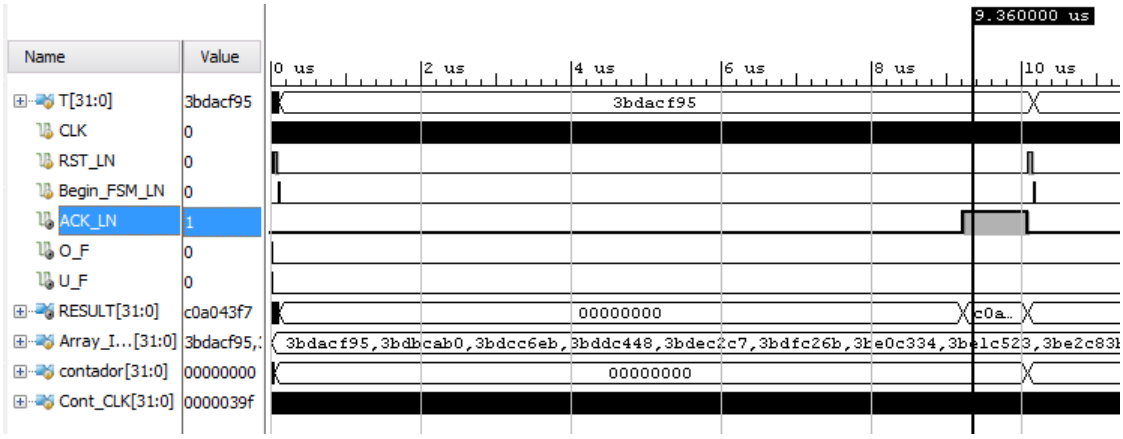

Figura 4.7: Simulación del linealizador implementado en Verilog, ingresando en la entrada un archivo de texto, con 1000 valores del intervalo de convergencia para el argumento del logaritmo natural

4.6 Simulación del circuito linealizador con algoritmo de CORDIC

Las simulaciones de la implementación del algoritmo, se realizaron mediante mil valores de entrada para el argumento del logaritmo natural, dentro del rango de convergencia, obteniendo mil valores de salida. Estos resultados experimentales se comparan con los valores reales, para dicha comparación se realizaron aproximaciones con 8, 12 y 15 iteraciones.

Primeramente se realiza una simulación que comprueba el funcionamiento en el rango de operación $0,00667769 < T < 0,58496687$. La linealización de los datos de prueba se realiza con la siguiente función:

$$Ln\,(T) \tag{4.2}$$

donde el argumento T es:

$$T = e^{-x} \tag{4.3}$$

El intervalo $0,536 < x < 5,009$ se divide en mil valores, estos se ingresan a la entrada por medio de un archivo de texto. El circuito linealizador CORDIC toma los datos, le aplica la función 5.2 y devuelve mil valores a través de otro archivo de texto, que se utilizará para análisis posteriormente.

4.6.1 Resultados de la simulación del rango de convergencia del circuito linealizador CORDIC

Para implementar un algoritmo o método numérico en hardware, es de suma importancia verificar que este funcione de manera adecuada en el rango de convergencia definido. La comprobación de la unidad de linealización mediante el algoritmo de CORDIC, se realizó por medio de una serie de pruebas, variando la cantidad de iteraciones para poder observar las diferencias entre exactitud y tiempo de ejecución. Se programó una simulación con mil valores de entrada, ingresados por medio de un archivo de texto previamente editado con los datos de entrada, con la función exponencial anteriormente descrita en la ecuación 5.3.

En la tabla 6.1 se puede observar el resumen de los resultados más relevantes para las simulaciones del rango de convergencia con distinta cantidad de iteraciones del algoritmo de CORDIC.

Tabla 4.2: Comparación de resultados experimentales obtenidos por medio de una simulación post-síntesis, a partir de los valores de entrada del rango de convergencia, utilizando 8, 12 y 15 iteraciones en el algoritmo de CORDIC. CLK=100MHz.

	8 iteraciones	12 iteraciones	15 iteraciones
Error máximo (%)	2,72	0,259	0,129
Error promedio (%)	0,40	0,0351	0,0257
Desviación estándar	0,39	0,043	0,0197
Número de ciclos	460	660	818
Tiempo completo de ejecución de la operación (μs)	4,6	6,6	8,18

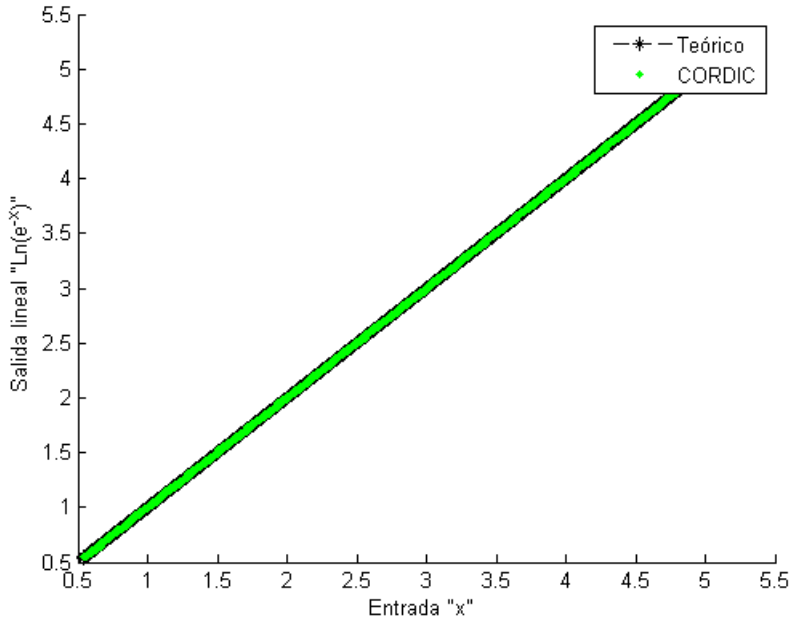

Figura 4.8: Comparación entre los datos de salida para la simulación post-sintesis del linealizador y la función logarítmica en Python, mediante mil valores de entrada dentro del rango de convergencia utilizando 8 iteraciones en el algoritmo de CORDIC

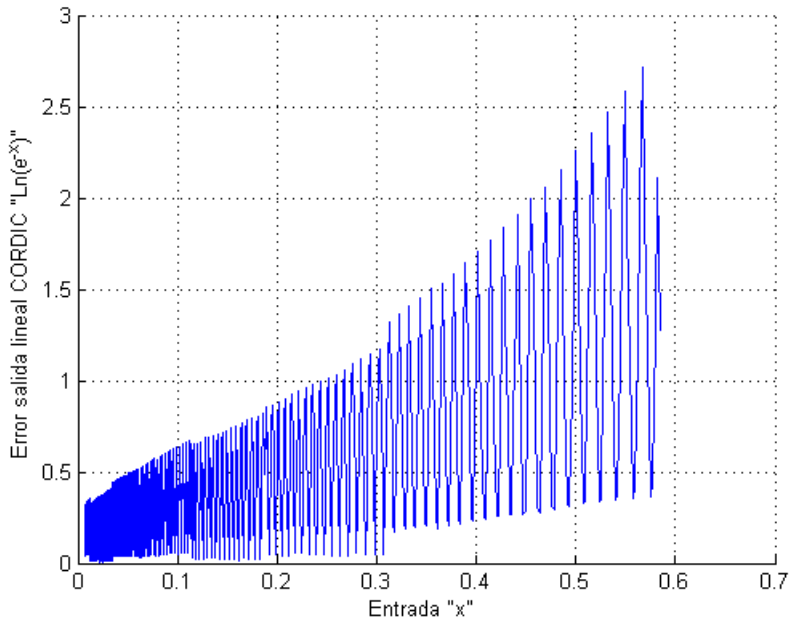

Figura 4.9: Porcentaje de error para los datos de la simulación post-síntesis del linealizador, con 8 iteraciones para el algoritmo de CORDIC, con un porcentaje de error máximo de 2,72% y un porcentaje de error promedio de 0,40%

Las simulaciones se hicieron ya sobre el código sintetizado. En el caso de 8 iteraciones, para el cálculo de cada valor se requieren 460 ciclos de reloj, desde el momento en se activa la señal Begin_LN hasta que se recibe la señal ACK_LN, que indica que se ha completado el cálculo. Es de suma importancia realizar la comparación entre el valor teórico y el valor calculado (figura 4.8). Para cada valor se calculó el error, la gráfica se puede observar en la figura 4.9. El porcentaje de error máximo es de 2,72% y el porcentaje de error promedio es de 0,40%.

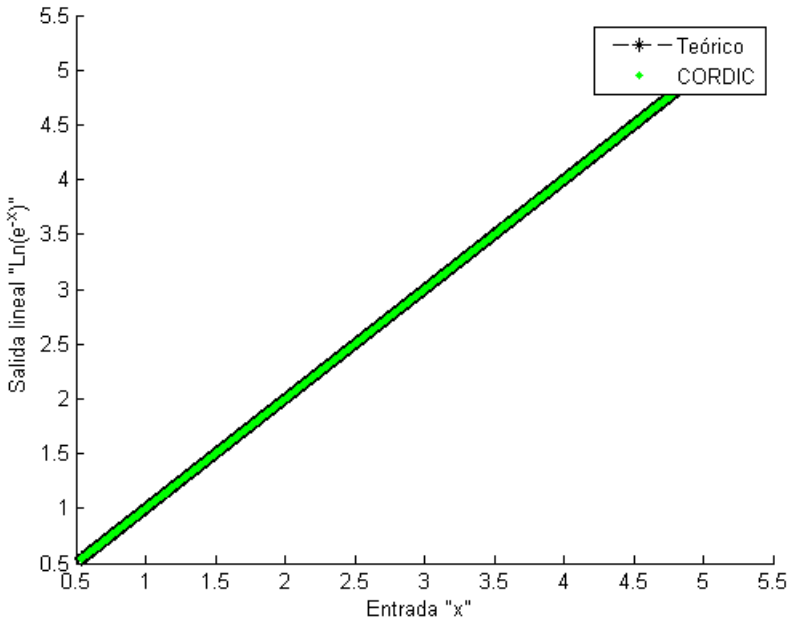

Figura 4.10: Comparación entre los datos de salida para la simulación post-sintesis del linealizador y la función logarítmica en Python, mediante mil valores de entrada dentro del rango de convergencia utilizando 12 iteraciones en el algoritmo de CORDIC

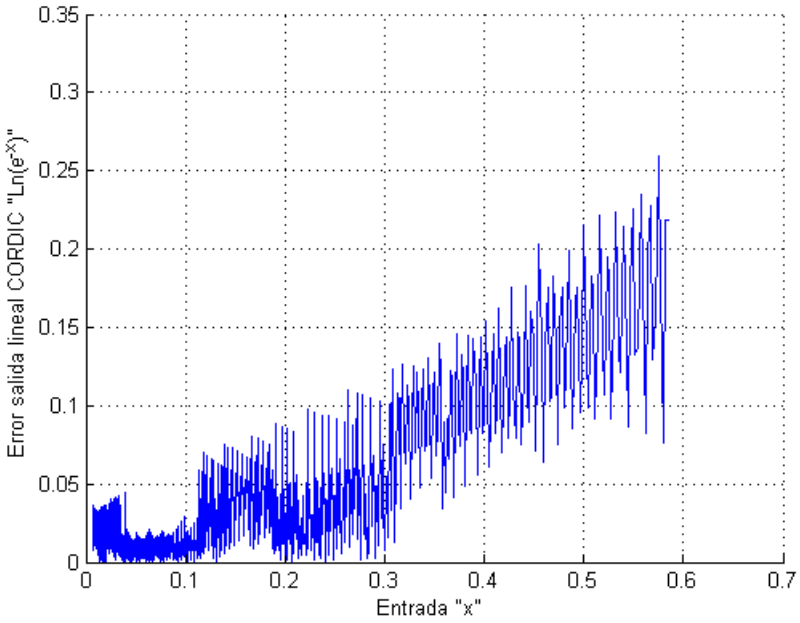

Figura 4.11: Porcentaje de error para los datos de la simulación post-síntesis del linealizador, con 12 iteraciones para el algoritmo de CORDIC, con un porcentaje de error máximo de 0,259% y un porcentaje de error promedio de 0,0351%.

Una mejor aproximación se puede realizar utilizando 12 iteraciones en el cálculo del logaritmo natural(ver figura 4.10). Se requieren 660 ciclos de reloj para ejecutar el cálculo completo. La figura 4.11 muestra el error en cada cálculo, donde el porcentaje de error máximo es de 0,259% y el porcentaje de error promedio es de 0,0351%.

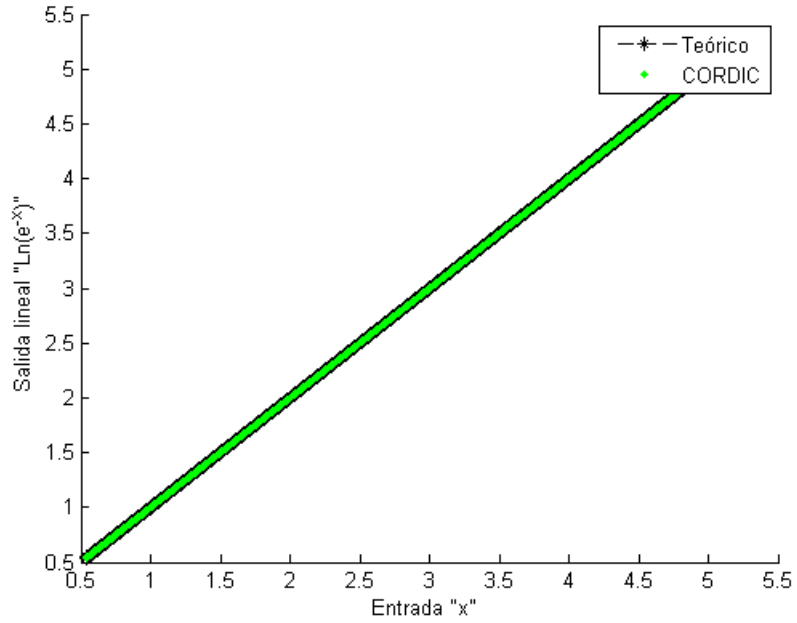

Figura 4.12: Comparación entre los datos de salida para la simulación post-sintesis del linealizador y la función logarítmica en Python, mediante mil valores de entrada dentro del rango de convergencia utilizando 15 iteraciones en el algoritmo de CORDIC

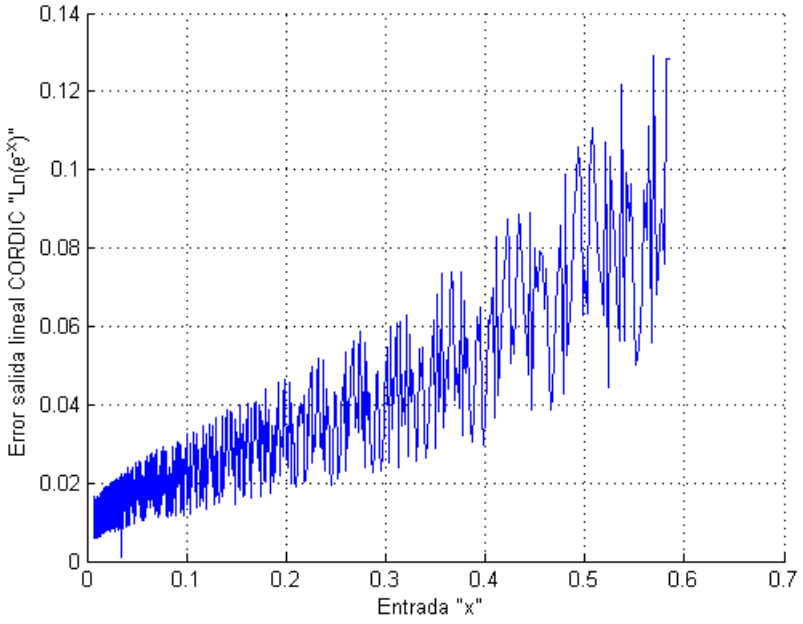

Figura 4.13: Porcentaje de error para los datos de la simulación post-síntesis del linealizador, con 15 iteraciones para el algoritmo de CORDIC, con un porcentaje de error máximo de 0,129% y un porcentaje de error promedio de 0,0257%.

Utilizando 15 iteraciones en el cálculo del logaritmo natural se logra la mejor aproximación. No obstante, se requieren 818 ciclos de reloj y se vuelve más lento el proceso(ver figura 4.12). La figura 4.13 muestra el error en cada cálculo, donde el porcentaje de error máximo es de 0,129% y el porcentaje de error promedio es de 0,0257%.

En las figuras de error 4.9, 4.11 y 4.13 se puede observar que el comportamiento del algoritmo es más exacto cuando los valores de corriente de entrada del argumento de linealizador "T" son pequeños, a medida que este argumento se vuelve mayor el porcentaje de error también incrementa.

Capítulo 5

Sistema de conversión coma flotante a coma fija y normalización

El circuito implementado para la linealización se basa en el formato IEEE 754, sin embargo el estimador de parámetros (unidad a la que el linealizador debe entregarle los datos) trabaja en coma fija. Se debe considerar una conversión entre ambos formatos, para esto se estudió cómo pasar de coma flotante a coma fija.

5.1 Diseño del sistema de conversión, normalización y control

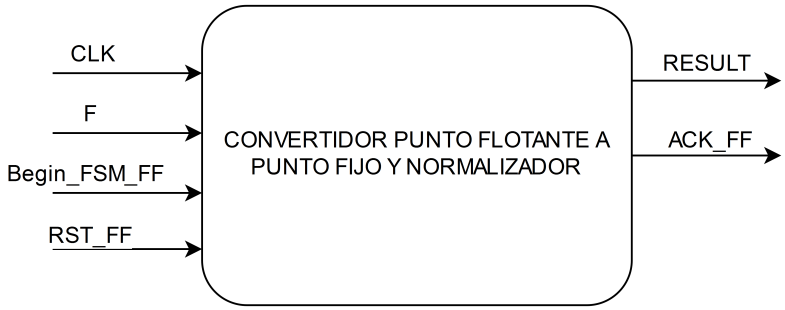

Figura 5.1: Diagrama general de entradas y salidas para el convertidor coma flotante a coma fija y normalizador.

La figura 5.1 contiene el bloque general de entradas y salidas para el sistema de conversión y normalización. Este posee 4 entradas: CLK , F , Begin_FF , RST_FF y 2 salidas: ACK_FF , RESULT.

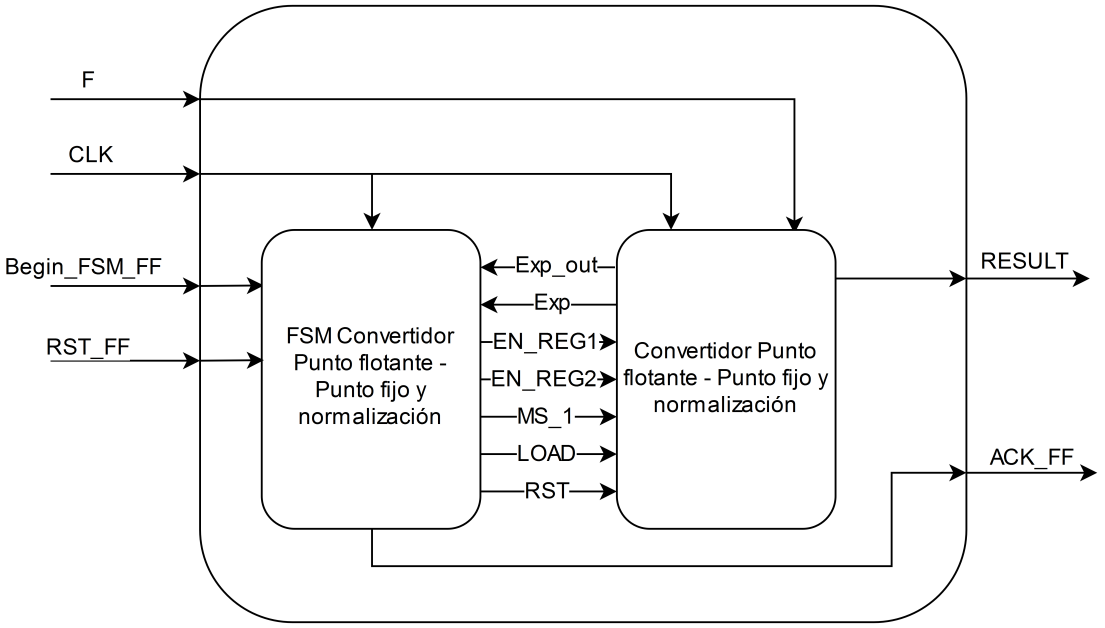

Figura 5.2: Sistema de conversión, normalización y control con su respectiva FSM. Se muestran señales de entrada, salida, datos y control.

El sistema de conversión y normalización de la figura 5.2 cuenta con dos módulos principales:

- *Convertidor-Normalizador*: realiza todas las operaciones requeridas para la conversión del formato coma flotante-coma fija y de la normalización.
- *Control*: se encarga de proveer las señales de control requeridas por el convertidor-normalizador, según las condiciones que se requiera en cada estado.

Señales de datos:

- F: dato de entrada en formato IEEE 754.
- $RESULT$: dato de salida en coma fija.

Señales de control:

- CLK: reloj del sistema.
- $Begin\_FF$: se encarga de iniciar la la unidad e indica a la máquina de estados que se debe iniciar la secuencia.
- RST_FF: restablece los valores iniciales del sistema de conversión y normalización.
- ACK_FF: indica cuando la conversión y la normalización han sido realizadas.

5.2 Convertidor coma flotante - coma fija y normalizador

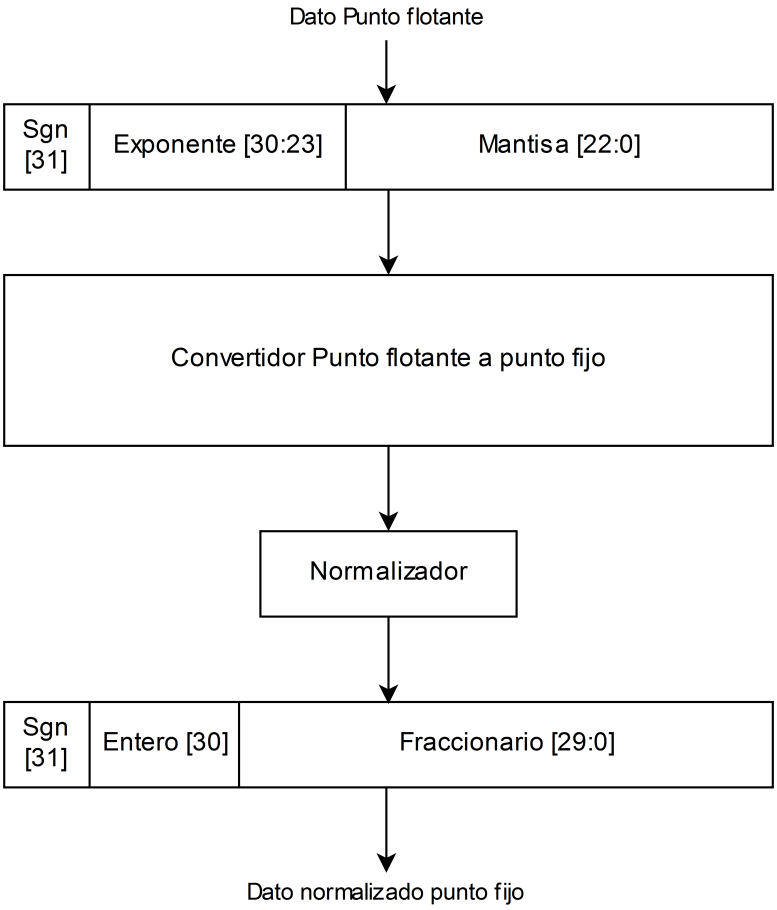

Figura 5.3: Diagrama general del sistema de conversión de coma flotante a coma fija y normalización corriente-tensión

En la figura 5.3 se puede observar el diagrama de solución propuesta para la arquitectura del convertidor-normalizador. Primeramente ingresa el número en formato IEEE 754 32-bits. Luego se efectúa la conversión a coma fija, en donde se asigna un bit de signo, 5 bits de parte entera y 26 bits para la parte fraccionaria. Este dato se procesa en una etapa de normalización y se obtiene el resultado. Este valor final esta normalizado para la corriente y tensión del panel, $V = [0, 1]$ e $i = [-1, 1]$. Para el formato coma fija solo se requiere un bit de signo, un bit en la parte entera y 30 para la parte fraccionaria (ver figura 5.3). El aumento de bits en la parte fraccionaria indica una mejor precisión en el resultado.

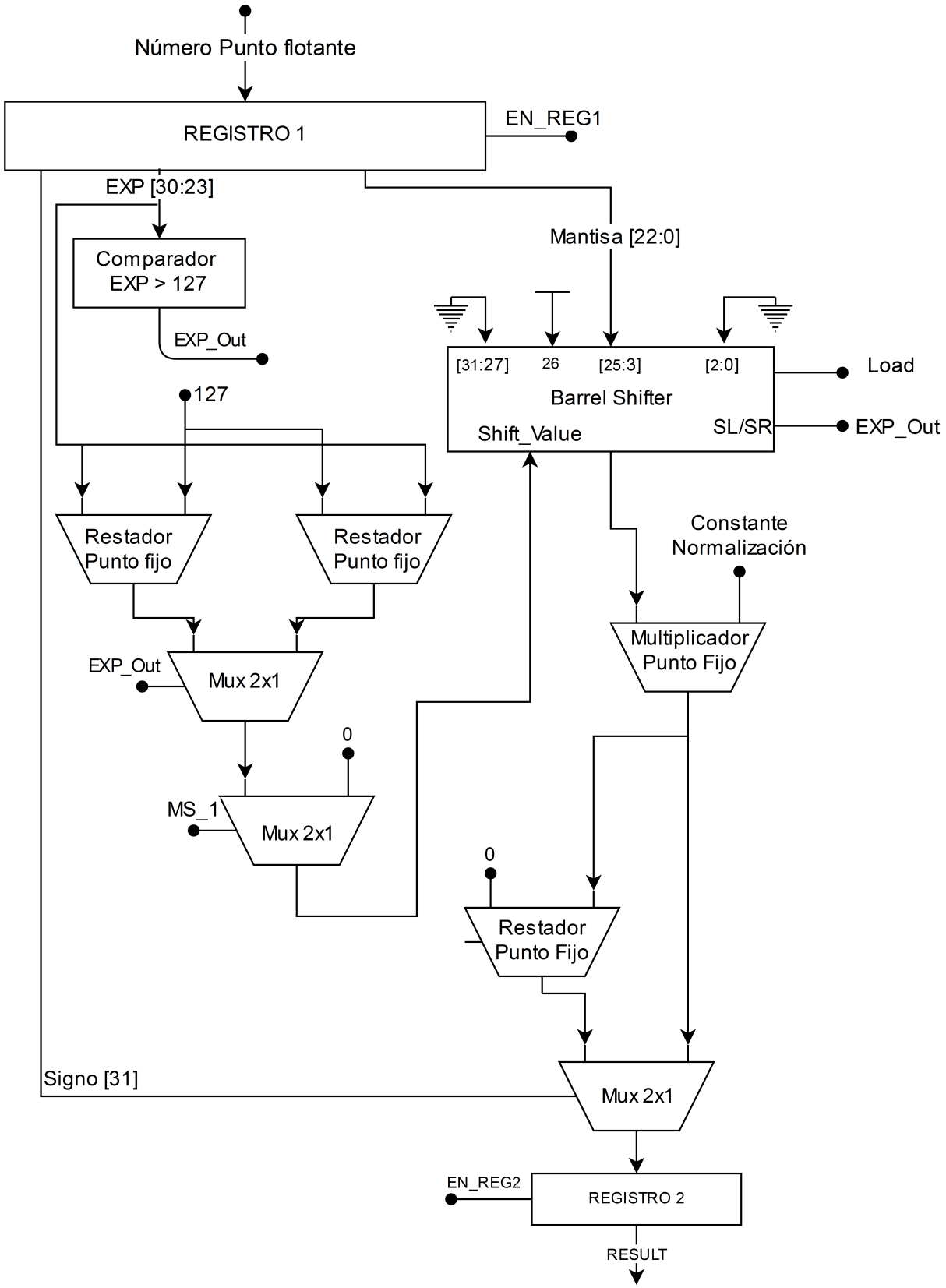

Figura 5.4: Módulo de conversión coma flotante a coma fija y normalización de corriente $[-1, 1]$ y tensión $[0, 1]$, con un dato de entrada en coma flotante y una salida en coma fija normalizada.

Como se observa en la figura 5.4, la etapa de conversión de formatos contiene un comparador. Este se utiliza para saber si el número, en formato IEEE 754, contiene un exponente mayor o menor que 127. Si el número es igual a 127, indica un exponente de 0, si es mayor que 127, la bandera de salida del comparador es igual a 1 $(EXP\_Out = 1)$, si se da esta condición, se debe realizar la operación $EXP - 127$. Si el exponente es menor que 127, la bandera EXP_Out es 0 y la operación es $127 - EXP$. Ambas operaciones indican la cantidad de desplazamientos que se deben realizar en el desplazador de barril, este se utiliza para ajustar la mantisa según sea el valor del exponente. La dirección de los desplazamientos se puede controlar mediante una señal que posee el desplazador de barril, esta es conectada a la bandera del comparador EXP_Out. El dato de entrada para el "barrel-shifter" esta compuesto por el bit más significativo en alto (fijo) y 23 bits de la mantisa del dato de entrada en coma flotante; este dato compuesto se le aplican los desplazamientos para obtener el resultado en coma fija.

Para normalizar un dato se requiere una división entre el máximo valor que se puede procesar. No obstante, dividir es generalmente caro en sistemas aritméticos, por lo que esta división se transforma en una multiplicación por una constante fraccionaria. Esta etapa de normalización tiene como entrada el valor convertido a coma fija, y es normalizado por medio de un multiplicador en coma fija, con una respectiva constante de normalización.

Las constantes de normalización se pueden calcular con la siguiente ecuación:

$$C_{norm} = \frac{1}{Valor_{MAX}} \tag{5.1}$$

La etapa de conversión y normalización se utiliza tanto para la corriente i_{pv} como para la tensión V_{pv} del panel. Esta constante de normalización varía para cada circuito:

$$Cv_{norm} = \frac{1}{18,1} = 0,055248618 \tag{5.2}$$

$$Ci_{norm} = \frac{1}{Ln\,(0,00667769)} = 0,199641045 \tag{5.3}$$

donde Cv_{norm} es la constante de normalización de la tensión y Ci_{norm} la constante de normalización de la corriente.

Después de la normalización, se debe tomar en cuenta el signo del valor inicial sin convertir (coma flotante). La unidad de conversión-normalización realiza una resta $0 - Dato\_coma\_fija$, y el multiplexor 2x1 del resultado selecciona el valor final en coma fija. Si el bit 32 del valor inicial es cero, el valor final en coma fija es positivo, de lo contrario el valor final en coma fija es negativo.

5.3 Control para el convertidor coma flotante - coma fija y normalizador

La arquitectura diseñada para el convertidor-normalizador en su mayoría es combinacional. No obstante, se requiere un sistema de control que detecta si se deben realizar desplazamientos, y cuando se debe almacenar datos en registros.

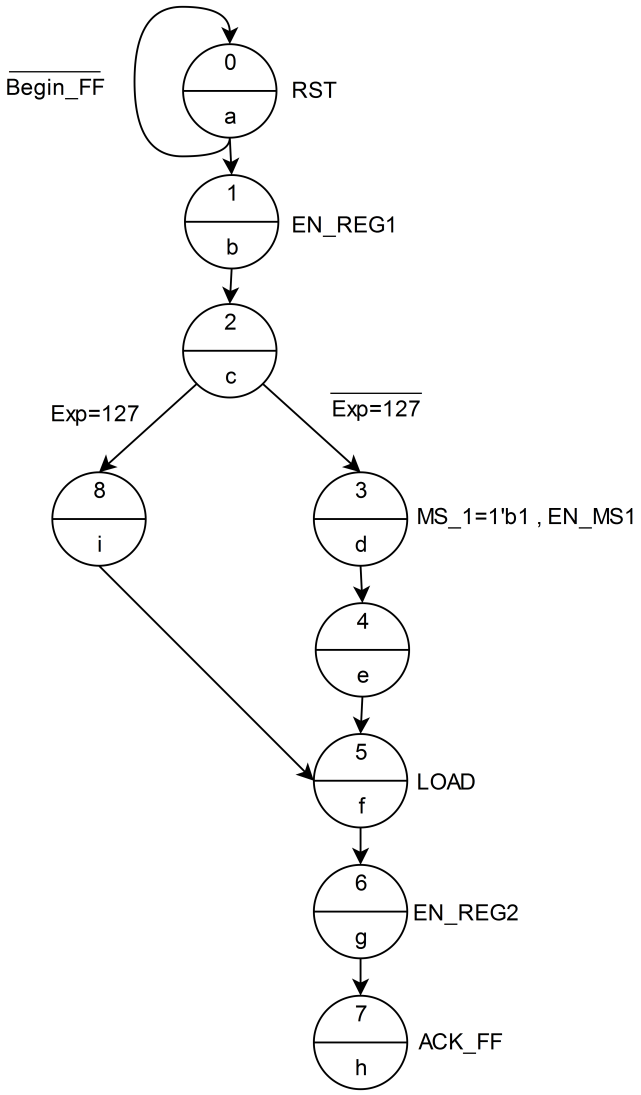

Figura 5.5: Diagrama de control del convetidor-normalizador, diseñado mediante una máquina de estados finita

El control de esta arquitectura se realiza por medio de una máquina de estado finita. El funcionamiento de la maquina se describe a continuacion:

- Estado (a): espera la señal $Begin\_FF$ para que la unidad de conversión-linealización sea iniciada, de manera que se ejecute un reset en los registros.
- Estado (b): guarda el dato en el *Registro 1*.

- Estado (c): verifica la condición $EXP = 127$, esta determina si se deben realizar desplazamientos.
- Estado (f): almacena en el desplazador de barril el dato convertido.
- Estado (g) almacena el resultado final en el *Registro 2*.
- Estado (h) indica mediante la bandera ACK_FF que el dato ya fue convertido y normalizado.

5.4 Verificación del módulo conversión-normalización sobre una placa de desarrollo Nexys-4

Se implementa la unidad de conversión-normalización y la unidad de control en bloques separados, de manera que se pudieran realizar pruebas sin dependencia de los bloques entre sí, para una mejor depuración de errores y re-diseño. Se realizan las simulaciones al bloque completo y se verifica su funcionamiento como se muestra en la figura 5.6.

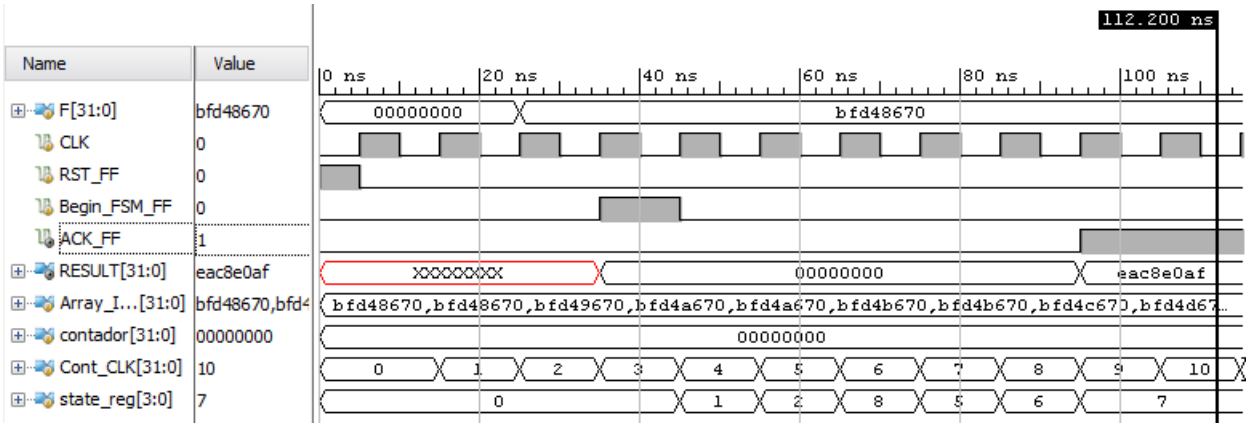

Figura 5.6: Simulación del circuito de conversión y normalización, ingresando en la entrada un archivo de texto, con 1000 valores de corriente lineal.

5.5 Resultados de la simulación post-síntesis del sistema de conversión-normalización

En la implementación de un diseño en hardware, es de suma importancia simular y verificar que este funcione de manera adecuada al comportamiento esperado teóricamente. Para la comprobación de la unidad de conversión-normalización, se realizó una serie de pruebas, programando una simulación con mil valores de entrada, ingresados por medio de un archivo de texto previamente editado. Éste contiene los datos de entrada del comportamiento según el modelo del panel. Las pruebas de esta unidad se realizaron con valores de corriente i_{pv} y valores de tensión V_{pv}, como se muestra en la tabla 5.1.

Tabla 5.1: Comparación de resultados experimentales obtenidos por medio de una simulación post-síntesis, a partir de los valores de entrada de corriente y tensión para el circuito de conversión-normalización. CLK=100MHz

	Corriente	Tensión
Error máximo (%)	0,0024837	0,0024837
Error promedio (%)	0,002478	0,002478
Desviación estándar ($x10^{-6}$)	1,94953	1,94953
Número de ciclos	8	8
Tiempo total de ejecución de la unidad (ns)	80	80

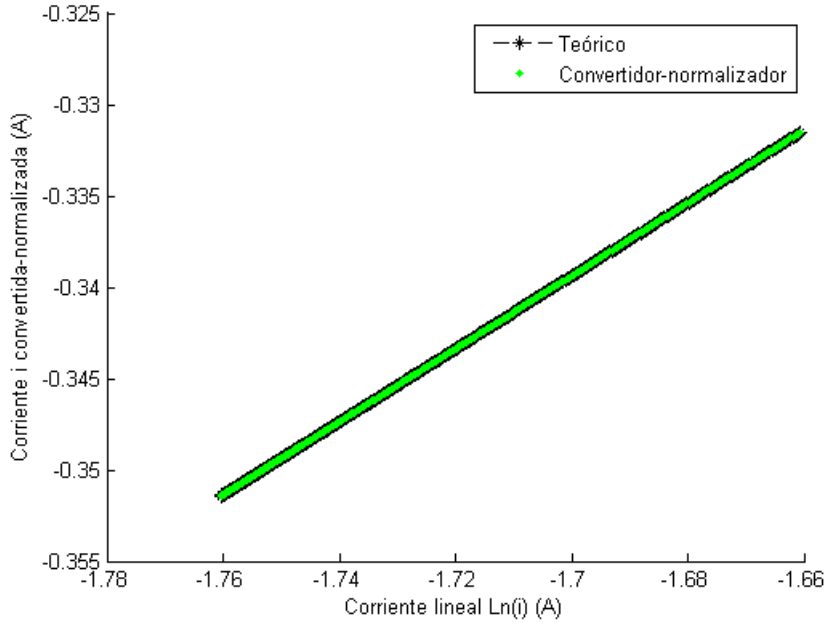

Figura 5.7: Comparación entre la conversión-normalización de corriente i_{pv} teórica y la simulación post-síntesis del circuito

En la figura 5.7 se presenta la comparación entre los resultados obtenidos teóricamente y experimentalmente, a través de la simulación post-sintesis del circuito normalizador-linealizador. Se contrastan los resultados contra el modelo en Python de estas operaciones.

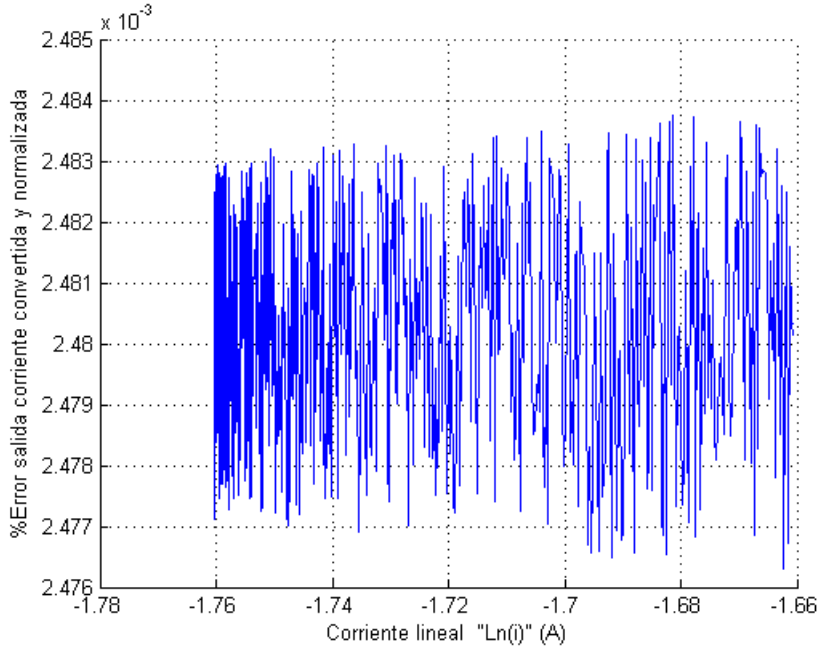

Figura 5.8: Porcentaje de error entre la conversión-normalización de corriente i_{pv} teórica y experimental del circuito. Los valores teóricos se obtienen del modelo en Python de las operaciones deseadas.

En la figura 5.8 se puede observar el error entre la conversión de la corriente normalizada teórica y experimental, con un error porcentual máximo de 0,0024837%, un error promedio de 0,002478% y una desviación estándar de $1,94953 \cdot 10^{-6}$.

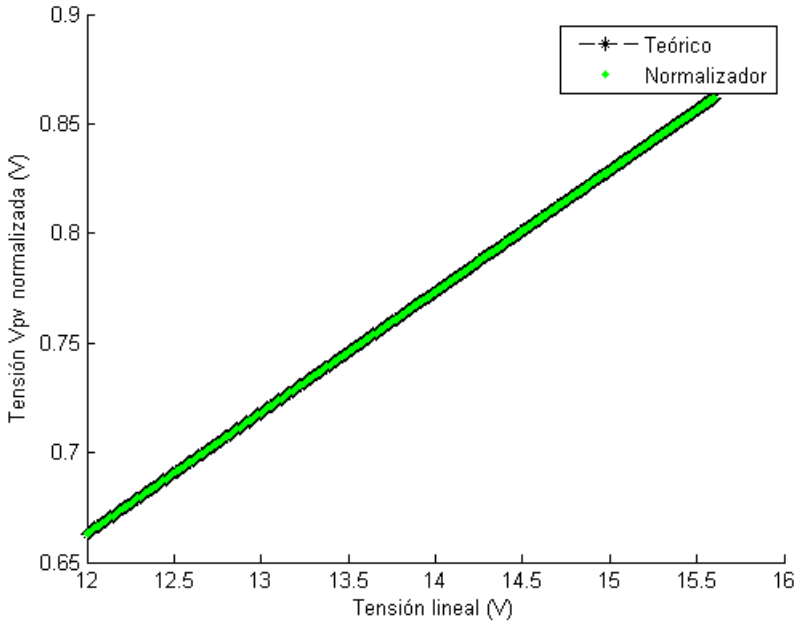

Figura 5.9: Comparación entre la conversión-normalización de tensión V_{pv} teórica y la simulación post-síntesis del circuito

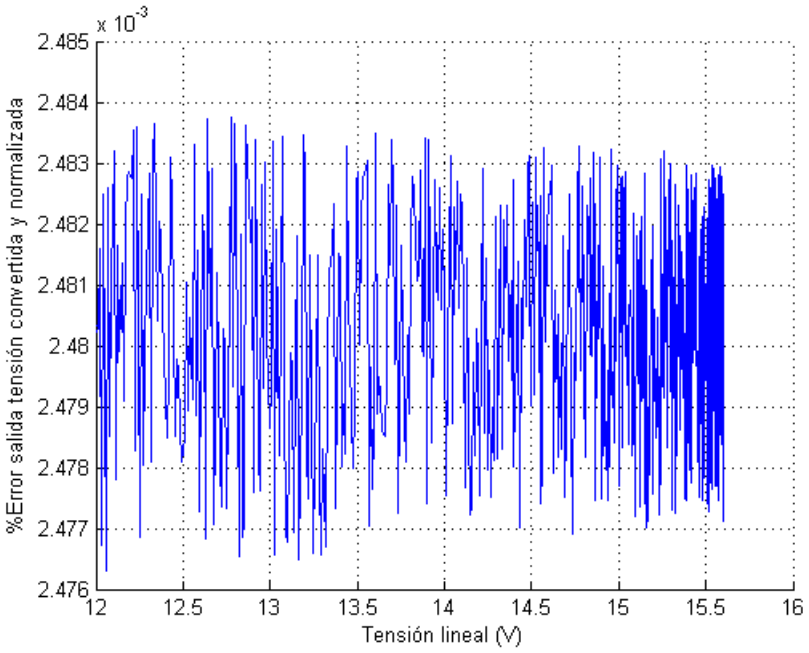

Figura 5.10: Porcentaje de error entre la conversión-normalización de tensión V_{pv} teórica y del circuito. Los valores teóricos se obtienen del modelo en Python de las operaciones deseadas.

Un análisis similar al que se realizó para la conversión-normalización de la corriente, se utiliza para la conversión de la tensión. En la figura 5.9 se muestran los resultados obtenidos con una tensión de entrada y su normalización, tanto teórica como experimental. En la figura 5.10 se puede observar el error con un máximo de 0,0024837%, un error promedio de 0,002478% y una desviación estándar de $1,94953 \cdot 10^{-6}$.

Esta unidad contiene muchos bloques combinacionales, por lo que se requieren pocos ciclos de reloj en la máquina de estados para realizar la conversión y la normalización.

Capítulo 6

Prueba del sistema completo sobre una placa desarrollo Nexys-4

Las unidades desarrolladas se incorporan finalmente en el sistema completo buscado:

- *Corriente* i_{pv}: el bloque para la corriente requiere las etapas de linealización, conversión de la salida del linealizador de coma flotante a coma fija, y un normalizador con una salida de corriente entre el rango [-1,1].
- *Tensión* V_{pv}: la tensión del panel se trabaja de manera lineal, solo se requiere de un normalizador que contenga la salida de tensión entre el intervalo [0,1].

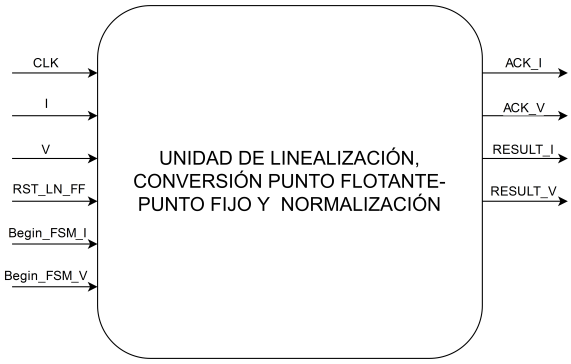

Figura 6.1: Diseño general de entradas y salidas para el sistema de linealización, conversión y normalización de corriente y tensión para un panel fotovoltaico.

El sistema de la figura 6.1 cuenta con 6 entradas y 4 salidas.

Entradas:

- *CLK*: reloj del sistema.
- *I*: dato de corriente no lineal, en formato IEEE 754
- *V*: dato de tensión en formato IEEE 754
- *RST_LN_FF*: reset para las unidades de corriente y tensión.

- *Begin_FSM_I*: inicia la máquina de estados del linealizador de corriente.
- *Begin_FSM_V*: inicia la máquina de estados de el convertidor coma flotante-coma fija para la tensión.

Salidas:

- *ACK_I*: indica que el resultado de las operaciones para la corriente se encuentra listo.
- *ACK_V*: indica que el resultado de las operaciones para la tensión se encuentra listo.
- *RESULT_I*: resultado de corriente lineal y normalizado en formato coma fija.
- *RESULT_V*: resultado de de tensión normalizado en formato coma fija.

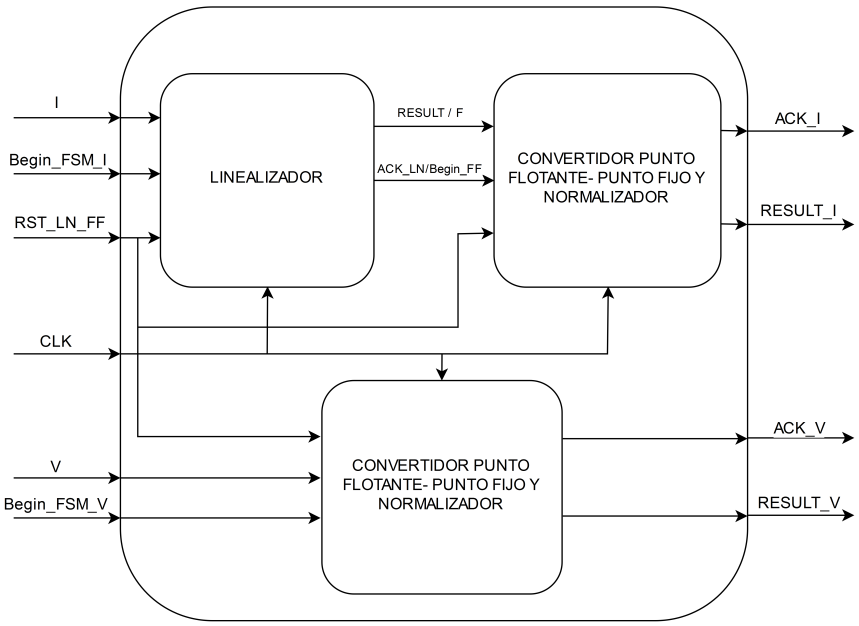

Figura 6.2: Diagrama de interconexión de los distintos bloques de procesamiento que componen unidad final de linealización

La figura 6.2 muestra la distribución de bloques funcionales sincronizados con un mismo reloj (CLK), para la corriente i_{pv} y tensión V_{pv} del panel fotovoltaico. Dentro de la sección de corriente se realiza primeramente la linealización, iniciando con las señales RST_LN_FF y Begin_FSM_I y el dato de entrada I (corriente i_{pv}), una vez ejecutada la operación se envía una señal indicando que el dato esta listo ACK_LN; esta se conecta a la señal de entrada e inicio del convertidor-normalizador de corriente Begin_FF. Un ciclo de reloj antes de que inicie la conversión-normalizacion, el resultado de la linealización ($RESULT$) esta listo. Este resultado linealizado se conecta a la entrada F del convertidor-normalizador. La ejecución del resultado final linealizado y normalizado, se comprueba con la señal ACK_I, esta indica que la conversión-normalización fue realizada y el resultado se encuentra en RESULT_I (corriente lineal y normalizada en coma fija y').

El bloque de tensión funciona de manera similar al de corriente, iniciando con las señales RST_LN_FF y Begin_FSM_V y el dato de entrada V (tensión V_{pv}). Una vez ejecutada

la operación se envía una señal indicando que el dato esta listo ACK_V y el resultado se despliega en RESULT_V (tensión normalizada en coma fija z').

6.1 Simulación del sistema completo

Para esta comprobación se utilizaron 1000 valores que describen el comportamiento real de un PV, utilizando el modelo del panel. Este toma un valor de tensión V_{pv} para cada valor de tiempo a partir de la ecuación 6.1, y se sustituye en la ecuación 6.2 obteniendo valores de corriente de entrada i_{pv} para el linealizador.

$$V_{pv} = V_{cte} + 0.3 * V_{cte} * sin(2 * \pi * 100 * t) \tag{6.1}$$

$$i_{pv} = Ig - \frac{V_{pv}}{R_p} - Is * \left(e^{\frac{V_{pv} * \alpha}{2}}\right) \tag{6.2}$$

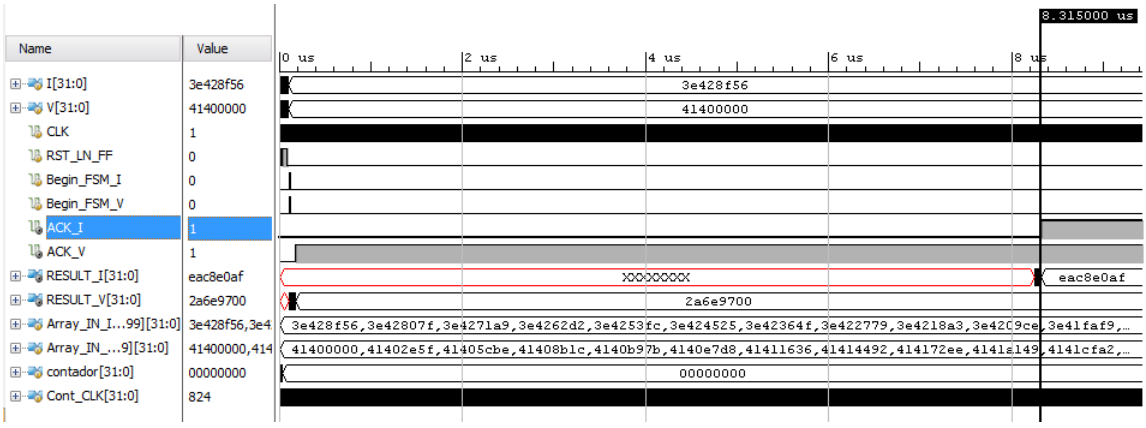

Figura 6.3: Simulación del sistema de linealización-conversión-normalización de corriente y sistema de conversión- normalización de tensión para un panel fotovoltaico

La simulación básica "behavioral" efectuada por Vivado, verifica el funcionamiento adecuado del circuito a manera de software y de lógica. Sin embargo, esta no es suficiente para conocer la funcionalidad correcta temporal, por lo que se realizan sobre el mismo banco de pruebas las simulaciones *post-syntesis y post-implementation*. La figura 6.3 muestra la simulación de tiempos posterior a la implementación, utilizado los valores de entrada de tensión y corriente a partir del modelo del panel.

6.2 Sistema para realizar las pruebas en una placa de desarrollo Nexys-4

Para desarrollar las pruebas en la FPGA, se requirió diseñar e implementar un circuito para insertar datos a la entrada del linealizador-normalizador, sean procesados y posteriormente enviados por medio de transmisión serial hacia un computador.

En la figura 6.4 a) pertenece al diagrama de flujo para el circuito que se utilizo para la verificación en la FPGA. En la figura 6.4 b) se muestra el diagrama de flujo utilizado para realizar las pruebas de alto nivel mediante python y una interfaz en octave para obtener resultados de la FPGA, por medio de transmisión RS232. Finalmente se realiza la comparación entre los valores teóricos y los resultados experimentales del circuito completo.

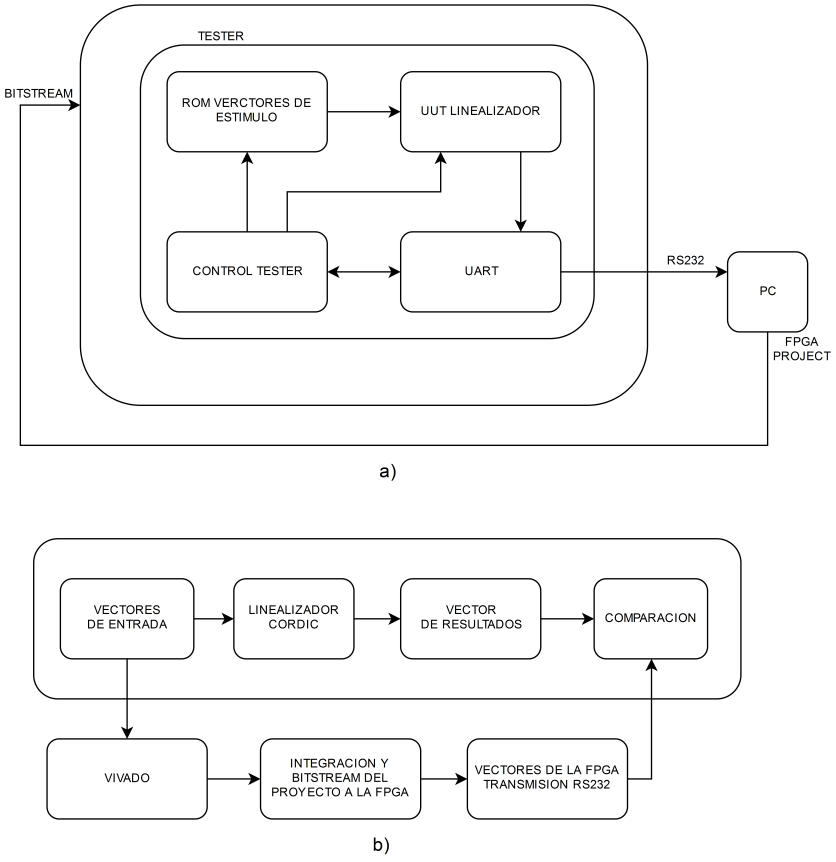

Figura 6.4: a) Diagrama de flujo general para el ambiente de verificación utilizado en la FPGA, utilizando un tester con una memoria ROM para los vectores de entrada y una UART para la transmisión de los vectores de resultados hacia el computador. b) Diagrama de flujo general para el ambiente de verificación utilizado en el computador, para realizar mediante una inteface de octave, la comparación entre los resultados teóricos y los resultados obtenidos de la FPGA.

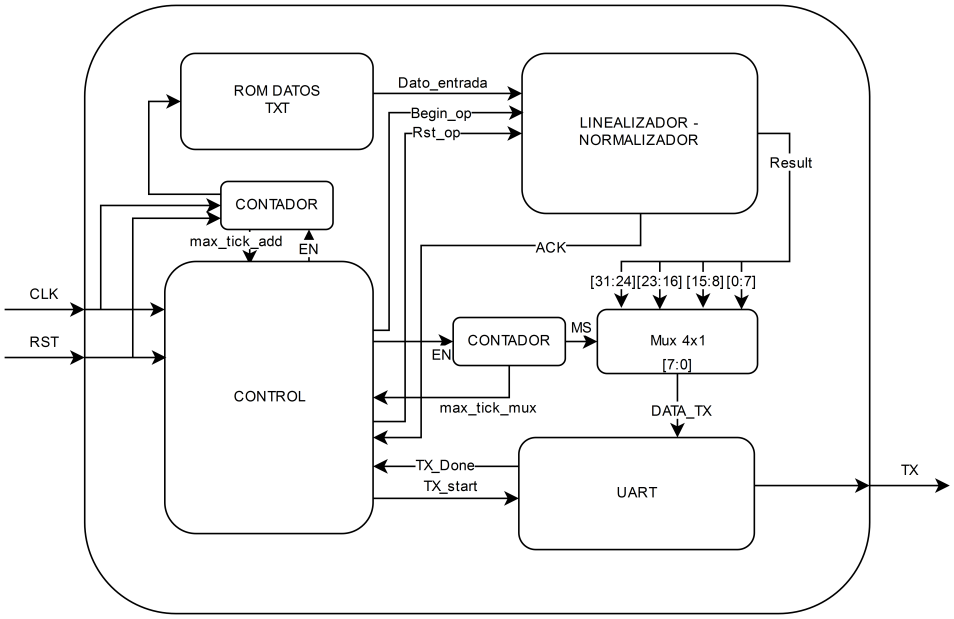

Figura 6.5: Detalle a nivel de bloques del sistema usado para verificar la unidad desarrollada.

El diagrama de la figura 6.5 muestra con detalle el sistema montado para la verificación final de todo el sistema desarrollado. Primeramente se ingresan 1024 datos almacenados en una memoria ROM. Un archivo de texto contiene los datos de entrada para el linealizador-normalizador y se carga desde la memoria ROM. Para direccionar esta memoria se usa un contador de 10 bits. La sección de transmisión posee una unidad UART (transmisión serial), que se utiliza para enviar los resultados de la unidad de linealización-normalización hacia el computador. Los resultados linealizados y noramlizados tienen un formato coma fija de 32 bits y la UART envía únicamente paquetes de 8 bits, por lo que se requiere enviar 4 paquetes por cada resultado. Esto se logra por medio de una configuración de un multiplexor 4x1 y un contador; las entradas del multiplexor contienen el resultado dividido en paquetes de 8 bits y por medio del contador se selecciona el multiplexor según sea el paquete que se desea enviar. Por último se cuenta con un control para el circuito que se encarga de llevar la sincronía de los datos. Se realiza una carga en el contador de la memoria ROM, se direcciona la primera posición de la ROM y se inicia el procesamiento por parte de la unidad de linealización-normalización, con la señal $BEGIN\_OP$; el control espera a que el resultado este listo por medio de la señal ACK. Cuando se da la condición ACK=High, el sistema se encuentra listo para transmitir. Entonces el contador del multiplexor selecciona el primer paquete y lo envía. El siguiente paquete se puede transmitir hasta que el modulo UART envié de vuelta al control, la señal TX_DONE y así sucesivamente hasta que se envíen los 4 paquetes. Cuando la selección del multiplexor esta en 2b'11 (contador del multiplexor), la señal MAX_TICK_MUX indica al control que debe realizar una cuenta para la dirección de la ROM y asi tomar el siguiente dato a procesar. Este proceso se repite hasta que la cuenta de las direcciones de la ROM es 1023 y la señal $MAX\_TICK\_ADD = 1$. La recepción de datos desde el computador se realizó por medio de un puerto USB y un "script" realizado en Matlab.

Este recibe los paquetes de un byte en hexadecimal y los concatena en paquetes de 4 bytes, para formar el dato en 32 bits, posteriormente se convierte a decimal.

6.3 Resultados de las pruebas sobre la placa de desarrollo Nexys-4

La verificación final del circuito completo brinda una mejor información acerca del porcentaje de error total obtenido dentro de la conexión de las tres etapas linealización, conversión y normalización.

Se realiza en la placa de desarrollo la verificación del sistema completo, con 8,12 y 15 iteraciones en el linealizador. Esta prueba contiene un barrido de 1000 datos de entrada obtenido a partir del modelo teórico del panel fotovoltaico, esto con el fin de observar un comportamiento similar al real. Los valores de corriente son pequeños debido a la diferencia de corrientes y perdidas en el panel fotovoltaico, por lo que los porcentajes de error serán menores debido a que el circuito posee una mejor aproximación; en el extremo inferior del intervalo de convergencia del algoritmo CORDIC. La visualización de los resultados obtenidos se realiza de una mejor manera mediante gráficos que indican como se comporta el circuito ante datos de entrada, salida, y porcentaje de error. A partir de los resultados de las simulaciones, se puede analizar la frecuencia de ejecución a la que se obtiene un resultado linealizado-normalizado, esta depende del número de iteraciones que se utilice. El tiempo de ejecución de la unidad completa, contempla los ciclos de cada uno de los bloques funcionales que componen la ruta de ejecución de la corriente este circuito. Si tomamos el bloque para la tensión, este solo cuenta con la conversión-normalización y no depende del número de iteraciones por lo que se requieren de 8 ciclos de reloj por cada dato.

Tabla 6.1: Comparación de resultados experimentales obtenidos por del sistema de verificación implementado en una placa de desarrollo Nexys-4, a partir de los valores de entrada del del modelo del panel y utilizando 8, 12 y 15 iteraciones en el algoritmo de CORDIC. CLK=100MHz.

	8 iteraciones	12 iteraciones	15 iteraciones
Error máximo (%)	0,922	0,259	0,129
Error promedio (%)	0,455	0,0351	0,0257
Desviación estándar	0,2695	0,0253	0,0082
Número de ciclos	468	668	826
Tiempo total de ejecución de la unidad (μs)	4,68	6,68	8,26

Utilizando 8 iteraciones en el algoritmo de CORDIC y a partir de las simulaciones post-síntesis efectuadas por medio de la herramienta Vivado, se requiere de 468 ciclos de reloj para completar la ejecución de un dato, este se compone de 460 ciclos de reloj para el linealizador y 8 ciclos de reloj para el convertidor-normalizador, donde la velocidad de ejecución es de 4,68μs (217kHz) con un reloj de sistema de 100MHz.

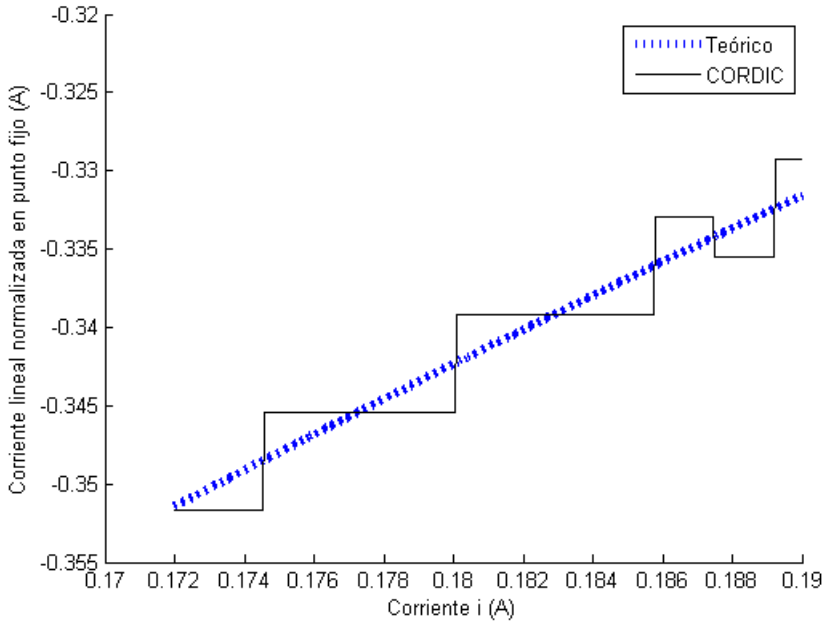

Figura 6.6: Comparación entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 8 iteraciones en el algoritmo de CORDIC del linealizador

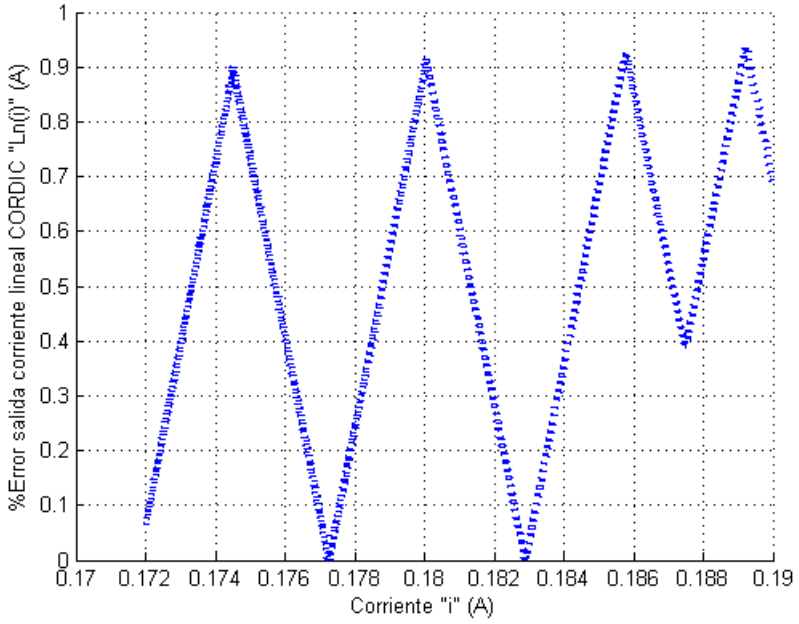

Figura 6.7: Porcentaje de error entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 8 iteraciones en el algoritmo de CORDIC del linealizador

En la figura 6.6 se puede observar la comparación entre los datos obtenidos del circuito implementado en una placa Nexys-4, contra los datos teóricos al realizar la misma función del circuito. Debido a que son solo 8 iteraciones el circuito posee un resultados aceptables pero poco exactos. El gráfico muestra que el algoritmo trata de mantenerse cerca de los valores reales, sin embargo posee un comportamiento escalonado, en donde para cierta cantidad de valores de entrada posee un mismo valor de salida constante cercano al valor real.

La figura 6.7 muestra el porcentaje de error asociado a cada valor de entrada linealizado y normalizado en formato coma fija, donde el porcentaje de error máximo es de 0,922% y un porcentaje de error promedio de 0,455% y una desviación estándar de 0,2695, esto para valores de corriente derivados a partir del modelo teórico del panel fotovoltaico.

Utilizando 12 iteraciones en el algoritmo de CORDIC, se requiere de 668 ciclos de reloj para realizar el procesamiento completo de un dato de entrada, 660 ciclos de reloj son utilizados por el linealizador y 8 ciclos de reloj para el convertidor-normalizador. Este se ejecuta con una velocidad de 6.68μs (151kHz), utilizando un reloj de sistema de 100MHz.

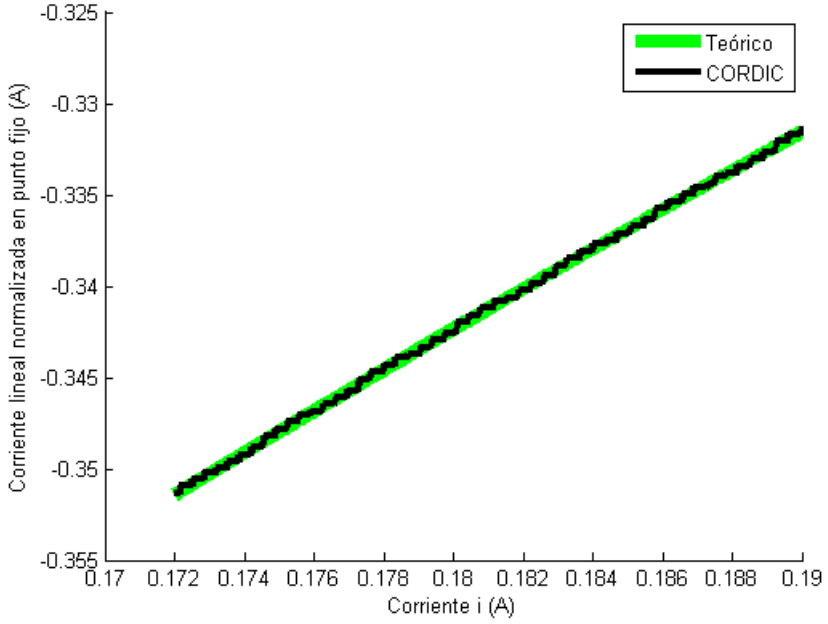

Figura 6.8: Comparación entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 12 iteraciones en el algoritmo de CORDIC del linealizador

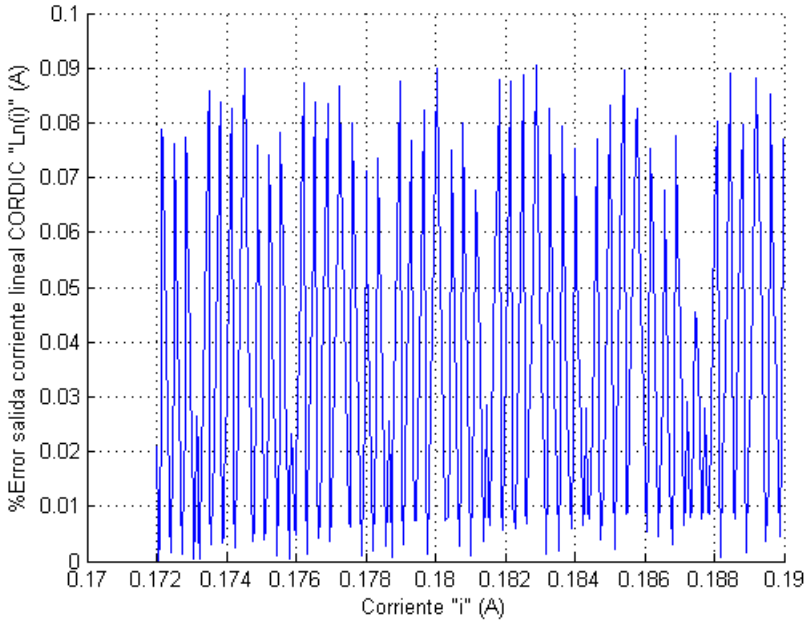

Figura 6.9: Porcentaje de error entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 12 iteraciones en el algoritmo de CORDIC del linealizador

En la figura 6.8 se puede observar la comparación entre los datos obtenidos experimentalmente contra los datos teóricos al realizar la misma función del circuito utilizando 12 iteraciones. Brinda una mejor exactitud contra 8 iteraciones y se puede observar que presenta una mayor suavidad en la curva, sin embargo presenta un comportamiento escalonado pero con un valores más cercanos a la curva real. La figura 6.9 muestra el porcentaje de error asociado a cada valor de entrada linealizado con 12 iteraciones y normalizado en formato coma fija. Donde el porcentaje de error máximo es de 0,0897% y un porcentaje de error promedio de 0,0345% y una desviación estándar de 0,0253, esto para valores de corriente derivados a partir del modelo teórico del panel fotovoltaico.

En el sistema del panel es de suma importancia el tiempo de muestreo según sea la frecuencia del panel, esto implica velocidad de ejecución dentro del cálculo. Se logró determinar que con 15 iteraciones se obtienen resultados con muy buena exactitud,sin embargo el tiempo de ejecución aumenta a 826 ciclos de reloj, tomando en cuenta que 818 ciclos son requeridos por el linealizador y 8 ciclos de reloj para la conversión de formato IEEE 754 a coma fija y normalización. Donde la velocidad de ejecución es de 8.26μs (121kHz) con un reloj de sistema de 100MHz.

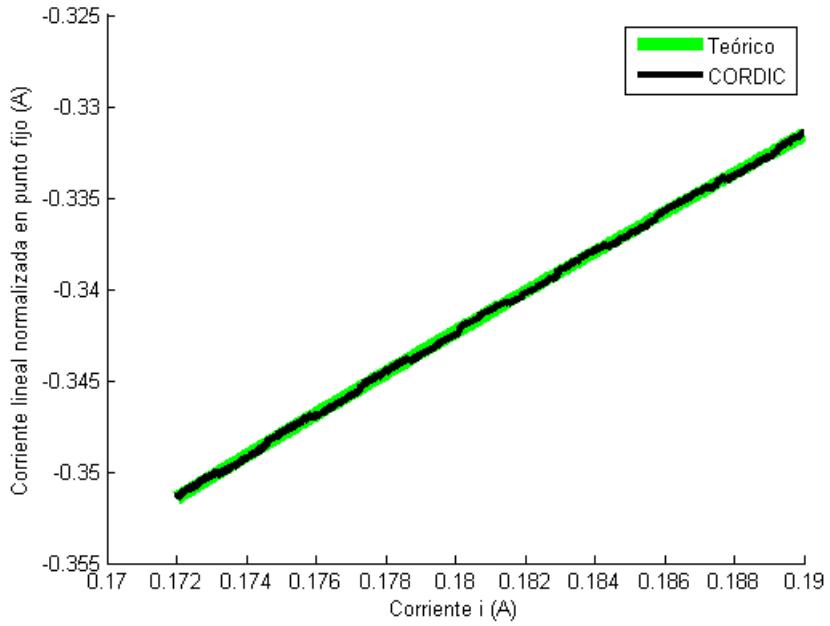

Figura 6.10: Comparación entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 15 iteraciones en el algoritmo de CORDIC del linealizador

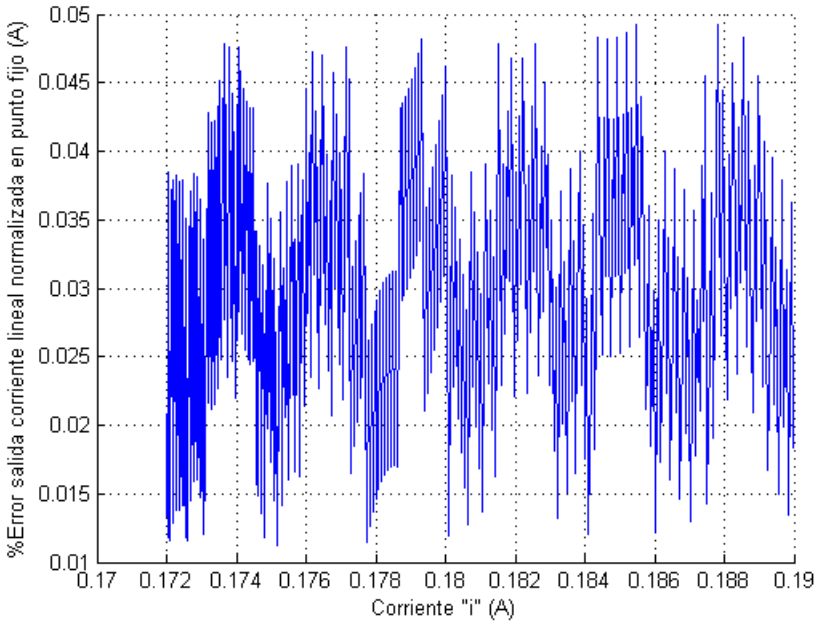

Figura 6.11: Porcentaje de error entre el valor teórico y el valor obtenido del circuito de linealización-conversión-normalización de corriente con 15 iteraciones en el algoritmo de CORDIC del linealizador

Con 15 iteraciones el valor experimental es muy similar al valor teórico, como se observa en la figura 6.10, y se puede comprobar en la figura 6.11 donde se muestra que el porcentaje de error es muy cercano a cero, con un valor máximo de 0,0483% y un error promedio de 0,0292% y una desviación estándar de 0,0082, esto para valores de corriente derivados a partir del modelo teórico del panel fotovoltaico.

El sistema de optimización se utilizará en un panel fotovoltaico que posee una frecuencia de 100kHz, es decir que el sistema de optimización debe realizar un cálculo completo a una velocidad mayor que el panel. Aproximadamente el sistema de linealización y normalización toma un 40% del total del tiempo de ejecución del sistema de optimización, por lo que se requiere realizar el cálculo en el tiempo indicado, para esto se deberá utilizar 8 iteraciones.

6.4 Recursos utilizados

Tabla 6.2: Resumen del reporte post implementación del uso de dispositivos generado por la herramienta Vivado, para la FPGA Artix-7 dentro de la placa Nexys4

Recurso	Utilizados	Disponibles	Utilizados%
LUT	1017	63400	1.60
FF	912	126800	0.72
DSP	4	240	1.67
IO	134	210	63.81
BUFG	1	32	3.12

En la tabla 6.2 se muestra el uso de recursos para la implementación el circuito completo en la la FPGA Artix 7. Debe considerarse que el uso de recursos IO es engañoso, pues esta unidad irá incrustada dentro de un sistema más grande (ver figura 6.1).

6.5 Reporte de tiempos

Dentro del reporte de tiempos se analizan las peores rutas "ruta critica". Para un buen diseño se requiere de slacks positivos, en el caso de "setup" un slack positivo indica que el dato llega a tiempo antes de ser utilizado por el siguiente latch. En el caso del "hold" indica que se mantiene por el tiempo adecuado antes de ser procesado por otro bloque. Si ambos slacks son cero, el diseño esta en el limite y puede ser un poco arriesgado en sincronización. El setup slack se puede calcular como:

$$setup\_slack = tiempo\_de\_setup\_requerido\_por\_el\_dato - tiempo\_del\_dato\_estable \quad (6.3)$$

El hold slack se puede calcular como:

$$hold\_slack = tiempo\_de\_cambio\_de\_dato - tiempo\_de\_hold\_requerido\_por\_el\_dato \quad (6.4)$$

Tabla 6.3: Resumen del reporte post implementación de tiempos del circuito completo, a partir de la herramienta Vivado.

Peor slack de todas las rutas	tiempo (ns)
Setup	0,459
Hold	0,109
Pulse width	4,5

En la tabla 6.3 se muestran los valores obtenidos para los slacks del circuito del linealizador-normalizador, como se puede observar son positivos y cumplen con los requerimientos del diseño del circuito.

6.6 Consumo de potencia

Con la ayuda de las herramientas de análisis de potencia de Vivado, se pudo obtener los resultados que pertenecen al consumo de potencia, tanto estática como dinámica. En la tabla 6.4 se muestra el valor para cada consumo, y el total.

Tabla 6.4: Resumen del reporte post implementación de la potencia estática, dinámica y total, de la herramienta Vivado, para el sistema de linealización-normalización.

Potencia	Consumo de potencia (mW)
Dinámica	11
Estática	91
Total	102

Para los recursos utilizados se puede realizar un estudio de consumo de potencia dinámica, cuyos resultados se muestran en la tabla 6.5. El mayor consumo de potencia se presenta por parte del reloj del sistema y señales.

Tabla 6.5: Resumen del reporte generado por la herramienta Vivado que indica el consumo de potencia de diversos elementos.

On-chip (mW)	Consumo de potencia (mW)
Clocks	4
Signals	4
Logic	3
DSP	0
I/O	0

Capítulo 7

Conclusiones y recomendaciones

7.1 Conclusiones

- Se comprobó que es posible sintetizar la unidad de linealización-normalización con precisión aceptable utilizando versiones reducidas del estándar IEEE 754 (32bits).
- Se comprobó en el circuito linealizador que a mayor número de iteraciones mayor exactitud presenta el resultado, a costa de un mayor tiempo de ejecución.
- Se verificó que para el circuito linealizador, el porcentaje de error máximo fue de 2,74% con 8 iteraciones en el rango de convergencia del algoritmo de CORDIC.
- Se comprobó que para el circuito convertidor-normalizador tanto de corriente como de tensión, el porcentaje de error máximo fue de 0,0024%.
- Se verificó que para el circuito completo, el porcentaje de error máximo fue de 0,922% con 8 iteraciones en el algoritmo de CORDIC.
- Se determinó que el módulo de la corriente consume más recursos que el módulo de tensión, dado que la corriente requiere de un modulo de linealización y a su vez mayor cantidad de hardware.

7.2 Recomendaciones

- Se utilizó un único sumador coma flotante para reducir el área del circuito. Sin embargo, el espacio usado por este en la FPGA es muy poco, por lo tanto se podría realizar modificaciones para reducir el número de ciclos de cada iteración de la máquina de estados, implementando una arquitectura con cálculo de X_i , Y_i y Z_i de manera paralela. Es decir utilizar un sumador de coma flotante para cada variable de manera que se realicen los cálculos simultáneos. Esta variación implica un aumento en el área que no obstante optimizaría en al menos tres veces más la velocidad de ejecución de todo el sistema.
- Si se utiliza algún otro tipo de panel con el sistema y se requiere un rango de linealización más amplio se puede utilizar un algoritmo de CORDIC hiperbólico con una extensión de las iteraciones de al menos dos. Esto no obstante implica un

crecimiento en el uso de recursos y un tiempo mayor de ejecución.

Capítulo 8

Referencias bibliográficas

[1] Suskis, Pavels, and Ilya Galkin. "Enhanced photovoltaic panel model for MATLAB-simulink environment considering solar cell junction capacitance." Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE. IEEE, 2013.

[2] González-Longatt, Francisco M. "Model of photovoltaic module in Matlab." II CIBELEC 2005 (2005): 1-5.

[3] C. Meza, R. Ortega. "Control and estimation scheme for PV central inventers", in 24th International Conference on information, Comunication and Automation Technologies, Nov, 2013

[4] Chiang, Ching-Tsan, Tung-Sheng Chiang, and Hou-Sheng Huang. "Modeling a photovoltaic power system by CMAC-GBF." Photovoltaic Energy Conversion, 2003. Proceedings of 3rd World Conference on. Vol. 3. IEEE, 2003.

[5] Ibrahim, Muhammad Nasir, et al. "Hardware Implementation of Math Module Based on CORDIC Algorithm Using FPGA." Parallel and Distributed Systems (ICPADS), 2013 International Conference on. IEEE, 2013.

[6] Walther, John S. "A unified algorithm for elementary functions." Proceedings of the May 18-20, 1971, spring joint computer conference. ACM, 1971.

[7] Llamocca-Obregón, Daniel R., and Carla P. Agurto-Ríos. "A fixed-point implementation of the expanded hyperbolic CORDIC algorithm." Latin American applied research 37.1 (2007): 83-91.

[8] Whitehead, Nathan, and Alex Fit-Florea. "Precision and performance: Floating point and IEEE 754 compliance for NVIDIA GPUs." rn (A+ B) 21 (2011): 1-1874919424.

[9] Bello, C., et al. "Relevador portátil de curvas IV de paneles fotovoltaicos como herramienta de diagnostico in situ de sistemas de generación fotovoltaica." Avances en Energías Renovables y Medio Ambiente 13 (2009): 77-83.

[10] Raygoza, Juan José, et al. "Implementación en hardware de un sumador de punto flotante basado en el estándar IEEE 754-2008." e-Gnosis 7 (2009).

[11] Bube, Richard. Fundamentals of solar cells: photovoltaic solar energy conversion. Elsevier, 2012.

[12] Boudabous, Anis, et al. "Implementation of hyperbolic functions using CORDIC algorithm." Microelectronics, 2004. ICM 2004 Proceedings. The 16th International Conference on. IEEE, 2004.

[13] Mano, M. Morris. Arquitectura de computadoras. Pearson Educación, 1994.

[14] Floyd, Thomas L. Fundamentos de sistemas digitales. Vol. 7. Prentice Hall, 2006.

[15] Rodríguez, D. "Diseño e implementación de una unidad aritmético-lógica de coma flotante para un procesador de aplicación específica", Escuela de Ingeniería en Electrónica, Instituto Tecnológico de Costa Rica, Enero, 2016.

[16] Cerdas-Robles, R., Rodriguez, A., Chacon-Rodriguez, A., Julian, P. "Design of an IDM-based determinant computing unit for a 130nm low power CMOS ASIC acoustic localization processor," in Circuits & Systems (LASCAS), 2015 IEEE 6th Latin American Symposium on , vol., no., pp.1-4, 24-27 Feb. 2015.